# Formelsammlung
# theoretische Informatik I

<Marco.Moeller@macrolab.de>

Stand: 27.05.2005 - Version: 1.0.3
Erhältlich unter http://privat.macrolab.de

Diese Formelsammlung basiert auf der Vorlesung "Theoretische Informatik 1" von Prof. Dr. Heinrich Werner an der Universität Kassel im Sommersemester 2004.



## Inhaltsverzeichnis

Die hinter den Überschriften angegebenen Nummern beziehen sich auf die Seitenzahlen des Buches “Theoretische Informatik - kurzgefasst” von Uwe Schöning (4. Auflage).

Grundlagen der Logik und der Mengenlehre sind in meiner “Formelsammlung Mathe I/II für Informatiker” nachzuschlagen.

# 1 Mathematische Grundlagen 185

## 1.1 Alphabet und Symbole 185

### 1.1.1 Definition

Als *Alphabet* bezeichnen wir eine endliche, nicht-leere Menge $\Sigma$. Elemente aus $w \in \Sigma$ heißen *Zeichen* oder *Symbole*.

- $\Sigma^*$ = Menge aller Wörter, die sich aus $\Sigma$ bilden lassen incl. leerem Wort $\varepsilon$
- $\Sigma^+$ = Menge aller Wörter, die sich aus $\Sigma$ bilden lassen ohne $\varepsilon$
- $|\Sigma|$ = Anzahl der verschiedenen Symbole in $\Sigma$ (Mächtigkeit der Menge $\Sigma$)

### 1.1.2 Konkatenation

Sei $\circ$ ein Symbol für die Konkatenation, dann ist die algebraische Struktur $(\Sigma^*, \circ)$ eine *Halbgruppe mit neutralem Element*, ein so genanntes *Monoid.*

- Abgeschlossenheit
  $x \in \Sigma^* \wedge y \in \Sigma^* \Rightarrow x \circ y = xy \in \Sigma^*$
- Assoziativität
  $(x \circ y) \circ z = x \circ (y \circ z) = xyz$
- neutrales Element
  $\varepsilon \circ x = x \circ \varepsilon = x$

### 1.1.3 Wort

Ein Wort (mathm. ein Tupel) entsteht durch hintereinanderschreiben (Konkatenation) von Symbolen aus $\Sigma$. $\varepsilon$ steht für das leere Wort (eine art neutrales Element).

- $w^n = \underbrace{ww\ldots w}_{n-mal} \quad (n \in \mathbb{N}_0)$
- $w^0 = \varepsilon$

### 1.1.4 Länge eines Wortes

Für ein Wort $w$ bezeichnet $|w|$ seine Länge, d.h. die Anzahl der in ihm enthaltenen Zeichen.

- $|\varepsilon| = 0$
- $|uv| = |u| + |v|$
- $|w^n| = n\,|w|$

Diese Regeln haben eine gewisse Ähnlichkeit mit derer der Logarithmus Funktion.

## 1.2 Sprachen 186

### 1.2.1 Definition

Eine Teilmenge von $\Sigma^*$ wird als *Sprache* Bezeichnet. ($A \in \Sigma^*$)

### 1.2.2 Rechenregeln

Da es sich bei Sprachen um Mengen handelt, gelten für sie (fast) alle Operatoren und Regeln der Mengen.

Eine Sprache lässt sich aus anderen Sprachen zusammensetzen (ähnlich Konkatenation). Dies bedeutet, das der ein Wort aus der neuen Sprache, aus den Teilen der anderen Sprachen zusammengesetzt wird.

- $A^0 = \{\varepsilon\}$
- $A^1 = A$
- $A^{n+1} = AA^n$
- $A^i A^j = A^{i+j}$
- $\left(A^i\right)^j = A^{ij}$
- $A^* = \cup_{n\geq 0} A^n$
- $A^+ = \cup_{n\geq 1} A^n$
- Menge aller Sprachen (Potenzmenge über $\Sigma^*$)
  $\mathcal{P}(\Sigma^*) = 2^{\Sigma^*} = \{A | A \subseteq \Sigma^*\}$

### 1.2.3 Relationen auf Sprachen 187

Eine zweistellige Relation zwischen $a$ und $b$ ($aRb$) mit $R \subset \Sigma^* \times \Sigma^*$ ist wahr, wenn $(a, b) \in R$ und falsch, wenn $(a, b) \notin R$.

- Hintereinandschreiben von Relationen:
  $RS = \{(x, y) \,|\exists_\Sigma^z : xRz \wedge zSz\}$
- $R^0 = \{x, x|x \in \Sigma^*\}$ = Identität
- $R^{n+1} = RR^n$
- $R = \cup_{n\geq 0} R^n$
- $R^+ = \cup_{n\geq 1} R^n$
- $R$ heißt *reflexiv* wenn:
  $\forall_\Sigma^x : xRx$
- $R$ heißt *transitiv* wenn:
  $xRy \wedge yRz \Rightarrow xRz$
- $R^*$ ist die kleinste reflexive und transitive Relation, die $R$ umfasst (die reflexive und transitive *Hülle* von $R$)
  - kleiner als hoch $*$ geht es nicht, da $R$ sonst nicht transitiv wäre (es muss jede Hintereinanderausführung von zwei Realtionen enthalten)
  - $R^0$ muss enthalten sein, damit auch die Reflexivität gilt.
- $R$ heißt *symmetrisch* wenn:
  $xRy \Leftrightarrow yRx$
- $R$ heißt *Äquivalenzrelation* wenn:
  $R$ ist reflexiv und transitiv und symmetrisch

### 1.2.4 Äquivalenzklassen & Index

Jedem Element aus dem Grundbereich (hier: $\Sigma^*$) kann die Menge der Elemente zugeordnet werden, die zu $x$ äquivalent sind. Diese werden mit $[x]_R$ bezeichnet.

$$[x]_R = \{y|yRx\} = \{y|xRy\}$$

Wenn $R$ aus dem Kontext hervorgeht, schreiben wir auch einfach $[x]$. Diese Mengen heißen *Äquivalenzklassen.*

Die Grundmenge $\Sigma^*$ lässt sich in Äquivalenzklassen zerlegen:

$$\Sigma^* = \dot{\cup}_{k=0}^{n} [x_k] \quad \text{Index}(R) = n$$

Mit $\text{Index}(R)$ bezeichnen wir die Anzahl der verschiedenen Äquivalenzklassen, die $R$ hat.

- $\text{Index}(R)$ ist die maximale Anzahl von Elementen $x_1, x_2, \ldots, x_n, \ldots$ mit $\neg(x_i R x_j)$ $(i \neq j)$
- Falls $\text{Index}(R) < \infty$ gilt, so sagen wir: $R$ hat einen endlichen Index

## 1.3 Komplexitätsfunktionen (O-Notation) 189 / AlgoDS Skript Seite 13

Diese dienen zum abschätzen des (asymptotischen) Verhaltens einer Funktion. Eigentlich werden hiermit Mengen von Funktionen angegeben, aber es hat sich eingebürgert trotzdem = zu schreiben (wenn das $O$ rechts steht). Z.B:

$$5n^2 + 1000n + 10^{100} = O\left(n^2\right)$$

### 1.3.1 Obere Schranke

$$O(g(n)) = \{f(n) | \exists c > 0 \exists n_0 > 0 \forall n > n_0 : 0 \leq f(n) \leq cg(n)\}$$

### 1.3.2 Untere Schranke

$$\Omega(g(n)) = \{f(n) | \exists c > 0 \exists n_0 > 0 \forall n > n_0 : 0 \leq cg(n) \leq f(n)\}$$

### 1.3.3 Genaue Schranke

$$\theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Wenn die Untere und Obere Schranke übereinstimmen, kann man dies als Genaue Schranke angeben.

### 1.3.4 Ergänzung

$$o(g(n)) = \{f(n) | \forall c > 0 \exists n_0 > 0 \forall n > n_0 : 0 \leq f(n) < cg(n)\}$$

Gibt eine Funktion an, die *auf jeden Fall* größer ist als die Eingabefunktion. Sie ist also von einem Höheren Grad als $O(g(n))$.

# 2 Allgemeines zu Sprachen

## 2.1 Grammatiken 13

### 2.1.1 Definition 13

Eine *Grammatik* ist ein 4-Tupel $G = (V, \Sigma, P, S)$, das folgende Bedingungen erfüllt.

- $V$ ist eine endliche Menge, die Menge der *Variablen*
- $\Sigma$ ist eine endliche Menge, das *Terminalalphabet*
- $V \cap \Sigma = \emptyset$
- $S \in V$ *Startsymbol*
- $P$ ist die endliche Menge der *Produktionen* oder *Regeln*. Formal: $P \subset (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ ist endlich.
- $\forall_P^{l \to r} : |l|_V \geq 1$

Seien $u, v \in (V \cup \Sigma)^*$. Wir definieren die Relation $u \Rightarrow_G v$ ($u$ geht unter $G$ unmittelbar über in $v$), falls

- $\exists_{(V \cup \Sigma)^*}^{x,y} : u = xyz \wedge v = xy'z$
- $(y \to y') \in P$

Wenn klar, ist welche Grammatik gemeint ist, so schreiben wir einfach $u \Rightarrow v$

Beispiel für Schreibweise (mit erweiterter Schreibweise / Backus-Naur-Form):

$$\begin{aligned} G &:= (V, \Sigma, P, S) \\ V &:= \{S, N, Z\} \\ \Sigma &:= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, \epsilon\} \\ P &:= \{S \to ZN; \\ &\quad Z \to \epsilon | -; \\ &\quad N \to 1|2|3|4|5|6|7|8|9|NN\} \end{aligned}$$

### 2.1.2 Ableitungen 14

Eine Folge von Wörtern $(w_0, w_1, \ldots, w_n)$ mit $w_0 = S, w_n \in \Sigma^*$ und $w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$ heißt *Ableitung* von $w_n$.

Ein Wort $w \in (V \cup \Sigma)^*$, das also noch Variablen enthält (wie es typischerweise im Verlauf einer Ableitung vorkommt) heißt auch *Satzform*.

Ein Wort $w \in (\Sigma)^*$, das ausschließlich *Terminale* enthält heißt auch *Terminalwort*.

- Ableiten ist ein *nichtdeterministischer* Prozess (es steht im allgemeinen nicht fest, auf was etwas abgeleitet werden muss).

### 2.1.3 Sprache einer Grammatik 14

Die von $G = (V, \Sigma, P, S)$ dargestellte (erzeugte, definierte) *Sprache* ist:

$$L(G) = \{w \in \Sigma^* | S \Rightarrow_G^* w\}$$

Dies bedeutet, das man mit einer beliebigen Kombination von Ableitungen, irgendwie zu dem Wort $w$ kommen können muss.

### 2.1.4 Darstellung von Ableitungen 16

Graphisch kann man sich das Ableiten wie ein endlich verzweigter, aber im allgemeinen unendlich großen Baum vorstellen. Mit $S$ als Wurzel, und Wörtern der Sprache als Blätter.

## 2.2 Chomsky-Hierarchie 17

Die Menge aller Grammatiken (und der dazugehörigen Sprachen) lässt sich in folgende Kategorien unterteilen. Die Sprachen werden abgekürzt mit $\mathcal{L}_n$. Dabei gilt:

$$\mathcal{P}(\Sigma^*) \supset \mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$$

### 2.2.1 Typ 0 oder rekursiv aufzählbar

Jede Grammatik ist automatisch vom Typ 0. Es gibt hier also keine Einschränkungen. (Außer das es eine korrekte Grammatik sein muss)

- sind nicht *entscheidbar* (ob ein Wort in der Sprach liegt oder nicht, siehe Wortproblem)

### 2.2.2 Typ 1 oder kontextsensitiv

wenn: $\forall_P^{(w_1 \to w_2)} : |w_1| \leq |w_2|$

- Typ 1 Grammatiken sind monoton, das heißt, das die Satzformen nie kürzer werden können.
- $\varepsilon \notin \mathcal{L}_1$ gilt nach der aktuellen Definition. Wenn dies doch erwünscht, dann muss folgende Sonderregel erlaubt werden:
  - $(S \to \varepsilon) \in P$ wenn gilt: $\forall_P^{(l \to r)} : r \neq S$
  - falls $S$ doch auf rechten Seiten vorkommt, einfach ein neues $S'$ definieren, als neue Startvariable benutzen, und folgende Regeln aufnehmen: $(S' \to S);(S' \to \varepsilon)$

### 2.2.3 Typ 2 oder kontextfrei

wenn: $\forall_P^{(w_1 \to w_2)} : w_1 \in V$

- Wenn Regeln der Form $A \to \varepsilon$ existieren kann man diese aus Grammatik entfernen um sie kontextfrei zu machen, indem die Regeln entsprechend angepasst werden. (Buch S. 19):
  - Zerlegen der Menge der Variablen $V$ in $V_1$ und $V_2$, so dass für alle $A \in V_1$ (und nur für diese) gilt $A \Rightarrow^* \varepsilon$. Dabei wie folgt vorgehen:
    - Wenn $(A \to \varepsilon) \in P$, so ist $A \in V_1$
    - Weitere Variablen findet man sukzessive dadurch, dass es in $P$ eine Regel $B \to A_1 \dots A_k, k \geq 1$, gibt mit $A_i \in V_1$ $(i = 1, \dots, k)$. Nach endlich vielen Schritten hat man $V_1$gefunden
    - Entfernen der Form $(A \to \varepsilon) \in P$ und fügen für jede Regel der Form $B \to xAy$ mit $B \in V, A \in V_1, xy \in (V \cup \Sigma)^+$ eine weitere Regel der Form $B \to xy$ zu $P$ hinzu. Dies muss entsprechend oft durchgeführt werden.
    - Ergebnis ist eine von $\varepsilon$-Regeln befreite Grammatik $G'$
- Wenn ein Wort $x$ auf verschieden Arten erzeugt werden kann, ist die Grammatik *mehrdeutig*.
- Eine kontextfreie Sprache $A$ heißt *inhärent mehrdeutig*, wenn jede Grammatik mit $L(G) = A$ mehrdeutig ist.

### 2.2.4 Typ 3 oder regulär

wenn: $\forall_P^{(w_1 \to w_2)} : w_1 \in V \wedge w_2 \in (\Sigma \cup \Sigma V)$

- Es gibt keine reguläre Sprache, die *inhärent Mehrdeutig* ist

## 2.3 Wortproblem 21

Das *Wortproblem* für Typ 1-Sprachen (und damit auch für Typ 2 ,3) ist entscheidbar. Es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes $x \in \Sigma^*$ in *endlicher* Zeit entscheidet, ob $x \in L(G)$ oder $x \notin L(G)$.

### 2.3.1 Algorithmus zum Lösen des Wortproblems bei Typ 1 - Grammatiken

Vorweg zur Notation: Bei $T_n^m$ handelt es sich um die Menge aller Satzformen, die maximal $m$ Symbole haben, und in maximal $n$ Ableitungsschritten aus dem Startsymbol $S$ ableitbar sind.

Um das Wortproblem für ein Wort $w \in \Sigma^*$ mit der Länge $|w| = m$ zu lösen, muss geprüft werden, ob es in der Menge $T_n^m$ (mit genügend großem $n$) enthalten ist. $T_n^m$ lässt sich wie folgt konstruieren:

- Die nullte Ableitung ist hier als identische Ableitung definiert. D. h., dass nur $S$ selber als Element der Menge $T_0^m$ infrage kommt: $T_0^m = \{S\}$
- Die folgenden Schritte können nun alle nach dem gleichen (iterativen) Schema ablaufen, und zwar solange, bis keine Änderung an $T_n^m$ mehr auftritt $(T_n^m = T_{n+1}^m)$.
  - Nehme die letzte Menge $(T_{n-1}^m)$, und ziehe die vorletzten Menge $(T_{n-2}^m)$ und alle Terminalwörter ab
  - bilde alle Ableitungen dieser Menge.
  - Dies Vereinigt mit der letzten Menge $(T_{n-1}^m)$ ergibt die neue Menge $(T_n^m)$
  - das ganze Formal:
    $$T_{n+1}^m = T_n^m \cup \mathrm{Abl}_m\left(T_n^m \setminus \left(T_{n-1}^m \cup \Sigma^*\right)\right)$$
    Unter $\mathrm{Abl}_m(X)$ versteht man die Menge aller Ableitungen von Elementen aus $X$ die maximal $m$ Symbole besitzen. Formal[1]:
    $$\mathrm{Abl}_m(X) = \left\{w \in (V \cup \Sigma)^* \mid |w| \leq m \wedge \exists_X^{w'} : w' \Rightarrow w\right\}$$
- Das Wortproblem für $w$ ist nun gelöst: $w \in T_n^m \Leftrightarrow w \in L(G)$

Dieser Algorithmus hat eine Laufzeit von $O(c^n)$

[1] Diese Definitionen entsprechen nicht ganz den Angaben in der Literatur, sind aber leichter handhabbar, da die Ableitung nur für eine viel kleiner Menge durchgeführt werden muss.

## 2.4 Syntaxbäume 23

Einer Ableitung eines Wortes $x$ in einer Typ 2 (oder 3) Grammatik $G$ kann man einen *Syntaxbaum* oder *Ableitungsbaum* zuordnen. Sei $x \in L(G)$ und sei $S \Rightarrow x_0 \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_n = x$ eine Ableitung eines Wortes $x$. Um den Baum zu erstellen gehe wie folgt vor:

- Beschrifte die Wurzel des Syntaxbaumes mit der Startvariable $S$
- Für $i = 1, \ldots, n$ gehe man wie folgt vor:
  - Falls im $i$-ten Ableitungsschritt (also beim Übergang von $x_{i-1}$ nach $x_i$) gerade die Variable $A$ durch ein Wort $z$ ersetzt wird (wegen $(A \to Z) \in P$):
    - Sehe im Syntaxbaum $|z|$ viele Söhne von $A$ vor
    - beschrifte diese mit den einzelnen Zeichen von $z$

Auf diese Weise erhält man eine Baum mit den Symbolen aus $x$ an den Blättern.

- Verschiedene Ableitungen können den selben Syntaxbaum besitzen
- *Linksableitung*: Immer am weitesten Links stehende Variable zuerst ableiten
- *Rechtsableitung:* Immer am weitesten Rechts stehende Variable zuerst ableiten
- Für ein und das selbe Wort kann es verschiedene Syntaxbäume geben. Die Grammatik ist dann *mehrdeutig*.
- Eine kontextfreie Sprache $A$ heißt *inhärent mehrdeutig*, wenn jede Grammatik mit $L(G) = A$ mehrdeutig ist.

## 2.5 Backus-Naur-Form 25

Hierbei handelt es sich um einen Formalismus zum kompakten niederschreiben von Typ 2-Grammatiken. Kurz *BNF*. Folgende neue Schreibweisen für Produktionen werden eingeführt:

- Metaregel ("|" = Metasymbol)
  $A \to \beta_1;\ A \to \beta_2;\ \ldots;\ A \to \beta_n =$
  $A \to \beta_1 | \beta_2 | \ldots | \beta_n$

Folgende Schreibweisen gehören zusätzlich zur *erweiterten BNF* (*EBNF*):

- Eventueller Teil:
  $A \to \alpha\,[\beta]\,\gamma =$
  $A \to \alpha\beta\gamma;\ A \to \alpha\gamma$
- Beliebig häufige Wiederholung:
  $A \to \alpha\,\{\beta\}\,\gamma =$
  $A \to \alpha\gamma;\ A \to \alpha B\gamma;\ B \to \beta;\ B \to \beta B$

Durch die *(E)BNF* werden exakt die Typ 2 - Sprachen dargestellt.

# 3 Reguläre Sprachen (Typ 3) 27

## 3.1 Deterministische (endliche) Automaten

### 3.1.1 Definition 27

Ein deterministischer (endlicher) Automat, kurz DFA (deterministic finite automation), wird auf ein Eingabewort angesetzt und erkennt, bzw. akzeptiert dieses Wort, oder auch nicht. Ein DFA M wird spezifiziert durch ein 5-Tupel

$$M = (Z, \Sigma, \delta, z_0, E)$$

- $Z$ Menge der *Zustände*
- $\Sigma$ *Eingabealphabet*
- $Z \cap \Sigma = \emptyset$
- $z_0 \in Z$ *Startzustand*
- $E \subseteq Z$ Menge der *Endzustände*
- $\delta : Z \times \Sigma \to Z$
  *(Zustands-)Überführungsfunktion*

### 3.1.2 Zustandsgraph 27

Darstellen lässt sich ein Automat durch einen *Zustandsgraphen*, (gerichtet und beschrifteter Graph) der sich wie folgt konstruieren lässt:

- Zeichne für jeden Zustand $z_n$ aus der Menge $z_n \in Z$ einen Kreis, und beschrifte ihn mit $z_n$
- Umkreise alle Kreise (doppelt Umkreisen) die einen $z_n \in E$ Endzustand enthalten
- Markiere den Startzustand $z_0$ mit einem nicht beschrifteten Pfeil der auf ihn Zeigt (entspringt aus dem Nichts)
- Für jede Überführungsvorschrift: $\delta(z_n, a) = z_m$
  - Zeichen einen Pfeil von $z_n$ nach $z_m$ (wenn noch nicht vorhanden)
  - beschrifte ihn (zusätzlich) mit $a$

### 3.1.3 Sprache eines DFA 29

Zu einem gegebenen DFA $M = (Z, \Sigma, \delta, z_0, E)$ definieren wir eine Funktion $\hat{\delta} : Z \times \Sigma^* \to Z$ durch eine induktive Definition wie folgt (erweitert die Definition von $\sigma$ von Einzelzeichen zu ganzen Wörtern):

Mit $z \in Z,\ x \in \Sigma^*,\ a \in \Sigma$

$$\begin{aligned} \hat{\delta}(z, \varepsilon) &= z \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x) \end{aligned}$$

Die Menge der akzeptierten Wörter bildet die durch den Automaten dargestellte oder definierte Sprache:

$$T(M) = \left\{ x \in \Sigma^* | \hat{\delta}(z_0, x) \in E \right\}$$

- eine Sprache $L$ wird durch einen endlichen Automaten erkannt $\Leftrightarrow$ die Sprache ist regulär ($L \in \mathcal{L}_3$)
- das lösen des Wortproblems ist mit Hilfe eines DFA in $O(n)$ möglich.

### 3.1.4 Grammatik zu einem gegebenen DFA 30

Aus einem DFA $M = (Z, \Sigma, \delta, z_0, E)$ lässt sich wie folgt eine (reguläre / Typ 3) Grammatik $G = (V, \Sigma, P, S)$ konstruieren:

- $\Sigma$ die Mengen der Terminale sind gleich
- $Z = V$ die Menge der Variablen entspricht der Menge der Zustände
- $S = z_0$ die Startvariable ist der Startzustand des Automaten
- Für jede Überführungsvorschrift: $\delta(z_n, a) = z_m$
  - falls $z_m \in E$:

    - füge eine Produktion $(z_n \to a)$ zu $P$ (der Menge der Produktionen) hinzu
    - wenn es eine Regel in $\sigma$ gibt, die von $z_m$ fortführt $(\exists_Z^{z_k} : \exists_\Sigma^b : \delta(z_m, b) = z_k)$
        - füge eine Produktion $(z_n \to az_m)$ zu $P$ (der Menge der Produktionen) hinzu
- falls $z_m \notin E$:
    - füge eine Produktion $(z_n \to az_m)$ zu $P$ (der Menge der Produktionen) hinzu

$M$ und $G$ beschreiben die gleiche Sprache:

$$T(M) = L(G)$$

## 3.2 Nichtdeterministische (endliche) Automaten

### 3.2.1 Definition 30

Bei einem nichtdeterministischen (endlichen) Automaten (*NFA* nondeterministic finite automation) ist es zugelassen, dass vom selben Zustand $z \in Z$ aus mehrere (oder auch garkeine) Pfeile ausgehen, die mit $a \in \Sigma$ beschriftet sind.

Ein NFA $M$ ist ein 5-Tupel $M = (Z, \Sigma, \delta, S, E)$:

- $Z$ *Zustandsmenge* (endlich)
- $\Sigma$ *Eingabealphabet* (endlich)
- $Z \cap \Sigma = \emptyset$
- $\delta : Z \times \Sigma \to \mathcal{P}(Z)$
  *(Zustands-)Überführungsfunktion*
- $S \subseteq Z$ *Startzustandsmenge*
- $E \subseteq Z$ *Endzustandsmenge*

### 3.2.2 Zustandsgraph

Vom Prinzip her wie beim DFA (siehe 3.1.2 auf der vorherigen Seite). Einzige Unterschiede sind:

- Es gibt mehrere Startsustände
- Es kann von einem Zustand zwei Pfeile mit der gleichen Beschriftung zu unterschiedlichen Zuständen geben

### 3.2.3 Sprache des NFA 31

Zu einem gegebenen NFA $M = (Z, \Sigma, \delta, S, E)$ definieren wie eine Funktion $\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ durch eine induktive Definition wie folgt (erweitert die Definition von $\sigma$ von Einzelzeichen zu ganzen Wörtern):

$$\begin{aligned} \hat{\delta}(Z', \varepsilon) &= Z' \qquad \forall Z' \subseteq Z \\ \hat{\delta}(Z', ax) &= \cup_{z \in Z'} \hat{\delta}(\delta(z, a), x) \end{aligned}$$

Die Menge der akzeptierten Wörter bildet die durch den NFA akzeptierte Sprache:

$$T(M) = \left\{ x \in \Sigma^* | \hat{\delta}(S, x) \cap E \neq \emptyset \right\}$$

- eine Sprache $L$ wird durch einen endlichen Automaten erkannt $\Leftrightarrow$ die Sprache ist regulär $(L \in \mathcal{L}_3)$
- Jede von einem NFA akzepierbare Sprache ist auch durch einen DFA akzeptierbar

### 3.2.4 Vom NFA zum DFA - Potenzmengenkonstruktion 32

Sei $M = (Z, \Sigma, \delta, S, E)$ ein gegebener NFA. Wir konstruieren einen DFA $M' = (Z', \Sigma, \delta', z_0', E')$, der ebenfalls die gleiche Sprache erkennt $T(M) = T(M')$ wie folgt (der neue DFA hat Mengen als Zustände):

- Terminalalphabet $\Sigma$ bleibt unverändert.
- $Z' = \mathcal{P}(Z)$
  die neuen Zustände sind alle möglichen Teilmengen der alten Zustandsmenge
- $z_0 = S$
  die Startzustandsmenge ist der neue Startzustand
- $E' = \{Z' \subseteq Z | Z' \cap E \neq \emptyset\}$
  Alle Zustandsmengen, die Endzustände enthalten sind auch neue Endzustände
- $\delta'(Z_k, a) = \cup_{z \in Z_k} \delta(z, a) = \hat{\delta}(Z_k, a) \quad (Z_k \in Z')$
  Das Bild eines Urbildes, ist die Menge von allen möglichen Ableitungen von Elementen aus dem Urbild.

Hierbei entstehen in der Regel sehr viele unnötige Zustände und Überführungsfunktionen. Besser ist den Algorithmus aus 3.2.5 zu nutzen.

Der Laufzeitkomplexität ist dieses Algorithmus ist $O(c^n)$.

### 3.2.5 Vom NFA zum DFA - effizientere Methode

Bei folgenden Algorithmus bietet es sich an, zunächst einen Graphen von $M$ zu zeichnen, und anhand diesem den Graphen von $M'$ anhand des Algorithmusses zu erstellen. Diesen Graphen sollte man dann erst zum Schluss als Funktion nierderschreiben.

Sei $M = (Z, \Sigma, \delta, S, E)$ ein gegebener NFA. Wir konstruieren einen DFA $M' = (Z', \Sigma, \delta', z_0', E')$, der ebenfalls die gleiche Sprache erkennt $T(M) = T(M')$ wie folgt (der neue DFA hat Mengen als Zustände):

- Terminalalphabet $\Sigma$ bleibt unverändert.
- $z_0' = S$
  die Startzustandsmenge ist der neue Startzustand
- Folgendes solange ausführen, bis sich an $\delta$ und $Z'$ nichts mehr ändert.
  - Für jedes Element $z_x \in Z'$:
    - Für jedes Terminal $t \in \Sigma$:
      - Bestimme für jedes Element $a$ der Menge die den akt. Zustand $z_x$ bildet, die Menge der neuen Zustände bezüglich dem Terminal $t$. $\cup_{a \in z_x} \{b \in Z | \delta(a, t) = b\}$
      - Fasse diese zu einer neuen Menge $z_{neu}$ zusammen. $z_{neu} = \cup_{a \in z_x} \{b \in Z | \delta(a, t) = b\}$
      - Wenn $z_{neu}$ noch nicht in $Z'$ enthalten ($z_{neu} \notin Z'$), füge es in die Zustandsmenge $Z'$ hinzu. $Z' \cup \{z_{neu}\}$
      - Wenn die Menge $z_{neu}$ ein Endzustand aus $E$ enthält und $z_{neu}$ noch nicht in $E'$ enthalten ist $\left(\left(\exists^a_{z_{neu}} : a \in E\right) \wedge z_{neu} \notin E'\right)$, dann füge $z_{neu}$ der Menge $E'$ hinzu.
      - Wenn es in $\delta$ noch keine Regel der Gestalt $\delta(z_x, t) = z_{neu}$ gibt, ergänze diese.

Der Laufzeitkomplexität ist dieses Algorithmus ist $O(c^n)$.

### 3.2.6 Von einer Grammatik zum NFA 35

Für jede reguläre Grammatik $G = (V, \Sigma, P, S)$ (Typ 3) gibt es einen NFA $M = (Z, \Sigma, \delta, S', E)$ mit $L(G) = M(G)$. Diese lässt sich wie folgt konstruieren:

- $Z = V \cup \{X\}, \quad X \notin V$
  Die Zustandsmenge besteht aus der Menge der Variablen, zuzüglich einer neuen Variable (hier $X$)

- $S' = \{S\}$
  Der Startzustand ist die Menge, die nur die Startvariable enthält
- $E = \begin{cases} \{S, X\}, & (S \to \varepsilon) \in P \\ \{X\}, & (S \to \varepsilon) \notin P \end{cases}$
  Die Menge der Endzustände enthält $X$ und wenn es eine Produktion der Form $(S \to \varepsilon) \in P$ gibt zusätzlich auch noch $S$
- $B \in \delta(A, a), \quad (A \to aB) \in P$
  Erstelle für alle Produktionen der Form $(A \to aB) \in P$ eine Regel für $\delta$ mit $B \in \delta(A, a)$
- $X \in \delta(A, a), \quad (A \to a) \in P$
  Erstelle für alle Produktionen der Form $(A \to a) \in P$ eine Regel für $\delta$ mit $X \in \delta(A, a)$

Der Laufzeitkomplexität ist dieses Algorithmus ist $O(n)$.

## 3.3 Reguläre Ausdrücke 36

Reguläre Ausdrücke sind spezielle Formen, mit denen (reguläre / Typ 3) Sprachen definiert werden können. Folgendes sind reguläre Ausdrücke (mit jeweils den von Ihnen beschriebenen Sprachen):

- $\gamma = \emptyset \Rightarrow L(\gamma) = \emptyset$
- $\gamma = \varepsilon \Rightarrow L(\gamma) = \{\varepsilon\}$
- $\forall_\Sigma^a : \gamma = a \Rightarrow L(\gamma) = \{a\}$
- $\gamma = \alpha\beta \Rightarrow L(\gamma) = L(\alpha) L(\beta)$
- $\gamma = (\alpha|\beta) \Rightarrow L(\gamma) = L(\alpha) \cup L(\beta)$
- $\gamma = (\alpha)^* \Rightarrow L(\gamma) = L(\alpha)^*$

Jedem regulären Ausdruck lässt sich ein NFA (und dadurch auch DFA) zuordnen, der die gleiche Sprache beschreibt.

## 3.4 Das Pumping Lemma 39

Das Pumping Lemma (Schleifenlemma, Lemma von Bar-Hillel, $uvw$-Theorem) ist ein wichtiges Hilfsmittel, um zu Zeigen, das eine Sprache nicht regulär ist.

Sei $L$ eine reguläre Sprache. Dann gibt es eine Zahl $n$, so dass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, so dass folgende Eigenschaften erfüllt sind:

- $|v| \geq 1$
- $|uv| \leq n$
- $\forall i \geq 0 : uv^i w \in L$

Es gibt auch Sprachen die das Pumping Lemma erfüllen, und nicht regulär sind. Das bedeutet, das mit dem Pumping Lemma nur das nicht regulärsein von Sprachen gezeigt werden kann.

$$L \in \mathcal{L}_3 \Rightarrow \left[\exists_{\mathbb{N}}^n : \left(\forall_L^x |x| \geq n : \left(\exists_{\Sigma^*}^{u,v,w} x = uvw \wedge |v| \geq 1 \wedge |uv| \leq n : \left(\forall i \geq 0 : \left(uv^i w \in L\right)\right)\right)\right)\right]$$

Umgekehrt gilt:

$$\left[\forall_{\mathbb{N}}^n : \left(\exists_L^x |x| \geq n : \left(\forall_{\Sigma^*}^{u,v,w} x = uvw \wedge |v| \geq 1 \wedge |uv| \leq n : \left(\exists i \geq 0 : \left(uv^i w \notin L\right)\right)\right)\right)\right] \Rightarrow L \notin \mathcal{L}_3$$

- Eine Sprache, bei der mehrere Exponenten in irgendwelchen Beziehungen zueinander stehen, sind (laut Pumping Lemma) nicht regulär.

## 3.5 Äquivalenzrelation und Minimalautomat 42

### 3.5.1 Definition Äquivalenzrelation 42

Es gilt $xR_Ly$ genau dann, wenn für alle Wörter $z \in \Sigma^*$ gilt: $xz \in L \Leftrightarrow yz \in L$:

$$(xR_Ly) \Leftrightarrow (\forall^z_{\Sigma^*} : xz \in L \Leftrightarrow yz \in L)$$

Mit Hilfe von $R_L$ wird die Sprache $L$ in disjunkte Äquivalenzklassen unterteilt. Wenn der Index von $R_L$ ($\text{Index}(R_L)$ ist die Anzahl solcher Äquivalenzklassen, siehe 1.2.4 auf Seite 5) endlich ist, ist $L$ eine reguläre Sprache:

$$(\text{Index}(R_L) < \infty) \Leftrightarrow L \in \mathcal{L}_3$$

### 3.5.2 Algorithmus Minimalautomat 46

Gegeben sei ein DFA $M = (Z, \Sigma, \delta, z_0, E)$, der nach folgenden Algorithmus in eine Zustandsminimalen DFA $M' = (Z', \Sigma, \delta', z_0, E')$ überführt wird.

- entferne alle Zustände aus $Z$ die vom Startzustand aus nicht erreicht werden können.
- Stelle eine Kreuztabelle aller Zustandspaare $\{z, z'\}$ mit $z \neq z'$ von $M$ auf (entspricht einer quadratischen Matrize, von der nur das ober/unter Dreieck ohne die Hauptdiagonale genommen wurde)
- Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$ (oder umgekehrt).
- Wiederhole folgendes, bis sich keine Änderung in der Tabelle mehr ergibt
  - Für jedes noch unmarkierte Paar $\{z, z'\}$ und jedes $a \in \Sigma$ (es reicht, das dies für ein Terminal gilt) teste, ob $\{\delta(z, a), \delta(z', a)\}$ bereits markiert ist. Wenn ja: markiere auch $\{z, z'\}$
- Alle jetzt noch unmarkierten Paare können jeweils zu einem Zustand verschmolzen werden. Das heißt, das man wenn z.B. das Paar $(z, z')$ nicht markiert ist, wird für beide zusammen ein neuer Zustand geschaffen.

Dieser Algorithmus hat (bei geeigneter Implementierung (siehe Hopcroft/Ullmann)) die Laufzeitkomplexität $O(n^2)$

## 3.6 Abschlusseigenschaften 48

Die regulären Sprachen sind abgeschlossen unter:

- Vereinigung
  $L(G_1) \cup L(G_2)$
- Schnitt
  Wort muss in beiden Sprachen vorkommen
- Komplement
  Bei Grammatik: Endzustände ersetzen gegen Variablen ohne die alten Endzustände
- Produkt
  Konkatenation der Wörter aus den einzelnen Sprachen
  $L(G_1)\, L(G_2)$
- Stern
  Wiederholen der Wörter

Umkehrschluss nicht möglich: Wenn nichtreguläre Sprachen kombiniert werden, kann das Resultat trotzdem eine reguläre Sprache sein. Das einzige was sich Aussagen lässt ist, dass wenn eine Sprache nicht Regulär ist, mindestens eine Ihrer Komponenten auch nicht regulär war.

## 3.7 Entscheidbarkeit 49

### 3.7.1 Wortproblem

Das *Wortproblem* (gegeben: $x$, gefragt: ist $x \in L(G)$ bzw. $x \in T(M)$) ist mit Hilfe eines DFA leicht möglich. Verfolge einfach Zeichen für Zeichen die Zustandsübergänge im Automaten, die durch die Eingabe $x$ hervorgerufen werden. Falls ein Endzustand erreicht wird, liegt $x$ in der Sprache. Die Laufzeitkomplexität beträgt hier $O(n)$.

### 3.7.2 Leerheitsproblem

Das *Leerheitsproblem* stellt bei gegebenem $G$ (bzw. $M$) die Frage, ob $L(G) = \emptyset$ (bzw. $T(M) = \emptyset$). Sei $M$ ein gegebener Automat für die Sprache. $T(M)$ ist genau dann leer, wenn es keinen Pfad vom (einem) Startzustand zu einem Endzustand gibt.

### 3.7.3 Endlichkeitsproblem

Das *Endlichkeitsproblem* stellt bei gegebenem $G$ (bzw. $M$) die Frage, ob $|L(G)| < \infty$ (bzw. $T(M) < \infty$), also ob die definierte Sprache endlich oder unendlich ist.

Hier gibt es zwei Verfahren:

- Sei $n$ die Pumping Lemma (siehe 3.4 auf Seite 13) Zahl zu $L$. Es gilt:

$$|L| = \infty \Leftrightarrow \exists_L^x : |x| \geq n \wedge |x| < 2n$$

  - Diese Argument verläuft so, dass alle Wörter der Länge $\geq n$ und $< 2n$ auf Mitgliedschaft in $L$ geprüft werden müssen.
  - Die Laufzeitkomplexität beträgt $O(c^n)$

- Es ist $|T(M)| = \infty$ genau dann, wenn es in dem von Startzustand erreichbaren Teilgraphen einen Zyklus gibt, und wenn von diesem Zyklus aus ein Endzustand erreichbar ist.

$$\exists_{\text{Startzustand}}^{z_0} : \exists_{\text{Zustand}}^{z_1} : \exists_{\text{Endzustand}}^{z_2} : \quad z_0 \Rightarrow^* z_1 \Rightarrow^+ z_1 \Rightarrow^* z_2$$

  - wesentlich effizienter als erste Methode

### 3.7.4 Schnittproblem

Das *Schnittproblem* stellt bei gegebenen $G_1, G_2$ (bzw. $M_1, M_2$) die Frage, ob $L(G_1) \cap L(G_2) = \emptyset$ (bzw. $T(M_1) \cap T(M_2) = \emptyset$). Hierfür muss man eine Sprache (Grammatik) konstruieren, die aus $G_1$ und $G_2$ hervorgeht, und die beide Grammatiken sozusagen simultan durchläuft. Hiermit ist dieses Problem auf das Leerheitsproblem zurückgeführt.

### 3.7.5 Äquivalenzproblem

Das *Äquivalenzproblem* stellt bei gegebene $G_1, G_2$ (bzw. $M_1, M_2$) die Frage, ob $L(G_1) = L(G_2)$ (bzw. $T(M_1) = T(M_2)$). Hierzu gibt es mehrere Lösungsmöglichkeiten:

- Per Minimalautomat:
  - Wenn Sprachen nicht als DFA's vorliegen, diese dahingehend transformieren.
  - Jeweils den Minimalautomaten konstruieren
  - Diese auf Isomorphie (Umbenennung der Variablen) untersuchen, wenn ja, dann sind die Sprachen gleich.
- Rückführung auf Leerheitsproblem:

$$L_1 = L_2 \Leftrightarrow \left(L_1 \cap \overline{L_2}\right) \cup \left(\overline{L_1} \cap L_2\right) = \emptyset$$

Am Effizientesten ist die Methode des Minimalautomaten, wenn die Sprache bereits als DFA vorliegt. Allgemein ist das Problem *NP-hart*, hat also eine Laufzeitkomplexität von $O(c^n)$.

# 4 Kontextfreie Sprachen (Typ 2) 51

Werden gebraucht, um korrekt geklammerte Ausdrücke zu erstellen.

## 4.1 Normalformen 52

Für die Normalformen müssen folgende Regeln gelten (durch Umformungen erreichbar):

- $\varepsilon$-Regeln entfernen (siehe 2.2.3 auf Seite 8)
- Kettenregeln ($A \to B$) entfernen
  - Bestimme äquivalente Variable:
    - $A \leftrightarrow B$ gdw. $A \Rightarrow^* B$ und $B \Rightarrow^* A$
    - $[A] := \{B \in V | A \leftrightarrow B\}$
    - Wähle ein $B_0 \in [A]$aus, uns ersetze alle Vorkommen von $B \in [A]$ durch $B_0$
  - Sei $V = \{A_1, A_2, \ldots, A_n\}$ die aktuelle Menge der Variablen, und zwar so sortiert, dass aus $(A_i \to A_j) \in P$ folgt $i < j$
  - Wir gehen nun von von hinten nach vorne vor und eliminieren für $k = n-1, \ldots, 1$ alle Regeln der Form $(A_k \to A_{k'}), k' > k$. Seien die Regeln, mit $A_{k'}$auf der linken Seite, gegeben durch $A_{k'} \to x_1|x_2|\ldots|x_k$. Wir fügen dann die Regeln $A_k \to x_1|x_2|\ldots|x_k$ hinzu.

### 4.1.1 Chomsky Normalform 53

Zu jeder kontextfreien Grammatik $G$ mit $\varepsilon \notin L(G)$ existiert eine *Chomsky Normalform* (CNF). Die Chomsky Normalform liegt vor, wenn alle Regeln eine der folgenden Formen haben:

- $A \to AB$
- $A \to a$

Ist $w \in L(G)$, so gilt:

$$S \Rightarrow_G^{2|w|-1} w$$

Ein Algorithmus für die Umformung findet sich im Buch (vorher Kettenregeln, und $\varepsilon$-Regeln entfernen!)

### 4.1.2 Greifenbach Normalform 54

Zu jeder kontextfreien Grammatik $G$ mit $\varepsilon \notin L(G)$ existiert eine *Greifenbach Normalform*. Diese liegt vor, wenn alle Regeln eine der folgenden Formen haben:

- $A \to aB_1B_2\ldots B_k \quad (k \geq 0)$

Die Greifenbach Grammatik ist eine Erweiterung der regulären Grammatik, dort wäre nur $k = 0$ und $k = 1$ zugelassen.

Ein Algorithmus für die Umformung findet sich im Buch.

## 4.2 Das Pumping Lemma (uvwxy-Theorem) 56

Sei $L$ eine reguläre Sprache. Dann gibt es eine Zahl $n \in \mathbb{N}$, sodass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen als $x = uvwxy$ mit folgenden Eigenschaften:

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. $\forall i \geq 0 : uv^i wx^i y \in L$

$$L \in \mathcal{L}_2 \Rightarrow \left[\exists_{\mathbb{N}}^{n} : \left(\forall_{L}^{z} |z| \geq n : \left(\exists_{\Sigma^*}^{u,v,w,x,y} z = uvwxy \wedge |vx| \geq 1 \wedge |vwx| \leq n : \left(\forall i \geq 0 : \left(uv^i wx^i y \in L\right)\right)\right)\right)\right]$$

Das Pumping Lemma (Schleifenlemma, Lemma von Bar-Hillel, $uvwxy$-Theorem) ist ein wichtiges Hilfsmittel, um zu Zeigen, das eine Sprache nicht regulär ist.

Die Kontraposition lautet wie folgt:

$$\left[\forall_{\mathbb{N}}^{n} : \left(\exists_{L}^{z} |z| \geq n : \left(\forall_{\Sigma^*}^{u,v,w,x,y} z = uvwxy \wedge |vx| \geq 1 \wedge |vwx| \leq n : \left(\exists i \geq 0 : \left(uv^i wx^i y \notin L\right)\right)\right)\right)\right] \Rightarrow L \notin \mathcal{L}_2$$

### 4.2.1 Satz für einelementige Alphabete 62

Jede kontextfreie Sprache über einem einelementigen Alphabet ist bereits regulär.

### 4.2.2 Sprachen mit Wiederholungen

- Eine Sprache, bei der keine Exponenten in irgendwelchen Beziehungen zueinander stehen, sind (laut Pumping Lemma) regulär.
- Eine Sprache, bei der genau zwei Exponenten in irgendwelchen Beziehungen zueinander stehen, sind (laut Pumping Lemma) Kontext frei.
- Eine Sprache, bei der drei oder mehr Exponenten in irgendwelchen Beziehungen zueinander stehen, sind (laut Pumping Lemma) Kontext sensitiv.

## 4.3 Abschlusseigenschaften 62

Die kontextfreien Sprachen sind abgeschlossen unter:

- Vereinigung
  $L(G_1) \cup L(G_2)$
- Produkt
  Konkatenation der Wörter aus den einzelnen Sprachen
  $L(G_1)\, L(G_2)$
- Stern
  Wiederholen der Wörter

Umkehrschluss nicht möglich: Wenn nicht kontextfreie Sprachen kombiniert werden, kann das Resultat trotzdem eine kontextfreie Sprache sein. Das einzige was sich Aussagen lässt ist, dass wenn eine Sprache nicht kontextfreie ist, mindestens eine Ihrer Komponenten auch nicht kontextfrei war.

- Der Durchschnitt einer (deterministischen) kontextfreien Sprache mit einer regulären Sprache ist wieder (deterministisch) kontextfrei

## 4.4 Der CYK-Algorithmus 64

Der *CYK-Algorithmus* (Cocke, Younger, Kasami) ist eine Möglichkeit das Wortproblem für kontextfreie Sprachen zu lösen. Dafür muss die Grammatik in der Chomsky Normalform vorliegen (gegeben falls umformen).

1. Erstelle eine Tabelle, deren Zeilenanzahl $z$ gleich der Länge des Wortes $w$ ist, für das dass Wortproblem gelöst werden soll. Die Spaltenanzahl $s$ ist um eins höher.
2. Trage in die erste Zeile die einzelnen Terminale von $w$ ein.
3. Trage in die Zweite Spalte alle Variablen ein, aus denen man die Variablen oberhalb von ihnen ableiten kann.
4. Fülle die restlichen Zellen nun zeilenweise aus. Beachte, das man in der $j$ ten Zeile immer nur die ersten $|w| - j + 1$ Zellen ausfüllen braucht (kann).
   (a) in die Aktuelle Zelle $x_{j,i}$ kommen alle Variablen $V$ hinein, aus denen man $AB$ ableiten kann ($V \to AB$), wobei $A$ ein Element der Menge aller Zelleninhalte oberhalb ($\uparrow$) von $x_{j,i}$ und $B$ ein Element der Menge aller Zelleninhalte diagonal rechts oberhalb ($\nearrow$) von $x_{j,i}$ ist.
5. Wenn das Startsymbol $S$ ein Element der ersten Zelle der letzten Zeile ist, dann ist $w \in L(G)$ ansonsten $w \notin L(G)$.

## 4.5 Kellerautomaten 67

### 4.5.1 Definition 67

Ein (nichtdeterministischer) *Kellerautomat*, kurz PDA (pushdown automation), wird auf ein Eingabewort ist eine Erweiterung von DFA / NFA um das einen Keller als Speicher. Ein PDA $M$ wird spezifiziert durch ein 6-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$$

- $Z$ endliche Menge der (internen) *Zustände*
- $\Sigma$ endliches *Eingabealphabet*
- $\Gamma$ endliches *Kelleralphabet*
- $z_0 \in Z$ *Startzustand*
- $\delta : Z \times (\Sigma \cap \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ Überführungsfunktion
  - z.B. $(z', B_1 \dots B_k) \in \delta(z, a, A)$
  - $(z', B_1 \dots B_k) \in \delta(z, \epsilon, A)$ spontaner Zustandswechsel ohne lesen eines neuen Terminals
- $\# \in \Gamma$ das unterste *Kellersymbol*

Eine *Konfiguration* $k$ eines Kellerautomaten ist durch ein Tripel $k \in Z \times \Sigma^* \times \Gamma^*$ gegeben.

- Startkonfiguration zur Eingabe $x \in \Sigma^*$: $(z_0, x, \#)$

Die *Rechenschrittrelation* $\vdash$ist wie folgt definiert:

$$(z, a_1 \dots a_n, A_1 \dots A_m) \vdash \begin{cases} (z', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m) & (z', B_1 \dots B_k) \in \delta(z, a_1, A_1) \\ (z', a_1 a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m) & (z', B_1 \dots B_k) \in \delta(z, \varepsilon, A_1) \end{cases}$$

- es wird immer in Abhängigkeit vom obersten Symbol auf dem Stack, dem akt. internen Zustand, und *eventuell* (wenn es eine Regel mit einem $\varepsilon$ an zweiter Stelle gibt, dann nicht) dem Eingabeterminal ein neuer interner Zustand und ein Satz an Symbolen bestimmt, die auf den Stack gelegt werden, nachdem das oberste Symbol auf dem Stack entfernt wurde.

$$k \vdash^* k' \Leftrightarrow \left(\exists m \geq 0 : \exists k_0, k_1, \ldots, k_m : k = k_0 \wedge k' = k_m \wedge \forall^i_{[0,m-1]} : k_i \vdash k_{i+1}\right)$$

Die von $M$ (durch leeren Keller) *akzeptierte Sprache* ist:

$$N(M) = \{x \in \Sigma^* | \exists^z_Z : (z_0, x, \#) \vdash^* (z, \varepsilon, \varepsilon)\}$$

- $L$ kontextfrei $\Leftrightarrow$ $L$ wird von einem nichtdeterministischen Kellerautomaten akzeptiert
- Zu jedem PDA $M$ gibt es einen PDA $M'$ mit nur einem Zustand, sodass $N(M) = N(M')$ gilt
  - durch Transformation in Grammatik, und anschließende Rücktransformation.

### 4.5.2 Von Grammatik zum Kellerautomaten 72

Gegeben ist eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ (Chomskynormalisierung *nicht* erforderlich, sogar eher kontraproduktiv, da die alles viel komplizierter macht). Ausgegeben wird ein PDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$.

- $Z = \{z\}$
- $\Sigma$ wird 1:1 übernommen
- $\Gamma = V \cup \Sigma$
- $z_0 = z$
- $\# = S$ Startvariable wird als unterstes Kellersymbol genommen
- Erstellen der Überführungsfunktion $\delta$
  - Für alle Produktionen $(l \to r) \in P$
    - Füge eine Regel der Form $\delta(z, \varepsilon, l) = (z, r)$ hinzu
  - Für alle Terminale $a \in \Sigma$
    - Füge eine Regel der Form $\delta(z, a, a) = (z, \varepsilon)$ hinzu

### 4.5.3 Von Kellerautomaten zur Grammatik 73

Gegeben ist ein PDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$. Ausgegeben wird eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$.

- $\Sigma$ wird 1:1 übernommen
- $S$ wird mit $S$ besetzt
- $V = \{S\} \cup (Z \times \Gamma \times Z)$
- Erstelle Produktionsregeln $P$
  - Für alle Zustände $z \in Z$
    - Füge eine Regel der Form $S \to (z_0, \#, z)$ hinzu
  - Für alle $(z', \varepsilon) \in \delta(z, a, A)$
    - Füge eine Regel der Form $(z, A, z') \to a$ hinzu
  - Für alle $(z_1, B) \in \delta(z, a, A)$ und alle $z' \in Z$
    - Füge eine Regel der Form $(z, A, z') \to a\,(z_1, B, z')$ hinzu
  - Für alle $(z_1, BC) \in \delta(z, a, A)$ und alle $z', z_2 \in Z$
    - Füge eine Regel der Form $(z, A, z') \to a\,(z_1, B, z_2)\,(z_2, C, z')$ hinzu

Ist $G$ in Greibach Normalform, so kann ein PDA $M$ mit $L(G) = N(M)$ konstruiert werden, der keine $\varepsilon$-Übergänge hat.

### 4.6 Deterministisch kontextfreie Sprachen 76

Ein Kellerautomat $M$ heißt *deterministisch*, falls für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$ gilt:

$$|\delta(z, a, A)| + |\delta(z, \varepsilon, A)| \leq 1$$

Ferner hat ein deterministischer Kellerautomat (Deterministic Push Down Automation = DPDA) eine ausgezeichnete Menge von *Endzuständen* $E \subseteq Z$.

$$T(M) = \{x \in \Sigma^* | \exists z \in E : (z_0, x, \#) \vdash^* (z, \varepsilon, \alpha)\}$$

ist die von einem DPDA akzeptierte Sprache.

Eine Sprache $L \subseteq \Sigma^*$ heißt *deterministisch kontextfrei*, wenn es einen DPDA $M$ gibt mit $L = T(M)$.

- $\mathcal{L}_2 \subset$ deterministisch kontextfreie Sprachen $\subset \mathcal{L}_3$
- Stimmen mit den $LR(k)$-Sprachen überein (siehe Compilerbau)
- Die deterministisch kontextfreien Sprachen sind
  - unter Komplementbildung abgeschlossen
  - *nicht* unter Durchschnitt abgeschlossen
  - *nicht* unter Komplementbildung abgeschlossen
- Der Durchschnitt einer (deterministischen) kontextfreien Sprache mit einer regulären Sprache ist wieder (deterministisch) kontextfrei

### 4.7 Entscheidbarkeit 78

**Wortproblem** entscheidbar (CYK-Algorithmus)

**Leerheitsproblem** entscheidbar

- Idee: markiere Variable $A$ mit $\{x \in \Sigma^* | A \Rightarrow^* x\} \neq \emptyset$

**Endlichkeitsproblem** entscheidbar

- $|L| = \infty \Leftrightarrow \exists x \in L : n \leq |x| < 2n$

**Gleichheit** Wenn $L_1$ det. kontextfrei, und $L_2$ regulär. Ist $L_1 = L_2$ entscheidbar? Ja, siehe Buch.

**Äquivalenzproblem** für DPDA's entscheidbar

## 5 Kontextsensitive Sprachen (Typ 1) und Typ 0 Sprachen 79

### 5.1 Kuroda Normalform 79

Eine Typ 1 Grammatik ist in *Kuroda Normalform*, falls alle Regeln eine der 4 Formen haben:

- $A \to a$
- $A \to B$
- $A \to BC$
- $AB \to CD$

Hierbei stehen $A, B, C, D$ für Variablen und $a$ für ein Terminalsymbol.

- Für jede Typ 1 Grammatik $G$ mit $\varepsilon \neq L(G)$ gibt es eine Grammatik $G'$ in Kuroda Normalform mit $L(G) = L(G')$
  - Durch aufbrechen der Regeln, ähnlich Chomsky-Normalisierung

## 5.2 Turingmaschine 81

### 5.2.1 Definition 81

Eine *Touringmaschine* (Kurz: TM) ist gegeben durch ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$$

- $Z$ die endliche *Zustandsmenge*
- $\Sigma$ das *Eingabealphabet*
- $\Gamma \supset \Sigma$ das *Arbeitsalphabet*
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, N, R\}$ im deterministischen Fall
  (bzw. $\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$ im nichtdeterministischen Fall)
  die *Überführungsfunktion*
- $z_0 \in Z$ der *Startzustand*
- $\square \in (\Gamma \backslash \Sigma)$ das *Blank* (*Leerzeichen*)
- $E \subseteq Z$ die Menge der *Endzustände*

Die Überführungsfunktion bedeutet informal:
$\delta$(alter Zustand,Symbol vor Lesekopf)=(neuer Zustand,neues Symbol auf Band,Kopfbewegung LinksRechtNeutral)

Eine *Konfiguration* der TM $M$ ist ein Wort $k \in \Gamma^* Z \Gamma^+$. $k$ deckt den von $\square$ verschiedenen Teil des Bandes ab. Dabei sind die einzelnen Teile des Wortes: Band Links vom Lesekopf; Akt. Zustand; Band unter und Rechts vom Lesekopf.

Die *Startkonfiguration* für Eingabe $x \in \Sigma^*$ ist $z_0 x$.

Die *Berechnungsrelation* $\vdash$ ersetzt den Zustand durch den neuen Zustand, ersetzt das Symbol auf dem Band durch das neue Symbol, und führt die Kopfbewegung aus, indem bei L (R) ein Zeichen von Links nach Rechts neben dem Zustand (bzw. andersherum) bewegt wird. Falls sich der Kopf am Bandrand befindet, können an den Rändern zusätzliche $\square$ eingefügt werden.

Die von einem TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ *akzeptierte Sprache* ist definiert als

$$T(M) = \{x \in \Sigma^* | z_0 x \vdash^* \alpha z \beta \, (\alpha, \beta \in \Gamma^*, z \in E)\}$$

- Die durch allgemeine TMen akzeptierten Sprachen sind genau die Typ 0 Sprachen

### 5.2.2 Linear Beschränkte Turingmaschinen 84

Eine nichtdeterministische TM $M$ heißt *linear beschränkt*, wenn für alle $a_1 a_2 \dots a_{n-1} a_n \in \Sigma^+$ und alle Konfigurationen $\alpha z \beta$ mit $z_0 a_1 a_2 \dots a_{n-1} \hat{a_n} \vdash^* \alpha z \beta$ folgendes gilt: $|\alpha\beta| = n$.

Hierbei ist $\Sigma' = \Sigma \cup \{\hat{a} | a \in \Sigma\}$ (Alphabet um modifizierte Symbole ergänzt)

$T(M) = \{a_1 a_2 \dots a_{n-1} a_n \in \Sigma^* | z_0 a_1 \dots a_{n-1} \hat{a}_n \vdash^* \alpha z \beta \, (\alpha, \beta \in \Gamma^*, z \in E)\}$

Informal bedeutet dies, das der TM nur ein Band der Länge $n$ zurverfügung steht, und alle Berechnungen auf diesem Durchgeführt werden. Dazu muss Das letzte Zeichen auf dem Band markiert werden, damit die Maschine das Ende erkennt. Das erste Zeichen ist wird als erstes beim Start gekennzeichnet, da die Maschine hier beginnt.

- Die von linear beschränkten, nichtdeterministischen TMen (LBAs) akzeptierten Sprachen sind genau die kontextsensitiven Sprachen.

## 5.3 Umwandlung TM in Grammatik und umgekehrt 84

Siehe hierzu im Skript.

### 5.4 Abschlusseigenschaften 86

Die Klasse der Kontextsensitiven Sprachen ist unter Komplementbildung abgeschlossen.



# 6 Überblick über Sprachen und dazugehörige Werkzeuge 88

### 6.1 Beschreibungsmittel 89

| | |
|---|---|
| Typ 3 | reguläre Grammatik, DFA, NFA, reguläre Ausdrücke |
| det. kontextfrei | $LR(k)$- Grammatik |
| Typ 2 | kontextfreie Grammatik, Kellerautomat (PDA) |
| Typ 1 | kontextsensitive Grammatik, linear beschränkter Automat (LBA) |
| Typ 0 | Typ 0 - Grammatik, Turingmaschine (TM) |

### 6.2 (Nicht)Determinismus 89

| Nichtdet. Automat | Determ. Automat | äquivalent? |
|---|---|---|
| NFA | DFA | ja |
| PDA | DPDA | nein |
| LBA | DLBA | ? |
| TM | DTM | ja |

### 6.3 Abschlusseigenschaften 89

| | Schnitt | Vereinigung | Komplement | Produkt | Stern |
|---|---|---|---|---|---|
| Typ 3 | ja | ja | ja | ja | ja |
| Det. kf. | nein | nein | ja | nein | nein |
| Typ 2 | nein | ja | nein | ja | ja |
| Typ 1 | ja | ja | ja | ja | ja |
| Typ 0 | ja | ja | nein | ja | ja |

### 6.4 Entscheidbarkeit 90

| | Wortproblem | Leerheitsproblem | Äquivalenzproblem | Schnittproblem |
|---|---|---|---|---|
| Typ 3 | ja mit $O(n)$ | ja | ja | ja |
| Det. kf. | ja mit $O(n)$ | ja | ja | nein |
| Typ 2 | ja mit $O(n^3)$ | ja | nein | nein |
| Typ 1 | ja mit $O(c^n)$ | nein | nein | nein |
| Typ 0 | nein | nein | nein | nein |

# Index