SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

18. April 2005

Auch für reguläre Grammatiken gilt ein entsprechender Satz über die "Entfernbarkeit" nullierbarer Nichtterminale:

Lemma 10

Sei $G = (V, \Sigma, P, S)$ eine Chomsky-Grammatik, so dass für alle Regeln $\alpha \to \beta \in P$ gilt:

$$\alpha \in V \text{ und } \beta \in \Sigma^* \cup \Sigma^* V \ .$$

Dann ist $L(G)$ regulär.

Beweis:
Übungsaufgabe! □

Auch für reguläre Grammatiken gilt ein entsprechender Satz über die "Entfernbarkeit" nullierbarer Nichtterminale:

Lemma 10

Sei $G = (V, \Sigma, P, S)$ eine Chomsky-Grammatik, so dass für alle Regeln $\alpha \to \beta \in P$ gilt:

$$\alpha \in V \text{ und } \beta \in \Sigma^* \cup \Sigma^* V \ .$$

Dann ist $L(G)$ regulär.

Beweis:

Übungsaufgabe! □

Beispiel 11

Typ 3: $L = \{a^n;\ n \in \mathbb{N}\}$, Grammatik:
$$\begin{aligned} S &\to a, \\ S &\to aS \end{aligned}$$

Typ 2: $L = \{a^n b^n;\ n \in \mathbb{N}_0\}$, Grammatik:
$$\begin{aligned} S &\to \epsilon, \\ S &\to T, \\ T &\to ab, \\ T &\to aTb \end{aligned}$$

Wir benötigen beim Scannen einen Zähler.

Typ 1: $L = \{a^n b^n c^n;\ n \in \mathbb{N}\}$, Grammatik:
$$\begin{aligned} S &\to aSXY, \\ S &\to aXY, \\ XY &\to YX, \\ aX &\to ab, \\ bX &\to bb, \\ bY &\to bc, \\ cY &\to cc \end{aligned}$$

Wir benötigen beim Scannen mindestens zwei Zähler.

Beispiel 11

Typ 3: $L = \{a^n;\ n \in \mathbb{N}\}$, Grammatik:

$$\begin{aligned} S &\to a, \\ S &\to aS \end{aligned}$$

Typ 2: $L = \{a^n b^n;\ n \in \mathbb{N}_0\}$, Grammatik:

$$\begin{aligned} S &\to \epsilon, \\ S &\to T, \\ T &\to ab, \\ T &\to aTb \end{aligned}$$

Wir benötigen beim Scannen einen Zähler.

Typ 1: $L = \{a^n b^n c^n;\ n \in \mathbb{N}\}$, Grammatik:

$$\begin{aligned} S &\to aSXY, \\ S &\to aXY, \\ XY &\to YX, \\ aX &\to ab, \\ bX &\to bb, \\ bY &\to bc, \\ cY &\to cc \end{aligned}$$

Wir benötigen beim Scannen mindestens zwei Zähler.

Beispiel 11

Typ 3: $L = \{a^n;\ n \in \mathbb{N}\}$, Grammatik:

$$\begin{aligned} S &\to a, \\ S &\to aS \end{aligned}$$

Typ 2: $L = \{a^n b^n;\ n \in \mathbb{N}_0\}$, Grammatik:

$$\begin{aligned} S &\to \epsilon, \\ S &\to T, \\ T &\to ab, \\ T &\to aTb \end{aligned}$$

Wir benötigen beim Scannen einen Zähler.

Typ 1: $L = \{a^n b^n c^n;\ n \in \mathbb{N}\}$, Grammatik:

$$\begin{aligned} S &\to aSXY, \\ S &\to aXY, \\ XY &\to YX, \\ aX &\to ab, \\ bX &\to bb, \\ bY &\to bc, \\ cY &\to cc \end{aligned}$$

Wir benötigen beim Scannen mindestens zwei Zähler.

Die Backus-Naur-Form (BNF) ist ein Formalismus zur kompakten Darstellung von Typ-2-Grammatiken.

- Statt

$$\begin{aligned} A &\rightarrow \beta_1 \\ A &\rightarrow \beta_2 \\ &\vdots \\ A &\rightarrow \beta_n \end{aligned}$$

schreibt man

$$A \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n \ .$$

Die Backus-Naur-Form (BNF) ist ein Formalismus zur kompakten Darstellung von Typ-2-Grammatiken.

- Statt

$$\begin{aligned} A &\rightarrow \alpha\gamma \\ A &\rightarrow \alpha\beta\gamma \end{aligned}$$

schreibt man

$$A \rightarrow \alpha[\beta]\gamma \ .$$

Die Backus-Naur-Form (BNF) ist ein Formalismus zur kompakten Darstellung von Typ-2-Grammatiken.

- Statt

$$\begin{aligned} A &\rightarrow \alpha\gamma \\ A &\rightarrow \alpha\beta\gamma \end{aligned}$$

schreibt man

$$A \rightarrow \alpha[\beta]\gamma \ .$$

(D.h., das Wort β kann, muss aber nicht, zwischen α und γ eingefügt werden.)

Die Backus-Naur-Form (BNF) ist ein Formalismus zur kompakten Darstellung von Typ-2-Grammatiken.

- Statt

$$\begin{aligned} A &\rightarrow \alpha\gamma \\ A &\rightarrow \alpha B\gamma \\ B &\rightarrow \beta \\ B &\rightarrow \beta B \end{aligned}$$

schreibt man

$$A \rightarrow \alpha\{\beta\}\gamma\ .$$

Die Backus-Naur-Form (BNF) ist ein Formalismus zur kompakten Darstellung von Typ-2-Grammatiken.

- Statt

$$\begin{aligned} A &\to \alpha\gamma \\ A &\to \alpha B\gamma \\ B &\to \beta \\ B &\to \beta B \end{aligned}$$

schreibt man

$$A \to \alpha\{\beta\}\gamma\ .$$

(D.h., das Wort β kann beliebig oft (auch Null mal) zwischen α und γ eingefügt werden.)

Beispiel 12

$$
\begin{aligned}
\langle \text{Satz} \rangle &\rightarrow \langle \text{Subjekt} \rangle \langle \text{Prädikat} \rangle \langle \text{Objekt} \rangle \\
\langle \text{Subjekt} \rangle &\rightarrow \langle \text{Artikel} \rangle \langle \text{Attribut} \rangle \langle \text{Substantiv} \rangle \\
\langle \text{Prädikat} \rangle &\rightarrow \text{ist} | \text{hat} | \ldots \\
\langle \text{Artikel} \rangle &\rightarrow \epsilon | \text{der} | \text{die} | \text{das} | \text{ein} | \ldots \\
\langle \text{Attribut} \rangle &\rightarrow \{ \langle \text{Adjektiv} \rangle \} \\
\langle \text{Adjektiv} \rangle &\rightarrow \text{gross} | \text{klein} | \text{schön} | \ldots
\end{aligned}
$$

2.3 Das Wortproblem

Beispiel 13 (Arithmetische Ausdrücke)

$$
\begin{array}{lcl}
\text{<exp>} & \longrightarrow & \text{<term>} \\
\text{<exp>} & \longrightarrow & \text{<exp>} + \text{<term>} \\
\text{<term>} & \longrightarrow & (\text{<exp>}) \\
\text{<term>} & \longrightarrow & \text{<term>} \times \text{<term>} \\
\text{<term>} & \longrightarrow & a \mid b \mid \ldots \mid z
\end{array}
$$

Aufgabe eines Parsers ist nun, zu prüfen, ob eine gegebene Zeichenreihe einen gültigen arithmetischen Ausdruck darstellt und, falls ja, ihn in seine Bestandteile zu zerlegen.

2.3 Das Wortproblem

Beispiel 13 (Arithmetische Ausdrücke)

$$
\begin{array}{lcl}
\text{<exp>} & \longrightarrow & \text{<term>} \\
\text{<exp>} & \longrightarrow & \text{<exp>} + \text{<term>} \\
\text{<term>} & \longrightarrow & (\text{<exp>}) \\
\text{<term>} & \longrightarrow & \text{<term>} \times \text{<term>} \\
\text{<term>} & \longrightarrow & a \mid b \mid \ldots \mid z
\end{array}
$$

Aufgabe eines Parsers ist nun, zu prüfen, ob eine gegebene Zeichenreihe einen gültigen arithmetischen Ausdruck darstellt und, falls ja, ihn in seine Bestandteile zu zerlegen.

Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

Definition 14

1. **Wortproblem:** Gegeben ein Wort $w \in \Sigma^*$, stelle fest, ob

$$w \in L(G) \, ?$$

2. **Ableitungsproblem:** Gegeben ein Wort $w \in L(G)$, gib eine Ableitung $S \to_G^* w$ an, d.h., eine Folge

$$S = w^{(0)} \to_G w^{(1)} \to_G \cdots \to_G w^{(n)} = w$$

mit $w^{(i)} \in (\Sigma \cup V)^*$ für $i = 1, \ldots, n$.

3. **uniformes Wortproblem:** Wortproblem, bei dem jede Probleminstanz sowohl die Grammatik G wie auch die zu testende Zeichenreihe w enthält. Ist G dagegen global festgelegt, spricht man von einem nicht-uniformen Wortproblem.

Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

Definition 14

1. **Wortproblem:** Gegeben ein Wort $w \in \Sigma^*$, stelle fest, ob

$$w \in L(G) \, ?$$

2. **Ableitungsproblem:** Gegeben ein Wort $w \in L(G)$, gib eine Ableitung $S \rightarrow_G^* w$ an, d.h., eine Folge

$$S = w^{(0)} \rightarrow_G w^{(1)} \rightarrow_G \cdots \rightarrow_G w^{(n)} = w$$

mit $w^{(i)} \in (\Sigma \cup V)^*$ für $i = 1, \ldots, n$.

3. **uniformes Wortproblem:** Wortproblem, bei dem jede Probleminstanz sowohl die Grammatik G wie auch die zu testende Zeichenreihe w enthält. Ist G dagegen global festgelegt, spricht man von einem nicht-uniformen Wortproblem.

Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

Definition 14

1. **Wortproblem:** Gegeben ein Wort $w \in \Sigma^*$, stelle fest, ob

$$w \in L(G) \, ?$$

2. **Ableitungsproblem:** Gegeben ein Wort $w \in L(G)$, gib eine Ableitung $S \to_G^* w$ an, d.h., eine Folge

$$S = w^{(0)} \to_G w^{(1)} \to_G \cdots \to_G w^{(n)} = w$$

   mit $w^{(i)} \in (\Sigma \cup V)^*$ für $i = 1, \ldots, n$.

3. **uniformes Wortproblem:** Wortproblem, bei dem jede Probleminstanz sowohl die Grammatik G wie auch die zu testende Zeichenreihe w enthält. Ist G dagegen global festgelegt, spricht man von einem nicht-uniformen Wortproblem.

Bemerkung:
Das uniforme wie auch das nicht-uniforme Wortproblem ist für Typ-0-Sprachen (also die rekursiv-aufzählbare Sprachen) nicht entscheidbar. Wir werden später sehen, dass es zum Halteproblem für Turingmaschinen äquivalent ist.

Es gilt jedoch

Satz 15

Für kontextsensitive Grammatiken ist das Wortproblem entscheidbar.
Genauer: Es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes w in endlicher Zeit entscheidet, ob $w \in L(G)$.

Bemerkung:
Das uniforme wie auch das nicht-uniforme Wortproblem ist für Typ-0-Sprachen (also die rekursiv-aufzählbare Sprachen) nicht entscheidbar. Wir werden später sehen, dass es zum Halteproblem für Turingmaschinen äquivalent ist.

Es gilt jedoch

Satz 15

Für kontextsensitive Grammatiken ist das Wortproblem entscheidbar.

Genauer: Es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes w in endlicher Zeit entscheidet, ob $w \in L(G)$.

Bemerkung:
Das uniforme wie auch das nicht-uniforme Wortproblem ist für Typ-0-Sprachen (also die rekursiv-aufzählbare Sprachen) nicht entscheidbar. Wir werden später sehen, dass es zum Halteproblem für Turingmaschinen äquivalent ist.

Es gilt jedoch

Satz 15

Für kontextsensitive Grammatiken ist das Wortproblem entscheidbar.
Genauer: Es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes w in endlicher Zeit entscheidet, ob $w \in L(G)$.

Beweisidee:

Angenommen $w \in L(G)$. Dann gibt es eine Ableitung

$$S = w^{(0)} \to_G w^{(1)} \to_G \cdots \to_G w^{(n)} = w$$

mit $w^{(i)} \in (\Sigma \cup V)^*$ für $i = 1, \ldots, n$.

Da aber G kontextsensitiv ist, gilt (falls $w \neq \epsilon$)

$$|w^{(0)}| \leq |w^{(1)}| \leq \cdots \leq |w^{(n)}| \,,$$

d.h., es genügt, alle Wörter in $L(G)$ der Länge $\leq |w|$ zu erzeugen.

Beweisidee:

Angenommen $w \in L(G)$. Dann gibt es eine Ableitung

$$S = w^{(0)} \to_G w^{(1)} \to_G \cdots \to_G w^{(n)} = w$$

mit $w^{(i)} \in (\Sigma \cup V)^*$ für $i = 1, \ldots, n$.

Da aber G kontextsensitiv ist, gilt (falls $w \neq \epsilon$)

$$|w^{(0)}| \leq |w^{(1)}| \leq \cdots \leq |w^{(n)}| \,,$$

d.h., es genügt, alle Wörter in $L(G)$ der Länge $\leq |w|$ zu erzeugen.

Beweis:

Sei o.B.d.A. $w \neq \epsilon$ und sei

$$T_m^n := \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w' \text{ lässt sich aus } S \text{ in } \leq m \text{ Schritten ableiten}\}$$

Diese Mengen kann man für alle n und m induktiv wie folgt berechnen:

$$\begin{aligned} T_0^n &:= \{S\} \\ T_{m+1}^n &:= T_m^n \cup \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w'' \to w' \text{ für ein } w'' \in T_m^n\} \end{aligned}$$

Beachte: Für alle m gilt: $|T_m^n| \leq \sum_{i=1}^{n} |\Sigma \cup V|^i$.
Es muss daher immer ein m_0 geben mit

$$T_{m_0}^n = T_{m_0+1}^n = \ldots =: T_n \,.$$

□

Beweis:

Sei o.B.d.A. $w \neq \epsilon$ und sei

$$T_m^n := \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w' \text{ lässt sich aus } S \text{ in } \leq m \text{ Schritten ableiten}\}$$

Diese Mengen kann man für alle n und m induktiv wie folgt berechnen:

$$\begin{aligned} T_0^n &:= \{S\} \\ T_{m+1}^n &:= T_m^n \cup \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w'' \to w' \text{ für ein } w'' \in T_m^n\} \end{aligned}$$

Beachte: Für alle m gilt: $|T_m^n| \leq \sum_{i=1}^{n} |\Sigma \cup V|^i$.
Es muss daher immer ein m_0 geben mit

$$T_{m_0}^n = T_{m_0+1}^n = \ldots =: T_n \,.$$

□

Beweis:

Sei o.B.d.A. $w \neq \epsilon$ und sei

$$T_m^n := \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w' \text{ lässt sich aus } S \text{ in } \leq m \text{ Schritten ableiten}\}$$

Diese Mengen kann man für alle n und m induktiv wie folgt berechnen:

$$\begin{aligned} T_0^n &:= \{S\} \\ T_{m+1}^n &:= T_m^n \cup \{w' \in (\Sigma \cup V)^*;\ |w'| \leq n \text{ und } w'' \to w' \text{ für ein } w'' \in T_m^n\} \end{aligned}$$

Beachte: Für alle m gilt: $|T_m^n| \leq \sum_{i=1}^{n} |\Sigma \cup V|^i$.
Es muss daher immer ein m_0 geben mit

$$T_{m_0}^n = T_{m_0+1}^n = \ldots =: T_n \ .$$

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
$T' := T$
$T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beweis:

Algorithmus:

$n := |w|$
$T := \{S\}$
$T' := \emptyset$
while $T \neq T'$ **do**
  $T' := T$
  $T := T' \cup \{w' \in (V \cup \Sigma)^+;\ |w'| \leq n, (\exists w'' \in T')[w'' \to w']\}$
od
if $w \in T$ **return** "ja" **else return** "nein" **fi**

□

Beispiel 16

Gegeben sei die Typ-2-Grammatik mit den Produktionen

$$S \to ab \text{ und } S \to aSb$$

sowie das Wort $w = abab$.

$$\begin{aligned}
T_0^4 &= \{S\} \\
T_1^4 &= \{S, ab, aSb\} \\
T_2^4 &= \{S, ab, aSb, aabb\} \quad aaSbb \text{ ist zu lang!} \\
T_3^4 &= \{S, ab, aSb, aabb\}
\end{aligned}$$

Beispiel 16

Gegeben sei die Typ-2-Grammatik mit den Produktionen

$$S \to ab \text{ und } S \to aSb$$

sowie das Wort $w = abab$.

$$\begin{aligned} T_0^4 &= \{S\} \\ T_1^4 &= \{S, ab, aSb\} \\ T_2^4 &= \{S, ab, aSb, aabb\} \quad aaSbb \text{ ist zu lang!} \\ T_3^4 &= \{S, ab, aSb, aabb\} \end{aligned}$$

Also lässt sich das Wort w mit der gegebenen Grammatik nicht erzeugen!

Beispiel 16

Gegeben sei die Typ-2-Grammatik mit den Produktionen

$$S \to ab \text{ und } S \to aSb$$

sowie das Wort $w = abab$.

$$\begin{aligned} T_0^4 &= \{S\} \\ T_1^4 &= \{S, ab, aSb\} \\ T_2^4 &= \{S, ab, aSb, aabb\} \quad aaSbb \text{ ist zu lang!} \\ T_3^4 &= \{S, ab, aSb, aabb\} \end{aligned}$$

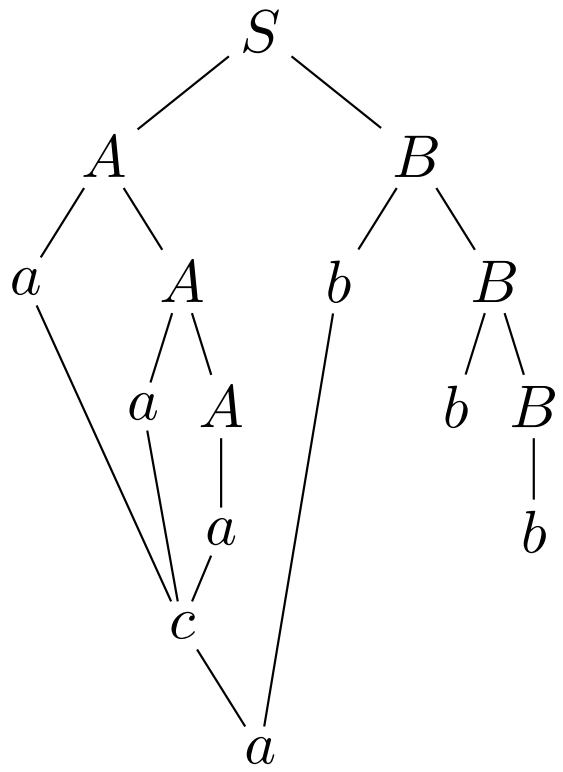
Bemerkung: Der angegebene Algorithmus ist nicht sehr effizient! Für kontextfreie Grammatiken gibt es wesentlich effizientere Verfahren, die wir später kennenlernen werden!

2.4 Ableitungsgraph und Ableitungsbaum

Grammatik:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aA \\
A &\rightarrow a \\
B &\rightarrow bB \\
B &\rightarrow b \\
aaa &\rightarrow c \\
cb &\rightarrow a
\end{aligned}
$$

Beispiel:



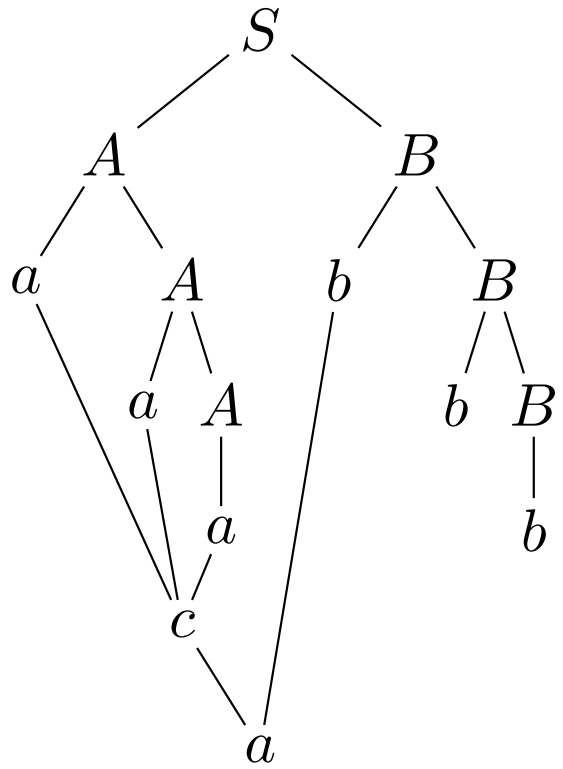
Die Terminale ohne Kante nach unten entsprechen, von links nach rechts gelesen, dem durch den Ableitungsgraphen dargestellten Wort.

2.4 Ableitungsgraph und Ableitungsbaum

Grammatik:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aA \\
A &\rightarrow a \\
B &\rightarrow bB \\
B &\rightarrow b \\
aaa &\rightarrow c \\
cb &\rightarrow a
\end{aligned}
$$

Beispiel:



Der Ableitungsgraph entspricht der Ableitung

$$
\begin{aligned}
S &\rightarrow AB \rightarrow aAB \rightarrow aAbB \rightarrow aaAbB \rightarrow aaAbbB \rightarrow \\
&\rightarrow aaabbB \rightarrow aaabbb \rightarrow cbbb \rightarrow abb
\end{aligned}
$$

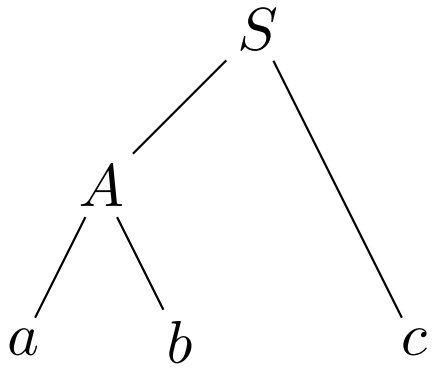
Beobachtung: Bei kontextfreien Sprachen sind die Ableitungsgraphen immer Bäume.

Beispiel 17

Grammatik:

$$
\begin{aligned}
S &\to aB \\
S &\to Ac \\
A &\to ab \\
B &\to bc
\end{aligned}
$$

Ableitungsbaum:



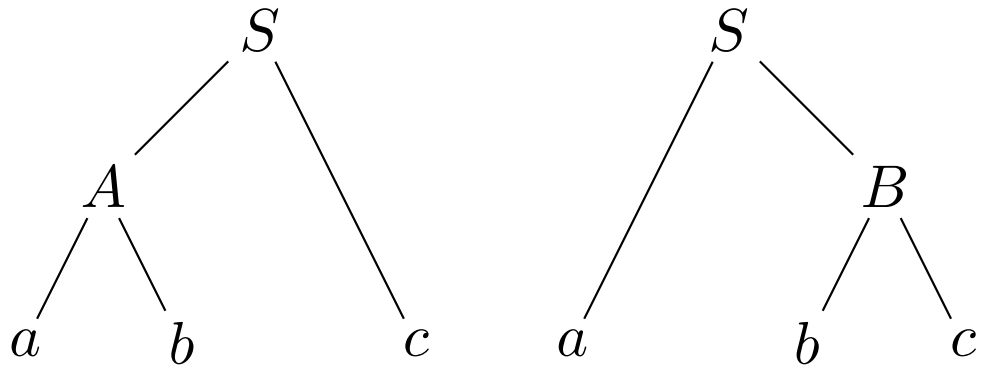
Beobachtung: Bei kontextfreien Sprachen sind die Ableitungsgraphen immer Bäume.

Beispiel 17

Grammatik:

$$\begin{aligned} S &\to aB \\ S &\to Ac \\ A &\to ab \\ B &\to bc \end{aligned}$$

Ableitungsbäume:



Für das Wort abc gibt es zwei verschiedene Ableitungsbäume.

Definition 18

- Eine Ableitung

$$S = w^{(0)} \to w^{(1)} \to \cdots \to w^{(n)} = w$$

eines Wortes w heißt Linksableitung, wenn für jede Anwendung einer Produktion $\alpha \to \beta$ auf $w^{(i)} = x\alpha z$ gilt, dass sich keine Regel der Grammatik auf ein echtes Präfix von $x\alpha$ anwenden lässt.

- Eine Grammatik heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch mehrdeutig.
- Eine Sprache L heißt eindeutig, wenn es für L eine eindeutige Grammatik gibt. Ansonsten heißt L mehrdeutig.

Bemerkung: Für eindeutige monotone Grammatiken ist das Wortproblem sehr einfach lösbar.

Definition 18

- Eine Ableitung

$$S = w^{(0)} \rightarrow w^{(1)} \rightarrow \cdots \rightarrow w^{(n)} = w$$

eines Wortes w heißt Linksableitung, wenn für jede Anwendung einer Produktion $\alpha \rightarrow \beta$ auf $w^{(i)} = x\alpha z$ gilt, dass sich keine Regel der Grammatik auf ein echtes Präfix von $x\alpha$ anwenden lässt.

- Eine Grammatik heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch mehrdeutig.
- Eine Sprache L heißt eindeutig, wenn es für L eine eindeutige Grammatik gibt. Ansonsten heißt L mehrdeutig.

Bemerkung: Für eindeutige monotone Grammatiken ist das Wortproblem sehr einfach lösbar.

Definition 18

- Eine Ableitung

$$S = w^{(0)} \rightarrow w^{(1)} \rightarrow \cdots \rightarrow w^{(n)} = w$$

  eines Wortes w heißt Linksableitung, wenn für jede Anwendung einer Produktion $\alpha \rightarrow \beta$ auf $w^{(i)} = x\alpha z$ gilt, dass sich keine Regel der Grammatik auf ein echtes Präfix von $x\alpha$ anwenden lässt.
- Eine Grammatik heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch mehrdeutig.
- Eine Sprache L heißt eindeutig, wenn es für L eine eindeutige Grammatik gibt. Ansonsten heißt L mehrdeutig.

Bemerkung: Für eindeutige monotone Grammatiken ist das Wortproblem sehr einfach lösbar.

Definition 18

- Eine Ableitung

$$S = w^{(0)} \rightarrow w^{(1)} \rightarrow \cdots \rightarrow w^{(n)} = w$$

  eines Wortes w heißt Linksableitung, wenn für jede Anwendung einer Produktion $\alpha \rightarrow \beta$ auf $w^{(i)} = x\alpha z$ gilt, dass sich keine Regel der Grammatik auf ein echtes Präfix von $x\alpha$ anwenden lässt.
- Eine Grammatik heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch mehrdeutig.
- Eine Sprache L heißt eindeutig, wenn es für L eine eindeutige Grammatik gibt. Ansonsten heißt L mehrdeutig.

Bemerkung: Für eindeutige monotone Grammatiken ist das Wortproblem sehr einfach lösbar.

Definition 18

- Eine Ableitung

$$S = w^{(0)} \rightarrow w^{(1)} \rightarrow \cdots \rightarrow w^{(n)} = w$$

  eines Wortes w heißt Linksableitung, wenn für jede Anwendung einer Produktion $\alpha \rightarrow \beta$ auf $w^{(i)} = x\alpha z$ gilt, dass sich keine Regel der Grammatik auf ein echtes Präfix von $x\alpha$ anwenden lässt.
- Eine Grammatik heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch mehrdeutig.
- Eine Sprache L heißt eindeutig, wenn es für L eine eindeutige Grammatik gibt. Ansonsten heißt L mehrdeutig.

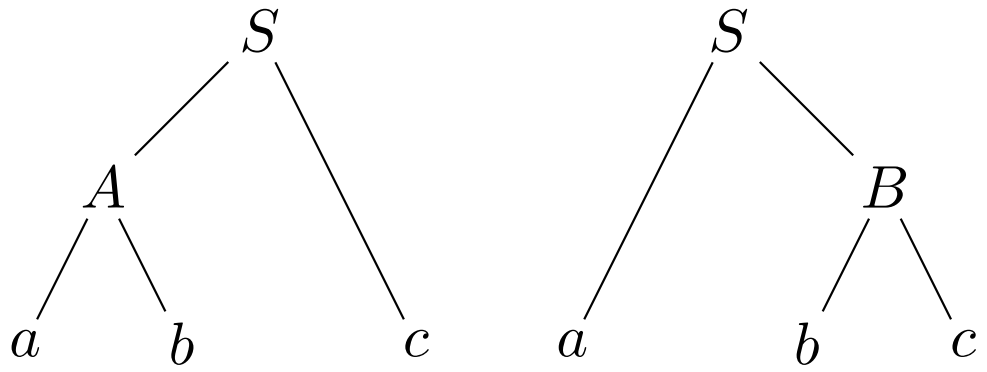
Bemerkung: Für eindeutige monotone Grammatiken ist das Wortproblem sehr einfach lösbar.

Beispiel 19

Grammatik:

$$
\begin{aligned}
S &\rightarrow aB \\
S &\rightarrow Ac \\
A &\rightarrow ab \\
B &\rightarrow bc
\end{aligned}
$$

Ableitungsbäume:



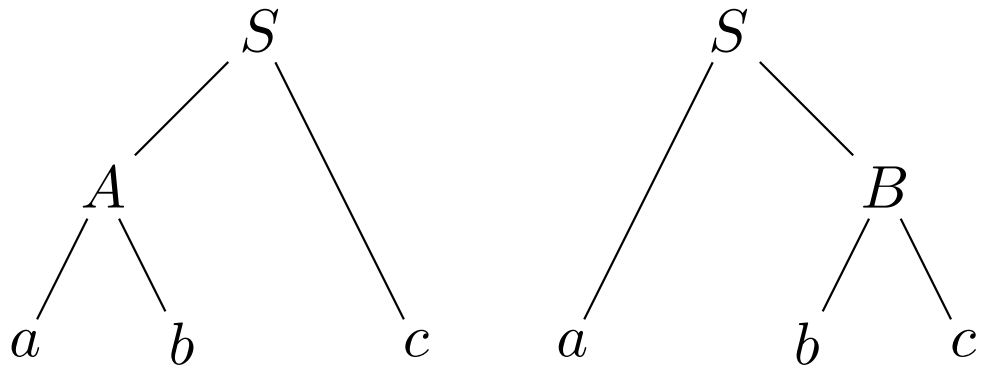
Beide Ableitungsbäume für das Wort abc entsprechen Linksableitungen.

Beispiel 19

Grammatik:

$$
\begin{aligned}
S &\rightarrow aB \\
S &\rightarrow Ac \\
A &\rightarrow ab \\
B &\rightarrow bc
\end{aligned}
$$

Ableitungsbäume:



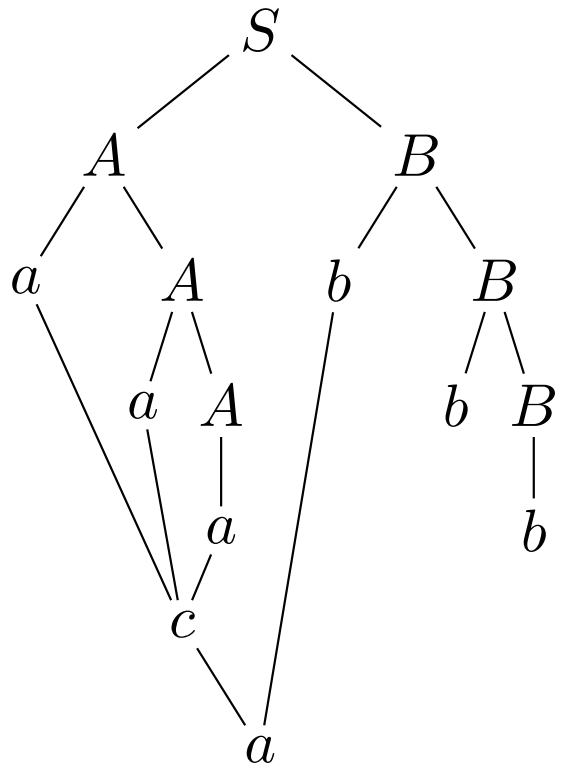
Beide Ableitungsbäume für das Wort abc entsprechen Linksableitungen.

Beispiel 20

Grammatik:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aA \\
A &\rightarrow a \\
B &\rightarrow bB \\
B &\rightarrow b \\
aaa &\rightarrow c \\
cb &\rightarrow a
\end{aligned}
$$

Ableitung:



Eine Linksableitung ist

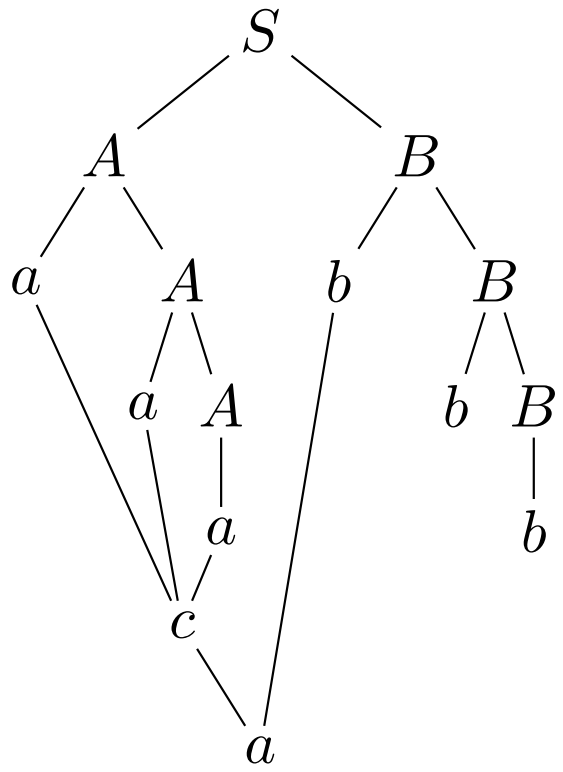
$$
\begin{aligned}
S &\rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaB \rightarrow cB \rightarrow \\
&\rightarrow cbB \rightarrow aB \rightarrow abB \rightarrow abb
\end{aligned}
$$

Beispiel 20

Grammatik:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aA \\
A &\rightarrow a \\
B &\rightarrow bB \\
B &\rightarrow b \\
aaa &\rightarrow c \\
cb &\rightarrow a
\end{aligned}
$$

Ableitung:



Eine andere Linksableitung ist

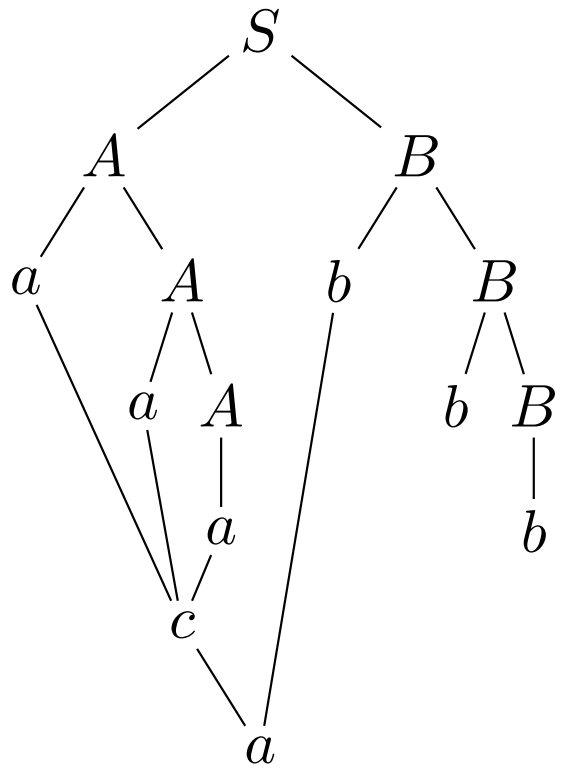
$$S \rightarrow AB \rightarrow aB \rightarrow abB \rightarrow abb\ .$$

Beispiel 20

Grammatik:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aA \\
A &\rightarrow a \\
B &\rightarrow bB \\
B &\rightarrow b \\
aaa &\rightarrow c \\
cb &\rightarrow a
\end{aligned}
$$

Ableitung:



Die Grammatik ist also mehrdeutig.