

# Theoretische Informatik - ZHAW 2024

## Alphabete, Wörter und Sprachen

<b>Alphabet</b>	endliche, nichtleere Menge von Symbolen (Bsp.: $\Sigma = \{a, b, c\}$ )
<b>Wort (String)</b>	endliche Folge von Symbolen über einem Alphabets (Bsp.: 101 über $\Sigma = \{1, 0\}$ ) <b>leeres Wort (<math>\epsilon</math>)</b> enthält keine Symbole $\rightarrow$ ist Wort über jedem Alphabet und hat <b>Länge 0!</b> $\rightarrow$ eindeutige Folgen brauchen <b>kein Komma</b> , w = aaa schon! (aa, a oder a, a, a oder a, aa)
<b>Länge (Wort)</b>	Anzahl Symbole der Folge (Bsp.:  a b c  = 5, Leerzeichen sind Symbole!)
<b>Häufigkeit</b>	Anzahl, welche ein Symbol in einem Wort vorkommt (Bsp.:  1231123 _1 = 3)
<b>Spiegelwort</b>	Wort mit <b>umgekehrter</b> Symbolreihenfolge (Bsp.: $(abc)^R = cba$ , $w = w^R \rightarrow$ <b>Palindrom</b> )
<b>Infix (Teilwort)</b>	$v$ ist Infix von $w$ , wenn $w = xv$ ( $x, y$ beliebige Wörter) $\rightarrow$ wenn $v \neq w \rightarrow$ <b>echtes Teilwort</b> ( $x$ oder $y$ nicht leer)
<b>Präfix (~vor)</b>	$v$ ist Präfix von $w$ , wenn $w = vy$ ( $y$ beliebiges Wort) $\rightarrow$ wenn $v \neq w \rightarrow$ <b>echtes Präfix</b> ( $y$ nicht leer)
<b>Suffix (~nach)</b>	$v$ ist Suffix von $w$ , wenn $w = xv$ ( $x$ beliebiges Wort) $\rightarrow$ wenn $v \neq w \rightarrow$ <b>echtes Suffix</b> ( $x$ nicht leer)
<b>Wörter mit <math> w  = k</math></b>	Die Menge aller Wörter der Länge $k$ wird mit $\Sigma^k$ bezeichnet ( $\Sigma^0$ ist <b>immer</b> $\{\epsilon\}$ ) Bsp.: $\Sigma = \{0, 1\}$ ist $\Sigma^3 = \{000, 001, 010, 011, 100, \dots\}$
<b>Menge aller Wörter</b>	Kleenesche Hülle = $\Sigma^*$ , $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ $\rightarrow$ positive Hülle (alle nichtleere Wörter) Bsp.: $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\} \rightarrow$ Binärwörter
<b>Konkatenation</b>	<b>Verkettung</b> von beliebigen Wörtern $x \circ y = xy$ (assoziativ aber <b>nicht</b> kommutativ!) Bsp.: $x = 101$ , $y = 00 \rightarrow xy = 10100$ Verkettung von Sprachen auch möglich! $AB = \{uv \mid u \in A \text{ und } v \in B\}$ Sprache $AB \rightarrow$ Wörter mit Präfix aus $A$ und Suffix aus $B$ .
<b>Wortpotenzen</b>	Wiederholung von Symbolen ( $x^{n+1} := x^n \circ x = x^{nx}$ ) Bsp.: $x^0 := \epsilon$ , $a^3 = aaa$
<b>Sprache</b>	<b>Teilmenge</b> von Wörtern über einem Alphabet (Bsp.: Java ist eine Sprache über ASCII) <ul style="list-style-type: none"> <li>- Wenn <math>\Sigma_1 \subseteq \Sigma_2</math> gilt und <math>L</math> eine Sprache über <math>\Sigma_1</math> ist, dann ist <math>L</math> auch eine Sprache über <math>\Sigma_2</math></li> <li>- <math>\Sigma^*</math> ist eine Sprache über jedem Alphabet <math>\Sigma</math></li> <li>- <math>\{\}</math> = <math>\emptyset</math> ist die leere Sprache für jedes Alphabet</li> <li>- <math>\{\epsilon\}</math> ist die Sprache, die aus dem leeren Wort <math>\epsilon</math> besteht für jedes Alphabet <math>\Sigma</math> Anmerkung: <math>\emptyset \neq \{\epsilon\}</math></li> <li>- Sprachen können unendlich viele Wörter haben (Wörter sind aber endlich lang)</li> <li>- Alphabet muss endlich sein</li> </ul>
<b>Entscheidungsproblem</b>	Input: Sprache $L$ und Wort $x$ Output: JA, wenn $x \in L$ und NEIN, wenn $x \notin L$

<ul style="list-style-type: none"> <li>■ <math>L(R S) = L(S R)</math></li> <li>■ <math>L(R(ST)) = L((RS)T)</math></li> <li>■ <math>L(R (S T)) = L((R S) T)</math></li> <li>■ <math>L(R(S T)) = L(RS RT)</math></li> <li>■ <math>L((R^*)^*) = L(R^*)</math></li> <li>■ <math>L(R R) = L(R)</math></li> </ul>	<ul style="list-style-type: none"> <li>■ <math>L(\emptyset) = \emptyset</math></li> <li>■ <math>L(\epsilon) = \{\epsilon\}</math></li> <li>■ <math>L(a) = \{a\}</math> für <math>a \in \Sigma</math></li> <li>■ <math>L(R^*) = L(R)^*</math></li> <li>■ <math>L(R S) = L(R) \cup L(S)</math></li> <li>■ <math>L(RS) = L(R)L(S)</math></li> </ul>
---	--

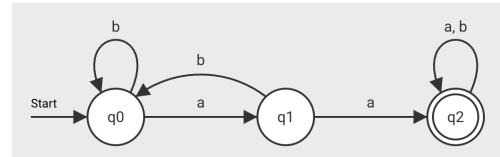
## Reguläre Ausdrücke (RA)

Operatoren:  $\rightarrow$  Sprache über dem Alphabet  $\{\emptyset, \epsilon, *, (,), |\}$

-  $A|B \rightarrow A$  oder  $B \rightarrow " * "$  vor Konkatenation vor " $|$ "

-  $A^* \rightarrow$  beliebig viele  $A$

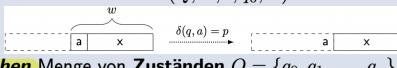
$\rightarrow$  Erweiterungen:  $R+ = R(R^*)$ ,  $R? = (R|\epsilon)$ ,  $[R_1, \dots, R_k] = R_1 | R_2 | \dots | R_k$



## Endliche Automaten (DEA und NEA)

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**  $\delta: Q \times \Sigma \rightarrow Q$   
 ■ **Startzustand**  $q_0 \in Q$   
 ■ **Menge der akzeptierenden Zustände**  $F \subseteq Q$

$$L(M) = \{xaay \mid x, y \in \{a, b, c\}^*\}$$

$$= \{w \in \{a, b, c\}^* \mid w \text{ enthält das Teilwort } aa\}$$

Bsp.-Übergang:  $\delta(q_0, a) = q_1$

$\Rightarrow$  wenn auf  $q_0$  und  $a$  lesen (im Input-Wort) dann geh nach  $q_1$

$\rightarrow$  "Abfall"-Zustände dürfen weggelassen werden

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{\delta(q_0, a) = q_1, \delta(q_0, b) = q_0, \delta(q_1, a) = q_2, \text{etc.}\}, \{q_2\})$$

- Das Input-Wort wird von **rechts nach links** gelesen, nach jedem Übergang zum nächsten Zeichen gesprungen

**Berechnung**  $\rightarrow$

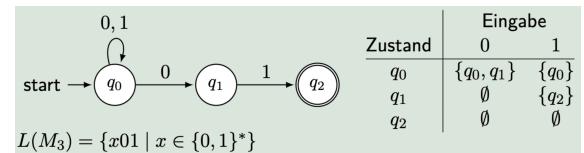
$$(q_a, w_1 w_2 \dots w_n) \vdash_M (q_b, w_2 \dots w_n) \vdash_M \dots \vdash_M (q_e, w_j \dots w_n)$$

wird abgekürzt als  $(q_a, w_1 w_2 \dots w_n) \vdash_M^* (q_e, w_j \dots w_n)$  geschrieben.

Berechnung ist verwerfend ( $q_e \notin F$ ) oder akzeptierend ( $q_e \in F$ )

## Nichtdeterminismus

- keine, eine oder mehrere Folgekonfigurationen
- solange **mindestens eine akzeptierte Berechnung** existiert!
- Bei Berechnung von NEA: jede mögliche Route aufschreiben
- Für jeden nicht eindeutigen Übergang wird eine neue Route erstellt



**Satz: NEAs und DEAs sind gleich mächtig!** -> NEA kann in DEA umgewandelt werden:



- Von Start-Zustand aus auflisten.
- Bei Nichtdeterminismus: Neuer "Kombi-Zustand" (AB)
- Bis keine neuen Zustände entstehen
- Alle mit altem akzeptierenden Zustand: neuer Endzustand

->  **$\epsilon$ -Übergang**: es wird nichts gelesen (Bsp.: Zahlen mit und OHNE Vorzeichen) Beim umformen berücksichtigen!

-> Vereinigung, Konkationation, Kleenesche Hülle, Komplement, Schnitt, Mengendifferenz von RAs ist auch eine RA!

- Jeder Zustand ist eine eigene **Zustandsklasse** und hat gewisse Eigenschaften (z.B. Anz. Nullen in w ist immer 3)

**Satz: ein EA und x, y zwei beliebige Wörter aus  $\Sigma^*$ , ->  $x, y \in [p]$ . => alle Wörter  $z \in \Sigma^*$ :  $xz \in L(M) \Leftrightarrow yz \in L(M)$**

## Untere Schranke für die Grösse endlicher Automaten:

Mit der vorherigen Technik können wir zeigen, dass ein Automat für eine bestimmte Sprache  $L$  mindestens  $n$  Zustände benötigt:

- 1 Überlegen, wie die  $n$  Zustandsklassen für den zu  $L$  gehörigen EA aussehen könnten.
- 2  $n$  unterschiedliche Wörter finden, von denen keine zwei zur gleichen Zustandsklasse gehören.
- 3 Für je zwei dieser Wörter mit Hilfe von (\*) und einem Widerspruchsbeweis zeigen, dass der EA zwischen diesen beiden Wörtern unterscheiden muss.
- 4 Daraus folgt direkt, dass jeder EA für  $L$  mindestens zwischen diesen  $n$  Zustandsklassen unterscheiden muss (und somit mind.  $n$  Zustände haben muss).

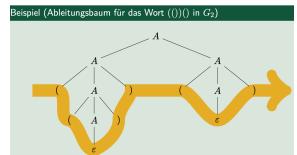
### Beispiel

Zeige: Jeder EA für die Sprache

$$L(M_9) = \{w \in \{0,1\}^* \mid |w|_0 \bmod 3 = 1\}$$

### Beweis:

- 1 Jeder EA für  $L(M_9)$  muss die Anzahl der gelesenen Nullen modulo 3 zählen und unterscheiden können.
- 2 Zum Beispiel:  $x_1 = \epsilon, x_2 = 0, x_3 = 00$
- 3 Widerspruch für alle Paare von Wörtern aufzeigen:  
Für  $x_1$  und  $x_2$ :  $z_{12} = \epsilon \Rightarrow x_1 z_{12} = \epsilon \notin L, x_2 z_{12} = 0 \in L$   
Für  $x_1$  und  $x_3$ :  $z_{13} = 0 \Rightarrow x_1 z_{13} = 0 \in L, x_3 z_{13} = 000 \notin L$   
Für  $x_2$  und  $x_3$ :  $z_{23} = \epsilon \Rightarrow x_2 z_{23} = 0 \in L, x_3 z_{23} = 00 \notin L$
- 4 Jeder EA für  $L(M_9)$  muss zwischen mindestens drei Zuständen unterscheiden. Der EA hat mind. 3 Zustände.  $\square$



auch: Parsebaum ↗

## Kontextfreie Grammatiken (KFG)

### Definition (Kontextfreie Grammatik)

Eine **kontextfreie Grammatik**  $G$  (KFG) ist ein 4-Tupel  $(N, \Sigma, P, A)$  mit

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

$$X \rightarrow \beta$$

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

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

Wenn  $L = L(G)$  -> kontextfreie Sprache

- **Mehrdeutige kontextfreie Grammatik:** mind. ein Wort hat mehrere Parsebäume
- **Achtung** beim Entwurf, dass nicht auch Wörter gebildet werden können die eigentlich gar nicht dazu gehören!

Für die Sprache  $\{0^n 1^n \mid n \in \mathbb{N}\}$ :

$$G = (\{A\}, \{0, 1\}, \{A \rightarrow 0A1, A \rightarrow \epsilon\}, A)$$

-> A kann mit 0A1 oder nichts ersetzt werden:

$$A \rightarrow 0A1 \rightarrow 00A11 \rightarrow 000A111 \rightarrow 000111 (0^3 1^3)$$

Ableitung: Folge von Ableitungsschritten -> Satzform a in Wort w abzuleiten.

$A \Rightarrow^* w = A$  erzeugt/generiert w ( $\Rightarrow^*$ : endliche Anzahl an Ableitungen)

-links-/rechtsseitige Ableitung: bei jedem Schritt: Nichtterminal ganz links/rechts ableiten

-erzeugte Sprache: ALLE Wörter die aus A ableitbar sind.

## Kellerautomaten (DKA und NKA)

- ist ein endlicher Automat mit zusätzlichem unbegrenzten **Stack** (first-in-first-out!) Beispiel: -> \$ heisst, dass Stack leer ist

### 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.
- $\Sigma$  ist das Alphabet der Eingabe.
- $\Gamma$  ist das Alphabet des Kellers.
- $\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$  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.

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, \$\}, \delta^*, q_0, \$, \{q_2\})$$

$$\delta^* = \{\delta(q_0, 0, 0) = (q_0, 00), \delta(q_0, 0, \$) = (q_0, 0\$), \text{etc.}\}$$

Berechnungsschritt:  $\delta(q, b, c) = (p, w)$

1. Ist im Zustand  $q$

2. liest das Symbol  $b$  (falls  $b = \epsilon$ : nichts lesen)

3. entfernt oberstes Stack-Symbol  $c$

4. schreibt das Wort  $w$  auf den Stack (von hinten nach vorne!)

5. wechselt in den Zustand  $p$

Wenn deterministisch: wenn  $(q, b, x) : (q, \epsilon, x)$  darf nicht auch noch dort existieren!

Nichtdeterminismus funktioniert wie bei den Automaten

- Konfiguration  $(q, w, y) = (\text{Zustand}, \text{verbleibende Eingabe}, \text{Stack-Inhalt})$  [**linkes Symbol = zuoberst!**])
- Berechnung von  $M$  auf  $w$ : Folge von Konfigurationen: Startkonfig zu einer Endkonfig (akzeptierend?) (wie oben^)
- Sprache ist **kontextfrei**, wenn es einen **NKA** gibt, der die Sprach erkennt! (NICHT zwingend umgekehrt)

## Turingmaschinen (TM)

- endlicher Automat mit Lese/Schreibkopf + unendliches Band (schreiben, lesen und hin/her bewegen)

### 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**  $Q = \{q_0, q_1, \dots, q_n\}$  ( $n \in \mathbb{N}$ ),  $q_0$  Startzustand
- **Eingabealphabet**  $\Sigma = \{a_1, a_2, \dots, a_m\}$  ( $m \in \mathbb{N}$ ),  $\sqcup$  Leerzeichen
- **Übergangsfunktion**  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times D$ ,  $D = \{L, R\}$ ,  $L$  Schreiben nach rechts,  $R$  Schreiben nach links
- **Menge von akzeptierenden Zuständen**  $F \subseteq Q$
- **Bandalphabet**  $\Gamma$  (endliche Menge von Symbolen) und  $\Sigma \subset \Gamma$  und  $\# \notin \Sigma$

$$M = (\{q_0, q_1, q_f\}, \{0, 1\}, \delta^*, q_0, \{\sqcup, 0, 1, \#\}, \sqcup, \{q_f\})$$

Berechnungsschritt:  $\delta(q, X_i) = (p, Y, R/L)$  (R/L = Rechts/Links)

-> (Von-Zustand, Lesen) = (Zu-Zustand, Schreiben, Richtung)

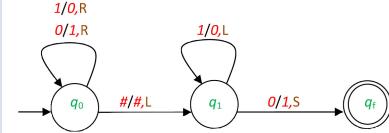
Startzustand: Lesekopf über erstem Zeichen der Eingabe

Der Übergang  $\delta(q, X_i) = (p, Y, R)$  spezifiziert einen Berechnungsschritt mit einer Bewegung nach rechts:

$$X_1 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 \dots X_{i-1} Y p X_{i+1} \dots X_n$$

Der Übergang  $\delta(q, X_i) = (p, Y, L)$  spezifiziert einen Berechnungsschritt mit einer Bewegung nach links:

$$X_1 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$



Akzeptiert=TM endet in akzeptiertem Zustand -> Sprache von TM = "rekursiv aufzählbar"

Jede Erweiterung einer TM (Speicher, mehrere Bänder, Köpfe, einseitiges Band, etc.) ist gleich mächtig, wie die normale TM.

Universelle TM (Gödelnummer - Binäre Codierung für TM) -> Bsp.: 010010001010011001 ... 010111 1100

### 1. Schritt: Zustände von TM:

$q_1$  = Startzustand  
 $q_2$  = akzeptierender Zustand  
 $q_3, q_4, \dots$  = Rest Zustände  
Codierung:  $q_a = a$  Nullen  
Bsp.:  $q_2 = 00$

### 2. Schritt: Bandsymbole

$X_1$  = Symbol 0  
 $X_2$  = Symbol 1  
 $X_3$  = Symbol  $\sqcup$  (blank)  
 $X_4, X_5, \dots$  = Rest  
Codierung:  $X_a = a$  Nullen

### 3. Schritt: Richtung

$D_1$  = Symbol 0  
 $D_2$  = Symbol 1  
Codierung:  
 $D_a = a$  Nullen

### 4 + 5. Schritt: Zusammensetzen

- Zwischen alle Elemente eine 1 (Trennzeichen)  
- Zwischen alle Übergänge zwei 1  
- Nach allen Übergängen drei 1 danach Input  
(Gödelnummer kann auch dezimal codiert sein:  
-> in binär umwandeln und führende 1 entfernen)

## Berechnungsmodelle

Turing-These: was algorithmisch gelöst werden kann, ist auch von einer Turing-Maschine lösbar.

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

$$T(w) = \begin{cases} u & \text{falls } T \text{ auf } w \in \Sigma^* \text{ angesetzt, nach endlich vielen Schritten mit } u \text{ auf dem Band anhält} \\ \uparrow & \text{falls } T \text{ bei Input } w \in \Sigma^* \text{ nicht anhält.} \end{cases}$$

### Primitiv Rekursiv

$\forall n \in \mathbb{N}$  und jede Konstante  $k \in \mathbb{N}$  die n-stellige konstante Funktion:  
 $c_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$  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:  
 $\pi_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$  mit  $\pi_k^n(x_1, \dots, x_k, \dots, x_n) = k$   
n = Anzahl der Argumente, k = Position des Arguments

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

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

### LOOP (primitiv rekursiv)

Zuweisungen:  
Variabel = Variabel  $\pm$  Konstante  
Sequenzen verbinden: P ; Q  
Schleifen: Loop x Do P End

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

### WHILE (Turing)

Erweiterung von LOOP:  
While Bedingung Do  
While x1 > 0 Do  
x1 = x1 - 1;  
Loop x2 Do  
x0 = x0 + 1  
End  
End

Zuweisungen: wie LOOP  
(Un)bedingte Sprunganweisung:  
Goto Mr / If xi = c Then Goto Mr  
Stoppanweisung: Halt

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

### GOTO (Turing vollständig)

## Entscheidbarkeit

- Entscheidbar: Es existiert ein Algorithmus, welcher für jede Eingabe eine Antwort liefern kann (Ja/Nein)

- Semi-entscheidbar: Es existiert ein Algo., welcher für jede Eingabe ( $\in A$ ) "Ja" zurück gibt. Wenn Eingabe  $\notin A$ , terminiert der Algo nie.

- A als auch  $\bar{A}$  sind semi entscheidbar  $\Leftrightarrow A$  ist entscheidbar

$$(\bar{A} = \Sigma^* \setminus A = \{\omega \in \Sigma^* \mid \omega \notin A\})$$

- Sind A, B (semi-) entscheidbare Sprachen, dann sind auch  $A \cup B$  und  $A \cap B$  (semi-) entscheidbare Sprachen.

### Entscheidbarkeit mit Turingmaschinen

$A \subset \Sigma^*$  heisst entscheidbar, wenn TM  $T$  existiert, sodass:

- Bandinhalt  $x \in A$   $T$  hält mit Bandinhalt «1» (Ja) an
- Bandinhalt  $x \in \Sigma^* \setminus A$   $T$  hält mit Bandinhalt «0» (Nein) an

Äquivalente Aussagen:

- $A \subset \Sigma^*$  ist entscheidbar
- Es existiert eine TM, die das Entscheidungsproblem  $T(\Sigma, A)$  löst
- Es existiert ein WHILE-Programm, dass bei einem zu  $A$  gehörenden Wort stets terminiert  $\rightarrow$  Entscheidungsverfahren für  $A$

### Semi-Entscheidbarkeit mit Turingmaschinen

$A \subset \Sigma^*$  heisst semi-entscheidbar, wenn TM  $T$  existiert, sodass:

- Bandinhalt  $x \in A$   $T$  hält mit Bandinhalt «1» (Ja) an
- Bandinhalt  $x \in \Sigma^* \setminus A$   $T$  hält nie an

Äquivalente Aussagen:

- $A \subset \Sigma^*$  ist semi-entscheidbar
- $A \subset \Sigma^*$  ist rekursiv aufzählbar
- Es gibt eine TM, die zum Entscheidungsproblem  $T(\Sigma, A)$  nur die positiven («Ja») Antworten liefert und sonst gar keine Antwort
- Es gibt ein WHILE-Programm, dass bei einem zu  $A$  gehörenden Wort stets terminiert und bei Eingabe von Wörtern die nicht zu  $A$  gehören nicht terminiert

## Reduktion: $A \leq B = A$ ist reduzierbar auf B

-> Es gibt eine Funktion  $F(\omega)$  so dass:  $\forall \omega \in \Sigma^* \quad \omega \in A \Leftrightarrow F(\omega) \in B$

Bsp:  $P_1$ : gegeben n. Ist n durch 3 teilbar? /  $P_2$ : gegeben n. Ist n durch 6 teilbar

Funktion:  $F(\omega) = \omega \times 2$ : -> Wenn Zahl durch drei, dann ist Zahl mal zwei, durch 6 teilbar.

Also gesuchte Funktion: wenn bei  $P_2$  mit  $F(\omega)$  Ja rauskommt, auch bei  $P_1$  mit  $\omega$

->  $P_1$  ist auf  $P_2$  reduzierbar. (Hier wäre  $P_2$  auch auf  $P_1$  reduzierbar:  $F(\omega) = \omega : 2$ )

**Satz:**  $A \leq B$  und  $B \leq C \Rightarrow A \leq C$  / ist B (semi / nicht)entscheidbar und  $A \leq B$ , dann ist A auch (semi / nicht)entscheidbar!

**Halteproblem:**  $H := \{\omega \# x \in \{0, 1, \#\}^* \mid T_\omega \text{ angesetzt auf } x \text{ hält}\}$  ( $\#$  = Trennung zwischen codierter TM und Input)

- leeres Halteproblem  $H_0$ :

→  $H_0 := \{\omega \in \{0, 1\}^* \mid T_\omega \text{ angesetzt auf das leere Band hält}\}$

- spezielles Halteproblem  $H_S$ :

→  $H_S := \{\omega \in \{0, 1\}^* \mid T_\omega \text{ angesetzt auf } \omega \text{ hält}\}$

=>  $H_0, H_S$  und  $H$  sind semi-entscheidbar.

**Beweisidee**  $H_S$  nicht entscheidbar:

Eine Maschine, welche bei Input folgendes macht: hält eine TM auf dem Input? Ja: laufe weiter, Nein: halte an

Wenn man nun aber diese Maschine mit ihrem eigenen Code füttert entsteht ein Paradox: würde sie halten, laufe weiter... aber Moment mal... sie sollte doch anhalten... okay.. dann läuft sie wohl weiter... aber Moment mal, wenn sie weiterläuft, sollte die Maschine stoppen. ähh?

Reduzieren auf  $H$ : Funktion:  $F(x) = x \# x \rightarrow H$  ist auch nicht entscheidbar!

H reduzieren auf  $H_0$ :  $F() = x$  (wir schreiben einfach was aufs Band und verhalten uns dann wie in  $H$ )

Da:  $H_S \leq H \leq H_0$ , zeigen dass  $H_0$  semi entscheidbar =  $H$  ist semi-entscheidbar ->  $H_0$  semi-entscheidbar: TM auf leerem Band laufen lassen

## Komplexitätstheorie

- **Zeitkomplexität:** Laufzeit des besten Programms, welches das Problem löst

- **Platzkomplexität:** Speicherplatzbedarf des besten Programms

- **Beschreibungskomplexität:** Länge des kürzesten Programms.

Zeitbedarf: Auf einer TM, die immer anhält: Anzahl von Konfigurationsübergängen - **schlechtester Fall!** :  $\text{Time}_M(n) = \max\{\text{Time}_M(\omega) \mid |\omega| = n\}$

## Komplexitätsmessung:

- $f \in O(g)$ : es existiert ein  $n_0$ , so dass für alle  $n \geq n_0$  gilt:  $f(n) \leq c \cdot g(n) \rightarrow f$  wächst asymptotisch **nicht schneller** als  $g$   
-> obere Schranke, falls es eine TM gibt, die das Problem in der Zeitkomplexität löst
- $f \in \Omega(g)$ : es existiert ein  $n_0$ , so dass für alle  $n \geq n_0$  gilt:  $f(n) \geq 1/d \cdot g(n) \rightarrow f$  wächst asymptotisch **mindestens so schnell** wie  $g$   
-> untere Schranke, falls alle TM, die das Problem lösen, diese untere Schranke haben
- $f \in \Theta(g)$ : es gilt  $f \in O(g)$  und  $f \in \Omega(g)$ : →  $f$  und  $g$  sind asymptotisch **gleich**

Beispiele:

- $O(n)$        $7n + 4$
- $O(n^3)$        $25n^2 + n^3 + 100n$
- $O(n^2 \cdot \log(n))$        $n^2 + n \cdot n \cdot (\log(n)) + 20n^2 + 50n \cdot 100$
- $O(2^n)$        $10^{20} + 3n^3 + 2^n + 2^{10} \cdot 2^{30}$

## Rechenregeln für O-Notation:

- konstante Vorfaktoren ignorieren:  $c \cdot f(n) \in O(f(n))$
- Für Konstanten  $c$  gilt:  $c \in O(1)$
- Bei Polynomen: höchste Potenz entscheidet:  $a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 \in O(n^k)$
- Transitivität:  $f(n) \in O(g(n))$  und  $g(n) \in O(h(n)) \rightarrow f(n) \in O(h(n))$

**Klasse P:** Alle Probleme, die von einer **deterministischen TM in Polynomzeit** gelöst werden können

**Klasse NP:** Alle Probleme die von einer **NICHT-deterministischen TM in Polynomzeit** gelöst werden können ( $\neq$  nicht polynominal!)

## Polynomzeit-Verifizierer

“Zeuge”: mögliche Lösung für ein Problem. Wenn die Lösung in polynomialer Zeit verifiziert werden kann: Polynomzeit-Verifizierer

=>  $P \triangleq$  Lösung finden in Polynomzeit,  $NP \triangleq$  Lösung verifizieren in Polynomzeit

$P = NP$  ist noch nicht geklärt

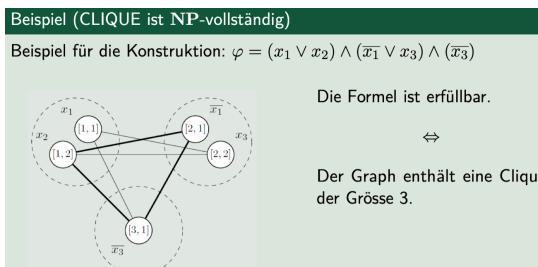
Eine Sprache  $L$  heisst NP-schwer, falls für alle Sprachen  $L' \in NP$  gilt dass  $L' \leq_p L$

→ D.h.  $L$  ist bezüglich der Lösbarkeit in polynomieller Zeit **mindestens so schwer** wie jedes einzelne Problem in  $NP$ .

Eine Sprache  $L$  heisst NP-vollständig, falls  $L \in NP$  und  $L$  ist NP-schwer.

-> D.h Wenn  $P_1$  NP-schwer und  $P_2$  in  $NP$  enthalten ist und eine polynomiale Reduktion  $P_1 \leq_p P_2$  existiert, dann ist  $P_2$  NP-vollständig.

→ Wenn für ein NP-vollständiges Problem gezeigt wird, dass es in  $P$  liegt, dann gilt:  $N = NP$



Laufzeiten-Reihenfolge:

$$O(1) < O(\log \log n) < O(\sqrt{\log n})$$

$$< O(\log \sqrt{n}) < O(\log n) < O(\sqrt{n})$$

$$< O(n) < O(n \log n) < O(n^2) < O(n^3) < O(n^c)$$

$$< O(2^n) < O(n!) < O(n^n)$$

