Einführung in die Computerlinguistik

Parsing

Dozentin: Wiebke Petersen

WS 2004/2005

1 Nachtrag zu den Abschlußeigenschaften kontextfreier Sprachen

Kontextfreie Sprachen sind abgeschlossen bezüglich der Vereinigung von Mengen: Seien $G_1 = (N_1, T_1, S_1, P_1)$ und $G_2 = (N_2, T_2, S_2, P_2)$ zwei kontextfreie Grammatiken, die die Sprachen $L(G_1)$ und $L(G_2)$ generieren, dann ist $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ mit $P = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$ die Grammatik, die $L(G_1) \cup L(G_2)$ generiert ($N_1 \uplus N_2$ ist die disjunkte Vereinigung von N_1 und N_2).

Beispiel:

$G_1 = \langle\{\mathrm{S}_1, \mathrm{A}\}, \{a, b\}, \mathrm{S}_1, P_1\rangle$	$G_2 = \langle\{\mathrm{S}_2, \mathrm{A}\}, \{a, b\}, \mathrm{S}_2, P_2\rangle$
$P_1 = \left\{ \begin{array}{lcl} \mathrm{S}_1 & \to & \mathrm{A}\mathrm{S}_1 b \\ \mathrm{S}_1 & \to & \epsilon \\ \mathrm{A} & \to & aa \end{array} \right\}$	$P_2 = \left\{ \begin{array}{lcl} \mathrm{S}_2 & \to & b\mathrm{S}_2\mathrm{A} \\ \mathrm{S}_2 & \to & \epsilon \\ \mathrm{A} & \to & \mathrm{A}a \\ \mathrm{A} & \to & a \end{array} \right\}$
$L(G_1) = a^{2n}b^n$	$L(G_2) = b^n a^m$ mit $m \geq n$

Dann ist G eine Grammatik, die $L(G_1) \cup L(G_2)$ generiert:

$$G = \langle\{\mathrm{S}, \mathrm{A}_1, \mathrm{A}_2, \mathrm{S}_1, \mathrm{S}_2\}, \{a, b\}, \mathrm{S}, P\rangle \; P = \left\{ \begin{array}{llllll} \mathrm{S} & \to \mathrm{S}_1 & \mathrm{S} & \to \mathrm{S}_2 & & \\ \mathrm{S}_1 & \to & \mathrm{A}_1\mathrm{S}_1 b & \mathrm{S}_1 & \to & \epsilon \\ \mathrm{A}_1 & \to & aa & & & \\ \mathrm{S}_2 & \to & b\mathrm{S}_2\mathrm{A}_2 & \mathrm{S}_2 & \to & \epsilon \\ \mathrm{A}_2 & \to & \mathrm{A}_2 a & \mathrm{A}_2 & \to & a \end{array} \right\}$$

2 Kompositionalität kontextfreier Grammatiken

Vorsicht, auch wenn kontextfreie Sprachen bezüglich der Vereinigung von Mengen abgeschlossen sind, heißt das nicht, daß man Grammatiken ohne weiteres modular aufbauen und dann durch "Vereinigung" zu einer größeren zusammenfügen kann.

$G_1 = \langle\{\mathrm{S}, \mathrm{V}, \mathrm{VP}, \mathrm{EN}\}, \{\mathrm{Mary}, \mathrm{sings}\}, \mathrm{S}, P_1\rangle$

$$P_1 = \left\{ \begin{array}{lcl} \mathrm{S} & \to & \mathrm{EN\ VP} \\ \mathrm{VP} & \to & \mathrm{V} \\ \mathrm{V} & \to & \mathrm{sings} \\ \mathrm{EN} & \to & \mathrm{Mary} \end{array} \right\}$$

$G_2 = \langle\{\mathrm{S}, \mathrm{V}, \mathrm{VP}, \mathrm{EN}\}, \{\mathrm{Mary}, \mathrm{John}, \mathrm{loves}\}, \mathrm{S}, P_1\rangle$

$$P_2 = \left\{ \begin{array}{lcl} \mathrm{S} & \to & \mathrm{EN\ VP} \\ \mathrm{VP} & \to & \mathrm{V\ EN} \\ \mathrm{V} & \to & \mathrm{loves} \\ \mathrm{EN} & \to & \mathrm{Mary} \\ \mathrm{EN} & \to & \mathrm{John} \end{array} \right\}$$

3 Parsing

- *to parse* (grammatisch zerlegen) abgeleitet von *pars* (griechisch) Teil

- Ein Parser ist ein Automat, der einer Zeichenkette aufgrund einer Grammatik einen Derivationsbaum zuordnet.

$$\begin{array}{c}\text{Grammatik}\\ + \\ \text{Zeichenkette}\end{array} \longrightarrow \text{Derivationsbaum}$$

4 Unterschied Recognizer – Parser

Beides sind Automaten

Recognizer: stellt ausschließlich fest, ob eine Zeichenfolge ein Wort der von der Grammatik generierten Sprache ist oder nicht (Kellerautomat).

Parser: erstellt den Derivationsbaum einer Zeichenfolge im Bezug auf die Grammatik.

5 Parsingstrategien

Parsingstrategien unterscheiden sich darin, in welcher Reihenfolge die Knoten eines Derivationsbaums erstellt werden.

Man unterscheidet zwei Hauptstrategien voneinander

- **inputgetriebenes** Parsing (**bottom up**)
- **theoriegetriebenes** Parsing (**top down**)

Zusätzlich charakterisiert man Parsingstrategien mit folgenden Begriffen:

depth-first $\leftrightarrow$ breadth-first

left-to-right $\leftrightarrow$ right-to-left

6 top-down, left-to-right, depth-first Parser

top-down:

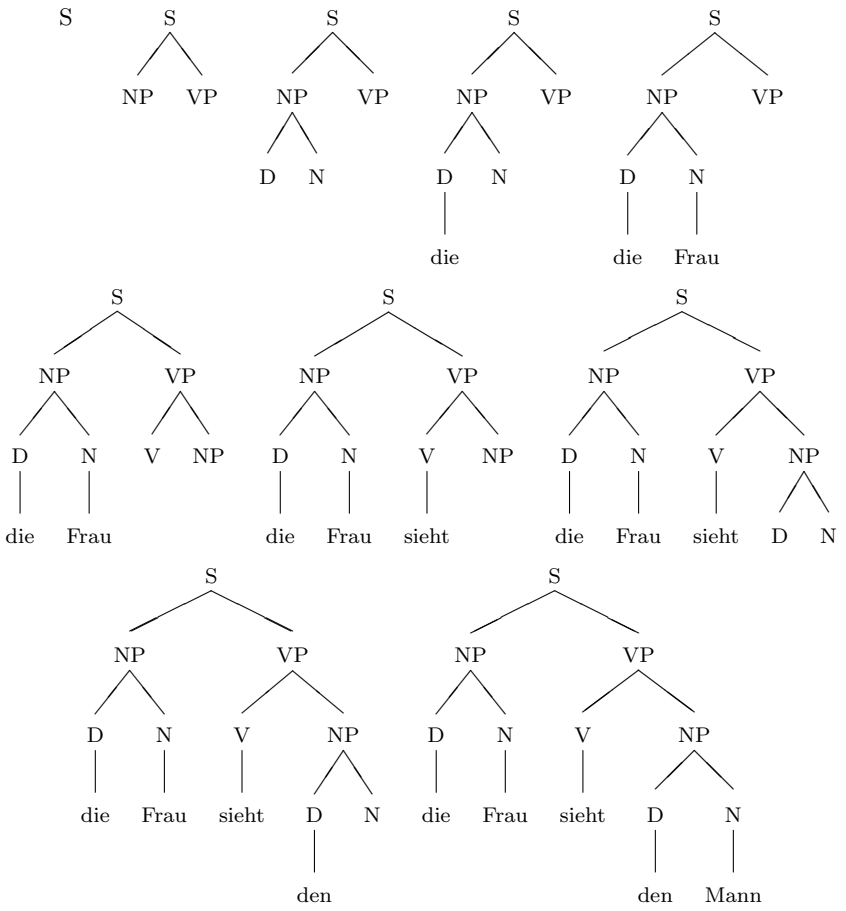
- Parser **beginnt beim Startsymbol** S und versucht, durch sukzessive Regelanwendung schließlich bei der Eingabekette zu landen.
- Regelanwendungen (von links nach rechts) nennt man **Expansion**.
- Das Einlesen eines Elements der Eingabekette nennt man **Scan**.

left-to-right: Der Parser versucht immer den am weitesten links stehenden Knoten des Ableitungsbaums zu expandieren oder mit diesem Knoten einen Scan durchzuführen.

depth-first: Der Parser versucht immer die am weitesten unten stehenden Knoten (das sind immer die zuletzt gebildeten) weiter zu expandieren oder hier einen Scan durchzuführen.

7 Beispiel:

$$P = \left\{ \begin{array}{llllllllll} \text{S} & \rightarrow & \text{NP VP} & \text{VP} & \rightarrow & \text{V NP} & \text{NP} & \rightarrow & \text{D N} \\ \text{D} & \rightarrow & \text{die} & \text{D} & \rightarrow & \text{den} & & & \\ \text{N} & \rightarrow & \text{Frau} & \text{N} & \rightarrow & \text{Mann} & \text{V} & \rightarrow & \text{sieht} \end{array} \right\}$$



8 Linksrekursion

Top-down, left-to-right, depth-first Parser terminieren nicht bei Grammatiken, die linksrekursive Regeln beinhalten!

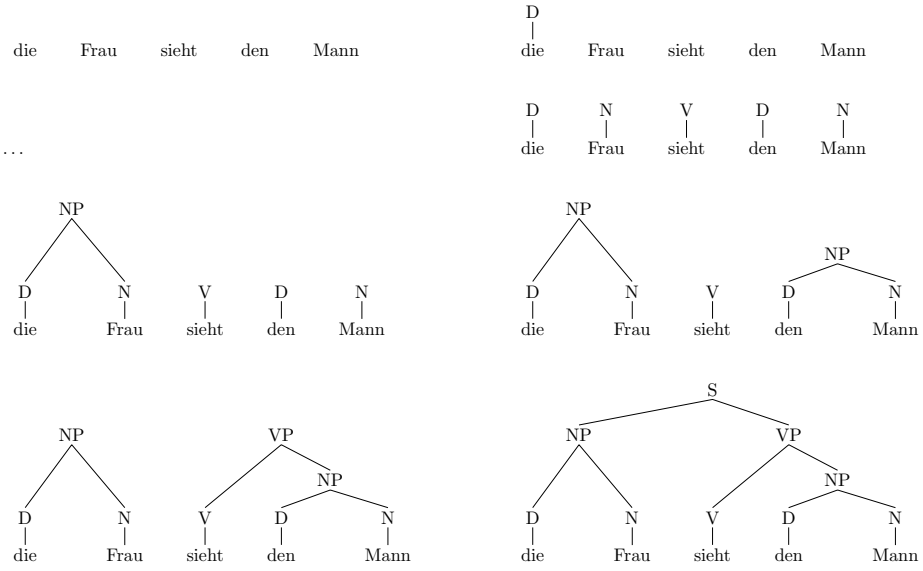
$$\begin{array}{rcl} \text{S} & \rightarrow & \text{S und S} \\ \text{NP} & \rightarrow & \text{NP PP} \\ \text{T} & \rightarrow & \text{T O T} \end{array}$$

9 bottom-up, breadth-first, left-to-right Parser

bottom-up:

- Parser **beginnt bei der Eingabekette** und versucht, durch sukzessives **rückwärtiges** Anwenden der Regeln (von rechts nach links) schließlich bei dem Startsymbol S zu landen.

10 Beispiel:



11 ϵ-Regeln

Bottom-up, breadth-first, left-to-right Parser terminieren nicht bei Grammatiken, die ϵ-Regeln beinhalten, da eine solche Regel jederzeit anwendbar ist!

$$S \rightarrow \epsilon$$

(B.2) Backtracking

- Verschiedenen Aktionsmöglichkeiten des Parsers $\Rightarrow$ Entscheidung

- Entscheidung kann falsch gewesen sein, d.h.
  - top-down: gewählte Regel führt nicht zur Ableitung der Eingabe
  - bottom-up: Eingabesatz kann nicht auf das Startsymbol reduziert werden

- Falsche Entscheidung kann sich erst zu späterem Zeitpunkt bemerkbar machen

- Fehlerbehebung nach dem Prinzip des **Backtracking**: Alle Schritte, bis zu der letzten Stelle, an der eine Wahlmöglichkeit bestand, rückgängig machen und dort eine andere Wahl treffen

(B.2) Backtracking: Beispiel

Ableitung des Satzes *der Hund sieht die Katze* (top-down, links-nach-rechts, Tiefe-Zuerst mit Backtracking):

Nr.	Satzform	Eingabe	Schritt
1	S	*der Hund sieht die Katze*	
2	NP VP	*der Hund sieht die Katze*	E
3	Det N VP	*der Hund sieht die Katze*	E
4	*der* N VP	*der Hund sieht die Katze*	E
5	N VP	*Hund sieht die Katze*	S
6	*Hund* VP	*Hund sieht die Katze*	E
7	VP	*sieht die Katze*	S
8	V	*sieht die Katze*	E
9	*bellt*	*sieht die Katze*	E
$8'$	V	*sieht die Katze*	B zu 8
$9'$	*sieht*	*sieht die Katze*	E
$7''$	VP	*sieht die Katze*	B zu 7
$8''$	V NP	*sieht die Katze*	E
$9''$	*bellt* NP	*sieht die Katze*	E
$8'''$	V NP	*sieht die Katze*	B zu $8''$
$9'''$	*sieht* NP	*sieht die Katze*	E
$10'''$	NP	*die Katze*	S
$11'''$	Det N	*die Katze*	E
$12'''$	*der* N	*die Katze*	E
$11''''$	Det N	*die Katze*	B zu $11'''$
$12''''$	*die* N	*die Katze*	E
$13''''$	N	*Katze*	S
$14''''$	*Hund*	*Katze*	E
$13'''''$	N	*Katze*	B zu $13''''$
$14'''''$	*Katze*	*Katze*	E
$15'''''$			S, akzept.

(E = Expandieren, S = Scannen, B = Backtracken)

Universität zu Köln
Institut für Sprach- und Informationsverarbeitung

Proseminar Computerlinguistik
Sommersemester 2004
Jürgen Hermes

Referentin: Mara Keune
Datum: 05.05.2004

Bottom-up-Parsing

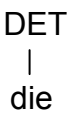
- Bottom-up: eine Analyserichtung (Gegenteil: Top-down)
- Analyse von unten nach oben
- Startzustand: der zu analysierende Satz, bestehend aus terminalen (lexikalische) Kategorien
- ***datengesteuerte Verarbeitung***: Satz wird mit Hilfe von Syntaxregeln und Lexikoneinträgen in einem oder mehreren Schritten auf das Startsymbol (S) zurückgeführt
- ***Reduktion*** ausgehend vom Wort: die Regeln der Syntax werden ***von rechts nach links*** angewendet, die Symbole der rechten Seite werden durch das Symbol der linken Seite ersetzt
  Beispiel: S → NP VP
  Reduktion [NP VP] → [S]
- Zielzustand: Startsymbol (S)

Einfaches Beispiel: Die Sonne scheint.

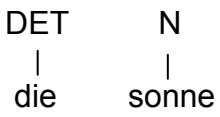
Syntaxregeln:
S → NP VP
VP → V
NP → DET N

Lexikon:
die ∈ DET
sonne ∈ N
scheint ∈ V

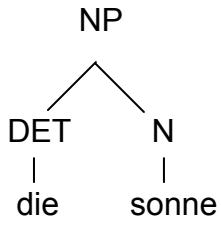
(1)



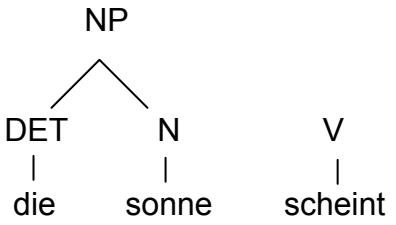
(2)



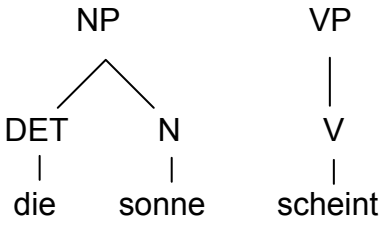
(3)



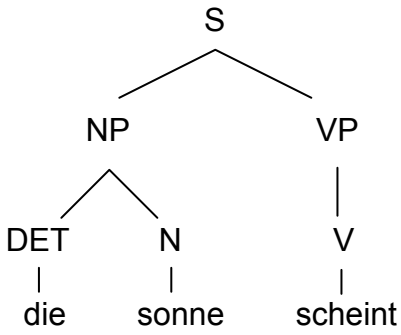
(4)



(5)



(6)



Unterschied Recognizer – Parser

Sind beide Automaten.

Recognizer: stellt ausschließlich fest, ob Satz in der Sprache möglich oder nicht möglich ist
Satz in der Sprache möglich → Ausgabe: TRUE
Satz in der Sprache nicht möglich → Ausgabe: FALSE

Parser: erzeugt zusätzlich Strukturbeschreibungen

Deterministischer Shift-Reduce-Recognizer

Verwendung von zwei Stacks:

1. Stack: noch nicht analysierter Satz(abschnitt)
2. Stack: (Zwischen)Ergebnisse der Analyse

Operationen:

- ***Shift***: Wort wird von dem ersten auf den zweiten Stack übertragen, nur wenn Reduce nicht möglich
- ***Reduce***: Inhalt des zweiten Stacks wird mit Hilfe einer Syntaxregel oder eines Lexikoneintrags reduziert

Algorithmus

Wenn <Stack 2 nicht leer und Regel anwendbar>
Dann <*Reduce* (Stack 2)>
Sonst
    Wenn <Stack 1 nicht leer>
    Dann <*Shift* (Stack 1, Stack 2)
    Sonst
        Wenn <Stack 2 == S>
        Dann <RETURN (True)>
        Sonst <RETURN (False)>

Beispiel Shift-Reduce-Recognizer: Die Sonne scheint.

Syntaxregeln:

S → NP VP	NP → DET N	VP → V

Lexikon:

die ∈ DET	sonne ∈ N	scheint∈V

Operation	**Stack 1**	**Stack 2**
Anfangszustand	[die, Sonne, scheint]	[ ]
Shift	[sonne, scheint]	[die]
Reduce	[sonne, scheint]	[DET]
Shift	[scheint]	[sonne, DET]
Reduce	[scheint]	[N, DET]
Reduce	[scheint]	[NP]
Shift	[ ]	[scheint, NP]
Reduce	[ ]	[V, NP]
Reduce	[ ]	[VP, NP]
Reduce	[ ]	[S]

TRUE

Backtracking

Problem: Ambiguitäten

1. wenn verwendete Grammatik nicht deterministisch ist
   Beispiel: VP → V
   VP → V NP
2. wenn Lexikon Einträge enthält, die einem Wort verschiedene lexikalische Kategorien zuordnen
   Beispiel: antworten ∈ V
   antworten ∈ N

➔Algorithmus, der den richtigen Pfad auswählt wird benötigt

Problemlösung: Backtracking (Rücksetzen)

- ein Pfad wird so lange verfolgt, bis er sich als falsch erweist
- die Verarbeitungsschritte werden bis zu der zuletzt getroffenen Entscheidung für einen bestimmten Pfad zurückgesetzt
- ein Alternativpfad wird gewählt
- jeder Syntaxregel und jedem Lexikoneintrag wird einer Zahl zugeordnet (Regel- bzw. Eintragsindex)
- ein 3. Stack wird implementiert, der Regel- und Eintragsindizes bzw. die Zahl 0 als Markierung für die Shift-Operation enthält

Shift-Reduce-Recognizer mit Backtracking

Backtracking ist erforderlich, wenn in einer Situation weder Reduce noch Shift möglich ist und der Endzustand (S) noch nicht erreicht wurde.

Verschiedene Backtracking-Prozeduren

- **nach Rücknahme einer Reduktion**
  Situation 1: es gibt eine noch nicht geprüfte Reduktionsmöglichkeit
  → andere Reduktionsmöglichkeit wird vorgenommen
  Situation 2: es gibt keine noch nicht geprüfte Reduktionsmöglichkeit, aber ein neues Element kann geshiftet werden (Stack 1 enthält mind. 1 Element)
  → neues Element wird geshiftet
  Situation 3: es gibt keine weitere Reduktionsmöglichkeit und keinen weiteren möglichen Shift
  → die letzte Reduktion wird zurückgenommen
  → mindestens ein weiteres Backtracking
- **nach Rücknahme eines Shifts:**
  Situation 4: letzte Operation war ein Shift
  → Shift-Operation wird zurückgenommen
  → mindestens ein weiteres Backtracking

Beispiel Shift-Reduce-Recognizer mit Backtracking:
Computer erzeugen Antworten.

Syntaxregeln:
1: S → NP VP 2: VP → V 3: VP → V NP 4: NP → N

Lexikon:
5: V = {antworten, erzeugen} 6: N = {computer, antworten}

Operation	**Stack 1**	**Stack 2**	**Stack 3**
Start	[c, e, a]	[ ]	[ ]
Shift	[e, a]	[c]	[0]
Reduce	[e, a]	[N]	[6,0]
Reduce	[e, a]	[NP]	[4,6,0]
Shift	[a]	[e, NP]	[0,4,6,0]
Reduce	[a]	SA [V, NP]	[5,0,4,6,0]
Reduce	[a]	[VP, NP]	[2,5,0,4,6,0]
Reduce	[a]	[S]	[1,2,5,0,4,6,0]
Shift	[ ]	LA [a, S]	[0,1,2,5,0,4,6,0]
Reduce	[ ]	[V, S]	[5,0,1,2,5,0,4,6,0]
Reduce	[ ]	[VP, S]	[2,5,0,1,2,5,0,4,6,0]
Backtracking	[ ]	[a, S]	[0,1,2,5,0,4,6,0]
Reduce	[ ]	[N, S]	[6,0,1,2,5,0,4,6,0]
Reduce	[ ]	[NP, S]	[4,6,0,1,2,5,0,4,6,0]
Backtracking	[a]	[V, NP]	[5,0,4,6,0]
Shift	[ ]	LA [a, V, NP]	[0,5,0,4,6,0]
Reduce	[ ]	[V, V, NP]	[5,0,5,0,4,6,0]
Reduce	[ ]	[VP, V, NP]	[2,5,0,5,0,4,6,0]
Backtracking	[ ]	[a, V, NP]	[0,5,0,4,6,0]
Reduce	[ ]	[N, V, NP]	[6,0,5,0,4,6,0]
Reduce	[ ]	[NP, V, NP]	[4,6,0,5,0,4,6,0]
Reduce	[ ]	[VP, NP]	[3,4,6,0,5,0,4,6,0]
Reduce	[ ]	[S]	[1,3,4,6,0,5,0,4,6,0]

TRUE

: Ambiguitäten

SA = syntaktische Ambiguität
LA = lexikalische Ambiguität

Vom Recognizer zum Parser

Möglichkeiten der Strukturbeschreibung

1. Möglichkeit:
die angewandten Regeln werden anschließend in umgekehrter Reihenfolge zur Konstruktion eines Phrasenstrukturbaums benutzt

2. Möglichkeit:
Stack 2 baut die Struktur parallel zur Ableitung auf und enthält nicht nur eine Ableitungszeile, sondern eine Liste von Strukturbeschreibungen
Beispiel:

Operation	Stack 1	Stack 2	Stack 3
Start	[c, e, a]	[ ]	[ ]
Shift	[e, a]	[c]	[0]
Reduce	[e, a]	[N]	[6,0]
Reduce	[e, a]	[NP(N)]	[4,6,0]
		...	
		[S(VP(NP(N)V)NP(N))]	

Literatur

ALLEN, James 1995, *Natural Language Understanding*. Redwood City: Benjamin/Cummings, Kap. 3.

CARSTENSEN, Kai-Uwe et al. 2001, *Computerlinguistik und Sprachtechnologie*. Heidelberg/Berlin: Spektrum.

HELLWIG, Peter 1989, „Parsing natürlicher Sprache“, in: Bátori/Lenders/Putschke. *Computerlinguistik.* Handbücher zur Sprach- und Kommunikationswissenschaft. Band 4. Berlin: De Gruyter, S. 348-432.

NAUMANN, Sven & Hagen Langer 1994, *Parsing. Eine Einführung in die maschinelle Analyse natürlicher Sprache*. Stuttgart: Teubner.

(B.3) Probleme beim Parsen

- Top-Down: Linksrekursion;
  Bottom-Up: ϵ-Regeln

- Nichtdeterminismus

$\Rightarrow$ Echte und lokale Ambiguitäten müssen behandelt werden – z.B. durch Backtracking

$\Rightarrow$ Ineffiezientes Arbeiten durch wiederholte Konstituentenanlyse