TU WIEN Informatics

Reguläre und kontextfreie Sprachen

Grundzüge digitaler Systeme (192.134)

Vortrag von: Gernot Salzer

Fakultät für Informatik
TU Wien
E-Mail: gds@list.tuwien.ac.at

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Operationen auf formalen Sprachen

Σ Alphabet, d.h., endliche, nicht-leere Menge atomarer Symbole
w Wort über Σ, (endliche) Folge von Zeichen aus dem Alphabet Σ
ε Leerwort
Σ^* Menge aller endlichen Wörter über Σ (inklusive Leerwort)
$w \cdot w' = ww'$ Verkettung der Wörter $w, w' \in \Sigma^*$

Seien $L, L' \subseteq \Sigma^*$ zwei Sprachen.

$$L \cup L' = \{ w \mid w \in L \text{ oder } w \in L' \} \qquad \text{Vereinigung}$$

$$L \cdot L' = \{ w \cdot w' \mid w \in L, w' \in L' \} \qquad \text{Verkettung}$$

$$L^0 = \{\varepsilon\} \qquad \text{Potenzen}$$
$$L^{n+1} = L \cdot L^n \quad (n \geq 0)$$

$$L^+ = \bigcup_{n \geq 1} L^n$$
$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^+ = \{\varepsilon\} \cup L^+ \qquad \text{Kleene-Stern}$$

Verkettung

$\{\mathtt{a},\mathtt{b}\} \cdot \{\mathtt{b},\mathtt{c},\mathtt{d}\} = \{\mathtt{ab},\mathtt{ac},\mathtt{ad},\mathtt{bb},\mathtt{bc},\mathtt{bd}\}$
$\{\mathtt{b},\mathtt{c},\mathtt{d}\} \cdot \{\mathtt{a},\mathtt{b}\} = \{\mathtt{ba},\mathtt{bb},\mathtt{ca},\mathtt{cb},\mathtt{da},\mathtt{db}\}$

$(\{\mathtt{a},\mathtt{b}\} \cdot \{1,2\}) \cdot \{\#,\$\} = \{\mathtt{a1\#},\mathtt{a1\$},\mathtt{a2\#},\mathtt{a2\$},\mathtt{b1\#},\mathtt{b1\$},\mathtt{b2\#},\mathtt{b2\$}\}$
$= \{\mathtt{a},\mathtt{b}\} \cdot (\{1,2\} \cdot \{\#,\$\})$

$\{\mathtt{a},\mathtt{b}\} \cdot \{\varepsilon\} = \{\mathtt{a} \cdot \varepsilon, \mathtt{b} \cdot \varepsilon\} = \{\mathtt{a},\mathtt{b}\} = \{\varepsilon \cdot \mathtt{a}, \varepsilon \cdot \mathtt{b}\} = \{\varepsilon\} \cdot \{\mathtt{a},\mathtt{b}\}$

$\{\mathtt{a},\mathtt{b}\} \cdot \{\} = \{\} \cdot \{\mathtt{a},\mathtt{b}\} = \{\}$

$\{\varepsilon\} \cdot \{\varepsilon\} = \{\varepsilon\}$
$\{\} \cdot \{\} = \{\varepsilon\} \cdot \{\} = \{\} \cdot \{\varepsilon\} = \{\}$

Beobachtungen:

- Sprachverkettung ist nicht kommutativ.
- Sprachverkettung ist assoziativ.
- $\{\varepsilon\}$ ist neutrales Element bzgl. Sprachverkettung.
- $\{\}$ ist Nullelement bzgl. Sprachverkettung.

Potenzen von $\{a, 42\}$

$$
\begin{aligned}
L &= \{a, 42\} \\
L^0 &= \{\varepsilon\} \\
L^1 &= L \cdot L^0 = L \cdot \{\varepsilon\} = L = \{a, 42\} \\
L^2 &= L \cdot L^1 = L \cdot L = \{aa, a42, 42a, 4242\} \\
L^3 &= L \cdot L^2 = \{aaa, aa42, a42a, a4242, 42aa, 42a42, 4242a, 424242\} \\
&\vdots \\
L^+ &= \bigcup_{n \geq 1} L^n = L^1 \cup L^2 \cup L^3 \cup \cdots \\
&= \{a, 42, aa, a42, 42a, 4242, aaa, aa42, a42a, a4242, 42aa, \dots\} \\
L^* &= \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots \\
&= \{\varepsilon, a, 42, aa, a42, 42a, 4242, aaa, aa42, a42a, a4242, 42aa, \dots\}
\end{aligned}
$$

Potenzen eines Alphabets Σ

$\Sigma^0 = \{\varepsilon\}$	$\Sigma^1 = \Sigma$
Σ^n	alle Σ-Wörter der Länge n (d.h., mit n Symbolen)
$\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$	alle Σ-Wörter ohne Leerwort
$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$	alle Σ-Wörter mit Leerwort

Rechengesetze für formale Sprachen

A, B, C ... formale Sprachen

Vereinigung

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$\{\} \cup A = A$$
$$A \cup \{\} = A$$
$$A \cup B = B \cup A$$
$$A \cup A = A$$

Verkettung

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$\{\varepsilon\} \cdot A = A$$
$$A \cdot \{\varepsilon\} = A$$
$$\{\} \cdot A = \{\}$$
$$A \cdot \{\} = \{\}$$

Distributivität

$$A \cdot (B \cup C) = (A \cdot B) \cup (A \cdot C)$$
$$(B \cup C) \cdot A = (B \cdot A) \cup (C \cdot A)$$

Kleene-Stern und -Plus

$$(A^*)^* = A^*$$
$$A^* \cdot A = A^+$$
$$A \cdot A^* = A^+$$
$$(A \cup \{\varepsilon\})^* = A^*$$
$$A^+ \cup \{\varepsilon\} = A^*$$

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Reguläre Sprachen

Alle Sprachen, die aus einem Alphabet mit Hilfe von Vereinigung, Verkettung und Stern gebildet werden können.

Anwendungen:

- Betriebssytem-Shells: DOS („Wildcards“), UNIX-Shells (wie `sh`, `csh`, `ash`, `bash`, `zsh`), ...
- UNIX Kommandozeilenprogramme: `grep`, `awk`, `ed`, `sed`, ...
- Editoren: `vi`, `emacs`, ...
- Compilerbau: *Tokens* bilden reguläre Sprache, die durch sog. Scanner (Lexer) wie `lex` oder `flex` verarbeitet werden.
- Programmiersprachen: PERL, TCL, PHP, PYTHON, RUBY, R, JAVA, JAVASCRIPT, .NET-Sprachen, ...
- Websprachen: XML Schema, XQuery, XPath, DTDs, ...
- Datenbanken: MySQL, Oracle, PostgreSQL, ...
- ...

Regulären Sprachen über einem Alphabet

Die Menge der regulären Sprachen über Σ, $\mathcal{L}_{\text{reg}}(\Sigma)$, ist die kleinste Menge, sodass gilt:

- $\{\}$, $\{\varepsilon\}$ und $\{s\}$ sind reguläre Sprachen (für alle $s \in \Sigma$).
- Wenn L und L' reguläre Sprachen sind, dann auch $L \cup L'$, $L \cdot L'$ und L^*.

Reellen Numerale: reguläre Sprache über $\Sigma = \{0, \ldots, 9, ., \mathtt{E}, +, -\}$

$$\begin{aligned} \mathit{real} &= \mathit{digit} \cdot \mathit{digit}^* \cdot \{.\} \cdot \mathit{digit}^* \cdot (\{\varepsilon\} \cup \mathit{scale}) \\ \mathit{scale} &= \{\mathtt{E}\} \cdot \{+, -, \varepsilon\} \cdot \mathit{digit} \cdot \mathit{digit}^* \\ \mathit{digit} &= \{0, \ldots, 9\} = \{0\} \cup \cdots \cup \{9\} \end{aligned}$$

Wichtig: Unterscheide Symbole des Alphabets von Meta-Symbolen!

$0, \ldots, 9, ., \mathtt{E}, +, -$ ... Symbole des Alphabets
ε, *real*, *scale*, *digit* ... Meta-Symbole, Abkürzungen

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Reguläre Ausdrücke

Ausdrücke wie $digit \cdot digit^*$ und $digit^* \cdot digit$ sind ununterscheidbar: Beides sind semantische Beschreibungen der Menge aller Ziffernfolgen. Um Aussagen über die Form der Ausdrücke treffen zu können, benötigen wir eine formale Sprache.

Reguläre Ausdrücke (algebraische Notation)

Die regulären Ausdrücke über Σ sind die kleinste Menge, für die gilt:

- $\emptyset$, ε und s sind reguläre Ausdrücke (für alle Symbole $s \in \Sigma$).
- Sind X und Y reguläre Ausdrücke, dann auch $(X + Y)$, (XY) und X^*.

Vereinfachte Klammerung: $+$ bindet am schwächsten, $*$ am stärksten. Keine Klammern bei gleichartigen Operatoren (wegen Assoziativität).

Die Sprache $\mathcal{L}(X)$ zu einem regulären Ausdruck X ist definiert durch:

$$\begin{aligned} \mathcal{L}(\emptyset) &= \{\} & \mathcal{L}(X + Y) &= \mathcal{L}(X) \cup \mathcal{L}(Y) \\ \mathcal{L}(\varepsilon) &= \{\varepsilon\} & \mathcal{L}(XY) &= \mathcal{L}(X) \cdot \mathcal{L}(Y) \\ \mathcal{L}(s) &= \{s\} \quad \text{für } s \in \Sigma & \mathcal{L}(X^*) &= (\mathcal{L}(X))^* \end{aligned}$$

Regulärer Ausdruck für die reellen Numerale

$R = DD^* \,.\, D^*(\varepsilon + S)$
$S = \mathtt{E}(+ + - + \varepsilon)DD^*$
$D = 0 + 1 + \cdots + 9$

(R, S und D sind Abkürzungen für die jeweiligen regulären Ausdrücke.)

Die zugehörigen Sprachen:

$$\begin{aligned}\mathcal{L}(D) &= \mathcal{L}(0 + 1 + \cdots + 9)\\ &= \mathcal{L}(0) \cup \mathcal{L}(1) \cup \cdots \cup \mathcal{L}(9)\\ &= \{0\} \cup \{1\} \cup \cdots \cup \{9\}\\ &= \mathit{digit}\end{aligned}$$

$$\begin{aligned}\mathcal{L}(S) &= \mathcal{L}(\mathtt{E}(+ + - + \varepsilon)DD^*)\\ &= \mathcal{L}(\mathtt{E}) \cdot \mathcal{L}(+ + - + \varepsilon) \cdot \mathcal{L}(D) \cdot \mathcal{L}(D^*)\\ &= \{\mathtt{E}\} \cdot (\mathcal{L}(+) \cup \mathcal{L}(-) \cup \mathcal{L}(\varepsilon)) \cdot \mathit{digit} \cdot \mathcal{L}(D)^*\\ &= \{\mathtt{E}\} \cdot (\{+\} \cup \{-\} \cup \{\varepsilon\}) \cdot \mathit{digit} \cdot \mathit{digit}^*\\ &= \mathit{scale}\end{aligned}$$

$$\mathcal{L}(R) = \cdots = \mathit{real}$$

Zwei reguläre Ausdrücke X und Y heißen äquivalent, geschrieben $X = Y$, wenn $\mathcal{L}(X) = \mathcal{L}(Y)$ gilt.

$((a + b)^* + \varepsilon)^* = (a + b)^*$

$$
\begin{aligned}
\mathcal{L}(((a + b)^* + \varepsilon)^*) &= \cdots \\
&= (\{a, b\}^* \cup \{\varepsilon\})^* \\
&= (\{a, b\}^*)^* \qquad \text{da } \varepsilon \in L^* \text{ für alle } L \\
&= (\{a, b\}^*)^0 \cup (\{a, b\}^*)^1 \cup (\{a, b\}^*)^2 \cup \cdots \\
&= \{a, b\}^* \cup (\{a, b\}^*)^0 \cup (\{a, b\}^*)^2 \cup \cdots \\
&= \{a, b\}^* \qquad \text{da } L^* \text{ alle Wörter über } L \text{ enthält} \\
&= \cdots \\
&= \mathcal{L}((a + b)^*)
\end{aligned}
$$

Reguläre Ausdrücke in EBNF-Notation

EBNF ... Erweiterte Backus-Naur-Form (Formalismus zur Beschreibung der Syntax von Programmiersprachen, der reguläre Ausdrücke zulässt)

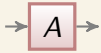
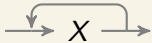
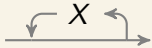
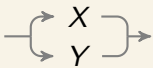
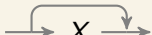
XY	$X \cdot Y$	Aufeinanderfolge
$X \mid Y$	$X \cup Y$	Alternativen
$[X]$	$\{\varepsilon\} \cup X$	Option
$\{X\}$	X^*	Wiederholung
(X)	(X)	Gruppierung
$"s"$	$\{s\}$	Symbol

Reelle Numerale in EBNF-Notation

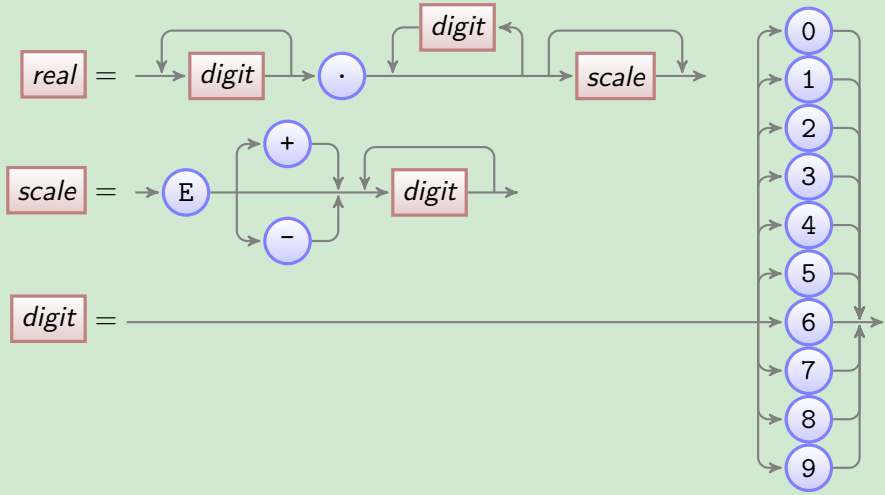
```
real  = digit {digit} "." {digit} [scale]
scale = "E" ["+" | "-"] digit {digit}
digit = "0" | "1" | "2" | ··· | "9"
```

Reguläre Ausdrücke als Syntaxdiagramme

Syntaxdiagramme ... graphische Form der EBNF

	A	Abkürzung
	$\{s\}$	Symbol
	X^+	Wiederholung ≥ 1
	X^*	Wiederholung ≥ 0
	$X \cdot Y$	Aufeinanderfolge
	$X \cup Y$	Alternativen
	$\{\varepsilon\} \cup X$	Option

Reelle Numerale als Syntaxdiagramm



Reguläre Ausdrücke im informatischen Alltag

UNIX := *„30 definitions of regular expressions living under one roof“* [Donald E. Knuth]

`grep -E -e "`*regexp*`"` *file*

liefert alle Zeilen der Datei *file*, die eine Zeichenkette enthalten, die dem regulären Ausdruck *regexp* (in POSIX ERE Syntax) entspricht.

Verschiedene „Standards“:

- POSIX Basic Regular Expressions
- POSIX Extended Regular Expressions
- PERL Regular Expressions
- ...

Posix Extended Regular Expressions (Ere)

regexp	trifft zu auf
\s	Zeichen s
s	s, falls kein Sonderzeichen
.	alle Zeichen
^	Zeilenanfang
$	Zeilenende
$[s_1 \cdots s_n]$	ein Zeichen aus $\{s_1, \ldots, s_n\}$
$[\hat{}\, s_1 \cdots s_n]$	alle Zeichen außer $s_1, \ldots, s_n$
(X)	X

regexp	trifft zu auf
XY	X gefolgt von Y
$X \mid Y$	X oder Y
$X*$	≥ 0 Mal X
$X+$	≥ 1 Mal X
$X?$	≤ 1 Mal X
$X\{i\}$	i Mal X
$X\{i,\}$	$\geq i$ Mal X
$X\{i,j\}$	i bis j Mal X

Reelle Numerale als Ere

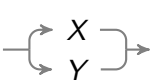
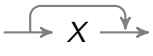
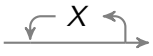
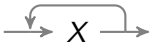
$digit = \{0, \ldots, 9\}$ `[0-9]`

$scale = \{\mathrm{E}\} \cdot \{+, -, \varepsilon\} \cdot digit \cdot digit^*$ `E[+-]?[0-9]+`

$real = digit \cdot digit^* \cdot \{.\} \cdot digit^* \cdot (\{\varepsilon\} \cup scale)$ `^[0-9]+\.[0-9]*(E[+-]?[0-9]+)?$`

`[0-9]` ... Kurzform von `[0123456789]`; analog `[a-zA-Z]` für Buchstaben

Zusammenfassung der Notationen

Reg. Sprache	Algebra	EBNF	Syntaxdiagramm	POSIX ERE	
A	A	A			Abkürzung
$\{\}$	$\emptyset$				Leersprache
$\{\varepsilon\}$	ε				Leerwortsprache
$\{s\}$	s	"s"		\s	Terminalsymbol
$X \cdot Y$	XY	XY		XY	Aufeinanderfolge
$X \cup Y$	$X + Y$	$X \mid Y$		$X \mid Y$	Alternativen
$\{\varepsilon\} \cup X$	$\varepsilon + X$	$[X]$		$X?$	Option
X^*	X^*	$\{X\}$		$X*$	Wiederholung ≥ 0
X^+	XX^*, X^+	$X\{X\}$		$X+$	Wiederholung ≥ 1
(X)	(X)	(X)		(X)	Gruppierung

Weitere POSIX-Notationen:

POSIX ERE	Bedeutung	Reguläre Sprache
s	s, falls kein Sonderzeichen	$\{s\}$
.	alle Zeichen	Σ
^	Zeilenanfang	
$	Zeilenende	
$[s_1 \cdots s_n]$	eines der Zeichen s_i	$\{s_1, \ldots, s_n\}$
$[\hat{}\, s_1 \cdots s_n]$	alle Zeichen außer $s_1, \ldots, s_n$	$\Sigma - \{s_1, \ldots, s_n\}$
$X\{i\}$	i Mal X	X^i
$X\{i,\}$	$\geq i$ Mal X	$X^i \cdot X^*$
$X\{i,j\}$	i bis j Mal X	$X^i \cup X^{i+1} \cup \cdots \cup X^j$

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Eigenschaften regulärer Sprachen

Abschlusseigenschaften reguläre Sprachen

Reguläre Sprachen sind abgeschlossen gegenüber

- Vereinigung, Verkettung und Stern (warum?)
- Durchschnitt, Komplement, Differenz, Homomorphismen, Quotientenbildung, ...

Entscheidbarkeit eines Problems: Es gibt ein Verfahren (einen Algorithmus), das für jede Eingabe die richtige Antwort ja/nein liefert.

Nicht entscheidbar: Halteproblem Ihrer Lieblingsprogrammiersprache

- Gegeben ein Programm mit einer Eingabe, wird es anhalten?

Entscheidbare Probleme regulärer Sprachen

- Gegeben ein Wort w und einen regulären Ausdruck X, gilt $w \in \mathcal{L}(X)$? (Wortproblem)
- Gegeben zwei reguläre Ausdrücke X und Y, gilt $\mathcal{L}(X) = \mathcal{L}(Y)$? (Äquivalenzproblem)
- Gegeben einen regulären Ausdruck X, ist $\mathcal{L}(X)$ leer/endlich/unendlich?

Ausdruckskraft regulärer Sprachen

Die regulären Sprachen sind genau jene, die von endlichen Automaten akeptiert werden, d.h.:

- Zu jedem regulären Ausdruck X gibt es einen endlichen Automaten $\mathcal{A}$, sodass $\mathcal{L}(\mathcal{A}) = \mathcal{L}(X)$ gilt.
- Zu jedem endlichen Automaten $\mathcal{A}$ gibt es einen regulären Ausdruck X, sodass $\mathcal{L}(X) = \mathcal{L}(\mathcal{A})$ gilt.

Nicht regulär sind Sprachen, deren Analyse ein unbegrenztes Gedächtnis erfordert:

- Klammerausdrücke: $\{\texttt{()}, \texttt{(())}, \texttt{()()}, \texttt{((()))}, \texttt{(())()}, \texttt{()()()}, \ldots\}$
- $\{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 0 \,\} = \{\varepsilon, \mathtt{ab}, \mathtt{aabb}, \mathtt{aaabbb}, \ldots\}$
- $\{\, \mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0 \,\} = \{\varepsilon, \mathtt{abc}, \mathtt{aabbcc}, \mathtt{aaabbbccc}, \ldots\}$
- Palindrome: Wörter, die identisch mit ihrem Spiegelbild sind.
  $\{\mathtt{otto}, \mathtt{anna}, \mathtt{reliefpfeiler}, \text{ o genie der herr ehre dein ego}, \ldots\}$
- Doppelwörter: $\{\, ww \mid w \in \Sigma^* \,\}$ (falls $|\Sigma| > 1$)

Formale Sprachen – Übersicht

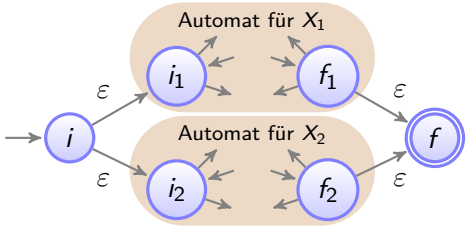
1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Vom regulären Ausdruck zum Automaten

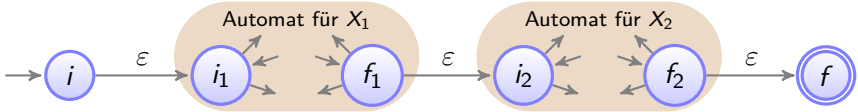
Automat für $\emptyset$:

Automat für $s \in \Sigma \cup \{\varepsilon\}$:

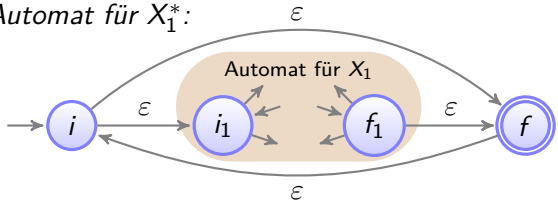
Automat für $X_1 + X_2$:



Automat für $X_1 X_2$:

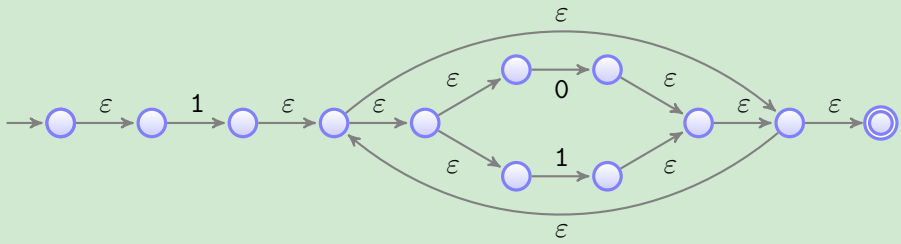


Automat für X_1^*:

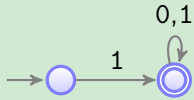


Automat für Binärnumerale ohne führende Null

Regulärer Ausdruck: $1(0+1)^*$



Minimaler deterministischer Automat:



Manuelle Konstruktion von Automaten:

1. Konstruiere die Automaten zu einfachen Teilsprachen „durch Hinschauen“.
2. Verwende die allgemeine Konstruktion mit ε-Übergängen für undurchsichtige Situationen.

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Vom Automaten zum regulären Ausdruck

R ... Menge der regulären Ausdrücke über Σ

Verallgemeinerter endlicher Automat

... wird beschrieben durch ein 5-Tupel $\mathcal{A} = \langle Q, \Sigma, \delta, i, f \rangle$, wobei

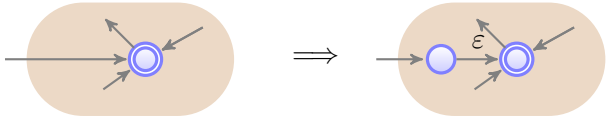
- Q ... endliche Zustandsmenge
- Σ ... Eingabealphabet
- $\delta\colon (Q - \{f\}) \times (Q - \{i\}) \to R$ ... Übergangsfunktion
- $i \in Q$ ... Anfangszustand
- $f \in Q$, $f \neq i$ ... Endzustand

Unterschiede zu „normalen“ Automaten:

- keine Übergänge in den Anfangszustand;
- nur ein Endzustand, der nicht Anfangszustand ist;
- keine Übergänge weg vom Endzustand;
- nur ein Übergang zwischen je zwei Zuständen;
- Übergänge beschriftet mit regulären Ausdrücken.

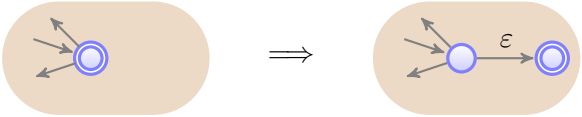
Umwandlung eines endlichen Automaten in einen verallgemeinerten

- Übergänge in den Anfangszustand oder Anfangszustand ist Endzustand:



- Mehrere Endzustände:

- Übergänge weg vom Endzustand:



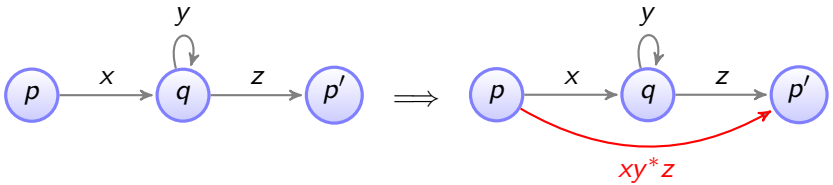
- Mehrere Übergänge zwischen zwei Zuständen:

Vom verallgemeinerten Automaten zum regulären Ausdruck

Gegeben: Verallgemeinerter Automat $\mathcal{A} = \langle Q, \Sigma, \delta, i, f \rangle$

Für jeden Zustand $q \in Q - \{i, f\}$ führe folgende Schritte durch:

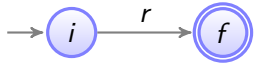
1. Füge zwischen allen Nachbarn p, p' von q neue Übergänge hinzu:



   (x, y und z bezeichnen reguläre Ausdrücke.)

2. Entferne q und alle Kanten von und nach q aus dem Automaten.

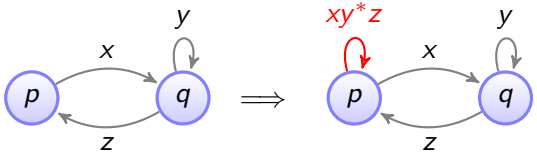
Restautomat:



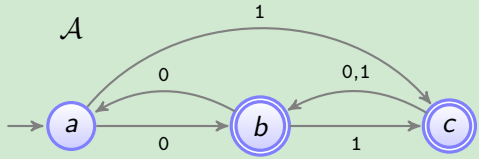
Ergebnis: r ist ein regulärer Ausdruck mit $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$.

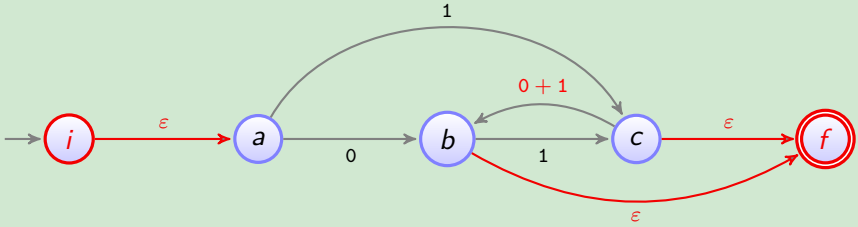
Anmerkungen:

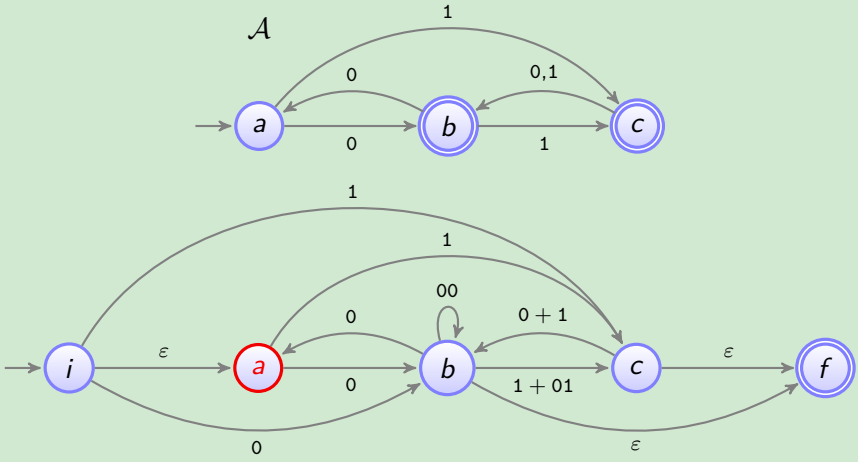
- Falls p und p' derselbe Knoten sind, erhält man:

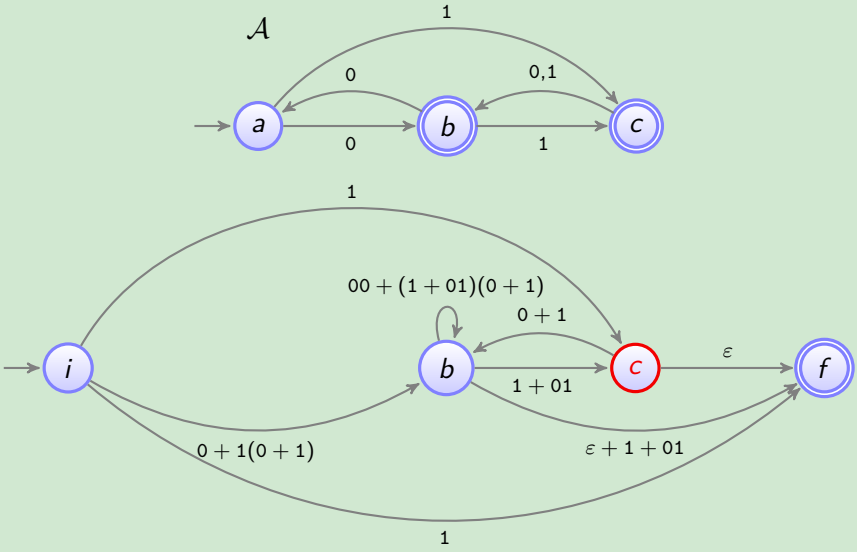


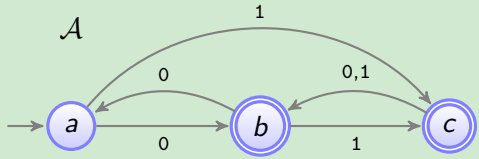
- Falls die Schleife mit dem Ausdruck y nicht existiert, entfällt y^*.
- Falls der Übergang p–p' bereits existiert, wird xy^*z addiert.

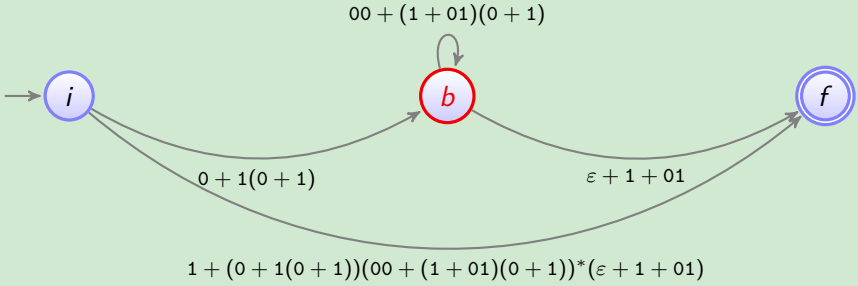

$\mathcal{A}$
1
0
0,1
a
b
c
0
1

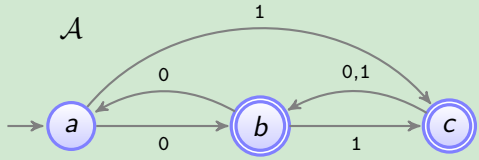

1
0 + 1
i
ε
a
b
c
ε
f
0
1
ε

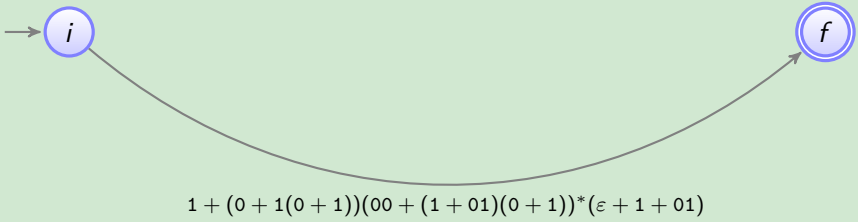

$\mathcal{A}$
1
0
0,1
a
b
c
0
1
1
1
00
0
0 + 1
ε
a
b
c
ε
i
f
0
1 + 01
0
ε


$\mathcal{A}$
1
0
0,1
a
b
c
0
1
1
$00 + (1 + 01)(0 + 1)$
$0 + 1$
i
b
c
ε
f
$1 + 01$
$0 + 1(0 + 1)$
$\varepsilon + 1 + 01$
1


$\mathcal{A}$
1
0
0,1
a
b
c
0
1


$00 + (1 + 01)(0 + 1)$
i
b
f
$0 + 1(0 + 1)$
$\varepsilon + 1 + 01$
$1 + (0 + 1(0 + 1))(00 + (1 + 01)(0 + 1))^*(\varepsilon + 1 + 01)$





$r = 1 + (0 + 10 + 11)(00 + 10 + 11 + 010 + 011)^*(\varepsilon + 1 + 01)$

Es gilt: $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$.

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Grenzen regulärer Sprachen

Was reguläre Ausdrücke und endliche Automaten beschreiben können:

- lexikale Elemente in Programmiersprachen: Identifier, Numerale, ...
- Morphologie von Worten in natürlichen Sprachen: korrekte Fall-/Zahl-/Zeitendungen, ...
- Systeme, die nur ein endliches Gedächtnis besitzen

Was sie nicht beschreiben können:

- geklammerte Ausdrücke in Programmiersprachen
- geschachtelte Programmstrukturen wie
  `if ... while ... if ... endif ... endwhile ... endif`
- Schachtelung von Haupt- und Nebensätzen in natürlichen Sprachen
- Systeme mit einem (potentiell) unendlichen Speicher

Sprache der wohlgeklammerten Ausdrücke ist nicht regulär

$\{(), (()), ()(), ((())), (())(), ()()(), \ldots\}$

$\{ a^n b^n \mid n \geq 0 \}$ ist nicht regulär

$\{\varepsilon, ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, \ldots\}$

Kontextfreie Grammatiken

- Menge von Ersetzungsregeln
- 2 Arten von Symbolen: Terminale, Nonterminale
- Ausgehend vom Start-Nonterminal werden Nonterminale solange ersetzt, bis nur noch Terminale bleiben.

$$
\begin{array}{ll}
\mathit{Satz} \to \mathit{HwP}\ \mathit{ZwP} & \mathit{Art} \to \texttt{der} \mid \texttt{den} \\
\mathit{HwP} \to \mathit{Art}\ \mathit{Hw} & \mathit{Hw} \to \texttt{Hund} \mid \texttt{Mond} \\
\mathit{ZwP} \to \mathit{Hzw}\ \mathit{HwP}\ \mathit{Zw} & \mathit{Hzw} \to \texttt{wird} \\
 & \mathit{Zw} \to \texttt{anbellen}
\end{array}
$$

$$
\begin{aligned}
\mathit{Satz} &\Rightarrow_p \mathit{HwP}\ \ \mathit{ZwP} \\
&\Rightarrow_p \mathit{Art}\ \ \mathit{Hw}\ \ \mathit{Hzw}\ \ \mathit{HwP}\ \ \mathit{Zw} \\
&\Rightarrow_p \texttt{der}\ \ \texttt{Hund}\ \ \texttt{wird}\ \ \mathit{Art}\ \ \mathit{Hw}\ \ \texttt{anbellen} \\
&\Rightarrow_p \texttt{der}\ \ \texttt{Hund}\ \ \texttt{wird}\ \ \texttt{den}\ \ \texttt{Mond}\ \ \texttt{anbellen}
\end{aligned}
$$

Generative Grammatiken gehen zurück auf *Noam Chomsky* (Linguist, Philosoph, Aktivist; *1928).

Kontextfreie Grammatik

... wird beschrieben durch ein 4-Tupel $G = \langle V, T, P, S \rangle$, wobei

- V ... Menge der Nonterminalsymbole (Variablen)
- T ... Menge der Terminalsymbole ($V \cap T = \{\}$)
- $P \subseteq V \times (V \cup T)^*$ ... Produktionen
- $S \in V$ ... Startsymbol

Schreibweise: $A \to w$ statt $(A, w) \in P$

$A \to w_1 \mid \cdots \mid w_n$ statt $A \to w_1, \ldots, A \to w_n$

Ableitbarkeit

... in einem Schritt:

$x\,A\,y \Rightarrow x\,w\,y$, falls $A \to w \in P$ und $x, y \in (V \cup T)^*$.

... in mehreren Schritten:

$u \overset{*}{\Rightarrow} v$, falls

- $u = v$, oder
- $u \Rightarrow u'$ und $u' \overset{*}{\Rightarrow} v$ für ein Wort $u' \in (V \cup T)^*$.

Von Grammatik G generierte Sprache

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

Grammatiken G_1 und G_2 heißen äquivalent, wenn $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ gilt.

$G = \langle \{S\}, \{\mathtt{a}, \mathtt{b}\}, \{S \to \varepsilon \mid \mathtt{a}\,S\,\mathtt{b}\}, S \rangle$

$S \overset{*}{\Rightarrow} \mathtt{aabb}$, weil $\quad S \Rightarrow \mathtt{a}\,S\,\mathtt{b} \Rightarrow \mathtt{aa}\,S\,\mathtt{bb} \Rightarrow \mathtt{aa}\,\varepsilon\,\mathtt{bb} = \mathtt{aabb}$

$\mathcal{L}(G) = \{ \mathtt{a}^n\mathtt{b}^n \mid n \geq 0 \} = \{\varepsilon, \mathtt{ab}, \mathtt{aabb}, \mathtt{aaabbb}, \ldots\}$

Kontextfreie Sprachen

Eine Sprache heißt kontextfrei, wenn es eine kontextfreie Grammatik gibt, die sie generiert.

Verschiedene Ableitbarkeitsbegriffe

Linksableitbarkeit: $x\,A\,y \Rightarrow_L x\,w\,y$ falls $A \to w \in P$ und $x \in T^*$
(In jedem Schritt wird die linkeste Variable ersetzt.)
Rechtsableitbarkeit: $x\,A\,y \Rightarrow_R x\,w\,y$ falls $A \to w \in P$ und $y \in T^*$
(In jedem Schritt wird die rechteste Variable ersetzt.)
Parallelableitbarkeit: $x_0\,A_1\,x_1 \cdots A_n\,x_n \Rightarrow_P x_0\,w_1\,x_1 \cdots w_n\,x_n$
falls $A_1 \to w_1, \ldots, A_n \to w_n \in P$ und $x_0, \ldots, x_n \in T^*$
(In jedem Schritt werden alle Variablen ersetzt.)

- $\overset{*}{\Rightarrow}_L$, $\overset{*}{\Rightarrow}_R$ und $\overset{*}{\Rightarrow}_P$ sind eingeschränkte Ableitungsrelationen:
  Manche Wörter $w \in (V \cup T)^*$ sind mit $\overset{*}{\Rightarrow}$ herleitbar ($S \overset{*}{\Rightarrow} w$),
  aber nicht mit diesen Relationen.
- Sie können aber jedes Wort der Sprache ableiten. Für alle $w \in T^*$ gilt:
  $S \overset{*}{\Rightarrow} w$ gdw. $S \overset{*}{\Rightarrow}_L w$ gdw. $S \overset{*}{\Rightarrow}_R w$ gdw. $S \overset{*}{\Rightarrow}_P w$

$$\begin{aligned}\mathcal{L}(G) &= \{\, w \in T^* \mid S \overset{*}{\Rightarrow} w \,\} \quad = \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_L w \,\} \\ &= \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_R w \,\} = \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_P w \,\}\end{aligned}$$

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Kontextfreie Grammatiken – Beispiele

Syntax aussagenlogischer Formeln

$G = \langle \{Fm, Op, Var\}, T, P, Fm \rangle$, wobei

$T = \{\top, \bot, \neg, (,), \wedge, \uparrow, \vee, \downarrow, \Leftrightarrow, \not\Leftrightarrow, \Rightarrow, \Leftarrow\} \cup \mathcal{V}$

$$P = \{\ Fm \to Var \mid \top \mid \bot \mid \neg Fm \mid (\ Fm\ Op\ Fm\ )\ ,$$
$$Op \to \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \Leftrightarrow \mid \not\Leftrightarrow \mid \Rightarrow \mid \Leftarrow\ ,$$
$$Var \to \mathtt{A} \mid \mathtt{B} \mid \mathtt{C} \mid \cdots\ \}$$

$((\mathtt{A} \uparrow \mathtt{B}) \uparrow (\mathtt{A} \uparrow \mathtt{B}))$ ist eine aussagenlogische Formel, weil:

$$\begin{aligned}
Fm &\Rightarrow_L (\ Fm\ Op\ Fm\ ) & &\Rightarrow_L (\ (\ Fm\ Op\ Fm\ )\ Op\ Fm\ ) \\
&\Rightarrow_L (\ (\ Var\ Op\ Fm\ )\ Op\ Fm\ ) & &\Rightarrow_L (\ (\ \mathtt{A}\ Op\ Fm\ )\ Op\ Fm\ ) \\
&\Rightarrow_L (\ (\ \mathtt{A} \uparrow Fm\ )\ Op\ Fm\ ) & &\Rightarrow_L (\ (\ \mathtt{A} \uparrow Var\ )\ Op\ Fm\ ) \\
&\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ )\ Op\ Fm\ ) & &\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow Fm\ ) \\
&\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ Fm\ Op\ Fm\ )\ ) & &\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ Var\ Op\ Fm\ )\ ) \\
&\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ \mathtt{A}\ Op\ Fm\ )\ ) & &\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ \mathtt{A} \uparrow Fm\ )\ ) \\
&\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ \mathtt{A} \uparrow Var\ )\ ) & &\Rightarrow_L (\ (\ \mathtt{A} \uparrow \mathtt{B}\ ) \uparrow (\ \mathtt{A} \uparrow \mathtt{B}\ )\ )
\end{aligned}$$

Kontextfreie Grammatiken – Beispiele

Syntax aussagenlogischer Formeln

$G = \langle \{Fm, Op, Var\}, T, P, Fm \rangle$, wobei

$T = \{\top, \bot, \neg, (,), \wedge, \uparrow, \vee, \downarrow, \Leftrightarrow, \not\Leftrightarrow, \Rightarrow, \Leftarrow\} \cup \mathcal{V}$

$P = \{\ Fm \to Var \mid \top \mid \bot \mid \neg Fm \mid (\ Fm\ Op\ Fm\ )\ ,$
$\quad Op \to \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \Leftrightarrow \mid \not\Leftrightarrow \mid \Rightarrow \mid \Leftarrow\ ,$
$\quad Var \to \mathtt{A} \mid \mathtt{B} \mid \mathtt{C} \mid \cdots\ \}$

$((\mathtt{A} \uparrow \mathtt{B}) \uparrow (\mathtt{A} \uparrow \mathtt{B}))$ ist eine aussagenlogische Formel, weil:

$Fm \Rightarrow_P (\ Fm\ Op\ Fm\ )$
$\Rightarrow_P (\ (\ Fm\ Op\ Fm\ ) \uparrow (\ Fm\ Op\ Fm\ )\ )$
$\Rightarrow_P (\ (\ Var \uparrow Var\ ) \uparrow (\ Var \uparrow Var\ )\ )$
$\Rightarrow_P (\ (\mathtt{A} \uparrow \mathtt{B}) \uparrow (\mathtt{A} \uparrow \mathtt{B})\ )$

Kontextfreie Grammatiken - Beispiele

Reelle Numerale

$G = \langle V, T, P, \mathit{Real} \rangle$, wobei

$V = \{\mathit{Real}, \mathit{Scale}, \mathit{Sign}, \mathit{Digits}, \mathit{Digit}\}$
$T = \{0, \ldots, 9, ., \mathtt{E}, +, -\}$

und P folgende Produktionen sind:

$$
\begin{aligned}
\mathit{Real} &\to \mathit{Digit\ Digits}\ .\ \mathit{Digits\ Scale} \\
\mathit{Scale} &\to \varepsilon \mid \mathtt{E}\ \mathit{Sign\ Digit\ Digits} \\
\mathit{Sign} &\to \varepsilon \mid + \mid - \\
\mathit{Digits} &\to \varepsilon \mid \mathit{Digit\ Digits} \\
\mathit{Digit} &\to 0 \mid \cdots \mid 9
\end{aligned}
$$

Optionalität:
$A \to \varepsilon \mid B$ (A steht für optionales B)
Wiederholung:
$A \to \varepsilon \mid B\,A$ (A steht für wiederholtes B, auch 0 Mal)
$A \to B \mid B\,A$ (A steht für wiederholtes B, mind. 1 Mal)

Kontextfreie Grammatiken - Beispiele

Wohlgeformte Klammerausdrücke

WKA ist die kleinste Menge, sodass

- $\varepsilon \in WKA$
- $(w) \in WKA$, wenn $w \in WKA$
- $w_1 w_2 \in WKA$, wenn $w_1, w_2 \in WKA$

$$G = \langle \{W\}, \quad \{(,)\}, \quad \{W \to \varepsilon \mid (\,W\,) \mid W\,W\}, \quad W \rangle$$

Beispiel einer Ableitung:

$$\begin{aligned} W &\Rightarrow_P W\,W \\ &\Rightarrow_P (W)(W) \\ &\Rightarrow_P ()(W\,W) \\ &\Rightarrow_P ()((W)(W)) \\ &\Rightarrow_P ()(()()) \end{aligned}$$

$$\begin{aligned} \mathcal{L}(G) &= \{\varepsilon,\ (),\ (()),\ ()(),\ ((())),\ (())(),\ ()(()),\ ()()(), \ldots\} \\ &= WKA \end{aligned}$$

Induktive Definition vs. kontextfreie Grammatik

Induktive Definition für $\mathcal{M}$	**kontextfreie Grammatik für $\mathcal{M}$**
Mengen $\mathcal{M}$, $\mathcal{M}_0$, $\mathcal{A}_1$, $\mathcal{B}_1$, ...	Nonterminale M, M_0, A_1, B_1, ...
$\mathcal{M}$ ist die kleinste Menge, sodass:	$\mathcal{M} = \mathcal{L}(\langle \ldots, P, M \rangle)$, wobei P:
$\mathcal{M}_0 \subseteq \mathcal{M}$	$M \to M_0$
$f(x_1, \ldots, x_m) \in \mathcal{M}$, falls $x_1 \in \mathcal{A}_1, \ldots, x_m \in \mathcal{A}_m$	$M \to f(A_1, \ldots, A_m)$
$g(x_1, \ldots, x_n) \in \mathcal{M}$, falls $x_1 \in \mathcal{B}_1, \ldots, x_n \in \mathcal{B}_n$	$M \to g(B_1, \ldots, B_n)$
$\cdots \in \mathcal{M}$, falls ...	$M \to \ldots$
$h(x_1, x_2) \in \mathcal{M}$, falls $x_1, x_2 \in \mathcal{M}$	$M \to h(M, M)$
$h(x, x) \in \mathcal{M}$, falls $x \in \mathcal{M}$	keine Entsprechung, nicht kontextfrei

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Backus-Naur-Form (BNF)

Synonym für kontextfreie Produktionen

Historische Notation:

- Nonterminale in spitzen Klammern
- Terminale unter Anführungszeichen
- ::= als Trennsymbol

Durch diese Konvention entfällt die Notwendigkeit, die Mengen der Nonterminale (V) und der Terminale (T) anzugeben.

$$
\begin{aligned}
\langle \mathit{Real} \rangle &::= \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \, \texttt{"."} \, \langle \mathit{Digits} \rangle \, \langle \mathit{Scale} \rangle \\
\langle \mathit{Digit} \rangle &::= \texttt{"0"} \mid \cdots \mid \texttt{"9"} \\
\langle \mathit{Digits} \rangle &::= \varepsilon \mid \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \\
\langle \mathit{Scale} \rangle &::= \varepsilon \mid \texttt{"E"} \, \langle \mathit{Sign} \rangle \, \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \\
\langle \mathit{Sign} \rangle &::= \varepsilon \mid \texttt{"+"} \mid \texttt{"-"}
\end{aligned}
$$

Erweiterte Backus-Naur-Form (EBNF)

„Regular Right Part Grammar“ (RRPG)

- erlaubt reguläre Ausdrücke auf der rechten Seite von Produktionen
- Bessere Lesbarkeit durch Vermeidung von Rekursionen und Leerwörtern
- Kompaktere Spezifikationen
- keine echte Erweiterung: EBNF-Notationen lassen sich eliminieren

Elimination der EBNF-Notation:

$$A \to w_1\,(w_2)\,w_3 \quad \text{entspricht} \quad \begin{aligned} A &\to w_1\,B\,w_3 \\ B &\to w_2 \end{aligned}$$

$$A \to w_1\,\{w_2\}\,w_3 \quad \text{entspricht} \quad \begin{aligned} A &\to w_1\,B\,w_3 \\ B &\to \varepsilon \mid w_2\,B \end{aligned}$$

$$A \to w_1\,[w_2]\,w_3 \quad \text{entspricht} \quad \begin{aligned} A &\to w_1\,B\,w_3 \\ B &\to \varepsilon \mid w_2 \end{aligned}$$

Listen von Bezeichnern

EBNF: $IdList \rightarrow Id\ \{\ \texttt{","}\ Id\ \}$

Ohne EBNF:
$$\begin{aligned} IdList &\rightarrow Id\ B \\ B &\rightarrow \varepsilon \mid \texttt{","}\ Id\ B \end{aligned}$$

oder $IdList \rightarrow Id \mid Id\ \texttt{","}\ IdList$

Skalierungsfaktor der reellen Numerale

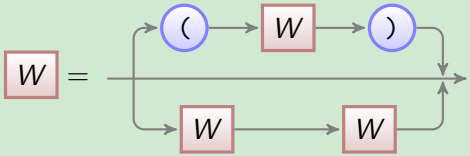
EBNF: $Scale \rightarrow \texttt{"E"}\ [\texttt{"+"} \mid \texttt{"-"}]\ Digit\ \{Digit\}$

Ohne EBNF:
$$\begin{aligned} Scale &\rightarrow \texttt{"E"}\ B_1\ Digit\ B_2 \\ B_1 &\rightarrow \varepsilon \mid \texttt{"+"} \mid \texttt{"-"} \\ B_2 &\rightarrow \varepsilon \mid Digit\ B_2 \end{aligned}$$

Syntaxdiagramme

Mit *rekursiven* Diagrammen können beliebige kontextfreie Grammatiken in EBNF dargestellt werden.

Wohlgeklammerte Ausdrücke



$$W \to (\ W\ )$$
$$\mid \varepsilon$$
$$\mid W\ W$$

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Style Guide für Grammatiken

Wiederholungsoperator statt Rekursion (wenn möglich).

Schlecht: $A \to a \mid A\,A$ oder $A \to a \mid a\,A$

Besser: $a\,\{\,a\,\}$ oder $a+$ oder ... (je nach gewählter Notation)

Optionsoperator statt Leerwort.

Schlecht: $\varepsilon \mid A$

Besser: $[\,A\,]$ oder $A\,?$ oder ... (je nach gewählter Notation)

Modularisierung statt Duplizierung.

Schlecht: $\mathit{Identifier} \rightarrow (\texttt{"a"} \mid \cdots \mid \texttt{"z"})\,\{\,\texttt{"a"} \mid \cdots \mid \texttt{"z"} \mid \texttt{"0"} \mid \cdots \mid \texttt{"9"}\,\}$

Besser:
$$\begin{aligned} \mathit{Identifier} &\rightarrow \mathit{Letter}\,\{\,\mathit{Letter} \mid \mathit{Digit}\,\} \\ \mathit{Letter} &\rightarrow \texttt{"a"} \mid \cdots \mid \texttt{"z"} \\ \mathit{Digit} &\rightarrow \texttt{"0"} \mid \cdots \mid \texttt{"9"} \end{aligned}$$

Verwendung sprechender Namen.

Schlecht:
$$\begin{aligned} X &\rightarrow Y\,\{\,Y \mid Z\,\} \\ Y &\rightarrow \texttt{"a"} \mid \cdots \mid \texttt{"z"} \\ Z &\rightarrow \texttt{"0"} \mid \cdots \mid \texttt{"9"} \end{aligned}$$

Besser: Siehe oben.

Klare Unterscheidung zwischen Terminalen, Nonterminalen und Metanotationen.

Schlecht: `\begin{tabular}[[Position]]{Spalte{Spalte}}`

Besser: `"\begin{tabular}"` [`"["` *Position* `"]"`] `"{"` *Spalte* { *Spalte* } `"}"`

`"\begin{tabular}"` [`"["` ⟨*Position*⟩ `"]"`] `"{"` ⟨*Spalte*⟩ { ⟨*Spalte*⟩ } `"}"`

Inhaltliche Strukturierung statt Minimierung der Nonterminale und Regeln.

Schlecht:
`"\begin{tabular}"[`"["(`"t"` | `"b"`)"]"]"{"(`"l"` | `"c"` | `"r"`){`"l"` | `"c"` | `"r"`}"}"`

Besser:
$$\ldots \rightarrow \texttt{"\textbackslash begin\{tabular\}"}\ [\mathit{Position}]\ \mathit{Spalten}$$
$$\mathit{Position} \rightarrow \texttt{"[b]"} \mid \texttt{"[t]"}$$
$$\mathit{Spalten} \rightarrow \texttt{"\{"}\ \mathit{Spalte}\ \{\ \mathit{Spalte}\ \}\ \texttt{"\}"}$$
$$\mathit{Spalte} \rightarrow \texttt{"l"} \mid \texttt{"c"} \mid \texttt{"r"}$$

Beispiel: Tabellen in LaTeX

TeX ... Textsatzsystem von Donald E. Knuth
LaTeX ... TeX-Makros für „logisches Markup“ von Leslie Lamport

```
\begin{tabular}[t]{lc}
Eintrag 11 & Eintrag 12 \\
Eintrag 21 &
  \begin{tabular}{rr}
  Eintrag 22 & ist selber \\
  eine       & Tabelle.
  \end{tabular}\\
Eintrag 31 &  Eintrag 32
\end{tabular}
```

Eintrag 11	Eintrag 12
Eintrag 21	Eintrag 22 ist selber eine Tabelle.
Eintrag 31	Eintrag 32

Gesucht: Kontextfreie Grammatik G in EBNF für die Sprache der LaTeX-Tabellen

Beispiel: Tabellen in LaTeX

$G = \langle V, T, P, \textit{Tabelle} \rangle$, wobei:

$$
\begin{aligned}
P = \{\ \textit{Tabelle} &\to \texttt{"\textbackslash begin\{tabular\}"}\ [\,\textit{Position}\,]\ \textit{Spalten} \\
&\quad\ \textit{Zeilen} \\
&\quad\ \texttt{"\textbackslash end\{tabular\}"}\ , \\
\textit{Position} &\to \texttt{"[b]"} \mid \texttt{"[t]"}\ , \\
\textit{Spalten} &\to \texttt{"\{"}\ \textit{Spalte}\ \{\,\textit{Spalte}\,\}\ \texttt{"\}"}\ , \\
\textit{Spalte} &\to \texttt{"l"} \mid \texttt{"c"} \mid \texttt{"r"}\ , \\
\textit{Zeilen} &\to \textit{Zeile}\ \{\ \texttt{"\textbackslash\textbackslash"}\ \textit{Zeile}\ \}\ , \\
\textit{Zeile} &\to \textit{Eintrag}\ \{\ \texttt{"\&"}\ \textit{Eintrag}\ \}\ , \\
\textit{Eintrag} &\to \{\ \textit{Tabelle} \mid \textit{Zeichen}\ \}\ , \\
\textit{Zeichen} &\to \texttt{"0"} \mid \cdots \mid \texttt{"9"} \mid \texttt{"a"} \mid \cdots \mid \texttt{"Z"} \mid \texttt{"."} \mid \texttt{"␣"}\ \}
\end{aligned}
$$

$$
\begin{aligned}
V &= \{\textit{Tabelle}, \textit{Position}, \textit{Spalten}, \textit{Spalte}, \textit{Zeilen}, \textit{Zeile}, \textit{Eintrag}, \textit{Zeichen}\} \\
T &= \{\ \texttt{"0"}, \ldots, \texttt{"9"}, \texttt{"a"}, \ldots, \texttt{"z"}, \texttt{"A"}, \ldots, \texttt{"Z"}, \\
&\quad\ \texttt{"."}, \texttt{"␣"}, \texttt{"\{"}, \texttt{"\}"}, \texttt{"["}, \texttt{"]"}, \texttt{"\&"}, \texttt{"\textbackslash"}\ \}
\end{aligned}
$$

Nur eine Rekursion für Schachtelung der Tabellen notwendig!

Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

N. Wirth: Programming in Modula-2

Appendix 1

The Syntax of Modula-2

```
1  ident = letter {letter | digit}.
2  number = integer | real.
3  integer = digit {digit} | octalDigit {octalDigit} ("B"|"C") |
4     digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [ScaleFactor].
6  ScaleFactor = "E" ["+"|"-"] digit {digit}.
7  hexDigit = digit |"A"|"B"|"C"|"D"|"E"|"F".
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
10 string = "'" {character} "'" | '"' {character} '"'.
11 qualident = ident {"." ident}.
12 ConstantDeclaration = ident "=" ConstExpression.
13 ConstExpression = SimpleConstExpr [relation SimpleConstExpr].
14 relation = "=" | "#" | "<>" | "<" | "<=" | ">" | ">=" | IN .
15 SimpleConstExpr = ["+"|"-"] ConstTerm {AddOperator ConstTerm}.
16 AddOperator = "+" | "-" | OR .
17 ConstTerm = ConstFactor {MulOperator ConstFactor}.
18 MulOperator = "*" | "/" | DIV | MOD | AND | "&" .
19 ConstFactor = qualident | number | string | set |
20    "(" ConstExpression ")" | NOT ConstFactor.
21 set = [qualident] "{" [element {"," element}] "}".
22 element = ConstExpression [".." ConstExpression].
23 TypeDeclaration = ident "=" type.
24 type = SimpleType | ArrayType | RecordType | SetType |
25    PointerType | ProcedureType.
26 SimpleType = qualident | enumeration | SubrangeType.
27 enumeration = "(" IdentList ")".
28 IdentList = ident {"," ident}.
29 SubrangeType = "[" ConstExpression ".." ConstExpression "]".
30 ArrayType = ARRAY SimpleType {"," SimpleType} OF type.
31 RecordType = RECORD FieldListSequence END.
32 FieldListSequence = FieldList {";" FieldList}.
33 FieldList = [IdentList ":" type |
34    CASE [ident ":"] qualident OF variant {"|" variant}
35    [ELSE FieldListSequence] END].
36 variant = CaseLabelList ":" FieldListSequence.
37 CaseLabelList = CaseLabels {"," CaseLabels}.
38 CaseLabels = ConstExpression [".." ConstExpression].
39 SetType = SET OF SimpleType.
40 PointerType = POINTER TO type.
41 ProcedureType = PROCEDURE [FormalTypeList].
42 FormalTypeList = "(" [VAR] FormalType
43    {"," [VAR] FormalType}] ")" [":" qualident].
44 VariableDeclaration = IdentList ":" type.
```



```
45 designator = qualident {"." ident | "[" ExpList "]" | "^"}.
46 ExpList = expression {"," expression}.
47 expression = SimpleExpression [relation SimpleExpression].
48 SimpleExpression = ["+"|"-"] term {AddOperator term}.
49 term = factor {MulOperator factor}.
50 factor = number | string | set | designator [ActualParameters] |
51    "(" expression ")" | NOT factor.
52 ActualParameters = "(" [ExpList] ")" .
53 statement = [assignment | ProcedureCall |
54    IfStatement | CaseStatement | WhileStatement |
55    RepeatStatement | LoopStatement | ForStatement |
56    WithStatement | EXIT | RETURN [expression] ].
57 assignment = designator ":=" expression.
58 ProcedureCall = designator [ActualParameters].
59 StatementSequence = statement {";" statement}.
60 IfStatement = IF expression THEN StatementSequence
61    {ELSIF expression THEN StatementSequence}
62    [ELSE StatementSequence] END.
63 CaseStatement = CASE expression OF case {"|" case}
64    [ELSE StatementSequence] END.
65 case = CaseLabelList ":" StatementSequence.
66 WhileStatement = WHILE expression DO StatementSequence END.
67 RepeatStatement = REPEAT StatementSequence UNTIL expression.
68 ForStatement = FOR ident ":=" expression TO expression
69    [BY ConstExpression] DO StatementSequence END.
70 LoopStatement = LOOP StatementSequence END.
71 WithStatement = WITH designator DO StatementSequence END .
72 ProcedureDeclaration = ProcedureHeading ";" block ident.
73 ProcedureHeading = PROCEDURE ident [FormalParameters].
74 block = {declaration} [BEGIN StatementSequence] END.
75 declaration = CONST {ConstantDeclaration ";"} |
76    TYPE {TypeDeclaration ";"} |
77    VAR {VariableDeclaration ";"} |
78    ProcedureDeclaration ";" | ModuleDeclaration ";".
79 FormalParameters =
80    "(" [FPSection {";" FPSection}] ")" [":" qualident].
81 FPSection = [VAR] IdentList ":" FormalType.
82 FormalType = {ARRAY OF} qualident.
83 ModuleDeclaration =
84    MODULE ident [priority] ";" {import} [export] block ident.
85 priority = "[" ConstExpression "]".
86 export = EXPORT [QUALIFIED] IdentList ";".
87 import = [FROM ident] IMPORT IdentList ";".
88 DefinitionModule = DEFINITION MODULE ident ";" {import}
89    [export] {definition} END ident ".".
90 definition = CONST {ConstantDeclaration ";"} |
91    TYPE (ident ["=" type] ";") |
92    VAR {VariableDeclaration ";"} |
93    ProcedureHeading ";" .
94 ProgramModule =
95    MODULE ident [priority] ";" {import} block ident "." .
```

N. Wirth: Programming in Modula-2

Appendix 1

The Syntax of Modula-2

```
1  ident = letter {letter | digit}.
2  number = integer | real.
3  integer = digit {digit} | octalDigit {octalDigit
4     digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [ScaleFactor].
6  ScaleFactor = "E" ["+"|"-"] digit {digit}.
7  hexDigit = digit |"A"|"B"|"C"|"D"|"E"|"F".
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
10 string = "'" {character} "'" | '"' {character} '"'.
11 qualident = ident {"." ident}.
12 ConstantDeclaration = ident "=" ConstExpres
13 ConstExpression = SimpleConstExpr [relatio
14 relation = "=" | "#" | "<>" | "<" | "<=" | ">"
15 SimpleConstExpr = ["+"|"-"] ConstTerm {A
16 AddOperator = "+" | "-" | OR .
17 ConstTerm = ConstFactor {MulOperator Co
18 MulOperator = "*" | "/" | DIV | MOD | AN
19 ConstFactor = qualident | number | string |
20    "(" ConstExpression ")" | NOT ConstFact
21 set = [qualident] "{" [element {"," elemen
22 element = ConstExpression ["," ConstExpre
23 TypeDeclaration = ident "=" type.
24 type = SimpleType | ArrayType | RecordType | SetType |
25    PointerType | ProcedureType.
26 SimpleType = qualident | enumeration | SubrangeType.
27 enumeration = "(" IdentList ")".
28 IdentList = ident {"," ident}.
29 SubrangeType = "[" ConstExpression ".." ConstExpression "]".
30 ArrayType = ARRAY SimpleType {"," SimpleType} OF type.
31 RecordType = RECORD FieldListSequence END.
32 FieldListSequence = FieldList {";" FieldList}.
33 FieldList = [IdentList ":" type |
34    CASE [ident ":"] qualident OF variant {"|" variant}
35    [ELSE FieldListSequence] END].
36 variant = CaseLabelList ":" FieldListSequence.
37 CaseLabelList = CaseLabels {"," CaseLabels}.
38 CaseLabels = ConstExpression [".." ConstExpression].
39 SetType = SET OF SimpleType.
40 PointerType = POINTER TO type.
41 ProcedureType = PROCEDURE [FormalTypeList].
42 FormalTypeList = "(" [VAR] FormalType
43    {"," [VAR] FormalType)} ")" [":" qualident].
44 VariableDeclaration = IdentList ":" type.
```



```
1  ident = letter {letter | digit}.
2  number = integer | real.
3  integer = digit {digit} | octalDigit {octalDigit} ("B"|"C")|
4     digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [ScaleFactor].
6  ScaleFactor = "E" ["+"|"-"] digit {digit}.
7  hexDigit = digit |"A"|"B"|"C"|"D"|"E"|"F".
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
```


```
72 ProcedureDeclaration = ProcedureHeading ";" block ident.
73 ProcedureHeading = PROCEDURE ident [FormalParameters].
74 block = {declaration} [BEGIN StatementSequence] END.
75 declaration = CONST {ConstantDeclaration ";"} |
76    TYPE {TypeDeclaration ";"} |
77    VAR {VariableDeclaration ";"} |
78    ProcedureDeclaration ";" | ModuleDeclaration ";".
79 FormalParameters =
80    "(" [FPSection (";" FPSection)] ")" [":" qualident].
81 FPSection = [VAR] IdentList ":" FormalType.
82 FormalType = [ARRAY OF] qualident.
83 ModuleDeclaration =
84    MODULE ident [priority] ";" {import} [export] block ident.
85 priority = "[" ConstExpression "]".
86 export = EXPORT [QUALIFIED] IdentList ";".
87 import = [FROM ident] IMPORT IdentList ";".
88 DefinitionModule = DEFINITION MODULE ident ";" {import}
89    [export] {definition} END ident ".".
90 definition = CONST {ConstantDeclaration ";"} |
91    TYPE {ident ["=" type] ";"} |
92    VAR {VariableDeclaration ";"} |
93    ProcedureHeading ";" .
94 ProgramModule =
95    MODULE ident [priority] ";" {import} block ident "." .
```

J. Gosling et al.: Java Language Specification

CHAPTER 18

Syntax

THIS chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of x.
- *{x}* denotes zero or more occurrences of x.
- *x | y* means one of either x or y.

Identifier:
IDENTIFIER

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , QualifiedIdentifier }

EnumBody:
{ *[EnumConstants] [,] [EnumBodyDeclarations]* }

EnumConstants:
EnumConstant
EnumConstants , *EnumConstant*

EnumConstant:
[Annotations] Identifier [Arguments] [ClassBody]

EnumBodyDeclarations:
; *{ClassBodyDeclaration}*

AnnotationTypeBody:
{ *[AnnotationTypeElementDeclarations]* }

AnnotationTypeElementDeclarations:
AnnotationTypeElementDeclaration
AnnotationTypeElementDeclarations AnnotationTypeElementDeclaration

AnnotationTypeElementDeclaration:
{Modifier} AnnotationTypeElementRest

AnnotationTypeElementRest:
Type Identifier AnnotationMethodOrConstantRest ;
ClassDeclaration
InterfaceDeclaration
EnumDeclaration
AnnotationTypeDeclaration

AnnotationMethodOrConstantRest:
AnnotationMethodRest
ConstantDeclaratorsRest

AnnotationMethodRest:
( ) *[*[]*]* [default] *ElementValue]*

591 606

J. Gosling et al.: Java Language Specification

CHAPTER 18

THIS chapter presents a grammar for the Java programming l

The grammar presented piecemeal in the preceding chapters (§
for exposition, but it is not well suited as a basis for a parser. The
in this chapter is the basis for the reference implementation. N
LL(1) grammar, though in many cases it minimizes the necessa

The grammar below uses the following BNF-style conventions

- *[x]* denotes zero or one occurrences of x.
- *{x}* denotes zero or more occurrences of x.
- *x | y* means one of either x or y.

Identifier:
IDENTIFIER

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , QualifiedIdentifier }

EnumBody:
{ *[EnumConstants] [,] [EnumBodyDeclarations]* }

EnumConstants:
EnumConstant

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of x.
- *{x}* denotes zero or more occurrences of x.
- *x | y* means one of either x or y.

AnnotationTypeElementDeclaration:
{Modifier} AnnotationTypeElementRest

AnnotationTypeElemRest:
Type Identifier AnnotationMethodOrConstantRest ;
ClassDeclaration
InterfaceDeclaration
EnumDeclaration
AnnotationTypeDeclaration

AnnotationMethodOrConstantRest:
AnnotationMethodRest
ConstantDeclaratorsRest

AnnotationMethodRest:
( ) *[*[]*]* `default` *ElementValue]*

J. Gosling et al.: Java Language Specification

CHAPTER 18

Syntax

THIS chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of x.
- *{x}* denotes zero or more occurrences of x.
- *x | y* means one of either x or y.

Identifier:
IDENTIFIER

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , QualifiedIdentifier }

EnumBody:
{ [EnumConstants] [,] [EnumBodyDeclarations] }

EnumConstants:
EnumConstant
EnumConstants , EnumConstant

EnumConstant:
[Annotations] Identifier [Arguments] [ClassBody]

EnumBodyDeclarations:
; {ClassBodyDeclaration}

AnnotationTypeBody:

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , QualifiedIdentifier }

( ) [[]] [default ElementValue]

J. Gosling et al.: Java Language Specification

CHAPTER 18

Syntax

THIS chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

EnumBody:
{ *[EnumConstants] [,] [EnumBodyDeclarations]* }

EnumConstants:
EnumConstant
EnumConstants , *EnumConstant*

EnumConstant:
[Annotations] Identifier [Arguments] [ClassBody]

EnumBodyDeclarations:
; *{ClassBodyDeclaration}*

AnnotationTypeBody:
{ *[AnnotationTypeElementDeclarations]* }

AnnotationTypeElementDeclarations:
AnnotationTypeElementDeclaration
AnnotationTypeElementDeclarations AnnotationTypeElementDeclaration

AnnotationTypeElementDeclaration:
Modifiers Type AnnotationTypeElementRest

AnnotationTypeElementRest:
Type Identifier AnnotationMethodOrConstantRest ;
ClassDeclaration
InterfaceDeclaration
EnumDeclaration
AnnotationTypeDeclaration

AnnotationMethodOrConstantRest:
AnnotationMethodRest
ConstantDeclaratorsRest

AnnotationMethodRest:
Identifier ( ) *[DefaultValue]*

DefaultValue:
default *ElementValue*

EnumConstants:
EnumConstant
EnumConstants , *EnumConstant*

EnumConstant:
[Annotations] Identifier [Arguments] [ClassBody]

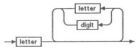
EnumBodyDeclarations:
; *{ClassBodyDeclaration}*

N. Wirth: Programming in Modula-2

Appendix 4

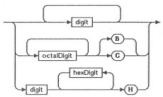
Syntax Diagrams

ident



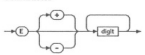
number

integer

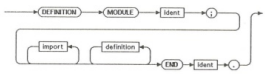


real

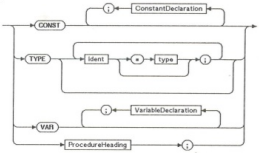
ScaleFactor



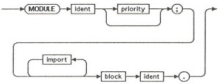
DefinitionModule



definition



ProgramModule



CompilationUnit

N. Wirth: Programming in Modula-2

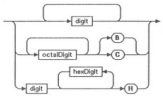
Appendix 4

Syntax Diagrams

ident

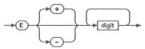
number

integer

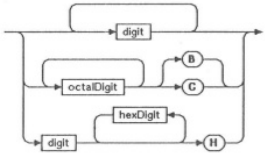


real

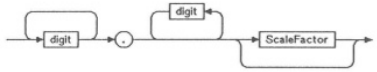
ScaleFactor



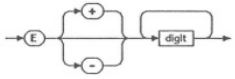
integer



real



ScaleFactor

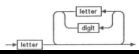


N. Wirth: Programming in Modula-2

Appendix 4

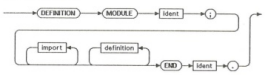
Syntax Diagrams

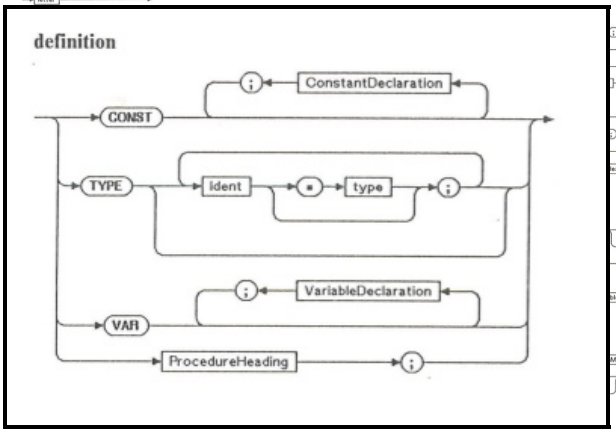
ident

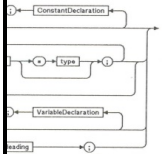


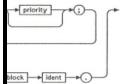
180 A4. Syntax Diagrams

DefinitionModule









Formale Sprachen – Übersicht

1. Operationen auf formalen Sprachen
2. Reguläre Sprachen
3. Reguläre Ausdrücke
4. Eigenschaften regulärer Sprachen
5. Vom regulären Ausdruck zum Automaten
6. Vom Automaten zum regulären Ausdruck
7. Kontextfreie Sprachen
8. Beispiele kontextfreier Grammatiken
9. Andere Notationen
10. Style Guide für Grammatiken
11. Grammatiken in freier Wildbahn
12. Jenseits der Kontextfreiheit

Jenseits der Kontextfreiheit

Formale Sprachen

$\{ a^n b^n c^n \mid n \geq 0 \}$ ist nicht kontextfrei.

Formale Sprachen

$\{ ww \mid w \in \{a, b, c\}^* \}$ ist nicht kontextfrei.

Programmiersprachen

Typen- und Deklarationsbedingungen nicht oder nur schwer kontextfrei darstellbar.

Natürliche Sprachen (Schweizer Dialekt)

Mer d'chind em Hans es huus
haend wele laa hälfe aastriiche.

Wir wollten die Kinder dem Hans
helfen lassen das Haus anzustreichen.

Kontextfreie Sprachen & Kellerautomaten

- Reguläre Sprache besitzen die gleich Ausdrucksstärke wie DEAs und NEAs.
- Gibt es auch einen passenden Automaten für Kontextfreie Sprachen?

  $\hookrightarrow$ Ja, nichtdeterministische Kellerautomaten (pushdown automata)
- Kellerautomat = endlicher Automat + Kellerspeicher (Stack)
  - Kellerspeicher ermöglicht unendlich viele „Zustände“
  - Automat sieht nur das oberste Symbol im Keller (und Zustandsübergänge hängen davon ab)
  - Bei einem Übergang können neue Symbole oben auf den Keller geschrieben werden oder das oberste Symbol kann gelöscht werden.

Andere Arten von Grammatiken

- Wir haben nur kontextfreie Grammatiken (Typ-2 Grammatik) betrachtet
- Es gibt ausdrucksstärkere Grammatiken:
  - uneingeschränkte formale Grammatik (Typ-0)
  - kontextsensitive Grammatik (Typ-1)
- ... und einfachere Arten von Grammatiken:
  - reguläre Grammatiken (Typ-3) definieren die regulären Sprachen und besitzen damit dieselbe Ausdrucksstärke wie endliche Automaten und reguläre Ausdrücke.

Zusammenfassung

- Grammatik definieren formale Sprachen
- Kontextfreie Grammatiken
  - ausdrucksstärker als reguläre Ausdrücke und endliche Automaten
  - darstellbar als rekursive Syntaxdiagramme
- Style Guide für Grammatiken
  - EBNF kombiniert reguläre Ausdrücke und kontextfreie Produktionen