

Alphabet, Wörter & Sprachen

Begriff	Erklärung / Beispiel	Begriff	Erklärung / Beispiel
Alphabet Σ	<i>endliche, nichtleere Menge an Symbolen</i>	echtes Suffix v_e	v_e ist ein echtes Suffix von w , wenn $v_e \neq w$ gilt.
Wort w	ist eine <i>endliche</i> Folge von <i>Symbolen</i>	Σ^k	<i>Menge</i> aller <i>Wörter</i> mit der Länge k .
leeres Wort ε	enthält <i>keine</i> <i>Symbole</i> , gehört zu <u>jedem</u> <i>Alphabet</i>	Σ^*	<i>Menge</i> aller <i>Wörter</i> über dem <i>Alphabet</i> Σ . (Kleenesche Hülle)
$ w $	Länge eines <i>Wortes</i> $ abc = 3$	Σ^+	<i>Menge</i> aller Wörter über dem <i>Alphabet</i> Σ <i>minus</i> dem leeren Wort. $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$
$ w _x$	Häufigkeit des <i>Symbols</i> x in dem <i>Wort</i> w . $ 100110 _1 = 3$	$x \circ y$	Konkatenation von zwei beliebigen <i>Wörtern</i>
w^R	Spiegelung des <i>Wortes</i> w $(abc)^R = cba$	Wortpotenz x^n	x ist ein beliebiges <i>Wort</i> über Σ ($x = ab$) $x^3 = ababab$
Teilwort (Infix) v	v ist ein <i>Teilwort</i> von w , wenn $w = xvy$ für beliebige <i>Wörter</i> x, y gilt.	Sprache L	ist eine Teilmenge von Σ^* $L \subseteq \Sigma^*$
echtes Teilwort v_e	v_e ist ein echtes <i>Teilwort</i> von w , wenn $v_e \neq w$ gilt.	leere Sprache \emptyset	ist die <i>Sprache</i> über jedem <i>Alphabet</i> .
Präfix v	v ist ein <i>Präfix</i> von w , wenn $w = vy$ für beliebige <i>Wörter</i> y gilt.	Konkatenation von Sprachen	$AB = \{uv \mid u \in A \wedge v \in B\}$ Nennt man eine Konkatenation der zwei Sprachen A & B .
echtes Präfix v_e	v_e ist ein echtes <i>Präfix</i> von w , wenn $v_e \neq w$ gilt.	A^*	die Kleenesche Hülle A^* der Sprach A ist durch $\{\varepsilon\} \cup A \cup AA \cup AAA \cup \dots$ definiert.
Suffix v	v ist ein <i>Suffix</i> von w , wenn $w = xv$ für beliebige <i>Wörter</i> x gilt.	Entscheidungsproblem	INPUT: <i>Sprache</i> L , <i>Wort</i> 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 | 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**
- $L_1 L_2$ ist auch **regulär**
- L_1^* ist auch **regulär**
- $\bar{L} = \Sigma^* \setminus L$ ist auch **regulär**
- $L_1 \cap L_2$ ist auch **regulär**
- $L_1 \setminus L_2$ ist auch **regulär**

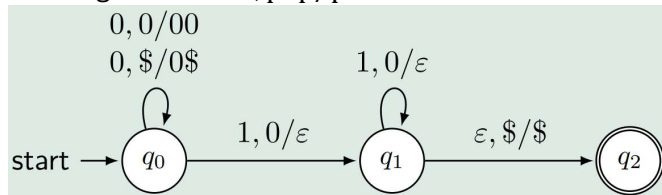
Kellerautomaten

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

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



Mit dem $\text{read} = \varepsilon$ & $\text{pop} = c \in \Gamma$, diese Übergangsfunktion mit $\text{pop} = c$ 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

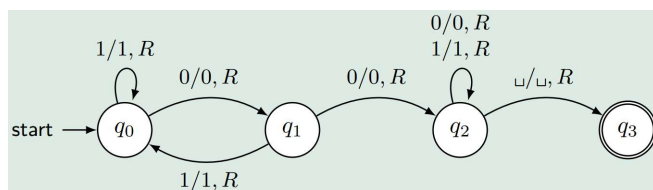
Definition (Turing-Maschine)

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

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

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 $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times D$, $D = \{L, R\}$,
- Startzustand $q_0 \in Q$,
- Menge von akzeptierenden Zuständen $F \subseteq Q$,
- Bandalphabet Γ (endliche Menge von Symbolen) und $\Sigma \subset \Gamma$ und
- Leerzeichen \sqcup , mit $\sqcup \in \Gamma$ und $\sqcup \notin \Sigma$.



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

$q_0 01001 \vdash 0q_1 1001 \vdash 01q_0 001 \vdash 010q_1 01$
 $\vdash 0100q_2 1 \vdash 01001q_2 \vdash 01001_q_3$
 \rightarrow 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 $q_n := 0^n \rightarrow$ Bsp. $q_2 = 00$, dabei ist $q_1 =$ Startzustand, $q_2 =$ Endzustand
2. $\Gamma = \{0, 1, \$, a, b, \dots, z\}$, 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:

Variable = Variable \pm Konstante

```

Loop x1 Do
  x2 = x2 + 1
End;
x0 = x2 + 0;
x0 = x0 + 1
  
```

While-Programm:

Erweiterung des Loop-Programm.

```

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

GoTo-Programm:

```

M1: x0 = x3 + 0
M2: IF x1=0 THEN GOTO M4
M3: x0 = x2 + 0
M4: HALT
  
```

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:

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

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

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

$$\pi_k^n : \mathbb{N}^n \rightarrow \mathbb{N} \text{ mit } \pi_k^n(x_1, \dots, x_k, \dots, x_n) = x_k$$

n = Anzahl der Argumente, k = Position des Arguments

Beispiel Addition:

$$\begin{aligned} \text{Add}(0, y) &= y \\ \text{Add}(x + 1, y) &= \text{Add}(x, y) + 1 \\ \text{Add}(0, y) &= \pi_1^1(y) \\ \text{Add}(x + 1, y) &= \eta(\pi_1^3(\text{Add}(x, y), x, y)) \end{aligned}$$

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: **JA**, ansonsten läuft TM unendlich weiter. Wenn A & \bar{A} beide **semi-entscheidbar** sind, dann ist A **entscheidbar**.

Reduktion: $A \leq B \Leftrightarrow A$ ist reduzierbar auf B .
 $P_1(x) :=$ Ist x eine Primzahl? & $P_2(x, y) :=$ 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, welche das Problem löst.	Speicherbedarf des besten Programms	Länge des kürzesten 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.

Laufzeit-Reihenfolge: $\mathcal{O}(1) < \mathcal{O}(\log \log n) < \mathcal{O}(\sqrt{\log n}) < \mathcal{O}(\log \sqrt{n}) < \mathcal{O}(\log n) < \mathcal{O}(\sqrt{n}) < \mathcal{O}(n) < \mathcal{O}(n * \log n) < \mathcal{O}(n^2) < \mathcal{O}(n^3) < \mathcal{O}(n^c) < \mathcal{O}(2^n) < \mathcal{O}(n!) < \mathcal{O}(n^n)$.

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.)