Automaten und Formale Sprachen

SoSe 2013 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

7. Juni 2013

Automaten und Formale Sprachen

Gesamtübersicht

- Organisatorisches
- Einführung
- Endliche Automaten und reguläre Sprachen
- **Kontextfreie Grammatiken und kontextfreie Sprachen**
- Chomsky-Hierarchie

Kontextfreie Grammatiken und kontrextfreie Sprachen

1. **Kontextfreie Grammatiken und Baumautomaten**
2. Automaten mit unendlichem Speicher
3. Normalformen
4. Nichtkontextfreie Sprachen
5. Algorithmen für kontextfreie Grammatiken

Eine **kontextfreie Grammatik** ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit:

- Σ ist das *Terminalalphabet*,
- N ist das *Nonterminalalphabet* (die *Variablenmenge*),
- $R \subset N \times (\Sigma \cup N)^*$ ist das Alphabet der *Regeln* oder *Produktionen*; übliche Schreibweise: $A \to \nu$ anstelle von $(A, \nu) \in R$, wobei $A \in N$ und $\nu \in (\Sigma \cup N)^*$ auch *linke Seite* bzw. *rechte Seite* der Regel heißen.
- $S \in N$ ist das *Startsymbol* oder *Anfangszeichen*.

Ein Wort über dem *Gesamtalphabet* $(\Sigma \cup N)$ heißt auch *Satzform*.

Ein Beispiel

$G = (\{a, b\}, \{S\}, R, S)$ mit den Regeln $r_1 = S \to aSb$ und $r_2 = S \to \lambda$.

a, b: die Terminalzeichen
S: die Nonterminalzeichen; hier gleichzeitig das Startzeichen
$\{S, a, b\}$: das Gesamtalphabet
$R = \{r_1, r_2\}$: die Regelmenge
$abS, aaSbb, aSaSa$: mögliche Satzformen, nicht alle ableitbar in G

Der **Ableitungsmechanismus** einer kontextfreien Grammatik:

1-Schritt-Ableitungsrelation $\Rightarrow_G$ zwischen zwei Satzformen u, v einer kfG G:
$u \Rightarrow_G v$ (manchmal kurz $u \Rightarrow v$) gdw.
es gibt Regel $A \to y$, sodass u und v wie folgt zerlegt werden können: $u = xAz$ und $v = xyz$; hierbei sind x und z wiederum Satzformen.

Etwas formaler, ggb. $G = (\Sigma, N, R, S)$:

$$\forall u, v \in (\Sigma \cup N)^* : u \Rightarrow_G v \iff$$

$$(\exists x, z \in (\Sigma \cup N)^* \exists (A \to y) \in R : u = xAz \wedge v = xyz)$$

$\Rightarrow^n$: n-Schritt-Ableitungsrelation.
$\overset{*}{\Rightarrow}$: Ableitung mit beliebig vielen Schritten.

Die von einer kfG G *erzeugte* oder *abgeleitete* Sprache ist gegeben durch:

$$L(G) := \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}.$$

Bequeme Schreibweise einer *Ableitung(sfolge)*:

$$u_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow u_3$$

Gilt $u \overset{*}{\Rightarrow} v$, so gilt für ein k: $u \Rightarrow^k v$, bezeugt durch die Ableitungsfolge $u = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_{k-1} \Rightarrow u_k = v$. Dieses k heißt auch *Länge der Ableitung*.

KF: Familie der *kontextfreien Sprachen* (der durch kontextfreie Grammatiken erzeugbaren Sprachen).

Ein Beispiel

$G = (\{a, b\}, \{S\}, R, S)$ mit den Regeln $r_1 = S \to aSb$ und $r_2 = S \to \lambda$.

Lemma: $L(G) = \{a^n b^n \mid n \geq 0\}$.

Beweis: Offenbar gilt $\lambda \in L(G)$ und $\lambda = a^0 b^0$, sodass wir diesen Fall hinfort außer Acht lassen können.
Per Induktion beweisen wir die folgende Behauptung:
Die einzigen in $k > 0$ Ableitungsschritten ableitbaren Wörter sind $a^k S b^k$ und $a^{k-1} b^{k-1}$.
✓ für $k = 1$.
IH: Die Behauptung gilt für $k > 0$. Betrachte eine Ableitung der Länge $k + 1$: $S \Rightarrow^k v \Rightarrow w$.
IH $\rightsquigarrow v = a^k S b^k$ (Die Möglichkeit $v = a^{k-1} b^{k-1}$ gestattet keine Fortführung.).
Anwendung von r_1 liefert $v \Rightarrow a^{k+1} S b^{k+1}$.
Anwendung von r_2 liefert $v \Rightarrow a^k b^k$.
Dies war zu zeigen. □

Rechtslinear und regulär

Eine kfG heißt *rechtslinear* gdw. alle rechten Regelseiten haben die Form zA oder z, wobei z ein Terminalwort ist und A ein Nichtterminalzeichen.

Satz: L ist regulär gdw. L ist durch rechtslineare kfG erzeugbar.

Beweis: Es sei $A = (Q, \Sigma, \delta, s, F)$ ein DEA (mit $\Sigma \cap Q = \emptyset$).
Für $\delta(q, a) = q'$ wird die Regel $q \to aq'$ in die simulierende Grammatik $G = (\Sigma, Q, R, s)$ aufgenommen; die einzigen anderen Regeln sind $q \to \lambda$ für alle $q \in F$.

Umgekehrt können die Nichtterminale einer rechtslinearen Grammatik als Zustände eines NEAs mit "Zustandsübergängen mit beliebigen Wörtern" gedeutet werden. Details als Übung. □

Folgerung: **REG** $\subset$ **KF**.

Eine Anwendung von KfG:
Beschreibung der Syntax (grammatisches Gerüst) natürlicher Sprachen

Satz $\rightarrow$ NP VP
NP $\rightarrow$ Artikel Nomen
VP $\rightarrow$ Verb
Artikel $\rightarrow$ “der”
Artikel $\rightarrow$ “die”
Nomen $\rightarrow$ “Hund”
Nomen $\rightarrow$ “Hunde”
Nomen $\rightarrow$ “Katze”
Verb $\rightarrow$ “beißen”

Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “die” “Hunde” “beißen”
Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “die” “Katze” “beißen”
Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “der” “Katze” “beißen”

Eine Anwendung von KfG: **Beschreibung arithmetischer Ausdrücke**

$\Sigma = \{v, -, +, *, /, (,)\}$, $N = \{E\}$.
Die Regeln seien die folgenden:

$E \to (E)$
$E \to -(E)$
$E \to (E + E)$
$E \to (E - E)$
$E \to (E * E)$
$E \to (E/E)$
$E \to v$

Beispielableitung:

$E \Rightarrow -(E)$
$\Rightarrow -((E + E))$
$\Rightarrow -(((E * E) + E))$
$\Rightarrow -(((-(E) * E) + E))$
$\Rightarrow -(((-(E) * E) + (E - E)))$
$\overset{*}{\Rightarrow} -(((-(v) * v) + (v - v)))$

Hinweis: Der *Scanner*, ein endlicher Automat, produziert idealerweise als Ausgabe die Eingabe für den *Parser* zwecks *syntaktischer Analyse* eines Programmtextes.

“Nebensächlichkeiten” wie Zahlen oder Zahlenvariablen werden in einen Statthalter v übersetzt.

Ein Wort hat viele Ableitungen

Linksableitung:

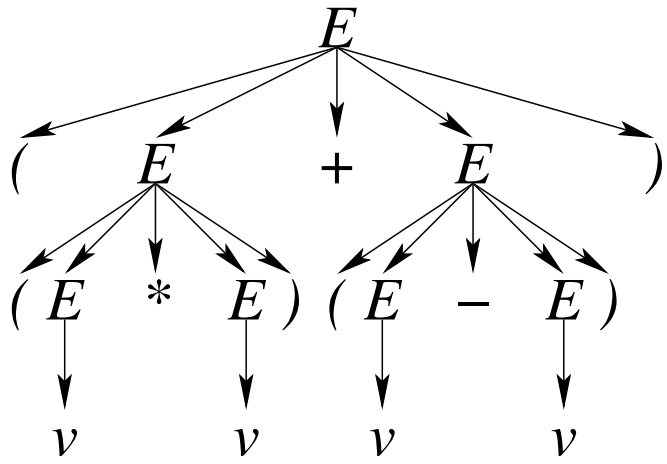
$$
\begin{aligned}
& E \Rightarrow -(E) \\
& \Rightarrow -((E + E)) \\
& \Rightarrow -(((E * E) + E)) \\
& \Rightarrow -(((-(E) * E) + E)) \\
& \Rightarrow -(((-(v) * E) + E)) \\
& \Rightarrow -(((-(v) * v) + E)) \\
& \Rightarrow -(((-(v) * v) + (E - E))) \\
& \Rightarrow -(((-(v) * v) + (v - E))) \\
& \Rightarrow -(((-(v) * v) + (v - v)))
\end{aligned}
$$

Rechtsableitung:

$$
\begin{aligned}
& E \Rightarrow -(E) \\
& \Rightarrow -((E + E)) \\
& \Rightarrow -((E + (E - E))) \\
& \Rightarrow -((E + (E - v))) \\
& \Rightarrow -((E + (v - v))) \\
& \Rightarrow -(((E * E) + (v - v))) \\
& \Rightarrow -(((E * v) + (v - v))) \\
& \Rightarrow -(((-(E) * v) + (v - v))) \\
& \Rightarrow -(((-(v) * v) + (v - v)))
\end{aligned}
$$

Ein Syntaxbaum (oder *Ableitungsbaum*) für

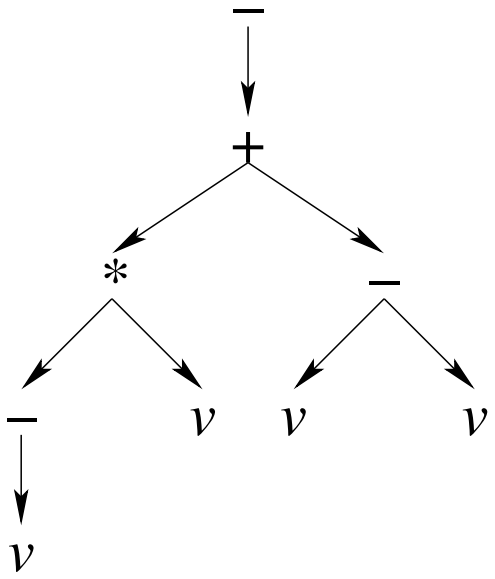
$$\mathrm{E} \Rightarrow (\mathrm{E}+\mathrm{E}) \Rightarrow ((\mathrm{E}*\mathrm{E})+\mathrm{E}) \Rightarrow ((\mathrm{E}*\mathrm{E})+(\mathrm{E}-\mathrm{E})) \overset{*}{\Rightarrow} ((v*v)+(v-v))$$



Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum

Rechtsableitung: Tiefensuche mit Rechtsabstieg durch Syntaxbaum

Operatorbaum: ein kompakter Syntaxbaum für das ursprüngliche Beispiel



Bäume formaler... Eine Erinnerung / Spezialisierung aus DS:

Unter dem *Typ einer algebraischen Struktur* versteht man ein Paar $(\mathcal{F}, \sigma)$.
Hierbei ist
$\mathcal{F}$ die Menge der *Funktionensymbole*,
$\sigma : \mathcal{F} \to \mathbb{N}$ liefert die *Stelligkeit* zu dem betreffenden Symbol.

Der Typ einer Struktur ist ein rein syntaktisches Objekt. (**syntaktische Ebene**)
Wenn es auf die Namen der Symbole nicht ankommt, erwähnt man oft auch nur den Stelligkeitstyp.

Informatisch gesprochen beschreibt ein Typ das Interface zwischen Strukturen.

Strukturen und Algebren

Es sei $A \neq \emptyset$ eine Menge und $n \in \mathbb{N}$.
Eine Abbildung $A^n \to A$ heißt *n-stellige Operation auf A*.
(Hier ist $A^n = A \times \cdots \times A$ das $(n-1)$-fache kartesische Produkt.)
Nullstellige Operationen heißen auch *Konstanten*.
$\mathrm{Op}_n(A)$ sei die Menge der n-stelligen Operationen auf A, d.h., $\mathrm{Op}_n(A) = A^{A^n}$.
$\mathrm{Op}(A) = \bigcup_{n=0}^{\infty} \mathrm{Op}_n(A)$.

Eine *algebraische Struktur* vom Typ $(\mathcal{F}, \sigma)$ ist ein Paar $\mathbb{A} = (A, F)$, $A \neq \emptyset$,
mit $F = \{f_{\mathbb{A}} \mid f \in \mathcal{F}\}$,
wobei jedem $f \in \mathcal{F}$ genau eine Operation $f_{\mathbb{A}} \in \mathrm{Op}_{\sigma(f)}(A)$ zugeordnet ist.

Strukturen bzw. Algebren liefern die **semantische Ebene**.

Terme über Algebren

Es sei $\mathbb{A} = (A, F)$ eine Algebra vom Typ $(\mathcal{F}, \sigma)$.
Dann sind *Terme über* $\mathbb{A}$ induktiv wie folgt definiert:

1. Jedes $a \in A$ ist ein Term.
2. Sind $t_1, \ldots, t_n$ Terme über $\mathbb{A}$ und ist $f \in \mathcal{F}$ mit $\sigma(f) = n$, so ist $f(t_1, \ldots, t_n)$ ein Term über $\mathbb{A}$.
3. Nichts anderes sind Terme über $\mathbb{A}$.

Wir sammeln alle Terme über $\mathbb{A}$ in der Menge $\mathrm{Term}(\mathbb{A})$.

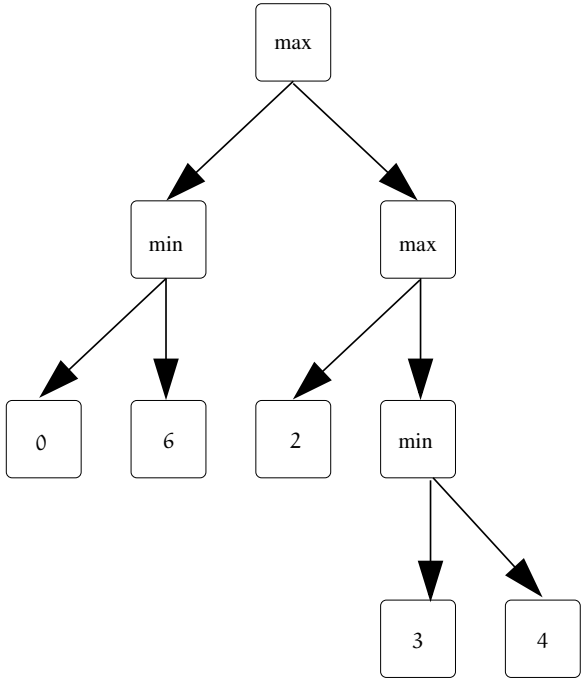
Beachte: Terme sind rein syntaktische Objekte.
Betrachte z.B. die Algebra $\mathbb{A} = (\mathbb{N}, \{\max_{\mathbb{N}}, \min_{\mathbb{N}}\})$ mit zwei zweistelligen Operationen.
Dann ist $\max(\min(0, 6), \max(2, \min(3, 4)))$ ein Term über $\mathbb{A}$.

Terme und Bäume

Einem Term über einer Algebra kann man durch einen knotenbeschrifteten gerichteten geordneten Baum darstellen.
Umgekehrt entsprechen einem knotenbeschrifteten gerichteten geordneten Baum Terme über einer geeignet definierten Algebra.
Wie wir sehen, kann man Berechnungen in Algebren über Terme definieren.
Dies erklärt die Bedeutung dieser Art von Bäumen z.B. im Compilerbau.

Im Beispiel:

$\max(\min(0,6),\max(2,\min(3,4)))$:



Zur Auswertung von Termen

Def.: Es sei $\mathbb{A} = (A, F)$ eine Algebra vom Typ $(\mathcal{F}, \sigma)$.
Wir definieren die *Auswertefunktion* $eval : \mathrm{Term}(\mathbb{A}) \to A$ induktiv wie folgt:

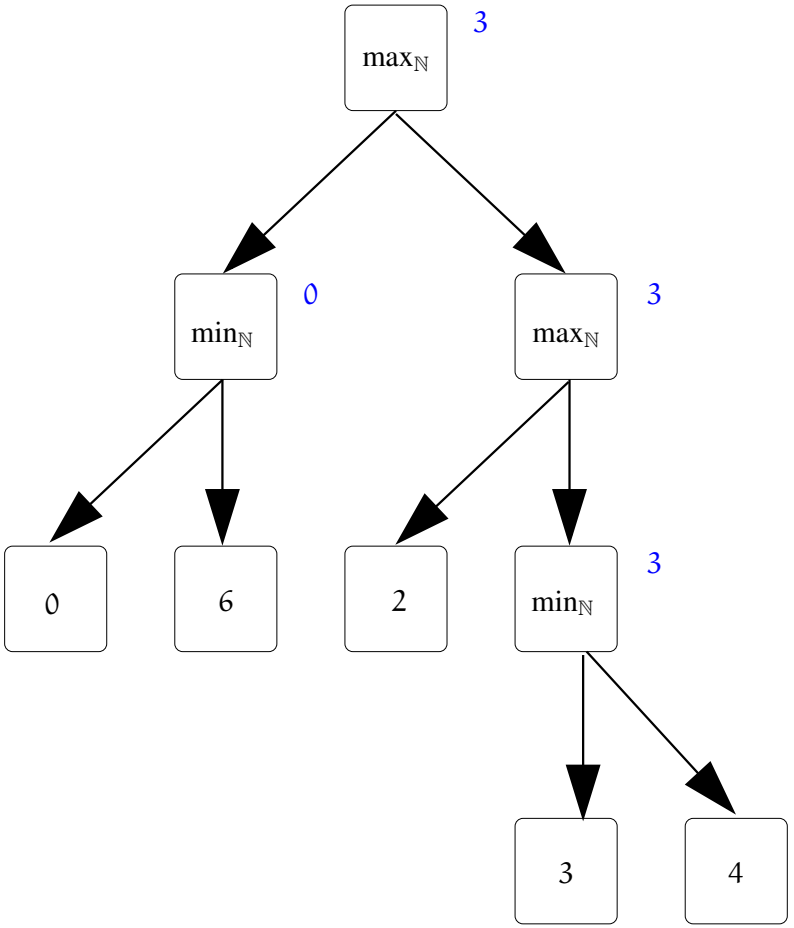
1. Für $a \in A$ sei $eval(a) := a$.
2. Sind $t_1, \ldots, t_n$ Terme über $\mathbb{A}$ und ist $f \in \mathcal{F}$ mit $\sigma(f) = n$, so ist

$$eval(f(t_1, \ldots, t_n)) := f_{\mathbb{A}}(eval(t_1), \ldots, eval(t_n)).$$

Zur Auswertung von Termen: Unser Beispiel

$$
\begin{aligned}
eval(\max(\min(0,6),\max(2,\min(3,4)) &= \max_{\mathbb{N}}(eval(\min(0,6)), eval(\max(2,\min(3,4)))) \\
&= \max_{\mathbb{N}}(\min_{\mathbb{N}}(eval(0), eval(6)), \max_{\mathbb{N}}(eval(2), eval(\min(3,4)))) \\
&= \max_{\mathbb{N}}(\min_{\mathbb{N}}(0,6), \max_{\mathbb{N}}(2, \min_{\mathbb{N}}(eval(3), eval(4)))) \\
&= \max_{\mathbb{N}}(0, \max_{\mathbb{N}}(2, \min_{\mathbb{N}}(3,4))) \\
&= \max_{\mathbb{N}}(0, \max_{\mathbb{N}}(2,3)) \\
&= \max_{\mathbb{N}}(0,3) \\
&= 3
\end{aligned}
$$

Zur Auswertung von Bäumen: Auswertung von unten nach oben



Querbezüge

Knotenbeschriftete geordnete gerichtete Bäume werden Ihnen im Laufe Ihres Studiums verschiedentlich begegnen.

- Ableitungsbäume sind ein bekanntes Konzept aus den Formalen Sprachen.
- Diese werden auch im Compilerbau benutzt; dort sind auch Auswertefunktionen nützlich.
- Semistrukturierte Daten (z.B. XML) lassen sich (abstrakt) so begreifen.

Bäume und endliche Automaten

Es sei $(\mathcal{F}, \sigma)$ der Typ einer algebraischen Struktur.
Einnerung: nullstellige Operatoren sind Konstanten und damit die Beschriftung von Blättern der Termbäume.
Wir interpretieren diese im Folgenden als Teil des Zustandsalphabets.
Es sei Q ein endliches Zustandsalphabet mit Endzustandsmenge $Q_f \subseteq Q$.
Zustandsübergangsmenge Δ mit Elementen der Bauart:
$(f, q_1, \ldots, q_n, q)$ mit $f \in \mathcal{F}$, $\sigma(f) = n$, $q_1, \ldots, q_n, q \in Q$.
Ein *endlicher Baumautomat* kann spezifiziert werden durch das Quadrupel

$$A = (Q, (\mathcal{F}, \sigma), Q_f, \Delta)$$

.

Bäume bzw. Terme

Ein *Baum* über $(\mathcal{F}, \sigma)$ (als rein syntaktisches Objekt) ist gegeben durch:

1. Jedes $a \in \{f \in \mathcal{F} \mid \sigma(f) = 0\}$ ist ein Baum.
2. Sind $t_1, \ldots, t_n$ Bäume über $(\mathcal{F}, \sigma)$ und ist $f \in \mathcal{F}$ mit $\sigma(f) = n$, so ist $f(t_1, \ldots, t_n)$ ein Baum über $(\mathcal{F}, \sigma)$.
3. Nichts anderes sind Bäume über $(\mathcal{F}, \sigma)$.

Bsp. an der Tafel mit $\mathcal{F} = \{\min, \max, v\}$ und $\sigma(\min) = \sigma(\max) = 2$, $\sigma(v) = 0$.
$\mathbb{B}(\mathcal{F}, \sigma)$ sei die Menge der Bäume über $(\mathcal{F}, \sigma)$.

Arbeitsweise endlicher Baumautomaten

Interpretiere $A = (Q, (\mathcal{F}, \sigma), Q_f, \Delta)$ als Algebra $\mathbb{A}_A = (2^Q, F)$ durch:

$$f(Q_1, \ldots, Q_n) := \{q \in Q \mid \exists q_1 \in Q_1, \ldots, \exists q_n \in Q_n : (f, q_1, \ldots, q_n, q) \in \Delta\}$$

für $f \in \mathcal{F}$ mit $\sigma(f) = n$.
Beachte, dass wir nullstellige Operatoren (Blattbeschriftungen) als Zustände und weiter als einelementige Zustandsmengen interpretieren.
Daher sind Bäume über $(\mathcal{F}, \sigma)$ auch Terme über $\mathbb{A}$.
A akzeptiert die Baumsprache $B(A) := \{b \in \mathbb{B}(\mathcal{F}, \sigma) \mid eval_{\mathbb{A}_A}(b) \cap Q_f \neq \emptyset\}$.
Eine Baumsprache B (Menge von Bäumen) ist *regulär* gdw. es gibt einen endlichen Baumautomaten, der B akzeptiert.

Bsp. (Forts.): $\mathcal{F} = \{\min, \max, v\}$, $Q = \{v, q\}$, Δ enthält (f, r, s, v) für $\sigma(f) = 2$ und $r, s \in Q$ sowie $(\min, v, v, q)$ und $(\max, q, v, q)$.
Welche Baumsprache akzeptiert der Automat $A = (Q, (\mathcal{F}, \sigma), \{q\}, \Delta)$?

Wort-Algebra: Eine weitere Algebra für $(\mathcal{F}, \sigma)$

Nullstellige Operationssymbole werden als Buchstaben interpretiert
$\rightsquigarrow$ Alphabet Σ $\rightsquigarrow$ Grundmenge Σ^+ für die Algebra $\mathbb{A}_\Sigma$
Jedes einstellige Operationssymbol wird als Identität gedeutet.
Jedes mehrstellige Operationssymbol wird als Konkatenation der Argumentwörter gelesen.
Die Terme dieser Algebra sind gerade die Bäume von $(\mathcal{F}, \sigma)$.
Für das Ergebnis der Auswertung eines Baumes $\mathfrak{b}$ in dieser Algebra schreiben wir auch:

$$\mathtt{yield} : \mathbb{B}(\mathcal{F}, \sigma) \to \Sigma^+, \quad \mathtt{yield}(\mathfrak{b}) := \mathit{eval}_{\mathbb{A}_\Sigma}(\mathfrak{b})$$

Wie sieht im Beispiel $\mathtt{yield}(\mathrm{B}(\mathrm{A}))$ aus?

Der Satz von Doner, Thatcher und Wright

Satz: Es sei $L \subseteq \Sigma^*$ eine Sprache.
L ist kontextfrei gdw. $L = yield(B(A))$ für einen regulären Baumautomaten A.

Idee 1: Eine Regel $(f, q_1, \ldots, q_n, q)$ des Baumautomaten wird zur Grammatikregel $(f, q) \to (f_1, q_1) \cdots (f_n, q_n)$ für geeignet geratene Symbole $f_1, \ldots, f_n$.

Idee 2: Eine kontextfreie Regel $A \to B_1 \cdots B_n$ wird zur Regel $(X_n, B_1, \ldots, B_n, A)$ des Baumautomaten; hierbei gibt es genau ein Symbol X_n der Stelligkeit n.

Weitere Kommentare

- Die Idee 2 führt unmittelbar zu einer Formalisierung des Ableitungsbaumbegriffs; hierbei werden lediglich die “Dummy-Symbole” X_n nicht hingeschrieben, dafür allerdings die Nichtterminale (also die Zustände des Baumautomaten).
- Die Hintereinanderschaltung beider Ideen führt dazu, dass es zu jeder kontextfreien Grammatik eine äquivalente G gibt, bei der wir jedem Nichtterminalzeichen A eine Stelligkeit $\sigma(A)$ zuordnen können, sodass falls $A \to w$ eine Regel von G ist, so gilt $\sigma(A) = \ell(w)$.
- In der Literatur wird zumeist zwischen “Blattbeschriftungen” und Zuständen getrennt. Dann braucht man jedoch weitere Regeln der Form $(a, q) \in \Delta$ für $a \in \{f \in \mathcal{F} \mid \sigma(f) = 0\}$ und $q \in Q$.

Hintegründe—nochmals

- Endliche (Bottom-Up) Baumautomaten sind ein zentrales Konzept für die Bearbeitung semistrukturierter Daten.
- Es gelten für sie ähnliche Gesetze wie für die Wortautomaten; z.B. gibt es eine Potenzmengenkonstruktion, die die Gleichwertigkeit von nichtdeterministischen und deterministischen Automaten erklärt, und ein Pumping Lemma (Pumpen entlang von Pfaden des Baumes).
- “Bottom-Up-Auswertung” von Bäumen / Termen ist für viele Auswertungen bei Compilern wichtig.
- Die “Ableitungsbäume” sind daher für Compiler essentiell, nicht nur die “Wörter” (also die Programmtexte); das gilt vermehrt für Anwendungen im Bereich Computerlinguistik.