Kapitel 8

# Algorithmen für kontextfreie Sprachen

Abschnitt 8.1

## Parsing

- Die Syntax von Programmiersprachen wie JAVA ist typischerweise durch kontextfreie Grammatiken spezifiziert.
- Ein Programm wird dem Compiler einfach als Textfile, oder abstrakter, als Wort über dem Terminalalphabet, gegeben.
- Der Compiler muss zunächst die Struktur des Programms rekonstruieren; dies geschieht, indem er einen Ableitungsbaum für das Programm konstruiert.
  Diesen Prozess nennt man Parsing und den Teil des Compilers, der ihn ausführt, Parser.
- Parser werden aber nicht nur in Compilern benötigt, sondern auch anderen Stellen, wo man strukturierte Information aus einem Textfile rekonstruieren muss.
  Beispielsweise benötigt eine Webbrowser einen HTML-Parser.



## Beispiel 8.1

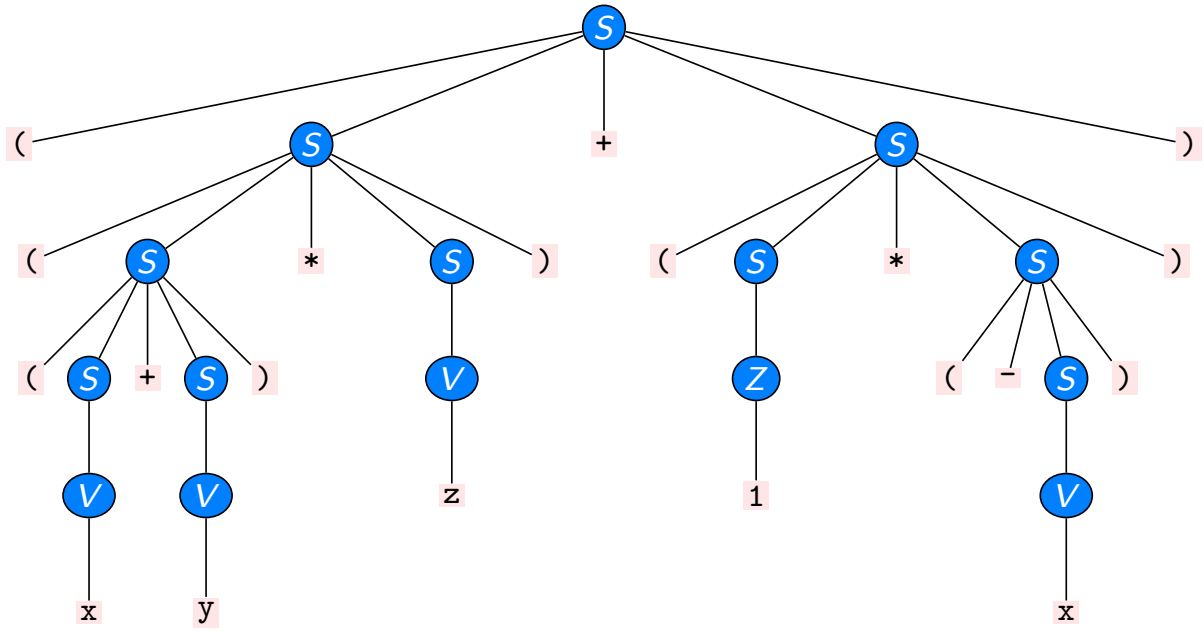
Wir betrachten die Sprache $L_T$ der arithmetischen Terme mit den Operatoren + und $*$ (zweistellig) und $-$ (einstellig), den Konstanten 0 und 1 und Variablen $\mathtt{x}, \mathtt{y}, \mathtt{z}$. Das Alphabet ist

$$\Sigma_T := \{+, *, -, 0, 1, \mathtt{x}, \mathtt{y}, \mathtt{z}, (, )\}.$$

Folgende kontextfreie Grammatik $\mathcal{G}_T$ erzeugt die Terme:

$$\begin{aligned} S &\to (S{+}S) \mid (S{*}S) \mid (-S) \mid Z \mid V \\ Z &\to 0 \mid 1 \\ V &\to \mathtt{x} \mid \mathtt{y} \mid \mathtt{z} \end{aligned}$$

Ableitungsbaum für das Wort $\texttt{(((x+y)*z)+(1*(-x)))} \in L_T$:




Die Grammatik $\mathcal{G}_T$ ist eindeutig, das heißt, jedes Wort besitzt einen eindeutigen Ableitungsbaum.



# Ein Parser für die Grammatik $\mathcal{G}_T$

PARSE(*text*)

```
 1: a ← first symbol of text
 2: initialise stack S
 3: while a ≠ end of file do
 4:     if a = ( or a = + or a = * or a = - then
 5:         PUSH(S, a)
 6:     else if a = 0 or a = 1 then
 7:         T ← NUMTREE(a); PUSH(S, T)
 8:     else if a = x or a = y or a = z then
 9:         T ← VARTREE(a); PUSH(S, T)
10:     else if a = ) then
11:         T ← REDUCE(S); PUSH(S, T)
12:     end if
13:     a ← next symbol of text
14: end while
15: T ← POP(S)
16: if T is a tree and S = ∅ then
17:     return T
18: end if
```

# Ein Parser für die Grammatik $G_T$ (Forts.)

NumTree($a$)

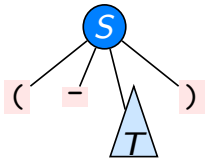
1: **return**


VarTree($a$)

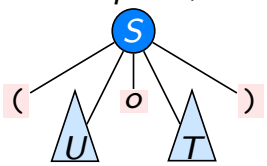
1: **return**


Reduce($\mathcal{S}$)

1: $T \leftarrow \text{Pop}(\mathcal{S})$
2: **if** $T$ is a tree **then**
3: $o \leftarrow \text{Pop}(\mathcal{S})$
4: **if** $o =$ - **then**
5: $p \leftarrow \text{Pop}(\mathcal{S})$
6: **if** $p =$ ( **then**
7: **return**




8: **end if**
9: **else if** $o =$ + or $o =$ * **then**
10: $U \leftarrow \text{Pop}(\mathcal{S})$; $p \leftarrow \text{Pop}(\mathcal{S})$
11: **if** $U$ is a tree and $p =$ ( **then**
12: **return**




13: **end if**
14: **end if**
15: **end if**

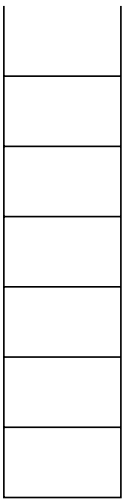


## Beispiel 8.2

### Eingabewort

```
( 1 * ( - x ) )
```

### Stack



### Ableitungsbaum




### Eingabewort

```
( 1 * ( - x ) )
```

## Beobachtung 8.3

*Unser Parser für die Grammatik $\mathcal{G}_T$ arbeitet im Prinzip wie ein deterministischer Kellerautomat (DPDA) für die Sprache $L_T$.*



# DPDA für die arithmetischen Terme

Folgender DPDA erkennt die Sprache $L_{T\$} := \{x\$ \mid x \in L_T\}$.
$\mathcal{A} = \big(\{q_0, q_1, \ldots, q_6\}, \Sigma \cup \{\$\}, \Gamma, \Delta, q_0, Z_0, \{q_6\}\big)$, wobei

$$\Gamma = \{Z_0, T, X_(, X_+, X_*, X_-\}$$

und $\Delta$ besteht aus folgenden Transitionen:

**Stackaufbau**
$(q_0, (, \star, q_0, X_( \star)$
$(q_0, +, \star, q_0, X_+ \star)$
$(q_0, *, \star, q_0, X_* \star)$
$(q_0, -, \star, q_0, X_- \star)$ für $\star \in \Gamma$

**Zahlen**
$(q_0, 0, \star, q_0, T \star)$
$(q_0, 1, \star, q_0, T \star)$ für $\star \in \Gamma$

**Variablen**
$(q_0, \mathtt{x}, \star, q_0, T \star)$
$(q_0, \mathtt{y}, \star, q_0, T \star)$
$(q_0, \mathtt{z}, \star, q_0, T \star)$ für $\star \in \Gamma$

**Stackabbau (Reduce)**
$(q_0, ), T, q_1, \varepsilon)$

**Minus**
$(q_1, \varepsilon, X_-, q_2, \varepsilon)$
$(q_2, \varepsilon, X_(, q_0, T)$

**Plus und Mal**
$(q_1, \varepsilon, X_\circ, q_3, \varepsilon)$ für $\circ \in \{+, *\}$
$(q_3, \varepsilon, T, q_4, \varepsilon)$
$(q_4, \varepsilon, X_(, q_0, T)$

**Akzeptanz**
$(q_0, \$, T, q_5, \varepsilon)$
$(q_5, \varepsilon, Z_0, q_6, \varepsilon)$

- Unsere Grammatik für die arithmetischen Terme ist "leicht zu parsen". Insbesondere ist sie eindeutig, und die Sprache der Terme ist DPDA-erkennbar.
- Unser Parser ist ein LR-Parser: der Text wird von links nach rechts gelesen, und die Hauptarbeit (Reduce) wird am rechten Ende der Regeln gemacht.
- Manche Grammatiken sind noch einfacher und erlauben LL-Parser, bei denen man schon beim Lesen des ersten Symbols der Regel (also am linken Ende) entscheidet, wie zu verfahren ist.
- Die Theorie kontextfreier Sprachen und deterministischer PDAs ist so weit entwickelt, dass man Parser in der Regel automatisch generieren kann.

Mehr dazu in der Vorlesung Compilerbau.



Abschnitt 8.2

# Das Wortproblem für kontextfreie Grammatiken

Der CYK-Algorithmus (Cocke-Younger-Kasami) erlaubt es, Ableitungen von Wörtern in kontextfreien Grammatiken in Chomsky-Normalform zu finden.

Es ist dabei nicht notwendig, dass die Grammatiken eindeutig sind.

Der Einfachheit halber betrachten wir nur das Entscheidungsproblem:

> *Gegeben eine kontextfreie Grammatik $\mathcal{G}$ und ein Wort $w$, entscheide, ob $w \in L(\mathcal{G})$*

(Das Wortproblem für kontextfreie Grammatiken.)





## Eingabe

Kontextfreie Grammatik $\mathcal{G} = (N, \Sigma, P, S)$ in CNF, Wort $w = a_1 \dots a_n \in \Sigma^*$

## Algorithmus

1. Berechne für alle $i, j$ nach wachsender Distanz $d = j - i$ die Mengen

$$N_{ij} = \{A \in N \mid A \xrightarrow{*} \underbrace{w[i,j]}_{=a_i a_{i+1} \dots a_j} \}.$$

2. Bestimme, ob $S \in N_{1n}$.

Die Berechnung der $N_{ij}$ erfolgt induktiv über $d$.

# Induktive Berechnung der $N_{ij}$

Induktionsanfang $d = 0$ (also $i = j$)

$$A \in N_{ii} \iff A \xrightarrow{*} a_i \iff A \to a_i \in P.$$

Induktionsschritt $d > 0$ (also $i < j$)

$A \in N_{ij} \iff A \xrightarrow{*} w[i,j]$

$\iff$ es gibt eine Regel $A \to BC$ und ein $k$ mit $i \leq j < k$, so dass

$B \xrightarrow{*} w[i,k]$ und $C \xrightarrow{*} w[k+1,j]$

$\iff$ es gibt eine Regel $A \to BC$ und ein $k$ mit $i \leq j < k$, so dass

$B \in N_{ik}$ und $C \in N_{k+1\,j}$.



# Beispiel 8.4

Grammatik $\mathcal{G}$:

$$S \to SA \mid a, \quad A \to BS, \quad B \to BB \mid BS \mid b \mid c$$

Eingabewort $w = abaabca$

Tabelle für die $N_{ij}$-Werte

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a$ | $N_{11}$ | $N_{12}$ | $N_{13}$ | $N_{14}$ | $N_{15}$ | $N_{16}$ | $N_{17}$ |
| $b$ | | $N_{22}$ | $N_{23}$ | $N_{24}$ | $N_{25}$ | $N_{26}$ | $N_{27}$ |
| $a$ | | | $N_{33}$ | $N_{34}$ | $N_{35}$ | $N_{36}$ | $N_{37}$ |
| $a$ | | | | $N_{44}$ | $N_{45}$ | $N_{46}$ | $N_{47}$ |
| $b$ | | | | | $N_{55}$ | $N_{56}$ | $N_{57}$ |
| $c$ | | | | | | $N_{66}$ | $N_{67}$ |
| $a$ | | | | | | | $N_{77}$ |

Ableitung von $w$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a$ | $S$ | $N_{12}$ | $N_{13}$ | $N_{14}$ | $N_{15}$ | $N_{16}$ | $N_{17}$ |
| $b$ | $B$ | $N_{23}$ | $N_{24}$ | $N_{25}$ | $N_{26}$ | $N_{27}$ | |
| $a$ | $S$ | $N_{34}$ | $N_{35}$ | $N_{36}$ | $N_{37}$ | | |

$S \vdash SA \vdash SAA \vdash aAA \vdash aBSA \vdash aBSSA \vdash abSSA \vdash abaSA$

$\vdash abaaA \vdash abaaBS \vdash abaaBBS \vdash abaabBS \vdash abaabcS \vdash abaabca.$

**Eingabe:** $\mathcal{G} = (N, \Sigma, P, S)$ kontextfrei in CNF, $w = a_1 \dots a_n \in \Sigma^*$

```
1: for i = 1 to n do
2:     N_ii ← {A ∈ N | A → a_i ∈ P}
3: end for
4: for d = 1 to n − 1 do
5:     for i = 1 to n − d do
6:         j ← i + d
7:         N_ij ← ∅
8:         for k = i to j − 1 do
9:             N_ij ← N_ij ∪ {A | A → BC ∈ P with B ∈ N_ik and C ∈ N_{k+1j}}
10:        end for
11:    end for
12: end for
13: if S ∈ N_1n then
14:    accept
15: else
16:    reject
17: end if
```



# Laufzeitanalyse

Zeitaufwand (grobe obere Abschätzung) bei festem $\mathcal{G}$.

- Initialisierung: $O(n)$
- Innere Schleife über $k$-Werte: $O(n)$
- Mittlere Schleife über $i$-Werte: $O(n \cdot n)$
- Äußere Schleife über $d$-Werte: $O(n \cdot n \cdot n)$

Insgesamt $O(n) + O(n^3)$, d.h. $O(n^3)$.

## Satz 8.5

*Der CYK-Algorithmus löst das Wortproblem für kontextfreie Grammatiken (in CNF) in kubischer Zeit in der Länge des betrachteten Wortes.*

Abschnitt 8.3

# Grenzen der algorithmischen Lösbarkeit

## Grundlegende algorithmische Probleme

Wie bei den endlichen Automaten betrachten wir folgende grundlegende algorithmische Probleme, hier für kontextfreie Grammatiken:

1. Wortproblem: Gegeben kontextfreie Grammatik $\mathcal{G}$, Wort $w$.
   Ist $w \in L(\mathcal{G})$?
2. Leerheitsproblem: Gegeben kontextfreie Grammatik $\mathcal{G}$.
   Ist $L(\mathcal{G}) = \emptyset$?
3. Unendlichkeitsproblem: Gegeben kontextfreie Grammatik $\mathcal{G}$ .
   Ist $L(\mathcal{G})$ unendlich?
4. Inklusionsproblem: Gegeben kontextfreie Grammatiken $\mathcal{G}_1, \mathcal{G}_2$.
   Gilt $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$?
5. Äquivalenzproblem: Gegeben kontextfreie Grammatiken $\mathcal{G}_1, \mathcal{G}_2$.
   Gilt $L(\mathcal{G}_1) = L(\mathcal{G}_2)$?

Das Wortproblem für kontextfreie Grammatiken lässt sich mit Hilfe des CYK-Algorithmus folgendermaßen lösen.

## Eingabe

Kontextfreie Grammatik $\mathcal{G}$, Wort $w \in \Sigma^* \setminus \{\varepsilon\}$.

## Algorithmus

1. Konstruiere zu $\mathcal{G}$ äquivalente Grammatik $\mathcal{G}'$ in CNF.
2. Entscheide $w \in L(\mathcal{G}')$ mit Hilfe des CYK-Algorithmus.

Weil sich die Umwandlung einer Grammatik in CNF in Polynomialzeit durchführen lässt (vgl. Beweis von Satz 6.20), ist die Laufzeit insgesamt polynomiell.

## Bemerkung 8.6

Mit einem ähnlichen Verfahren wie dem zur Elimination von $\varepsilon$-Regeln (vgl. Lemma 6.17) kann man auch in Polynomialzeit entscheiden, ob eine Grammatik das leere Wort erzeugt.





## Beispiel 8.7

Grammatik $\mathcal{G}$

$$\begin{aligned} S &\to AaB \mid aB \\ A &\to AA \mid Sbb \\ B &\to Scc \mid A \mid DD \\ C &\to EaS \mid SS \\ D &\to SAB \mid bE \\ E &\to aab \end{aligned}$$

Ist $L(\mathcal{G}) = \emptyset$?

Nein!

$$S \to aB \to aDD \to abED \to abaabD \to abaabbE \to abaabbaab.$$

# Terminierende Nichtterminale

## Definition 8.8

Sei $\mathcal{G} = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.
Ein Nichtterminalsymbol $A \in N$ ist terminierend in $\mathcal{G}$, wenn es ein Wort $w \in \Sigma^*$ gibt, so dass $A \xrightarrow{*} w$.

## Beobachtung 8.9

*Sei $\mathcal{G} = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.*

*Dann gilt*

$$L(\mathcal{G}) \neq \emptyset \iff S \text{ ist terminierend.}$$



# Markierungsalgorithmus

## Eingabe

Kontextfreie Grammatik $\mathcal{G}$.

## Ausgabe

Liste der terminierenden Nichtterminale von $\mathcal{G}$.

## Algorithmus

1. Markiere auf den rechten Regelseiten alle Terminale.
2. Solange Regel mit unmarkierter linker Seite $A$ existiert, so dass die rechte Seite voll markiert ist: markiere $A$ überall.
3. Gib die Liste aller markierten Nichtterminale aus.

Grammatik $\mathcal{G}$

$$
\begin{array}{lcl}
S & \to & AaB \mid aB \\
A & \to & AA \mid Sbb \\
B & \to & Scc \mid A \mid DD \\
C & \to & EaS \mid SS \\
D & \to & SAB \mid bE \\
E & \to & aab
\end{array}
$$

$$
\begin{array}{lcl}
S & \to & A\underline{a}B \mid \underline{a}B \\
A & \to & AA \mid S\underline{bb} \\
B & \to & Scc \mid A \mid DD \\
C & \to & E\underline{a}S \mid SS \\
D & \to & SAB \mid \underline{b}E \\
E & \to & \underline{aab}
\end{array}
$$

$$
\begin{array}{lcl}
S & \to & A\underline{a}B \mid \underline{a}B \\
A & \to & AA \mid S\underline{bb}
\end{array}
$$



# Korrektheit des Markierungsalgorithmus I

## Satz 8.10

*Der Markierungsalgorithmus berechnet korrekt die terminierenden Nichtterminalsymbole der Eingabegrammatik.*

**Beweis (Skizze).** Sei $\mathcal{G} = (N, \Sigma, P, S)$ die Eingabegrammatik.
Der Algorithmus hält nach spätestens $|N|$ Durchläufen der Schleife in Schritt 2 an, weil in jedem Durchlauf ein neues Nichtterminal markiert wird.

## Behauptung 1

*Falls ein Nichtterminal $A \in N$ vom Algorithmus markiert wird, so ist $A$ terminierend.*

Diese Behauptung kann man per Induktion über die Anzahl der Durchläufe der Schleife in Schritt 2 des Algorithmus (also die Anzahl der bereits markierten Nichtterminale) beweisen.

# Korrektheit des Markierungsalgorithmus II

## Behauptung 2

*Falls ein Nichtterminal $A \in N$ terminierend ist, wird es vom Algorithmus markiert.*

Diese Aussage lässt sich per Induktion über die Länge einer Ableitung eines Terminalwortes aus $A$ beweisen. □



# Laufzeitanalyse

Eine naive Implementierung des Markierungsalgorithmus läuft in quadratischer Zeit:

- Die Schleife in Schritt 2 muss höchstens $|N|$ mal durchlaufen werden.
- Jeder Durchlauf der Schleife benötigt Zeit $O(|\Delta|)$.

Es geschicktere Variante des Algorithmus läuft in Linearzeit.

## Korollar 8.11

*Das Leerheitsproblem für kontextfreie Grammatiken lässt sich in Linearzeit lösen.*

# Unendlichkeit, Inklusion und Äquivalenz

- Für das Unendlichkeitsproblem gibt es einen einfachen Algorithmus, der auf dem Pumping-Lemma beruht.
  (Übung)

  Mit etwas mehr Aufwand lässt sich das Problem auch in Polynomialzeit lösen.

- Das Inklusionsproblem und das Äquivalenzproblem sind unentscheidbare Probleme, das heißt, für diese Probleme gibt es überhaupt keinen Algorithmus, der sie (für alle Eingaben) löst.

  Mehr dazu in der Vorlesung Berechenbarkeit und Komplexität.



# Weitere unentscheidbare Probleme für kontextfreie Grammatiken

1. Universalitätsproblem
   Gegeben Grammatik $\mathcal{G}$ (mit Terminalalphabet $\Sigma$), entscheide, ob $L(\mathcal{G}) = \Sigma^*$.
2. Durchschnittsproblem
   Gegeben zwei Grammatiken $\mathcal{G}, \mathcal{G}'$, entscheide, ob $L(\mathcal{G}) \cap L(\mathcal{G}') = \emptyset$.
3. Regularitätsproblem
   Gegeben Grammatik $\mathcal{G}$, entscheide, ob $L(\mathcal{G})$ eine reguläre Sprache ist.
4. Eindeutigkeitsproblem
   Gegeben Grammatik $\mathcal{G}$, entscheide, ob $\mathcal{G}$ eindeutig ist, das heißt, ob jedes Wort in $L(\mathcal{G})$ einen eindeutigen Ableitungsbaum hat.