Peter Hellwig: Kurs Maschinelle Syntaxanalyse (Parsing)

Diese Unterrichtsmaterialien sind frei verwendbar.

Feedback willkommen an mailto:hellwig@cl.uni-heidelberg.de

Der Kurs hält sich im Wesentlichen an das Skript Peter Hellwig "Natural Language Parsers - A course in Cooking". Es wird vorausgesetzt, dass man das Skript parallel zum Kurs durcharbeiten. Die Folien haben komplementären Charakter.

In jeder Stunde wird in ein neuer Prototyp von Parser eingeführt, anhand dessen jeweils bestimmte Eigenschaften besprochen werden. Dazu gibt es jeweils eine Hausaufgabe. Für den Lernerfolg ist es sehr wichtig, die Hausaufgabe akribisch auszuführen und anschließend mit der Musterlösung zu vergleichen. Nur so bekommt man einen wirklichen Eindruck von dem betreffenden Algorithmus und kann die im Kurs entwickelten Kriterien zur Beurteilung oder zur Entwicklung eines eigenen Parsers anwenden.

INHALT

1. Stunde: Parser und Grammatik

What is Parsing?

(i) In computer science:

Parsing is the assignment of a structural description to a character string.

(ii) In linguistics:

Parsing is the assignment of a syntactic description to a sentence.

(iii) In knowledge-based systems:

Parsing is the assignment of a knowledge representation to an utterance.

Parser and Grammar

A language is a set of character strings. The well-formed character strings of a language are defined by a grammar.

A parser can therefore be defined more specifically as a computer program that assigns a structural description to a given string relative to a given grammar.

Prerequisites of a parser:

(i) A grammar formalism (*knowledge representation)*

(ii) A grammar (*knowledge)*

(iii) An algorithm (*knowledge processing)*

Die Idee der generativen Grammatik

Struktur

Welche der folgenden Sätze sind strukturell nahe verwandt?

(1) Er vertraut jedem

(2) Wer vertraut diesem Gauner

(3) Er bringt mir die Milch

(4) Kein Mensch glaubt ihm

(5) Jeder Lehrer hilft seinem Schüler

(6) Wer versteckte die Eier

(7) Dem hilft kein Mensch

(8) Die Schildkröte bewegt den Kopf

(1) Er vertraut jedem

(2) Wer vertraut diesem Gauner

(3) Er bringt mir die Milch

(4) Kein Mensch glaubt ihm

(5) Jeder Lehrer hilft seinem Schüler

(6) Wer versteckte die Eier

(7) Dem hilft kein Mensch

(8) Die Schildkröte bewegt den Kopf

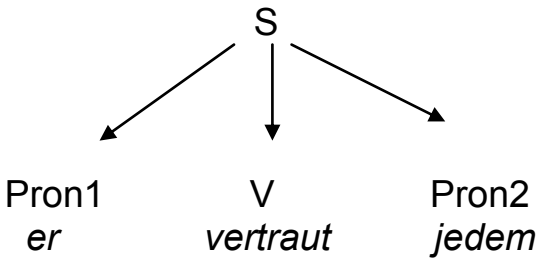
Lösung: (1), (2), (4), (5), (7)

Wir schreiben eine Phrasenstrukturgrammatik für diese Sätze.

Typische Aufgabe dabei: Zerlegen, Klassifikation, Kombination

(1) *er vertraut jedem*

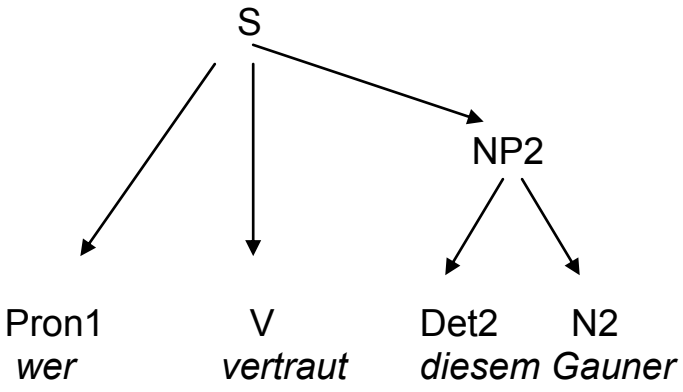
S	->	Pron1 + V + Pron2
Pron1	->	er
V	->	vertraut
Pron2	->	jedem



(2) *wer vertraut diesem Gauner*

S	->	Pron1 + V + Pron2
Pron1	->	er
V	->	vertraut
Pron2	->	jedem

Pron1	->	wer
S	->	Pron1 + V + NP2
NP2	->	Det2 + N2
Det2	->	diesem
N2	->	Gauner



(4) *kein Mensch glaubt ihm*

S	->	Pron1 + V + Pron2
Pron1	->	er
V	->	vertraut
Pron2	->	jedem
Pron1	->	wer
S	->	Pron1 + V + NP2
NP2	->	Det2 + N2
Det2	->	diesem
N2	->	Gauner
V	->	glaubt
Pron2	->	ihm
S	->	NP1 + V + Pron2
NP1	->	Det1 + N1
Det1	->	kein
N1	->	Mensch

Verallgemeinerung der bisherigen Analyse, neue Grammatik:

Regeln:

S	->	NP1 + V + NP2
NP1	->	Pron1
NP1	->	Det1 + N1
NP2	->	Pron2
NP2	->	Det2 + N2

Lexikon:

V	=	*vertraut, glaubt*
Pron1	=	*er, wer*
Pron2	=	*jedem, ihm*
Det1	=	*kein*
Det2	=	*diesem*
N1	=	*Mensch*
N2	=	*Gauner*

(1) Er vertraut jedem

(2) Wer vertraut diesem Gauner

(3) Er bringt mir die Milch

(4) Kein Mensch glaubt ihm

(5) Jeder Lehrer hilft seinem Schüler

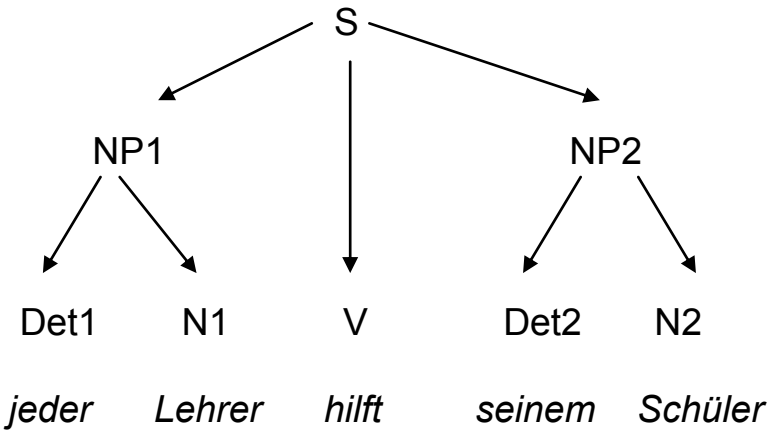
(6) Wer versteckte die Eier

(7) Dem hilft kein Mensch

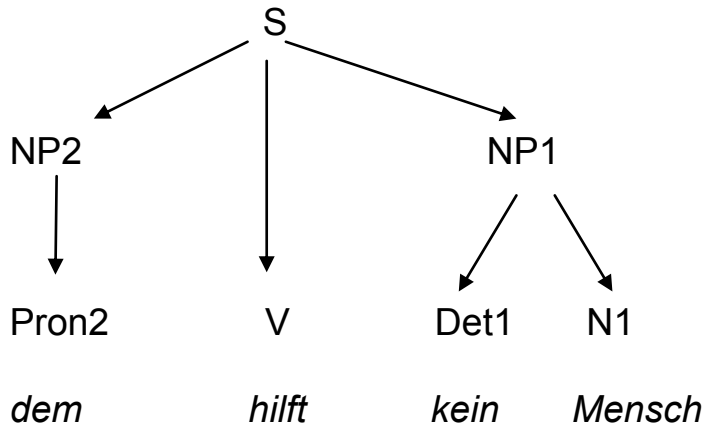
S -> NP1 + V + NP2
NP1 -> Pron1
NP1 -> Det1 + N1
NP2 -> Pron2
NP2 -> Det2 + N2

S -> NP2 + V + NP1
Pron2 -> *dem*

(5) *jeder Lehrer hilft seinem Schüler*



(7) *Dem hilft kein Mensch*



Formale Definition einer sog. kontextfreien Phrasenstrukturgrammatik G

G = (T, NT, R, S)

T	-	Menge von Terminalsymbolen (=die Elemente der Sprache)
NT	-	Menge von nicht-terminalen Symbolen (= Kategorien)
R	-	Menge von Regeln der Form α -> β (= Zerlegungen)
		$\alpha \in NT, \quad \beta \subseteq (NT \cup T)$
S	-	Startsymbol (= beschriebene Einheit, Satz)
		$S \in NT$

Formale Definition einer Sprache L

L = G(L) = { S | S wird durch G generiert }

Komplexe Kategorien

Bisherige Grammatik:

```
S     ->   NP1  +  V  +  NP2
S     ->   NP2  +  V  +  NP1
NP1   ->   Pron1
NP1   ->   Det1  +  N1
NP2   ->   Pron2
NP2   ->   Det2  +  N2
```

Äquivalente Grammatik mit komplexen Kategorien:

```
kategorie[satz]    ->    kategorie[np]  kasus[nominativ]  person[C]  numerus[C] +
                         kategorie[verb]  person[C]   numerus[C]+
                         kategorie[np]  kasus[dativ]

kategorie[satz]    ->    kategorie[np]  kasus[dativ]  +
                         kategorie[verb]  person[C]   numerus[C] +
                         kategorie[np]  kasus[nominativ]  person[C]  numerus[C]

kategorie[np] kausus[C]  person[C] numerus[C]    ->
                         kategorie[pronomen] kasus[C] person[C]  numerus[C]

kategorie[np] kausus[C]  person[3] numerus[C]    ->
                         kategorie[artikelwort] kasus[C]  numerus[C]  +
                         kategorie[nomen] kasus[C] person[C] numerus[C]
```

Nur noch 4 Regeln, die obendrein weiterreichend und genauer sind!

Analysetechniken

Verschiebeprobe

Könnte ich das geblümte Kleid im Schaufenster anprobieren?

Das geblümte Kleid im Schaufenster könnte ich anprobieren.

Das geblümte Kleid könnte ich im Schaufenster anprobieren.

Ich könnte das geblümte Kleid im Schaufenster anprobieren.

Ich könnte im Schaufenster das geblümte Kleid anprobieren.

---> zum Abgrenzen von Satzgliedern

[Könnte] [ich} [das geblümte Kleid im Schaufenster] {anprobieren}?

[Könnte] [ich} [das geblümte Kleid] [im Schaufenster] {anprobieren}?

Austauschprobe

*Das Gesetz tritt in Kraft / *in Macht*

Er gab seine Bemühungen auf

***Er verstärkte seine Bemühungen auf**

---> zum Erkennen fester Syntagmen

Ich meine, daß er versprochen hat, das Handout zu kopieren

Ich meine das.

Könnte ich das geblümte Kleid im Schaufenster anprobieren?

Könnte ich das anprobieren?

Könnte ich das dort anprobieren?

Was könnte ich anprobieren?

---> zum Abgrenzen von Satzgliedern und anderen Syntagmen

Er/sie/der Freund/wer müde ist schläft

Er liebt seine Freundin/Geld/was aufregend ist

Er gibt ihr eine Mark

***Er schläft seine Freundin**

***Er liebt seine Freundin eine Mark**

---> zum Feststellen von Valenz, Rektion, Hierarchie

Ich wasche mich /dich / mein Auto

*Ich freue mich / *dich/ *mein Auto*

Wir waschen uns / einander

***Wir freuen einander**

---> zur Abgrenzung von reflexiven Verben und nur reflexiv gebrauchten Verben

Weglaßprobe

Wir haben etwas gegen Frost

Wir haben etwas

**Wir haben gegen Frost*

---> zur Festellung von Valenz, Abhängigkeiten, Hierarchie

Er kommt schnell -> er kommt

*Er kommt vielleicht -> *er kommt*

---> zur Beobachtung semantischer Unterschiede

Exklusions- und Koordinationsprobe

**Er schlug die Schaufensterscheibe und den Weg zum Bahnhof ein*

Er spielte die Flöte und die Guitarre

**Er spielte die Flöte und den ganzen Abend*

Er spielte die Flöte den ganzen Abend

> zur Identifizierung syntaktischer Rollen (gleiche koordinierbar, ungleiche nicht)

Paraphraseprobe

Sie ist leicht zu lieben

Es ist leicht, sie zu lieben

Sie ist bereit zu lieben

**Es ist bereit, sie zu lieben*

schwer, schwerlich, unmöglich, innig?

---> zum Erkennen von Rollen, (Tiefen)struktur

Fragetest, Negationstest, Imperativtest u.a.

*Das Gesetz tritt in Kraft - *in was tritt das Gesetz*

Was tritt in Kraft?

Sie fahren mit Abstand am besten.

Wie fahren Sie am besten - mit Abstand

Wie fahren Sie? - mit Abstand am besten

Er schläft tief - er schläft nicht tief

*Er schläft hoffentlich - *er schläft nicht hoffentlich*

*Schlafe tief! *Schlafe hoffentlich!*

---> zur Analyse satz-semantischer Unterschiede

Valenzanalyse: Prädikate (P), Ergänzungen (E1, E2...), Adverbialangaben (A)

(1) Die Ärzte haben keine Beweise für den Ebola-Virus gefunden

P: haben - gefunden
E1: die Ärzte / sie
E2: keine Beweise für den Ebola-Virus / nichts / ein Mittel gegen AIDS

jemand findet etwas

Analysiere ebenso:

(2) Die Weltgesundheitsorganisation will den Gouverneur der Hauptstadt dazu bewegen, denen, die nicht mit dem Ebola-Virus infiziert sind, die Weiterreise zu erlauben.

(3) Ein zerzauster Bart bedeutete bei den alten Griechen Trauer und Verzweifelung

(4) Jeder Mistkäfer ist in den Augen seiner Mutter eine Gazelle

(5) Vor allem das Geschäft mit Firmenkunden läuft extrem schlecht.

(6) Der Anpassungsbedarf gerade des größten deutschen Geldhauses ist gewaltig

(7) Vermutlich vier bewaffnete Männer haben am Dienstag eine Bank in Berlin überfallen

(8) Darin heißt es, das Kraftwerk werde ohne Genehmigung betrieben

(9) Die Flüchtlinge sitzen jetzt seit mehreren Tagen 150 Kilometer vor der Hauptstadt an einer Straßensperre fest, mit der das zairische Militär das Einschleppen des Virus nach Kinshasa verhindern will.

(10) Wer sich heute ans Internet anschließt, bekommt von all dem wenig mit.

(11) Die Antwort interessiert weniger als die englische Übersetzung

(12) Als Vorsitzender der Konferenz mahnte er vergangene Woche im amerikanischen Seattle seine Kollegen, daß "wir uns vom großen Dampfer in ein Schnellboot verwandeln müssen."

(13) Die Anlage, so die Atomaufseher, weiche in sicherheitstechnisch erheblicher Weise von der Genehmigung ab

(14) Die RWE hat zwei Wochen Zeit, Stellung zu nehmen

Lösungen:

2) Die Weltgesundheitsorganisation will den Gouverneur der Hauptstadt dazu bewegen, denen, die nicht mit dem Ebola-Virus infiziert sind, die Weiterreise zu erlauben.

P: will bewegen
E1: Die Weltgesundheitsorganisation
E2: den Gouverneur der Hauptstadt
E3: dazu, denen, die nicht mit dem Ebola-Virus infiziert sind, die Weiterreise zu erlauben.

jemand bewegt jemanden zu etwas

(3) Ein zerzauster Bart bedeutete bei den alten Griechen Trauer und Verzweifelung

P: bedeutete
E1: Ein zerzauster Bart
E2: Trauer und Verzweifelung
A: bei den alten Griechen

etwas bedeutet etwas

(4) Jeder Mistkäfer ist in den Augen seiner Mutter eine Gazelle

P: ist - eine Gazelle
E1: Jeder Mistkäfer
A: in den Augen seiner Mutter
jemand ist etwas

(5) Vor allem das Geschäft mit Firmenkunden läuft extrem schlecht.

P: läuft - extrem schlecht
E1: Vor allem das Geschäft mit Firmenkunden

etwas läuft gut/schlecht

(6) Der Anpassungsbedarf gerade des größten deutschen Geldhauses ist gewaltig

P: ist - gewaltig
E1: Der Anpassungsbedarf gerade des größten deutschen Geldhauses

etwas ist gewaltig

(7) Vermutlich vier bewaffnete Männer haben am Dienstag eine Bank in Berlin überfallen

P: haben überfallen
E1: Vermutlich vier bewaffnete Männer
E2: eine Bank in Berlin
A: am Dienstag

jemand überfällt jemanden/eine Institution

(8) Darin heißt es, das Kraftwerk werde ohne Genehmigung betrieben

P: heißt es
E1: das Kraftwerk werde ohne Genehmigung betrieben
A: Darin

es heißt so-und-so

(9) Die Flüchtlinge sitzen jetzt seit mehreren Tagen 150 Kilometer vor der Hauptstadt an einer Straßensperre fest, mit der das zairische Militär das Einschleppen des Virus nach Kinshasa verhindern will.

P: sitzen fest
E1: Die Flüchtlinge
A1: jetzt seit mehreren Tagen
A2: 150 Kilometer vor der Hauptstadt
A3: an einer Straßensperre fest, mit der das zairische Militär das Einschleppen des Virus nach Kinshasa verhindern will.

jemand sitzt fest

(10) Wer sich heute ans Internet anschließt, bekommt von all dem wenig mit.

P: bekommt mit - wenig
E1: Wer sich heute ans Internet anschließt
E2: von all dem

jemand bekommt von etwas wenig/viel mit

(11) Die Antwort interessiert weniger als die englische Übersetzung

P: interessiert - weniger als
E1: Die Antwort
E1: die englische Übersetzung

etwas interessiert

(12) Als Vorsitzender der Konferenz mahnte er vergangene Woche im amerikanischen Seattle seine Kollegen, daß "wir uns vom großen Dampfer in ein Schnellboot verwandeln müssen."

P: mahnte
E1: er
E2: seine Kollegen
E3: daß "wir uns vom großen Dampfer in ein Schnellboot verwandeln müssen."
A1: Als Vorsitzender der Konferenz
A2: vergangene Woche
A3: im amerikanischen Seattle

jemand mahnt jemanden, daß etwas der Fall ist

(13) Die Anlage, so die Atomaufseher, weiche in sicherheitstechnisch erheblicher Weise von der Genehmigung ab
P: weiche ab
E1: Die Anlage
E2: von der Genehmigung
A1: in sicherheitstechnisch erheblicher Weise
X: so die Atomaufseher

etwas weicht von etwas ab

(14) Die RWE hat zwei Wochen Zeit, Stellung zu nehmen

P: hat - Zeit
E1: Die RWE
E2: Stellung zu nehmen
A: zwei Wochen

jemand hat Zeit, um etwas zu tun

2. Stunde: Prototypischer Top-down Parser mit Rücksetzen (PT-1)

Beispielgrammatik G1

Regeln
(R-1) **S -> NP VP**
(R-2) **VP -> vi**
(R-3) **VP -> vt NP**
(R-4) **VP -> vt NP PP**
(R-5) **NP -> n**
(R-6) **NP -> det n**
(R-7) **NP -> det adj n**
(R-8) **PP -> prep NP**

Lexikon
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}*

G1 generiert u.a.: *tourists sleep* *the tourists sleep*
many foreign tourists see the pyramids
many foreign tourists visit the pyramids in Egypt...

PT-1. Top-down parser with backtracking (Struktogramm -> 3.Stunde)

"Many foreign tourists see the pyramids"

INPUT TABLE

Input:	*many*	*foreign*	*tourists*	*see*	*the*	*pyramids*
Lexicon:	**det**	**adj**	**n**	**vt**	**det**	**n**
Position:	1	2	3	4	5	6

WORKING TABLE

A	Derivations	Explanation	P
1	**S**	state at start	1
2	**NP VP**	expansion R-1	
3	**n VP**	expansion R-5	
2	**NP VP**	B=1 back to A=2	1
3	**det n VP**	expansion R-6	
4	**n VP**	recognized det	2

BACKTRACKING STORE

B	Back to
0	
1	P=1 **A=2** R-6
0	
1	P=1 **A=2** R-7

Fortsetzung ...

Input:	*many*	*foreign*	*tourists*	*see*	*the*	*pyramids*
Lexicon:	**det**	**adj**	**n**	**vt**	**det**	**n**
Position:	1	2	3	4	5	6

2	**NP VP**	B=1 back to A=2	1
3	**det adj n VP**	expansion R-7	
4	**adj n VP**	recognized det	2
5	**n VP**	recognized adj	3
6	**VP**	recognized n	4
7	**vi**	expansion R-2	

0	
1	P=4 **A=6** R-3

6	**VP**	B=1 back to A=6	4
7	**vt NP**	expansion R-3	
8	**NP**	recognized vt	5
9	**n**	expansionR-5	
8	**NP**	B=2 back to A=8	5
9	**det n**	expansion R-6	
10	**n**	recognized det	6
11	**-**	recognized n	7

0	
1	P=4 **A=6** R-4
2	P=5 **A**=8 R-6
1	cf. above
2	P=5 **A=8** R-7

One parse found.
Construct parse tree with **R-1, R-7, R-3, R-6.**

8	**NP**	B=2 back to A=8	5
9	**det adj n**	expansion R-7	
10	**adj n**	recognized det	6

1	cf. above

6	**VP**	B=1 back to A=6	4
7	**vt NP PP**	expansion R-3	
8	**NP PP**	recognized vt	5
9	**n PP**	expansionR-5	

0	
1	P=5 **A=8** R-6

8	**NP PP**	B=1 back to A=8	5
9	**det n PP**	expansion R-6	
10	**n PP**	recognized det	6
11	**PP**	recognized n	7
12	**prep NP**	expansion R-8	

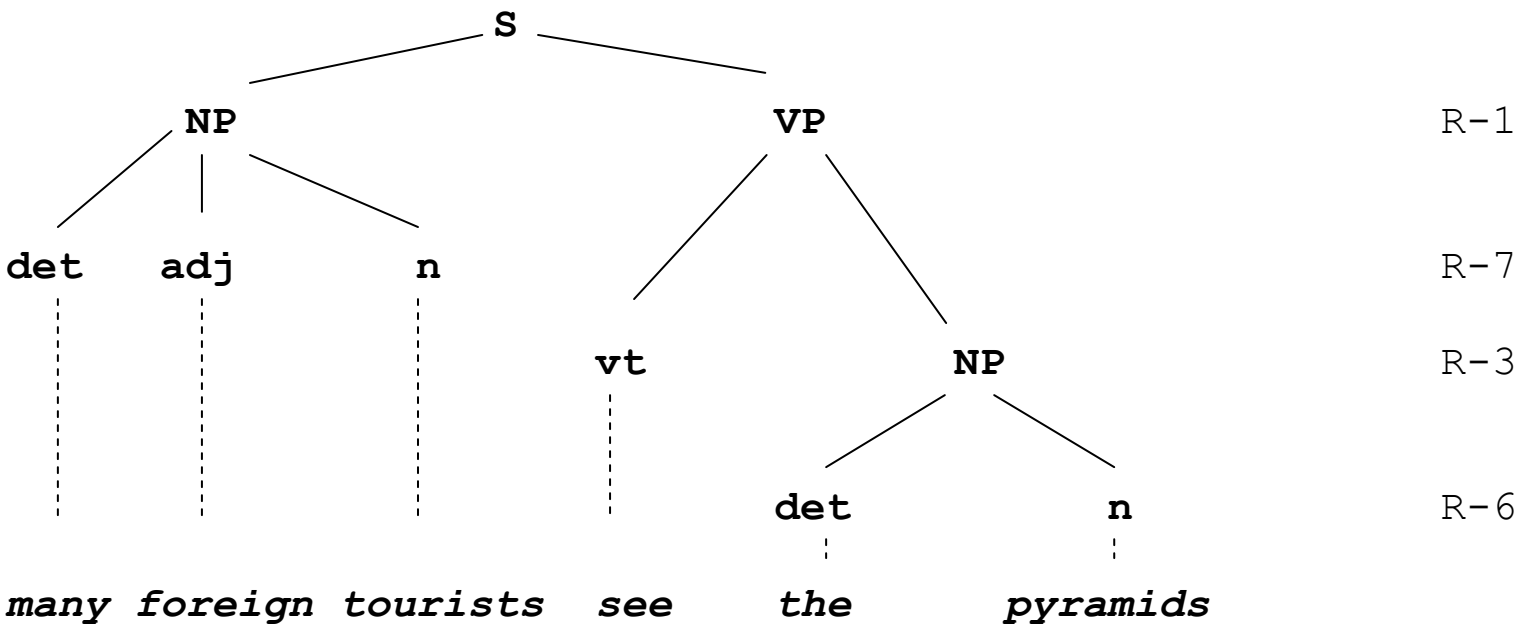
0	
1	P=5 **A=8** R-7

Position is out of range.

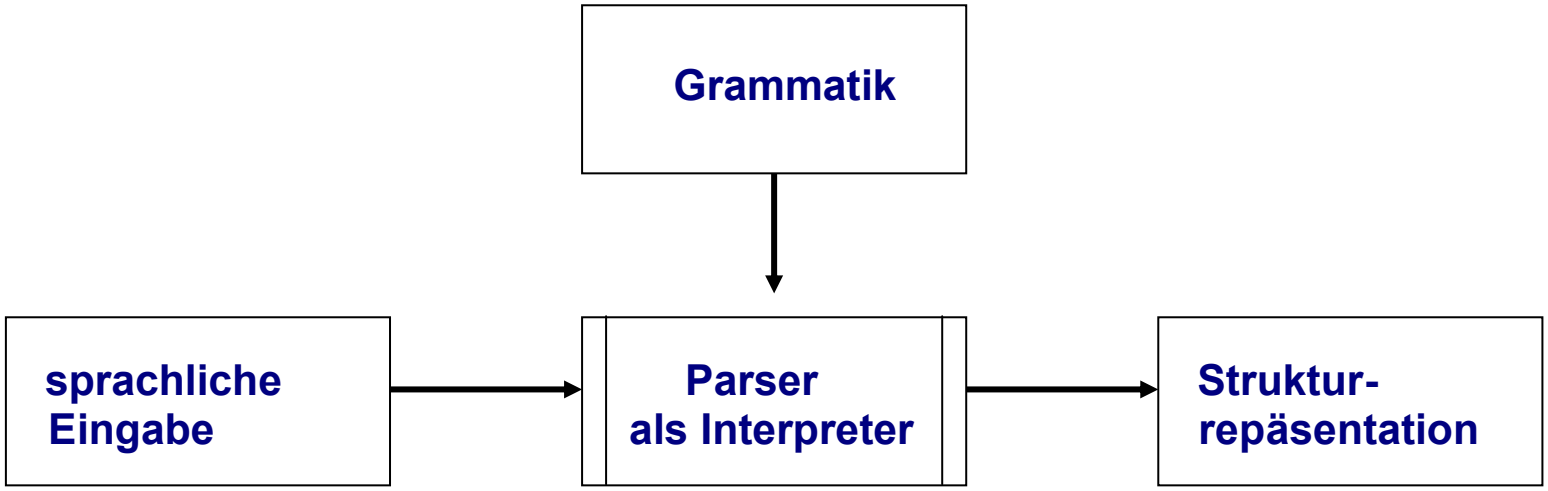
8	**NP PP**	B=1 back to A=8	5
9	**det adj n PP**	expansion R-7	
10	**adj n PP**	recognized det	6

0	

Key for constructing the parse tree: **1,7,3,6**

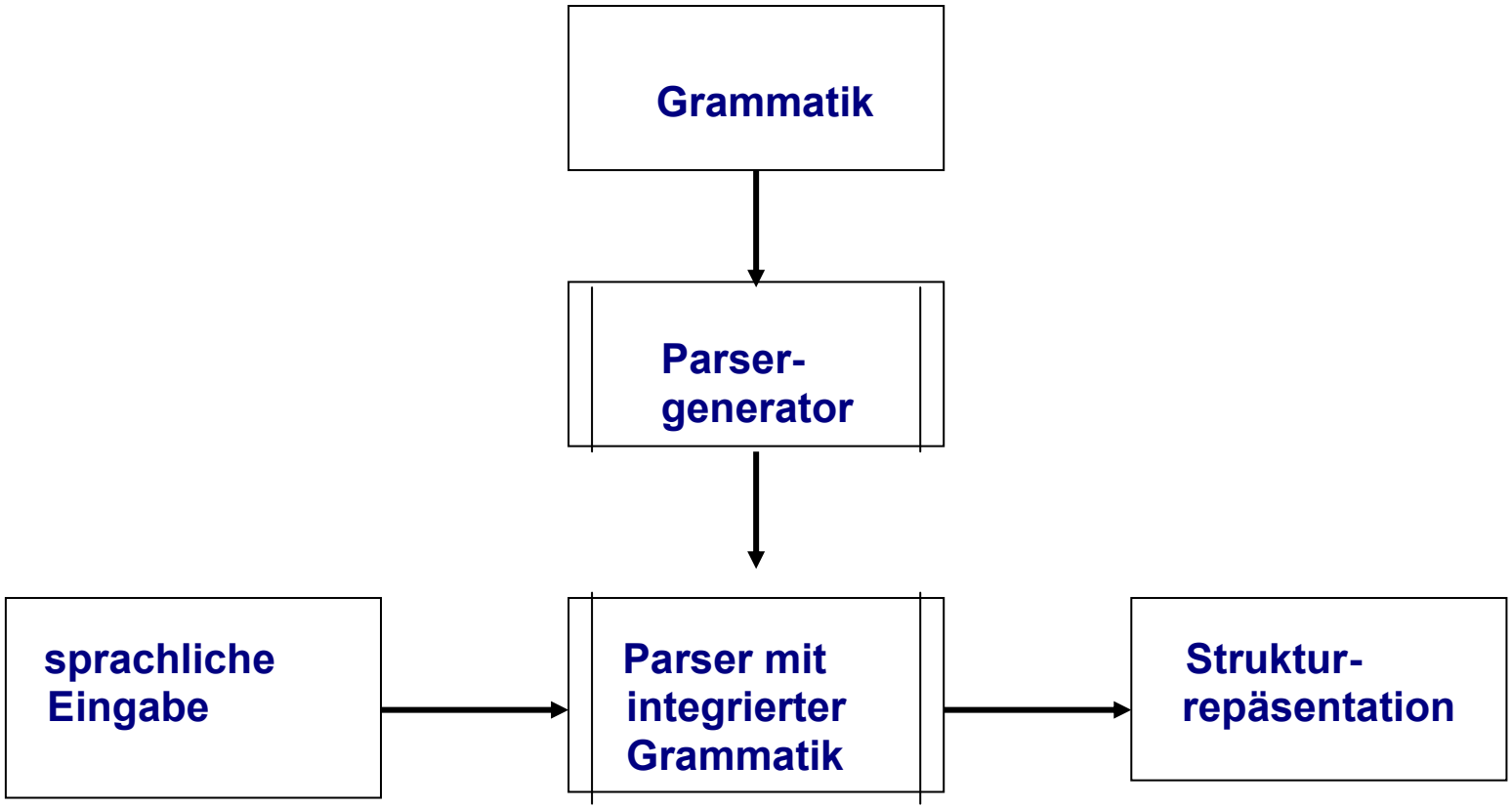


Verhältnis Grammatik und Parser



1. Interpretierender Parser

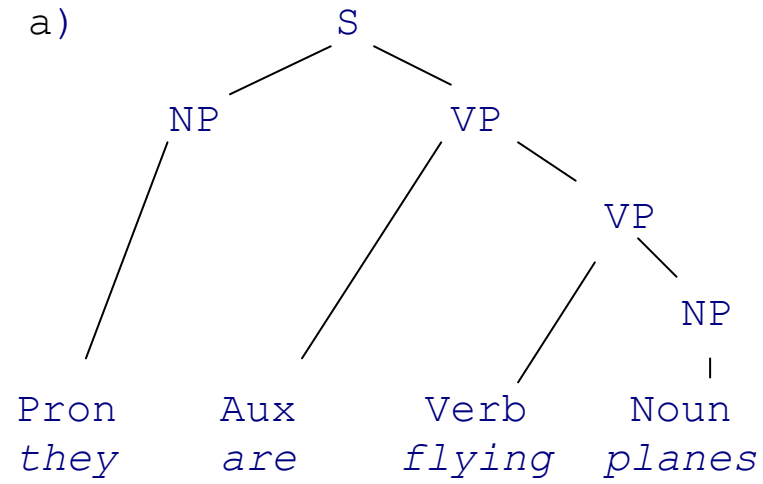
2. Prozeduraler Parser

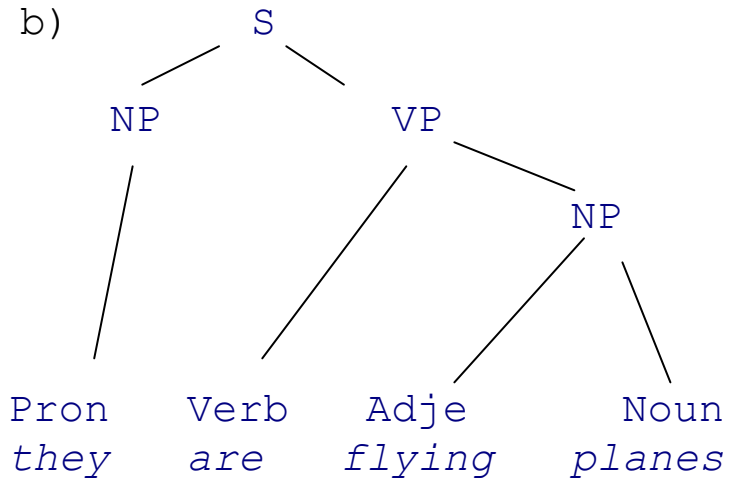


3. Compilierter Parser

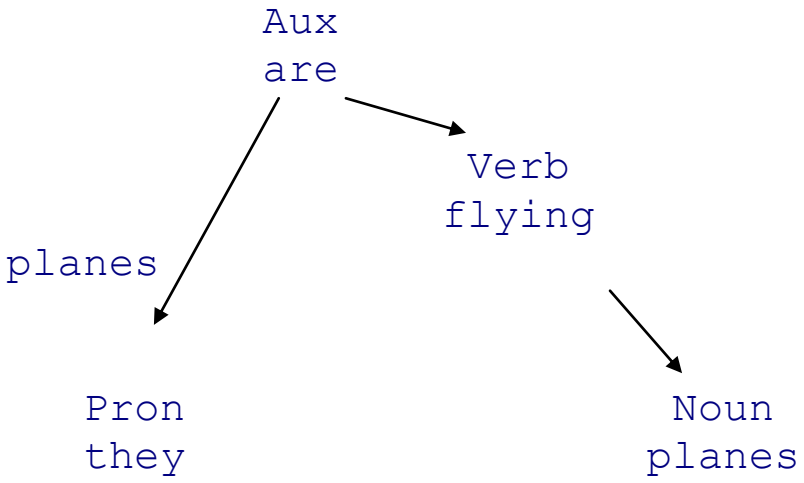
Grammatiktyp

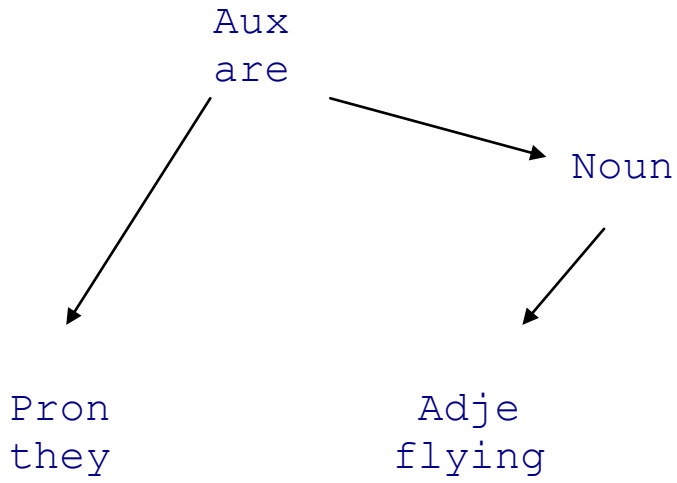
Konstituentenstruktur



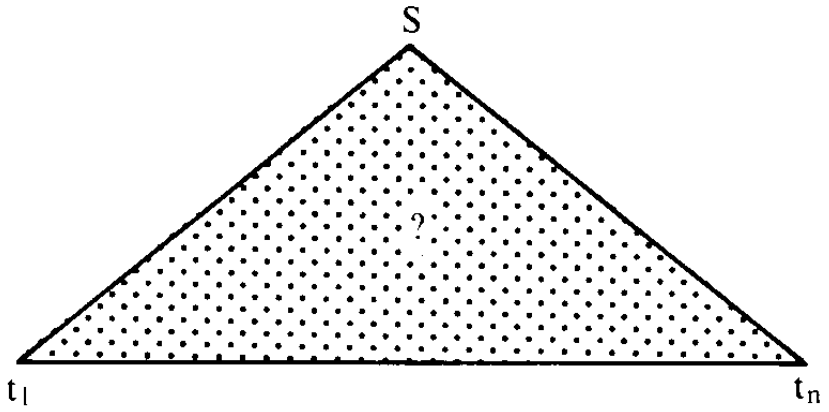


Dependenzstruktur

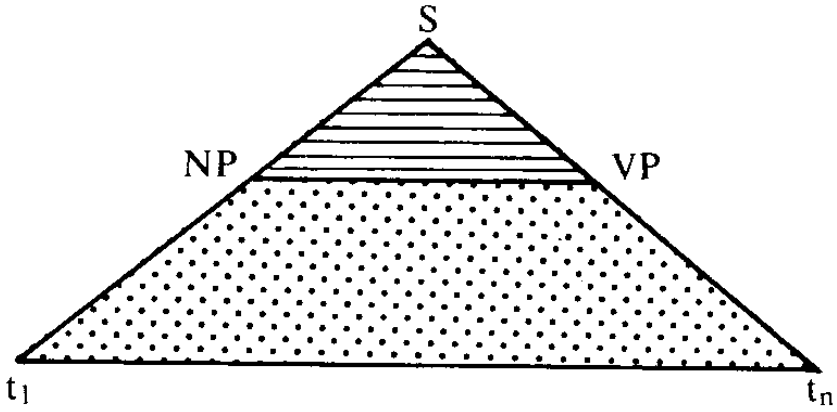




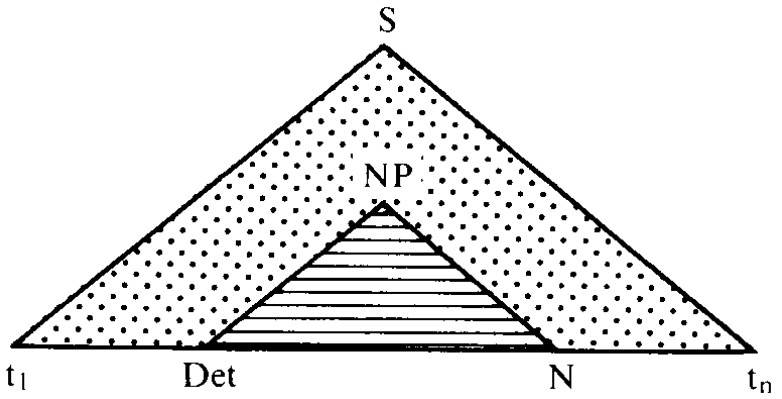
Erkennungsprozedur



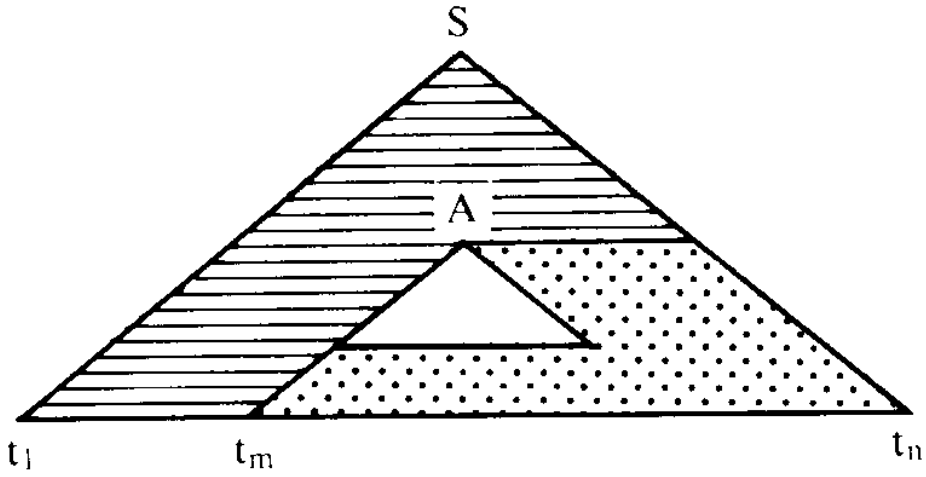
der Suchraum



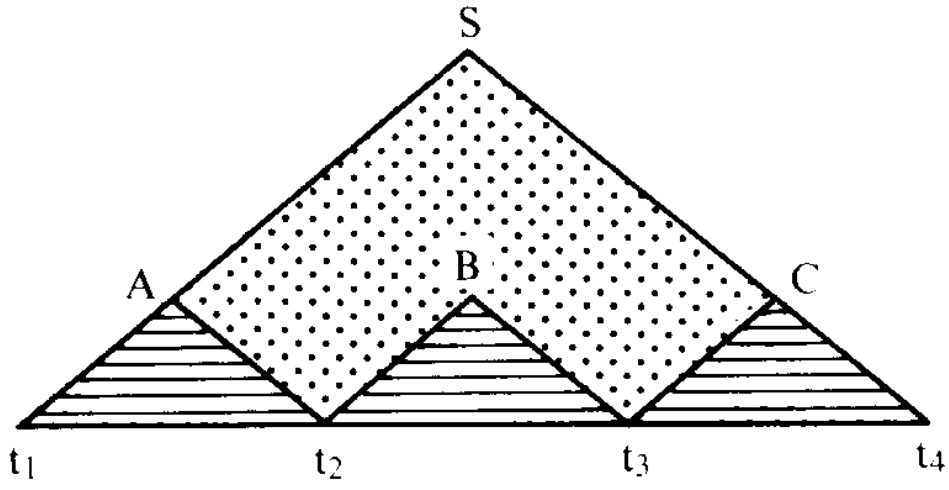
top-down Richtung



bottom-up Richtung



Tiefe-zuerst



Breite zuerst

Checklist PT-1

(1) connection between grammar and parser

- interpreting parser PT-1
- procedural parser
- compiled parser

(2) Linguistic structure assigned

- constituency descriptions PT-1
- dependency descriptions

(3) Grammar specification format

- production rules PT-1
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) PT-1
- category reduction (bottom-up)
- state transition
- slot filling

(5) Processing the input

- from left to right or from right to left PT-1
- one-pass (depth-first) PT-1
- several passes (breadth-first)
- left-associative PT-1
- non-continuously (island parsing)

(6) Handling of alternatives

- backtracking PT-1
- parallel processing
- looking ahead

(7) Control of results

- goal oriented recognition of final result(s) PT-1
- all intermediate results stored (chart)

3. Stunde: Stacks und rekursive Programmaufrufe

Struktogramm für PT-1 Top-down Parser with Backtracking:

Parser:

Lexicon phase

Initialisation

Repeat until **current position** exceeds last word in the input

Expansion

Recognition

if no recognition

then **Backtracking**

if no backtracking then reject input

If working space is empty then accept input, else reject

Lexicon phase:

assign **category** (or categories) to each **word** in the input.

Initialisation:

Set **current position** to position of the 1st word.

The first entry of the working space is the **current derivation** with the symbol S in it.

The first symbol in the current derivation ist the **due symbol**

The **current rule** is the first rule.

The **backtracking stack** is empty

Expansion:

loop through rules, beginning with current rule

if due symbol equals symbol on left side of rule

create new derivation: replace due symbol of current derivation by right side of rule; store new derivation in new entry of working space

if due symbol is equal to another rule

put current position, current entry and next rule on backtracking stack

repeat Expansion with current rule = first rule

Recognition:

if due symbol equals one of the categories of word at current position

create new derivation:
delete due symbol of current derivation;
store new derivation in new entry of working space
Increase currect position by 1

reject

Backtracking:

Restore current position and current entry from last entry on the backtracking stack, set current rule to rule on the stack

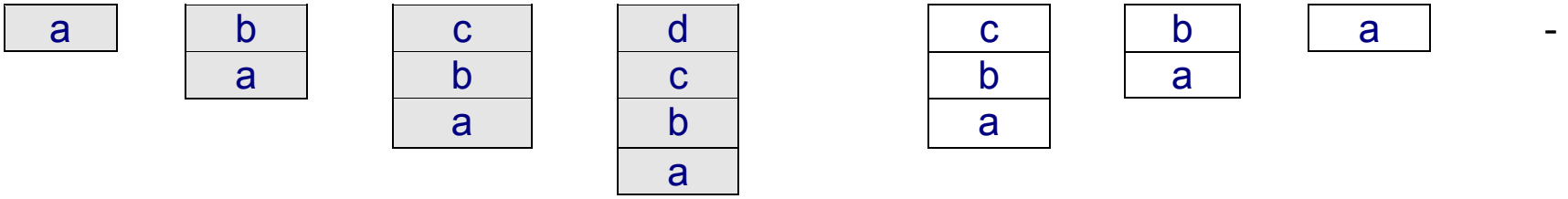
Remove current entry from stack

Stapelspeicher/Stack/ Push-down Store

- eine Liste
- zugänglich nur das letzte hineingeschriebene Element
- um an frühere Elemente zu gelangen, muß man die späteren erst entfernen
- also Prinzip: was zuletzt hinein kommt, kommt als erster wieder heraus

Hinein a, b, c, d

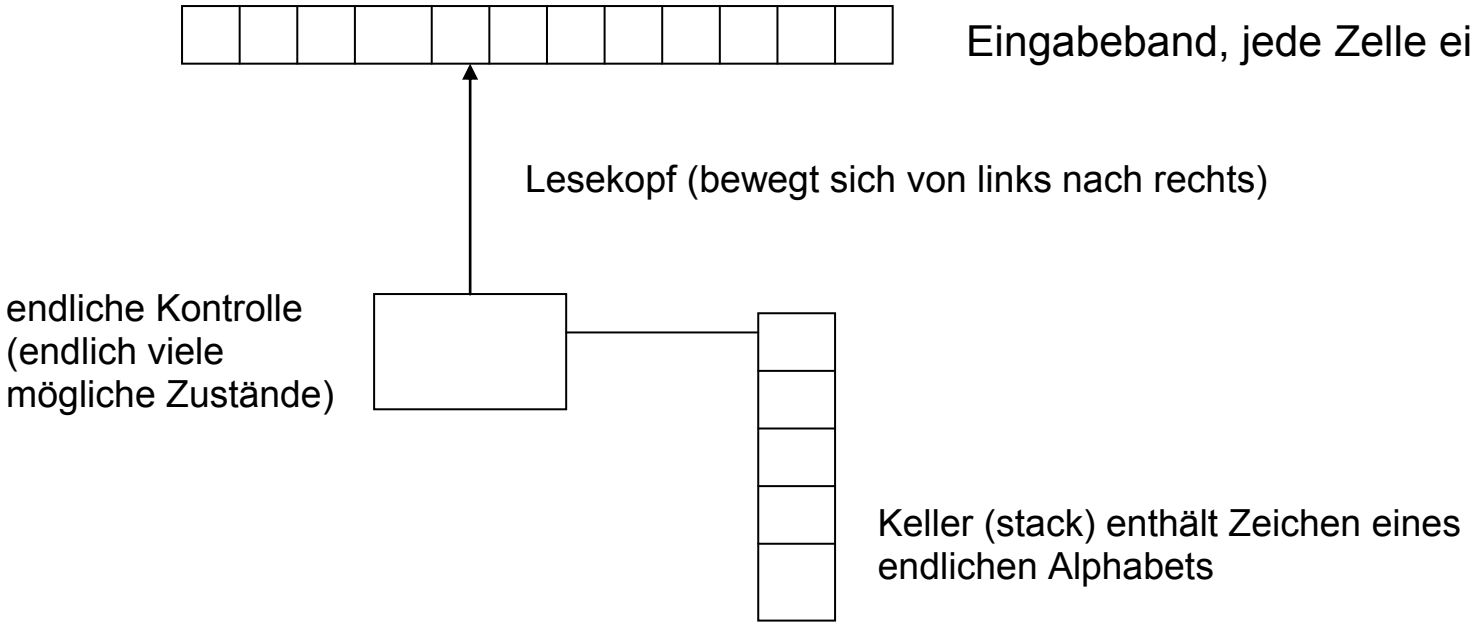
Hinaus d, c, b, a



Höhe des Stapels

Stapelspeicher eignen sich zur Kontrolle von Programmen mit bestimmtem Ablauf

Kellerautomat



ein Schritt:

abhängig vom gelesenen Zeichen, vom Zustand der endlichen Kontrolle und vom obersten Kellersymbol wird ausgeführt:

- Lese-Kopf nach rechts
- Zustand der endlichen Kontrolle ändert sich
- oberstes Keller-Symbol wird entfernt oder neues aufgelegt.

Formale Def.:

ein PUSH-DOWN-Automat ist ein 6-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$

mit

Q – *endliche Menge Zustände*

Σ - *endliche Menge Eingabealphabet*

Γ - *endliche Menge Kelleralphabet*

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow C,$

wobei C Menge endliche Teilmenge von $Q \times \Gamma^*$

q_0 – *Startzustand*

$Z_0 \in \Gamma$: *Unterstes Kellersymbol*

Kellerautomat geeignet um kontextfreie Sprachen zu erkennen

Beispiel: $S = a^n b^n$

ab, aabb, aaabbb, aaaabbbb usw.

Ablauf

- zunächst im Zustand q0: für jedes gelesene „a“ wird ein Zeichen A auf den Keller gelegt,
- falls ein „b“ gelesen wird, Übergang in Zustand q1
- in q1: für jedes gelesene „b“ wird ein A vom Keller entfernt
- sind die Eingabe leer und der Keller gleichzeitig leer, ist die Eingabe ein S

Kellerautomat auch geeignet, um Rekursionen (Schachtelungen) zu kontrollieren

Rekursion - Ein Programm ruft sich selbst auf

Beispiel: ein Programm, das einen Baum abarbeitet und die Knoten zählt:

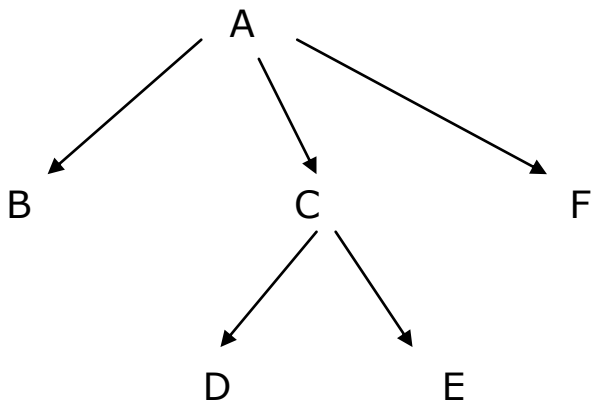
PROGRAMM (Knoten, Zahl)

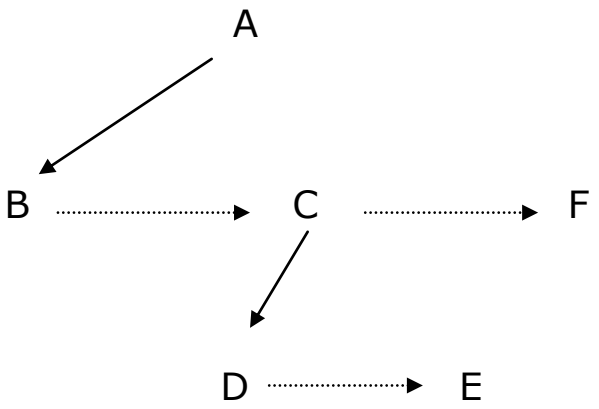
Zahl= Zahl +1

wenn Knoten Tochter hat, dann **PROGRAMM (Tochter, Zahl)**

wenn Knoten Schwester hat, dann **PROGRAMM (Schwester, Zahl)**

Ende Programm



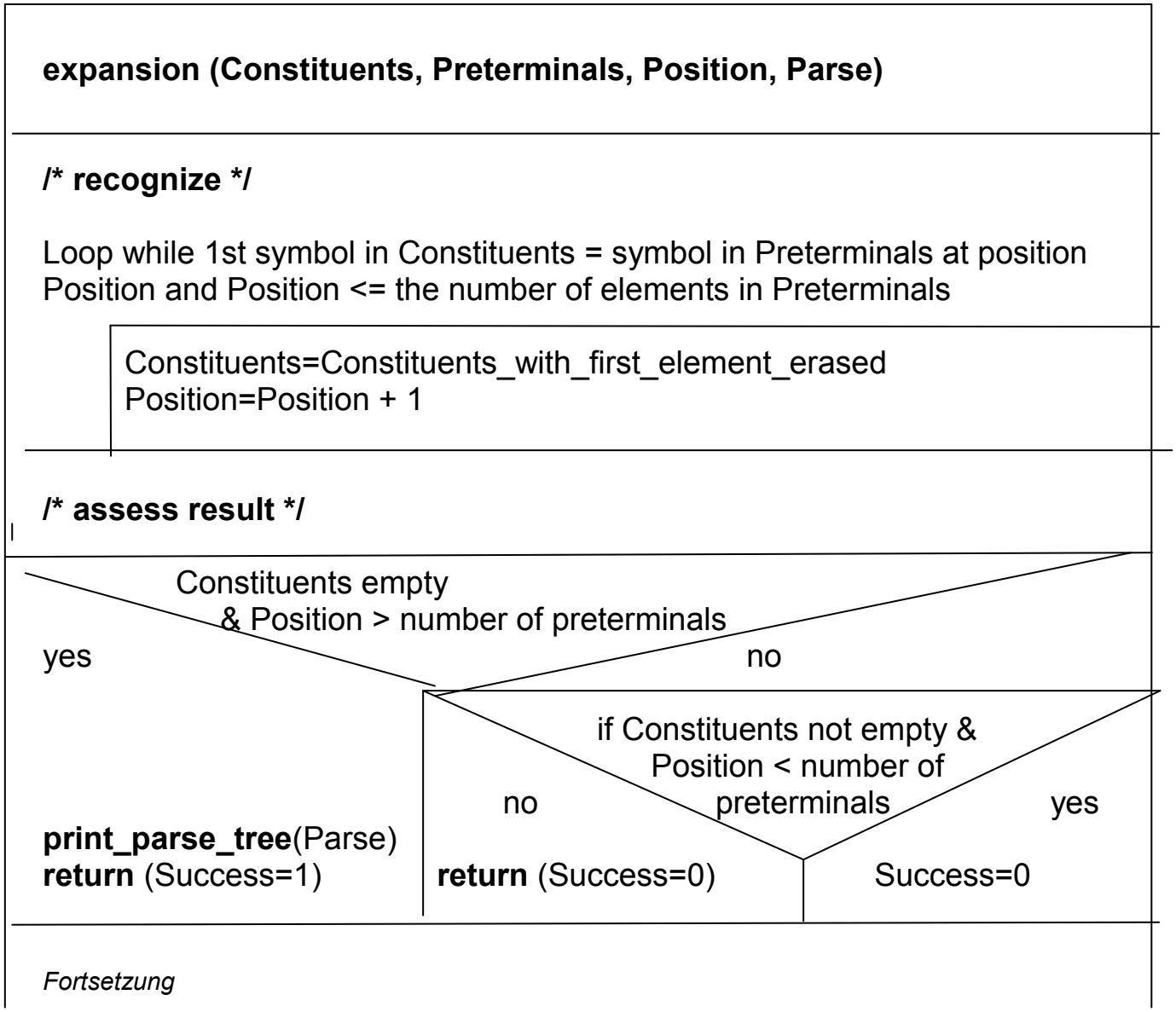


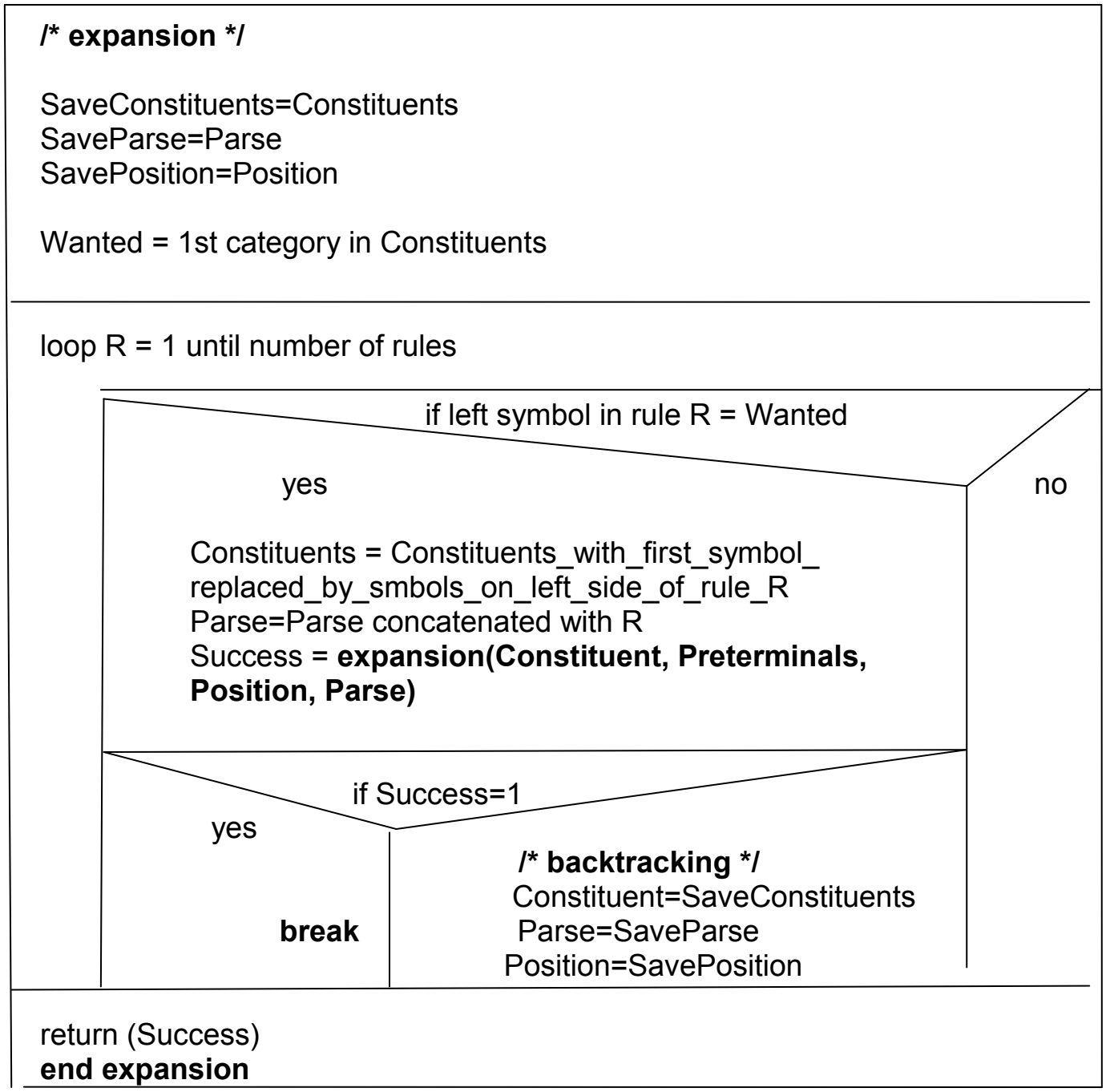

```
PROGRAMM (A, 0)
Zahl=1
Knoten=A
        PROGRAMM (B-Tochter,1)
        Zahl=2
        Knoten=B
                PROGRAMM (C-Schwester,2)
                Zahl=3
                Knoten=C
                        PROGRAMM(D-Tochter,3)
                        Zahl=4
                        Knoten=D
                                PROGRAMM(E-Schwester,4)
                                Zahl=5
                                Knoten=E
                                Ende
                        Ende
                        PROGRAMM(F-Schwester,5)
                        Zahl=6
                 Ende   Ende
        Ende
Ende
```

Top-down Parser mit Backtracking mit rekursivem Programmaufruf

Struktogramm

parser()
Sentence = **read** () /* a sentence is read from the input */
Preterminals = **look_up_lexicon** (Sentence) /* this function outputs a sequence of preterminal categories which are associated with the words in the sentence according to the lexicon */
Constituents="S" /* initial configuration */ Position=1 Parse=0 Success = **expansion**(Constituents, Preterminals, Position, Parse)
end parser


expansion (Constituents, Preterminals, Position, Parse)
/* recognize */
Loop while 1st symbol in Constituents = symbol in Preterminals at position Position and Position <= the number of elements in Preterminals
Constituents=Constituents_with_first_element_erased
Position=Position + 1
/* assess result */
Constituents empty
& Position > number of preterminals
yes
no
if Constituents not empty &
Position < number of
preterminals
no
yes
print_parse_tree(Parse)
return (Success=1)
return (Success=0)
Success=0
Fortsetzung


/* expansion */
SaveConstituents=Constituents
SaveParse=Parse
SavePosition=Position
Wanted = 1st category in Constituents
loop R = 1 until number of rules
if left symbol in rule R = Wanted
yes
no
Constituents = Constituents_with_first_symbol_
replaced_by_smbols_on_left_side_of_rule_R
Parse=Parse concatenated with R
Success = expansion(Constituent, Preterminals,
Position, Parse)
if Success=1
yes
break
/* backtracking */
Constituent=SaveConstituents
Parse=SaveParse
Position=SavePosition
return (Success)
end expansion

GRAMMAR G1

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}*

Eingabe: **They visit friends**

parser
Sentence="they visit friends"
Preterminals="n,vt,n"
Constituents="S"
Position=1
Parse=0

Success=expansion("S", "n, vt, n", 1, 0)
Success=0
SaveConstituents="S"
SaveParse=0
SavePosition=1
Wanted="S"
Loop R=1
Constituents="NP,VP"
Parse=1

Success=expansion("NP,VP", "n, vt, n", 1, 1)
Success=0
SaveConstituents="NP,VP"
SaveParse=1
SavePosition=1
Wanted="NP"
Loop R=1,2,3,4,5
Constituents="n,VP"
Parse=1-5

Success=expansion("n,VP", "n, vt, n", 1, 1-5)
Constituents="VP"
Preterminals="vt,n"
Position=2
Success=0
SaveConstituents="VP"
SaveParse=1-5
SavePosition=2
Wanted="VP"
Loop R=1,2
Constituents="vi"
Parse=1-5-2

Success=expansion("vi", "n, vt, n", 2, 1-5-2)
Success=0
SaveConstituents="vi"
SaveParse=1-5-2
SavePosition=2
Wanted="vi"
Loop R=1,2,3,4,5,6,7,8
return Success=0
end expansion

```
                Constituent="VP"
                Parse=1-5
                Position=2
                Loop R=3
                Constituents="vt, NP"
                Parse=1-5-3
                        Success=expansion("vt,NP", "n, vt, n", 2, 1-5-3)
                        Constituents="NP"
                        Preterminals="n"
                        Position=3
                        Success=0
                        SaveConstituents="NP"
                        SaveParse=1-5-3
                        SavePosition=3
                        Wanted="NP"
                        Loop R=1,2,3,4,5
                        Constituents="n"
                        Parse=1-5-3-5
                                Success=expansion("n","n, vt, n", 3, 1-5-3-5)
                                Constituents=""
                                Preterminals=""
                                Position=4
                                print_parse_tree(1-5-3-5)
                                return Success=1
                                end expansion
                        break
                        return Success=1
                        end expansion
                break
                return Success=1
                end expansion
        break
        return Success=1
        end expansion
end parser
```

Evaluation of the parser prototypes

Efficiency

The parser should be as efficient as possible both in space and time.

- control of alternatives
- overgeneration

Coverage

The capacity of the parser must cover the specific phenomena of natural language

- linguistic adequacy
- type of rules

Drawing up Lingware

Drawing up linguistic resources for the parser should be easy.

- Perspicuity
- Side effects

Evaluation PT-1

(1) Efficiency.

- Blind generation of expansions.
- Schematic backtracking, many times.
- The same work is done repeatedly because all intermediate results are lost that were created beyond the point of backtracking.
- Nevertheless, the first complete result is usually reached much earlier than the time needed to explore all possible alternatives.
- This is due to the left-to-right depth-first expansion of categories.
- The parser arrives at terminal categories that can be verified relatively soon and only such terminal categories are generated that fit into the left context.

(Besides, the algorithm of PT-1 is, in principle, the same as the proof mechanism of PROLOG. Parsers that rely on the build-in mechanisms of PROLOG (*Pereira/Warren 1983)* share the degree of efficiency with PT-1.)

(2) Coverage.

- Context-free grammar,
- however left-recursive rules are not admitted.
- A lot of phenomena of natural language are not covered by the example prototype.
- For example, the consumption technique is not suitable for calculating agreement.

(3) Drawing up lingware.

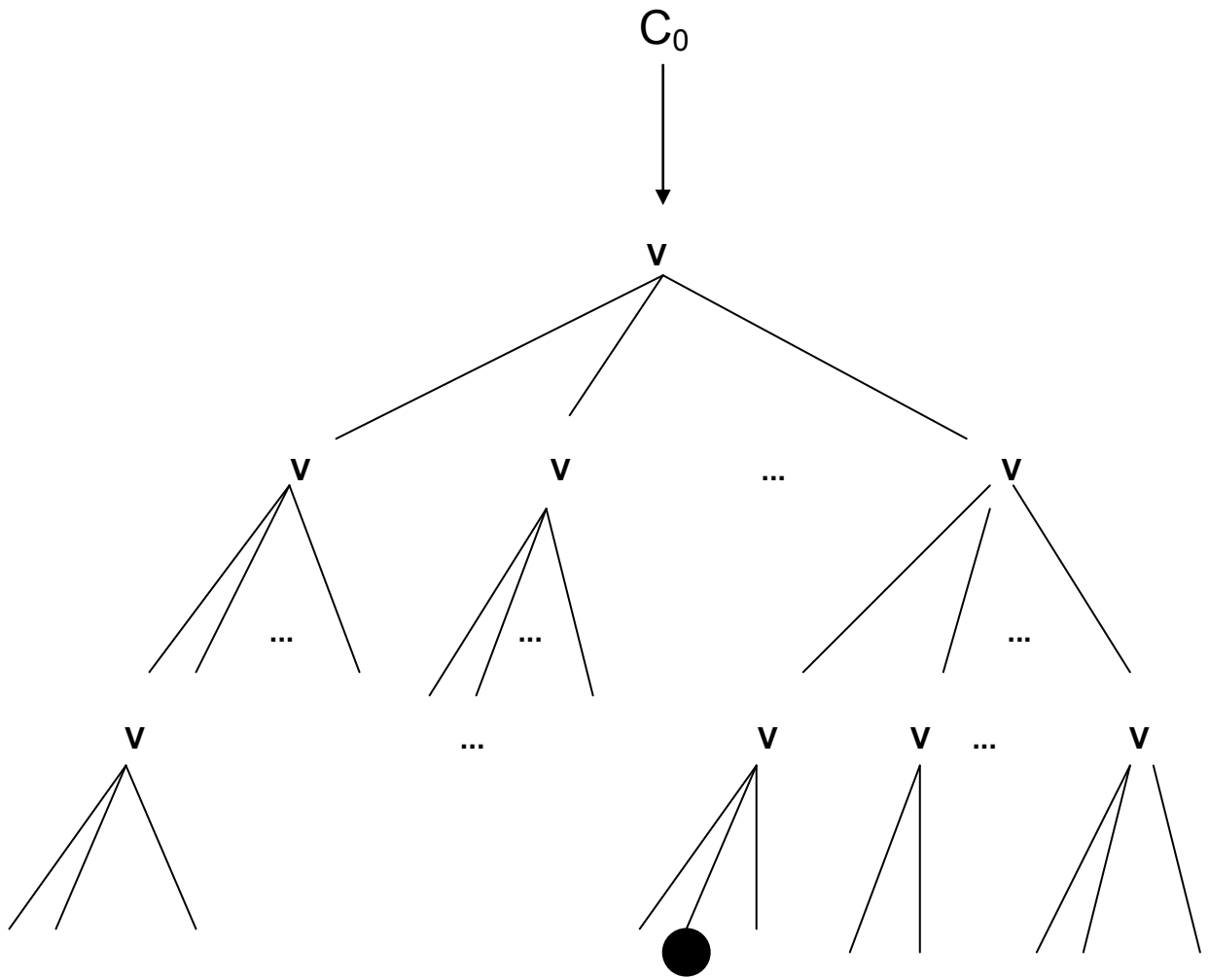
- A well-known grammar type;
- side effects result from the change of rules.

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

Das Problem mit alternativen Regeln

Deterministische Expansion

Nicht-deterministische Expansion



Struktogramm Top-down Parser with Parallel Processing:

Parser:

Lexicon phase

assign **category** (or categories) to each **word** in the input.

Initialisation

Set **current position** to position of the 1st word.
The first entry of the working space is the **only derivation** with the **symbol S** in it as the due symbol.

Repeat until **current position** exceeds last word in the input

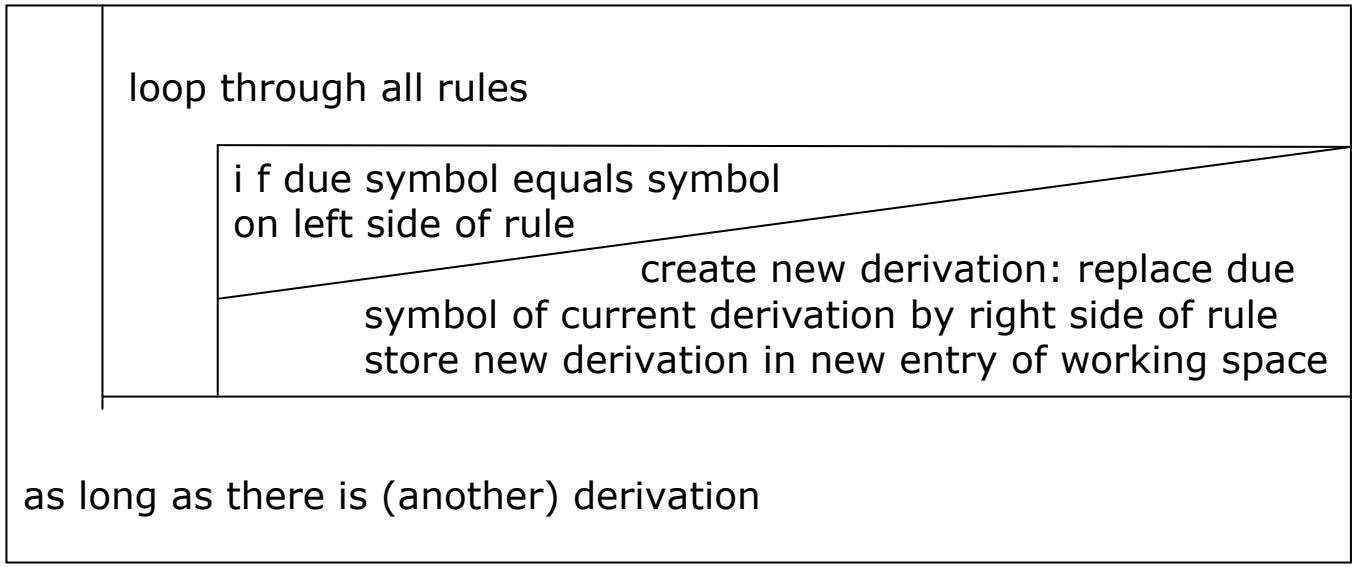
Expansion

Recognition

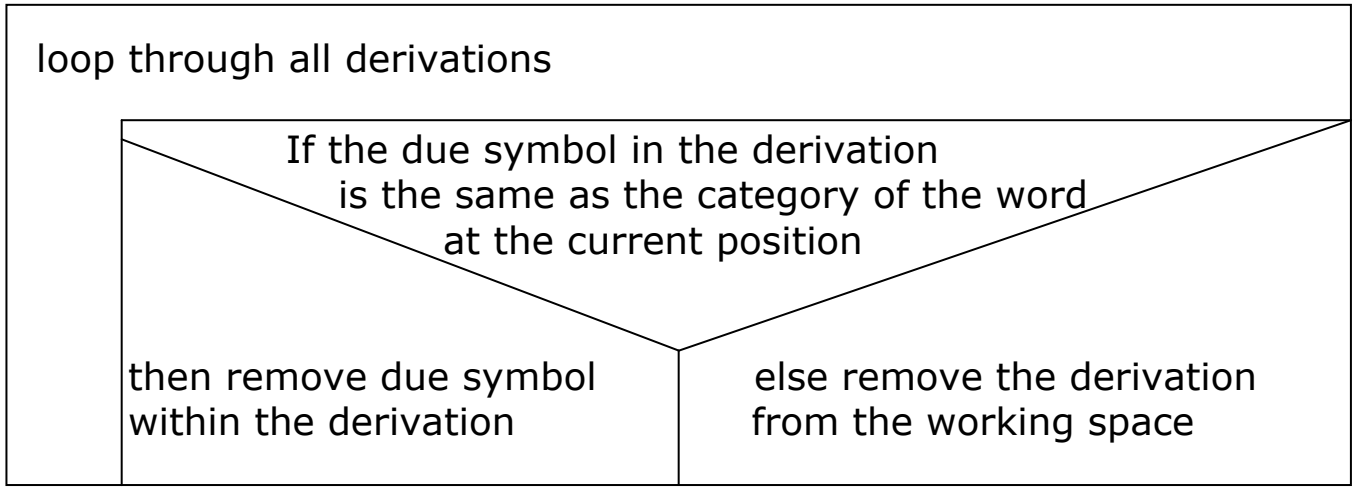
augment position

If working space is empty then accept input, else reject

Expansion:



Recognition:



GRAMMAR G1

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}*

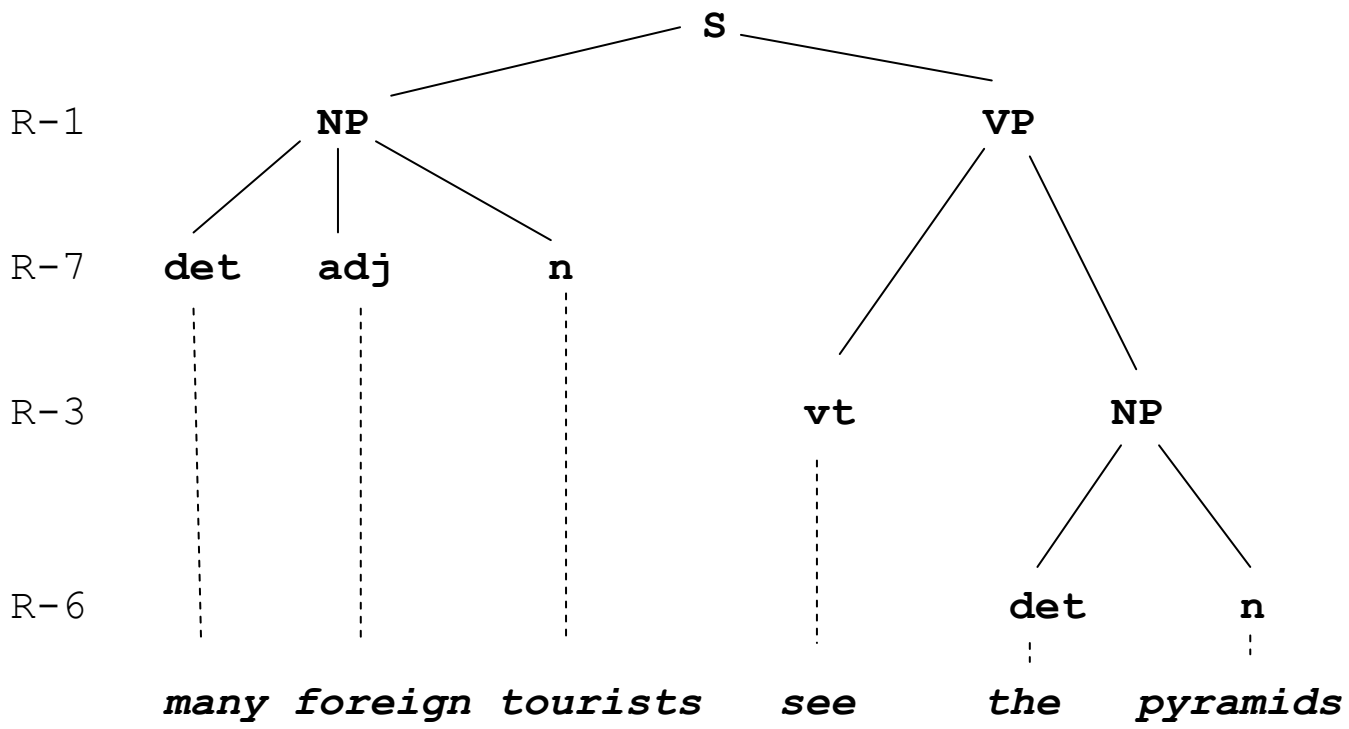
"Many foreign tourists see the pyramids"

WORKING AREA

Position	Input	Lexicon	Derivation	Explanation No.of applied rules
1	*many*	**det**	S NP VP n VP **det n VP** **det adj n VP**	start 1 1,5 1,6 1,7
			n VP **adj n VP**	1,6 1,7 recognized det
2	*foreign*	**adj**	*n VP*	1,7 recognized adj
3	*tourists*	**n**	VP	1,7 recognized n
4	*see*	**vt**	vi **vt NP** **vt NP PP**	1,7,2 1,7,3 1,7,4
			NP NP PP	1,7,3 1,7,4 recognized vt

5	*the*	**det**	n **det n** **det adj n** n PP **det n PP** **det adj n PP**	1,7,3,5 1,7,3,6 1,7,3,7 1,7,4,5 1,7,4,6 1,7,4,7
			n adj n **n PP** adj n PP	1,7,3,6 1,7,3,7 1,7,4,6 1,7,4,7 recognized det
6	*pyramids*	**n**	- PP	**1,7,3,6** 1,7,4,7 recognized n

Key for constructing the parse tree: 1,7,3,6


S
R-1
NP
VP
R-7
det
adj
n
R-3
vt
NP
R-6
det
n
many
foreign
tourists
see
the
pyramids

Evaluierung PT-2

Efficiency

This parser always creates the maximum number of expansions. As a consequence, the same amount of effort is needed for finding the first result as for finding all results.

On the other hand, no work is done twice as with PT-1.

Coverage and lingware

The same as PT-1.

Checklist PT-1, PT-2

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2
- procedural parser
- compiled parser

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2
- dependency descriptions

(3) Grammar specification format

- production rules **PT-1** PT-2
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2
- category reduction (bottom-up)
- state transition
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2
- one-pass (depth-first) **PT-1** PT-2
- several passes (breadth-first)
- left-associative **PT-1** PT-2
- non-continuously (island parsing)

(6) Handling of alternatives

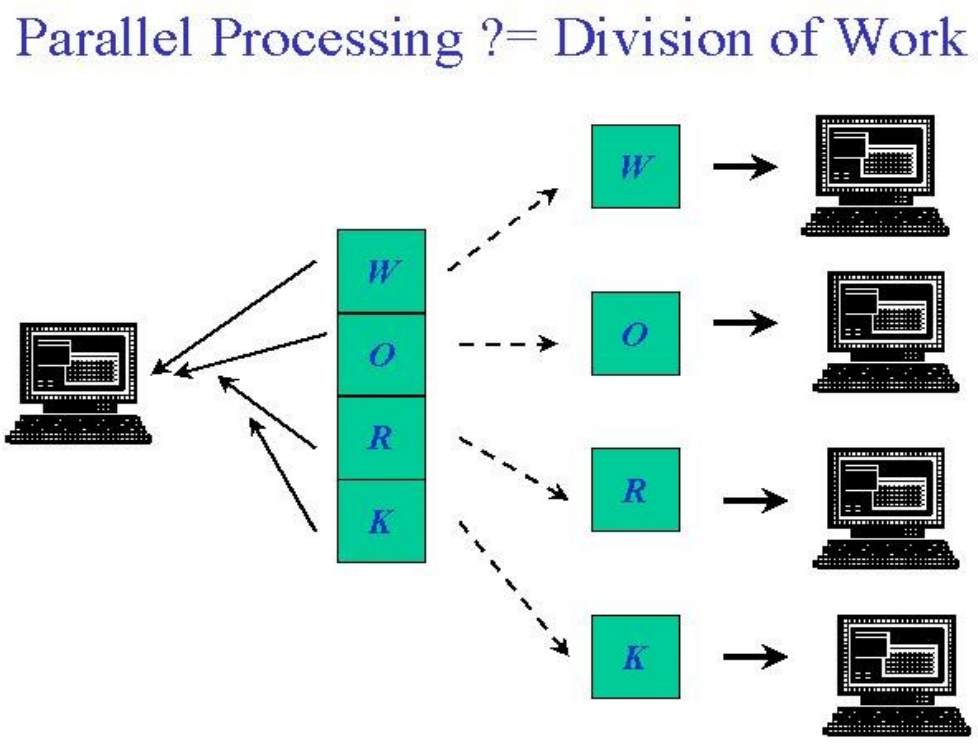
- backtracking **PT-1**
- parallel processing PT-2
- looking ahead

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2
- all intermediate results stored (chart)

Generelles zur Parallelverarbeitung

- Zerlegen einer großen Aufgabe in Teilaufgaben, die gleichzeitig ausgeführt werden können
- "Gleichzeitig" heißt mit mehreren Prozessoren



➢ ***PT-2 ist nicht wirklich ein paralleler Parser!***

Computer Architekturen

[nach Flynn, M. J., "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computing C-21, No. 9, Sep 1972, pp 948-960].

Alle diese Architekturen haben Auswirkungen auf **die Sicht der Dinge**.
Parallelverarbeitung ist in erster Linie eine geistige, konzeptuelle Aufgabe.

In unserem Fall: Was verläuft beim Sprachverstehen parallel? Eine linguistische Frage!!

(1) SISD - Single Instruction, Single Data stream

- dies ist der übliche, sequentielle Computer
- alle Befehle werden nacheinander ausgeführt
- alle Daten werden so nacheinander manipuliert

  B(I) = A(I) *4

  load (A(1))
  mult 4
  store B(1)

Im Prinzip ist das Programm zu jedem Zeitpunkt in einem bestimmten Zustand und es ist möglich, die Folge der Zustände zu rekonstruieren

.

(2) SIMD - Single instruction, Multiple data streams

- mehrere Prozessoren arbeiten synchron
- jeder führt exakt dieselben Schritte aus, aber jeder auf einer anderen Datenmenge

Für exakt parallele Probleme geeignet

z.B. Durchsuchen von vielen Dokumenten nach einem bestimmten Begriff

gemeinsame Lösung großer Aufgaben (z.B. weltweites PC-Netz)

(3) MISD - Multiple instruction, Single data stream

- gibt es kaum.

(4) MIMD - Multiple instruction, Multiple data streams

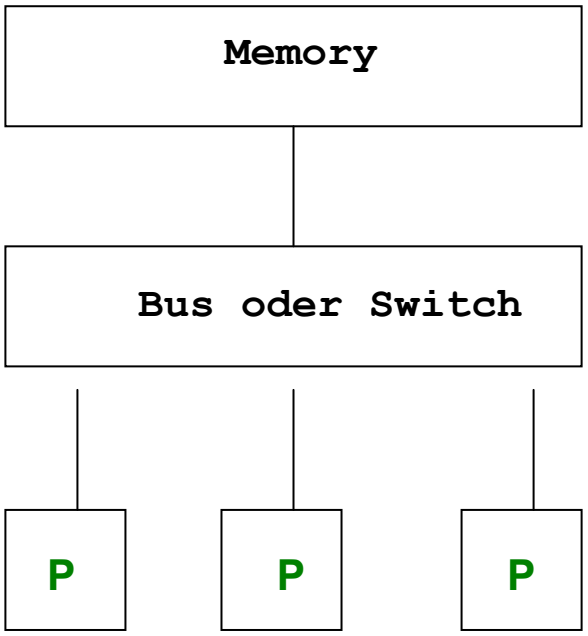
- Mehrere Prozessoren führen verschiedene Befehle auf verschiedenen Daten aus
- Problem: Kommunikation, Organisation

1. Ein Prozessor kontrolliert die anderen (process manager)

```
initialize
start task a
start task b
wait for  processor
start task c
wait for  processor
start task d
wait for all  tasks to finish
merge results
done
```

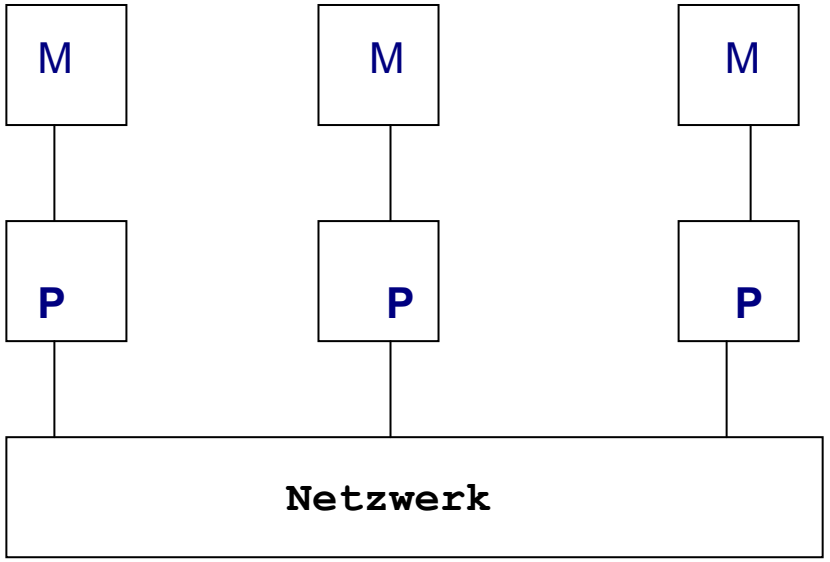
- Verteilung der Aufgaben auf verschiedene Prozessoren, wird vom Betriebssystem unterstützt und es gibt Programmiersprachen dafür.
- Automatische Verteilung von Aufgaben auf verschiedene Prozessoren: Effekt gering, wenn der Quellcode nicht darauf abgestimmt ist.

2. Shared Memory Multiprocessors



- einfach zu überschauen
- Konsistenzproblem: lock, unlock nötig
- Zugriff zum gemeinsamen Speicher wird zum Flaschenhals

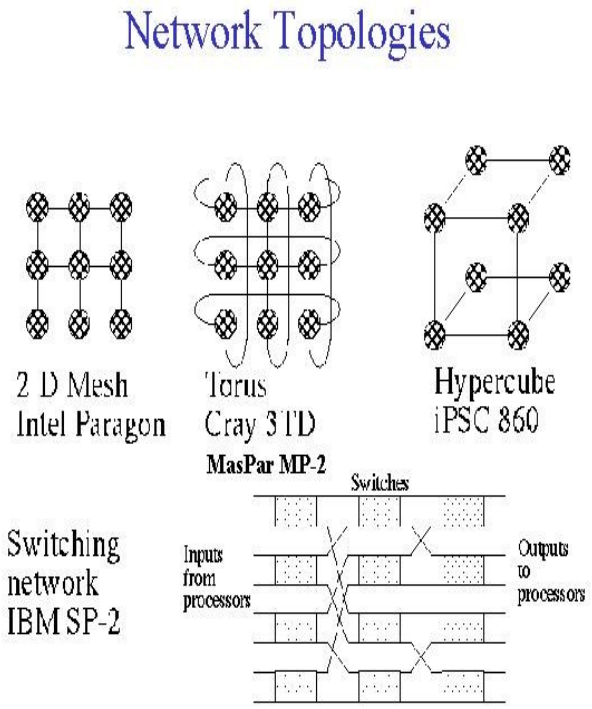
3. Distributed Memory Multiprocessors



- jeder Prozessor hat seine eigenen Daten
- benötigt er Daten oder Instruktionen von einem anderen, werden diese über ein Netzwerk verschickt (Message passing)
- Problem des Timings
  geschickter Algorithmus nötig - Aufteilung - so dass kein Prozessor warten muss

4. Channels, Queues

- Die Prozessoren sind beliebig verknüpft.
- Jeder stellt unabhängig vom Empfänger etwas in die Schlange; der Empfänger entnimmt es unabhängig vom Sender.



Die Äquivalenz von Grammatiken

- **schwach äquivalent**: G1 und G2 sind äquivalent, wenn sie dieselbe Menge Zeichenketten akzeptieren (d.h. ist Sprache L1 = L2?).

Welche der folgenden Grammatiken sind äquivalent?

I.

```
S -> a   X   b
X -> X   b
X -> a   X
X -> a
```

II.

```
S -> X
X -> a   Y
Y -> b   X
Y -> b
```

III.

```
S -> X   Y
X -> a
X -> a  X
Y -> Y   b
Y -> b
```

- **stark äquivalent:** G1 und G2 sind stark äquivalent, wenn sie schwach äquivalent sind (sie akzeptieren dieselbe Sprache) und die Strukturen, die G1 und G2 den Zeichenketten zuordnen, isomorph sind (m.a.W. wenn sie dieselben "Bäume" generieren).

Normalform-Grammatik

- Unter einer Menge von schwach äquivalenten Grammatiken wird eine ausgewählt.

- Es gibt jeweils einen Algorithmus um jede der schwach äquivalenten Grammatiken in die ausgewählte umzuformen. Die ausgewählte Grammatik ist die sog. "Normalform".

Greibach Normalform

G sei eine kontextfreie Grammatik <N, T, R, S>

G ist in Greibach-Normalform, wenn es nur Regeln der Form

A -> a B
A -> a gibt, mit a Є T und B Є N*

d.h. jede Regel beginnt mit einem terminalen Symbol auf der rechten Seite, ggf. gefolgt von einem oder mehreren nicht-teminalen Symbolen.

Zu jeder kontextfreien Grammatik G gibt es eine schwach äquivalente kontextfreie Grammatik G´ in *Greibach-Normalform.*

Herstellen der Greibach Normalform

1. Regeln der Grammatik nacheinander anwenden, linke Symbole durch die Symbole auf den rechten Seiten der Regeln solange ersetzt, bis lauter Regeln entstanden sind, deren erste unmittelbare Konstituente eine lexikalische ist.

   also top-down Tiefe-zuerst expandieren, wie bei PT-1

2. Einführung von Hilfssymbolen, so dass alle übrigen Konstituenten außer der ersten in den Regeln nicht-lexikalische Kategorien haben.

Predictive Analyzer für Greibach-Normalform Grammatiken

- Top-down mit Rücksetzen

- dabei wird Anwendung auf Regeln beschränkt, deren linker Aufhänger mit der Kategorie des nächsten Elements in Eingabe identisch ist.

Dies läßt sich so interpretieren, daß auf der Grundlage einer anstehenden Kategorie und einem Element in der Eingabe eine Vorhersage (prediction) darüber gemacht werden kann, welche Konstituenten folgen werden oder zumindest folgen können. Durch folgende Form der Grammatikregeln (nach Kuno 1965) wird dieser Gedanke noch unterstrichen:.

(A,a) | B*

oder (A,a) | ^

wobei A ein einzelnes nicht-lexikalisches Symbol, a ein einzelnes lexikalisches Symbol und B* eine Kette nicht-lexikalischer Symbole darstellen. A entspricht der Kategorie einer Produktion in Greibach-Normalform, a entspricht dem linken Aufhänger und B* dem Rest der Regel. '^' im zweiten Regelschema bedeutet, daß kein Rest vorhanden ist. Der Ausdruck links von | heißt Argumentpaar, der Ausdruck rechts davon heißt Prädiktion.

GRAMMATIK G1

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

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

Greibach Normalform, schwach äquivalent zu G1:

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

Eingabe: computer verarbeiten eingaben nach regeln

Lexikon: **n** **vt** **n** **präp** **n**

Position: (1) (2) (3) (4) (5)

ARBEITSSPEICHER: **EINGABE:** **RÜCKSETZPEICHER:**

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

Evaluation:

- Die Argumentpaare der prädiktiven Grammatik stellen eine Verbindung zwischen top-down und bottom-up Informationen dar. Die prädiktive Analyse ist also zugleich erwartungs- und datengesteuert. Das macht sie relativ effizient (8 statt 34 Symbol-Ersetzungen bei PT1, 1 x Rücksetzen statt 8 mal)

- Grammatik in Greibach Normalform sehr umfangreich (3500 Regeln für das Englische), aber direkter Zugriff auf die passenden Regeln möglich, sodass Menge nicht ins Gewicht fällt.

- Nachteil: Die Phrasenstrukturbäume der prädiktiven Grammatik sind völlig andere als G1. Die Originalgrammatik und die Greibach-Grammatik sind eben nicht stark äquivalent.

Top-down Parser mit Vorausschautabelle

Matrix mit terminalen und nicht-terminalen Symbolen. Felder enthalten die Regeln nach denen man vom nicht-terminalen Symbol zum terminalen gelangt

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

	S	NP	VP	PP
vi	0	0	2	0
vt	0	0	3,4	0
det	1	6,7	0	0
adj	0	0	0	0
n	1	5	0	0
präp	0	0	0	8

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

A	Derivation	Erklärung	P	B		B	A	P	R
(1)	**S**	Start	1	0					
(2)	**NP VP**	S/n R=1	1	0					
(3)	**n VP**	NP/n R=5	1	0					
(4)	**VP**	erkannt n	2	0					
(5)	**vt NP**	VP/det R=3	2	1		1	4	2	4
(6)	**NP**	erkannt vt	3	1					
(7)	**n**	NP/n R=5	3	1					
(8)	**-**	erkannt n	3	1					
(4)	**VP**	rückgesetzt	2	0		0			
(5)	**vt NP PP**	VP/vt R=4	2	0					
(6)	**NP PP**	erkannt vt	3	0					
(7)	**n PP**	NP/n R=5	3	0					
(8)	**PP**	erkannt n	4	0					
(9)	**präp NP**	PP/präp R=8	4	0					
(10)	**NP**	erkannt präp	5	0					
(11)	**n**	NP/n R=5	5	0					
(12)	**-**	erkannt n	6	0					

Checklist PT-1, PT-2, PT-3

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3
- procedural parser
- compiled parser

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3
- dependency descriptions

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3
- category reduction (bottom-up)
- state transition
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3
- one-pass (depth-first) **PT-1** PT-2 PT-3
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3
- non-continuously (island parsing)

(6) Handling of alternatives

- backtracking **PT-1**
- parallel processing PT-2
- looking ahead PT-3

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3
- all intermediate results stored (chart)

6. Stunde: Top-Down Chart Parser (PT-4, Early Algorithmus)

PT-4. Top-down chart parser with divided productions

(Earley Algorithmus, "Active Chart Parser")

wir haben noch drei Kasten Bier ... zu bezahlen

<table>
<tr><td>wir</td><td>haben</td><td>noch</td><td>drei</td><td>Kasten</td><td>Bier</td><td>zu</td><td>bezahlen</td></tr>
<tr><td rowspan="4">pron</td><td>verb</td><td rowspan="3">adv</td><td rowspan="2">num</td><td>noun</td><td>noun</td><td>particle</td><td>verb</td></tr>
<tr><td rowspan="3">aux</td><td colspan="2">NP</td><td colspan="2" rowspan="3">inf-verb</td></tr>
<tr><td colspan="3">NP</td></tr>
<tr><td colspan="4">NP</td></tr>
</table>

Well-formed Substring Table (sog. **Chart**): eine Verwaltungsstruktur für Teilergebnisse

- linke und rechte Grenze von Segmenten, Teilergebnis für das Segment
- Zusammenbau der Segmente

Beim Earley Parser: Vorgehen Top-down

- Verschieden mögliche Zustände
- Was von einer Regel ist abgearbeitet, was muß noch abgearbeitet werden?

Darstellung durch „divided productions“

A -> $_1$B C D

A -> B $_2$ C D

A -> B C $_3$ D

A -> B C D $_4$

(Zustände werden normalerweise bloß durch einen Punkt angezeigt)

A -> • B C D

A -> B . C D

A -> B C . D

A -> B C D .

3 Schritte abwechselnd:

je nach Situation in den Dvided Productions

- Predictor
  wenn hinter dem Punkt ein nicht-terminales Symbol kommt • NT
  -> es werden alle Productions gebildet, welche NT expandieren, jeweils mit Punkt davor

- Scanner
  wenn hinter dem Punkt ein terminales Symbol kommt • t
  -> es wird geschaut, ob in der Eingabe an betreffender Stelle das Symbol erwartet wird, wenn ja wird die lfd. Position erhöht und der Punkt um das Symbol herumgeschoben t •

- Completor
  wenn der Punkt am Ende einer Derivation steht X • , d.h. eine Konstituente ist komplett
  -> es wird in der Chart gesucht, wo an linksangrenzender Stelle ein • X steht,
  d.h. wo die fertige Konstituente Verwendung finden kann,
  -> die gefundene Derivation wird neu in den Arbeitsspeicher gestellt, wobei der Punkt über X hinweg weitergeschoben wird

GRAMMAR G2 (linksrekursiv, mehrdeutig!)

Rules
(R-1) S -> NP VP
(R-2) VP -> vi
(R-3) VP -> vt NP
(R-4) **VP -> VP PP**
(R-5) NP -> n
(R-6) NP -> det n
(R-7) NP -> det adj n
(R-8) **NP -> NP PP**
(R-9) 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}*
prep	=	*{in, by, with}*

INPUT TABLE

Input:	*they*	*study*	*fish*	*in*	*cans*
Lexicon:	**n**	**vt**	**vi/n**	**prep**	**n**
Margins:	0 1	1 2	2 3	3 4	4 5

WORKING TABLE

Section 0:

	Divided productions	Left mgn.	Right mgn.
(1)	# -> .S	0	0
(2)	S -> .NP VP	0	0
(3)	NP -> .n	0	0
(4)	NP -> .det n	0	0
(5)	NP -> .det adj n	0	0
(6)	NP -> .NP PP	0	0
(*)	NP -> .n	0	0
(*)	NP -> .det n	0	0
(*)	NP -> .det adj n	0	0
(*)	NP -> .NP PP	0	0

Section 1:

(7)	**NP -> n.**	0	1
(8)	S -> NP. VP	0	1
(9)	NP -> NP. PP	0	1
(10)	VP -> .vi	1	1
(11)	VP -> .vt NP	1	1

Zeilen mit (*) werden in Wirklichkeit nicht gespeichert!

(12)	VP -> .VP PP	1	1
(13)	PP -> .prep NP	1	1
(*)	VP -> .vi	1	1
(*)	VP -> .vt NP	1	1
(*)	VP -> .VP PP	1	1

Section 2:

(14)	VP -> vt. NP	1	2
(15)	NP -> .n	2	2
(16)	NP -> .det n	2	2
(17)	NP -> .det adj n	2	2
(18)	NP -> .NP PP	2	2
(*)	NP -> .n	2	2
(*)	NP -> .det n	2	2
(*)	NP -> .det adj n	2	2
(*)	NP -> .NP PP	2	2

Section 3:

(19)	**NP -> n.**	2	3
(20)	**VP -> vt NP.**	1	3
(21)	NP -> NP. PP	2	3
(22)	**S -> NP VP.**	0	3
(23)	VP -> VP.PP	1	3
(24)	PP -> .prep NP	3	3
(25)	**# -> S.**	0	3

Section 4:

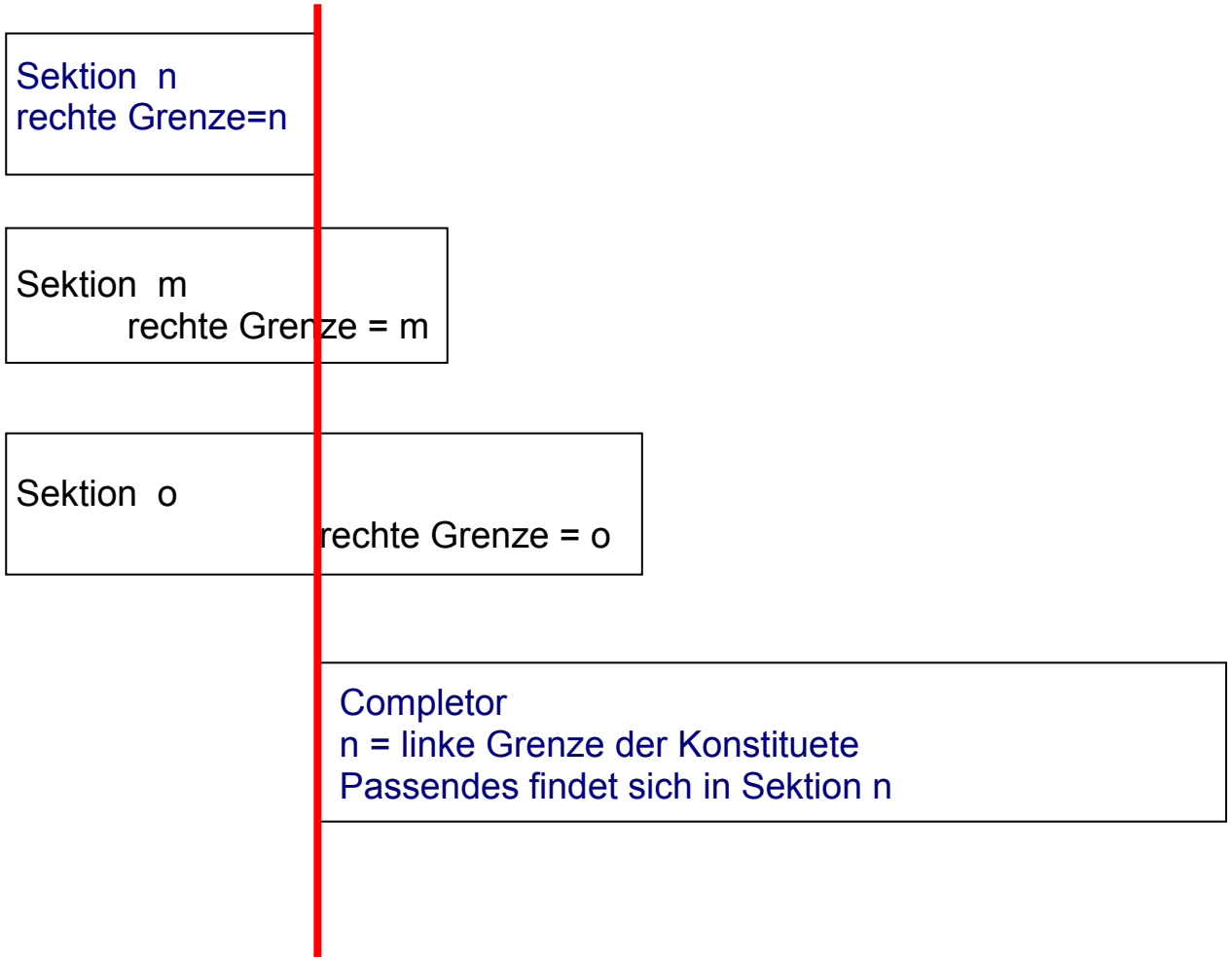
(26)	PP -> prep. NP	3	4
(27)	NP -> .n	4	4
(28)	NP -> .det n	4	4
(29)	NP -> .det adj n	4	4
(30)	NP -> .NP PP	4	4
(*)	NP -> .n	4	4
(*)	NP -> .det n	4	4
(*)	NP -> .det adj n	4	4
(*)	NP -> .NP PP	4	4

Section 5:

(31)	**NP -> n.**	4	5
(32)	**PP -> prep NP.**	3	5
(33)	NP -> NP. PP	4	5
(34)	**NP -> NP PP.**	2	5
(35)	**VP -> VP PP.**	1	5
(36)	PP -> .prep NP	5	5
(37)	**VP -> vt NP.**	1	5
(38)	NP -> NP. PP	2	5
(39)	**S -> NP VP.**	0	5
(40)	**S -> NP VP.**	0	5
(*)	PP -> .prep NP	5	5
(41)	**# -> S.**	0	5
(42)	**# -> S.**	0	5

Sektionen

praktisch zum Suchen eines linken Nachbarn im Completorschritt, der gerade die aktuelle Konstituente erwartet



Active Chart?

Active Chart Parser (Winograd 1983) ist genau dasselbe wie der Earley Parser, nur die Metaphorik ist eine andere.

Ein echter "active chart parser" ist ein solcher, in dem die nächsten Schritte von der Situation in der Chart u.U. gepaart mit einer sog. Agenda abhängig sind. Dazu kann man sich andere Abläufe ausdenken als die sture Abfolge von Predictor, Scanner, Completor.

Beispiel

Left corner Parsing

- Iniative liegt beim Scanner; dieser holt die lexikalische Kategorie des nächsten Eingabeelements in den Arbeitsbereich (als erster Eintrag einer neuen Sektion)
- Prediktor sucht nur Regel, die mit der gefundenen Kategorie beginnen
- Completer funktioniert wie üblich

-> ausprobieren!

Weitere Möglichkeiten

Produktionen können durch Greibach-Form oder durch Vorausschau eingeschränkt werden
(nur solche Produktionen, deren erste Konstituente mit nächstem Eingabesymbol übereinstimmt oder zu diesem führt)

Der Parser kann mehrere und beliebige Startkategorien bekommen:

(1) # -> S
(2) # -> NP

Der Parser kann on-line zur Texteingabe mitlaufen. Immer wenn eine #-Produktion fertig ist, wird eine Ausgabe abgespeichert. Der Benutzer kann von hinten her Text löschen, ohne dass die Analyse vor der Löschstelle wiederholt werden muss.

Reichweite von Chart Parsern

Der Early-Parser hat kein Problem mit Linksrekursion.

NP -> NP PP (siehe oben, tödlich für PT-1)

Er erlaubt Einbau von fakultativen Kategorien und Tilgungsregeln:

NP -> det (adj) N - erst wenn *det* gefunden, gibt es zwei Produktionen

NP -> det • adj N

NP -> det • N

A -> ε (=leeres Element) angewendet auf • A führt sofort zu A •

Checklist PT-1, PT-2, PT-3, PT-4

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4**
- procedural parser
- compiled parser

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4**
- dependency descriptions

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4**
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4**
- category reduction (bottom-up)
- state transition
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4**
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4**
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4**
- non-continuously (island parsing)

(6) Handling of alternatives

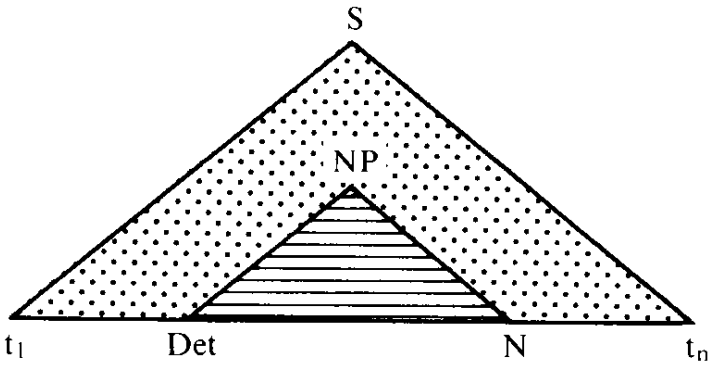
- backtracking **PT-1**
- parallel processing PT-2
- looking ahead PT-3
- welformed substring table **PT-4**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3
- all intermediate results stored (chart) **PT-4**

7. Stunde: Bottom-Up Chart Parser (PT-5, Cocke Algorithmus)

Allgemeines zum Bottom-up Parsing



...

NP -> Det N

...

Reduktion: Anwendung der Regeln Symbole rechts vom Pfeil nach Symbol links vom Pfeil, d.h. bei passenden unmittelbaren Konstituenten bilde die übergeordnete Konstituente

Zwei Aktionen: shift (vorrrücken), reduce (Regel anwenden)

Right handle = die äußerst rechte Kategorie in einer Regel
Prüfen, ob diese dem aktuellen Symbol in der Eingabe entspricht

Konflikte

- shift/reduce
- shift/shift wenn Symbol doppeldeutig
- reduce/reduce wenn weitere Regel passt

Shift-Reduce Parser mit Rücksetzen

GRAMMATIK G1 (sortiert nach rechtem Aufhänger)

```
NP ->           n      (R-5)        vi   = {rechnen, antworten}
NP ->       det n      (R-6)        vt   = {verarbeiten, erzeugen}
NP ->   det adj n      (R-7)        det  = {die, keine}
VP ->       vt NP      (R-3)        adj  = {beliebigen}
PP ->     präp NP      (R-8)        n    = {computer, eingaben,
VP ->    vt NP PP      (R-4)                regeln, antworten, disketten}
VP ->          vi      (R-2)        präp = {auf, nach}
S  ->       NP VP      (R-1)
```

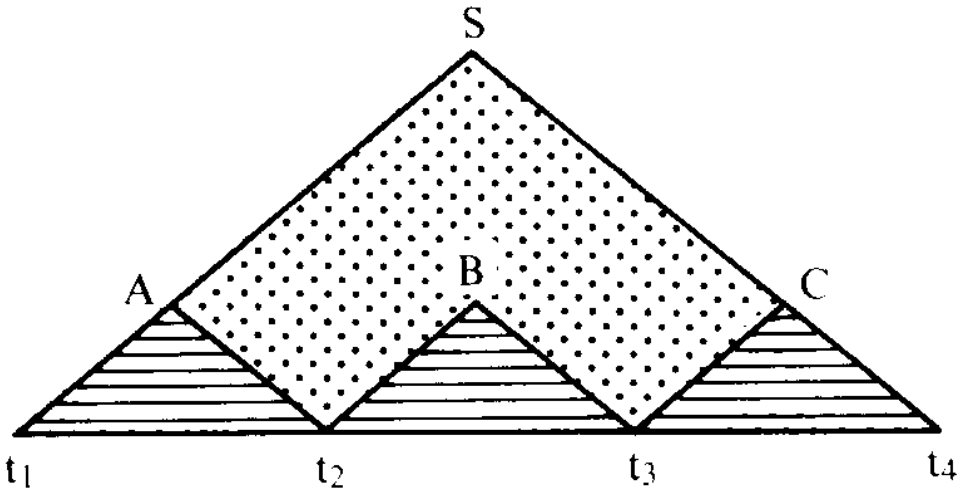
Eingabe:	**computer**	**erzeugen**	**antworten**	**nach**	**regeln**
Lexikon:	**n**	**vt**	**vi/n**	**präp**	**n**
Position:	(1)	(2)	(3)	(4)	(5)

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

```
3    (4)    NP vt vi                                          Rücksetzen B=3
4    (5)    NP vt vi präp                                     Hinzufügen präp
5    (6)    NP vt vi präp n                                   Hinzufügen n
5    (7)    NP vt vi präp NP                                  Reduktion R-5
5    (8)    NP vt vi PP                                       Reduktion R-8
-------------------------------------------------------------------------
2    (3)    NP vt                                             Rücksetzen B=2
3    (4)    NP vt n                                           Hinzufügen n
3    (5)    NP vt NP              (2)    4     S/R           Reduktion R-5
3    (6)    NP VP                 (3)    5     S/R           Reduktion R-3
3    (7)    S                     (4)    6     S/R           Reduktion R-1
4    (8)    S präp                                           Hinzufügen präp
5    (9)    S präp n                                         Hinzufügen n
5   (10)    S präp NP                                        Reduktion R-5
5   (11)    S PP                                             Reduktion R-8
-------------------------------------------------------------------------
3    (6)    NP VP                                             Rücksetzen B=4
4    (7)    NP VP präp                                        Hinzufügen präp
5    (8)    NP VP präp n                                      Hinzufügen n
5    (9)    NP VP präp NP                                     Reduktion R-5
5   (10)    NP VP PP                                          Reduktion R-8
-------------------------------------------------------------------------
3    (5)    NP vt NP                                          Rücksetzen B=3
4    (6)    NP vt NP präp                                     Hinzufügen präp
5    (7)    NP vt NP präp n                                   Hinzufügen n
5    (8)    NP vt NP präp NP                                  Reduktion R-5
5    (9)    NP vt NP PP                                       Reduktion R-8
5   (10)    NP VP                                             Reduktion R-4
5   (11)    S                                                 Reduktion R-1
                                                              Erfolg
```

(6 Zeilen mehr als PT-1, und der hat nach 14 Zeilen schon ein Ergebnis.)

Shift-Reduce Parser Breite-zuerst



GRAMMATIK G1

Regeln:

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

Lexikon:

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

Eingabe: **computer verarbeiten keine beliebigen eingaben**

Lexikon:	**n**	**vt**	**det**	**adj**	**n**
Position:	(1)	(2)	(3)	(4)	(5)

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

*) Ursprung = die zum Zeitpunkt der Erzeugung der Zeile aktuelle Folge, auf welche die neue Folge zurückgeht;
Regel = die auf die aktuelle Folge angewandte Regel.

Bottom-Up Chart Parser nach Cocke, Kasami und Younger (PT-5)

Grammatik in Chomsky-Normalform (rechts immer zwei Konstituenten)

GRAMMATIK G1

Regeln:

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

Lexikon:

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

- Die folgenden Regeln sind schon in Chomsky-Form

bisherige Grammatik	Chomsky Grammatik
S -> NP + VP	S -> NP + VP
VP -> vt + NP	VP -> vt + NP
NP -> det + n	NP -> det + n
PP -> präp + NP	PP -> präp + NP

- Regeln mit mehr als zwei IC zerlegen in mehrere Regeln mit jeweils 2 ICs, wobei man neue zusammenfassende Kategorien einführen muss:

bisherige Grammatik	Chomsky Grammatik
VP -> vt NP PP NP -> det + adj + n	VP -> vt + NPPP NPPP -> NP + VP NP -> det + Adjn Adjn -> adj + n

- Regeln mit nur einem IC auf der rechten Seite beseitigen, indem man das Symbol der rechten Seite direkt überall dort in den Regeln als Alternative einsetzt, wo das Symbol auf der linken Seite schon vorkommt.

bisherige Grammatik	Chomsky Grammatik
VP -> vi NP -> n	S -> NP + vi S -> n + VP S -> n + vi VP -> vt + n PP -> präp + n

➢ *Fazit: Die nachträgliche Umformung einer Grammatik in Chomsky Normalform ergibt ziemlich unmotivierte Konstituenten.*

Bei einem Dependenzansatz macht die Chomsky-Form eher Sinn:

Eine auf dem Head - Complement und Head - Modifier basierende Grammatik:

Rules
(R-1) $S \rightarrow N_{u,d}\ \mathbf{V_i}$
(R-2) $V_i \rightarrow \mathbf{V_t}\ N_{u,d}$
(R-3) $V_i \rightarrow \mathbf{V_i}\ PP$
(R-4) $N_d \rightarrow det\ \mathbf{N_{u,a}}$
(R-5) $N_a \rightarrow adj\ \mathbf{N_u}$
(R-6) $N_u \rightarrow \mathbf{N_u}\ PP$
(R-7) $PP \rightarrow \mathbf{prep}\ N_{u,d}$

Lexicon
$\mathbf{V_i}$ = {sleep, fish}
$\mathbf{V_t}$ = {study, visit, see, enjoy}
det = {the, no, my, many}
adj = {foreign, beautiful}
$\mathbf{N_u}$ = {tourists, pyramids, friends, fish, cans}
$\mathbf{N_d}$ = {Egypt, we, they}
prep= {in, by, with}

Input:	*they*	*study*	*fish*	*in*	*cans*
Lexicon:	N_d	V_t	V_i/N_u	prep	N_u
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)	N_d	0	1	-	-	shift *they*

Section 2:

(2)	V_t	1	2	-	-	shift *study*

Section 3:

(3)	V_i	2	3	-	-	shift *fish*
(4)	N_u	2	3	-	-	shift *fish*
(5)	V_i	1	3	**2**	4	reduce by R-2

(6)	**S**	0	3	1	**5**	reduce by R-1

Section 4:

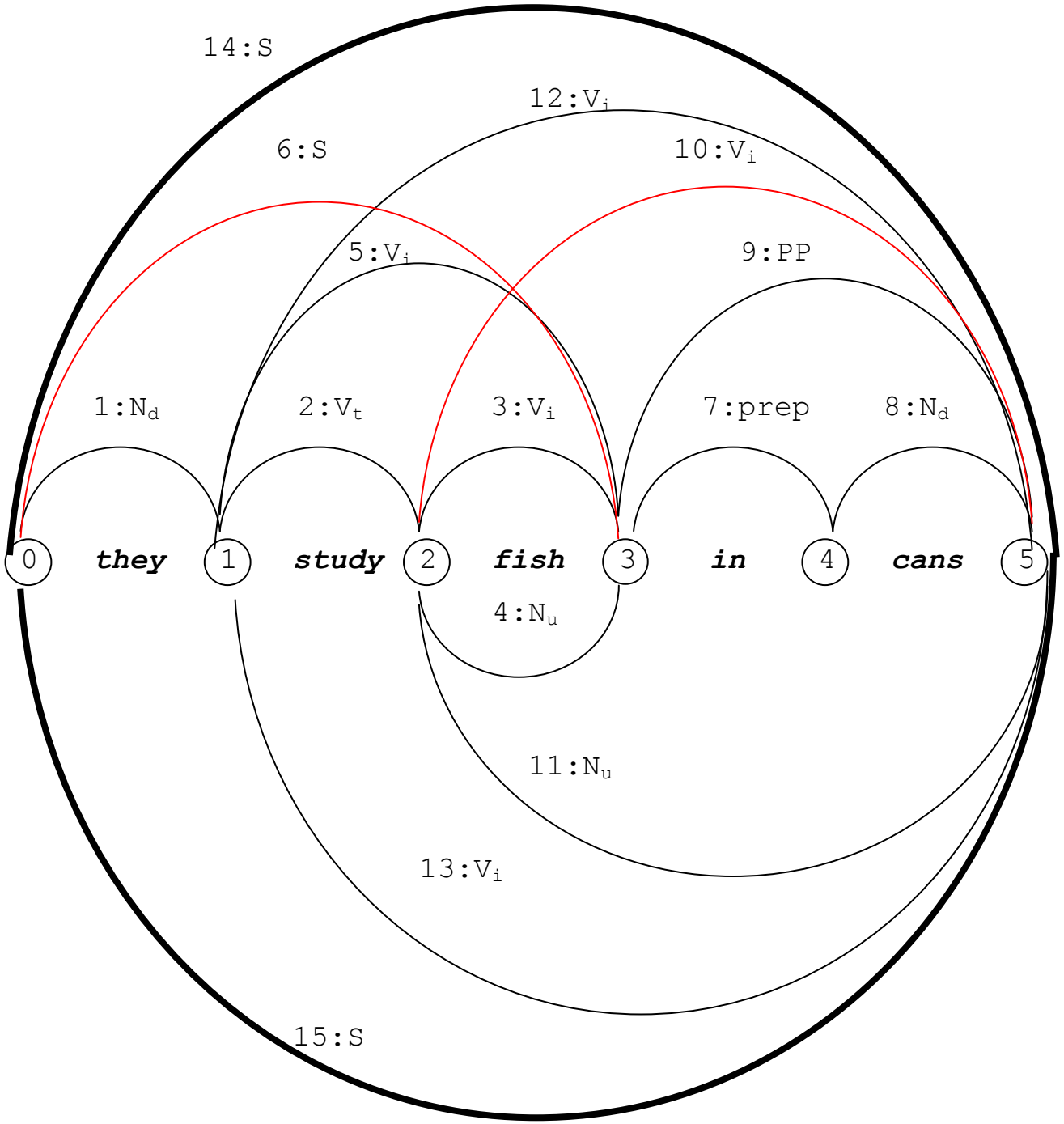
(7)	**prep**	3	4	-	-	shift *in*

Section 5:

(8)	**N_d**	4	5	-	-	shift *cans*
(9)	**PP**	3	5	**7**	8	reduce by R-7
(10)	**V_i**	2	5	**3**	9	reduce by R-3
(11)	**N_u**	2	5	**4**	9	reduce by R-6
(12)	**V_i**	1	5	**5**	9	reduce by R-3
(13)	**V_i**	1	5	**2**	11	reduce by R-2
(14)	**S**	0	5	1	**12**	reduce by R-1
(15)	**S**	0	5	1	**13**	reduce by R-1

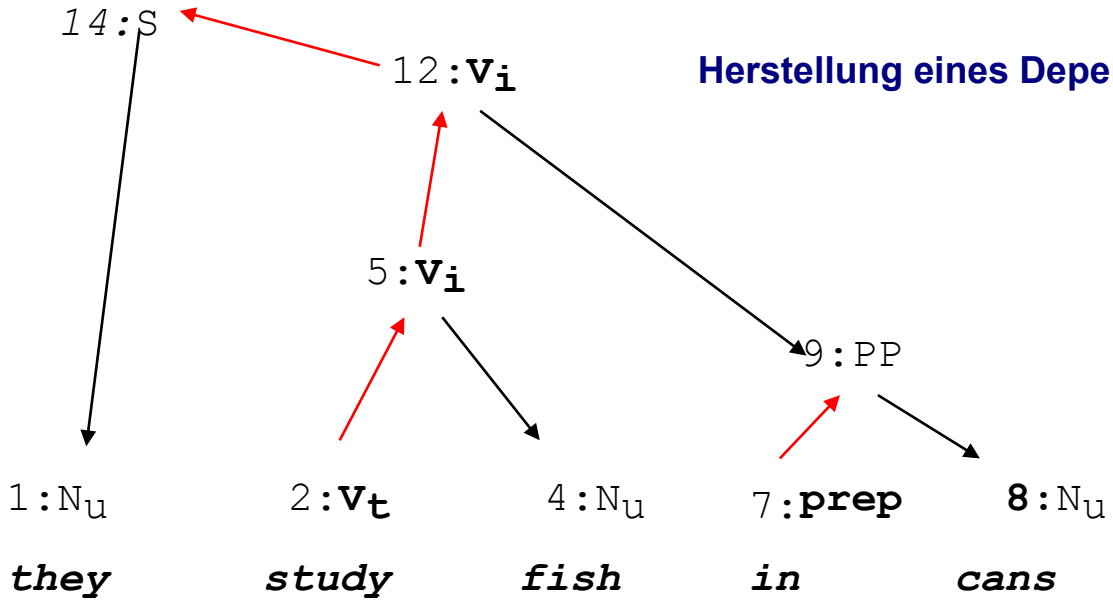
Steuerung: reduce vor shift, ein Durchlauf = Prinzip Tiefe zuerst

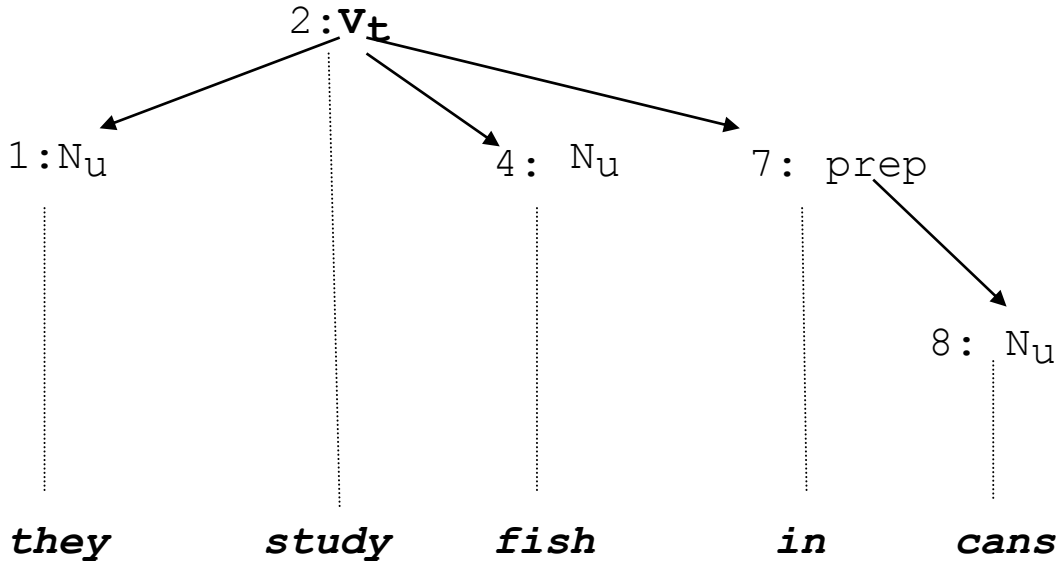
(der Original Cocke-Algorithmus ging shift vor reduce, Mehrfachdurchlauf = Breite zuerst)


14:S
12:Vi
6:S
10:Vi
5:Vi
9:PP
1:Nd
2:Vt
3:Vi
7:prep
8:Nd
0
they
1
study
2
fish
3
in
4
cans
5
4:Nu
11:Nu
13:Vi
15:S

Chart

Herstellung eines Dependenzbaumes





(14:S)

(12:V_i (1:N_u))

(5:V_i (1:N_u) (9:PP))

(2:V_t (1:N_u) (4:N_u) (9:PP))

(2:V_t (1:N_u) (4:N_u) (7:prep (8:N_u)))

(2: study (1: they)(4: fish)(7:in (8: cans)))

Verallgemeinerter Bottom-up Parser mit Teilergebnistabelle (Hellwig)

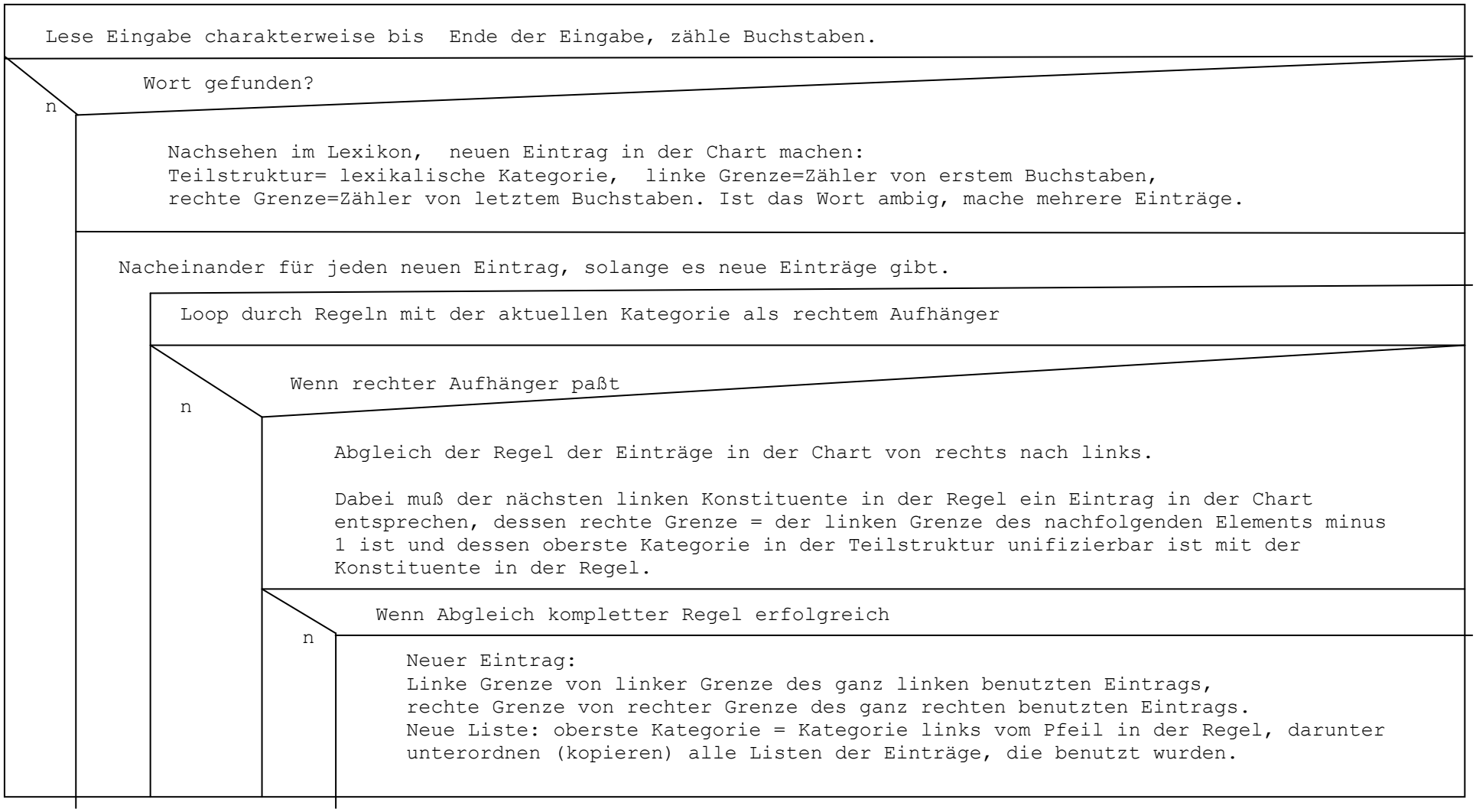
- Grammatik braucht keine Chomsky-Normalform zu sein, sondern darf Produktionen mit beliebig vielen unmittelbaren Konstituenten enthalten.

- Parser soll on-line über Satzgrenzen hinweg laufen; daher linke und rechte Grenzen als "offsets" der Buchstaben vom Anfang des Textes her angegeben.

- In jedem Eintrag der Chart wird der komplette Strukturbaum für das Segment gespeichert (also keine Erzeugung am Ende des Parsingvorgangs aus vielen Einträgen in der Chart).

- Anmerkung: Strukturbäume (Listen) als Teilergebnisse ermöglichen eine mehr als lokale Unifikation von komplexen grammatischen Kategorien der beteiligten Konstituenten (Kongruenz).

Algorithmus:



Beispielgrammatik aus der ersten Stunde:

(R-1) kategorie[satz] -> kategorie[np] kasus[nomin] person[C] numerus[C] +
kategorie[verb] person[C] numerus[C]+
kategorie[np] kasus[dativ]

(R-2) kategorie[satz] -> kategorie[np] kasus[dativ] +
kategorie[verb] person[C] numerus[C] +
kategorie[np] kasus[nominativ] person[C] numerus[C]

(R-3) kategorie[np] kausus[C] person[C] numerus[C] ->
kategorie[pronomen] kasus[C] person[C] numerus[C]

(R-4) kategorie[np] kausus[C] person[3] numerus[C] ->
kategorie[artikelwort] kasus[C] numerus[C] +
kategorie[nomen] kasus[C] person[C] numerus[C]

Lexikon:

vertraut, glaubt, hilft	kategorie[verb] person[1] numerus[sing]
er, keiner, der	kategorie[pronomen] kasus[nomin] person[3] numerus[sing]
jedem, ihm, niemandem	kategorie[pronomen] kasus[dativ] person[3] numerus[sing]
dem, seinem	kategorie[artikelwort] kasus[dativ] numerus[sing]
kein, ein, unser	kategorie[artikelwort] kasus[nomin] numerus[sing]
diesem, dem, jedem	kategorie[artikelwort] kasus[dativ] numerus[sing]
Student, Mensch	kategorie[nomen] kasus[nomin] person[3] numerus[sing]
Gauner, Lehrer, Schüler	kategorie[nomen] kasus[nomin, dativ] person[3] numerus[sing]
Studenten, Menschen, Förster	kategorie[nomen] kasus[dativ] person[3] numerus[sing]

Eingabe und Lexikon:

seinem Lehrer vertraut kein Schüler	1-7 8-14 15-23 24-28 29-36	kategorie[**artikelwort**] kasus[dativ] numerus[sing] kategorie[**nomen**] kasus[nomin, dativ] person[3] numerus[sing] kategorie[**verb]** person[1] numerus[sing] kategorie[**artikelwort]** kasus[nomin] numerus[sing] kategorie[**nomen**] kasus[nomin, dativ] person[3] numerus[sing]

Chart

lfd	LG	RG	Teilstruktur	Erklärung
(1)	1	7	(kategorie[**artikelwort**] kasus[dativ] numerus[sing])	Lexikon
(2)	8	14	(kategorie[**nomen**] kasus[nomin, dativ] person[3] numerus[sing])	Lexikon
(3)	1	14	(kategorie[**np]** kausus[dativ] person[3] numerus[sing] (kategorie[**artikelwort**] kasus[dativ] numerus[sing]) (kategorie[**nomen**] kasus[dativ] person[3] numerus[sing]))	(1)+(2) nach R-4
(4)	15	23	(kategorie[**verb**] person[1] numerus[sing])	Lexikon
(5)	24	28	(kategorie[**artikelwort]** kasus[nomin] numerus[sing])	Lexikon
(6)	29	36	(kategorie[**nomen]** kasus[nomin, dativ] person[3] numerus[sing])	Lexikon
(7)	24	36	(kategorie[**np**] kausus[nomin] person[3] numerus[sing] (kategorie[artikelwort] kasus[nomin] numerus[sing]) (kategorie[nomen] kasus[nomin] person[3] numerus[sing]))	(5)+(6) nach R-4
(8)	1	36	(kategorie[**satz**] (kategorie[**np]** kausus[dativ] person[3] numerus[sing] (kategorie[**artikelwort**] kasus[dativ] numerus[sing]) (kategorie[**nomen**] kasus[dativ] person[3] numerus[sing])) (kategorie[**verb**] person[1] numerus[sing]) (kategorie[**np**] kausus[nomin] person[3] numerus[sing] (kategorie[**artikelwort**] kasus[nomin] numerus[sing]) (kategorie[**nomen**] kasus[nomin] person[3] numerus[sing])))	(3)+(4)+(7) nach R-2

Evaluation

(1) **Efficiency**

- es wird nichts zweimal analysiert
- alle Zwischenergebnisse gespeichert
- aber erhebliche Übergenerierung (weil Linksanschluss fehlt)

.

(2) **Coverage**

- kontextfreie Grammatik
- keine Tilgungsregeln
- Dependenzstruktur möglich (bei binären Regeln für jede "Konnexion")
- Anfangssymbol braucht nicht festgelegt zu werden, dadurch gleichzeitig Parsen verschiedener Strukturen

(3) **Drawing up lingware**

- übliche PSG-Grammatiken
- komplexe Kategorien und Unifikation möglich

Checklist PT-1, PT-2, PT-3, PT-4, PT-5

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- procedural parser
- compiled parser

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- dependency descriptions **PT-5**

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4**
- category reduction (bottom-up) **PT-5**
- state transition
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4**
- non-continuously (island parsing)

(6) Handling of alternatives

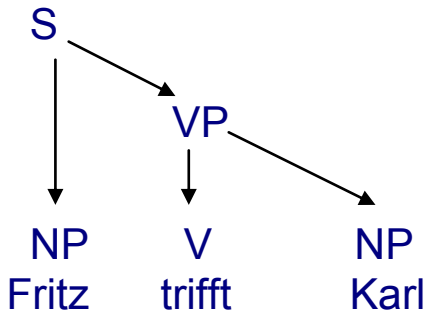
- backtracking **PT-1**
- parallel processing PT-2
- looking ahead PT-3
- welformed substring table **PT-4** **PT-5**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3
- all intermediate results stored (chart) **PT-4** **PT-5**

8. Stunde: Bottom-up Chart Parsing mit einer Kategorialgrammatik

Die Grundidee der Kategorialgrammatik (stammt aus der formalen Logik):
Ein Funktor wird auf bestimmte links oder rechts angrenzende Elemente angewendet, daraus entsteht eine Entität des gleichen oder eines neuen Typs.



- Funktor *treffen* wird auf die NP *Karl* angewandt und es entsteht ein Funktor *den Karl treffen*
- Funktor *den Karl treffen* wird auf *Fritz* angewandt und es entsteht ein Satz.

Eine Kategorialgrammatik ist völlig lexikalisiert. Es gibt keine Regeln.

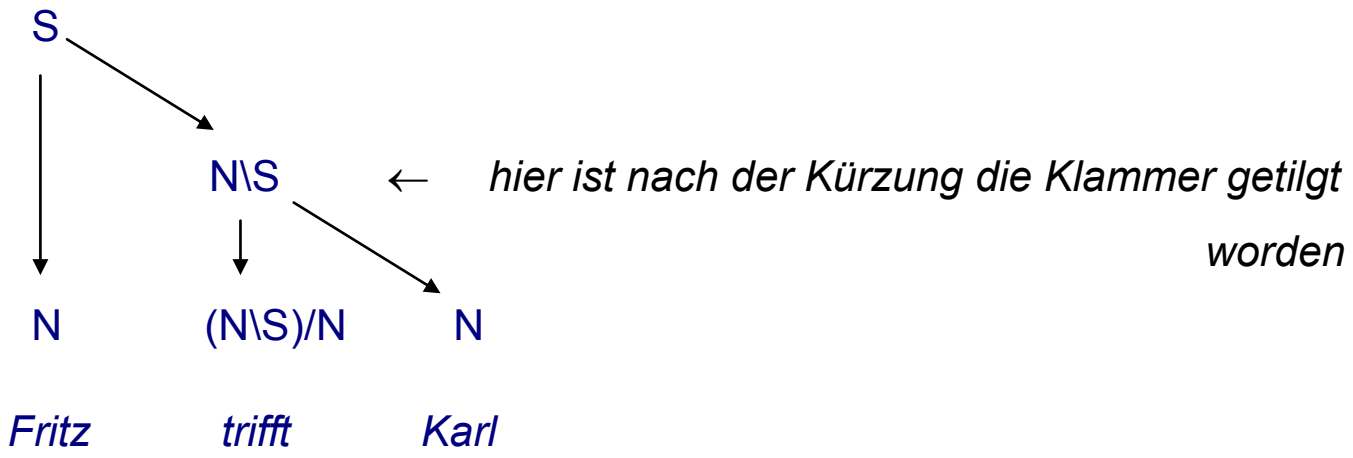
Kategorien werden in der Kategorialgrammatik den Wörtern im Lexikon zugeordnet. Sie sind wie folgt definiert:

- ein beliebiger Name ist eine Kategorie
- sind α und β Kategorien, so sind auch $(\alpha)/\beta$ und $\alpha\backslash(\beta)$ Kategorien
- umschließt ein Klammerpaar eine atomare Kategorie (einen Namen ohne Schrägstriche) so werden die Klammern getilgt

Die Wörter aus obigem Beispiel könnten im Lexikon folgende Kategorien bekommen

Fritz	**N**
trifft	(N\S)/ N
Karl	N

Analyseverfahren: Die zusammengesetzten Kategorien werden fortgesetzt um die passenden Kategorien der Elemente in der Umgebung gekürzt, bis nur noch eine Kategorie für die gesamte Eingabe übrig bleibt. Beispiel:



Normalerweise wird ein Funktor in einem Schritt auf genau ein Argument angewendet (die Zusammenfassungen sind also binär). Ein nach links oder rechts geneigter Schrägstrich zeigt in der Kategorie des Funktors an, dass das betreffende Argument unmittelbar links bzw. rechts vom Funktor stehen muss. Klammern umschließen zusammengesetzte Kategorien, die in Bezug auf die Kürzungen eine Einheit darstellen sollen.

Die allgemeine Regel für eine Reduktion mit einer rechts stehenden Konstituente ('forward cancelling')

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

Die Regel für eine Reduktion mit einer links stehenden Konstituente ('backwards cancelling')

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

Dabei sind α und β beliebige einfache oder zusammengesetzte Kategorien. Zur Abarbeitung bietet sich ein dem Cocke-Algorithmus entsprechendes Verfahren an.

Beispiel

Grammatik aus voriger Stunde ...

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

umgeformt in lexikalische Kategorien:

{sleep, fish}	**$N_{u,d}\backslash S$**
{study, visit, see, enjoy}	**$(N_{u,d}\backslash S)/N_{u,d}$**
{the, no, my, many}	**$N_d/N_{u,a}$**
{foreign, beautiful}	**N_a/N_u**
{tourists, pyramids, friends, fish, cans}	**N_u**
{Egypt, they}	**N_d**
{in, by, with}	**$(S\backslash S))/N_{u,d}$** **$(N_u\backslash N_u)/N_{u,d}$**

Scanner: Lese nächstes Wort bis Ende der Eingabe.

Entnehme Kategorie aus dem Lexikon und mache neuen Eintrag in der Chart:
Kategorie, linke Grenze des Wortes, rechte Grenze des Wortes, linkes IC leer, rechtes IC leer.
Ist das Wort ambig, mache entsprechend mehrere Einträge.

Nnacheinander für jeden neuen Eintrag, solange es neue Einträge gibt:

Completor: alle Einträge in der Chart durchgehen, deren rechte Grenze (Sektionsnummer) mit linker Grenze des aktuellen Eintrags übereinstimmt :

Wird die Kategorie des aktuellen Eintrags rechts erwartet von dem gefundenen Eintrag
(Fall [a/b] [b])
oder erwartet der aktuelle Eintrag die Kategorie des gefundenen Eintrags auf der linken Seite
(Fall [a] [a\b]) ?

n

Neuen Eintrag machen:
Kategorie = die Kategorie "über" dem Slash übernehmen (d.i. der Kürz-Vorgang),
linke Grenze von linker Grenze des gefundenen Eintrags übernehmen,
rechte Grenze von rechter Grenze des aktuellen Eintrags übernehmen.

Input:	*my*	*friends*	*in*	*Egypt*	*sleep*
Lexicon:	[$N_d/N_u,a$]	[N_u]	[$(S\backslash S))/N_u,d$] [$(N_u\backslash N_u)/N_u,d$]	[N_d]	[$N_u,d\backslash S$]
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)	[$N_d/N_u,a$]	0	1	-	-	scanner *my*

Section 2:

(2)	[N_u]	1	2	-	-	scanner *friends*
(3)	[N_d]	0	2	1	2	completed 1 by 2

Section 3:

(4)	**[(S\S))/$N_{u,d}$]**	2	3	-	-	scanner *in*
(5)	**[(N_u\N_u)/$N_{u,d}$]**	2	3	-	-	scanner *in*

Section 4:

(6)	**[N_d]**	3	4	-	-	scanner *Egypt*
(7)	**[S\S]**	2	4	4	6	completed 4 by 6
(8)	**[N_u\N_u]**	2	4	5	6	completed 5 by 6
(9)	**[N_u]**	1	4	2	8	completed 8 by 2
(10)	**[N_d]**	0	4	1	9	compledted 1 by 9

Section 5:

(11)	**[$N_{u,d}$\S]**	4	5	-	-	scanner *sleep*
(12)	**[S]**	3	5	6	11	completed 11 by 6
(13)	**[S]**	1	5	9	11	completed 11 by 9
(14)	**[S]**	0	5	10	11	completed 11 by 10

9. Stunde: Tabellengesteuerter Shift-Reduce Parser (PT-6)

Beispiel aus dem Skript "A Course in Cooking":

ACTION TABLE

Z	det	adj	n	prep	vi	vt	$
0	sh9		sh8				
1							acc
2					sh4	sh5	
3							re1
4							re2
5	sh9		sh8				
6				sh13			re3
7							re4
8				re5	re5	re5	re5
9		sh11	sh10				
10				re6	re6	re6	re6
11			sh12				
12				re7	re7	re7	re7
13	sh9		sh8				
14							re8

GO TO TABLE

NP	PP	VP	S
2			1
		3	
6			
	7		
14			

Grammar G1

(re1) **S -> NP VP**

(re2) **VP -> vi**

(re3) **VP -> vt NP**

(re4) **VP -> vt NP PP**

(re5) **NP -> n**

(re6) **NP -> det n**

(re7) **NP -> det adj n**

(re8) **PP -> prep NP**

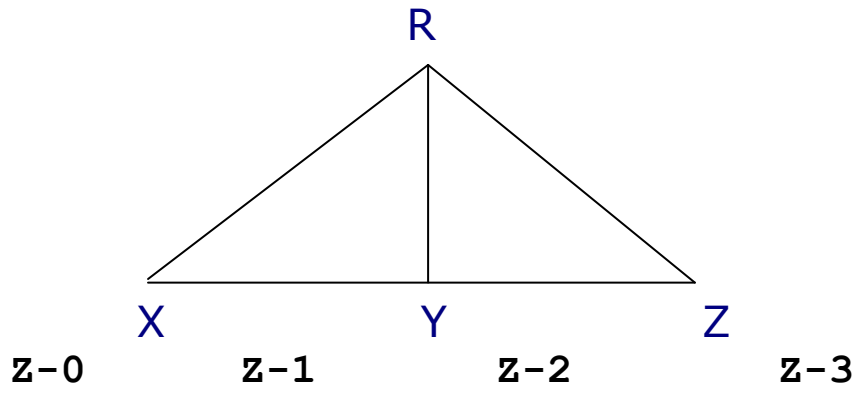
0	0	starting state	**INPUT**
1	0 —*n*— 8	shift 8	**they** **n**
1	0 —*NP(n)*— 2	reduce 5, goto 2	**visit** **vt**
2	0 —*NP(n)*— 2 —*vt*— 5	shift 5	**friends** **n**
3	0 —*NP(n)*— 2 —*vt*— 5 —*n*— 8	shift 8	**in** **prep**
3	0 —*NP(n)*— 2 —*vt*— 5 —*NP(n)*— 6	reduce 5, goto 6	**Egypt** **n**
4	0 —*NP(n)*— 2 —*vt*— 5 —*NP(n)*— 6 —*prep*— 13	shift 13	
5	0 —*NP(n)*— 2 —*vt*— 5 —*NP(n)*— 6 —*prep*— 13 —n— 8	shift 8	
5	0 —*NP(n)*— 2 —*vt*— 5 —*NP(n)*— 6 —*prep*— 13 —*NP(n)*— 14	reduce 5, goto 14	
5	0 —*NP(n)*— 2 —*vt*— 5 —*NP(n)*— 6 —*PP(prep NP(n))*— 7	reduce 8, goto 7	
5	0 —*NP(n)*— 2 —*VP(vt NP(n) PP(prep NP(n)))*— 3	reduce 4, goto 3	
5	0 —*S(NP(n) VP(vt NP(n) PP(prep NP(n))))*— 1	reduce 1, goto 1	
		accept	

Working Space (Netzwerk der Ergebnisse):

Konstruktion der Tabellen für den tabellengesteuerten Shift-Reduce Parser

Der erste Trick von PT-6 ist es, die Grammatik in Zustände der Abarbeitung der Eingabe zu zerlegen und statt der Regeln eine Tabelle für die Übergänge von einem Zustand zum anderen zu speichern. Bei der Abarbeitung einer Regel gibt es n+1 Zustände, wobei n die Zahl der Symbole auf der rechten Seite ist.

```
R -> X + Y + Z
```



Die Zustände kann man in der Regel durch einen Punkt markieren. Vgl. die *divided productions* in PT-4

```
R -> .X  Y  Z
R ->  X.Y  Z
R ->  X  Y.Z
R ->  X  Y  Z .
```

Beispiel:

GRAMMATIK (rekursiv und mehrdeutig)

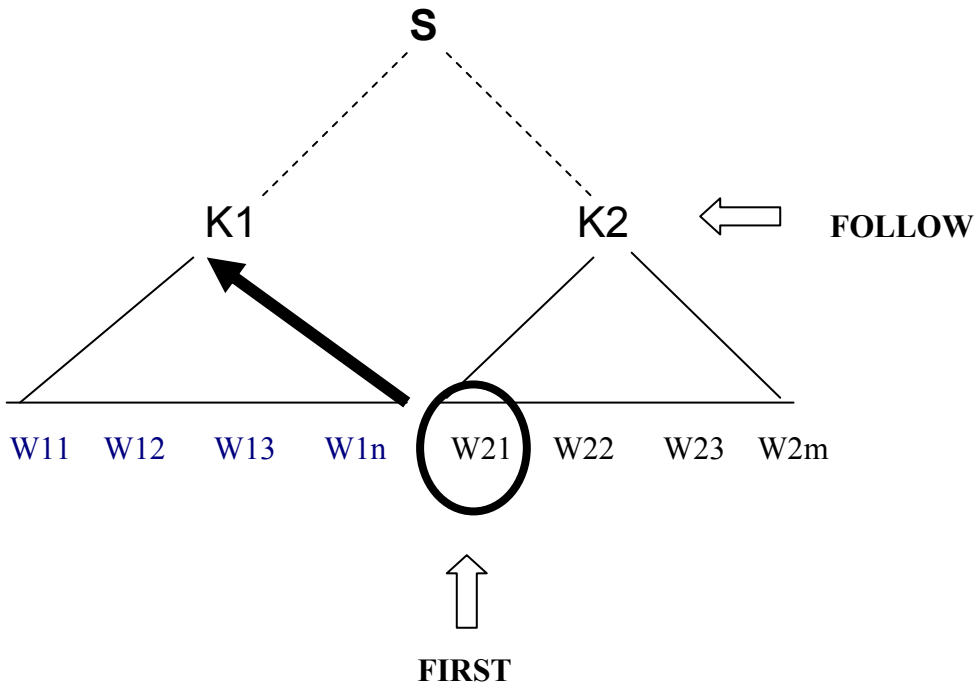
(re1)	S -> NP VP	(re6)	NP -> det n
(re2)	VP -> vi	(re7)	NP -> det adj n
(re3)	VP -> vt NP	(re8)	NP -> NP PP
(re4)	VP -> VP PP	(re9)	PP -> prep NP
(re5)	NP -> n		

Zustände:

	Zustände zu den Regeln:	**Expansionen dazu:**	**Reduktionsregel:**
(Z-0)	S' -> .S	S -> .NP VP NP -> .n NP -> .det n NP -> .det adj n NP -> .NP PP	
(Z-1)	S' -> S.		acc
(Z-2)	S -> NP .VP	VP -> .vi VP -> .vt NP VP -> .VP PP	
(Z-3)	S -> NP VP.		re 1

(Z-4)	VP -> vi.		re 2
(Z-5)	VP -> vt. NP	NP -> .n NP -> .det n NP -> .det adj n NP -> .NP PP	
(Z-6)	VP -> vt NP.		re 3
(Z-7)	VP -> VP .PP	PP -> .prep NP	
(Z-8)	VP -> VP PP.		re 4
(Z-9)	NP -> n.		re 5
(Z-10)	NP -> det. n NP -> det. adj n		
(Z-11)	NP -> det n.		re 6
(Z-12)	NP -> det adj. n		
(Z-13)	NP -> det adj n.		re 7
(Z-14)	NP -> NP. PP	PP -> .prep NP	
(Z-15)	NP -> NP PP.		re 8
(Z-16)	PP -> prep. NP	NP -> .n NP -> .det n NP -> .det adj n NP -> .NP PP	
(Z-17)	PP -> prep NP.		re 9

Der zweite Trick von PT-6 ist es, erst nach den Regeln zu reduzieren, wenn das erste Element (FIRST) der folgenden Konstituente (FOLLOW) aufgetreten ist (das ist die sog. *Vorausschau*)



GRAMMATIK (rekursiv und mehrdeutig)

(re1)	S -> NP VP	(re6)	NP -> det n
(re2)	VP -> vi	(re7)	NP -> det adj n
(re3)	VP -> vt NP	(re8)	NP -> NP PP
(re4)	VP -> VP PP	(re9)	PP -> prep NP
(re5)	NP -> n		

Funktion FOLLOW

FOLLOW(S) = $	**$**
FOLLOW(VP) = FOLLOW(S) FOLLOW(VP) = PP	**$, PP**
FOLLOW(NP) = VP FOLLOW(NP) = FOLLOW(VP) FOLLOW(NP) = PP FOLLOW(NP) = FOLLOW(PP)	**VP, $, PP**
FOLLOW(PP) = FOLLOW(VP) FOLLOW(PP) = FOLLOW(NP)	**$, PP, VP**

Funktion FIRST

FIRST($) = **$**

FIRST(VP) = **vi, vt**

FIRST(NP) = **n, det**

FIRST(PP) = **prep**

Algorithmus für die Erzeugung der Tabellen

1. Zerlege jede Regel in der Grammatik in die möglichen Zustände der Abarbeitung.
   Dabei zählen zum Zustand des Beginns einer Regel auch der Anfangszustände der Regeln, die rekursiv das Anfangssymbol ersetzen.

2. Numeriere die Zustände von 0 bis n.

3. Mache eine Matrix mit Zeilen z=Zustände und Spalten t= terminale Symbole der Grammatik, d.i. die AKTIONASTABELLE.

4. Mache eine Matrix mit Zeilen z=Zustände und Spalten NT=nicht terminale Symbole der Grammatik, d.i. die SPRUNGTABELLE

5. Loop durch die Zustände z und in jedem Zustand durch die Regeln.

5a. Falls **.t** in einer Regel in Zustand z (=Punkt vor terminalem Symbol t),
   schreibe in die **Aktionstabelle**, Zeile z, Spalte t : **"shift g",**
   wobei **g** der Zustand von **t.** (= Punkt nach terminalem Symbol t in derselben Regel).

5b Falls **.NT** in einer Regel in Zustand z (= Punkt vor nicht-terminalem Symbol NT),
schreibe in die **Sprungtabelle**, Zeile z, Spalte NT : **"goto g",**
wobei **g** der Zustand von **NT.** (= Punkt nach nicht-terminalem Symbol NT in derselben Regel).

5c Falls **X -> Y Z .** (= Ende einer Regel,)
schreibe **"reduce r"** (wobei **r** die Nummer der Regel ist) in die **Aktionstabelle**, Zeile z, Spalten FIRST von FOLLOW(Z) wobei Spalte **$** für das Satzende steht.

Ist die Regel **S.** so schreibe **"accept".**

Aus obiger Zerlegung ergeben sich so folgende nicht-deterministische Tabellen:

ACTION TABLE

Z	det	adj	n	prep	vi	vt	$
0	sh10		sh9				
1							acc
2					sh4	sh5	
3							re1
4				re2			re2
5	sh10		sh9				
6				re3			re3
7				sh16			
8				re4			re4
9				re5	re5	re5	re5
10		sh12	sh11				
11				re6	re6	re6	re6
12			sh13				
13				re7	re7	re7	re7
14				sh16			
15				re8	re8	re8	re8
16	sh10		sh9				
17				re9	re9	re9	re9

GO TO TABLE

NP	PP	VP	S
2,14			1
		3,7	
6,14			
	8		
	15		
14,17			

GRAMMAR G2

(re1) S -> NP VP
(re2) VP -> vi
(re3) VP -> vt NP
(re4) **VP -> VP PP**
(re5) NP -> n
(re6) NP -> det n
(re7) NP -> det adj n
(re8) **NP -> NP PP**
(re9) PP -> prep NP

G2 ambiguous!

Beispiel der Benutzung:

Input:	*my*	*friends*	*in*	*Egypt*	*sleep*
Lexicon:	**det**	**n**	**prep**	**n**	**vi**
Position:	1	2	3	4	5

P:	**States and Descriptions:**	**Explanation:**
0	**0**	starting state
1	**0 --det—10**	shift 10
2	**0 --det-- 10 --n—11**	shift 11
2	**0 ----NP(det)(n)----| 2 14**	reduce R-6, goto 2 & 14
3	**0 ----NP(det)(n)----| 2 14 --prep—16**	2 fails shift 16
4	**0 ----NP(det)(n)-- 14 --prep-- 16 --n--- 0**	shift 9

```
                                                  14
4   0 ----NP(det)(n)-- 14 --prep-- 16-NP(n)---|                  reduce R-5,
                                               17                goto 14 & 17

4   0 ----NP(det)(n)-- 14 --PP (prep)NP(n)---15                  14 fails,
                                                                 reduce R-9

                                                   2
4   0 ----NP ( NP(det)(n))(PP (prep)NP(n))----|                  reduce R-8
                                                  14             goto 2 & 14

5   0 ----NP ( NP(det)(n))(PP (prep)NP(n))-- 2 --vi---  4        shift 4,
                                                                 14 fails

                                                          3
5   0 ----NP ( NP(det)(n))(PP (prep)NP(n))-- 2 -VP(vi)--|        reduce R-2,
                                                          7      goto 3 & 6

5   0 ----S (NP ( NP(det)(n))(PP (prep)NP(n)))(VP(vi))-- 1       reduce R-1,
                                                                     accept
```

Evaluation

- **Efficiency**

  - Sehr effizient, läuft deterministisch ab, wo andere Parser Backtracking machen
  - Gründe für die Effizienz
    - ✓ Verbindung top-down (beim Herstellen der Tabellen) und bottom-up (at run-time)
    - ✓ Vorausschau - d.h. tatsächlich Abwarten - bis erstes Element der nächsten Konstituente gesehen ist, vermeidet shift/reduce Irrtümer
    - ✓ links-assoziativ anders als die anderen bottom-up Parser, wenig overgeneration

  - Erstellung der Tabellen kostet evtl. viel Rechenzeit.

- **Coverage**

Unrestricted context-free rules.

- **Drawing up lingware**

Nachteil, dass nach jeder Änderung der Grammatik die Tabellen neue errechnet werden müssen

Checklist PT-1, PT-2, PT-3, PT-4, PT-5, PT-6

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- procedural parser
- compiled parser PT-6

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- dependency descriptions **PT-5**

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- transition networks
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4** PT-6
- category reduction (bottom-up) **PT-5** PT-6
- state transition PT-6
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4** PT-6
- non-continuously (island parsing)

(6) Handling of alternatives

- backtracking **PT-1**
- parallel processing PT-2
- looking ahead PT-3 PT-6
- welformed substring table **PT-4** **PT-5**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3 PT-6
- all intermediate results stored (chart) **PT-4** **PT-5**

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

Parsing mit Mustern und Übergangsnetzwerken

- Grundgedanke: gleichzeitigen Fortschreitens innerhalb zweier Symbolsequenzen, dem Muster und der Eingabe.

- bestimmte Bedingungen, unter denen ein Muster zu einer Eingabe passt

Bei der unmittelbar zeichenkettenorientierten Mustererkennung stehen die Symbole in den Mustern in direkter Analogie zu den Symbolen in der Eingabe.

- graphisch als Netze repräsentiert

    - **Knoten** = Zustände des Erkenners bei der Abarbeitung des Musters
    - **Kanten** = etikettiert mit den Symbolen des Musters, die angeben, unter welcher Voraussetzung von einem Zustand zum nächsten übergegangen werden darf.

Das Erkennen eines Ausdrucks der Objektsprache erscheint als ein bestimmter Pfad durch das Netz.

Typische Ausdrücke in Mustern	Bedeutung
ε	leere Zeichenkette
a	ein Vorkommen des Elements "a"
\K	das Vorkommen eines beliebigen Elements aus der Menge \K
rs	Konkatenation der von r und s bezeichneten Zeichenketten
r|s	das alternative Vorkommen der von r und s bezeichneten Zeichenketten
r+	Konkatenation aus einem oder mehreren Vorkommen der von r bezeichneten Zeichenkette
r*	Konkatenation der Zeichenketten aus null oder beliebig vielen Vorkommen der von r bezeichneten Zeichenkette
r?	ein optionales Vorkommen der von r bezeichneten Zeichenkette
(r) und {r}	Klammerpaare gruppieren Ausdrücke hinsichtlich Konkatenation und Alternation

Reguläre Ausdrücke

- Es gibt sogenannte reguläre Sprachen (Chomsky-Hierarchie, Typ 3).

- Zur Formulierung von Mustern für eine reguläre Sprache benutzt man reguläre Ausdrücke.

- Reguläre Ausdrücke lassen sich von einem endlichen erkennenden Automaten den Ausdrücken der Objektsprache zuordnen.

- Dazu werden sie zunächst in ein nicht-deterministisches finites Übergangsnetzwerk ('finite state transition network', FTN) umgeformt.

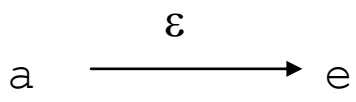
Definition eines Formalismus M für eine reguläre Sprache L und Übersetzung in FTN

Zuerst elementaren Ausdrücken nach (i) bis (iii) bearbeiten, dann Teilausdrücke nach (iv) bis (ix) kombinieren.

```
a = Anfangszustand
e = Endzustand
zi = Zwischenzustand
```

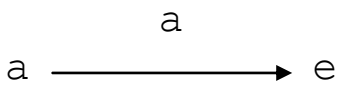
(i) 'ε' ist ein Ausdruck von M und bezeichnet ein Vorkommen der leeren Zeichenkette.

Zum Ausdruck 'ε' konstruiere das Netzwerk



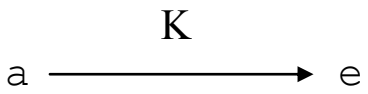
(ii) Ist 'a' ein Element des Vokabulars von L, so ist 'a' ein Ausdruck von M und bezeichnet ein Vorkommen des Elements 'a' aus L.

Zum Ausdruck 'a' konstruiere das Netzwerk



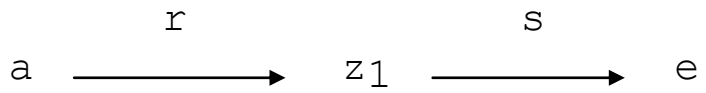
(iii) Ist 'K' eine Teilmenge des Vokabulars von L, so ist 'K' ein Ausdruck von M und bezeichnet das Vorkommen eines beliebigen Elements der Kategorie 'K' in L.

Zum Ausdruck **'K'** konstruiere das Netzwerk



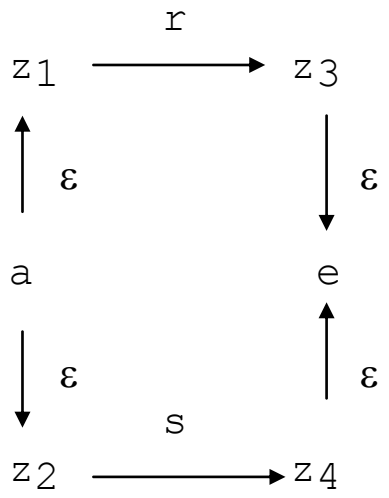
(iv) Sind 'r' und 's' Ausdrücke von M, so ist 'rs' ein Ausdruck von M und bezeichnet die Konkatenation der von 'r' und 's' bezeichneten Zeichenketten in L.

Zum Ausdruck 'rs' konstruiere das Netzwerk



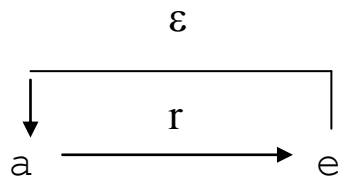
(v) Sind 'r' und 's' Ausdrücke von M, so ist 'r|s' ein Ausdruck von M und bezeichnet das alternative Vorkommen der von 'r' und 's' bezeichneten Zeichenketten in L.

Zum Ausdruck 'r|s' konstruiere das Netzwerk



(vi) Ist 'r' ein Ausdruck von M, so ist $'r^{+}'$ ein Ausdruck von M und bezeichnet die Kon-katenation aus einem oder mehreren Vorkommen der von 'r' bezeichneten Zeichenkette in L.

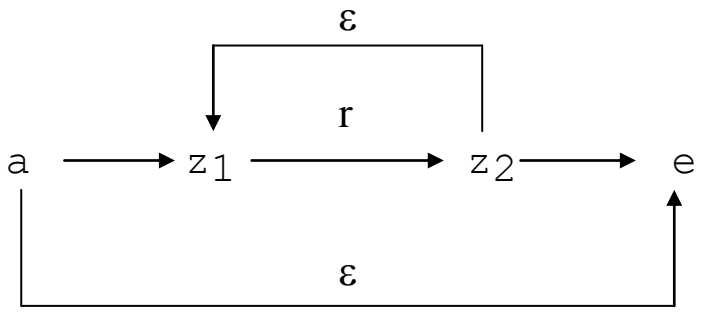
Zum Ausdruck $'r^{+}'$ konstruiere das Netzwerk



(vii) Ist 'r' ein Ausdruck von M, so ist $'r^{*}'$ ein Ausdruck von M und bezeichnet die Konkatenation der Zeichenketten aus null oder beliebig vielen Vorkommen der von 'r' bezeichneten Zeichenkette in L,

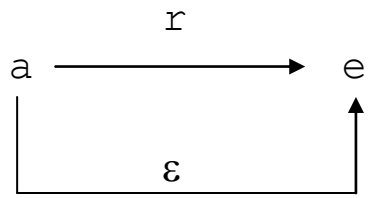
d.h. $r^{*} = r^{+} \mid \varepsilon$.

Zum Ausdruck $'r^{*}'$ konstruiere das Netzwerk



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

Zum Ausdruck 'r?' konstruiere das Netzwerk



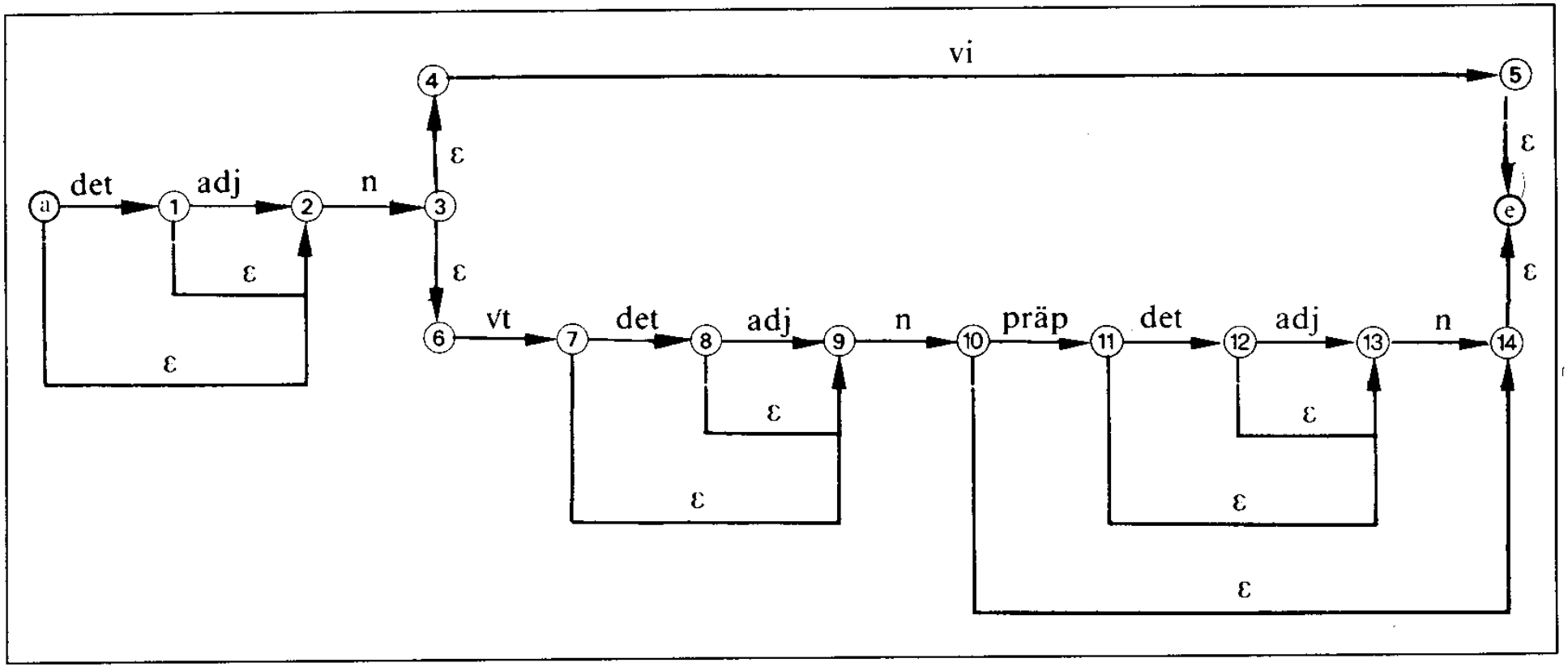
(ix) Die Klammerpaare '(', ')' und '{', '}' werden dazu benutzt, Ausdrücke von M hinsichtlich der Konkatenation und Alternation zu gruppieren. Ist ' r' ein Ausdruck von M, so sind auch '(r)' und '{r}' Ausdrücke von M.

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

Regulärer Ausdruck (äquivalent G1)

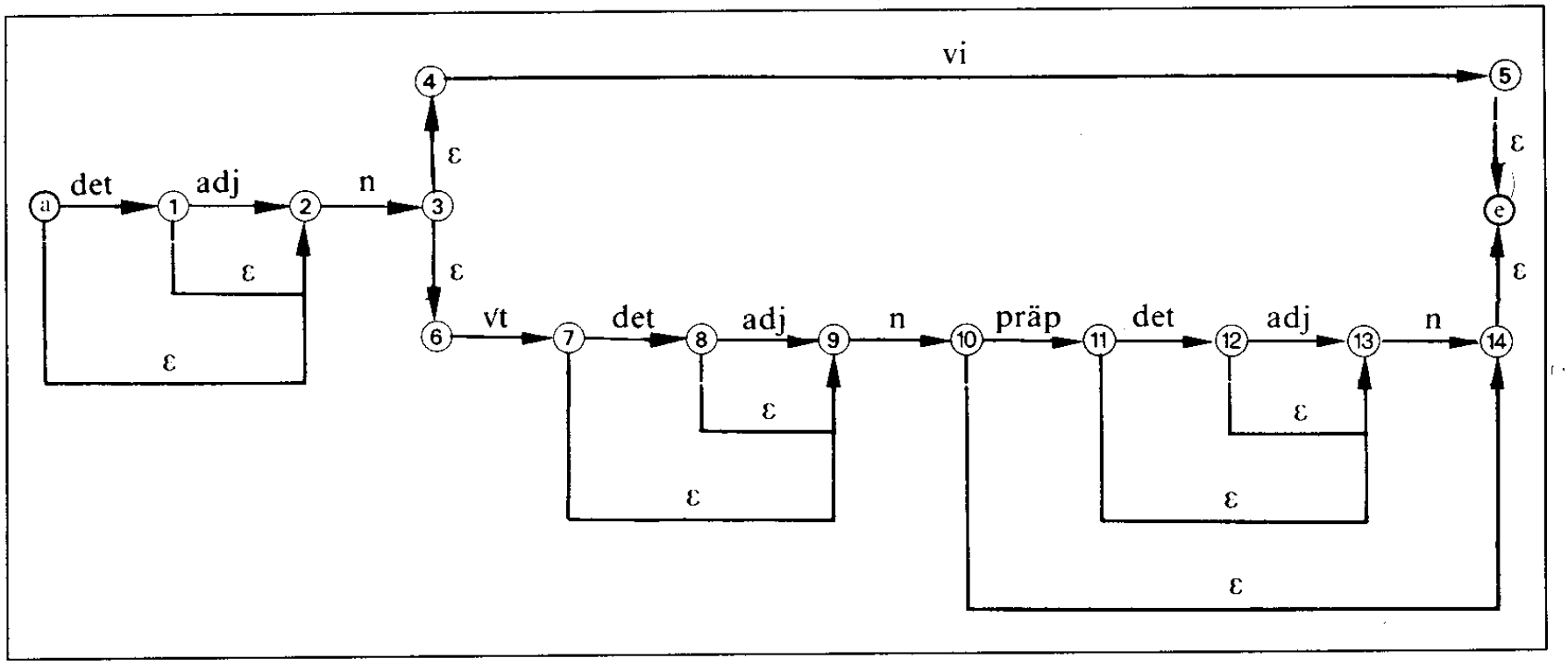
(det adj?)? n {vi | vt (det adj?)? n (präp (det adj?) n)? }

nicht-deterministisches FTN dazu



Konstruktion einer Übergangstabelle für einen deterministischen endlichen Erkenner

- Ausgangspunkt ist ein nicht-deterministisches Übergangsnetzwerk.
- Beginnend mit dem Anfangszustand werden zu jedem Knoten alle **Mengen** von Zuständen gebildet, die vom gegebenen Zustand aus über ε-Kanten und/oder eine Kante mit einem bestimmten elementaren Symbol erreicht werden können.
- Diese Zustandmengen bilden nun die Zustände in der Übergangtabelle für den deterministischen Erkenner.
- Als Ausgangszustände werden sie den Zeilen der Tabelle zugeordnet; als Zielzustände werden sie in die Spalten eingetragen, die den Elementen oder Kategorien des Vokabulars von L zugeordnet sind.
- Mithilfe der Tabelle lassen sich also deterministisch die Mengen aller möglichen Zustände bestimmen, in denen sich ein nicht-deterministischer Erkenner befinden kann, nachdem das nächste Eingabesymbol gelesen worden ist.
- Diejenigen Zustandsmengen, die den Endzustand des nicht-deterministischen Übergangs-netzwerkes einschließen, werden in der Tabelle als mögliche Endzustände des deterministischen Erkenners ausgezeichnet.



ÜBERGANGSTABELLE FÜR EINEN DETERMINISTISCHEN ERKENNER:

a/2	det	1/2
a/2	n	3/4/6
1/2	adj	2
1/2	n	3/4/6
2	n	3/4/6
3/4/6	vi	5/e
3/4/6	vt	7/9
7/9	det	8/9
7/9	n	10/14/e

8/9	adj	9
8/9	n	10/14/e
9	n	10/14/e
10/14/e	präp	11/13
11/13	det	12/13
11/13	n	14/e
12/13	adj	13
13	n	14/e

Graphische Darstellung der Tabelle

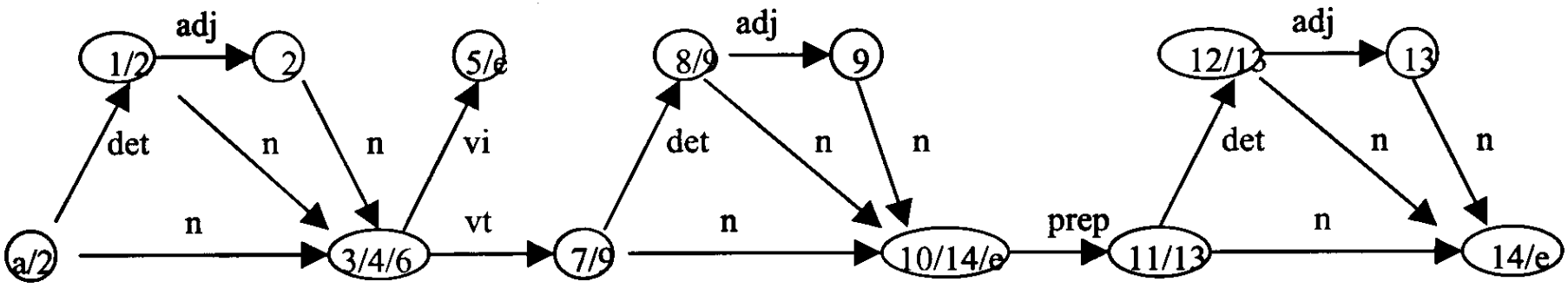


Tabelle als Matrix von Ausgangszuständen und terminalen Elementen, mit Zielzuständen in den Feldern:

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

Parser-Beispiel

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

ENDLICHER ERKENNER:

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

Definition eines Endlichen Automaten

Endlicher Erkenner, Finite State Recognizer A = {Z, E, f, S, E}

1. eine Menge von Zuständen **Z**
2. eine Menge von Eingabesymbolen **E** (einschließlich der leeren Zeichenkette)
3. eine Übergangsfunktion **f: Z x E $\rightarrow$ Z**
4. ein Startzustand **S**
5. eine Menge von Endzuständen **E $\subset$ Z**

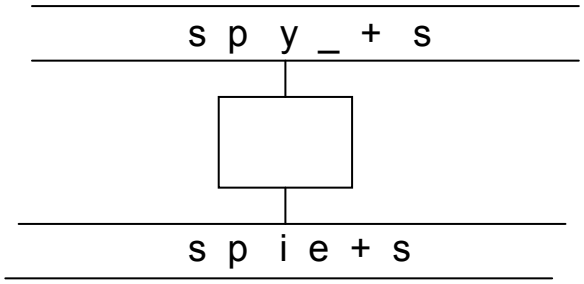
Finite State Transducer A = {Z, E, A, f, S, E}

wie Endlicher Erkenner aber mit Output, also

3. eine Übergangsfunktion **f: Z x E x A $\rightarrow$ Z**

Bei Vorliegen bestimmter Eingabesymbole werden gemäß der Übergangsfunktion bestimmte Ausgabesymbole erzeugt (oder m.a.W. der Übergang von einem Zustand in den anderen hängt von Paaren aus E x A ab)

Man kann sich einen Transducer als Automat mit zwei Bändern vorstellen



Beispiel Morphologie (aus Hellwig, Dependency Unification Grammar)

Morphologische Veränderungen werden (bei der Definition morphologischer Klassen) mittels des Attributes „change“ beschrieben:

- CDATA von "change" enthält einen Bedingungsteil und einen Ersetzungsteil, durch den Seperator "/" getrennt.
- Bedingungsteil wie Ersetzungsteil können Einzelzeichen ebenso wie Mengensymbole enthalten.
- Mengensymbole wie überhaupt alle Metasymbole beginnen mit dem Zeichen "\"
- Sollen Zeichen oder Zeichenketten aus dem Bedingungsteil im Ersetzungsteil wieder auftauchen, werden sie im Bedingungsteil geklammert. Im Ersetzungsteil wird auf die betreffenden Zeichen referiert mit einer Variablen der Form "$"n, wobei n eine Ziffer ist, welche angibt, die wievielte Klammer aus dem Bedingungsteil die gewünschte Zeichenkette enthält.
- Im Bedingungsteil kann "." als generische Variable vorkommen. Sie steht für ein beliebiges Zeichen. Zeichenketten werden in der üblichen Form beschrieben: "(.*)" , "(.+)"
- Im Ersetzungsteil stehen die Metazeichen "/up" für Großschreibung und "/low" für Kleinschreibung.

Die Zeichenmengen werden mittels Attribut "charset" definiert.

Reduplikation in Agta

takki	leg	*taktakki*	legs
labáng	patch	*lablabáng*	patches
uffu	thigh	*ufuffu*	*t*highs

```
charset="C=[tklbngf] V=[aiu]"
```

1. Klasse (Konsonant am Anfang)

```
change= "(\C)(\V)(\C)(\C|\V)+/$1$2$3$1$2$3$4"
```

oder

```
change= "(\C)(\V)(\C)(.+)/$1$2$3$1$2$3$4"
change= "(\C\V\C)(.+)/$1$1$2"
change= "([tklbngf][aiu][tklbngf])(.+)/$1$1$4"
```

2. Klasse (Vokal am Anfang)

```
change= "(\V\C)(.+)/$1$1$2"
```

Reduplikation in Chamorro

dankolo	big	*dankololo*	very big
bunita	pretty	*bunitata*	very pretty
metgot	strong	*metgogot*	very strong1.

1. Vokal am Ende

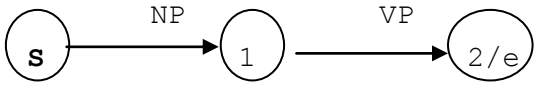
```
change= "(.+)(\C\V)/$1$2$2"
```

2. Konsonant am Ende

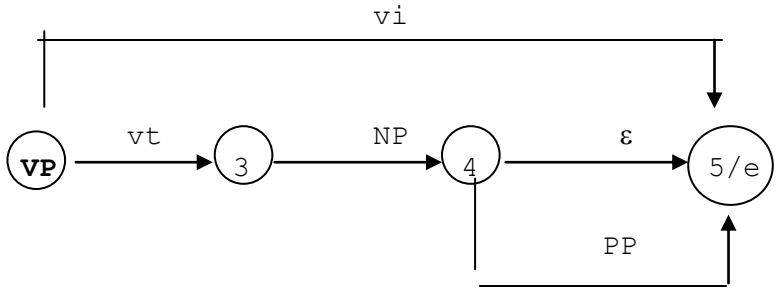
```
change= "(.+)(\C)(\V)[\C)/$1$2$3$2$3$4"
```

Parsing mit Rekursiven Übergangsnetzwerken (RTN)

Umformung von Phrasenstrukturgrammatiken in RTNs

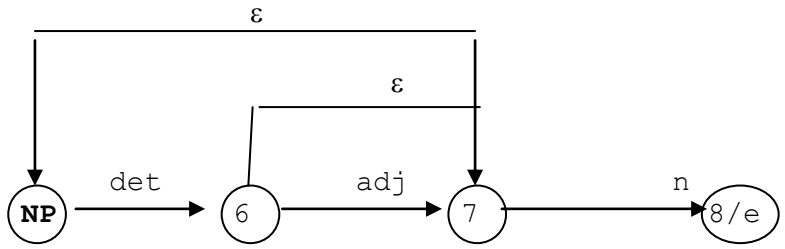


Für jedes nicht-terminale Symbol wird ein eigenes Netz erzeugt.

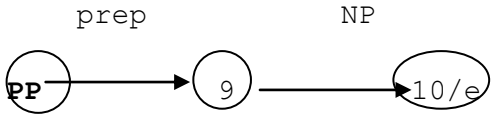


Jedes Netz entspricht einem Unterprogramm.

Jedes nicht -terminale Symbol an einer Kante bedeutet ein Unterprogramm-Aufruf



Jedes terminale Symbol an einer Kante muss direkt in der Eingabe vorkommen



Prozedurale Netzwerke gleichen Flussdiagrammen.

Input:	*many*	*foreign*	*tourists*	*enjoy*	*the*	*pyramids*
Lexicon:	**det**	**adj**	**n**	**vt**	**det**	**n**
Position:	1	2	3	4	5	6

WORKING AREA

W	P	Current state	Cate-gory	Target state	Pop-up Stack
(1)	1	S	**NP**	1	-
(2)	1	NP	**ε**	7	1
backtracking					
(2)	1	NP	**det**	6	1
(3)	2	6	**ε**	7	1
backtracking					
(3)	2	6	**adj**	7	1
(4)	3	7	**n**	8/e	1
(5)	3	1	**VP**	2/e	-
(6)	3	VP	**vi**	5/e	2/e
backtracking					
(6)	4	VP	**vt**	3	2/e
(7)	4	3	**NP**	4	2/e
(8)	4	NP	**ε**	7	2/e,4
backtracking					
(8)	5	NP	**det**	6	2/e,4
(9)	5	6	**ε**	7	2/e,4
(10)	6	7	**n**	8/e	2/e,4
(11)	6	4	**ε**	5/e	2/e
(16)	6	2/e	**-**	-	-

BACKTRACKING STORE

	Back to W	Alternative transition
(0)		
(1)	2	det - 6
(0)		
(1)	3	adj - 7
(0)		
(1)	6	vt - 3
(0)		
(1)	8	det - 6
(0)		
(1)	9	adj - 7
(2)	11	PP - 5/e

One parse found. The parser will continue backtracking.

Evaluation

Efficiency

Endliche Automaten sind äußerst effizient, wenn die Beschreibung des Sprachphänomens mit regulären Ausdrücken möglich ist.

RTN sind äquivalent zu kontextfreien Grmmatiken, im Grunde wie PT-1.

Coverage

Ziemlich begrenzt. In der Syntax kommt man mit regulären Ausdrücken nicht aus und auch Transducer erzeugen normalerweise keine Strukturbeschreibungen

Checklist PT-1, PT-2, PT-3, PT-4, PT-5, PT-6, PT-7

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- procedural parser
- compiled parser PT-6 **PT-7**

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7**
- dependency descriptions **PT-5**

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- transition networks **PT-7**
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4** PT-6
- category reduction (bottom-up) **PT-5** PT-6
- state transition PT-6 **PT-7**
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7**
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7**
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4** PT-6
- non-continuously (island parsing)

(6) Handling of alternatives

- backtracking **PT-1** **(PT-7)**
- parallel processing PT-2 **(PT-7)**
- looking ahead PT-3 PT-6
- wellformed substring table **PT-4** **PT-5**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3 PT-6 **PT-7**
- all intermediate results stored (chart) **PT-4** **PT-5**

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

Zugrunde liegt ein Rekursives Übergangsnetzwerk (RTN) - vgl. vorige Stunde.
Dies entspricht einer kontextfreien Grammatik.

Erweiterung des RTN zum ATN

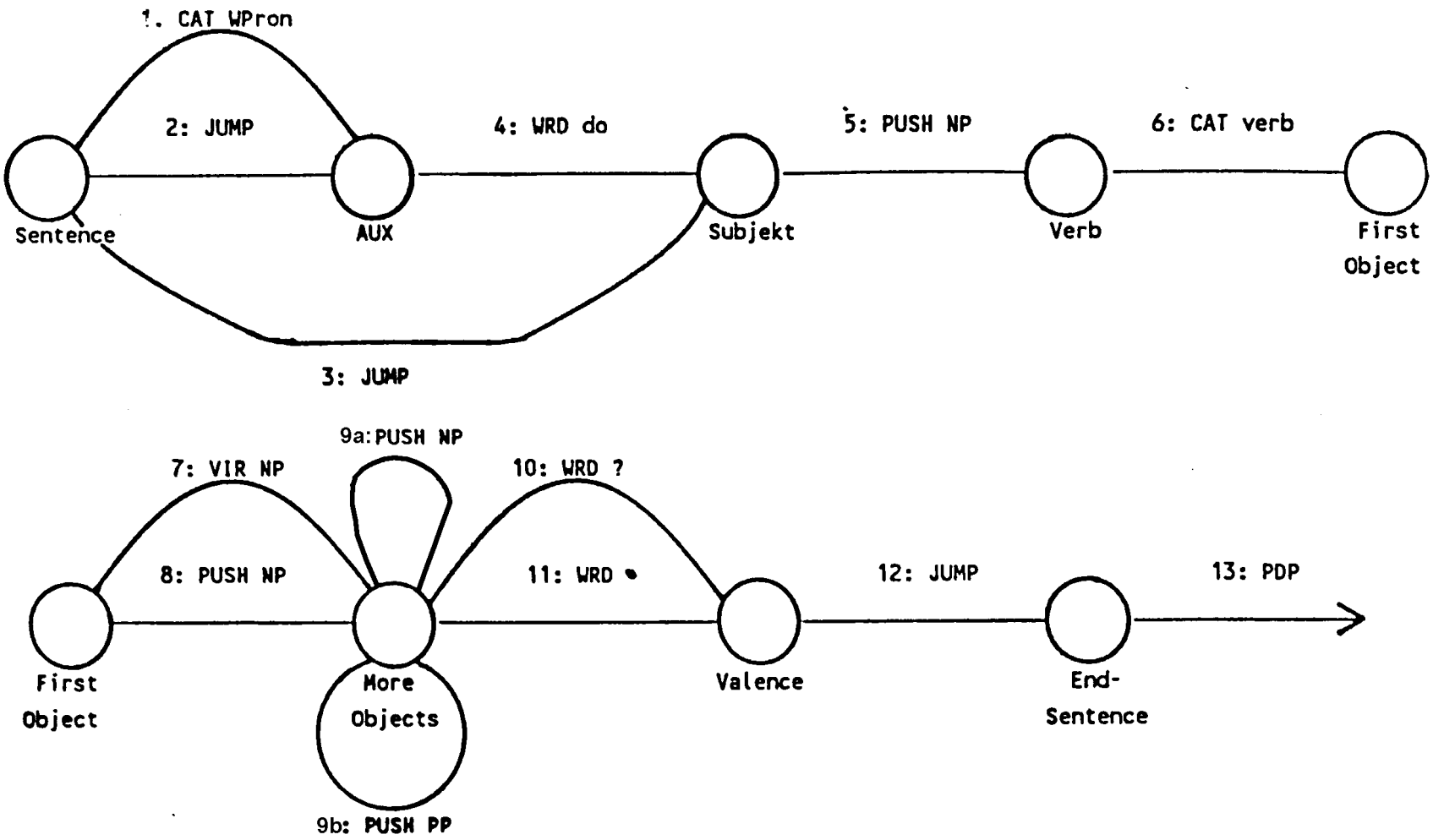
- Tests vor Traversierung einer Kante
- Aktion nach Traversierung einer Kante
- beliebig viele Register können in Tests konsultiert und durch Aktionen verändert werden
- auch globale Variablen (sog. HOLD Register) sind möglich

Dies entspricht einer Turing Maschine, d.h. wir haben maximale generative Kapazität.

➢ Zu viel Kapazität ist nicht wünschenswert! Genauso wie zu wenig.

Durch Festlegung auf eine beschränkte Menge von Kantentypen, Tests und Aktionen wird die generative Kapazität reduziert.

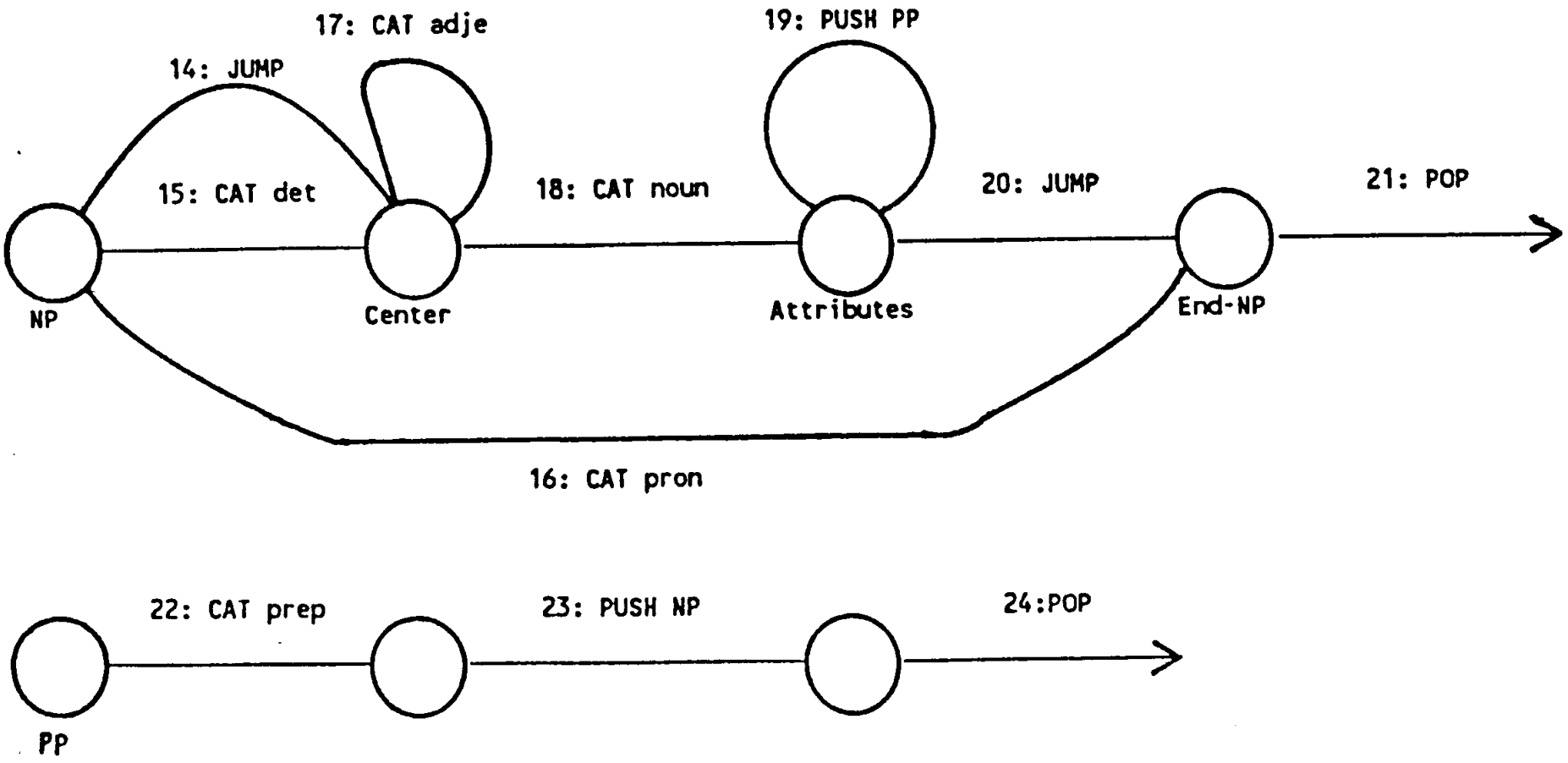
ATN-Grammatik in Form eines Netzwerkes



Gudrun sleeps. Does Gudrun sleep? Gudrun loves her cat.

Gudrun gives the book to me. Did Gudrun give you the book? Gudrun talked to me about the wedding.

Did Gudrun return the book to the library of congress? What did Gudrun give to you? etc.


17: CAT adje
19: PUSH PP
14: JUMP
15: CAT det
18: CAT noun
20: JUMP
21: POP
NP
Center
Attributes
End-NP
16: CAT pron
22: CAT prep
23: PUSH NP
24:POP
PP

ATN Formalismus (Definition im BNF Format)

<augmented transition network> := <set of arcs>*

<set of arcs> := <*node*> <arc>*

<arc> := <WRD-arc> | <CAT-arc> | <JUMP-arc> | <VIR-arc> | <PUSH-arc> | <POP-arc>

<WRD-arc> := WRD <*lexical element*> <test>* <action>* <transition>

<CAT-arc> := CAT <*lexical category*> <test>* <action>* <transition>

<JUMP-arc> := JUMP <transition> <test>* <action>*

<VIR-arc> := VIR <*constituent*> <test>* <action>* <transition>

<PUSH-arc> := PUSH <*node*> <test>* <pre-action>* <test>*<action>* <transition>

<POP-arc> := POP <form> <test>*

<test> := T | <*condition*>

<pre-action> := <send register>

<action> := <*condition*> <set register> | <*condition*> <lift register> | <HOLD-action>

<set register> := SETR <*register*> <form> | ADDR <*register*> <form>

<send register> := SENDR <*register*> <form>

<lift register> := LIFTR <*register*> <form>

<HOLD-action> := HOLD <*constituent*> <form>

<form> := <current lexical item> | <current category> | <content of register> | <build construction>

<current lexical item> := *

<current category> := GETF <*attribute*> | "<*value*>"

<content of register> := GETR <*register*>

<build construction> := BUILDQ <*frame*> <*register*>*

<transition> := TO <*node*>

Der ATN-Formalismus ist im Prinzip eine Programmiersprache (oft wird LISP benutzt)

Register

Für jedes Unternetzwerk, d.h. für jede Rekursionstiefe, gibt es eine eigene Registermenge (so wie lokale Variablen in einem Unterprogramm). Sollen Inhalte von Registern auf einer anderen Stufe verfügbar sein, müssen sie explizit übergeben werden (Pre-Action SENDR und Post-Action LIFTR).

Normal werden alle Register in einer einzigen Register-list gespeichert. Diese Liste besteht aus Paaren [Registername, Form]. Einfügen mit SETR, Entnehmen mit GETR.

Beispiel

Befehle:

```
SETR number[singular]
SETR person[3]
SETR verb_form[infinitive]
SETR illocution[question]
```

Register-List

number	singular
person	3
verb_form	infinitive
illocution	question

Algorithmus

(A) Navigation durch das Netz

- Bei alternativen Kanten aktuelle Konfiguaration im Backtracking Stack speichern. Beim Auflaufen bei irgendeiner Kante entsprechend zurücksetzen.

- Bei jeder Kante vorab die angegebenen Tests machen.

- Bei jeder Kante am Schluß die angegebenen Aktionen ausführen.

Außerdem:

- WRD-Kante: Wort an der Kante = Wort in der Eingabe? Weiter zum TO-Knoten

- CAT-Kante: Kategorie an der Kante = Kategorie in der Eingabe? Weiter zum TO-Knoten

- JUMP-Kante: ohne Vorrücken in der Eingabe zum angegebenen TO-Knoten.

- VIR-Kante: Konstituente an der Kante = Konstituente in der HOLD-list?
  Dann ohne Vorrücken in der Eingabe zum angegebenen TO-Knoten.

- PUSH-Kante: ggf. Pre-action ausführen. Aktuelle Konfiguration auf den Pop-up Stack. Neue Konfiguration für Unternetz anfangen. Neue Registerliste für die entsprechende Rekursionstiefe anfangen. Weiter ab erstem Knoten im Unternetz.

- POP-Kante: zur Konfiguration, die als oberste im Pop-up Stack liegt, zurückkehren. Die entsprechende PUSH-Kante traversieren. Weiter zum TO-Knoten.

(B) Tests

- Der Wahrheitswert einer aussagenlogischen Verknüpfung wird berechnet. Darin geht es vor allem um Inhalte von Registern und Merkmale der Eingabe .

(C) Aktionen

- SETR: Das Paar [Registername, Form] in die aktuelle Register-list stellen. Ist schon ein Paar mit Registername vorhanden, wird es überschrieben.

- ADDR: Das Paar [Registername, Form] der aktuellen Register-list hinzufügen. Ist schon ein Paar mit Registername vorhanden, wird es nicht überschrieben.

- SENDR: Pre-action, nur bei PUSH-Kante. Das Paar [Registername, Form] an die Register-list des Unternetzwerkes übergeben.

- LIFTR: Aktion bei POP-Kanten.. Das Paar [Registername, Form] an die Register-list des übergeordneten Netzwerkes übergeben.

- HOLD: Die angegebene Konstituente in die HOLD-list eintragen. Die HOLD-list kann von allen Ebenen aus beim Traversieren einer VIR-Kante konsultiert werden (wie eine globale Variable)..

(D) Aktionen für Formen

- * steht für das Lexem an der aktuellen Eingabeposition.

- GETF gibt den Wert des angegebenen Attributs an der aktuellen Eingabeposition zurück, z.B. "GETF Kasus" steht für den Kasus des Eingabewortes, also entweder für "singular" oder "plural".

- GETR gibt die Form im spezifizierten Register zurück. "F" wenn Register leer.

- BUILDQ dient zum Aufbau der Strukturbeschreibung. Rahmen: Beliebige Zeichen und eine Anzahl +. Dazu eine Folge von Registernamen. Für jedes + wird der Inhalt des analogen Registers substituiert. * im Rahmen steht für das Lexem an aktueller Eingabeposition.

Eine konkrete Grammatik als ATN-Programm

(in Pseudocode; entspricht obigem Netzwerk)

```
node Sentence

    arc 1: CAT W_pron
        action HOLD NP *
        transition TO Aux

    arc 2: JUMP
        transition TO Aux

    arc 3: JUMP
        action SETR illocution "assertion"
        action SETR verb_form "finite"
        transition TO Subject

node Aux

    arc 4: WRD do
        action SETR number <GETF number>
        action SETR person <GETF person>
        action SETR verb_form "infinitive"
        action SETR illocution "question"
        transition TO Subject
```

```
node Subject

    arc 5: PUSH NP
        pre-action SENDR number
        pre-action SENDR person
        action ADDR complements "subject"
        action ADDR arguments <BUILDQ (subject: *)>
        transition TO Verb

node Verb

    arc 6: CAT verb
        test <GETR verb_form> equals <GETF form> or
            {<GETR number> includes <GETF number> and
             <GETR person> includes <GETF person>}
        action SETR predicate *
        transition TO First_Object

node First_Object

    arc 7: VIR NP
        action ADDR complements "dir_object"
        action ADDR arguments <BUILDQ (dir_object: +) NP>
        action SETR dirobj "no_prep"
        transition To More_objects
```

```
    arc 8: PUSH NP
        action ADDR complements "dir_object"
        action ADDR arguments <BUILDQ (dir_object: *)>
        action SETR dirobj "prep"
        transition TO More_objects

node More_Objects

    arc 9a: PUSH NP
        test <GETR dirobj> equals "no_prep"
        action ADDR complements "indir_object"
        action ADDR arguments <BUILDQ (indir_object: *)>
        transition TO More_objects

    arc 9b: PUSH PP
        test <GETR dirobj> equal "prep"
        test if <GETR preposition> equals "to" then
        action
            {ADDR complements "indir_object"
            ADDR arguments <BUILDQ (indir_object: *)>}
        test else
        action
            {ADDR complements <BUILDQ prep_object + preposition>
            ADDR arguments <BUILDQ (prep_object: + *) preposition>}
        transition TO More_Objects
```

```
    arc 10: WRD ?
        test <GETR illocution> equals "question"
        transition TO Valence

    arc 11: WRD .
        test <GETR illocution> equals "assertion"
        transition TO Valence

node Valence

    arc 12: JUMP
        test <GETF valence> includes <GETR complements>
        transition TO End_sentence

node End_sentence

    arc 13: POP
        BUILDQ (illocution: + (predicate: + + ))
        illocution predicate arguments


node NP
    arc 14: JUMP
        transition TO Center

    arc 15: CAT det
        action SETR determiner <BUILDQ (determiner: *)>
        transition TO Center
```

```
    arc 16: CAT pron
        test <GETR number> is empty or equals <GETF number>
        test <GETR person> is empty or equals <GETF person>
        action SETR number <GETF number>
        action SETR person <GETF person>
        action SETR structure <GETF *>
        transition TO End_NP

node Center
    arc 17: CAT adje
        action ADDR attributes <BUILDQ: (attrib: *)>
        transition TO Center

    arc 18: CAT noun
        test <GETR number> is empty or equals <GETF number>
        test <GETR person> is empty or equals "3rd"
        action SETR number <GETF number>
        action SETR person "3rd"
        action SETR head <GETF *>
        transition TO Attributes

node Attributes
    arc 19: PUSH PP
        action ADDR attributes <BUILDQ: (attrib: *)>
        transition TO Attributes
```

```
    arc 20: JUMP
        action SETR structure <BUILDQ + + + head determiner
            attributes>
        transition TO End_NP

node End_NP

    arc 21: POP structure
        action LIFTR number
        action LIFTR person


node PP

    arc 22: CAT prep
        action SETR preposition <GETF *>
        transition TO Prep_NP

node Prep_NP

    arc 23: PUSH NP
        action SETR structure <BUILDQ ( * )>
        transition TO End_PP

node End_PP

    arc 24: POP structure
        action LIFTR preposition
```

LEXICON

sleep	lex[sleep] cat[verb] form[infinitive] valence[subject]
sleep	lex[sleep] cat[verb] form[finite] person[1st, 2nd] number[singular] valence[subject]
sleeps	lex[sleep] cat[verb] form[finite] person[3rd] number[singular] valence[subject]
sleep	lex[sleep] cat[verb] form[finite] person[1st, 2nd, 3rd] number[plural] valence[subject]
feed	lex[feed] cat[verb] form[infinitive] valence[subject, direct_object, indirect_object]
feed	lex[feed] cat[verb] form[finite] person[1st, 2nd] number[singular] valence[subject, direct_object, indirect_object]
feeds	lex[feed] cat[verb] form[finite] person[3rd] number[singular] valence[subject, direct_object, indirect_object]
feed	lex[feed] cat[verb] form[finite] person[1st, 2nd, 3rd] number[plural] valence[subject, direct_object, indirect_object]
do	lex[do] cat[verb] form[infinitive]
do	lex[do] cat[verb] form[finite] person[1st, 2nd] number[singular]
does	lex[do] cat[verb] form[finite] person[3rd] number[singular]

do	lex[do] cat[verb] form[finite] person[1st, 2nd, 3rd] number[plural]
Gudrun	lex[Gudrun] cat[noun] person[3rd] number[singular]
cat	lex[cat] cat[noun] person[3rd] number[singular]
fish	lex[fish] cat[noun] person[3rd] number[singular, plural]
I	lex[I] cat[pron] person[1st] number[singular]
you	lex[you] cat[pron] person[2nd] number[singular, plural]
what	lex[what] cat[W_pron]
the	lex[the] cat[det]
her	lex[her] cat[det]
silly	lex[silly] cat[adje]
to	lex[to] cat[prep]

TRACE: ***What does Gudrun feed her cat***

P:	Word:	Configurations:	Alter-natives:
1	**what**	*node* **Sentence** *arc 1:* CAT W_pron *action* HOLD NP[what] *transition* TO Aux	arc 2
2	**does**	*node* **Aux** *arc 4:* WRD do *action* SETR number[singular] *action* SETR person[3rd] *action* SETR verb_form[infinitive] *action* SETR illocution[question] *transition* TO Subject	
3	**Gudrun**	*node* **Subject** *arc 5:* PUSH NP *pre-action* SENDR number[singular] *pre-action* SENDR person[3rd]	
		node **NP** *arc 14:* JUMP *transition* TO Center	arc 15
		node **Center** *arc 17:* CAT adje ***fails*** *arc 18:* CAT noun *test* number[singular] equals number[singular] *test* person[3rd] equals person[3rd] *action* SETR number[singular]	arc 18

		action SETR person[3rd] *action* SETR head[Gudrun] *transition* TO Attributes	
4	**feed**	*node* **Attributes** *arc 19:* PUSH PP	arc 20
		node **PP** *arc 22:* CAT prep ***fails***	
		arc 20: JUMP *action SETR* structure[Gudrun] *transition* TO End_NP	
		node **End_NP** *arc 21:* POP [Gudrun] *action* LIFTR number[singular] *action* LIFTR person[3rd]	
		action ADDR complements[subject] *action* ADDR arguments[subject: Gudrun)] *transition* TO Verb	
		node **Verb** *arc 6:* CAT verb *test* verb_form[infinitive] equals form[infinitive] *action* SETR predicate[feed] *transition* TO First_Object	

5	**her**	*node* **First_Object**	
		arc 7: VIR NP	arc 8
		action ADDR complements[subject, dir_object]	
		action ADDR arguments[(subject: Gudrun)(dir_object: what)]	
		action SETR dirobj[no_prep]	
		transition TO More_objects	
		node **More_Objects**	
		arc 9a: PUSH NP	arc 9b
		test dirobj[no_prep] equals "no:prep"	
		node **NP**	
		arc 14: JUMP	arc 15
		transition TO Center	
		node **Center**	
		arc 17: CAT adje ***fails***	arc 18
		arc 18: CAT noun ***fails***	
		arc 15: CAT det	arc 16
		action SETR determiner [(determiner: her)]	
		transition TO Center	
6	**cat**	*node* **Center**	
		arc 17: CAT adje ***fails***	arc 18
		arc 18: CAT noun	
		test number is empty	
		test person is empty	
		action SETR number[singular]	
		action SETR person[3rd]	

```
      action SETR head[cat]
      transition TO Attributes

7  ?  node Attributes
      arc 19: PUSH PP                                          arc 20

      node PP
         arc 22 CAT prep fails

      arc 20: JUMP
      action SETR structure[cat (determiner:her)]
      transition TO End_NP

   node End_NP
      arc 21: POP [cat (determiner: her)]
      action LIFTR number[singular]
      action LIFTR person[3rd]

   action ADDR complements [subject,
          dir_object, indir_object]
   action ADDR arguments[(subject: Gudrun)(dir_object:
           what)
          (indir_object: cat
          (determiner: her))]
   transition TO More_Objects


node More_Objects
   arc 9b: PUSH PP                                             arc 10

   node PP
      arc 22: CAT prep fails
```

```
   arc 10: WRD ?                                                    arc 11
   test [question] equals "question"
   transition TO Valence

node Valence
   arc 12: JUMP
   test [subject, dir_object, indir_object] includes
   complements[subject, dir_object, indir_object]
   transition TO End_sentence

node End_sentence
   arc 13: POP [(illocution: question
        (predicate: feed
        (subject: Gudrun)
        (dir_object: what)
        (indir_object: cat
        (determiner: her))]
```

RESULT

```
(illocution: question
  (predicate: feed
     (subject: Gudrun)
     (dir_object: what)
     (indir_object: cat
          (determiner: her))]
```

Evaluation

Efficiency

- hängt vom Programmierer ab,
- kann genau angepasst und daher sehr effizient sein,
- allerdings hoher Aufwand

Coverage.

- im Grunde können alle Phänomene abgedeckt werden, für die es überhaupt Kriterien gibt
- allerdings schwer kontrollierbar

Drawing up lingware.

- prozedurale Sichtweise,
- keine Trennung von Programm und Lingware
- sehr unübersichtlich,
- schwer zu entwickeln und zu warten

Rat: The ATN formalism should be augmented by a UNIFY action that operates on complex categories as a whole.

Checklist PT-1, PT-2, PT-3, PT-4, PT-5, PT-6, PT-7, PT-8

➢

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4** **PT-5**
- procedural parser **PT-8**
- compiled parser PT-6 **PT-7**

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8**
- dependency descriptions **PT-5** **PT-8**

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- transition networks **PT-7** **PT-8**
- complement slots

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4** PT-6
- category reduction (bottom-up) **PT-5** PT-6
- state transition PT-6 **PT-7** **PT-8**
- slot filling

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8**
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8**
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4** PT-6 **PT-8**
- non-continuously (island parsing)

(6) Handling of alternatives

- backtracking **PT-1** **(PT-7)** **PT-8**
- parallel processing PT-2 **(PT-7)**
- looking ahead PT-3 PT-6
- welformed substring table **PT-4** **PT-5**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3 PT-6 **PT-7** **PT-8**
- all intermediate results stored (chart) **PT-4** **PT-5** **(PT-8)**

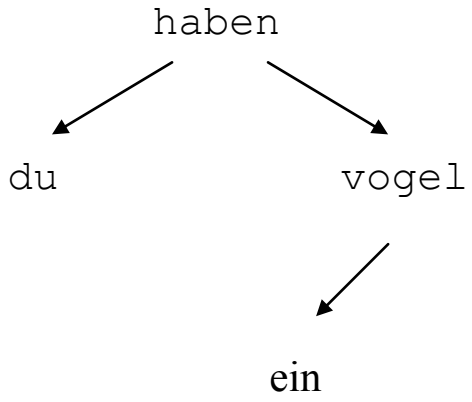
12. Stunde: Slot-Filler Parser für Dependenzgrammatiken (PT-9)

Definition einer Dependenz-Grammatik (angelehnt an DUG)

Repräsentation von Dependenzbäumen als Klammerstrukturen,komplexe Kategorien

Du hast einen Vogel

```
(lexem[haben]
     (lexem[du])
     (lexem[vogel]
          (lexem[ein])));
```



Du hast einen Vogel mit morpho-syntaktischen Katgeorien und Kongruenzmarkierung durch "C"

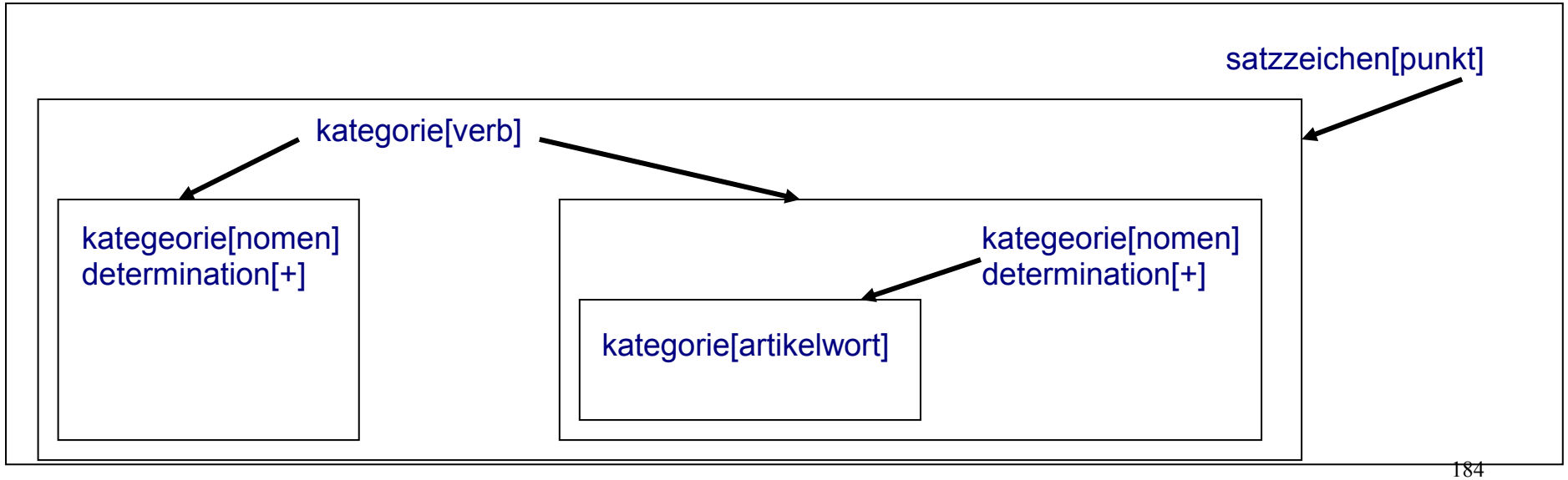
```
 (lexem[haben] wortart[verb] person[zweite] numerus[singular]
     (lexem[du] wortart[nomen] person[zweite,C] numerus[singular,C])
     (lexem[vogel wortart[nomen] numerus[singular]  kasus[akkusativ]
      genus[maskulinum]
          (lexem[ein] wortart[artikelwort] numerus[singular,C] kasus[akkusativ]
          genus[maskulinum,C])));
```

Während eine Phrasenstrukturgrammatik (PSG) die Zerlegung von größeren in kleinere Konstituenten beschreibt, beschreibt eine Dependenzgrammatik (DG) die Beziehung zwischen dominierenden lexikalischen Einheiten (heads) und Ergänzungen (complements, Komplemente).

PSG:

<table>
<tr><td colspan="5">kategeorie[satz] satztyp[hauptsatz]</td></tr>
<tr><td colspan="4">kategeorie[verb] ergaenzung[+]</td><td rowspan="3">satzzeichen[punkt]</td></tr>
<tr><td rowspan="2">kategeorie[nomen]
determination[+]</td><td rowspan="2">kategeorie[verb]
ergaenzung[-]</td><td colspan="2">kategeorie[nomen] determination[+]</td></tr>
<tr><td>kategeorie[artikelwort</td><td>kategeorie[nomen]
determination[-]</td></tr>
<tr><td>**Fritz**</td><td>**hilft**</td><td>**seinem**</td><td>**Vater**</td><td>**.**</td></tr>
</table>

DG:



Auch eine PSG kann man so schreiben, daß unter den unmittelbaren Konstituenten eine die dominierende Einheit (also der head) ist (z.B. N in der NP, V in der VP, P in der PP) und die grammatische Kategorie von dieser Einheit an die übergeordnete Konstituente weitergegeben wird:

PSG-REGEL (Subjekt zum Verb links davon):

(kategorie[verb] ergaenzung[+] stellungstyp[verb_zweit, verb_end]) **--->**	
(kategorie[nomen] kasus[nominativ] numerus[C] person[C] determination[+])	**(kategorie[verb] form[finit] numerus[C] person[C] verbtyp[C] subcat [subjekt] ergaenzung[-])**

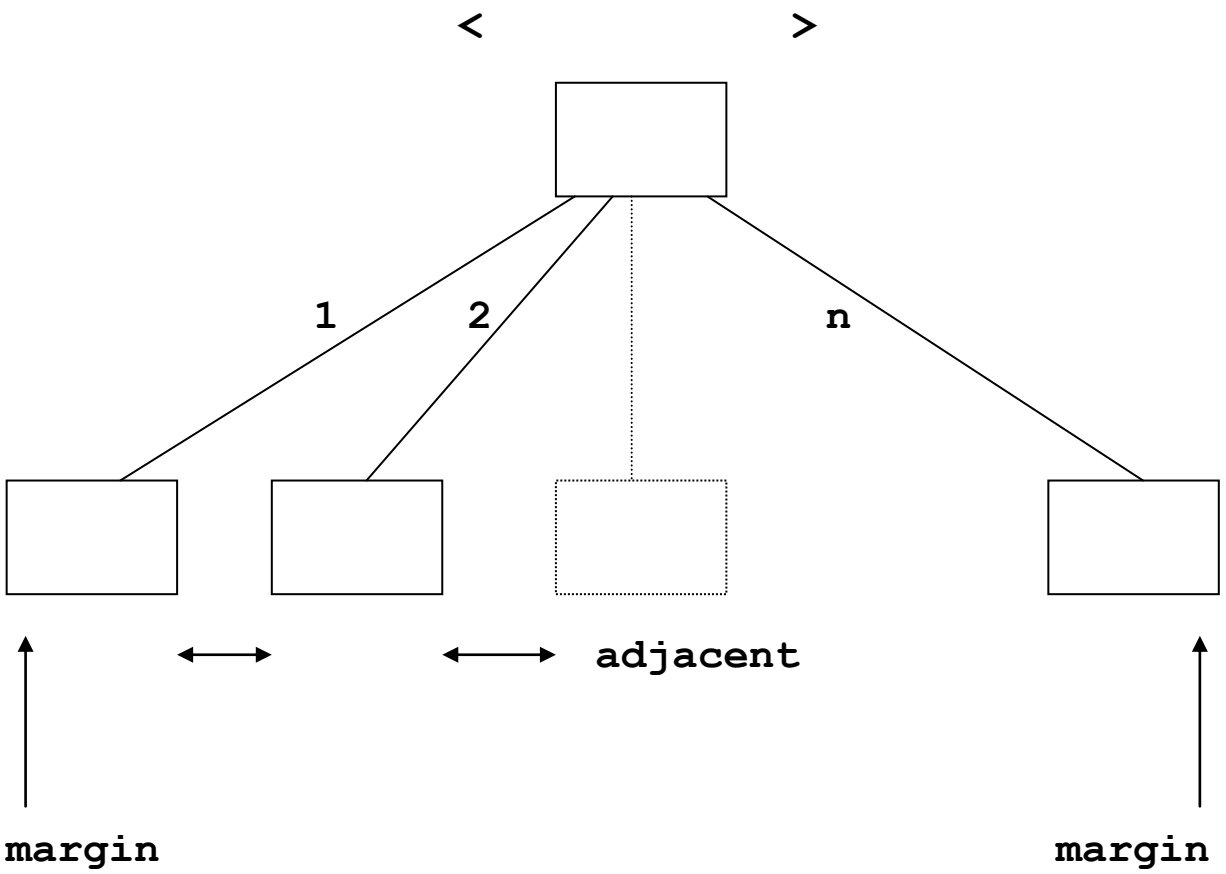
Statt Regeln gibt es in der DUG **Templates,** d.s. Muster für die Verbindung von einem übergeordneten Element und einer dependenten Konstituente mit einer bestimmten syntagmatischen Rolle:

DG-TEMPLATE (Subjekt zum Verb links davon):

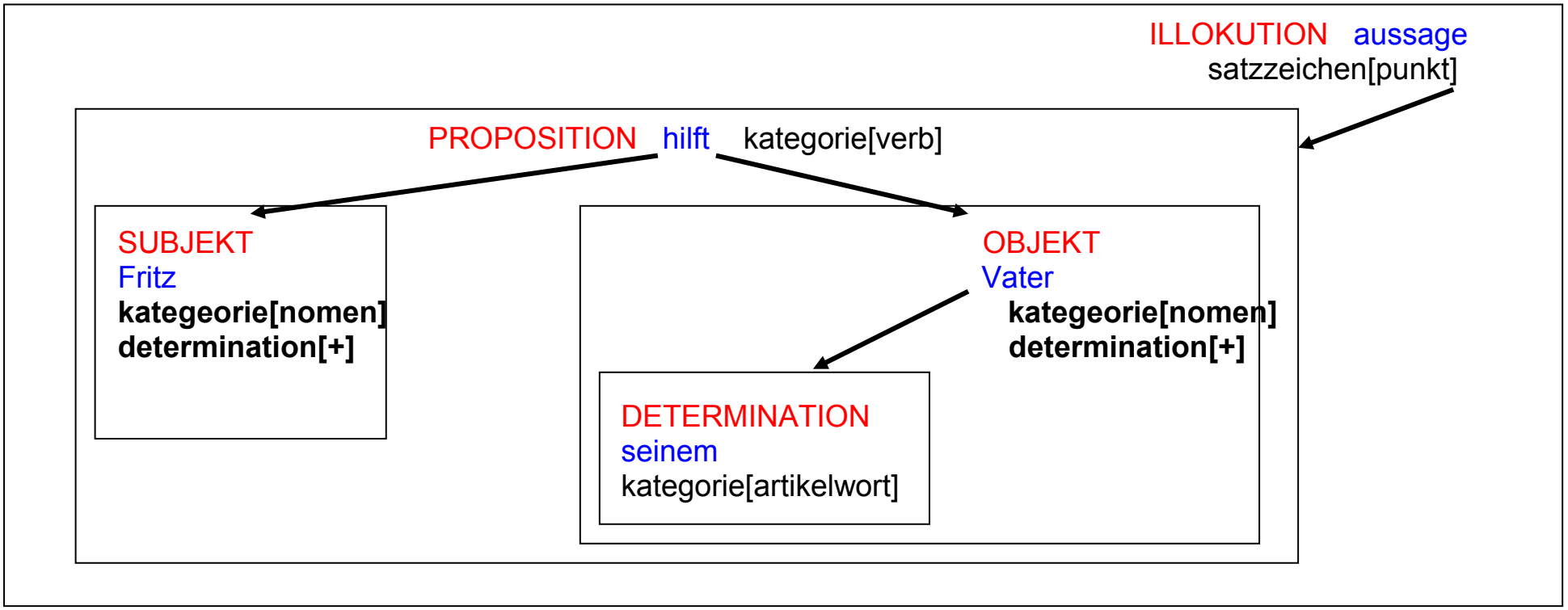
(template[+subjekt] kategorie[verb] form[finit] position[2] stellungstyp[verb_zweit, verb_end]
(< slot[regular] rolle[subjekt] kategorie[nomen] kasus[nominativ] numerus[C] person[C] determination[+] position[1]))

- *Jedes Template hat einen Namen. Über diesen wird es einem Wort in einem Synframe (s.u.) lexikalisch zugeordnet.*
- *Die untergeordnete Konstituente im Muster ist eine Art Variable (slot), in die der Kopf einer konkreten Konstituente aus der Umgebung eingesetzt werden kann.*
- *Die Reihenfolge der dependenten Konstituente relativ zum Head wird mit "<" für links und ">" für rechts markiert. Mit dem Attribut "position" können die Positionen von Konstituenten direkt angegeben werden.*
- *"C" als Wert eines Merkmals besagt, dass dieser Wert an den head weitergeben wird und mit dem dortigen Wert desselben Attributs übereinstimmen muss.*

Übersicht über Positionsmerkmale:



Für eine DG spricht, dass die Konstituenten mit syntagmatische Rollen versehen werden können:




```
(rolle[ILLOKUTION] lexem[aussage']...        Er hilft seinem Vater.

(rolle[ILLOKUTION] lexem[e_frage']...        Hilft er seinem Vater?

(rolle[ILLOKUTION] lexem[w_frage']...        Wer hilft meinem Vater?

(rolle[ILLOKUTION] lexem[aufforderung']...   Hilf deinem Vater!
```

Hat ein dominierendes Element (z.B. ein Verb) mehrere mögliche Ergänzungen, so wird für jede Ergänzung ein eigenes Template geschrieben. Im Lexikon werden die entsprechenden Komplemente in einer Synframe genannten Struktur aufgezählt (im Prinzip ist das ein Valenzwörterbuch):

SYNFRAMES (mit Subjekten und Objekten):

```
(lexem[kommen] kategorie[verb]
      (complement[+subjekt]

(lexem[freuen] kategory[verb]
      (complement[+subjekt, +subjekt_dass, +subjekt_infzu])
      (complement[+akk_objekt]))

(lexeme[sehen] kategorie[verb]
      (complement[+subjekt])
      (complement[+trans_objekt, +trans_objekt_dass]))
```

entspricht:

er kommt; mein Freund kommt;

das Angebot freut ihn; daß er kommt, freut den Vater; sie begrüßen zu dürfen, freut mich;

ich sehe das Haus; ich sehe, daß er kommt.

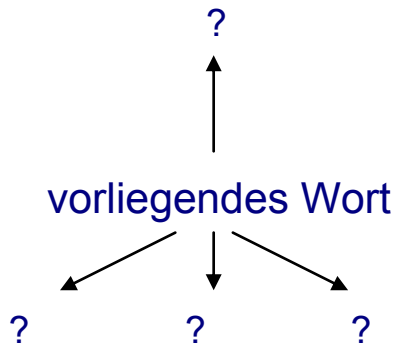
Adjunkte

Für die Unterscheidung von Komplementen und Adjunkten gibt es einen technischen Grund und einen inhaltlichen. Wir beschäftigen uns hier mit dem technischen:

Ein Wort an einem Knoten in einem Dependenzbaum kann in folgender Konstellation stehen:

(1)

(2)



d.h. es kann selbst von einem anderen Wort (1) abhängen und es kann mehrere abhängige Phrasen (2) dominieren (wobei jede Phrase wieder aus einem Dependenzbaum besteht, an dessen oberer Stelle ein Wort steht).

Entsprechend gibt es technisch zwei Möglichkeiten, um in einer Dependenzgrammatik eine syntagmatische Relation zu beschreiben:

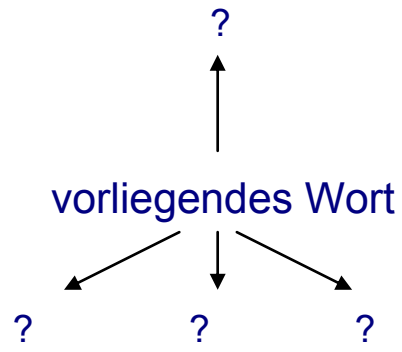
(1) zum vorliegenden Wort wird die Klasse der Wörter angegeben, von denen das Wort abhängig sein kann,
(2) zum vorliegenden Wort werden die Klassen der Wörter angegeben, die das Wort dominieren kann.

Da Abhängen und Dominieren inverse Relationen sind, hat man die Wahl zwischen (1) und (2): Entweder sucht sich ein Wort sozusagen selbst einen Head (in diesem Fall ist es ein "Adjunkt") oder es ist ein Head und verlangt nach Ergänzungen (m.a.W. nach "Komplementen").

Im ersten Fall haben wir eine bottom-up Suche, im zweiten eine top-down Suche.

(1)

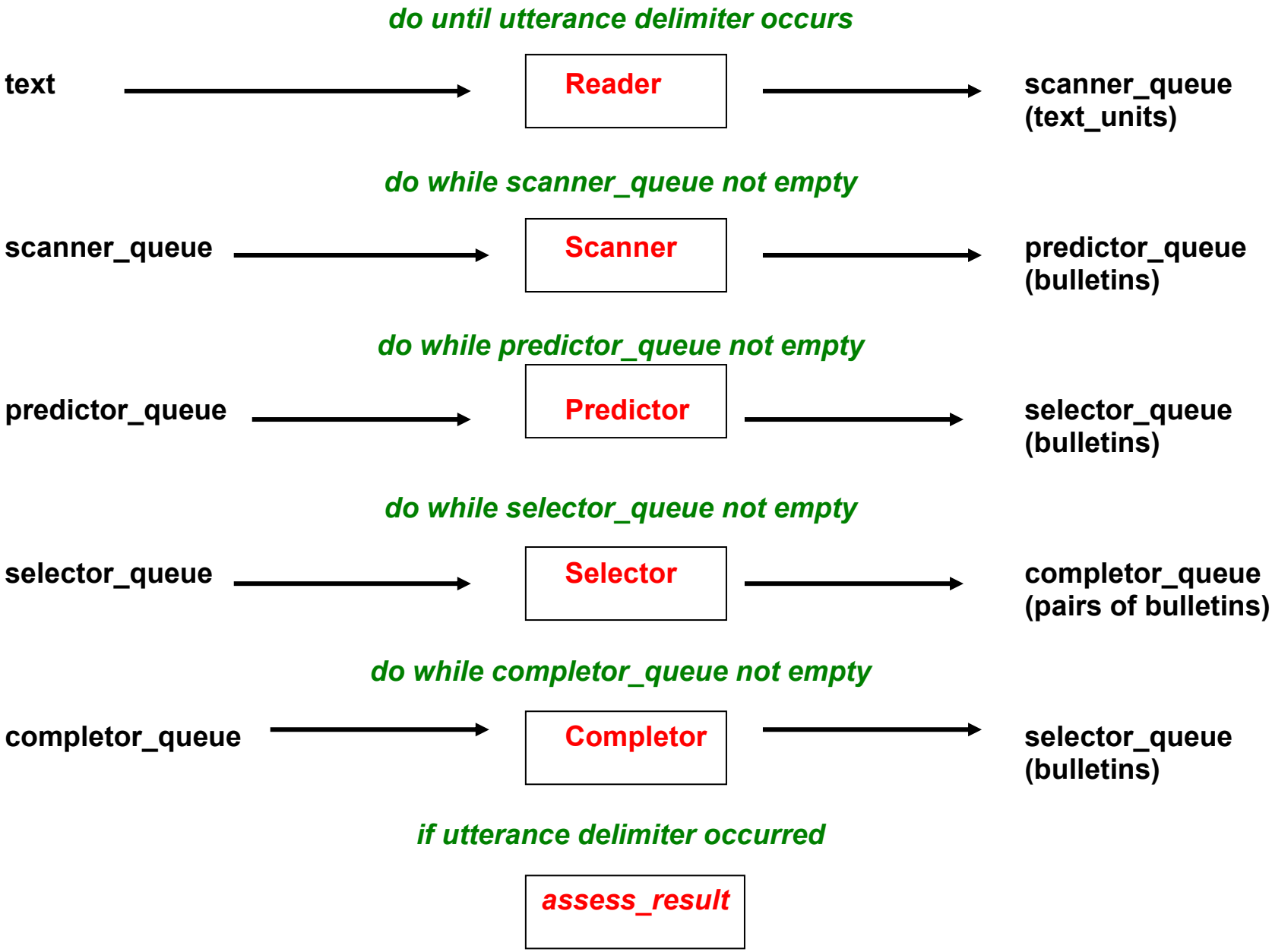
(2)



Wenn abhängige Konstituenten optional sind, ist es "billiger", sie als Adjunkte zu behandeln, statt als Komplemente. Als Adjunkte werden sie nämlich nur verarbeitet, wenn sie auch da sind ("datengesteuert"), während alle Komplement-Slots bei jedem Vergleich zweier Wörter im Satz überprüft werden müssen ("erwartungsgesteuert"), auch wenn das betreffende Element - da optional - gar nicht vorkommt.

Ein Dependenzparser sollte beides können. Wir betrachten im Folgenden aber nur das Komplement-Parsing.

Parallel arbeitender Slot-Filler Parser



Scanner

- Reads the input until the end of file.
- Extracts the lexemes and the morpho-syntactic information for each lexical item.
- Allocates a separate bulletin for each reading
- Sends the bulletin to the Predictor via the predictor-queue.

Predictor

- Reads next bulletin in the predictor-queue.
- Looks up the valency references for the lexeme.
- Inspects all the templates mentioned and copies all applicable slots into the bulletin.
- Sends the augmented bulletin to the Selector via the selector-queue.

Selector

- Reads the next bulletin from the selector queue.
- Retrieves all other bulletins (via the chart) that are candidates for combination with the given bulletin.
- Sends the resulting pairs of bulletins to the Completer via the completer-queue.

Completer

- Reads the next pair of bulletins from the completer-queue
- Inspects both bulletins for mutual slots.
- If a slot is found, then checks whether the other bulletin meets the filler requirements.
- Unifies each attribute within slot and filler.
- Unifies each attribute of filler and head if marked for agreement.
- Allocates a new bulletin and saves the information about the combined items in it.
- Sends the bulletin to the selector-queue.

BEISPIEL - MORPHO-SYNTACTIC LEXICON

sleep	(lex[*sleep*] cat[verb] form[infinitive]);
sleep	(lex[*sleep*] cat[verb] form[finite] person[1st, 2nd] number[singular]);
sleeps	(lex[*sleep*] cat[verb] form[finite] person[3rd] number[singular]);
sleep	(lex[*sleep*] cat[verb] form[finite] number[plural]);
feed	(lex[*feed]* cat[verb] form[infinitive]);
feed	(lex[*feed]* cat[verb] form[finite] person[1st, 2nd] number[singular]);
feeds	(lex[*feed]* cat[verb] form[finite] person[3rd] number[singular]);
supply	(lex[*supply]* cat[verb] form[infinitive]);
supplies	(lex[*supply]* cat[verb] form[finite] person[3rd] number[singular]);
fed	(lex[*feed]* cat[verb] form[past_part]);
do	(lex[*do]* cat[verb] form[infinitive]);
do	(lex[*do]* cat[verb] form[finite] person[1st, 2nd] number[singular]);
does	(lex[*do]* cat[verb] form[finite] person[3rd] number[singular]);
do	(lex[*do]* cat[verb] form[finite] number[plural]);
did	(lex[*do]* cat[verb] form[finite] person[3rd] number[singular]);
has	(lex[*have]* cat[verb] form[finite] person[3rd] number[singular]);
Gudrun	(lex[*Gudrun]* cat[noun] person[3rd] number[singular]);
cat	(lex[*cat]* cat[noun] person[3rd] number[singular]);
cats	(lex[*cat]* cat[noun] person[3rd] number[plural]);
fish	(lex[*fish]* cat[noun] person[3rd]);

I	(lex[*I*] cat[pron] person[1st] number[singular] case[subject]);
me	(lex[*I*] cat[pron] case[object]);
you	(lex[*you*] cat[pron] person[2nd]);
he	(lex[*he*] cat[pron] person[3rd] number[singular] case[subject]);
him	(lex[*he*] cat[pron] case[object]);
what	(lex[*what*] cat[wh_pron] person[3rd] number[singular] mode[quest,C]);
who	(lex[*who*] cat[wh_pron] person[3rd] number[singular] case[subject] mode[quest,C]);
whom	(lex[*who*] cat[wh_pron] case[object] mode[quest,C]);
the	(lex[*the*] cat[dete]);
a	(lex[*a*] cat[dete] number[singular]);
all	(lex[*all*] cat[dete] number[plural]);
her	(lex[*her*] cat[dete]);
silly	(lex[*silly*] cat[adje]);
to	(lex[*to*] cat[prep]);
with	(lex[*with*] *cat[*prep]);
?	(lex[*question'*] cat[particle] utterance[+]);
.	(lex[*assertion'*] cat[particle] utterance[+]);

TEMPLATES

```
(template[+question]
 role[ILLOCUTION] cat[particle] sent_position[7]
         (< slot[oblig] role[PREDICATE] cat[verb] form[finite] mode[quest]
         sent_position[2,C]))

(template[+assertion]
 role[ILLOCUTION] cat[particle] sent_position[7]
         (< slot[oblig] role[PREDICATE] cat[verb] form[finite] mode[assert]
         sent_position[4,C]))

(template[+aux_subject]
 cat[verb] form[finite] sent_position[2]
         (> slot[oblig] role[SUBJECT] cat[noun] mode[quest,C]) number[C]
         person[C] sent_position[3,C]))

(template[+aux_subject]
 cat[verb] form[finite] sent_position[2]
         (> slot[oblig] role[SUBJECT] cat[pron] case[subject] mode[quest,C])
         number[C] person[C] sent_position[3,C]))
```

```
(template[+infinitiv ]
 cat[verb] form[finite] sent_position[2]
         (> slot[oblig, nuleus] role[PRED_COMPLEMENT] cat[verb] form[infinitive]
         mode[C] sent_position[4,C]))

(template[+participle ]
 cat[verb] form[finite] sent_position[2]
         (> slot[oblig, nucleus] role[PRED_COMPLEMENT] cat[verb] form[past_part]
         mode[C] sent_position[4,C))

(template[+subject]
 cat[verb] form[finite] sent_position[4]
         (< slot[oblig] role[SUBJECT] cat[noun] mode[assert,C]) number[C]
         person[C] sent_position[3,C]))

(template[+subject]
 cat[verb] form[finite] sent_position[4]
         (< slot[oblig] role[SUBJECT] cat[pron] case[subject] mode[assert,C])
         number[C] person[C] sent_position[3,C]))

(template[+subject]
 cat[verb] form[finite] sent_position[4]
         (< slot[oblig] role[SUBJECT] cat[wh_pron] mode[quest,C] person[C]
         number[C] sent_position[1,C]))
```

```
(template[+dir_object]
 cat[verb] sent_position[4]
         (> slot[optional] role[DIR_OBJECT] cat[noun] sent_position[5,6,C]))

(template[+dir_object]
 cat[verb] sent_position[4]
         (> slot[optional] role[DIR_OBJECT] cat[pron] case[object]
         sent_position[5,C]))

(template[+dir_object]
 cat[verb] form[infinitive] sent_position[4]
         (< slot[optional] role[DIR_OBJECT] cat[wh_pron] mode[quest,C]
         case[object] sent_position[1,C] discont[left]))

(template[+indir_object]
 cat[verb] sent_position[4]
         (> slot[optional] role[INDIR_OBJECT] cat[noun] sent_position[5,C]))

(template[+indir_object]
 cat[verb] sent_position[4]
         (> slot[optional] role[INDIR_OBJECT] lex[to] cat[prep]
         sent_position[6,C]))
```

```
(template[+indir_object]
 cat[verb] form[infinitive] sent_position[4]
         (< slot[optional] role[INDIR_OBJECT] lex[to] cat[prep] mode[quest,C]
         sent_position[1,C] discont[left]))

(template[+count]
 cat[noun] number[singular] np_position[3]
         (< slot[oblig] role[DETERMINER] cat[dete] number[C] np_position[1,C]))

(template[+count]
 cat[noun] number[plural] np_position[3]
         (< slot[optional] role[DETERMINER] cat[dete] number[C]
         np_position[1,C]))

(template[+mass]
 cat[noun] number[singular] np_position[3]
         (< slot[optional] role[DETERMINER] cat[dete] np_position[1,C]))

(template[+attribute]
 cat[noun] np_position[3]
         (< slot[optional, sequence] role[ATTRIBUTE] cat[adje] np_position[2,C]))
```

```
(template[+prep_attribute]
 cat[noun] np_position[3]
         (> slot[optional] role[PREP_ATTRIBUTE] cat[prep] np_position[4,C]))

(template[+prep_phrase]
 cat[prep] pp_position[1]
         (> slot[oblig] role[PREP_COMPL] cat[noun] pp_position[2,C]))

(template[+prep_phrase]
 cat[prep] pp_position[1]
         (> slot[oblig] role[PREP_COMPL] cat[pron] case[object]
         pp_position[2,C]))
```

SYNFRAMES (equivalent to lexical subcategorization in generative grammar)

```
(lexeme[sleep] cat[verb]
    (complement[+subject]

(lexeme[feed] cat[verb]
    (complement[+subject, +direct_object, +indicrect_object])

(lexeme[do] cat[verb]
    (complement[+aux_subject, +infinitiv])

(lexeme[have] cat[verb]
    (complement[+aux_subject, +participle])

(lexeme[cat] cat[noun]
    (complement[+count, +attribute])

(lexeme[fish] cat[noun]
    (complement[+count, +attribute])

(lexeme[fish] cat[noun]
    (complement[+mass, +attribute])

(lexeme[to] cat[prep]
    (complement[+prep_phrase])

(lexeme[with] cat[prep]
    (complement[+prep_phrase])

(lexeme[question'] cat[particle]
    (complement[+question])

(lexeme[assertion'] cat[particle]
    (complement[+assertion])
```

1 MORPHO-SYNTACTIC LEXICON

Scanner

```
does    (lex[do] cat[verb] form[finite] person[3rd] number[singular]);
```

2 SYNFRAME (VALENCY REFERENCES)

Predictor

```
lex[do] cat[verb]   ->  +aux_subject, +infinitiv
```

3 TEMPLATES

```
(template[+aux_subject]
(cat[verb] form[finite] sent_position[2]
           (> slot[oblig] role[SUBJECT] cat[noun] mode[quest,C]) number[C]
           person[C] sent_position[3,C])));

(template[+aux_subject]
(cat[verb] form[finite] sent_position[2]
           (< slot[oblig] role[SUBJECT] cat[pron] case[subjective] mode[quest,C])
           number[C] person[C] sent_position[1,C])));

(template[+infinitiv]
(cat[verb] form[finite] sent_position[2]
           (> slot[oblig, nucleus] role[PRED_COMPLEMENT] cat[verb]
           form[infinitive] mode[C] sent_position[4,C])));
```

4 BULLETIN

4 BULLETIN

(2)	**does**	1	5	-	-	N

(lex[*do*] cat[verb] form[finite] person[3rd] number[singular] sent_position[2]

{(> slot[oblig] role[SUBJECT] cat[noun] mode[quest,C]) number[C] person[C] sent_position[3,C])
(< slot[oblig] role[SUBJECT] cat[pron] case[subjective] mode[quest,C]) number[C] person[C] sent_position[1,C]) }

(> slot[oblig, nucleus] role[PRED_COMPLEMENT] cat[verb] form[infinitive] mode[C] sent_position[4,C]));

INPUT

Input:	what	does	Gudrun	feed	her	cat	?
Position:	1	2	3	4	5	6	7

WORKING AREA (BULLETINS) **Selector, Completer**

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

(lex[***what***] cat[wh pron] person[3rd] number[singular] mode[quest,C]);

(2)	**does**	2	2	-	-	N

(lex[***do***] cat[verb] form[finite] person[3rd] number[singular] sent_position[2]
{(> **slot**[oblig] role[SUBJECT] cat[noun] mode[quest,C]) number[C] person[C] sent_position[3,C])|
(> **slot**[oblig] role[SUBJECT] cat[pron] case[subject] mode[quest,C]) number[C] person[C]) sent_position[3,C]) }
(> **slot**[oblig, nuleus] role[PRED_COMPLEMENT] cat[verb] form[infinitive] mode[C] sent_position[4,C]));

(3)	**Gudrun**	3	3	-	-	N

```
(lex[Gudrun] cat[noun] person[3rd] number[singular]);
```

(4)	**does Gudrun**	2	3	(2)	(3)	N

```
(lex[do] cat[verb] form[finite] person[3rd] number[singular] mode[quest]
sent_position[2,3]
     (> role[SUBJECT] lex[Gudrun] cat[noun] mode[quest,C]) number[singular,C]
     person[3rd,C] sent_position[3,C])
     (> slot[oblig, nuleus] role[PRED_COMPLEMENT] cat[verb] form[infinitive]
     mode[C] sent_position[4,C]));
```

(5)	**feed**	4	4	-	-	N

```
(lex[feed] cat[verb] form[infinitive] sent_position[4]
     {(> slot[optional] role[DIR_OBJECT] cat[noun] sent_position[5,6,C]) |
      (> slot[optional] role[DIR_OBJECT] cat[pron] case[object]
     sent_position[5,C]) |
      (< slot[optional] role[DIR_OBJECT] cat[wh_pron] mode[quest,C] case[object]
     sent_position[1,C] discont[left])}
     {(> slot[optional] role[INDIR_OBJECT] cat[noun] sent_position[5,C]) |
      (> slot[optional] role[INDIR_OBJECT] lex[to] cat[prep] sent_position[6,C])
     |
      (< slot[optional] role[INDIR_OBJECT] lex[to] cat[prep] mode[quest,C]
     sent_position[1,C] discont[left])} );
```

(6)	**feed**	4	4	-	-	N

```
(lex[feed] cat[verb] form[finite] person[1st, 2nd]  number[singular]
sent_position[4]
     (< slot[oblig] role[SUBJECT] cat[pron] case[subject] mode[assert,C])
     number[C] person[C] sent_position[3,C])
     {(> slot[optional] role[DIR_OBJECT] cat[noun] sent_position[5,6,C]) |
      (> slot[optional] role[DIR_OBJECT] cat[pron] case[object]
     sent_position[5,C]) }
     {(> slot[optional] role[INDIR_OBJECT] cat[noun] sent_position[5,C]) |
      (> slot[optional] role[INDIR_OBJECT] lex[to] cat[prep]
     sent_position[6,C])} );
```

(7)	**does Gudrun feed**	2	4	(4)	(5)	Y

```
(lex[do] cat[verb] form[finite] person[3rd] number[singular]) mode[quest]
sent_position[2,3,4]
     (> role[SUBJECT] lex[Gudrun] cat[noun] mode[quest,C]) number[C] person[C]
     sent_position[3,C])
     (> role[PRED_COMPLEMENT] lex[feed] cat[verb] form[infinitive]
     sent_position[4,C])
         (< slot[optional] role[DIR_OBJECT] cat[wh_pron] mode[quest,C]
         case[object] sent_position[1,C] discont[left])));
```

(8)	**what does Gudrun feed**	1	4	(7)	(1)	N

```
(lex[do] cat[verb] form[finite] person[3rd] number[singular] mode[quest]
sent_position[1,2,3,4]
      (> role[SUBJECT] lex[Gudrun] cat[noun] mode[quest,C]) number[C] person[C]
      sent_position[3,C])
      (> role[PRED_COMPLEMENT] lex[feed] cat[verb] form[infinitive]
      sent_position[4,C]
         (< role[DIR_OBJECT] lex[what] cat[wh_pron] mode[quest,C] case[object]
         sent_position[1,C] discont[left])));
```

(9)	**her**	5	5	-	-	N

```
(lex[her] cat[dete]);
```

(10)	**cat**	6	6	-	-	N

```
(lex[cat] cat[noun] person[3rd] number[singular] np_position[3]
      (< slot[oblig] role[DETERMINER] cat[dete] number[C] np_position[1,C])
      (< slot[optional, sequence] role[ATTRIBUTE] cat[adje] np_position[2,C])
      (> slot[optional] role[PREP_ATTRIBUTE] cat[prep] np_position[4,C]));
```

(11)	**her cat**	5	6	(10)	(9)	N

```
(lex[cat] cat[noun] person[3rd] number[singular] np_position[1,3]
```

```
      (role[DETERMINER] lex[her] cat[dete] number[C] np_position[1,C])
      (> slot[optional] role[PREP_ATTRIBUTE] cat[prep] np_position[4,C]));
```

(12)	feed her cat	4	6	(5)	(11)	Y

```
(lex[feed] cat[verb] form[infinitive] sent_position[4,5]
      {(< slot[optional] role[DIR_OBJECT] cat[wh_pron]  mode[quest,C]
      case[object] sent_position[1,C] discont[left]) |
      (> slot[optional] role[DIR_OBJECT] cat[noun] sent_position[6,C])}
      (> role[INDIR_OBJECT] lex[cat] cat[noun] person[3rd] number[singular]
      np_position[1,3] sent_position[5,C]
         (role[DETERMINER] lex[her] cat[dete] number[C] np_position[1,C])));
```

(13)	feed her cat	4	6	(6)	(11)	N

```
(lex[feed] cat[verb] form[finite] person[1st, 2nd]  number[singular]
sent_position[4,5]
      (< slot[oblig] role[SUBJECT] cat[pron] case[subject] mode[assert,C])
      number[C] person[C] sent_position[3,C])
      (> slot[optional] role[DIR_OBJECT] cat[noun] sent_position[6,C])
      (> role[INDIR_OBJECT] lex[cat] noun person[3rd] number[singular]
      np_position[1,3] sent_position[5,C]
         (role[DETERMINER] lex[her] cat[dete] number[C] np_position[1,C])));
```

(14) **does Gudrun feed her cat** 2 6 (4) (12) Y

(lex[***do***] cat[verb] form[finite] person[3rd] number[singular] mode[quest] sent_position[2,3,4,5]
(> role[**SUBJECT**] lex[***Gudrun***] cat[noun] mode[quest,C]) number[singular,C] person[3rd,C] sent_position[3,C])
(> role[**PRED_COMPLEMENT**] lex[***feed***] verb form[infinitive] sent_position[4,6,C]
(< **slot**[optional] role[DIR_OBJECT] cat[wh_pron] mode[quest,C] case[object] sent_position[1,C] discont[left])
(> role[**INDIR_OBJECT**] lex[***cat***] cat[noun] person[3rd] number[singular] np_position[1,3] sent_position[5,C]
(< role[**DETERMINER**] lex[***her***] cat[dete] number[C] np_position[1,C])));

(15) **what does Gudrun feed her cat** 1 6 (14) (1) N

(lex[***do***] cat[verb] form[finite] person[3rd] number[singular] mode[quest] sent_position[1,2,3,4,5]
(> role[**SUBJECT**] lex[***Gudrun***] cat[noun] mode[quest,C]) number[singular,C] person[3rd,C] sent_position[3,C])
(> role[**PRED_COMPLEMENT**] lex[***feed***] verb form[infinitive] sent_position[1,4,6,C]
(< role[**DIR_OBJECT**] lex[***what***] cat[wh_pron] mode[quest,C] case[object] sent_position[1,C] discont[left])
(> role[**INDIR_OBJECT**] lex[***cat***] cat[noun] person[3rd] number[singular] np_position[1,3] sent_position[5,C]

```
                (< role[DETERMINER] lex[her] cat[dete] number[C]
                np_position[1,C])));
```

(16)	**?**	7	7	-	-	N

```
(role[ILLOCUTION] lex[question'] cat[particle] utterance[+] sent_position[7]
      (< slot[oblig] role[PREDICATE] cat[verb] form[finite] mode[quest]
      sent_position[2,C]));
```

(17)	**what does Gudrun feed her cat ?**	1	7	(16)	(15)	N

```
(role[ILLOCUTION] lex[question'] cat[particle] utterance[+]sent_position[7]
      (< role[PREDICATE] lex[do] cat[verb] form[finite] person[3rd]
      number[singular] mode[quest] sent_position[1,2,3,4,5]
         (> role[SUBJECT] lex[Gudrun] cat[noun] mode[quest,C])
         number[singular,C] person[3rd,C] sent_position[3,C])
         (> role[PRED_COMPLEMENT] lex[feed] verb form[infinitive]
         sent_position[1,4,6,C]
              (< role[DIR_OBJECT] lex[what] cat[wh_pron] mode[quest,C]
              case[object] sent_position[1,C] discont[left])
              (> role[INDIR_OBJECT] lex[cat] cat[noun] person[3rd]
              number[singular] np_position[1,3] sent_position[5,C]
                   (< role[DETERMINER] lex[her] cat[dete] number[C]
                   np_position[1,C])));
```

RESULT (without morho-syntactic categories):

```
(ILLOCUTION: question'
    (PREDICATE: do
        (SUBJECT: Gudrun)
        (PRED_COMPLEMENT: feed
            (DIR_OBJECT: what)
            (INDIR_OBJECT: cat
                (DETERMINER: her))));
```

Evaluation

Efficiency.

- bottom-up und (auf besondere Weise) top-down zugleich, "data-driven expectation"
- lexikalistisch, Suchraum der Alternativen sehr viel begrenzter als bei Regelgrammatiken
- Übergenerierung wegen mangelndem Linksanschluß

Coverage.

- Attributgrammatik (*Knuth* 1968, *Van Wijngaarden* 1969, *Pagan* 1981).
- Its power depends on the kind of attributes used.
- Positionsattribute übersteigen Kontextfreiheit.
- Weitere Attribute dosiert hinzu, bis notwenige Kapazität erreicht.

Drawing up lingware.

- Ziemlich einfach, überschaubar, ohne Seiteneffekte

Checklist PT-1, PT-2, PT-3, PT-4, PT-5, PT-6, PT-7, PT-8

(1) Connection between grammar and parser

- interpreting parser **PT-1** PT-2 PT-3 **PT-4** **PT-5** **PT-9**
- procedural parser **PT-8**
- compiled parser PT-6 **PT-7**

(2) Linguistic structure assigned

- constituency descriptions **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8**
- dependency descriptions **PT-5** **PT-8** **PT-9**

(3) Grammar specification format

- production rules **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6
- transition networks **PT-7** **PT-8**
- complement slots **PT-9**

(4) Recognition strategy

- category expansion (top-down) **PT-1** PT-2 PT-3 **PT-4** PT-6
- category reduction (bottom-up) **PT-5** PT-6
- state transition PT-6 **PT-7** **PT-8**
- slot filling **PT-9**

(5) Processing the input

- from left to right or from right to left **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8** **PT-9**
- one-pass (depth-first) **PT-1** PT-2 PT-3 **PT-4** **PT-5** PT-6 **PT-7** **PT-8** **PT-9**
- several passes (breadth-first)
- left-associative **PT-1** PT-2 PT-3 **PT-4** PT-6 **PT-8**
- non-continuously (island parsing) (**PT-9**)

(6) Handling of alternatives

- backtracking **PT-1** (**PT-7**) **PT-8**
- parallel processing PT-2 (**PT-7**)
- looking ahead PT-3 PT-6
- welformed substring table **PT-4** **PT-5** **PT-9**

(7) Control of results

- goal oriented recognition of final result(s) **PT-1** PT-2 PT-3 PT-6 **PT-7** **PT-8**
- all intermediate results stored (chart) **PT-4** **PT-5** (**PT-8**) **PT-9**

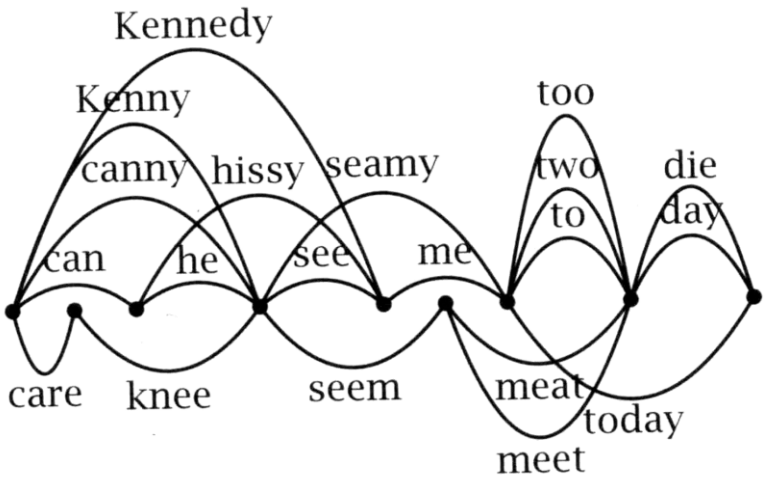
13. Stunde: Parsing mit statistischen Methoden

Gründe für den Einsatz von Statistik beim Parsen

- Beschleunigen des Parsers
- Disambiguierung, Wahl zwischen mehreren Parsingergebnissen
- Spracherkennung, Parsen bei unsicherer Eingabe

Tagger

"Kandidaten" in der Spracherkennung



can-V he-PRON see-V me-PRON today-ADV

Wortartentagger (part-of-speech-tagger) zur provisorischen syntaktischen Disambiguierung

Problem Ambiguität

Beispiel ENGTWOL Output - Fett = korrektes Tag

Pavlov	**PAVLOV N NOM SG PROPER**
had	**HAVE V PAST VFIN SVO**
	HAVE PCP2 SVO
shown	**SHOW PCP2 SVOO SVO SV**
that	ADV
	PRON DEM SG
	DET CENTRAL DEM SG
	CS
salivation	**N NOM SG**
...	

Unambiguous (1 tag)	**35,340**
Ambiguous (2–7 tags)	**4,100**
2 tags	3,760
3 tags	264
4 tags	61
5 tags	12
6 tags	2
7 tags	1

nach DeRose (1988)

http://128.214.88.72/cgi-bin/parser-demo.pl

English Machinese Phrase Tagger 4.3 analysis:

time	time	@PREMOD	N
	time	@NH	N
flies	fly	@NH	N PL
	fly	@MAIN	V IND PRES
like	like	@MAIN	V IND PRES
	like	@POSTMOD	PREP
	like	@PREMARK	PREP
an	an	@PREMOD	DET
arrow	arrow	@NH	N

Note: The Connexor Machinese demos are intended for evaluation purposes only.

Auflösung der Ambiguität in ENGTWOL mit speziellen, manuell erstellten Regeln

ADVERBIAL-THAT RULE
Given input: "that"
if
(+1 A/ADV/QUANT); /* *if next word is adj, adverb, or quantifier* */
(+2 SENT-LIM); /* *and following which is a sentence boundary,* */
(NOT -1 SVOC/A); /* *and the previous word is not a verb like* */
/* *'consider' which allows adjs as object complements* */
then eliminate non-ADV tags
else eliminate ADV tag

Andere Möglichkeit: Stochastische Methoden

Maximieren der folgenden Formel

P(word | tag) * P(tag | n previous tags)

= die höchste Wahrscheinlichkeit, daß ein Wort einen bestimmten Tag hat auf dem Hintergrund der Wahrscheinlichkeit, daß dieser Tag nach n vorangehenden Tags vorkommt.

Hidden-Morkov Modell (HMM)

bi-gram, tri-gram, n-gram Wahrscheinlichkeiten

	Second tag					
First tag	AT	BEZ	IN	NN	VB	PERIOD
AT	0	0	0	48636	0	19
BEZ	1973	0	426	187	0	38
IN	43322	0	1325	17314	0	185
NN	1067	3720	42470	11773	614	21392
VB	6072	42	4758	1476	129	1522
PERIOD	8016	75	4656	1329	954	0

Zählung einiger Wortarten-Digramme im Brown corpus

Markov-Kette = Übergangswahrscheinlichkeiten zwischen fortlaufenden n-Grammen

Berechnung der größten Wahrscheinlichkeit durch die ganze Kette

(mit bestimmten statistischen Formeln. z.B. Viterbi-Algorithmus)

Brill Tagger (Transformation-Based Learning)

Gegeben ein Lexikon mit Häufigkeiten

P(NN| race) =.98
P(VB| race) =.02

Zunächst wird das häufigste Tag genommen:

... is-VBZ expected-VBN to-TO race-NN tomorrow-NN

... the-DT race-NN for-IN outer-JJ space-NN

Sodann werden Verbesserungen angebracht, bis das Ergebnis perfekt ist

... is-VBZ expected-VBN to-TO race-**VB** tomorrow-NN

Verbesserungen mit Transformationregeln:

Umgebungsbedingung, tag1 -> tag2

Transformationen werden trainiert: so lange wie das Ergebnis noch nicht korrekt ist und das Resultat der Transformationen besser ist als die Ausgangstags

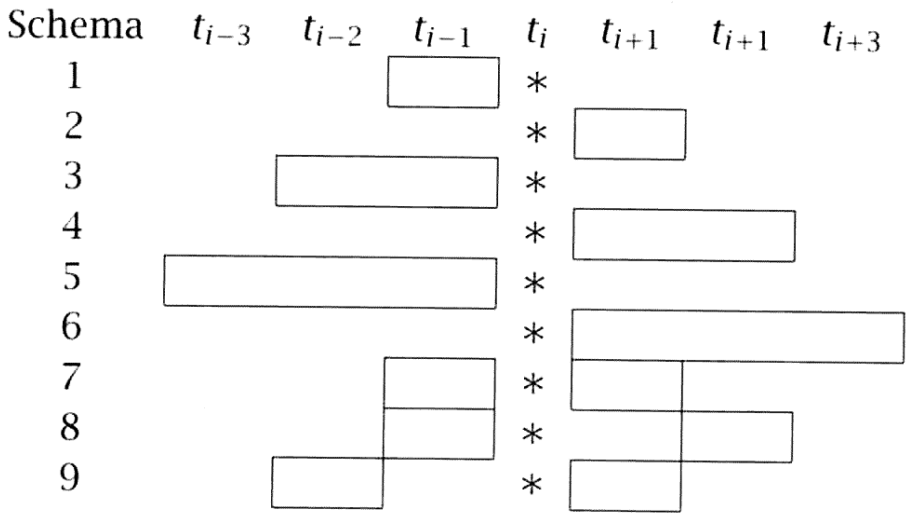


Table 10.7 Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag t^j occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag t^j occurs two positions earlier and tag t^k occurs in the following position."

Source tag	Target tag	Triggering environment
NN	VB	previous tag is TO
VBP	VB	one of the previous three tags is MD
JJR	RBR	next tag is JJ
VBP	VB	one of the previous two words is *n't*

Table 10.8 Examples of some transformations learned in transformation-based tagging.

Probalistic Context Free Grammar (PCFG)

Aufgabe: den wahrscheinlichsten Parse (häufigsten) zu finden (Disambiguierung oder Beschleunigung)

G= <NT, T, R, S, P>

P eine Menge von Wahrscheinlichkeiten der Regelanwendungen

A -> β [p]
P (A-> β)

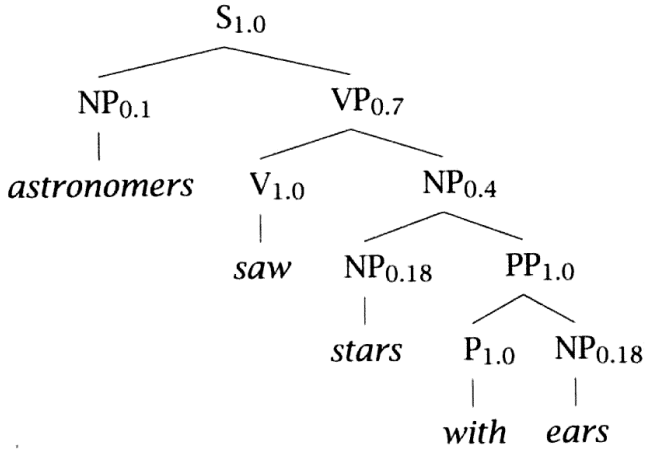
Summe p aller Regeln zu einem bestimmten NT = 1

$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that$ [.05] | the [.80] | a [.15]	
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book$	[.10]
$S \rightarrow VP$	[.05]	$Noun \rightarrow flights$	[.50]
$NP \rightarrow Det\ Nom$	[.20]	$Noun \rightarrow meal$	[.40]
$NP \rightarrow Proper\text{-}Noun$	[.35]	$Verb \rightarrow book$	[.30]
$NP \rightarrow Nom$	[.05]	$Verb \rightarrow include$	[.30]
$NP \rightarrow Pronoun$	[.40]	$Verb \rightarrow want$	[.40]
$Nom \rightarrow Noun$	[.75]	$Aux \rightarrow can$	[.40]
$Nom \rightarrow Noun\ Nom$	[.20]	$Aux \rightarrow does$	[.30]
$Nom \rightarrow Proper\text{-}Noun\ Nom$	[.05]	$Aux \rightarrow do$	[.30]
$VP \rightarrow Verb$	[.55]	$Proper\text{-}Noun \rightarrow TWA$	[.40]
$VP \rightarrow Verb\ NP$	[.40]	$Proper\text{-}Noun \rightarrow Denver$	[.40]
$VP \rightarrow Verb\ NP\ NP$	[.05]	$Pronoun \rightarrow you$ [.40] | I [.60]	

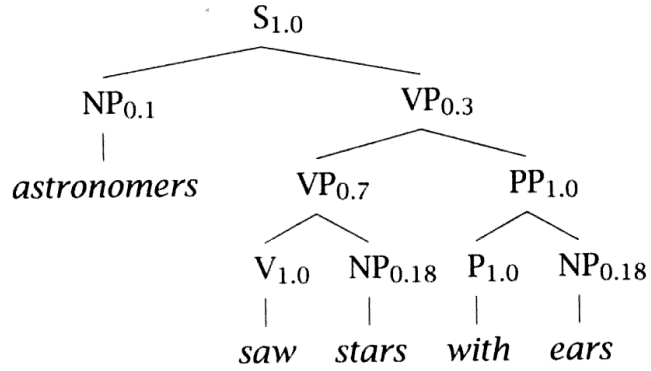
Ein anderes Beispiel:

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

t_1:



t_2:



$$
\begin{aligned}
P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0009072 \\
P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0006804 \\
P(w_{15}) &= P(t_1) + P(t_2) = 0.0015876
\end{aligned}
$$

Multiplikation der Wahrscheinlichkeiten aller benutzten Regeln

Manning/Schütze S. 384/385

- ✓ Jeder Parse Tree bekommt so einen Wert: Multiplikation aller Wahrscheinlichkeiten der Regeln für jeden Knoten.

- ✓ Die Wahrscheinlichkeit des Strings von Wörtern ist dieselbe wie die des zugehörigen Parsebaums. Dies kann in der Spracherkennung zur Einschränkung von Hypothesen eingesetzt werden.

Das geht auch für Teilstrings (z.B. zu jedem Eintrag in der Chart des Earley oder Cocke-Parsers)

Zwei Maße**:**

inside probability = die Multiplikation der Wahrscheinlichkeiten der Expansion einer Kategorie

outside probabilty = die Multiplikation der Wahrscheinlichkeiten der Reduktion der Kategorie auf S

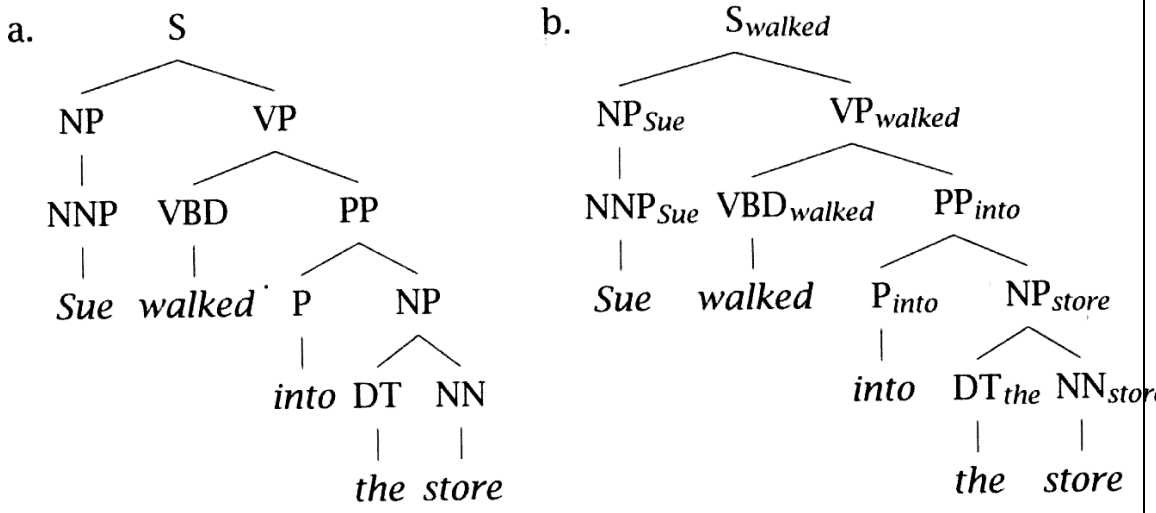
Kritik:

Unabhängigkeits-Axiom der PCFG ist falsch (insofern schlechter als n-gram Modelle)

Probalistic Lexicalized CFGs

	Verb			
Local tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

Regelhäufigkeit in Abhängigkeit von den Lexemen



Lexikalisierte Bäume

Woher kommen die statistischen Werte?

Können gar die Regeln statistisch gewonnen werden?

"grammar induction"

Evtl. Regelinduktion bei regulären Grammatiken möglich, bei kontextfreien und darüber ist das ausgeschlossen.

Grundlage sind vielmehr **manuelle annotierte** oder vorher korrekt geparste Sätze

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (PP against
                         (NP Arizona real estate loans)))))
         ,
         (S-ADV (NP-SBJ *)
                (VP reflecting
                    (NP (NP a continuing decline)
                        (PP-LOC in
                                (NP that market))))))
     .))
```

Beispiel aus der Penn Treebank

Chunking and Raising Parser von Yue-Ski Lee

- real-time Parser müssen schneller sein, als Technologie vollständiger Parser erlaubt
- statt dessen *partial parser*, die nur bestimmte Konstituenten herausziehen
- möglichst deterministisch
- Voraussetzung: lokale Ambiguität muss sofort aufgelöst werden,

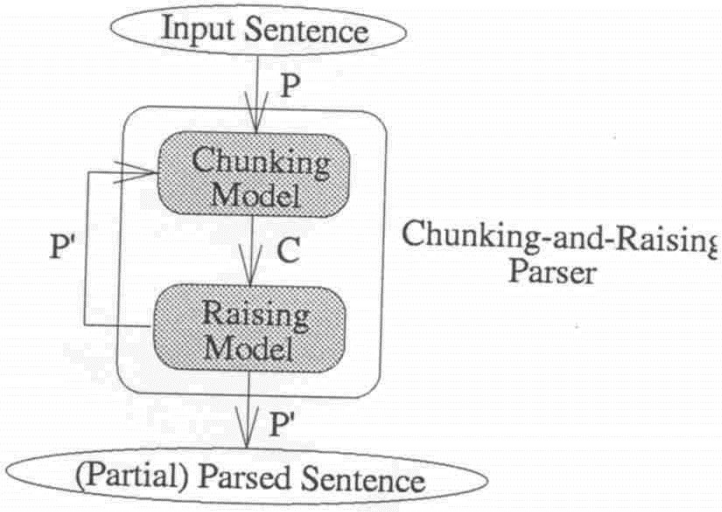
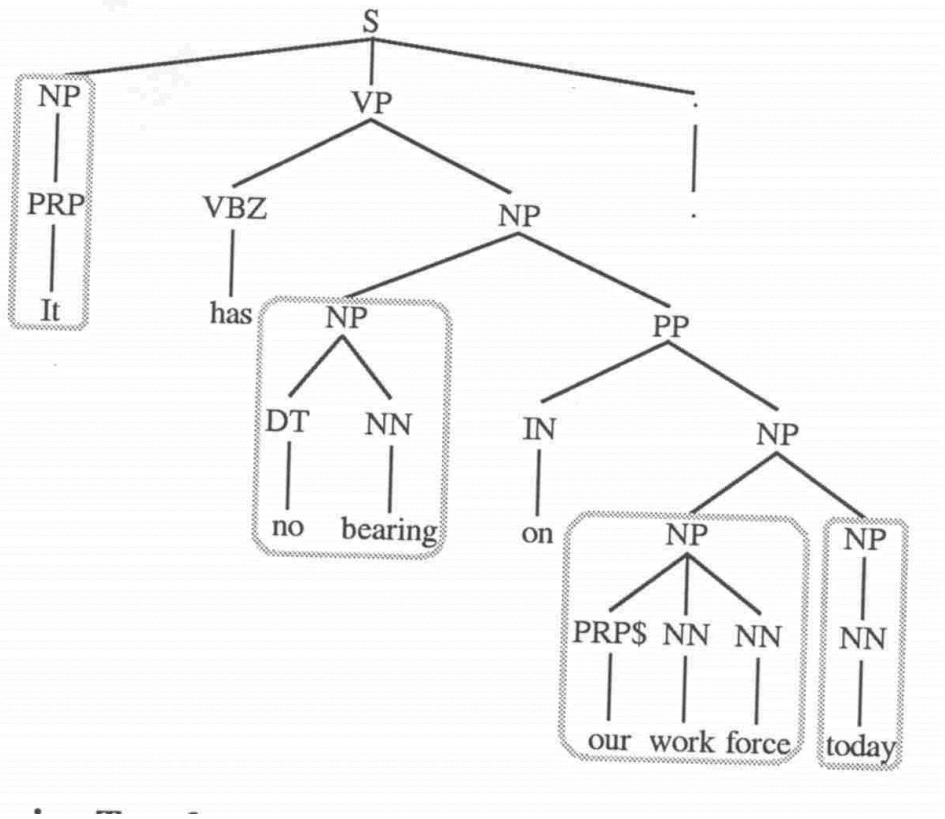


Fig. 1. A Chunking-and-Raising Scheme

aus der Penn Treebank gelernte Regeln:
(über 38.000 Sätze aus Wall Street Journal)

(*) PRP (VBZ) -> NP
(VBZ) DT NN (IN) -> NP
(IN) PRP$ NN NN (NN) -> NP
(NN) NN (.) -> NP



- Constrained Grammar (constraint = mit linkem und rechtem Kontext versehen)
- Regeln dem Baum entnommen, mit linkem und rechten Kontext
- und zwar auf verschiedenen Ebenen (levels)
- jede Regel mit Häufigkeit

Table 2. Constrained Rules for Levels 2, 3, 4, 5 and 6 Constrained Grammars

Level of Grammar	Constrained Rule
2	(IN) NP NP (.) -> NP
3	(NP) IN NP (.) -> PP
4	(VBZ) NP PP (.) -> NP
5	(NP) VBZ NP (.) -> VP
6	(*) NP VP . (*) -> S

ergab 33.800 Regeln = fast so viel wie Lern-Sätze

conflict rules - bei gleichen Tags und Kontext verschiedene Reduktion,
beruht auf inkonsistente Annotationen oder struktureller Ambiguität

Die weniger häufige Regel wird mit lexikalischer Information angereichert (wortnahe Disambiguierung); diese spezifischeren Regeln haben Vorrang.

Kritik

Häufigkeiten weichen nicht nur lexikalisch sondern auch stark je nach Position ab:

Expansion	% as Subj	% as Obj
NP → PRP	13.7%	2.1%
NP → NNP	3.5%	0.9%
NP → DT NN	5.6%	4.6%
NP → NN	1.4%	2.8%
NP → NP SBAR	0.5%	2.6%
NP → NP PP	5.6%	14.1%

Eigentlich sind es Semantik und Weltwissen, die Disambiguierung und schnelles Verstehen bewirken.

Sollte man statt Statistik lieber mehr Semantik hineinnehmen?

- Pipe Line Architektur: Syntax -> Semantik
- Integration: Zu jeder Syntaxregel eine semantische Regel, mit der ein Ausdruck eines Logikkalküls aufgebaut wird.

Ungewiss: trade-off des Aufwandes

Unifikation semantische Merkmale beim Parsen, insb. durch lexikalische Selektion in der DUG:

ein|stel|len <sw. V.; hat> **1. a)** *in etw. (als den dafür bestimmten Platz) stellen, einordnen*: die Bücher [in das Regal] e.; **b)** *an einem [dafür bestimmten] Platz vorübergehend, zeitweilig abstellen, unterstellen:* das Auto [in eine/einer Garage] e.; falsch (nicht an der richtigen Stelle) eingestellte Bücher. **2.** *in ein Arbeitsverhältnis nehmen. anstellen:* die Firma stellt vorläufig keine neuen Arbeitskräfte ein. **3. a)** *(ein technisches Gerät o. ä. in bestimmter Weise stellen, regulieren:* ein Fernglas scharf, eine Kamera auf die richtige Entfernung, den Zeiger auf eine Marke e.; das Radio, den Fernsehapparat leiser, schärfer, auf Zimmerlautstärke, auf einen bestimmten Sender e.; **b)** *(bei einem technischen Gerät) durch Betätigen der Armaturen o. ä. etw. Bestimmtes regulieren od. zum Arbeiten bringen:* die Lautstärke, die Entfernung, einen bestimmten Sender e.; **c)** *justieren:* die Scheinwerfer, die Zündung [neu) e. **4.** *[vorübergehend] nicht fortsetzen, mit einer Tätigkeit o. ä. aufhören:* die Produktion, Zahlungen, ein Gerichtsverfahren e.; der Feind stellte das Feuer ein. **5.** (e. + sich**) a)** *zu bestimmter Zeit an einen bestimmten Ort kommen:* ich stellte mich pünktlich bei ihm ein; **b)** *(als Folge von etw.) eintreten:* starke Schmerzen stellten sich ein; Zweifel stellten sich bei uns ein. **6.** (e. + sich) **a)** *sich innerlich od. durch entsprechendes Verhalten, durch bestimmte Maßnahmen auf etw. vorbereiten:* sich, auf jmds. Besuch, auf die neue Situation e.; (Duden. Deutsches Universalwörterbuch)	einstellen +subjekt *jemand* +transobjekt %Lokal, %richtung einstellen +subjekt *firma* +transobjekt *arbeitskraft* einstellen +subjekt *jemand* +transobjekt *gerät* einstellen +subjekt *jemand* +transobjekt *lautstärke, sender* einstellen +subjekt *jemand* +transobjekt *tätigkeit* einstellen +subjekt *jemand* +reflexiv +präpobjekt *bei* einstellen +subjekt *jemand* +reflexiv +präpobjekt *auf*

14. Stunde: Komplexitätstheorie

Ziel: den Aufwand abschätzen, den das Parsen je nach Eingabe verursacht.

LIT: Barton, Berwick und Ristad "Computational Complexity and Natural Language" MIT Press 1987

Eine anwendungsorientierte Wissenschaft interessiert sich nicht nur dafür, ob ein **Problem überhaupt lösbar** ist, sondern auch für den **Aufwand,** der dafür notwendig ist.

Problem im Falle eines Parsers: Erkennen, ob eine bestimmte Zeichenkette w zu einer von einer Grammatik G erzeugten Sprache L gehört, und wenn ja, welche Strukturbeschreibung S der Kette w gemäß G zuzuordnen ist.

Der Aufwand bemisst sich in **Rechenzeit und Speicherplatz**.

Die Komplexitätstheorie versucht bei der Abschätzung des Aufwandes **von konkreter Hardware und konkreten Algorithmen zu abstrahieren**, da diese kaum vergleichbar sind. Es kommen immer schnellere Computer auf den Markt. Und es gibt Programmierer, die mehr oder weniger geschickt sind. Im Zentrum der Komplexitätstheorie steht daher die **Schwierigkeit einer Aufgabe als solcher**. Rechenzeit und Speicherplatz sind dabei theoretische Größen, nicht tatsächliche CPU-Sekunden und Speicher-Bytes.

- Es geht um die **Größe des Problems.**

Theorie der formalen Sprachen

Generative Kapazität als Maß für den Aufwand?

Formale Definition einer *Sprache* L = die **Menge der Zeichenketten,** die in ihr gebildet werden können.

V sei die Menge der elementaren Zeichenketten

V*(das freie Monoid von V) ist die Menge aller Ketten, die sich aus dem Vokabular durch beliebige Kombination bilden lassen

Jede Teilmenge des freien Monoids über dem Vokabular ist eine Sprache im formalen Sinn.

Eine Grammatik G legt diejenige Teilmenge von V* fest, die genau die Menge der Zeichenketten der Sprache L enthält.

G ist üblicherweise ein **Ersetzungs- oder Produktionssystem <T, NT, R, S>**

Ausgehend vom Anfangssymbol, werden durch wiederholte Anwendung der Ersetzungsregeln sämtliche Zeichenketten der Sprache, und nur die, erzeugt.

Ersetzungssysteme bieten eine Handhabe, verschiedene **Typen von Sprachen** zu unterscheiden:

Typ 0 : allgemeine Regelsprachen

(nicht entscheidbare Sprachen)

Typ1: kontextsensitive Sprachen

a^i b^i c^i

{abc, aabbcc, aaabbbccc ...}

Typ2: Kontextfreie Sprachen

a^i b c^i

{abc, aabcc, aaabccc ...}

Typ3: reguläre Sprachen

a^i

{a, aa, aaa ...}

Mit regulären Grammatiken kann man Zeichenketten der Art i mal a generieren, aber nicht i mal a, i mal b. Kontext-freie Grammatiken können Zeichenketten der Art i mal a, b , i mal c generieren, aber nicht i mal a, i mal b, i mal c. Letzteres ist möglich mithilfe einer kontext-sensitiven Grammatik. Die uneingeschränkte Form der Ersetzungsregeln kann prinzipiell alle Sprachen generieren, aber ein Teil dieser Sprachen sind nicht-entscheidbar.

regulär:
A -> a
A -> A a

kontextfrei
S -> a S c
S -> b

kontext-sensitiv
S -> a SBC
S -> ab C
CB -> BC
aB -> ab
b B -> bb
C -> c

Chomsky Hierarchie

Klassifizierung	Grammatik:	Sprache:	Automat:	Netzwerke:
Typ-3 (reguläre)	einseitig lineare Grammatik	reguläre Sprache	endlicher Automat	finites Übergangsnetzwerk
Typ-2	kontextfreie Grammatik	kontextfreie Sprache	Keller-automat	rekursives Übergangsnetzwerk
Typ-1	kontextsensitive Grammatik	kontextsensitive Sprache	linear beschränkter	erweitertes Übergangsnetzwerk
Typ-0	allgemeine Regelgrammatik	allgemeine Regelsprache	Touring-maschine	erweitertes Übergangsnetzwerk

Da die Automaten unterschiedlich viele Zustände durchlaufen und daher verschieden lange für das Erkennen einer Zeichenkette brauchen, ist die **generative Kapazität oft als Maß** für den Aufwand angesehen worden, der z.B. zum Parsen der betreffenden Sprache notwendig ist.

Das ist aber zu grob, da wir ja auch für dieselbe G unterschiedlich effiziente Parser kennengelernt haben.

Komplexitätsmaß

$c * g(n)$

n sei die **Größe eines Problems**, z.B. die Zahl der Wörter in einem Satz, der analysiert werden soll.

c ist eine **Konstante**, z.B. der Zeit für das Laden des Lexikons und der Grammatik, g(n) ist eine Funktion des Aufwandes für von n abhängende Faktoren, z.B. der Anzahl von Vergleichsoperationen zwischen Wörtern und Regeln.

$f(n) = O(g(n))$

Man sagt, der Aufwand f zur Lösung eines Problems der Größe n ist von der Ordnung g(n), wenn gilt, dass der Aufwand f im schlimmsten Fall gleich der Konstanten c mal g(n) ist.

z.B. $n^3 + n^2 + 76$, - der Aufwand ist von der Ordnung $O(n^3)$

In der Praxis gelten Probleme als effizient lösbar, **deren Aufwand polynomisch von n** abhängt, also z.B. wenn g(n) gleich n, Logarithmus n, n mal Logerithmus n, oder n hoch drei ist, höchstens jedoch n hoch j für irgendein j , wobei natürlich j durchaus ins Gewicht fällt.

$O(n^j)$

Als nicht effizient lösbar müssen Probleme angesehen werden, bei denen der Aufwand zu ihrer Lösung **exponentiell** steigt, also z.B. g gleich j hoch n für irgendein j.

$O(j^n)$

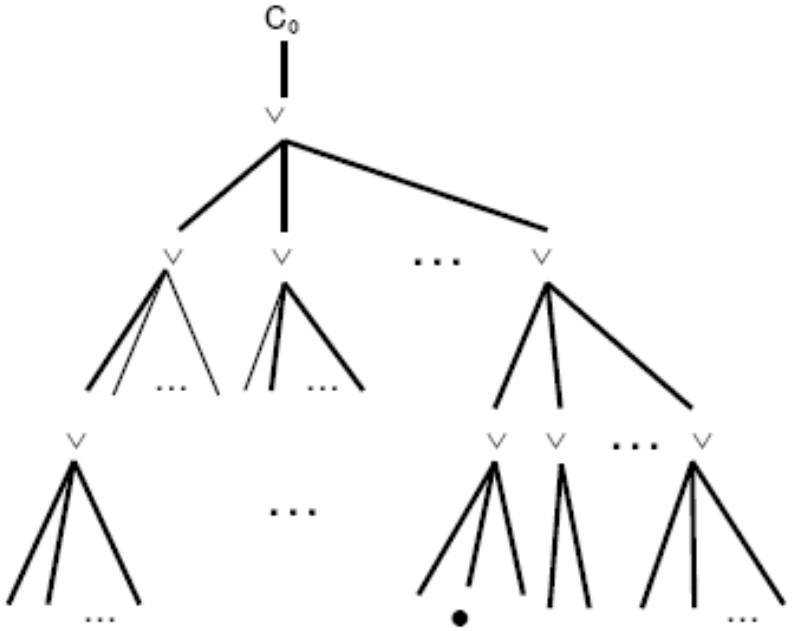
Angenommen, ein Rechenschritt dauert eine Mikrosekunde. Dann benötigt man bei

	Problemgrösse n		
Komplexität	10	50	100
n^3	.001 second	.125 second	1.0 second
2^n	.001 second	35,7 years	10^{15} centuries

Es ist bekannt, dass es für **reguläre Sprachen** Algorithmen gibt, deren Zeitaufwand **linear** zur Zahl der Eingabewörter steigt. Für **kontext-freie** Grammatiken liegt das Optimum bislang bei **n hoch 3**. Für den Early-Algoritmus wird z.B. im schlimmsten Fall ein Aufwand gleich dem **Umfang der Grammatik hoch 2 mal n hoch 3** angenommen, wobei der Aufwand automatisch sinkt, wenn die Grammatik von eingeschränkterem Typ ist.

Berwick zeigt jedoch, dass die **generative Kapazität als Komplexitätsmaß nicht ausreicht**. Die **Größe der Grammatik** kann z.B. sehr ins Gewicht fallen. Es wird z.B. damit argumentiert, dass die Generalized Phrase Structure Grammar (GPSG) im Prinzip in eine kontextfreie Grammatik überführt werden kann und daher für sie der Aufwand des Early-Algorithmusses zu veranschlagen sei. Dabei bleibt unberücksichtigt, dass unter Umständen die Größe der **Zielgrammatik exponentiell zum Umfang der Ausgangsgrammatik** steigt. Damit ist eine effiziente Verarbeitbarkeit natürlich nicht mehr gegeben.

Es ist notwendig, dass wir uns mehr **an den Zuständen orientieren, welche ein Automat einnehmen muss, um ein Problem zu lösen**.

Deterministische Turingmaschine	Nicht-deterministische Turingmaschine
	
$(\sim y \vee z) \wedge (x \vee y) \wedge \sim x$	$(x \vee \sim y \vee z) \wedge (y \vee z \vee u) \wedge (x \vee z \vee \sim u) \wedge (\sim x \vee y \vee u)$

Gradlinige Reihe von Folgerungen:

$\sim x$

$(x \vee y) \rightarrow y$

$(\sim y \vee z) \rightarrow z$

SAT(satisfiable) -Problem

kein gradliniger Weg, sondern alle Belegungen ausprobieren, d.s. 2 hoch n Wahrheitswertekombinationen wobei n = Zahl der Variablen

P-komplex - von deterministischem Automaten in polynomischer Zeit zu lösen

NP-komplex = exponentieller Aufwand

Die Methode der Komplexitätsabschätzung nach Berwick besteht nun darin, ein gegebenes Problem durch einen Algorithmus in polynomischer Zeit **auf ein Problem zurückzuführen, dessen Komplexitätsklasse bereits bekannt ist, also z.B. auf das SAT-Problem.** Im Prinzip ist jedes Entscheidungsproblem auf das SAT-Problem zurückführbar.

Schlußfolgerungen, die Berwick aus der Tatsache zieht, dass manche gängigen Grammatikformalismen offenbar NP-komplex sind:

Ersetzungssysteme | A -> B1 + B2 + ... + Bn | **ähneln verdächtig dem SAT-Problem.**

NP-komplex sind solche Probleme, die - wie das SAT-Problem - eine **kombinatorische Suche** erforderlich machen. Wenn ein solches Problem auftritt, besteht immer der Verdacht, **dass irgendetwas etwas fehlt.** Ein kombinatorischer Suchalgorithmus impliziert nämlich, **dass das Problem keine spezifische Struktur hat**, die man bei der Lösung ausnutzen könnte.

NP-komplexe Probleme sind nach Berwick **unnatürlich schwere Probleme.** Natürliche Probleme haben typischerweise eine mehr modulare und lokale Struktur. **Ein theoretisch NP-komplexer Grammatikformalismus sollte so revidiert werden**, dass er nicht mehr erlaubt, Probleme zu formulieren, die nicht effizient lösbar sind.

Da in der Wirklichkeit die Aufgabe, ob eine Zeichenkette zu einer Sprache gehört und welche Struktur sie hat, von Menschen effizient gelöst wird, heiß dies, dass der Grammatikformalismus mehr mit der **linguistischen Realität** in Übereinstimmung gebracht werden muss.

Zitat Berwick:

"Difficulties in processing a formalism do indicate that the formalism itself does not tell the whole story."

Die ausschließliche Orientierung an der gängigen Theorie der formalen Sprachen, wie sie in der Informatik gelehrt wird, macht unter Umständen blind für die wahre Natur der sprachlichen Phänomene.

Eine andere Möglichkeit ein formales System aufzubauen:

Beispiel Chemie. Sie geht von den kleinsten Bausteinen aus, nämlich von den Atomen und ihrer Verbindbarkeit aufgrund der freien Elektronen in ihrer Hülle. Nehmen wir sozusagen als Vokabular eine Menge von Wasserstoff-, Kohlenstoff- und Sauerstoffatomen. Jedes dieser Atome hat entsprechend seiner freien Elektronen unterschiedliche Wertigkeiten, nämlich 1, 4 und 2.

Indem festgelegt wird, dass bei jeder Verbindung je ein Wert des einen Atoms einem Wert eines anderen Atoms entsprechen muss und kein freies Elektron übrig bleiben darf, ist **die Menge aller möglichen Moleküle definiert** und die tatsächlich existierenden Moleküle sind als Teilmenge dieser Menge bestimmbar. In der Chemie werden die wohlgeformten Verbindungen mit Hilfe der bekannten Strukturformeln dargestellt. Aus den Zeichen für Wasserstoff, Kohlenstoff und Sauerstoff lassen sich z.B. die hier gezeigten Strukturformeln für Alkoholmoleküle bilden.

Genauso liegt der Ausweg aus der Komplexitätsfalle in der Computerlinguistik in der Lexikalisierung:

	regel-basierte Grammatiken	**lexikalisierte Grammatiken**
Konstituentenstruktur	PSG ATN	Kategorialgrammatik
Dependenzstruktur	Dependenzgrammatik (Gaifmantyp)	Valenzgrammatik (DUG)

15. Stunde: Parserevaluierung

Unterschiedliche Bauteile

Strukturdarstellung, Beschreibungsart:

- Konstituentenstruktur
- Dependenzstruktur

Spezifikation der Grammatik, Formalismus:

- Grammatik mit Ableitungsregeln
- Übergangsnetzwerke versch. Typs
- Lexikalisierte Grammatik (Kategorialgrammatik oder Frames/Slots)

Verhältnis Grammatik - Parser:

- Grammatik getrennt vom Programm (interpretierender Parser)
- Grammatik im Programm (prozedurale Grammatik)
- vorhergehende Compilation der Grammatik in eine interne Repräsentation

Erkennungstrategien:

- Kategorieexpansion, Top-down Strategie
- Kategoriereduktion. Bottom-up Strategie,
- Übergang zwischen Zuständen in Netzen,
- Slot Filling

Vorrücken:

- strikt von links nach rechts, Tiefe zuerst,
- mehrere Durchläufe auf versch. Ebenen, Breite zuerst
- sonstige

Kontrolle der grammatischen Alternativen und des Ergebnisses:

- Backtracking, schematisch mit Stapelspeicher oder gezielt
- Parallelverarbeitung
- Vorausschau (nächstes Terminal, nächste Konstituente)
- Teilergebnistabelle (Chart)
- Aktions- und Goto-Tabellen

Generierung des Ergebnisses:

- laufend
- nachträglich durch nochmalige Anwendung der Regeln

Evaluierung

Effizienz :

- bei der Kontrolle der grammatischen Alternativen
  - deterministisch oder nicht, wie viele Schritte, wie oft Rücksetzen?
- jedes Zwischenergebnis zur einmal oder evtl. mehrfach erzeugt?
- Übergenerierung (Erzeugen obsoleter Zwischenergebnisse
  - Erzeugung desselben Ergebnisses auf verschiedene Weise)?
- erwartungsgesteuert und/oder datengesteuert zur Vermeidung von Übergenerierung
- links assoziativ

Für welchen Typ von Grammatik:

- nur für reguläre Grammatiken,
- nur für kontextfreie Grammatiken,
- für kontextsensitive Grammatiken
- linksrekursive Regeln erlaubt?
- Tilgung (Epsilonregel) erlaubt?
- bestimmte Normalform notwendig?

Welche Phänomene beherrscht:

einfache syntaktische Hierarchien,	
unbeschränkte Kongruenzen,	*die Damen, die sich im Park trafen*
long-distance dependencies,	*what did Gudrun tell you she feeds her cat?*
diskontinuierliche Konstituenten,	*ausgerechnet auf dem Weg hatte er angehalten, der zur Grenze führt*
Koordination, Ellipsen	*Claudia wollte ein Eis und Emil eine Bratwurst*

Aufwand bei der Erstellung und dem Testen der Daten:

Transparenz und relative Einfachheit der grammatischen Beschreibung

Nähe zu gedruckten Ressourcen

wenig Seiteneffekte bei Erweiterung der Daten