Definition von Konstruktoren

Schließlich müssen wir die Definition eines Konstruktors übersetzen:
$K(t_2\ x_2, \ldots, t_n\ x_n)\{\ ss\ \}$

Wir übersetzen sie wie eine Funktiondefinition ohne Rückgabewert – erweitert um die Initialisierung des Verweises auf die Tabelle üM_K der überschreibbaren Methoden der Klasse K.

Die Makro-Instruktion **initVirtual** K steht als Abkürzung für:

$$\textbf{initVirtual}\ K \equiv \textbf{loadc}\ \_\text{üM}_K, \textbf{storem}\ 0, \textbf{alloc}\ -1$$

wobei $\_\text{üM}_K$ die Anfangsadresse der Tabelle üM_K ist.

Dann ist:

$$\text{code}^{\rho}\ (K(t_2\ x_2, \ldots, t_n\ x_n)\{\ ss\ \}) = \_K\text{: } \textbf{alloc}\ l, \textbf{enter}\ k, \textbf{initVirtual}\ K, \text{code}^{\rho'}\ ss, \textbf{return}\ r$$

wobei ρ' die Adressumgebung innerhalb des Konstruktors K nach Abarbeitung der formalen Parameter ist. (k, l, r wie Funktionsdefinition)

Definition von Konstruktoren

Beispiel: **list** (**int** x) { $info \leftarrow x$; $next \leftarrow$ **null**; }

Die Übersetzung des Konstruktors der Klasse **list** liefert:

$\_list$: **alloc** 0, **enter** 3, **loadc** $\_\text{üM}_{list}$, **storem** 0, **alloc** -1, **loadr** -4, **storem** 1, **alloc** -1, **loadc** 0, **storem** 2, **alloc** -1, **return** 4

Einzige Besonderheit:
Der Verweis auf die Tabelle der überschreibbaren Methoden wird gesetzt.

Die Referenz auf das Objekt, für den der Konstruktor aufgerufen wird, hat wieder die Adresse -3 (relativ zum FP). Entsprechend kann man auf die Attribute dieses Objekts mit den Befehlen **loadm** und **storem** zugreifen.

Aufruf von Konstruktoren der Oberklasse

Ein Konstruktor kann Konstruktoren der Oberklasse O aufrufen.
In C++ steht der Aufruf im Kopf der Konstruktor-Deklaration:
$K(t_2\ x_2, \ldots, t_m\ x_m) : O(e_2, \ldots, e_n)\{\ ss\ \}$

Der Konstruktor der Oberklasse muss vor der Auswertung des Rumpfs ss aufgerufen werden.
Ihm wird wieder der Verweis auf das aktuelle Objekt übergeben.

Die Adressumgebung der Klasse K wird dazu erweitert um die formalen Parameter $x_2, \ldots, x_m$ des Konstruktors.

Aufruf von Konstruktoren der Oberklasse

Da der Konstruktor der Oberklasse evtl. seine eigenen Versionen der überschreibbaren Methoden aufruft, warten wir mit der Initialisierung des Tabellenverweises, bis der Konstruktor der Oberklasse ausgeführt wurde:

$\text{code}^{\rho}\ (K(t_2\ x_2, \ldots, t_m\ x_m) : O(e_2, \ldots, e_n)\{\ ss\ \}) =$
$\_K$: **alloc** l, **enter** k, $\text{code}_W^{\rho_1}\ e_n$, $\ldots$, $\text{code}_W^{\rho_1}\ e_2$, **loadr** -3, **calld** $\_O$,
initVirtual K, $\text{code}^{\rho'}\ ss$, **return** r

wobei ρ_1 die Adressumgebung zur Auswertung der aktuellen Parameter des Konstruktors der Oberklasse und
ρ' die Adressumgebung innerhalb des Konstruktors K sind.

Wann wird die Tabelle der virtuellen Methoden angelegt?

Java macht es anders:
Dort kann der Konstruktor der Oberklasse schon auf die Methoden der Unterklasse zugreifen; die Tabelle der virtuellen Methoden wird dort vor dem Konstruktor-Aufruf für die Oberklasse angelegt.

Übersetzung einer Klassendefinition

Bei der Übersetzung einer Klassendefinition (**class cname**) muss die Tabelle $\text{üM}_{\textbf{cname}}$ für die Anfangsadressen der überschreibbaren Methoden angelegt werden:

- Hat die Klasse eine Oberklasse, so wird zunächst eine Kopie der entsprechenden Tabelle der Oberklasse angelegt,
- sonst eine leere Tabelle.

- Für jede neue überschreibbare Methode der Klasse wird diese Tabelle erweitert,
- für jede redefinierte Methode wird die Adresse überschrieben.

Die Anfangsadresse $\_\text{üM}_{\textbf{cname}}$ der Tabelle $\text{üM}_{\textbf{cname}}$ wird als globale Adresse in die Adressumgebung aufgenommen.

Mehrfach„ver“erbung

Einige Programmiersprachen (z.B. C++, Eiffel und Scala) erlauben einer Unterklasse K von mehreren Oberklassen $O_1,\ldots,O_k$ gleichzeitig zu erben (*multiple inheritance*).
Dafür hat sich der Begriff **mehrfache Vererbung** eingebürgert.
Sprachlich müsste es **mehrfache Beerbung** heißen.

Die Herausforderung besteht hier in der Erfindung raffinierter Implementierungstechniken, aber auch im Sprachentwurf selbst.

Ein Problem bereitet die Auflösung der möglichen Mehrdeutigkeiten:

- verschiedene Methoden gleichen Namens können geerbt werden oder
- dieselbe Oberklasse trägt auf mehreren Wegen zur Unterklasse bei.

Mehrfachbeerbung

Auch die Implementierungsidee, dass Attribute und überschreibbare Methoden in allen Unterklassen die gleiche Adresse haben wie in der Oberklasse, funktioniert nur, solange eine Klasse von *einer* Oberklasse erbt.

Stattdessen bekommt jede Klasse K eine Hash-Tabelle h_K, die jedem Methodennamen f die in K gültige Codeadresse der zugehörigen Implementierung zuordnet.

Teil V

Syntaktische Analyse

Theoretische Grundlagen

Problem

Wie definiert man eine unendliche Menge von (syntaktisch korrekten) Programmen?

Lösung

Wie bei natürlichen Sprachen:

durch eine Grammatik für die Programmiersprache!

Alphabet, Wort, formale Sprache

Definition

- Ein Alphabet X ist eine endliche, nichtleere Menge von Zeichen oder Symbolen.
- Ein Wort w über dem Alphabet X ist eine endliche Folge von Symbolen aus X, $w = a_1 \dots a_n$, $a_i \in X,\ n \geq 0,\ 1 \leq i \leq n$.
- Die Länge $|w|$ gibt die Anzahl der Zeichen des Wortes w an.
- Das leere Wort ε hat die Länge $|\varepsilon| = 0$.
- X^* ist die Menge aller Wörter über dem Alphabet X.
- Jede Teilmenge L von X^* ist eine formale Sprache.

Alphabet, Wort, formale Sprache

Beispiel

Alphabet:

$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \ . \ , +, -\}$$

Wörter aus T^*: z.B.

$$+12 - 3.+$$
$$+ + -$$
$$-12.3$$
$$23$$
$$\varepsilon$$
$$11111111.2$$

Kontextfreie Grammatik

Definition

Eine kontextfreie Grammatik G ist ein Tupel $G = (N, T, P, S)$ mit

- N ist ein Alphabet (nichtterminale Symbole),
- T ist ein Alphabet mit $T \cap N = \emptyset$ (terminale Symbole),
- P ist eine endliche Menge von Produktionen (Regeln) der Form $A \to \alpha$ mit $A \in N$ und $\alpha \in (N \cup T)^*$,
- $S \in N$ ist das Startsymbol.

Bemerkung

Eine Produktion $A \to \alpha$ aus P heißt A-Produktion.

Für alle A-Produktionen einer Grammatik
$A \to \alpha_1,\ A \to \alpha_2,\ \ldots\ ,\ A \to \alpha_n$ schreibt man kurz $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$.

Kontextfreie Grammatik

Oft werden nur die Produktionen angegeben.

Das Startsymbol der Grammatik und die Alphabete für die terminalen und die nichtterminalen Symbole sind dann *implizit* festgelegt:

- Das Symbol auf der linken Seite der *ersten* Produktion ist das Startsymbol.
- Jedes Symbol auf der linken Seite einer Produktion ist ein nichtterminales Symbol.
- Jedes *andere* Symbol auf der rechten Seite einer Produktion ist ein terminales Symbol.

Kontextfreie Grammatik

Beispiel (Festkommazahlen mit Vorzeichen)

$$
\begin{aligned}
P = \{ \quad & S \to VZB, \\
& V \to +, V \to -, V \to \varepsilon, && \text{-- Vorzeichen} \\
& Z \to D, Z \to DZ, && \text{-- Ziffernfolge} \\
& D \to 0, D \to 1, D \to 2, D \to 3, D \to 4, \\
& D \to 5, D \to 6, D \to 7, D \to 8, D \to 9, && \text{-- Digit} \\
& B \to \varepsilon, B \to . \, Z \quad \} && \text{-- Bruch}
\end{aligned}
$$

$G = \{N, T, P, S\}$

$N = \{S, V, Z, D, B\}$

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \ . \ , +, -\}$

$L \subseteq T^*$ enthält alle Festkommazahlen mit Vorzeichen

Ableitung

Problem

Wie erzeugt man aus einer kontextfreien Grammatik Wörter einer Sprache?

Definition

Seien γ_1 und γ_2 Wörter aus terminalen und nichtterminalen Symbolen, also $\gamma_1, \gamma_2 \in (N \cup T)^*$.

γ_2 lässt sich aus γ_1 bzgl. G in einem Schritt ableiten ($\gamma_1 \Longrightarrow \gamma_2$), falls gilt:

1. $\gamma_1 = \beta_1 A \beta_2$ mit $A \in N,\ \beta_1, \beta_2 \in (N \cup T)^*$
   γ_1 enthält ein nichtterminales Symbol A
2. $A \to \alpha$ ist eine Produktion aus P
   es gibt eine A-Produktion in P
3. $\gamma_2 = \beta_1 \alpha \beta_2$
   γ_2 entsteht durch Ersetzung von A durch α

erzeugte Sprache

Seien $\gamma_1, \ldots, \gamma_m \in (N \cup T)^*$, $m \geq 1$ und gelte $\gamma_i \Longrightarrow \gamma_{i+1}$, $1 \leq i \leq m-1$.

- Dann lässt sich γ_m aus γ_1 (in $m-1$ Schritten) ableiten.
- Kann man aus einem Wort $\beta \in (N \cup T)^*$ ein Wort γ (in Null oder mehr Schritten) ableiten, so schreibt man $\beta \overset{*}{\Longrightarrow} \gamma$.

Ein aus dem Startsymbol mit Regeln aus P ableitbare Wort heißt **Satzform** von G.

Definition

Die von G erzeugte (kontextfreie) Sprache $L(G)$ ist die Menge aller terminalen Wörter, die man vom Startsymbol S ableiten kann, d. h.

$$L(G) = \{w \in T^* \mid S \overset{*}{\Longrightarrow} w\}.$$

erzeugte Sprache

Beispiel (Festkommazahlen mit Vorzeichen)

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \ . \ , +, -\}$

$N = \{S, V, Z, D, B\}$

$P = \{ \quad S \to VZB, \quad V \to + \mid - \mid \varepsilon, \quad Z \to D \mid DZ,$
$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9, \quad B \to \varepsilon \mid . \ Z \ \}$

Ableitung von -12.3:
$S \Longrightarrow VZB \Longrightarrow VZ.Z \Longrightarrow -Z.Z \Longrightarrow -DZ.Z \Longrightarrow -DZ.D \Longrightarrow$
$-1Z.D \Longrightarrow -1D.D \Longrightarrow -12.D \Longrightarrow -12.3$

Ableitung von 23:
$S \Longrightarrow VZB \Longrightarrow VZ \Longrightarrow VDZ \Longrightarrow DZ \Longrightarrow 2Z \Longrightarrow 2D \Longrightarrow 23$

$-.3$ oder 23. sind nicht aus der Grammatik ableitbar!

Geordnete Anwendung von Produktionen

Definition

Wird in einer Ableitungsfolge

$$S \Longrightarrow \gamma_1 \Longrightarrow \gamma_2 \Longrightarrow \ldots \Longrightarrow \gamma_k = w \in T^*$$

stets das linkeste (rechteste) nichtterminale Zeichen in den γ_i ersetzt, so spricht man von einer Linksableitung (Rechtsableitung) des Wortes w.

Dann ist in der Definition des Ableitungsschritts $\beta_1 \in T^*$ ($\beta_2 \in T^*$).

Satz

Gilt $S \overset{*}{\Longrightarrow} w$ mit $w \in T^*$, so lässt sich w durch eine Linksableitung (Rechtsableitung) ableiten.

Linksableitung

Beispiel (Festkommazahlen mit Vorzeichen)

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \ . \ , +, -\}$

$N = \{S, V, Z, D, B\}$

$P = \{ \quad S \to VZB, \quad V \to + \mid - \mid \varepsilon, \quad Z \to D \mid DZ,$
$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9, \quad B \to \varepsilon \mid . \ Z \ \}$

Linksableitung von -12.3:
$S \Longrightarrow VZB \Longrightarrow -ZB \Longrightarrow -DZB \Longrightarrow -1ZB \Longrightarrow -1DB \Longrightarrow -12B \Longrightarrow$
$-12.Z \Longrightarrow -12.D \Longrightarrow -12.3$

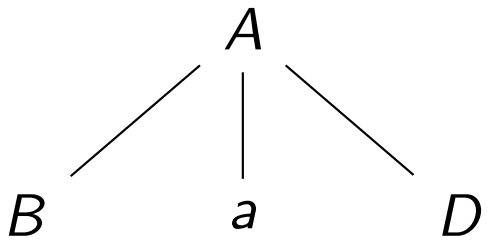
Linksableitung von 23:
$S \Longrightarrow VZB \Longrightarrow ZB \Longrightarrow DZB \Longrightarrow 2ZB \Longrightarrow 2DB \Longrightarrow 23B \Longrightarrow 23$

Ableitungsbaum

Oft möchte man beschreiben, welche Produktionen einer kontextfreien Grammatik ein Wort ableiten, ohne eine bestimmte *Reihenfolge* der Anwendungen der Produktionen anzugeben.

Ein **Ableitungsbaum** stellt eine Ableitung graphisch dar, abstrahiert aber von der Reihenfolge der Ableitungsschritte.

- Jeder interne Knoten eines Ableitungsbaums ist mit einem nichtterminalen Symbol A markiert,
  die direkten Nachfolger dieses Knotens tragen von links nach rechts die Symbole der rechten Seite einer A-Produktion.



Die Produktion $A \to BaD$ im Ableitungsbaum.

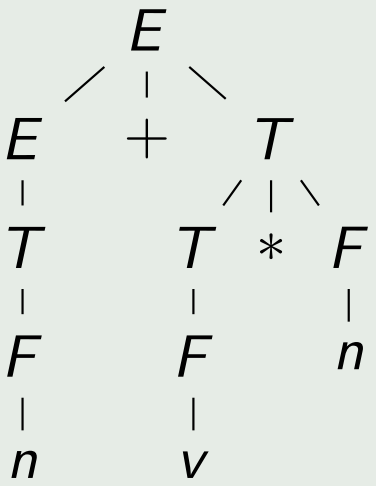
Ableitungsbaum

- Die Blätter eines Ableitungsbaums sind mit terminalen oder nichtterminalen Symbolen oder dem leeren Wort ε markiert und bilden von links nach rechts gelesen ein Wort.
  Dieses Wort lässt sich aus dem Symbol an der Wurzel des Ableitungsbaums ableiten.
- Ist die Wurzel mit dem Startsymbol der Grammatik G und sind alle Blätter mit terminalen Symbolen oder dem leeren Wort ε markiert;
  so liegt das abgelesene Wort in $L(G)$.
- Hat ein Wort $w \in L(G)$ mehrere Ableitungsbäume, so heißt die Grammatik G mehrdeutig.
  Hat jedes Wort $w \in L(G)$ genau einen Ableitungsbaum, so heißt die Grammatik G eindeutig.

Ableitungsbaum für $n + v \star n$

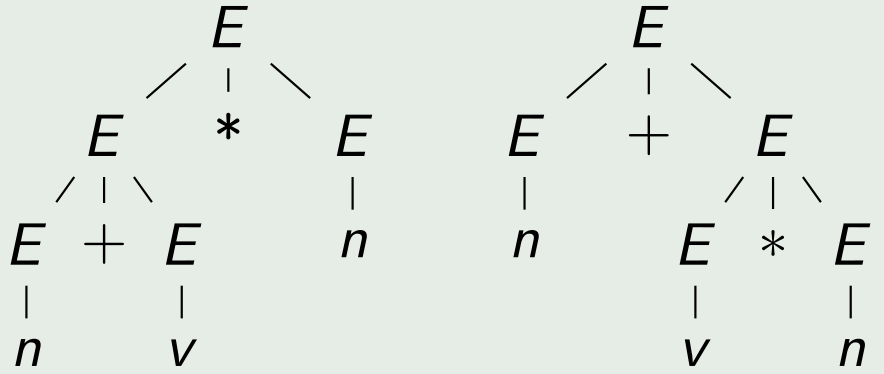
Beispiel (Grammatiken für arithmetische Ausdrücke)

$G_{4'} = (N_1, T_1, P_1, E)$ mit
$N_1 = \{Expr, Term, Faktor\}$,
$T_1 = \{number, variable, *, +, (,)\}$
und
$P_1 = \{ \quad E \to E + T \mid T,$
$\quad T \to T * F \mid F,$
$\quad F \to n \mid v \mid (E)\}$



$G_{4''} = (N_2, T_1, P_2, E)$ mit
$N_2 = \{E\}$ und
$P_2 = \{E \to E+E \mid E*E \mid (E) \mid n \mid v\}$

Grammatik $G_{4''}$ mehrdeutig, da zwei Ableitungsbäume für $n + v * n$



Reduktionsschritte

Der Ableitungsbaum beschreibt die syntaktische Struktur eines Wortes bzgl. einer gegebenen kontextfreien Grammatik.

Das Bestimmen der syntaktischen Struktur eines Wortes heißt Syntaxanalyse (*parsing*).

Die Syntaxanalyse ist eine Teilaufgabe von Übersetzern, da ein Programm überprüft werden muss, ob es ein Element der (Programmier-)Sprache ist.

Im Übersetzerbau betrachtet man eine Ableitungsfolge oft *rückwärts* vom Wort zum Startsymbol.
Man spricht dann statt von Ableitungsschritten von Reduktionsschritten.

Eine Ableitung beginnt stets mit dem Startsymbol, aber bei einer Reduktion muss man aufpassen, dass man mit dem Startsymbol (und nicht in einer Sackgasse!) endet.

Reduktionsschritte

Eine Reduktion heißt Linksreduktion,
wenn für jeden Reduktionsschritt $\alpha \Leftarrow \beta$ gilt:
In $\alpha = w_1 \dots w_n$ ($w_s \in T \cup N$, $1 \leq s \leq n$) wird ein Wort $w_i \dots w_j$ ersetzt
und keine Ersetzung, die links davon beginnt ($w_k \dots w_l$ mit $k < i$),
kann man durch Reduktionsschritte bis zum Startsymbol fortsetzen.

Eine Linksreduktion entspricht einer Rechtsableitung, allerdings werden die Produktionen in umgekehrter Reihenfolge angewendet.

Beispiel

Für Grammatik G_1 mit $P_1 = \{S \to S + S \mid S * S \mid (S) \mid a\}$
und $w = (a + a) * a$ erhält man die Linksreduktion
$(a + a) * a \Leftarrow (S + a) * a \Leftarrow (S + S) * a \Leftarrow (S) * a \Leftarrow S * a \Leftarrow S * S \Leftarrow S$

Backus-Naur-Form: Dialekt für kontextfreie Grammatiken

Kontextfreie Grammatiken von Programmiersprachen werden oft in einem „Dialekt“ beschrieben, der Backus-Naur-Form (BNF):

- Oft benutzt man die Zeichenfolge `::=` für das Relationssymbol $\rightarrow$.
- Nichtterminale Symbole haben die Form `<name>` .
- Terminale Symbole sind einfache Zeichen (z.B. `+`, `-`) oder Tokenklassen (z.B. **`tokenname`**).

Die erweiterte BNF kann Wiederholungen oder Auslassungen von Zeichenketten in *einer* „Produktion“ beschreiben:

- Eine Produktion $A \rightarrow \alpha\{\beta\}\gamma$ beschreibt, dass man aus A Zeichenketten ableiten kann, in denen β beliebig oft (auch nullmal) auftritt, also $A \stackrel{*}{\Longrightarrow} \alpha\beta^i\gamma,\ i \geq 0$.
- Eine Produktion $A \rightarrow \alpha[\beta]\gamma$ beschreibt den Fall, dass β höchstens einmal auftritt, also $A \Rightarrow \alpha\gamma$ oder $A \Rightarrow \alpha\beta\gamma$.

Backus-Naur-Form: Dialekt für kontextfreie Grammatiken

Beispiel (Grammatik in BNF für Festpunktzahlen)

<Zahl>	::=	<Vorzeichen> <Ziffern> [. <Ziffern>]
<Vorzeichen>	::=	$+ \mid - \mid \varepsilon$
<Ziffern>	::=	<Ziffer> { <Ziffer>}
<Ziffer>	::=	$0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

reduzierte kontextfreie Grammatiken

Wir wollen überflüssige Teile einer kontextfreien Grammatik entfernen:

Definition

Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik.
$a \in T$ heißt **nützlich**, falls a in einer Produktion aus P vorkommt.
$A \in N$ heißt **erreichbar**, falls es Wörter α, β gibt mit $S \overset{*}{\Longrightarrow} \alpha A \beta$.
$A \in N$ heißt **produktiv**, falls es ein Terminalwort $u \in T^*$ gibt mit $A \overset{*}{\Longrightarrow} u$.

G' heißt **reduzierte kontextfreie Grammatik** zu G, falls $L(G') = L(G)$,
jedes terminale Symbol nützlich ist und
jedes nichtterminale Symbol erreichbar und produktiv ist.

Um eine reduzierte Grammatik zu erhalten, entfernt man insbesondere alle unerreichbaren und unproduktiven nichtterminalen Symbole sowie alle Produktionen, in denen diese Symbole vorkommen.

reduzierte kontextfreie Grammatiken

Beispiel (1)

In der Grammatik $G = (\{S, X, Y, Z\}, \{a, b\}, P, S)$ mit

$$P = \{S \to aX,\ X \to bS \mid aYbY,\ Y \to ba \mid aZ,\ Z \to aZX\}$$

ist Y offenbar produktiv. Dann sind auch X und S produktiv.
Z ist nicht produktiv.

reduzierte kontextfreie Grammatiken

Beispiel (1)

In der Grammatik $G = (\{S, X, Y, \cancel{Z}\}, \{a, b\}, P, S)$ mit

$P = \{S \to aX,\ X \to bS \mid aYbY,\ Y \to ba \mid \cancel{aZ},\ \cancel{Z \to aZX}\}$

ist Z nicht produktiv.

Da alle (verbliebenen) nichtterminalen Symbole erreichbar und alle terminalen Symbole nützlich sind, lautet die reduzierte Grammatik

$G' = (\{S, X, Y\}, \{a, b\}, P', S)$ mit
$P' = \{S \to aX,\ X \to bS \mid aYbY,\ Y \to ba\}$.

reduzierte kontextfreie Grammatiken

Beispiel (2)

In der Grammatik $G = (\{S, U, V, X, Z\}, \{a, b, c, d\}, P, S)$ mit

$$P = \{S \to SZ \mid Sa \mid b,\ U \to V,\ X \to c,\ V \to Vd \mid d,\ Z \to ZX\}$$

ist Z erreichbar, ebenso X. (S ist immer erreichbar.)
U und V sind nicht erreichbar.
Z ist nicht produktiv.

reduzierte kontextfreie Grammatiken

Beispiel (2)

In der Grammatik $G = (\{S, \cancel{U}, \cancel{V}, X, \cancel{Z}\}, \{a, b, c, d\}, P, S)$ mit

$P = \{S \to \cancel{SZ} \mid Sa \mid b, \cancel{U \to V}, X \to c, \cancel{V \to Vd \mid d}, \cancel{Z \to ZX}\}$

sind U und V nicht erreichbar.
Z ist nicht produktiv.
Jetzt ist X nicht mehr erreichbar!

reduzierte kontextfreie Grammatiken

Beispiel (2)

In der Grammatik $G = (\{S, \cancel{U}, \cancel{V}, \cancel{X}, \cancel{Z}\}, \{a, b, c, d\}, P, S)$ mit

$P = \{S \to \cancel{SZ} \mid Sa \mid b,\ \cancel{U \to V},\ \cancel{X \to c},\ \cancel{V \to Vd \mid d},\ \cancel{Z \to ZX}\}$

sind U und V nicht erreichbar.
Z ist nicht produktiv.
Jetzt ist X nicht mehr erreichbar!

Da c und d nicht nützlich sind, lautet die reduzierte Grammatik:
$G' = (\{S\}, \{a, b\}, P', S)$ mit $P' = \{S \to Sa \mid b\}$.

Um eine reduzierte Grammatik zu erhalten,
entfernt man daher zuerst die nicht produktiven und dann die nicht erreichbaren nichtterminalen Symbole und schließlich die nicht nützlichen terminalen Symbole.

Sprachen und kontextfreie Grammatiken

Bemerkungen

1. *Für eine kontextfreie Sprache L gibt es viele verschiedene kontextfreie Grammatiken G mit $L(G) = L$.*
2. *Es gibt formale Sprachen, für die es keine kontextfreie Grammatik gibt.*
3. *Es ist nicht entscheidbar, ob zwei kontextfreie Grammatiken dieselbe Sprache erzeugen.*

Reguläre Grammatiken

Definition

Eine Grammatik $G = (N, T, P, S)$ heißt regulär, wenn alle Produktionen die Form $A \to \alpha B$ oder $A \to \alpha$ mit $A, B \in N$, $\alpha \in T^*$ haben.

Die von einer regulären Grammatik G erzeugte Sprache $L(G)$ heißt reguläre Sprache.

Reguläre Grammatiken

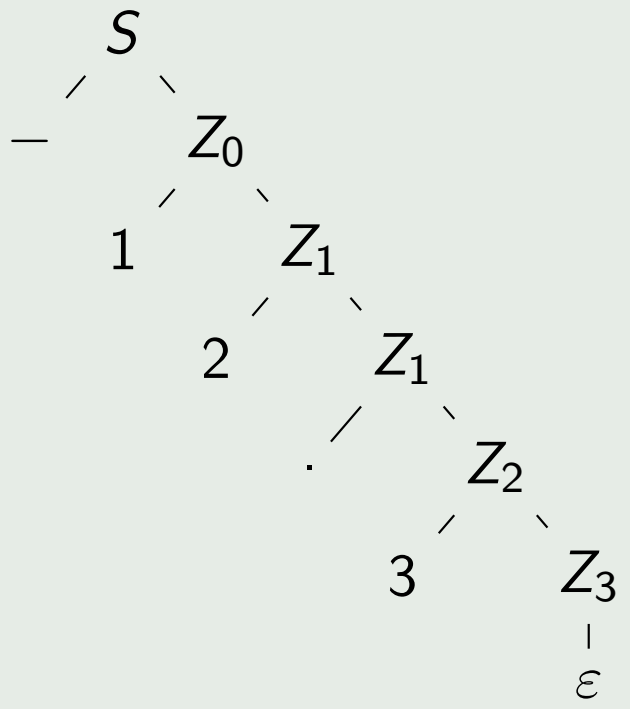
Bemerkungen

- *Jede endliche Sprache ist regulär!*
- *Jede reguläre Sprache ist kontextfrei!*
- *Man kann reguläre Grammatiken auch mit Produktionen definieren, die die Form $A \to B\alpha$ oder $A \to \alpha$ mit $A, B \in N$, $\alpha \in T^*$ haben. Beide Formen sind äquivalent, dürfen aber nicht gemeinsam auftreten.*
- *Es gibt kontextfreie Sprachen, für die es keine reguläre Grammatik gibt, z.B. $\{a^n b^n \mid n \geq 0\}$.*
- *Alle Ableitungen bzgl. einer regulären Grammatik sind Linksableitungen bzw. Rechtsableitungen.*
- *Die zugehörigen Ableitungsbäume sind besonders einfach.*

Reguläre Grammatiken

Beispiel (reguläre Grammatik für Festpunktzahlen)

$$
\begin{aligned}
&S \to +Z_0 \mid -Z_0 \mid Z_0,\\
&Z_0 \to 0Z_1 \mid 1Z_1 \mid 2Z_1 \mid 3Z_1 \mid 4Z_1 \mid 5Z_1 \mid 6Z_1 \mid 7Z_1 \mid 8Z_1 \mid 9Z_1,\\
&Z_1 \to 0Z_1 \mid 1Z_1 \mid 2Z_1 \mid 3Z_1 \mid 4Z_1 \mid 5Z_1 \mid 6Z_1 \mid 7Z_1 \mid 8Z_1 \mid 9Z_1 \mid \varepsilon \mid .Z_2,\\
&Z_2 \to 0Z_3 \mid 1Z_3 \mid 2Z_3 \mid 3Z_3 \mid 4Z_3 \mid 5Z_3 \mid 6Z_3 \mid 7Z_3 \mid 8Z_3 \mid 9Z_3,\\
&Z_3 \to 0Z_3 \mid 1Z_3 \mid 2Z_3 \mid 3Z_3 \mid 4Z_3 \mid 5Z_3 \mid 6Z_3 \mid 7Z_3 \mid 8Z_3 \mid 9Z_3 \mid \varepsilon
\end{aligned}
$$



- Höchstens ein Vorzeichen möglich
- Mindestens eine Ziffer (vor dem Dezimalpunkt) vorhanden
- Höchstens ein Dezimalpunkt möglich
- Mindestens eine Ziffer nach einem Dezimalpunkt vorhanden

Operationen auf Mengen von Wörtern

Definition

Wenn M, M_1 und M_2 Mengen von Wörtern über einem Alphabet T sind, dann ist

$$
\begin{aligned}
M_1 \cup M_2 &:= \{w \mid w \in M_1 \text{ oder } w \in M_2\}, \\
M_1\, M_2 &:= \{w \mid w = uv \text{ mit } u \in M_1 \text{ und } v \in M_2\}, \\
M^0 &:= \{\varepsilon\}, \\
M^1 &:= M, \\
M^i &:= M\, M^{i-1} \quad \text{für } i > 0, \\
M^* &:= \bigcup_{i=0}^{\infty} M^i, \\
M^+ &:= \bigcup_{i=1}^{\infty} M^i, \text{ d.h. } M^+ = M^* \setminus \{\varepsilon\}, \text{ falls } \varepsilon \notin M \text{ ist.}
\end{aligned}
$$

Operationen auf Mengen von Wörtern

Beispiel

Sei $M_1 = \{1, 10\}$ und $M_2 = \{0, 01, 001\}$.

$$\begin{aligned} M_1 \cup M_2 &= \{0, 1, 01, 10, 001\}, \\ M_1\, M_2 &= \{10, 100, 101, 1001, 10001\}, \\ M_1^* &= \{\varepsilon, 1, 10, 11, 101, 110, 111, \ldots\}, \\ M_1^+ &= \{1, 10, 11, 101, 110, 111, \ldots\}. \end{aligned}$$

Reguläre Ausdrücke

- Ein regulärer Ausdruck α über einem Alphabet T bezeichnet eine Menge von Wörtern $L(\alpha) \subseteq T^*$, also eine formale Sprache über T.
- Man definiert zunächst die einfachsten regulären Ausdrücke und beschreibt deren Wortmengen.
- Dann gibt man Operationen an, wie man aus einfachen Ausdrücken kompliziertere zusammensetzen kann und wie die zugehörigen Wortmengen zu bilden sind.

Reguläre Ausdrücke

Definition

Gegeben sei ein Alphabet T. Dann gilt:

0. Das Symbol $\emptyset$ ist ein regulärer Ausdruck für die Sprache $L(\emptyset) = \emptyset$.
1. Das Symbol ε ist ein regulärer Ausdruck für die Sprache $L(\varepsilon) = \{\varepsilon\}$.
2. Für jedes $a \in T$:
   Das Symbol a ist ein regulärer Ausdruck für die Sprache $L(a) = \{a\}$.

Sind α und β reguläre Ausdrücke für die Sprachen $L(\alpha)$ und $L(\beta)$, so ist

3. $\alpha \mid \beta$ ein regulärer Ausdruck für die Sprache $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$.
4. $\alpha\,\beta$ ein regulärer Ausdruck für die Sprache $L(\alpha\,\beta) = L(\alpha)\,L(\beta)$.
5. α^* ein regulärer Ausdruck für die Sprache $L(\alpha^*) = (L(\alpha))^*$.
6. α^+ ein regulärer Ausdruck für die Sprache $L(\alpha^+) = (L(\alpha))^+$.