Alphabete, Wörter und Sprachen

«Alphabet (Σ)»: <u>endliche, nichtleere</u> Menge.

- Leeres Wort = ε = «» = über <u>jedes</u> Alphabet.

«Wort (w)»: <u>endliche</u> Folge von Symbolen <u>über</u> ein bestimmtes Alphabet (*Achtung: Eigentlich ein Tupel; Wenn aber eindeutige Zuordnung der Symbole im Alphabet kann es weggelassen werden*).

⇨ Unendlich viele Wörter über ein Σ.

«Sprache (L)»: <u>unendliche oder endliche</u> Menge von Wörtern ($L \subseteq \sum^*$ ($\sum^*$Sprache über jedes Alphabet) .

- Leeres Wort nicht in jeder Sprache.
- Ø != {ε} (da [] != [«»])
- Konkatenation: AB != BA (Präfix A + Suffix B)
- L* = (L*)* (hat bereits jede Kombi von «w»)
- Ø über jedes Alphabet (mit <u>keinen</u> Wörtern)

Wortkonventionen

Definition	Beispiel	Beschreibung
$\|w\|$	$\|10011\| = 5$	Wortlänge
$\|w\|_X$	$\|abc\|_a = 1$	Symbolhäufigkeit (X)
w^R	$(abc)^R = cba$	Spiegelwort
$w^R = w$	$(anna)^R = anna$	Palindrom
$x \circ y\ (= xy)$	$ab \circ cd = abcd$	Konkatenation
$\|x \circ y\| = \|x\| + \|y\|$	-	Konkatenationlänge
$w = vy$	$w = \varepsilon abba$ Präfix hier = ε	Präfix v (echt wenn $y \neq \varepsilon$)
$w = xv$	$w = abba\varepsilon$ Suffix hier = ε	Suffix v (echt wenn $x \neq \varepsilon$)
$w = xvy$	$w = aabba$ Infix hier = ab «v» an einem Stück!	Infix (Teilwort) v (echt wenn $\neg(x = \varepsilon \land y = \varepsilon)$)
$w^X = www \ldots$	$w^3 = www$ $w^0 = \varepsilon$ $w^{n+1} = w^n \circ w$	Wortpotenz nach X (Achtung: 1. Symbol ist «inkl.» X)
Σ^* $= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots)$	$\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ $(= \Sigma^* - \{\varepsilon\})$	Kleenesche Hülle (immer unendlich)
Σ^k	$\Sigma^0 = \{\varepsilon\}$	Wörter mit Länge k. (nie unendlich)

Reguläre Ausdrücke

> Def.: Wörter die Sprachen definieren.
Möglichkeit Sprachen endlich darzustellen.

Es sei Σ ein beliebiges Alphabet. Die Sprache RA_Σ der **regulären Ausdrücke** über Σ ist wie folgt definiert:

- $\emptyset, \epsilon \in \mathsf{RA}_\Sigma$ leerer RegEx (nicht leeres Wort):
- $\Sigma \subset \mathsf{RA}_\Sigma$
- $R \in \mathsf{RA}_\Sigma \Rightarrow (R^*) \in \mathsf{RA}_\Sigma$
- $R, S \in \mathsf{RA}_\Sigma \Rightarrow (RS) \in \mathsf{RA}_\Sigma$ $L_1(\emptyset) = \{\}$
- $R, S \in \mathsf{RA}_\Sigma \Rightarrow (R|S) \in \mathsf{RA}_\Sigma$ $L_2(\varepsilon) = \{\varepsilon\}$

⇨ **«*» vor Konkatenation vor «|»**

⇨ Reguläre Sprache, falls RegEx existiert.

zB. $\{w \in \{a,b\}^* | a^m b^n \text{ mit } m > n\}$ nicht regulär.

> Beispiel:

L(R_2): Menge der Binärwörter mit abwechselnd Nullen und Einsen

$$R_2 = (1|\varepsilon)(01)^+(0|\varepsilon)$$

Endliche Automaten (EA)

Definition (Endlicher Automat)

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

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

mit

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

Anmerkung: Das **kartesische Produkt von** A **und** B ist definiert als

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}.$$

Definition (Nichtdeterministischer endlicher Automat)

Ein **nichtdeterministischer endlicher Automat** (NEA) ist ein Quintupel

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

wobei Q, Σ, q_0 und F wie beim deterministischen endlichen Automaten (ab jetzt: DEA) definiert sind und die Übergangsfunktion δ definiert ist als

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

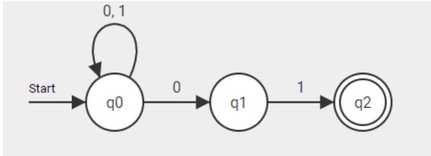
Endliche Automat (EA):

$\delta(q_1, a) = (q_2)$: $q_1 \xrightarrow{a} q_2$

Teilmengenkonstruktion Ablauf

> Ziel: Von einem NEA in einen DEA

> Anhand eines Beispiels:



	Zustandsm.	0	1
	~~Ø~~	~~Ø~~	~~Ø~~
->	A: {q0}	B:{q0, q1}	A:{q0}
	~~{q1}~~	~~Ø~~	~~{q2}~~
*	~~{q2}~~	~~Ø~~	~~Ø~~
->	B: {q0,q1}	B: {q0,q1}	C: {q0,q1}
-> *	C: {q0, q2}	B: {q0, q1}	A: {q0}
*	~~{q1, q2}~~	~~Ø~~	~~{q2}~~
->*	~~{q0, q1, q2}~~	~~{q0, q1}~~	~~{q0, q2}~~

Achtung (zu Punkt (4)):

Da q2 auch gestrichen wurde, zeigt jetzt darum neu q1 auch nirgendwo hin!

(1) Alle Zustandsmengen definieren / ablesen.

⇨ Jede mögliche Zustandskombination!

⇨ Bei 0 / 1 Spalte zB. {q0, q1} einfach eine Menge bilden aus den jeweiligen Mengen von q0 und q1.

(2) Kandidaten für Startzustand wählen.

⇨ q0 muss bei Zustandsmengen Spalte vorkommen (oben mit «->» markiert).

(3) Kandidaten für Endzustand wählen.

⇨ q2 muss bei Zustandsmengen Spalte vorkommen (oben mit «*» markiert).

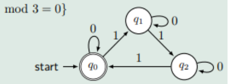
(4) Nicht erreichbare Zustände streichen.

⇨ Zustandsmenge nirgendwo anders in 0 oder 1 Spalte vorhanden (Achtung: q2!).

DEA zu 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.

$L_5 = \{ w \in \{0,1\}^* \mid |w|_1 \mod 3 = 0\}$

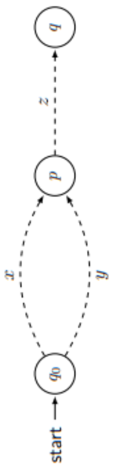


Nichtterminale:
Q_0, Q_1, Q_2

Produktionen:
$Q_0 \to 0Q_0 \mid 1Q_1 \mid \varepsilon$
$Q_1 \to 0Q_1 \mid 1Q_2$
$Q_2 \to 0Q_2 \mid 1Q_0$

Satz
Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein EA und x und y zwei beliebige Wörter aus Σ^*, so dass $x, y \in [p]$. Dann gilt für alle Wörter $z \in \Sigma^*$

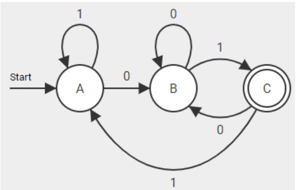
$$xz \in L(M) \iff yz \in L(M). \quad (*)$$



(5) Zustandsmengen benennen und neue Tabelle.

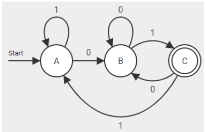
Zustandsm.	0	1
A	B	A
B	B	C
C	B	A

(6) DEA zeichnen (Start = NEA, End = gemäss (3)).



Berechnung von DEA (NEA analog)

> Beispiel:



$w = 1101$
$(A, 1101) \vdash (A, 101) \vdash (A, 01) \vdash (B, 1) \vdash (C, \varepsilon)$
⇨ w ist akzeptierend.

> Anmerkung zu NEA:

- Sobald ein Pfad akzeptierend, dann w akzeptierend.
- εNEA: Spontane Zustandsänderung durch ε.

> Anmerkung Allgemein:

- **DEA sind gleichmächtig zu RegEx** (Einfach beweisbar mit εNEA und 2 Automaten).
- Komplement von regulären Sprachen auch regulär.
- Zustandsklassen = Äquivalenzklassen

⇨ $L = \{a^n b^n | n \in \mathbb{N}\}$ ∞Klassen (=nicht regulär)
$w_1 = a, w_2 = aa \ldots \to z = b: w_1 z \in L \ \wedge \ w_2 z \notin L \ldots$

Kontextfreie Grammatik (KFG)

Definition (Kontextfreie Grammatik)

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

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

$$X \to \beta$$

mit **Kopf** $X \in N$ und **Rumpf** $\beta \in (N \cup \Sigma)^*$. ≙ "Satzform"
- A ist das **Startsymbol**, wobei $A \in N$.

> Ableitungen:

- KFG enthalten RegEx.
- Folge von Ableitungsschritten, so dass aus Startsymbol A von KFG G ein Wort w abgeleitet wird (w ist «ableitbar» in G).
  = w wird von A erzeugt/generiert($A \overset{*}{\Rightarrow} w$).
- Linksseitig ersetzt jedes Nichtterminal, welches ganz links ist (Rechtseitig analog).
- Kontextfreie Sprache: Falls für L ein KFG.

⇨ KFG = mehrdeutig falls mehrere Ableitungsbäume existieren.

> Beispiel KFG:

(a) $L_0 = \{ w \mid w \text{ ist eine beliebige Hexadezimalzahl} \}$

(b) $L_1 = \{ w \mid w \text{ ist eine Hexadezimalzahl } \geq 32 \}$

(a) $G = \{\{D, Z, A\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}, P, A\}$ mit den Produktionen P:

$D \to 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid a \mid b \mid c \mid d \mid e \mid f$
$Z \to 0 \mid D \mid ZZ$
$A \to 0 \mid D \mid DZ$

(b) $G = \{\{K, D, Z, A\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}, P, A\}$ mit den Produktionen P:

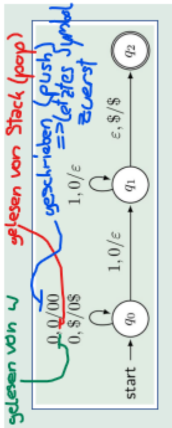
$K \to 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid a \mid b \mid c \mid d \mid e \mid f$
$D \to 1 \mid K$
$Z \to 0 \mid D \mid ZZ$
$A \to KZ \mid DZZ$

Kellerautomaten (KA)

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.
- $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to Q \times \Gamma^*$ 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.



Ein Berechnungsschritt $\delta(q, b, c) = (p, w)$ wird wie folgt interpretiert:

1. Der Automat befindet sich im Zustand q.
2. Der Automat liest das Symbol b von der Eingabe (falls $b = \varepsilon$, wird nichts gelesen).
3. Der Automat entfernt das oberste Kellersymbol c.
4. Der Automat schreibt das Wort w auf den Stack (von hinten nach vorne).
5. Der Automat wechselt in den Zustand p.

- **Eine Sprache ist kontextfrei, wenn sie von einem NKA erkannt wird** (nicht unbedingt von einem DKA).
- Kontextfreie Sprachen, welche von einem DKA erkannt werden, sind eindeutig.
- Determinismus Kriterien:
  - Falls $\delta(q1, b, c)$ definiert wurde, darf kein $\delta(q1, b, c)$ definiert sein (Eingabe + Stack anders!)
  - Falls $\delta(q1, b, c)$ definiert wurde, darf kein $\delta(q1, \varepsilon, c)$ sein.

> Berechnung Beispiel:

$(q_0, 0011, \$) \vdash (q_0, 011, 0\$) \vdash (q_0, 11, 00\$) \vdash (q_1, 1, 0\$)$
$\vdash (q_1, \epsilon, \$) \vdash (q_2, \epsilon, \$)$ → Ist notwendig (Stack muss nicht leer sein)
⇒ *Die Berechnung ist akzeptierend.*

Eine **Konfiguration von** M ist ein Element (q, w, γ) aus $Q \times \Sigma^* \times \Gamma^*$, wobei

- q für den Zustand steht,
- w die verbleibende Eingabe repräsentiert,
- γ für den Inhalt des Kellers steht.
  (Dabei steht das Symbol ganz links für das oberste Symbol.)

> Beispiel: NKA und/oder DKA erkennbar?

$L_2 = \{waw^R \mid w \in \{0,1\}^*\}, \Sigma = \{0, 1, a\}$ → DKA
$L_3 = \{ww \mid w \in \{0,1\}^*\}, \Sigma = \{0, 1\}$ → ✓
$L_4 = \{0^n 1^n 0^n \mid n > 0\}, \Sigma = \{0, 1\}$ → ✓

Beispiel Beweisablauf von «EA hat mind. n Zustände (Klassen)»

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: $x_1 = \varepsilon$, $x_2 = 0$, $x_3 = 00$
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. □

Kellerautomat (KA):

$\delta(q_1, a, b) = (q_2, w)$: $\quad q_1 \xrightarrow{a, b/w} q_2$

Turingmaschinen (TM)

Definition (Turing-Maschine)

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

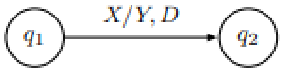
$$M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$$

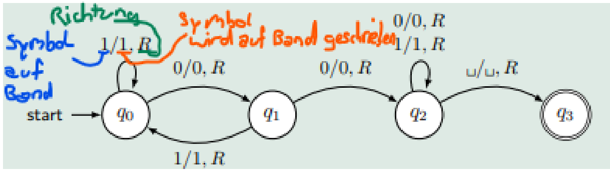
mit einer bzw. einem:

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

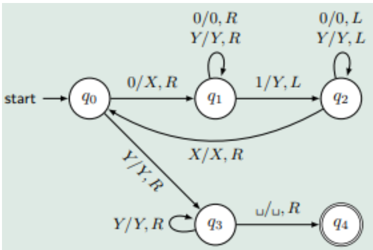
Turing-Maschine (TM):

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





> Beispiel:



Berechnungen mit M_2 für a) $w = 01$ und b) $w = 0011$:

a) $q_0 01 \vdash X q_1 1 \vdash q_2 XY \vdash X q_0 Y \vdash XY q_3 \vdash XY \sqcup q_4$

b) $q_0 0011 \vdash X q_1 011 \vdash X0 q_1 11 \vdash X q_2 0Y1 \vdash q_2 X0Y1 \vdash X q_0 0Y1 \vdash XX q_1 Y1 \vdash XXY q_1 1 \vdash XX q_2 YY \vdash X q_2 XYY \vdash XX q_0 YY \vdash XXY q_3 Y \vdash XXYY q_3 \vdash XXYY \sqcup q_4$

Leerzeichen nur wenn nötig angeben!

Beide Berechnungen enden in q_4, einem **akzeptierenden Zustand** $(q_4 \in F)$ von M_2.

> Anmerkungen:

- Eingabe wird anfangs auf das Band geschrieben (L/S-Kopf ist über 1. Symbol).
- **TM akzeptiert rekursiv aufzählbare Sprachen (auch rekursive Sprachen).**
- *Entscheidungsproblem umformuliert: Wort w in Sprache L?*
- Wenn TM anhält, dann fertig (akzeptierend falls Zustand akzeptierend).

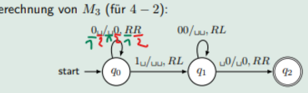
⇨ Bandinhalt ist dann das Resultat.

> TM-Erweiterungen (exkl. mehr Spur, Speicher):

- **Beispiel TM mit mehreren Bänder:**

Beispiel (Subtraktion zweier ganzer positiver Zahlen a und b, $(a > b)$)

b) Berechnung von M_3 (für $4 - 2$):

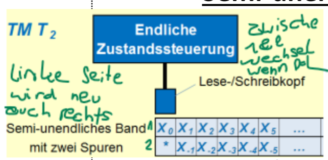


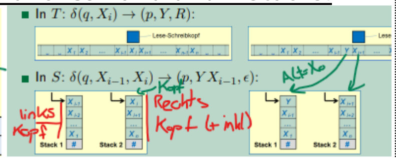
Band 1:	$q_0 0000100$	$\vdash q_0 000100$	$\vdash q_0 00100$	$\vdash q_0 0100$	$\vdash q_0 100$
Band 2:	q_0	$\vdash 0 q_0$	$\vdash 00 q_0$	$\vdash 000 q_0$	$\vdash 0000 q_0$
		$\vdash q_1 00$	$\vdash q_1 0$	$\vdash q_1$	$\vdash q_2$
		$\vdash 000 q_1 0$	$\vdash 00 q_1 0$	$\vdash 0 q_1 0$	$\vdash 00 q_2$

- **NTM:**

⇨ Nur Bandsymbol wichtig (in NTM dazu mehrere Funktionen im selben Zustand mit gleichem Symbol möglich).

- **Semi-unendliches Band und 2 Stacks:**





⇨ 2 Stack DKA gleichmächtig wie TM.

⇨ TM mit semi-∞ Band gleichmächtig TM.

⇨ Automat mit 2 Zähler gleichmächtig wie TM (Bspw. ist 14 mit k = 4, A=1, B=2, C=3: 14/k=Rest 2 = B, 3/k = Rest 3 = C …
=> Repräsentiert einen Stack als Zahl)

⇨ Zählermaschine mit 2 Zählern kann eine mit 3 Zählern simulieren. Diese kann eine Maschine mit 2 Stack simulieren.

> Universelle TM:

q1: Start
q2: End
q3…: Übrige

Bandsymbol
$X_1 = 0$, $X_2 = 1$, $X_3 = \sqcup$

$D_1 = L$
$D_2 = R$

Ein Übergang $\delta(q_i, \underbrace{X_j}_{\text{lesen}}) = (q_k, \underbrace{X_l}_{\text{schreiben}}, D_m)$ einer TM wird codiert über die Zeichenreihe:

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m \text{ mit } (i, j, k, l, m \in \mathbb{N})$$

⇨ Mit 111 = w (TM#w).

⇨ Mit 11 Abstände zwischen Übergänge.

Modelle der Berechenbarkeit

> Turing-Berechenbar: Partielle Funktion $T : \Sigma^* \to \Gamma^*$

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

⇨ *Total, wenn für jede Eingabe definiert.*

> Loop Programme:

- Variablen: `x0, x1, x2, ...`
- Konstanten: `0, 1, 2, 3, 4, ...`
- Trennzeichen: ;
- Zuweisung: =
- Operationszeichen: + und −
- Schlüsselwörter: `Loop, Do, End`

• Loop x Do
P
End
• P; Q
(schluss nicht

- x1 = x1 + x2 nicht erlaubt.
- x0 ist der Returnwert.

> While Programme:

- Terminieren nicht immer.
- Gleichmächtig zu TM und GOTO.
- Zähler ist im While Rumpf änderbar!

> Goto Programme:

- Variablen: `x0, x1, x2, ...`
- Konstanten: `0, 1, 2, 3, 4, ...`
- Marker: `M1, M2, ...`
- Zuweisung: =,
- Trennzeichen: ;, :
- Operationszeichen: + und −
- Schlüsselwörter: `Goto, If, Then, Halt`

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

Wie wird der Übergang $\delta(q_1, 1) = (q_3, 0, R)$ kodiert?
- der Zustand q_1 wird über 0 kodiert
- das Bandsymbol 1 über 00 kodiert
- der Zustand q_3 über 000 kodiert
- das Bandsymbol 0 über 0 kodiert
- und die Bewegung R über 00 kodiert

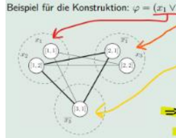
Das ergibt zusammengesetzt für $\delta(q_1, 1) = (q_3, 0, R)$: 0100100010100

- *Ein Entscheidungsverfahren für das Halteproblem wäre eine totale Funktion.*
- *Die Collatzzahlen sind semi-entscheidbar, da sie rekursiv aufzählbar sind.*
- *Komplement einer semi-entscheidbaren Sprache ist immer nicht-semi-entscheidbar!*
- *Komplement einer nicht-semi-entscheidbaren Sprache ist auch nicht-semi-entscheidbar / nicht entscheidbar.*

Beispiel SAT $\leq_p$ CLIQUE

SAT $\hat{=}$ NP Vollständig
⇒ Satz von Cook

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



= 3

Die Formel ist erfüllbar.

⇔

Der Graph enthält eine Clique der Grösse 3.

⇒ Keine Kante zwischen x und $\bar{x}$.

Beispiele warum Loop Programme / primitive Rekursion nicht gleichmächtig wie TM.

> Primitiv rekursive Funktionen: (gleich wie Loop)

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

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

Projektion: $\pi_k^n : \mathbb{N}^n \to \mathbb{N}$, $\pi_k^n(x_1, \dots, x_k, \dots, x_n) = x_k$

Einsetzen (immer noch primitiv rekursiv):

$$h : \mathbb{N}^n \to \mathbb{N} \quad \text{mit} \quad h(\vec{x}) = f(g_1(\vec{x}), \dots, g_k(\vec{x})),$$

Primitive Rekursion (immer noch primitiv rekursiv):

$$f(0, \vec{x}) = h(\vec{x})$$
$$f(k+1, \vec{x}) = g(f(k, \vec{x}), k, \vec{x})$$

Beispiel:

$$Add(0, y) = y \qquad Add(x+1, y) = Add(x, y) + 1$$

$$Add(0, y) = \pi_1^1(y) \qquad Add(x+1, y) = \eta(\pi_1^3(Add(x, y), x, y)).$$

> Ackermannfunktionen: (TM-berechenbar)

⇨ Totale Funktion: nicht Loopberechenbar «wachsen schneller»(exp. nach Parametern)

Die **Ackermannfunktion** $a : \mathbb{N}^2 \to \mathbb{N}$ ist durch die Gleichungen

$$a(0, m) = m + 1$$
$$a(n+1, 0) = a(n, 1)$$
$$a(n+1, m+1) = a(n, a(n+1, m))$$

gegeben.

> Loopinterpreter: (TM-berechenbar)

Ein **LOOP-Interpreter** ist eine Funktion $I : \mathbb{N}^2 \to \mathbb{N}$, die für jedes LOOP-Programm P und jede natürliche Zahl x die Gleichung

$$I(\langle P \rangle, x) = P_1(x)$$

⇨ Wenn x so gewählt wird, dass für jede N Zahl ein Loop Programm, dann genau 1 totales I.

Entscheidbarkeit

Eine Sprache $A \subset \Sigma^*$ heisst **entscheidbar**, wenn eine Turingmaschine T existiert, die das Entscheidungsproblem (Σ, A) löst. "liegt in Sprache oder nicht?"

⇒ T für $\forall x \in \Sigma^*$ anhalten!

Eine Sprache $A \subset \Sigma^*$ heisst **semi-entscheidbar**, wenn eine Turingmaschine T existiert, die sich wie folgt verhält:

- Wenn T mit Bandinhalt $x \in A$ gestartet wird, dann hält T nach endlich vielen Schritten mit Bandinhalt "1"(Ja) an.
- Wenn T mit Bandinhalt $x \in \Sigma^* \setminus A$ gestartet wird, dann hält T nie an.

⇒ Nur "ja" Antworten. Keine Antwort bei "Nein"

Bemerkung (Konsequenzen) Satz von Rice

- Es ist (im Allgemeinen) unmöglich mechanisch zu überprüfen, ob ein gegebenes Programm eine bestimmte Spezifikation erfüllt.
- Es ist (im Allgemeinen) unmöglich mechanisch zu überprüfen, ob ein gegebenes Programm frei von "bugs" ist.
- Es ist (im Allgemeinen) unmöglich mechanisch zu überprüfen, ob ein gegebenes Programm bei jeder Eingabe terminiert.
- Es ist (im Allgemeinen) unmöglich mechanisch zu überprüfen, ob zwei gegebene Programme dieselbe Funktionalität haben.

Entscheidungsverfahren: Wenn für Sprache A ein While Programm existiert (immer terminierend).

Semi-Entscheidungsverfahren: Wenn für Sprache A ein While Programm existiert und für Wörter nicht in A <u>nicht terminiert</u>.

Folgende Aussagen für $A \subset \Sigma^*$ sind äquivalent:

- *A ist rekursiv aufzählbar.* → Hält nur bei Ja an!
- *A ist semi-entscheidbar[1].* = ⇔ rekursive Sprachen sind hingegen entscheidbar.
- *A ist der Wertebereich einer totalen berechenbaren Funktion.*
- *A ist der Definitionsbereich einer berechenbaren Funktion.*

⇨ *(KFG sind entscheidbar.)*

> Bemerkungen:

- Jede entscheidbare Sprache ist auch semi-entscheidbar.
- Sprache A entscheidbar wenn A und Komplement von A semi-entscheidbar.
- Sprache A entscheidbar, dann auch Komplement von A entscheidbar. (Semi-entscheidbare nicht unbedingt.)
- Alle rekursiven Sprachen sind rekursiv aufzählbar.
- Primitiv rekursive Sprachen sind eine Teilmenge der rekursiven Sprachen.

> Reduktion (Analogie:Code wiederverwendbar):

Eine Sprache $A \subset \Sigma^*$ heisst auf eine Sprache $B \subset \Gamma^*$ **reduzierbar**, wenn es eine totale, Turing-berechenbare Funktion $F : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$ Reduktion $\hat{=}$ "Übersetzung"

$$w \in A \iff F(w) \in B$$

gilt. Ist die Sprache A auf die Sprache B reduzierbar, dann schreiben wir $A \preceq B$. ⇒ B "schwieriger"

⇨ Entscheidbarkeit von B gleich wie A.

- Ablauf: Man verändert die Eingabe von A bevor wir sie in Algorithmus von B Input. <u>Jedes Ja in A bildet Ja in B ab</u>. Nein analog

> Halteproblem:

Das **allgemeine Halteproblem** ist die Sprache
⇒ Hält TM w auf Input x?

$$H := \{w\#x \in \{0,1,\#\}^* \mid T_w \text{ angesetzt auf } x \text{ hält}\}.$$

Das **leere Halteproblem** ist die Sprache
⇒ hält w auf leerem Band an?

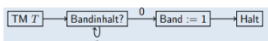
$$H_0 := \{w \in \{0,1\}^* \mid T_w \text{ angesetzt auf das leere Band hält}\}.$$

Das **spezielle Halteproblem** ist die Sprache
⇒ hält w auf sich selber an?

$$H_S := \{w \in \{0,1\}^* \mid T_w \text{ angesetzt auf } w \text{ hält}\}.$$

Halteproblem Eigenschaften

- $H_s \preccurlyeq H \preccurlyeq H_0$
- Alle sind semi-entscheidbar.
- Alle sind unentscheidbar: Beweis langt für Hs:
  - ⇨ Annahme: TM T entscheidet Hs.
  - ⇨ Neue TM P: führt zuerst T aus und dann Output negieren:



  - ⇨ Wenn nun der Entscheidungsalgorithmus Ja sagt (P hält an) hält P nicht an und vice versa.

Komplexitätstheorie

> O-Notation für Zeitkomplexität:

Laufzeit des besten Programms, welches das Problem löst.

Obere Schranke $\mathcal{O}$: Ein Algorithmus für Prüfung

Untere SchrankeΩ: Alle Algorithmen für Prüfung

$$\mathcal{O}(1) \subset \mathcal{O}(loglogn) \subset \mathcal{O}(logn) \subset \mathcal{O}(\sqrt[c]{logn})$$
$$\subset \mathcal{O}(\sqrt[c]{n}) \subset \mathcal{O}(n) \subset \mathcal{O}(n\ logn) = \mathcal{O}(logn!)$$
$$\subset \mathcal{O}(n^c) \subset \mathcal{O}(c^n) \subset \mathcal{O}(n!)$$

- $\mathcal{O}$: Effizientere TM existiert.
- Ω: Keine effizientere TM existiert

> P- und NP-Klasse:

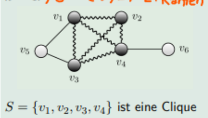
P := Alle polynomzeit endscheidbaren Sprachen (= «effizient» lösbare Probleme).

NP := Alle polynomzeit endscheidbaren Sprachen mittels einer NTM.

> CLIQUE-Problem:

Beispiel (CLIQUE-Problem)

$k = 4$: ; $G = (V, E)$ V: Knoten E: Kanten



$S = \{v_1, v_2, v_3, v_4\}$ ist eine Clique

Zur Bestimmung einer Clique in einem Graphen sind nur **exponentielle** Verfahren bekannt.

Wenn aber ein *Lösungskandidat* bekannt ist, kann in **polynomieller Zeit überprüft werden**, ob dies tatsächlich eine Lösung ist.

> Polynomzeit-Verifizierer (Alternative NP Def):

Sei $L \subseteq \Sigma^*$ eine Sprache und p: $\mathbb{N} \to \mathbb{N}$ eine Funktion. Eine TM M ist ein **p-Verifizierer** für L, falls M wie folgt auf allen Eingaben $w\#x$ für $w \in \Sigma^*$ und $x \in \{0,1\}^*$ arbeitet:

- $\text{Time}_M(w\#x) \leq p(|w|)$ für alle Eingaben $w\#x$.
- Für jedes $w \in L$ existiert ein $x \in \{0,1\}^*$ mit $|x| \leq p(|w|)$, so dass M die Eingabe $w\#x$ akzeptiert.
  x heisst **Zeuge** für $w \in L$.
- Für alle $w \notin L$ existiert kein Zeuge.

⇒ Fall $p \in \mathcal{O}(n^k)$, $k \in \mathbb{N}$, dann M ein Polynomzeit-V.

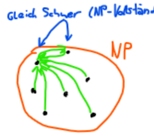
> NP-Schwer, -vollständig, Polyn. Reduktion:

$L_1 \preceq_p L_2$ bedeutet, dass L_2 mindestens so schwer ist in Bezug auf die Lösbarkeit in polynomieller Zeit wie L_1. = Polynomielle Reduktion

NP-Schwer: Wenn alle Sprachen / Probleme in NP auf dieses in polynomieller Zeit reduzierbar sind.

NP-Vollständig: Falls in NP und NP-Schwer.

Gleich Schwer (NP-Vollständig)



Alle Probleme in P Gleichschwer

Wenn P1 NP-Schwer und P2 in NP, dann ist, bei polynomieller Reduktion von P1 auf P2, P2 NP-vollständig.

Rechenregeln

- Konstante Vorfaktoren kann man ignorieren: $c \cdot f(n) \in \mathcal{O}(f(n))$
- Für eine Konstante c gilt: $c \in \mathcal{O}(1)$
- Bei Polynomen ist nur die höchste Potenz entscheidend: $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in \mathcal{O}(n^k)$
- Die $\mathcal{O}$-Notation ist transitiv: Aus $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n))$ folgt $f(n) \in \mathcal{O}(h(n))$

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.

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.

⟹ Rechenzeit in $\mathcal{O}(n^2)$.