Alphabet, Wörter & Sprachen

Begriff	Erklärung / Beispiel	Begriff	Erklärung / Beispiel
Alphabet Σ	endliche, nichtleere Menge an <mark>Symbolen</mark>	echtes Suffix v_e	v_e ist ein echtes Suffix von w, wenn $v_e \neq w$ gilt.
Wort w	ist eine endliche Folge von <mark>Symbolen</mark>	Σ^k	Menge aller Wörter mit der Länge k.
leeres Wort $arepsilon$	enthält keine <mark>Symbole,</mark> gehört zu <u>jedem</u> Alphabet	Σ^*	Menge aller Wörter über dem Alphabet Σ. (Kleenesche Hülle)
w	Länge eines Wortes abc = 3	Σ^+	Menge aller Wörter über dem Alphabet Σ minus dem leeren Wort. $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$
$ w _x$	Häufigkeit des Symbols x in dem Wort w . $ 100110 _1 = 3$	$x \circ y$	Konkatenation von zwei beliebigen Wörter
w^R	Spiegelung des Wortes w $(abc)^R = cba$	Wortpotenz x^n	x ist ein beliebiges Wort über Σ ($x = ab$) $x^3 = ababab$
Teilwort (Infix) v	v ist ein Teilwort von w, wenn w = xvy für beliebige Wörter x, y gilt.	Sprache L	ist eine Teilmenge von Σ^* $L \subseteq \Sigma^*$
echtes Teilwort v_e	v_e ist ein echtes Teilwort von w, wenn $v_e \neq w$ gilt.	leere Sprache Ø	ist die Sprache über jedem Alphabet.
Präfix v	v ist ein Präfix von w, wenn w = vy für beliebige Wörter y gilt.	Konkatenation von Sprachen	$AB = \{uv \mid u \in A\&v \in B\}$ Nennt man eine Konkatenation der zwei Sprachen A & B.
echtes Präfix v_e	v_e ist ein echtes Präfix von w , wenn $v_e \neq w$ gilt.	A^*	die Kleenische Hülle A^* der Sprach A ist durch $\{\varepsilon\} \cup A \cup AA \cup AAA \cup$ definiert.
Suffix v	v ist ein Suffix von w, wenn w = xv für beliebige Wörter x gilt.	Entscheidungsproblem	INPUT: Sprache L, Wort x OUTPUT: JA, wenn $x \in L$ oder NEIN, wenn $x \notin L$.

Reguläre Ausdrücke

Reguläre Ausdrücke sind Sprachen, welche Sprachen beschreiben oder endlich repräsentiert.

Syntax:

- 1 oder $0 = 1 \mid 0$
- beliebig oft $1 = 1^*$

Rechenregel:

- L(R|S) = L(S|R)
- L(R(ST)) = L((RS)T)
- L(R|(S|T)) = L((R|S)|T)
- L(R(S|T)) = L(RS|ST)
- $L((R^*)^*) = L(R^*)$
- L(R|R) = L(R)

Abschlusseigenschaften:

 L_1 und L_2 sind zwei **reguläre** Sprachen.

- $L_1 \cup L_2$ ist auch **regulär**
- L_1L_2 ist auch **regulär**
- L_1^* ist auch **regulär**
- $\overline{L} = \Sigma^* \setminus L$ ist au **regulär**
- $L_1 \cap L_2$ ist auch **regulär**
- $L_1 \setminus L_2$ ist auch **regulär**

Endliche Automaten

Endliche Automaten sind gleich mächtig wie reguläre Ausdrücke.

Definition (Endlicher Automat)

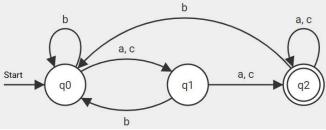
Ein (deterministischer) endlicher Automat (EA) ist ein Quintupel

$$M=(Q, \Sigma, \delta, q_0, F)$$

mit

- endlichen Menge von Zuständen $Q = \{q_0, q_1, \dots, q_n\}$ $(n \in \mathbb{N})$
- Eingabealphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$ $(m \in \mathbb{N})$
- Übergangsfunktion $\delta \colon Q \times \Sigma \to Q$
- Startzustand $q_0 \in Q$
- Menge der **akzeptierenden Zustände** $F \subseteq Q$

Beispiel DEA:



Die <u>Berechnung</u> für das Wort *abac* mit dem Beispiel-Automaten sieht wie folgt aus:

$$(q0, abac) \vdash (q1, bac) \vdash (q0, ac) \vdash (q1, c) \vdash (q2, \varepsilon)$$
 $\rightarrow Akzeptiert$

$\varepsilon - NEA$

Den ε – NEA zuerst in ein NEA umwandeln, in dem man alle ε -Übergänge eliminiert.

Danach kann man die <u>Teilmengenkonstruktion</u> wieder anwenden.

Nichtdeterminismus

Automaten sind deterministisch wenn gilt:

- Ein Automat über dem beliebigen Alphabet Σ , braucht **pro Zustand** und pro Element in Σ , eine Übergangsfunktion mit der Eingabe gleich dem Element.
- Keine ε -Übergänge existieren.

Nichtdeterministische Automaten sind gleich mächtig wie deterministische!

Mit einer <u>Teilmengenkonstruktion</u> kann ein NEA in einen DEA umgewandelt werden:

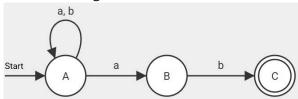


Tabelle aufstellen:

	Eingabe	
Zustand	a	b
Α	A, B	Α
В	Ø	С
С	Ø	Ø

 Tabelle umschreiben + nicht zu erreichende Zustände weglassen:

	Eingabe	
Zustand	a	b
Α	AB	Α
AB	AB	AC
AC	AB	Α

Kontextfreie Grammatiken

Die Kontextfreie Grammatik kann nicht reguläre Sprachen beschreiben.

Definition (Kontextfreie Grammatik)

Eine kontextfreie Grammatik G (KFG) ist ein 4-Tupel (N, Σ, P, A) mit

- N ist das Alphabet der **Nichtterminale** (Variablen).
- ullet Σ ist das Alphabet der **Terminale**.
- lacksquare P ist eine endliche Menge von **Produktionen** (Regeln). Jede Produktion hat die Form

$$X \to \beta$$

mit **Kopf** $X \in N$ und **Rumpf** $\beta \in (N \cup \Sigma)^*$.

• A ist das **Startsymbol**, wobei $A \in N$.

Ein KGF für die Sprache $\{0^n1^n \mid n \in \mathbb{N}\}:$

$$G_1 = (\{A\}, \{0, 1\}, P, A)$$

$$P = \{A \rightarrow 0A1, A \rightarrow \varepsilon\}$$

Linksseitige Ableitung: Das Nichtterminale Zeichen, ganz links, wird immer zuerst abgeleitet.

Rechtsseitige Ableitung: Das Nichtterminale

Zeichen, ganz rechts, wird immer zuerst abgeleitet.

Beispiel (Ableitungsbaum für das Wort (())() in G_2)

Mehrdeutige KGF: Für ein Wort existieren mehrere Parsebäume.

KFG für eine reguläre Sprache (DEA)

- 1. Zustand $q_i := \text{Nichtterminale Variable } Q_i$.
- 2. Transition $\delta(q_i, a) = q_j := \text{Produktion } Q_i \rightarrow aQ_i$.
- 3. akzeptierenden Zustand $q_i \in F := \text{Produktion}$ $Q_i \to \varepsilon$.
- 4. Das Nichtterminale Q_0 wird zum Startsymbol.

Seite 2 von 4 Nino Frei

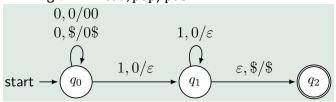
Kellerautomaten

Definition (deterministischer Kellerautomat)

Ein deterministischer Kellerautomat (KA) M ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, \$, F)$, wobei

- Q ist eine endliche Menge von Zuständen.
- ullet Σ ist das Alphabet der Eingabe.
- lue Γ ist das Alphabet des Kellers.
- $\delta: Q \times (\Sigma \cup \varepsilon) \times \Gamma \to Q \times \Gamma^*$ ist eine (partielle) Übergangsfunktion.
- $q_0 \in Q$ ist der Startzustand.
- $\blacksquare\ \$ \in \varGamma$ ist ein ausgezeichnetes Symbol vom Alphabet des Kellers.
- ullet $F\subseteq Q$ ist die Menge der akzeptierenden Zustände.

Berechnungsschritt: $\delta(q_i, \text{read}, \text{pop}) = (q_j, \text{push}) \rightarrow \text{im Diagramm: read, pop/push.}$



Mit dem $read = \varepsilon \& pop = c \in \Gamma$, diese Übergangsfunktion mit pop = c darf nur einmal Vorkommen pro Zustand, ansonsten ist der KA nichtdeterministisch.

Nichtdeterminismus

NKA sind nicht gleich mächtig wie DKA.

Eine Sprache ist kontextfrei, genau dann, wenn es einen NKA gibt, der die Sprache erkennt.

Turingmaschine

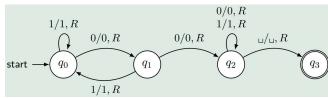
Definition (Turing-Maschine)

Eine (deterministische) Turing-Maschine (TM) ist ein 7-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \mathbf{u}, F)$$

mit einer bzw einem:

- endlichen Menge von **Zuständen** $Q = \{q_0, q_1, \dots, q_n\}$ $(n \in \mathbb{N})$,
- Eingabealphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$ $(m \in \mathbb{N})$,
- Übergangsfunktion $\delta \colon Q \times \Gamma \to Q \times \Gamma \times D$, $D = \{L, R\}$,
- Startzustand $q_0 \in Q$,
- Menge von akzeptierenden Zuständen $F \subseteq Q$,
- \blacksquare Bandalphabet \varGamma (endliche Menge von Symbolen) und $\varSigma \subset \varGamma$ und
- Leerzeichen \sqcup , mit $\sqcup \in \Gamma$ und $\sqcup \notin \Sigma$.



Die Berechnung für das Wort 01001 die Beispiel-Turingmaschine sieht wie folgt aus:

$$q_001001 \vdash 0q_11001 \vdash 01q_0001 \vdash 010q_101 \\ \vdash 0100q_21 \vdash 01001q_2 \vdash 01001_q_3 \\ \rightarrow \text{Akzeptierend}$$

Eine Sprache die von einer TM akzeptiert wird, ist rekursiv aufzählbar.

Nichtdeterminismus

Nichtdeterministische Turing-Maschinen sind gleich mächtig wie DTM.

Universelle Turingmaschinen

Bei einer universellen TM werden die

Übergangsfunktionen einer TM codiert mit folgenden Vorgehen:

- 1. Zustände $q_n := 0^n \to \operatorname{Bsp.} q_2 = 00$, dabei ist $q_1 = \operatorname{Startzustand}, q_2 = \operatorname{Endzustand}$
- 2. $\Gamma = \{0, 1, \$, a, b, ..., z\}$, Bandsymbol $\Gamma_3 = \$(blank) = 000$
- 3. Richtung L = 0 & R = 00

Trennzeichen:

- Zwischen den Elementen: 1
- Zwischen den Übergangsfunktionen: 11
- Ende der Turing-Definition: 111

Berechnungsmodelle

Eine Funktion ist Turing-berechenbar, wenn es eine TM gibt, die für alle Wörter anhält.

Loop-Programm:

x0 = x2 + 0;

= x0 + 1

Variable = Variable
$$\pm$$
 Konstante
Loop $\times 1$ Do
 $\times 2 = \times 2 + 1$
End;

While-Programm:

Erweiterung des Loop-Programm.

While
$$x1 > 0$$
 Do $x1 = x1 - 1$;
Loop $x2$ Do $x0 = x0 + 1$
End

GoTo-Programm:

M1:
$$x0 = x3 + 0$$

M2: IF $x1=0$ THEN GOTO M4
M3: $x0 = x2 + 0$
M4: HALT

Zusammenfassung THIN

While & GoTo Programme sind Turing-vollständig, während das Loop-Programm primitiv-rekursiv ist.

 $\forall n \in \mathbb{N}$ und jede Konstante $k \in \mathbb{N}$ die n-stellige konstante Funktion:

$$c_k^n = \mathbb{N}^n \to \mathbb{N} \text{ mit } c_k^n(x_1, ..., x_n) = k$$

Nachfolgerfunktion: $\eta: \mathbb{N} \to \mathbb{N}$ mit $\eta(x) = x+1$

 $\forall n \in \mathbb{N}, 1 < k < n$ die n-stellige Projektion auf die k-te Komponente:

$$\pi_k^n: \mathbb{N}^n \to \mathbb{N} \text{ mit } \pi_k^n(x_1, ..., x_k, ..., x_n) = k$$

n = Anzahl der Argumente, k = Position des Arguments

Beispiel Addition:

$$Add(0, y) = y$$

$$Add(x + 1, y) = Add(x, y) + 1$$

$$Add(0, y) = \pi_1^1(y)$$

$$Add(x + 1, y) = \eta(\pi_1^3(Add(x, y), x, y))$$

Entscheidbarkeit

Negative-Sprache \bar{A} : Output ist zur Sprache A «umgekehrt».

Entscheidbarkeit: TM haltet immer an \rightarrow Output: JA / NEIN. A entscheidbar $:= \bar{A}$ entscheidbar

Semi-Entscheidbarkeit: TM haltet nur an, wenn Output: $\overline{\text{JA}}$, ansonsten läuft TM unendlich weiter. Wenn $A \& \overline{A}$ beide semi-entscheidbar sind, dann ist A entscheidbar.

Reduktion: $A \le B \iff A$ ist reduzierbar auf B. $P_1(x) \coloneqq \operatorname{Ist} x$ eine Primzahl? & $P_2(x,y) \coloneqq \operatorname{Ist} x$ der kleinste Primfaktor von y? P_1 kann auf P_2 reduziert werden mit dem Input: $P_2(x,x)$.

Halteproblem:

- allgemeines Halteproblem H: Hält die TM T, wenn man sie auf x ansetzt?
- leeres Halteproblem H_0 : Hält die TM T, wenn man sie auf das leere Band ansetzt?
- spez. Halteproblem H_S : Hält die TM T, wenn man sie auf ihren eigenen Code ansetzt?

Beweisidee: Wir zeigen H_S ist nicht entscheidbar und reduzieren dann das Halteproblem: $H_S \leq H \leq H_0$.

 H_S ist nicht entscheidbar (durch Widerspruch), H_S ist auf H reduzierbar und H auf H_0 . Somit ist H_0 auch nicht entscheidbar.

Komplexitätstheorie

Zeitkomplexität	Platzkomplexität	Beschreibungskomplexität	
Laufzeit des besten Programms,	Speicherbedarf des besten	Länge des kürzesten Programms.	
welche das Problem löst.	Programms		

O-Notation (Landau Symbole):

- $f \in \mathcal{O}(g)$: f wächst nicht asymptotisch schneller als g.
- $f \in \Omega(g)$: f wächst asymptotisch mindestens so schnell wie g.
- $f \in \Theta(g)$: f und g wachsen asymptotisch gleich schnell.

 $\begin{array}{l} \textbf{Laufzeit-Reihenfolge} \colon \mathcal{O}(1) < \mathcal{O}(\log \log n) < \\ \mathcal{O}\left(\sqrt{\log n}\right) < \mathcal{O}\left(\log \sqrt{n}\right) < \mathcal{O}(\log n) < \mathcal{O}\left(\sqrt{n}\right) < \\ \mathcal{O}(n) < \mathcal{O}(n*\log n) < \mathcal{O}(n^2) < \mathcal{O}(n^3) < \mathcal{O}(n^c) < \\ \mathcal{O}(2^n) < \mathcal{O}(n!) < \mathcal{O}(n^n). \end{array}$

Klasse P: Alle Probleme, die von einer DTM in Polynomzeit gelöst werden.

Klasse NP: Alle Probleme die von einer NTM in Polynomzeit gelöst werden. (NP = nichtdeterministisch polynomiell)

Polynomzeit-Verifizierer:

Kann ein Zeuge (= mögliche Lösung für ein Problem) in polynomielle Zeit verifiziert werden, nennt man dies ein Polynomzeit-Verifizierer.

[P] :=Lösung finden in Polynomzeit [NP] :=Lösung verifizieren in Polynomzeit Ob nun [P] = [NP] ist noch nicht geklärt.

Eine Sprache L ist **NP-schwer**, falls alle anderen Sprachen $L' \in NP$ auf L reduziert werden kann.

Eine Sprache L ist **NP-vollständig**, falls $L \in NP \& L \in NP_{Schw}$

Wenn es uns gelingt eine NP-vollständiges Problem in P liegt, dann gilt: P = NP. Beispiel NP-vollständiges Problem: **SAT** (ist das Problem zu entscheiden, ob eine gegebene Formel in KNF erfüllbar ist.)