

Alphabete, Wörter, Sprachen

Alphabete endliche, nichtleere Mengen von Symbolen.

- $\Sigma_{\text{Bool}} = \{0, 1\}$ Boolesches Alphabet

Keine Alphabete: $\mathbb{N}, \mathbb{R}, \mathbb{Z}$ usw. (unendliche Mächtigkeit)

Wort endliche Folge von Symbolen eines bestimmten Alphabets.

Schreibweisen $|\omega|$ = Länge eines Wortes

$|\omega|_x$ = Häufigkeit eines Symbols x in einem Wort

ω^R = Spiegelwort/Reflection zu ω

Teilwort (Infix) v ist ein Teilwort (Infix) von ω ist, wenn $\omega = xvy$.

$\omega \neq v \rightarrow$ Echtes Teilwort, Präfix = Anfang, Suffix = Ende

Mengen von Wörtern Σ^k = Wörter der Länge k über Alphabet Σ

- $\Sigma^* = \underbrace{\Sigma^0}_{1} \cup \underbrace{\Sigma^1}_{1} \cup \underbrace{\Sigma^2}_{2} \cup \underbrace{\Sigma^3}_{4} \dots$ Kleensche Hülle

- $\Sigma^+ = \underbrace{\Sigma^1}_{2} \cup \underbrace{\Sigma^2}_{4} \cup \underbrace{\Sigma^3}_{8} \dots = \Sigma^* \setminus \{\varepsilon\}$ Positive Hülle

ε Leeres Wort (über jedem Alphabet) $\Sigma^0 = \{\varepsilon\}$

Konkatenation = Verkettung von zwei beliebigen Wörtern x und y
 $x \circ y = xy := (x_1, x_2 \dots x_n, y_1, y_2 \dots y_m)$

Wortpotenzen Sei x ein Wort über einem Alphabet Σ

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

Sprache über Alphabet Σ = Teilmenge $L \subseteq \Sigma^*$ von Wörtern

- $\Sigma_1 \subseteq \Sigma_2 \wedge L$ Sprache über $\Sigma_1 \rightarrow L$ Sprache über Σ_2
- Σ^* Sprache über jedem Alphabet Σ
- $\{\} = \emptyset$ ist die leere Sprache

Kleensche Hülle A^* von A : $\{\varepsilon\} \cup A \cup AA \cup AAA \cup \dots$

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

$w = uv \in \Sigma_A \cup \Sigma_B$ NUR wenn $u \in \Sigma_A$ und $v \in \Sigma_B$!!

Reguläre Ausdrücke Wörter, die Sprachen beschreiben (Regex)

RA_Σ Sprache der Regulären Ausdrücke über $\{\emptyset, \epsilon, *, (), , | \} \cup \Sigma$

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

Priorisierung von Operatoren

(1) $*$ = Wiederholung \rightarrow (2) Konkatenation \rightarrow (3) $|$ = Oder

Erweiterter Syntax

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

Reguläre Sprache A über dem Alphabet Σ heisst regulär, falls

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

$\forall R \in RA_\Sigma$ definieren wir die Sprache $L(R)$ von R wie folgt:

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\} \quad \forall a \in \Sigma$
- $L(R^*) = L(R)^*$
- $L((R^*)^*) = L(R^*)$
- $L(RS) = L(R)L(S)$
- $L(R \mid S) = L(S \mid R)$
- $L(R(ST)) = L((RS)T) = L(RS)$
- $L(R \mid (S \mid T)) = L((R \mid S) \mid T)$
- $L(R(S \mid T)) = L(RS \mid RT)$
- $L(R \mid R) = L(R) = L(R \mid \emptyset)$
- $L(R \mid S) = L(R) \cup L(S)$

Endliche Automaten

Endliche Automaten Maschinen, die Entscheidungsprobleme lösen

- Links nach rechts
- Keinen Speicher
- Keine Variablen
- Speichert aktuellen Zustand
- Ausgabe über akzeptierende Zustände

DEA deterministischer endlicher Automat: $M = (Q, \Sigma, \delta, q_0, F)$

Q : endliche Menge von Zuständen $q_0 \in Q$ Startzustand

Σ : endliches Eingabealphabet $F \subseteq Q$ Menge der

$\delta: Q \times \Sigma \rightarrow Q$ Übergangsfunktion akzeptierenden Zustände

DEA Funktionen $M = (Q, \Sigma, \delta, q_0, F) : \text{EA}$

Konfiguration M auf $\omega \in Q \times \Sigma^*$ (Start: (q_0, ω) , End: (q_n, ε))

Berechnungsschritt \vdash_M von M : $(q, \omega) \vdash_M (p, x)$

Berechnung:

$(q_a, \omega_1 \dots \omega_n) \vdash_M \dots \vdash_M (q_e, \omega_j \dots \omega_n) \rightarrow (q_a, \omega_1 \dots \omega_n) \vdash_M^* (q_e, \omega_j \dots \omega_n)$

Beispiel DEA (eindeutig) Sprache: $L(M) = \{1x1 \mid x \in \{0\}^*\}$



Berechnung

$\omega = 101 \rightarrow (q_0, 101) \vdash_M (q_1, 01) \vdash_M (q_1, 1) \vdash_M (q_2, \varepsilon) \rightarrow$ akzeptierend

$\omega = 10 \rightarrow (q_0, 10) \vdash_M (q_1, 0) \vdash_M (q_1, \varepsilon) \rightarrow$ verwerfend

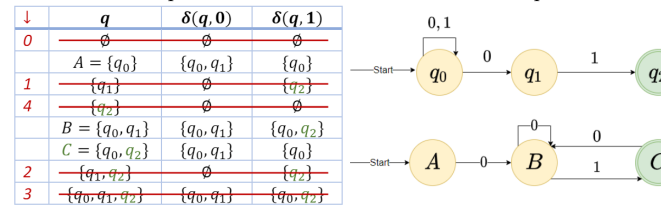
Nichtdeterministischer endlicher Automat (NEA)

Unterschied zum DEA: Übergangsfunktion $\delta: Q \times \Sigma \rightarrow P(Q)$

Ein ε -NEA erlaubt zusätzlich noch ε -Übergänge

Teilmengenkonstruktion \forall NEA kann in DEA umgewandelt werden

- $Q_{NEA} \rightarrow P(Q_{NEA}) = Q_{DEA}$ (Potenzmenge)
- Verbinden mit Vereinigung aller möglichen Zielzustände
- Nicht erreichbare Zustände eliminieren
- Enthält akzeptierenden Zustand = $F_{NEA} \rightarrow$ akzeptierend



Zustandsklasse $\Sigma^* = \bigcup_{p \in Q} [p]$ $[p] \cap [q] = \emptyset, \forall p \neq q, p, q \in Q$

Jedes Wort landet in einem Zustand, aber kein Wort landet nach dem Lesen in zwei Zuständen!

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

- Jeder EA für $L(M_9)$ muss die Anzahl der gelesenen Nullen modulo 3 zählen und unterscheiden können.
- Zum Beispiel: $x_1 = \varepsilon, x_2 = 0, x_3 = 00$
- Widerspruch für alle Paare von Wörtern aufzeigen:

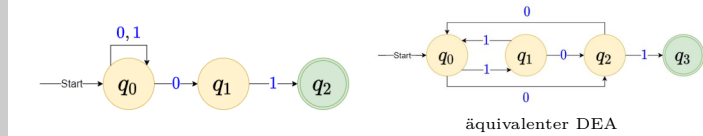
Für x_1 und x_2 : $z_{12} = \varepsilon \Rightarrow x_1 z_{12} = \varepsilon \notin L, x_2 z_{12} = 0 \in L$

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 : $z_{23} = \varepsilon \Rightarrow x_2 z_{23} = 0 \in L, x_3 z_{23} = 00 \notin L$

Jeder EA für $L(M_9)$ muss zwischen mindestens drei Zuständen unterscheiden. Der EA hat mind. 3 Zustände.

NEA (nicht eindeutig) Sprache: $L(M) = \{x01 \mid x \in \{0, 1\}^*\}$



Reguläre Sprachen und endliche Automaten

Reguläre Sprachen durch äquivalente Mechanismen beschreibbar

- Akzeptierender Mechanismus DEA, NEA, ε -NEA
- Beschreibender Mechanismus RA

Eigenschaften Seien L, L_1 und L_2 reguläre Sprachen über Σ

- Vereinigung: $L_1 \cup L_2 = \{\omega \mid \omega \in L_1 \vee \omega \in L_2\}$
- Schnitt: $L_1 \cap L_2 = \{\omega \mid \omega \in L_1 \wedge \omega \in L_2\}$
- Differenz: $L_1 - L_2 = \{\omega \mid \omega \in L_1 \wedge \omega \notin L_2\}$
- Komplement: $\bar{L} = \Sigma^* - L = \{\omega \in \Sigma^* \mid \omega \notin L\}$
- Konkatenation: $L_1 \cdot L_2 = L_1 L_2 = \{\omega = \omega_1 \omega_2 \mid \omega_1 \in L_1 \wedge \omega_2 \in L_2\}$
- Kleenesche Hülle: $L^* = \{\omega = \omega_1 \omega_2 \dots \omega_n \mid \omega_i \in L \text{ für alle } i \in \{1, 2, \dots, n\}\}$

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

- $((- \mid \varepsilon)(1, 2, 3, 4, 5, 6, 7, 8, 9)(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \mid 0).0$

Kontextfreie Grammatiken

Kontextfreie Grammatik (KFG) ist ein 4-Tupel (N, Σ, P, A) mit

- N : Alphabet der Nichtterminale (Variablen)
- Σ : Alphabet der Terminale
- P : endliche Menge von Produktionen mit der Form $X \rightarrow \beta$ Mit Kopf $X \in N$ und Rumpf $\beta \in (N \cup \Sigma)^*$
- A : Startsymbol, wobei $A \in N$

Ein Wort $\beta \in (N \cup \Sigma)^*$ nennen wir Satzform.

Seien α, β und γ Satzformen und $A \rightarrow \gamma$ eine Produktion.

- Ableitungsschritt mit Produktion $A \rightarrow \gamma$ $\alpha A \beta \rightarrow \alpha \gamma \beta$
- Ableitung Folge von Ableitungsschritten $\alpha \rightarrow \dots \rightarrow \omega$

Ableitungsbaum (Parsebaum)

KGF für die Sprache $L = \{0^n 1^m \mid n, m \in \mathbb{N}\}$

- $G_1 = \{\{A, B, C\}, \{0, 1\}, P, A\}$
- $P = \{A \rightarrow BC, B \rightarrow 0B \mid 0 \mid \varepsilon, C \rightarrow 1C \mid 1 \mid \varepsilon\}$

Ableitung von $\omega_1 = 011$:

$A \rightarrow BC \rightarrow 0C \rightarrow 01C \rightarrow 011$

Mehrdeutige KFG \exists Wort, das mehrere Ableitungsbäume besitzt. Mehrdeutigkeiten eliminieren: Korrekte Klammerung vom Benutzer erzwingen, Grammatik anpassen, Produktionen Vorrang geben

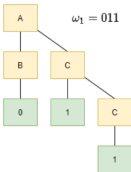
Reguläre Sprache durch KFG beschreiben $\forall L \exists \text{KFG}$

L reguläre Sprache $\Rightarrow \exists \text{DEA } M = (Q, \Sigma, \delta, q_0, F)$ mit $L(M) = L$

KFG für L bauen:

- \forall Zustand q_i gibt es ein Nichtterminal Q_i
- \forall Transition $\delta(q_i, a) = q_j$ erstelle Produktion $Q_i \rightarrow aQ_j$
- \forall akzeptierenden Zustand $q_i \in F$ erstelle Produktion $Q_i \rightarrow \varepsilon$
- Nichtterminal Q_0 wird zum Startsymbol A .

ACHTUNG: es dürfen keine Wörter, die nicht in L sind, abgeleitet werden können



KFG für die Sprache $L = \{0^n 1^n \mid n \in \mathbb{N}\}$

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

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

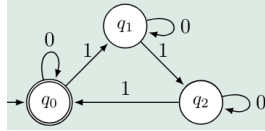
KFG für die Sprache $L_5 = \{w \in \{0, 1\}^* \mid |w|_1 \bmod 3 = 0\}$

Nichtterminale: Q_0, Q_1, Q_2

$Q_0 \rightarrow 0Q_0 \mid 1Q_1 \mid \varepsilon$

Produktionen: $Q_1 \rightarrow 0Q_1 \mid 1Q_2$

$Q_2 \rightarrow 0Q_2 \mid 1Q_0$



Beispiel für Ableitung von $w = 10011$: $Q_0 \Rightarrow 1Q_1 \Rightarrow 10Q_1 \Rightarrow 100Q_1 \Rightarrow 1001Q_2 \Rightarrow 10011Q_0 \Rightarrow 10011$

Kellerautomaten

KA auf englisch: PDA = Push Down Automat

Kellerautomaten (KA) besitzt «Speicher»

Deterministischer KA (DKA): $M = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$

Q : Menge von Zuständen $q_0 \in Q$: Anfangszustand

Σ : Alphabet der Eingabe $\$ \in \Gamma$: Symbol vom Alphabet des Kellers

Γ : Alphabet des Kellers $F \subseteq Q$: Akzeptierende Zustände

Übergangsfunktion: $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

NKA: $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow P(Q \times \Gamma^*)$ (Nichtdeterministischer KA)

Übergangsfunktion KA \forall Zustand q und \forall Symbole x, b gilt: wenn $\delta(q, b, c)$ definiert ist, dann ist $\delta(q, \varepsilon, x)$ undefiniert.

Darstellung Übergang $\delta(q, b, c) = (p, \omega)$: $q - b, c/\omega \longrightarrow p$

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

q = Aktueller Zustand ω = aktueller Stack + ω (push)

b = gelesene Eingabe neustes Symbol zuerst

c = entfernt von Stack (pop) p = Neuer Zustand

(q, b, c) wird als Konfiguration bezeichnet

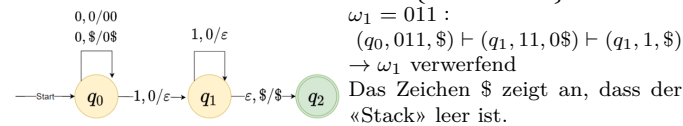
Sprache $L(M)$ eines Kellerautomaten M ist definiert durch

$L(M) = \left\{ \omega \in \Sigma^* \mid (q_0, \omega, \$) \vdash^* (q, \varepsilon, \gamma) \text{ für ein } q \in F \text{ und ein } \gamma \in \Gamma^* \right\}$

Elemente von $L(M)$ werden von M akzeptierte Wörter genannt.

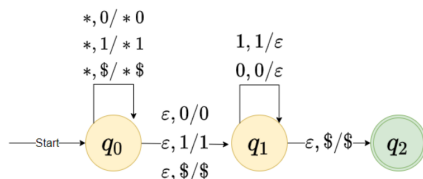
Damit w_x akzeptiert wird, muss $b = \varepsilon$ sein (Stack muss nicht leer sein)

Kellerautomat für kontextfreie Sprache $\{0^n 1^n \mid n > 0\}$



Eine Sprache ist kontextfrei, wenn sie von einem NKA akzeptiert wird (nicht unbedingt von DKA). Wenn von DKA erkannt, dann ist die Sprache eindeutig.

NKA $\{\omega \omega^R \mid \omega \in \{0, 1\}^*\}$



NKA/DKA erkennbar? $L_1 = \{0^n 1^n \mid n \in \mathbb{N}\}$, $\sigma = \{0, 1\} \rightarrow$ nein

$L_2 = \{waw^R \mid w \in \{0, 1\}^*\}$, $\sigma = \{0, 1, a\} \rightarrow$ ja (DKA)

$L_3 = \{ww \mid w \in \{0, 1\}^*\}$, $\sigma = \{0, 1\} \rightarrow$ nein

Turingmaschinen

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

Q : Menge von Zuständen $q_0 \in Q$: Anfangszustand

Σ : Alphabet der Eingabe $F \subseteq Q$: Akzeptierende Zustände

Γ und $\Sigma \subset \Gamma$: Bandalphabet \sqcup : Leerzeichen mit $\mu \in \Gamma$ und $\mu \notin \Sigma$

Übergangsfunktion: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times D, D = \{L, R\}$

Sie bestehen aus einem Lese-/Schreibkopf und einem unendlichen Band von Zellen.

Sie bildet das 2-Tupel (q, X) auf das Tripel (p, Y, D)

D = Direction

$q, p \in Q$ und $X, Y \in \Gamma$

X = Read

$q - X/Y, D \rightarrow p$

Y = Overwrite

Wenn TM anhält, dann fertig (akzeptierend falls in F). Resultat = Bandinhalt

Band Zellen mit je 1 Symbol, Anfangszustand: enthält Eingabe (endliches Wort aus Σ^*), alle anderen Zellen: \sqcup

Konfiguration einer TM Zustand der Zustandssteuerung, Position des Lese-/Schreibkopfes und Bandinhalt

Universelle TM Codierung der Übergangsfunktion (Gödelnummer)

Zustände $q_n := 0^n$

Symbole $\Gamma = \{0, 1, \$, a \in \Sigma\}$

($0 = 0, 1 = 00, \$ = 000$)

Richtung L = 0, R = 00

Trennzeichen

- zwischen Elementen: 1
- zwischen Übergangsfunktionen: 11
- Ende der Turingmaschine: 111 danach **Input**

bsp: 010010010100000001...0111111101...

Alle Arten von TMs gleichmächtig, da jede TM durch eine DTM simuliert werden kann.

Mehrband-Maschine

Spezifizieren Sie eine TM

M_4 , welche die Subtraktion

von zwei natürlichen Zahlen

($a - b$, mit $a \geq b$) realisiert.

Beispiel: $4 - 2 = 2$

		1	2	3	4	5	6	7	8	9
1	$q_0 0000100 \vdash$									
2	$q_0 \sqcup \vdash$									
1	$\sqcup q_0 000100 \vdash$									
2	$0q_0 \sqcup \vdash$									
1	$\sqcup \sqcup q_0 00100 \vdash$									
2	$00q_0 \sqcup \vdash$									
1	$\sqcup \sqcup \sqcup q_0 0100 \vdash$									
2	$000q_0 \sqcup \vdash$									
1	$\sqcup \sqcup \sqcup \sqcup q_0 100 \vdash$									
2	$0000q_0 \sqcup \vdash$									
1	$\sqcup \sqcup \sqcup \sqcup \sqcup q_1 0 \vdash$									
2	$000q_1 0 \vdash$									
1	$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup q_1 0 \vdash$									
2	$00q_1 0 \vdash$									
1	$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup q_1 0 \vdash$									
2	$0q_1 0 \vdash$									
1	$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup q_2 \sqcup \vdash$									
2	$00q_2 \sqcup \vdash$									

Berechnungsmodelle

Turing-berechenbar = kann von Turing-Maschine gelöst werden

Turing-berechenbare Funktion $T : \Sigma^* \rightarrow \delta^*$

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

\forall algorithmisch lösbare Problem ist turing-berechenbar \Rightarrow Computer \equiv TM

Primitiv rekursive Grundfunktionen $\forall n \in \mathbb{N}, \forall k \in \mathbb{N}$ (k = Konstante)

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 $\eta(x) = x + 1$

n -stellige Projektion auf die k -te Komponente:

$\pi_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$ mit $\pi_k^n(x_1, \dots, x_k, \dots, x_n) = x_k$ ($1 < k < n$)

n = Anzahl der Argumente, k = Position des Arguments

Primitive Rekursion von 2 Funktionen HEKLP

Primitiv rekursive Funktionen

- set zero: $c_0^1(x) = 0$
- $c_5^4 : \mathbb{N}^4 \rightarrow \mathbb{N}$ mit $c_5^4(x_1, x_2, x_3, x_4) = 5$
- add(0, x): $\pi_1^1(x) = x$
- $\pi_1^3 : \mathbb{N}^3 \rightarrow \mathbb{N}$ mit $\pi_1^3(x_1, x_2, x_3) = x_1$
- add(1, x): $\eta(x) = x + 1$
- $\pi_1^1 : \mathbb{N} \rightarrow \mathbb{N}$ mit $\pi_1^1(x) = x$
- add(2, x):
- $\pi_5^5 : \mathbb{N}^5 \rightarrow \mathbb{N}$ mit $\pi_5^5(x_1, x_2, x_3, x_4, x_5) = x_5$
- $\eta(\eta(x)) = x + 2$

Vorgängerfunktion: $p(x) = 0$ falls $x = 0$; $x - 1$ sonst

Addition: $add(x, y) = y$ falls $x = 0$; $add(x - 1, y) + 1$ sonst

$add(0, y) = \pi_2^2(y) = y$

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

Subtraktion: $sub(x, y) = 0$ falls $y = 0$; $sub(x - 1, y - 1)$ sonst

Multiplikation: $mult(x, y) = 0$ falls $y = 0$; $mult(x, y - 1) + x$ sonst

$mul(0, y) = 0$

$mul(x + 1, y) = add(mul(x, y), \pi_2^2(x, y)) = add(mul(x, y), y)$

Potenz: $exp(x, y) = 1$ falls $y = 0$; $mult(exp(x, y - 1), x)$ sonst

LOOP, WHILE, GOTO

Zuweisung: $xi = xj + c$ oder $xi = xj - c$

Return Wert = x_0

Nicht gleichmächtig wie TM

Variablen: x_0, x_1, x_2, \dots

Konstanten: $0, 1, 2, \dots$

Operationszeichen: $+$, $-$

Trennzeichen: $;$

Loop (primitiv-rekursiv)

Schlüsselwörter: Loop, Do, End

Sequenzen: P und $Q \rightarrow P; Q$

Schleifen: Loop x do $P \dots$ End

$add(x_1, x_2) = x_0$

Loop x_1 **Do**

$x_2 = x_2 + 1$

End;

$x_0 = x_2 + 0$

While (Turing vollständig)

Erweiterung der Sprache Loop

While $xi > 0$ Do ... End

$mul(x_1, x_2) = x_0$

While $x_1 > 0$ **Do**

$x_1 = x_1 - 1;$

Loop x_2 **Do**

$x_0 = x_0 + 1$

End

GoTo (Turing vollständig)

Schlüsselwörter: Goto, If, Else

Marker Mk: M1, M2, ...

Sprunganweisung:

If $xi = c$ Then Goto Mk

case distinction (x_1, x_2, x_3) = x_0

M1: $x_0 = x_3 + 0;$

M2: **If** $x_1 = 0$ **Then Goto** M4;

M3: $x_0 = x_2 + 0;$

M4: **Halt**

Maximum $\max(x1, x2)$

```

1 x0 = x2 + 0;
2 x3 = x1 - x2;
3 Loop x3 Do
4   x0 = x1 + 0
5 End

```

Addition $x1 + x2$

```

1 While x1 > 0 Do
2   x2 = x2 + 1;
3   x1 = x1 - 1
4 End;
5 x0 = x2 + 0

```

Absolute Difference $|x1 - x2|$

```

1 x4 = x2 + 0;
2 x3 = x1 - x2;
3 Loop x3 Do
4   x4 = x1 + 0
5 End;
6 x5 = x2 + 0;
7 x3 = x2 - x1;
8 Loop x3 Do
9   x5 = x1 + 0
10 End;
11 x0 = x4 - x5

```

Unvollständigkeit**Ackermannfunktion** $a: \mathbb{N}^2 \rightarrow \mathbb{N}$

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

wächst schneller als jede primitiv rekursive Funktion
 total (überall definiert)
 \Rightarrow nicht primitiv rekursiv
 aber: turing-berechenbar

Ackermannfunktion berechne $a(2, 1)$

$a(2, 1) = a(1, a(2, 0)) = a(1, a(1, 1)) = a(1, a(0, a(1, 0)))$
 $= a(1, a(0, a(0, 1))) = a(1, a(0, 2)) = a(1, 3) = a(0, a(1, 2))$
 $= a(0, a(0, a(1, 1))) = a(0, a(0, a(0, a(1, 0))))$
 $= a(0, a(0, a(0, a(0, 1)))) = a(0, a(0, a(0, 2)))$
 $= a(0, a(0, 3)) = a(0, 4) = 5$

Beweis dass Ackermannfunktion nicht primitiv rekursivDefiniere zu jeder primitiv rekursiven Funktion f eine Funktion $m(n, f) = \max \{f(\vec{x}) \mid \sum \vec{x} \leq n\}$

Zeige, dass $\exists k \in \mathbb{N}$ für jede primitiv rekursive Funktion f , so dass $\forall n \geq k \ m(n, f) < a(k, n)$ gilt

Definiere $a_1(x) = a(x, x)$. Wäre a_1 primitiv rekursiv, dann gäbe es ein k mit $m(n, a_1) < a(k, n)$ für $n \geq k$

Insbesondere gilt also für $n = k$:

$$\max \{a_1(x) \mid x \leq k\} = m(k, a_1) < a(k, k) = a_1(k).$$

Also kann a_1 und somit a nicht primitiv rekursiv sein.**LOOP-Interpreter** nicht LOOP-berechenbar, aber turing-berechenbar

Sei x Code eines Programms P : für jeden Input y soll $I(x, y) = P(y)$ gelten wenn $P(y)$ definiert

Minimum $\min(x1, x2)$

```

1 x0 = x2 + 0;
2 x3 = x2 - x1;
3 Loop x3 Do
4   x0 = x1 + 0
5 End

```

End**Division** $x1 / x2 + \text{remainder}$

```

1 x3 = x1 - x2;
2 x3 = x3 + 1;
3 While x3 > 0 Do
4   x3 = x3 - x2;
5   x0 = x0 + 1
6 End

```

End**Fibonacci** $\text{fib}(x1)$

```

1 x2 = 0 + 0;
2 x0 = 1 + 0;
3 While x1 > 0 Do
4   x3 = x0 + 0;
5   x0 = x0 + x2;
6   x2 = x3 + 0;
7   x1 = x1 - 1
8 End

```

End**Entscheidbarkeit****Entscheidbar** \exists Algorithmus, der \forall Eingabe eine Antwort liefert**Semi-entscheidbar:** \exists Algorithmus, der \forall Eingabe eine Antwort liefert, falls Antwort die Antwort «Ja» ist**Entscheidbarkeit** einer Sprache $A \subseteq \Sigma^*$ $A \subseteq \Sigma^*$ ist entscheidbar $\Leftrightarrow A$ und \bar{A} sind semi-entscheidbar A entscheidbar $\Leftrightarrow \bar{A}$ entscheidbar $\bar{A} = \Sigma^* \setminus A = \{\omega \in \Sigma^* \mid \omega \notin A\}$ (Komplement von A)**Entscheidbarkeit mit Turingmaschinen** $A \subseteq \Sigma^*$ 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: $A \subseteq \Sigma^*$ 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 $A \subseteq \Sigma^*$ 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: $A \subseteq \Sigma^*$ ist semi-entscheidbar

- $A \subseteq \Sigma^*$ 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 $A \leq B \Rightarrow A \subseteq \Sigma^*$ reduzierbar auf $B \subseteq \Gamma^*$ Gilt wenn $\exists T: \Sigma^* \rightarrow \Gamma^*$ so dass: $\forall \omega \in \Sigma^* \quad \omega \in A \Leftrightarrow F(\omega) \in B$ $T :=$ total Turing-berechenbare FunktionTransitiv: $A \leq B$ und $B \leq C \Rightarrow A \leq C$ $A \leq B \Rightarrow$ Entscheidbarkeit von B gleich wie von A Zeige dass $A \leq B$ $P_1(x) := x$ Primzahl? $P_2(x, y) := x$ kleinster Primfaktor von y ? $P_1(x) \leq P_2(x, y)$ mit $F(x) := P_2(x, x)$ **Halteproblem** Ist es möglich einen Algorithmus zu schreiben, der für jede TM entscheiden kann, ob sie hält oder nicht? (Nein!)Halteprobleme (HP) definiert als Sprachen: ($\#$ = Delimiter)**Allgemeines** $H := \{\omega \# x \in \{0, 1, \#\}^* \mid T_\omega \text{ angesetzt auf } x \text{ hält}\}$ **Leeres** $H_0 := \{\omega \in \{0, 1\}^* \mid T_\omega \text{ angesetzt auf das leere Band hält}\}$ **Spezielles** $H_S := \{\omega \in \{0, 1\}^* \mid T_\omega \text{ angesetzt auf } \omega \text{ hält}\}$ $H_0 \leq H_S \leq H$ H_0, H_S und H sind semi-entscheidbar und nicht entscheidbar.

H_S ist unentscheidbar Beweis durch Widerspruch: Wir wissen, dass H unentscheidbar ist. Angenommen H_S wäre entscheidbar. Dann wäre auch H entscheidbar, was ein Widerspruch ist.

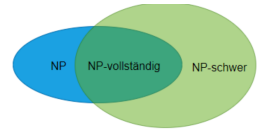
Mächtigkeit $\text{NEA} = \text{DEA} = \text{Reguläre Sprache (z.B. } 1^n 0^m) < \text{NKA} < \text{DKA} = \text{KFG} = \text{Kontextfreie Sprache (z.B. } 1^n 0^m) < \text{NTM} = \text{DTM} = \text{Rekursiv abzählbare Sprachen (z.B. } a^n b^n c^n)$

primitiv rekursive Funktionen $<$ Turing-berechenbare Funktionen**Komplexitätstheorie****Quantitative Gesetze und Grenzen**

- Zeitkomplexität: Laufzeit des besten Programms
- Platzkomplexität: Speicherplatz des besten Programms
- Beschreibungskomplexität: Länge des kürzesten Programms

Zeitbedarf von M auf Eingaben der Länge $n \in \mathbb{N}$ im schlechtesten Fall: $\text{Time}_M(n) = \max \{ \text{Time}_M(\omega) \mid |\omega| = n \}$ Sei M eine TM, die immer hält und sei die Eingabe $\omega \in \Sigma^*$ Zeitbedarf von M auf ω : $\text{Time}_M(\omega) = \text{Anzahl Konfigurationsübergänge in der Berechnung von } M \text{ auf } \omega$ **P vs NP** Klassifizierung von ProblemenEin Problem U heisst in Polynomzeit lösbar, wenn es eine obere Schranke $O(n^c)$ gibt für eine Konstante $c \geq 1$.

- $P \doteq$ Lösung finden in Polynomzeit
- $NP \doteq$ Lösung verifizieren in Polynomzeit

ACHTUNG: NP heisst nicht «nicht-polynomial» sondern «nicht-deterministisch polynomial»**NP-schwer und NP-vollständig**Eine Sprache L heisst NP-schwer, fallsfür alle Sprachen $L' \in NP$ gilt,dass $L' \leq_p L$ Eine Sprache L heisst NP-vollständig, falls $L \in NP$ und L ist NP-schwer. $P \subset NP$ gilt, aber $P \neq NP$ noch nicht bewiesen**Polynomzeit-Verifizierer** Überprüft die Eingaben in einem Problem Zeuge: Informationen einer gültigen Eingabe**Asymptotische Komplexitätsmessung** O -Notation

- $f \in O(g)$: $f(n) \leq c \cdot g(n)$
– f wächst asymptotisch nicht schneller als g
- $f \in \Omega(g)$: $f(n) \geq \frac{1}{d} \cdot g(n)$
– f wächst asymptotisch mindestens so schnell wie g
- $f \in \Theta(g)$: Es gilt $f(n) \in O(g(n))$ und $f(n) \in \Omega(g(n))$
– f und g sind asymptotisch gleich

Schranken für die Zeitkomplexität von U

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

Rechenregeln

- Konstante Vorfaktoren c ignorieren ($c \in O(1)$)
- Bei Polynomen ist nur die höchste Potenz entscheidend
- Transitiv: $f(n) \in O(g(n)) \wedge g(n) \in O(h(n)) \rightarrow f(n) \in O(h(n))$

Übersicht wichtigste Laufzeiten

$O(1) < O(\log(\log n)) < O(\sqrt{\log n}) < O(\log \sqrt{n}) < O(\log n)$
 $< O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3)$
 $< O(n^c) < O(2^n) < O(n!) < O(n^n)$

- Multiple-Choice** ✓ = richtig, ☒ = falsch
- ☒ Konkatenation von Sprachen = Vereinigung der zugrundeliegenden Alphabete
 - ☒ $L = \{\} \equiv K = \{\epsilon\}$
 - ☒ $\epsilon \in AB$ oder $\epsilon 100 \in AB$ für A enthält alle Binärwörter die mit 0 enden, B alle die mit 1 enden

- Reguläre Ausdrücke und Sprachen**
- ϵ = Sprache aller Binärzahlen Länge 0
 - $(1(0 \mid 1)^* 1) \mid 1$ = Sprache aller positiver ungerader Binärzahlen
 - $(0 \mid 0001)^*$ = Sprache aller Wörter die vor jeder 1 mind. 3 Nullen haben
 - $(0 \mid 1)^* 0(0 \mid 1)^* 0(0 \mid 1)^* 0(0 \mid 1)^*$ = Sprache aller Binärzahlen die mind. 4 Nullen enthalten

- regulär/nicht regulär:
- $\{(it)^n \mid n \in \mathbb{N}\} \rightarrow$ regulär
 - $\{w \in a, b^* \mid a^m b^n \text{ mit } m \neq n\} \rightarrow$ nicht regulär
 - Windows Dateinamen \rightarrow regulär
 - Sprache der Wörter mit doppelt so vielen 0 wie 1 \rightarrow nicht regulär

Zustandsklassen EA akzeptiert Binärwörter wo $|0|$ und $|1|$ gerade

Klasse $[q_0] = \{w \in \{0,1\}^* \mid |w|_0 = 2i \text{ und } |w|_1 = 2j \text{ für } i, j \in \mathbb{N}\}$

Klasse $[q_1] = \{w \in \{0,1\}^* \mid |w|_0 = 2i + 1 \text{ und } |w|_1 = 2j \text{ für } i, j \in \mathbb{N}\}$

Klasse $[q_2] = \{w \in \{0,1\}^* \mid |w|_0 = 2i + 1 \text{ und } |w|_1 = 2j + 1 \text{ für } i, j \in \mathbb{N}\}$

Klasse $[q_3] = \{w \in \{0,1\}^* \mid |w|_0 = 2i \text{ und } |w|_1 = 2j + 1 \text{ für } i, j \in \mathbb{N}\}$

$0 \Rightarrow i, 1 \Rightarrow j$

Widerspruchsbeweis dass jeder deterministische endliche Automat für die Sprache: $L = \{w \in \{a,b\}^* \mid w \text{ hat den Suffix } ab\}$ mindestens 3 Zustände braucht.

Für den Widerspruchsbeweis wählen wir die folgenden Wörter: $x_1 = [\epsilon], x_2 = a, x_3 = ab$

Widerspruch für alle Wortpaare:

x_1 und x_2 : $[z12] = [b] \Rightarrow [x1][z12] = [b] \notin L, [x2][z12] = [ab] \in L$

x_1 und x_3 : $[z13] = [\epsilon] \Rightarrow [x1][z13] = [\epsilon] \notin L, [x3][z13] = [ab] \in L$

x_2 und x_3 : $[z23] = [\epsilon] \Rightarrow [x2][z23] = [a] \notin L, [x3][z23] = [ab] \in L$

- Sprachen in EA umwandelbar**
- $L = \{w \in \{0,1\}^* \mid w = 0^n 10^n \text{ und } n \in \mathbb{N}\}$: DEA nicht möglich
 - $L = \{w \in \{0,1\}^* \mid |w|_1 \leq 3\}$: DEA möglich
 - $L = \{w \in \{0,1\}^* \mid w = 0^* 10^*\}$: DEA möglich
 - $L = \{w \in \{0,1\}^* \mid w = 0^n 1^m 0 \text{ und } n, m \in \mathbb{N}\}$: DEA möglich
 - $L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$: DEA nicht möglich

Ableitungsbaum für Grammatik der Chemie

Vereinfachtes bsp über das Wort ThI_2N^*

$G_{\text{Chem}} = \{\{S, E, D\}, \{Th, I, N, 2\}, P, S\}$ mit $P = \{S \rightarrow SE \mid E, E \rightarrow ED \mid Th \mid I \mid N, D \rightarrow 2 \}$

Rechtsseitige Ableitung: $S \rightarrow SE \rightarrow SN \rightarrow SEN \rightarrow SEDN \rightarrow SE2N \rightarrow SI2N \rightarrow EI2N \rightarrow ThI_2N$

Beweis Eindeutigkeit: Die kontextfreie Grammatik ist eindeutig, da es immer nur einen Ableitungsbaum gibt (nur einen Pfad der Rekursion möglich, S immer nur $\rightarrow SE$ und E immer nur $\rightarrow ED$).

- Grammatikaussagen** Betrachten Sie die folgenden zwei kontextfreien Grammatiken über dem gegebenen Alphabet $\Sigma = \{0,1\}$
- $K_1 = \{\{S\}, \Sigma, P_1, S\}$ mit $P_1 = \{S \rightarrow 0S1, S \rightarrow 01, S \rightarrow \epsilon\}$
- $K_2 = \{\{S, A\}, \Sigma, P_2, S\}$ mit $P_2 = \{S \rightarrow 0A, A \rightarrow A1 \mid \epsilon\}$

Entscheiden Sie, für welche Sprachen die Grammatikaussagen zustimmen: [K1] erzeugt die Sprache von Wörtern, in denen auf n Nullen n Einsen folgen. [K2] erzeugt die Sprache von Wörtern, in denen einer Null beliebig viele Einsen folgen.

Entscheiden Sie, mit welchen Grammatiken die folgenden Sprachen gebildet werden können:

$S_1 = \{w \in \{0,1\}^* \mid w = 0^n 1^n \wedge n \in \mathbb{N}\}$ [K1]

$S_2 = \{w \in \{0,1\}^* \mid w = 01^n \wedge n \in \mathbb{N}\}$ [K2]

$S_3 = \{w \in \{0,1\}^* \mid |w|_1 > |w|_0\}$ [Keines]

$S_4 = \{w \in \{0,1\}^* \mid |w|_1 < |w|_0\}$ [Keines]

$S_5 = \{w \in \{0,1\}^* \mid |w|_0 = 0 \wedge |w|_1 = 0\}$ [K1]

$S_6 = \{01\}$ [K1 und K2]

- Multiple-Choice** ✓ = richtig, ☒ = falsch
- ✓ Kellerautomat kann mit leerem Keller akzeptieren
 - ☒ Eingabealphabet können, müssen aber nicht identisch sein
 - ✓ Eine korrekt programmierte Turing-Maschine hält in jedem Fall an, wegen der Bedingung der endlich viele Einsen auf einer Seite
 - ✓ Die Turing Machine wird immer zuerst die Seite mit den endlich vielen Einsen bestimmen. Ansonsten wird sie nicht terminieren
 - ☒ Der Algorithmus, 1 nach rechts, 2 nach links, 3 nach rechts, usw., kann nicht verwendet werden, um zu bestimmen, welche Seite unendlich viele Einsen enthält
 - ✓ Die Turing-Maschine kann bestimmen, ob eine Seite unendlich viele Einsen enthält, wenn eine Seite endlich viele Einsen enthält

TM $M = (\{q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1, \cup, a, b\}, \delta, q_1, \cup, \{q_2\})$

$\delta(q_1, 1) = (q_4, 1, R),$
 $\delta(q_4, 0) = (q_3, 0, R),$
 $\delta(q_4, \cup) = (q_2, a, R),$
 $\delta(q_3, 1) = (q_4, b, R),$
 $\delta(q_3, 0) = (q_1, 0, R)$

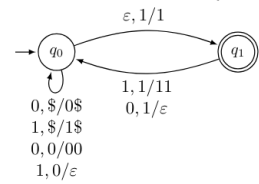
Busy-Beaver

terminierende TM, versucht grösstmögliche Ausgabe zu generieren (fixe Anzahl Zustände/-Symbolen auf Band)

Keller:

Schritt	linker Stack	rechter Stack	Band:
0		\$	\$
1		1 \$	1
2		1 \$	1
3		1 \$	1
4		1 1 \$	1 1
5		1 1 1 \$	1 1 1
6		1 1 1 \$	1 1 1
7		1 1 1 \$	1 1 1
8		1 1 1 \$	1 1 1
9		1 1 1 1 \$	1 1 1 1
10		1 1 1 1 \$	1 1 1 1
11		1 1 1 1 \$	1 1 1 1
12		1 1 1 1 \$	1 1 1 1
13		1 1 1 1 \$	1 1 1 1

Kellerautomat $L = \{w \in \{0,1\}^* \mid |w|_0 < |w|_1\}$



Polynomzeit-Verifizierer Gegeben: $a_1, \dots, a_n \in \mathbb{N}$ und b

Ausgabe: JA falls $\exists S \subseteq \{a_1, \dots, a_n\}$ mit $\sum_{a_i \in S} a_i = b$, sonst NEIN

Zeuge: konkrete Menge $a_1, \dots, a_n \in \mathbb{N}$

Verifizierer: prüft ob $\sum_{a_i \in S} a_i = b$

Nur $a_i \in S$ prüfen, da $a_i \notin S$ nicht zur Lösung beiträgt

Eingabe für Verifizierer: $w = (a_1 a_2 \dots a_n \# b)$ mit a_i und b unär codiert (z.B. a_i = 5 als 00000)

Informell: m ist die Länge der Eingabe insgesamt $\rightarrow a_1 a_2 \dots a_n \# b$ (Zahlen a_1, a_2, \dots, a_n und b). Für die Länge der Eingabe m gilt: max. n Zahlen endlicher Länge + n -mal die „1“ + b mit endlicher Länge (bzw. n Zahlen endlicher Länge) \Rightarrow Länge $(2m) \in \mathcal{O}(n)$

Diese "Probepfprüfung" dient dazu, sich mit der Art der Fragen vertraut zu machen. In den Prüfungen werden diese Fragetypen primär genutzt.

- Fragen mit einer **numerischen Antwort**: Zahlen immer ohne Leerzeichen, ohne Einheit und ohne 1000er-Trennzeichen eingeben. (Wenn die Einheit mit eingegeben werden soll (z.B. auch separat), wird dieses explizit verlangt).
Beispiel: 12000 oder 12,456 oder 1000000 (bitte nicht 1'000 oder 12 000 oder 1000 kbps eingeben)
- Fragen mit einer **Kurzantwort**: Immer ohne Leerzeichen eingeben.
Beispiel: abcde oder xyz
- Sollten Sie eine **Formel** eingeben: Ebenfalls **ohne Leerzeichen** und **ohne HTML-Features** eingeben.
Beispiel: x^2 für x^2 , $\log_2(x)$ für $\log_2(x)$ usw.
- Fragen mit einer **Freitext-Antwort**:
 1. **Ohne Anhang**: Der Text wird im Antwortfenster eingegeben. Dabei bitte die HTML-Features nicht verwenden.
 2. **Mit Anhang**: Erstellen Sie Ihre Antwort (Papier, elektr. Dokument usw.) und laden Sie die Datei hoch (z.B. kann das auch einfach ein Foto sein)
- Fragen mit einer **Auswahl von Antworten** (Multiple Choice): Wie allgemein bekannt. In der Regel ist nur eine Antwort richtig (und auch auswählbar) - für eine falsche Antwort gibt es keinen Abzug.
- Fragentyp "**Wahr / Falsch**" bzw. "**Ja / Nein**": Wie allgemein bekannt. In der Regel ist nur eine Antwort richtig - für eine falsche Antwort gibt es keinen Abzug.