# Alphabete, Wörter & Sprachen

## Alphabet

- Endliche, nichtleere Menge von Symbolen

$\Rightarrow$ N ist kein Alphabet (unendliche Mächtigkeit)

Anzahl Kombinationen: $Anzahl\ Symbole$^$Länge$

## Wort

- Endliche Folge von Symbolen eines Alphabets

Leere Wort $\varepsilon$

- Keine Symbole
- **Wort über jedem Alphabet**

Länge eines Wortes

$|\varepsilon| = 0$

$\Rightarrow$ Leerzeichen sind auch Symbole!

Spiegelwort $w^R$

Palindrom: $w = w^R$

Teilwort (Infix)

v ist ein Teilwort von w:

$$\underbrace{\boxed{x}\boxed{\quad v \quad}\boxed{y}}_{w}$$

$\Rightarrow$ Echtes Teilwort: nicht identisch mit w (x oder y leer)

- $\varepsilon$, $a$, $b$, $ab$, $abb$, $bb$, $abba$, $bba$ und $ba$ sind die Teilwörter von $abba$. w
- $abba$ ist kein echtes Teilwort von $abba$ (alle anderen ja).

Präfix

$\Rightarrow$ Echtes Präfix: nicht identisch mit w (y leer)

$$\underbrace{\boxed{\quad v \quad}\boxed{y}}_{w}$$

- $\varepsilon$, $a$, $ab$, $abb$ und $abba$ sind die Präfixe von $abba$.
- $abba$ ist kein echtes Präfix von $abba$ (alle anderen ja).

Suffix

$\Rightarrow$ Echtes Suffix: nicht identisch mit w (x leer)

$$\underbrace{\boxed{x}\boxed{\quad v \quad}}_{w}$$

- $abba$, $bba$, $ba$, $a$ und $\varepsilon$ sind die Suffixe von $abba$.
- $abba$ ist kein echtes Suffix von $abba$ (alle anderen ja).

Menge aller Wörter

- Menger alle Wörter der Länge k: $\Sigma^k$

- Für $\Sigma = \{a, b, c\}$ ist $\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. $3^2 = 9$

$\Sigma^0 = \{\varepsilon\}$

# Operationen über Wörtern

Kleenesche Hülle $\Sigma^*$

- Menge aller Wörter => «alle Wörter, welche man mit dem Alphabet bilden kann»

Für $\{0,1\}$ ist $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$. Wörter aus $\{0,1\}^*$ nennt man *Binärwörter*.

Positive Hülle $\Sigma^+$

- Menge aller nichtleeren Wörter

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup$$
$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$
$$\Sigma^* = \Sigma^+ \cup \Sigma^0 = \Sigma^+ \cup \{\varepsilon\}$$

Wortpotenzen

$$x^0 := \varepsilon$$
$$x^{n+1} := x^n \circ x = x^n x$$

$a^3 = a^2a = a^1aa = a^0aaa = aaa$

$bbababababbaaaabab = b^2(ab)^4ba^4bab = b(ba)^4b^2a^3(ab)^2$

$abbabbabbabbabbabbabbabba = a(bba)^9$

## Sprache

Sprache von Wörter über einem Alphabet: $L \subseteq \Sigma^*$

- Potenziell unendlich viele Wörter

$\{\varepsilon\}$ = Sprache, die aus dem leeren Wort besteht $\emptyset \neq \{\varepsilon\}$

leere Sprache: $\{\} = \emptyset$ (für jedes Alphabet)

Konkatenation

$$AB = \{uv \mid u \in A \text{ und } v \in B\}$$

$\Rightarrow$ AB besteht aus den Wörtern, die man **(ohne Überschneidung)** in ein Präfix aus A und ein Suffix aus B aufteilen kann

AB ist eine Sprache über dem Alphabet $\Sigma \cup \Gamma$

Kleenesche Hülle $A^*$

$$\{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$$

a) Welche Wörter gehören zur Sprache $\{aa, ab, ba, bb\}^*$?

| | | |
|---|---|---|
| $\varepsilon$ ✓ | $ababa$ ✗ | $abbaba$ ✓ |
| $abaaabb$ ✗ | $abbaabba$ ✓ | $aaaaa$ ✗ |

Entscheidungsproblem

ob ein Wort zu einer Sprache gehört oder nicht

Beispiel (gerade Zahl)

Gegeben:

- Alphabet $\Sigma = \{0, 1\}$
- Sprache $L_g = \{1w0 \mid w \in \Sigma^*\}$
- Wörter $x$ aus $\Sigma^*$


Der Test, ob $x$ eine gerade Zahl grösser Null ist, ist äquivalent zu der Entscheidung, ob $x \in L_g$ gilt.

Output: JA, falls $x \in L_g$, falls $x > 0$ und gerade

Nein, falls $x \notin L_g$, falls $x \leq 0$ oder ungerade

# Reguläre Ausdrücke

## Syntax

$$\emptyset, \epsilon \in RA_\Sigma$$
$$\Sigma \subset RA_\Sigma$$
$$R \in RA_\Sigma \Rightarrow (R^*) \in RA_\Sigma$$
$$R, S \in RA_\Sigma \Rightarrow (RS) \in RA_\Sigma$$
$$R, S \in RA_\Sigma \Rightarrow (R \mid S) \in RA_\Sigma$$

Operatoren

1. *
2. Konkatenation
3. |

Der Ausdruck $ab^*|c$ wird beispielsweise als $((a(b^*))|c)$ gelesen.

$RA_\Sigma$ Bsp $\varepsilon \in L(RA_\Sigma)$

$ab \in L(RA_\Sigma)$

$abc \notin L(RA_\Sigma)$

Erweiterte Syntax

$$R^+ = R(R^*)$$
$$R? = (R \mid \epsilon)$$
$$[R_1, \ldots, R_k] = R_1 | R_2 | \ldots | R_k$$

## Semantik

| | | |
|---|---|---|
| $L(\emptyset)$ | $= \emptyset$ | Leere Sprache |
| $L(\epsilon)$ | $= \{\varepsilon\}$ | Sprache , die nur das leere Wort enthält |
| $L(a)$ | $= \{a\}$ für $a \in \Sigma$ | Beschreibt die Sprache $\{a\}$ |
| $L(R^*)$ | $= L(R)^*$ | Kombinierten Wörter von $R$ |
| $L(R\|S)$ | $= L(R) \cup L(S)$ | Wörter die von $R$ oder $S$ beschrieben werder |
| $L(RS)$ | $= L(R)L(S)$ | Verkettungen von Wörtern ($R = prefix$) |

## Reguläre Sprachen

Eine Sprache A über dem Alphabet $\Sigma$ heisst regülar, falls gilt

- $A = L(R)$ für einen regulären Ausdruck $R \in RA_\Sigma$

Beispiele

- $R_1 = a^*b$ — $L(R_1) = \{b, ab, aab, aaab, \ldots\}$
- $R_2 = (aa)^*b^*aba$ — $L(R_2) = \{aba, baba, aaaba, aababa, \ldots\}$
- $R_3 = (a|ab)^*$ — $L(R_3) = \{\varepsilon, a, ab, aa, abab, \ldots\}$

$L(R_1)$: Menge der ganzen Zahlen in Dezimaldarstellung

$R_1 = 0|\underbrace{(-?)}_{(-|\varepsilon)}\ [1,\ldots,9][0,1,\ldots 9]^*$

Menge der Binärwörter mit abwechselnd Nullen und Einsen:

$R_2 = (01)^+0?|(10)^+1?$ (mindestens Länge 2)

beliebige Länge:

$R_2 = (0?)(10)^*(1?)$

Rechenregeln

$L(R \mid S) = L(S \mid R)$

$L(R(ST)) = L((RS)T)$

$L(R \mid (S \mid T)) = L((R \mid S) \mid T)$

$L(R(S \mid T)) = L(RS \mid RT)$

$L((R^*)^*) = L(R^*)$

$L(R \mid R) = L(R)$

Beweis:

- $L(R|S) = L(R) \cup L(S) = L(S) \cup L(R) = L(S|R)$
- $L(R(ST)) = L(R)L(ST) = L(R)L(S)L(T) = L(RS)L(T) = L((RS)T)$
- $L(R|(S|T)) = L(R) \cup L(S|T) = L(R) \cup L(S) \cup L(T) = L(R|S) \cup L(T) = L((R|S)|T)$
- $L(R(S|T)) = L(R)L(S|T) = L(R)((L(S) \cup L(T))) = (L(R)L(S)) \cup (L(R)L(T)) = L(R)L(S) \cup L(R)L(T) = L(RS) \cup L(RT) = L(RS|RT)$
- $L((R^*)^*) = L(R^*)$ folgt unmittelbar aus der Def. der Kleeneschen Hülle

für alle Sprachen gilt: $(A^*)^* = A^*$

$(\Sigma^*)^* = \Sigma^*$

- $L(R|R) = L(R) \cup L(R) = L(R)$

## Endliche Automaten (EA, DEA, NEA)

- Kein Speicher

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

- Endliche Menge von Zuständen
- Eingabealphabet, Übergangsfunktion, Startzustand, Endzustand (mind. 1)

Übergangsfunktion

$\delta$(aktueller Zustand, Eingabe) = nächster Zustand

## Konfiguration

Vollständige Beschreibung der Situation, in der sich der Automat befindet

Startkonfiguration: $(q_0, w) \in \{q_0\} \times \Sigma^*$

Endkonfiguration : $Q \times \{\varepsilon\} = (q, \varepsilon)$

Berechnungsschritt

Anwendung der Übergangsfunktion auf die aktuelle Konfiguration

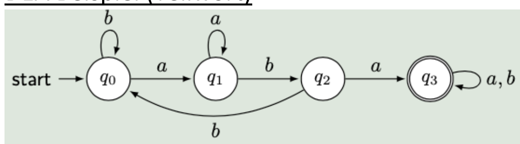
$(current\ state, noch\ zu\ lesen) \vdash_M (next\ state, suffix)$

$\Rightarrow$ Es gibt für jede Konfiguration einen Folgezustand (evtl. Abfallzustand)!

## Sprache (DEA)

- L(M) = Menge aller von M akzeptierten Wörter

DEA Beispiel (Teilwort)




Von $M_1$ akzeptierte Sprache:

$$L(M_1) = \{xabay \mid x, y \in \{a,b\}^*\}$$

## NEA

- L(M) = Menge aller von M akzeptierten Wörter

Übergangsfunktion

$$\delta: Q \times \Sigma \to \mathcal{P}(Q)$$

Berechnung

- Probiert alle möglichen Berechnungen durch

## Äquivalenz von NEA & DEA

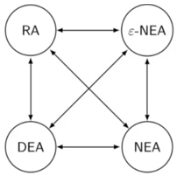
- Jeder DEA ist ein NEA und ein $\varepsilon$-NEA

**Teilmengenkonstruktion** (DEA's und NEA's sind gleich Mächtig)

1. $Q_{NEA} \to P(Q_{NEA}) = Q_{DEA}$ (Potenzmenge)
2. Verbinden mit Vereinigung aller möglichen Zielzustände
3. *Nicht erreichbare Zustände* eliminieren
4. Enthält akzeptierenden Zustand = $F_{NEA}$ → akzeptierend

| ↓ | $q$ | $\delta(q,0)$ | $\delta(q,1)$ |
|---|---|---|---|
| 0 | ~~$\emptyset$~~ | ~~$\emptyset$~~ | ~~$\emptyset$~~ |
| | $A = \{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| 1 | ~~$\{q_1\}$~~ | ~~$\emptyset$~~ | ~~$\{q_2\}$~~ |
| 4 | ~~$\{q_2\}$~~ | ~~$\emptyset$~~ | ~~$\emptyset$~~ |
| | $B = \{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_2\}$ |
| | $C = \{q_0, q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| 2 | ~~$\{q_1, q_2\}$~~ | ~~$\emptyset$~~ | ~~$\{q_2\}$~~ |
| 3 | ~~$\{q_0, q_1, q_2\}$~~ | ~~$\{q_0, q_1\}$~~ | ~~$\{q_0, q_2\}$~~ |

## Äquivalenz von DEA & RA




Klasse der regulären Sprachen:
Alle Sprachen, die von einem EA akzeptiert werden

## Abschlusseigenschaften regulärer Sprachen

Wenn L1 und L2 reguläre Sprachen sind, dann sind die folgenden Operationen auch regulär:

Vereinigung: $L_1 \cup L_2 = \{w \mid w \in L_1$ oder $w \in L_2\}$

Konkatenation: $L_1 \cdot L_2 = L_1 L_2 = \{w = w_1 w_2 \mid w_1 \in L_1$ und $w_1 \in L_2\}$

Kleenesche Hülle: $L^* = \{w = w_1 w_2 \dots w_n \mid w_i \in L$ für alle $i \in \{1, 2, \dots, n\}\}$

Komplement: $\overline{L} = \Sigma^* - L = \{w \in \Sigma^* \mid w \notin L\}$

Schnitt: $L_1 \cap L_2 = \{w \mid w \in L_1$ und $w \in L_2\}$

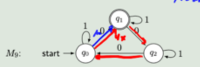
Differenz: $L_1 - L_2 = \{w \mid w \in L_1$ und $w \notin L_2\}$

## Zustandsklassen

Zustandsklassen von mod 3

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

Die Anzahl der Nullen Modulo 3 = 1
=> 1, 4, 7, 10




**Zustandsklassen von $M_9$:**

- $[q_0] = \{w \in \{0,1\}^* \mid$ (Anzahl Nullen in $w$) $\bmod 3 = 0\}$
- $[q_1] = \{w \in \{0,1\}^* \mid$ (Anzahl Nullen in $w$) $\bmod 3 = 1\}$
- $[q_2] = \{w \in \{0,1\}^* \mid$ (Anzahl Nullen in $w$) $\bmod 3 = 2\}$

## Grenzen von EA

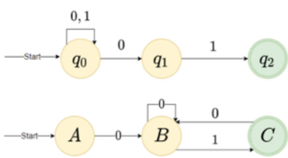
Untere Schranke für die Anzahl Zustände eines EA

**Zeige:** Jeder EA für die Sprache $L(M_9) = \{w \in \{0,1\}^* \mid |w|_0 \bmod 3 = 1\}$ hat mindestens 3 Zustände.

**Beweis:**

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

Nichtregularität




Unendlich viele Zustandsklassen -> kein EA möglich = kein RA

Beispiel: $\{0^n 1^n \mid n \in \mathbb{N}\}$ ist nicht regulär

## Kontextfreie Grammatiken

4-Tupel: $(N, \Sigma, P, A)$

N = Nichtterminale (Variablen), E = Terminale, P = endliche Menge von Produktionen (Regeln), A = Startsymbol

Beschreibung von $\{0^n 1^n \mid n \in \mathbb{N}\}$

$G_1 = (\{A\}, \{0,1\}, PA)$
$A \to 0A1$
$A \to \varepsilon$

Ableitung: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000111$

Beschreibung von (())()

$G_2 = (\{A\}, \{(,)\}, P, A)$ $\quad P = A \to (A) \mid AA \mid \varepsilon$

$A \Rightarrow AA \Rightarrow (A)A \Rightarrow (A)(A) \Rightarrow ((A))(A) \Rightarrow (())(A) \Rightarrow (())()$

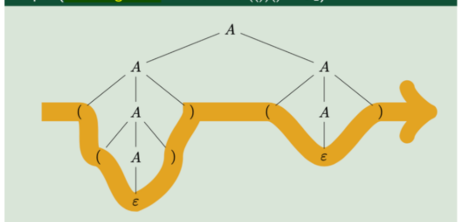
## Ableitung

Ableitbar: Folge von Ableitungsschritten -> aus einer Satzform wird das Wort w abgeleitet (=> kontextfreie Sprache)

Links/rechtsseitige Ableitung: Terminal am weitesten links oder rechts ersetzen

Ableitungsbaum

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




## Mehrdeutigkeit

⇨ Mehre Ableitungen für ein Wort möglich

Inhärent mehrdeutig

- Kf-Sprache, für die alle Grammatiken mehrdeutig sind

z.B. wenn die Wörter unterschiedliche Grammatiken haben und somit auch mehrere Ableitungsbäume

## DEA -> KFG

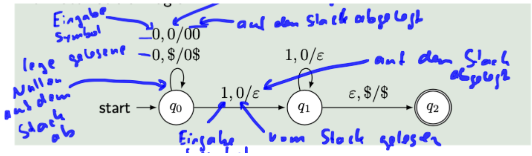
1. Für jeden Zustand $q_i$ gibt es ein Nichtterminal $Q_i$.
2. Für jede Transition $\delta(q_i, a) = q_j$ erstellen wir die Produktion $Q_i \to aQ_j$.
3. Für jeden akzeptierenden Zustand $q_i \in F$ erstellen wir die Produktion $Q_i \to \varepsilon$.
4. Das Nichtterminal $Q_0$ wird zum Startsymbol.

Technik: Teilwörter -> uRv; Rekursion -> Nichtterminal für Ausdruck

## Kellerautomaten

- EA mit zusätzlichem (unbegrenztem) Stack

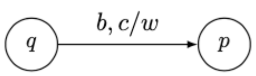
Beispiel für $\{0^n1^n \mid n > 0\}$




7-Tupel: $\left( \begin{array}{c} Q, \Sigma = \text{Eingabealphabet}, \\ \Gamma = \text{Kelleralphabet}, \delta, q_0, \$, F \end{array} \right)$

Übergangsfunktion

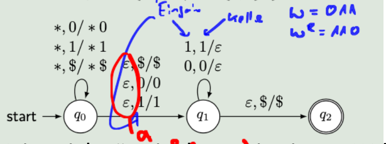
$$\delta(curr, input, stack) = (\text{next}, write\ to\ stack)$$




b: Eingabe,
c: gelesenes Symbol vom Stack
w: Wort, das auf Stack geschrieben wird

## NKA

Kellerautomat für die Sprache $\{ww^R \mid w \in \{0,1\}^*\}$:




Ein NKA ist nicht äquivalent mit einem KA!

## Berechnung

Gegeben sei $M_1$ (für die Sprache $L = \{0^n1^n \mid n > 0\}$) und die Wörter $w' = 0011$ u. $w'' = 011$.

Die Berechnungen für $w' = 0011$ u. $w'' = 011$ lauten:

- $(q_0, 0011, \$) \vdash (q_0, 011, 0\$) \vdash (q_0, 11, 00\$) \vdash (q_1, 1, 0\$)$
  $\vdash (q_1, \epsilon, \$) \vdash (q_2, \epsilon, \$)$
  Startkonfiguration
  $\Rightarrow$ *Die Berechnung ist akzeptierend.*
- $(q_0, 011, \$) \vdash (q_0, 11, 0\$) \vdash (q_1, 1, \$) \vdash \cancel{(q_2, 1, \$)}$ $\vdash (q_{abfall}, \varepsilon, \$)$
  Es ist kein weiterer Berechnungsschritt mehr möglich und es wird keine Endkonfiguration erreicht.
  $\Rightarrow$ *Die Eingabe $w''$ wird nicht akzeptierend.*

## Äquivalenz mit kontextfreien Sprachen

- NKA existiert = Sprache ist kontextfrei

Nicht kontextfreie Sprachen

$\{0^n1^n2^n \mid n > 0\}$ => nicht kontextfrei

- Ein Kellerautomat, der ein Wort aus $L$ akzeptieren würde, müsste sich die Anzahl der eingelesenen 0 und 1 merken.
- Mit dem Keller kann aber nur *einmal* eine Anzahl verglichen werden, danach sind die Symbole nicht mehr auf dem Keller.
- Mit Zuständen kann, wie beim EA, nicht beliebig gezählt werden.

⇨ Braucht eine Trennung der Worte (wie z.B. bei $L_2 = \{waw^R \mid w \in \{0,1\}^*\}, \Sigma = \{0,1,a\}$)

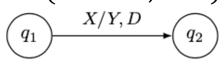
## Turingmaschinen

$$M = (Q, \Sigma, \Gamma = \text{Bandalphabet}, \delta, q_0, \sqcup, F)$$

**Bandalphabet:** endliche Menge von Symbolen

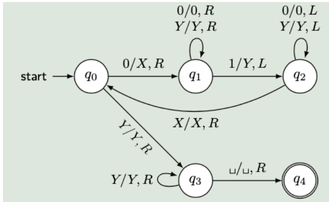
Übergang

$$\delta(current, \text{read}) = (next, \text{write symbol}, \text{movement})$$




$$\delta(q_1, X) = (q_2, Y, D)$$

Beispiel für $\{0^n1^n \mid n \geq 1\}$




Sprache: rekursiv aufzählbar

## Erweiterungen

Mehrere Spuren

- Band setzt sich aus mehreren Spuren zusammen
- Kann über eine TM mit einer Spur simuliert werden

Mehrere Bänder

- Gleichmächtig, wie 1-Band TM

## NTM

- Gleichmächtig wie eine deterministische TM (Beweis: Simulation der NTM mit 2-Band DTM)

## Beschränkungen

Semi-unendliches Band

- Nur in eine Richtung unendlich (z.B. rechts)
- Gleichmächtig wie TM

Maschine mit mehreren Stacks

- DKA mit mehreren Stacks
- 2-Stack KA ist gleichmächtig wie TM

- $S$ liest die Eingabe zunächst in den 1. Stack.
- Nun simuliert $S$ die Bewegung von $T$:
  Abhängig von der Bewegungsrichtung von $T$ wird ein Symbol vom 1. oder 2. Stack von $S$ gelesen (*pop*) und neu, ggf. modifiziert, in den jeweils anderen Stack geschrieben.

Zähler-Maschine

- Zähler-Maschine mit k Zähler = k-Stack-Maschine
- Stacks werden durch Zähler ersetzt

Wert des Zählers: >= 0, wird jeweils um 1 in- oder dekrementiert

2-Zähler Maschine gleichmächtig wie TM

a) Ein 2-Stack-Maschine kann eine TM simulieren.
b) Eine Zählermaschine mit 3 Zählern kann eine 2-Stack-Maschine simulieren.
c) Eine Zählermaschine mit 2 Zählern kann eine Zählermaschine mit 3 Zählern simulieren.

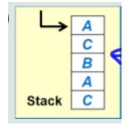
Stack -> Zahl

Gegeben: $\Gamma = \{A, B, C\}$
mit der Codierung: $A \to 1, B \to 2$ und $C \to 3$ $(k = 4)$.

A) Der Stack $s$ mit dem Inhalt $ACBAC$ ($A$ oberstes Element) ergibt die Zahl:

push(C): $0 * 4 + 3 = 3$ ($C = 3$ und Stack war leer $= 0$)
push(A): $3 * 4 + 1 = 13$ $(A = 1)$
push(B): $13 * 4 + 2 = 54$ $(B = 2)$
push(C): $54 * 4 + 3 = 219$ $(C = 3)$
push(A): $219 * 4 + 1 = 877$ $(A = 1)$

$\underset{A}{1} \cdot 4^0 + \underset{C}{3} \cdot 4^1 + \underset{B}{2} \cdot 4^2 + \underset{A}{1} \cdot 4^3 + \underset{C}{3} \cdot 4^4 = 1 + 12 + 32 + 64 + 768 = \underline{877}$




Zahl -> Stack

B) Die Zahl 921 repräsentiert den Stack mit dem Inhalt $ABABC$:

Pop: geteilt durch 4

$921/4 = 230$ Rest 1 (1 repräsentiert $A$):
$230/4 = 57$ Rest 2 (2 repräsentiert $B$):
$57/4 = 14$ Rest 1 (1 repräsentiert $A$):
$14/4 = 3$ Rest 2 (2 repräsentiert $B$):
und $3/4 = 0$ Rest 3 (3 repräsentiert $C$):

Polynomdarstellung:
$\underset{A}{1} \cdot 4^0 + \underset{B}{2} \cdot 4^1 + \underset{A}{1} \cdot 4^2 + \underset{B}{2} \cdot 4^3 + \underset{C}{3} \cdot 4^4 = 1 + 8 + 16 + 128 + 768 = \underline{921}$

**Pop:** Teilen durch k
**Push:** Multp. mit k

## UTM

- TM kann durch (unär) Codierung als UTM dargestellt werden
- Kodierung einer TM kann als Zahl dargestellt werden (führende 1) => rekonstruieren

## Berechnungsmodelle

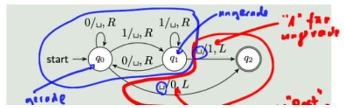
## Church-Turing-These

Turing-berechenbare Funktionen = intuitiv berechenbare Funktionen

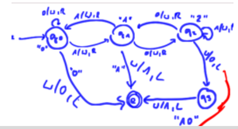
Turing-berechenbare Funktionen

Funktionen, die von einer TM berechnet werden können. Bsp.:

Mod2




Mod3



## LOOP-Programme

⇒ Terminieren immer -> endlich viele Schritte

- Variablen, Konstanten, +, -, Loop, Do, End
- Wert der Berechnung steht in x0
- Variablen können als Werte **nat. Zahlen >= 0** halten

Zuweisungen

x0 = Variable + Konstante    Bsp.: x0 = x1 + 0

Makros

**IF x1 = 0 THEN P**

```
x2 = x2 + 1;
Loop x1 Do
    x2 = x2 - 1
End;
Loop x2 Do
    P
End
```

P ← LOOP Prog.

**x0 = x1 + x2**

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

**IF x1 = 0 THEN P ELSE Q**

```
x2 = x2 + 1;
x3 = x3 + 1;
Loop x1 Do
    x2 = x2 - 1;
    x4 = x3 + 0
End;
Loop x2 Do
    P
End;
Loop x4 Do
    Q
End
```

## WHILE-Programme

Erweiterung der LOOP-Programme mit WHILE xi > 0 Do … End

- Jedes Loop-Programm ist auch ein While-Programm
- ⇨ Terminiert nicht immer!
- Turing-vollständig

## GOTO-Programme

Marker: M1, M2, …; Schlüsselwörter: Goto, If, Then, Halt

- Turing-vollständig

<u>Beispiel (Addition)</u>

```
M1: x0 = x1 + 0;
M2: If x2 = 0 Then Goto M6;
M3: x2 = x2 - 1;
M4: x0 = x0 + 1;
M5: Goto M2;
M6: Halt
```

## Primitiv rekursive Funktionen

Menge von Funktionen, die aus einfachen Grundfunktionen konstruiert werden

- ⇨ Loop-berechenbare Funktionen (<u>endliche Loops</u>)

<u>Grundfunktionen</u>

Konstante Funktion: $c_k^n = \mathbb{N}^n \to \mathbb{N}$ mit $c_k^n(x_1, \ldots, x_n) = k$

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

Projektion: $\pi_k^n = \mathbb{N}^n \to \mathbb{N}$ mit $\pi_k^n(\underbrace{x_1, \ldots, x_k \ldots, x_n}_{n = \text{Anz. Argumente}}) = k$

## Entscheidbarkeit

<u>Entscheidbar:</u> TM muss nach **endlich** vielen Schritten halten

- ⇨ Jede entscheidbare Sprache ist auch semi-entscheidbar

<u>Entscheidungsverfahren:</u> WHILE-Programm

## Semi-Entscheidbarkeit

- ⇨ Hält nicht bei invalider Eingabe
- ⇨ Rekursiv aufzählbar, total berechenbare Funktion

**Satz**

*Eine Sprache $A \subset \Sigma^*$ ist genau dann entscheidbar, wenn sowohl $A$ als auch $\overline{A}$ semi-entscheidbar ist.*

- *Ist $A \subset \Sigma^*$ eine entscheidbare Sprache, dann ist auch $\overline{A}$ eine entscheidbare Sprache.*
- *Sind $A, B$ (semi-) entscheidbare Sprachen, dann sind auch $A \cup B$ und $A \cap B$ (semi-) entscheidbare Sprachen.*

## Reduktion

Umformulierung (totale Turing-berechenbare Funktion) einer Problemstellung auf die andere

$A \preceq B$: B «kann mindestens gleich viel» wie A

<u>Transitivität:</u> $A \preceq B$ und $B \preceq C \Rightarrow A \preceq C$

**Satz**

*Für beliebige Sprachen $A \subset \Sigma^*$ und $B \subset \Gamma^*$ gilt:*

- *Ist $B$ entscheidbar und $A \preceq B$, dann ist auch $A$ entscheidbar.*
- *Ist $B$ semi-entscheidbar und $A \preceq B$, dann ist auch $A$ semi-entscheidbar.*

## Halteprobleme

- Ob eine TM auf gegebenem Input anhält

<u>Spezielles Halteproblem</u>

Input = Code der TM

**Nicht entscheidbar** -> es gibt kein TM, die das Hs entscheidet

<u>Leere Halteproblem:</u> Ob eine TM auf dem leeren Band anhält

- ⇒ Semi-entscheidbar: $H_s \preceq H \preceq H_0$

# Komplexitätstheorie

Klassifizierung von (entscheidbaren) Problemen nach ihrer Schwierigkeit (Komplexität)

<u>Zeitkomplexität</u>

- Anzahl Berechnungsschritte einer TM
- Laufzeit von grossen Eingaben ist ausschlaggebend

| $f(n)$ | 10 | 50 | 100 | 300 |
|---|---|---|---|---|
| $20\log_2(n)$ | $\approx 83$ | $\approx 113$ | $\approx 133$ | $\approx 165$ |
| $10n$ | 100 | 500 | 1 000 | 3 000 |
| $2n^2$ | 200 | 5 000 | 20 000 | 180 000 |
| $n^3$ | 1 000 | 125 000 | 1 000 000 | 27 000 000 |
| $2^n$ | 1 024 | 16 Ziffern | 31 Ziffern | 91 Ziffern |
| $n!$ | $\approx 3.6 \cdot 10^6$ | 65 Ziffern | 158 Ziffern | 615 Ziffern |

## Big-O-Notation

- Konstante Vorfaktoren können ignoriert werden
- $f \in \mathcal{O}(g)$ Es existiert ein $n_0 \in \mathbb{N}$ und ein $c \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt $f(n) \leq c \cdot g(n)$ $f$ wächst asymptotisch *nicht schneller* als $g$
- $f \in \Omega(g)$ Es existiert ein $n_0 \in \mathbb{N}$ und ein $d \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt $f(n) \geq \frac{1}{d} \cdot g(n)$ $f$ wächst asymptotisch *mindestens so schnell* wie $g$

𝒪( 1 ✔ )⊂ 𝒪( n • log(n) ✔ )⊂ 𝒪( 1000+n^2 ✔ )⊂ 𝒪( n^2 • log(n)+99 ✔ )⊂ 𝒪( n^(5/2) ✔ )⊂ 𝒪( 3^n+n^2 ✔ )⊂ 𝒪( 5^(n+3) ✔ )⊂ 𝒪( 3^(n!) ✔ )

$n\log(n) \in \mathcal{O}(n^2)$

- $f(n) = 25n^2 + n^3 + 100n$ $\in \mathcal{O}(n^3)$
- $f(n) = n^2 + n \cdot n(log(n)) + 20n^2 + 50n \cdot 100$ $\in \mathcal{O}(n^2 \cdot \log(n))$
- $f(n) = 50 \cdot log(n) + log^2(n) + 10 \cdot \sqrt[2]{n}$ $\in \mathcal{O}(\sqrt[2]{n})$ $\log(n) \cdot \log(n)$
- $f(n) = 10n \cdot log(\sqrt[2]{n}) + log(n) \cdot \frac{1}{2}n$ $\in \mathcal{O}(n \cdot \log(n))$
- $f(n) = 10^{20} + 3n^3 + 2^n + 2^{10} \cdot 2^{30}$ $\in \mathcal{O}(2^n)$

## Schranken

- $\mathcal{O}(f(n))$ ist eine **obere Schranke** für die Zeitkomplexität von $U$, falls **eine TM existiert**, die $U$ löst und eine Zeitkomplexität in $\mathcal{O}(f(n))$ hat.
- $\Omega(g(n))$ ist **untere Schranke** für die Zeitkomplexität von $U$, falls **für alle TM** $M$, die $U$ lösen, gilt, dass $\text{Time}_M(n) \in \Omega(g(n))$.

## P-Probleme

In Polynomzeit lösbar: $\mathcal{O}(n^c)$ mit $c \geq 1$

- ⇨ Lösung **finden** in Polynomzeit

## NP-Probleme

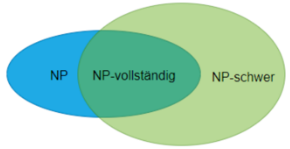
- Nichtdeterministisch polynomiell

Klasse aller von einer **NTM** in Polynomzeit **entscheidbaren** Sprachen

- ⇨ Lösung **verifizieren** in Polynomzeit

Menge aller Sprachen, für die ein Polynomzeit-Verfizierer existiert

<u>NP-Vollständigkeit</u>




**Definition (NP-schwer)**

Eine Sprache $L$ heisst **NP-schwer**, falls für alle Sprachen $L' \in \mathbf{NP}$ gilt, dass $L' \preceq_p L$.

D.h. $L$ ist bezüglich der Lösbarkeit in polynomieller Zeit mindestens so schwer wie jedes einzelne Problem in $\mathbf{NP}$.

<u>SAT</u>

*Wenn $P_1$ **NP**-schwer und $P_2$ in **NP** enthalten ist und eine polynomielle Reduktion $P_1 \preceq_p P_2$ existiert, dann ist $P_2$ **NP**-vollständig.*

<u>Clique NP-Vollständigkeit</u>

Setze $k = m$ (Anzahl der Klauseln).

Für jedes Auftreten eines Literals in einer Klausel $i$ an Stelle $j$ in $\varphi$ wird ein Knoten $[i, j]$ gesetzt. Kanten werden zwischen Knoten aus unterschiedlichen Klauseln gesetzt, falls die Literale nicht Negationen voneinander sind.

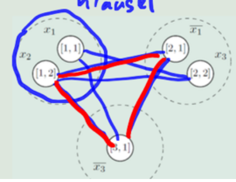
**Beispiel (CLIQUE ist NP-vollständig)**

Beispiel für die Konstruktion: $\varphi = \underbrace{(x_1 \vee x_2)}_{\text{Klausel}} \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_3})$

1|1 1|2 2|1 2|2 3|1

Klausel

literal




$x_1 = 0; x_2 = 1; x_3 = 0$
$\Rightarrow e = 1$

| $x_1$ | $x_2$ | $x_3$ | $x_1 \vee x_2$ | $\overline{x_1} \vee x_3$ | $\overline{x_3}$ | $e$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | | 0 |
| 0 | 0 | 1 | | | | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |

## Polynomzeit-Verifizierer

<u>Graph:</u> $G = (V = Knoten, E = Kanten)$

<u>Clique:</u> alle Knoten aus einer Teilmenge sind paarweise verbunden

- ⇨ Verifizierer existiert, falls man die Lösung für einen Lösungskandidat (Zeuge) findet

**Beispiel (Polynomzeit-Verifizierer für das CLIQUE-Problem)**

Eingabe:

- Ein ungerichteter Graph mit $n$ Knoten und eine Zahl $k$.
- Zeuge: Menge von Knoten, die in der Clique sind.
  Lösungskandidat

Der Verifizierer überprüft, ob es sich tatsächlich um eine Clique handelt, d. h. ob in dieser Menge zwischen je zwei Knoten eine Kante vorhanden ist.