Theoretische Informatik

Jil Zerndt
FS 2024

Alphabete, Wörter, Sprachen

Alphabete endliche, nichtleere Mengen von Symbolen.
- $\Sigma_{\text{Bool}} = \{0,1\}$ Boolsches 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 $\omega = xvy$.
$\omega \neq v \to$ Echtes Teilwort, Präfix = Anfang, Suffix = Ende

Mengen von Wörtern Σ^k = Wörter der Länge $\boldsymbol{k}$ über Alphabet Σ
- $\Sigma^* = \underbrace{\Sigma^0}_{1} \cup \underbrace{\Sigma^1}_{} \cup \underbrace{\Sigma^2}_{} \cup \underbrace{\Sigma^3}_{} \cdots$ Kleensche Hülle
- $\Sigma^+ = \underbrace{\Sigma^1}_{2} \cup \underbrace{\Sigma^2}_{4} \cup \underbrace{\Sigma^3}_{8} \ldots = \Sigma^* \backslash \{\varepsilon\}$ Positive Hülle

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

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

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

Kleenesche Hülle A^* von A: $\{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$
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 $\to$ (2) Konkatenation $\to$ (3) |= Oder

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

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

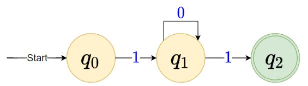
Endliche Automaten

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
Σ: endliches Eingabealphabet
$\delta : Q \times \Sigma \to Q$ Übergangsfunktion
$q_0 \in Q$ Startzustand
$F \subseteq Q$ Menge der akzeptierenden Zustände

DEA Funktionen $M = (Q, \Sigma, \delta, q_0, F)$: 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:
$(q_a, \omega_1 \ldots \omega_n) \vdash_M \ldots \vdash_M (q_e, \omega_j \ldots \omega_n) \to (q_a, \omega_1 \ldots \omega_n) \vdash_M^* (q_e, \omega_j \ldots \omega_n)$

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



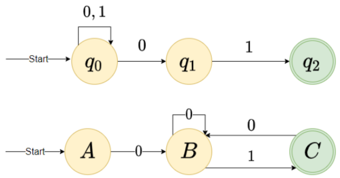
Konfiguration auf $\omega = 101$
- Startkonfiguration $\to (q_0, 101)$
- Endkonfiguration $\to (q_2, \varepsilon)$

Berechnung
$\omega = 101 \to (q_0, 101) \vdash_M (q_1, 01) \vdash_M (q_1, 1) \vdash_M (q_2, \varepsilon) \to$ akzeptierend
$\omega = 10 \to (q_0, 10) \vdash_M (q_1, 0) \vdash_M (q_1, \varepsilon) \to$ verwerfend

Nichtdeterministischer endlicher Automat (NEA)
Unterschied zum DEA: Übergangsfunktion $\delta : Q \times \Sigma \to P(Q)$
Ein ε-NEA erlaubt zusätzlich noch ε-Übergänge

Teilmengenkonstruktion $\forall$ NEA kann in DEA umgewandelt werden
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 $= F_{NEA} \to$ akzeptierend

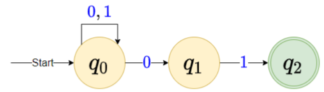
$\downarrow$	$\boldsymbol{q}$	$\boldsymbol{\delta(q,0)}$	$\boldsymbol{\delta(q,1)}$
0	~~$\emptyset$~~	~~$\emptyset$~~	~~$\emptyset$~~
	$A = \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
1	~~$\{q_1\}$~~	~~$\emptyset$~~	~~$\{q_2\}$~~
4	~~$\{q_2\}$~~	~~$\emptyset$~~	~~$\emptyset$~~
	$B = \{q_0, q_1\}$	$\{q_0, q_1\}$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

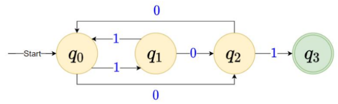


Zustandsklasse $\Sigma^* = \bigcup_{p \in Q} [p]$ $[p] \cap [q] = \emptyset, \forall p \neq q, p, q \in Q$
Jedes Wort landet in einem Zustand, aber kein Wort landet nach dem Lesen in zwei Zuständen!

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.
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.

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





äquivalenter DEA

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:
  $L^* = \{\omega = \omega_1 \omega_2 \ldots \omega_n \mid \omega_i \in L \text{ für alle } i \in \{1, 2, \ldots, n\}\}$

$L(R_1)$: Menge der ganzen Zahlen in Dezimaldarstellung
- $((- \mid \varepsilon)(1,2,3,4,5,6,7,8,9)(0,1,2,3,4,5,6,7,8,9) \mid 0).0$

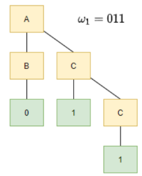
Kontextfreie Grammatiken

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 $X \to \beta$
  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 $A \to \gamma$ eine Produktion.
- Ableitungsschritt mit Produktion $A \to \gamma$ $\quad \alpha A \beta \to \alpha \gamma \beta$
- Ableitung Folge von Ableitungsschritten $\alpha \to \cdots \to \omega$

Ableitungsbaum (Parsebaum)
KGF für die Sprache $L = \{0^n 1^m | n, m \in \mathbb{N}\}$
- $G_1 = \{\{A, B, C\}, \{0, 1\}, P, A\}$
- $P = \{A \to BC,\ B \to 0B \mid 0 \mid \varepsilon,\ C \to 1C \mid 1 \mid \varepsilon\}$

Ableitung von $\omega_1 = 011$:
$A \to BC \to 0C \to 01C \to 011$



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 $\exists$ KFG
L reguläre Sprache $\Rightarrow \exists$ 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 $Q_i \to aQ_j$
- $\forall$akzeptierenden Zustand $q_i \in F$ erstelle Produktion $Q_i \to \varepsilon$
- Nichtterminal Q_0 wird zum Startsymbol A.

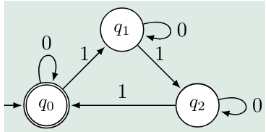
ACHTUNG: es dürfen keine Wörter, die nicht in L sind, abgeleitet werden können

KFG für die Sprache $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
$G_1 = (\{A\}, \{0,1\}, P, A)$
$P = \{A \to 0A1 | \varepsilon\}$

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

Nichtterminale: Q_0, Q_1, Q_2

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



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): $M = \left(Q, \Sigma, \boldsymbol{\Gamma}, \boldsymbol{\delta}, q_0, \$, F\right)$
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: $\boldsymbol{\delta} : \boldsymbol{Q} \times (\boldsymbol{\Sigma} \cup \boldsymbol{\varepsilon}) \times \boldsymbol{\Gamma} \to \boldsymbol{Q} \times \Gamma^*$
NKA: $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to P(Q \times \Gamma *)$ (Nichtdeterministischer KA)

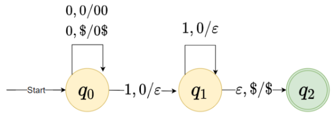
Übergangsfunktion KA $\forall$ Zustand q und $\forall$ Symbole x, b gilt:
wenn $\delta(q, b, c)$ definiert ist, dann ist $\delta(q, \varepsilon, x)$ undefiniert.
Darstellung Übergang $\delta(q, b, c) = (p, \omega)$: $\boldsymbol{q - b, c/\omega \longrightarrow p}$

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

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

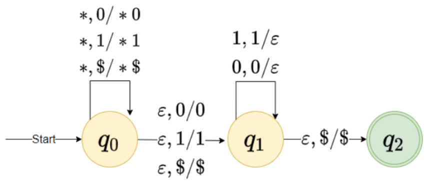
Kellerautomat für kontextfreie Sprache $\left\{0^n 1^n \mid n > 0\right\}$



$\omega_1 = 011$:
$(q_0, 011, \$) \vdash (q_1, 11, 0\$) \vdash (q_1, 1, \$)$
$\to \omega_1$ verwerfend
Das Zeichen \$ zeigt an, dass der «Stack» leer ist.

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? $L_1 = \{0^n 1^n 0^n \mid n \in \mathbb{N}\}$, $\sigma = \{0,1\} \to$ nein
$L_2 = \{waw^R \mid w \in \{0,1\}^*\}$, $\sigma = \{0, 1, a\} \to$ ja (DKA)
$L_3 = \{ww \mid w \in \{0,1\}^*\}$, $\sigma = \{0,1\} \to$ 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 $\sqcup \in \Gamma$ und $\sqcup \notin \Sigma$
Übergangsfunktion: $\boldsymbol{\delta : Q \times \Gamma \to Q \times \Gamma \times D, D = \{L, R\}}$

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
Y = Overwrite — $\boldsymbol{q - X/Y, D \to p}$
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

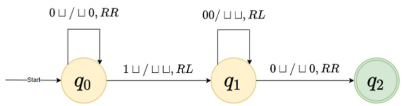
Universelle TM Codierung der Übergangsfunktion (Gödelnummer)
Zustände $q_n := 0^n$
Symbole $\Gamma = \{0, 1, \$, a \in \Sigma\}$
$(0 = 0,\ 1 = 00,\ \$ = 000)$
Richtung L = 0, R = 00

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

bsp: 0100100101010000001...01111111101...

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 ⊢	0⊔ / ⊔ 0, RR	0	0	0	0	1	0	0		
2	q_0 ⊔ ⊢										
1	⊔ $q_0$000100 ⊢	0⊔ / ⊔ 0, RR		0	0	0	1	0	0		
2	0q_0 ⊔ ⊢		0								
1	⊔⊔ $q_0$00100 ⊢	0⊔ / ⊔ 0, RR			0	0	1	0	0		
2	00q_0 ⊔ ⊢		0	0							
1	⊔⊔⊔ $q_0$0100 ⊢	0⊔ / ⊔ 0, RR				0	1	0	0		
2	000q_0 ⊔ ⊢		0	0	0						
1	⊔⊔⊔⊔ $q_0$100 ⊢	1⊔ / ⊔⊔, RL					1	0	0		
2	0000q_0 ⊔ ⊢		0	0	0	0					
1	⊔⊔⊔⊔⊔ $q_1$00 ⊢	00/ ⊔⊔, RL						0	0		
2	000$q_1$0 ⊢		0	0	0	0					
1	⊔⊔⊔⊔⊔⊔ $q_1$0 ⊢	00/ ⊔⊔, RL							0		
2	00$q_1$0 ⊢		0	0	0						
1	⊔⊔⊔⊔⊔⊔⊔ q_1	⊔ 0/ ⊔ 0, RR									
2	0$q_1$0 ⊢		0	0							
1	⊔⊔⊔⊔⊔⊔⊔⊔ q_2 ⊔										
2	00q_2 ⊔ ⊢		0	0							

Berechnungsmodelle

Turing-berechenbar = kann von Turing-Maschine gelöst werden
Turing-berechenbare Funktion $T : \Sigma^* \to \delta^*$
$$T(\omega) = \begin{cases} u & \text{falls T auf } \omega \in \Sigma^* \text{ angesetzt, nach endlich vielen Schritten mit u auf dem Band anhält} \\ \uparrow & \text{falls T bei Input } \omega \in \Sigma^* \text{ nicht hält} \end{cases}$$

$\forall$ 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: $c_k^n = \mathbb{N}^n \to \mathbb{N}$ mit $c_k^n(x_1, ..., x_n) = k$
Nachfolgerfunktion: $\eta : \mathbb{N} \to \mathbb{N}$ mit $\eta(x) = x + 1$
n-stellige Projektion auf die k-te Komponente:
$\pi_k^n : \mathbb{N}^n \to \mathbb{N}$ mit $\pi_k^n(x_1, ..., x_k, ..., x_n) = x_k$ $(1 < k < n)$
n = Anzahl der Argumente, k = Position des Arguments

Primitive Rekursion von 2 FUnktionen HEKLP

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

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) = $c_0^1(y) = 0$
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: $x0, x1, x2, \ldots$
Konstanten: $0, 1, 2, \ldots$
Operationszeichen: $+, -$
Trennzeichen: ;

Loop (primitiv-rekursiv)
Schlüsselwörter: Loop, Do, End
Sequenzen: P und $Q \to P; Q$
Schleifen: Loop x do P ... End

add(x1, x2) = x0
```
Loop  x1  Do
    x2  =  x2  +  1
End;
x0  =  x2  +  0
```

While (Turing vollständig)
Erweiterung der Sprache Loop
While $xi > 0$ Do ... End

mul(x1, x2) = x0
```
While  x1  >  0  Do
    x1  =  x1  -  1;
    Loop  x2  Do
        x0  =  x0  +  1
    End
End
```

GoTo (Turing vollständig)
Schlüsselwörter: Goto, If, Else
Marker Mk: M1, M2, ...
Sprunganweisung:
If $xi = c$ Then Goto Mk

case distinction (x1, x2, x3) = x0
```
M1:  x0  =  x3  +  0;
M2:  If  x1  =  0  Then  Goto  M4;
M3:  x0  =  x2  +  0;
M4:  Halt
```

Grundfunktionen LOOP und WHILE

Maximum max(x1, x2)

```
x0 = x2 + 0;
x3 = x1 - x2;
Loop x3 Do
    x0 = x1 + 0
End
```

Minimum min(x1, x2)

```
x0 = x2 + 0;
x3 = x2 - x1;
Loop x3 Do
    x0 = x1 + 0
End
```

Addition x1 + x2

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

Division x1 / x2 + remainder

```
x3 = x1 - x2;
x3 = x3 + 1;
While x3 > 0 Do
    x3 = x3 - x2;
    x0 = x0 + 1
End
```

Absolute Difference |x1 - x2|

```
x4 = x2 + 0;
x3 = x1 - x2;
Loop x3 Do
    x4 = x1 + 0
End;
x5 = x2 + 0;
x3 = x2 - x1;
Loop x3 Do
    x5 = x1 + 0
End;
x0 = x4 - x5
```

Fibonacci fib(x1)

```
x2 = 0 + 0;
x0 = 1 + 0;
While x1 > 0 Do
    x3 = x0 + 0;
    x0 = x0 + x2;
    x2 = x3 + 0;
    x1 = x1 - 1
End
```

Unvollständigkeit

Ackermannfunktion $a : \mathbb{N}^2 \to \mathbb{N}$
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 $a(2,1)$

$$\begin{aligned} a(2,1) &= a(1,a(2,0)) = a(1,a(1,1)) = a(1,a(0,a(1,0))) \\ &= a(1,a(0,a(0,1))) = a(1,a(0,2)) = a(1,3) = a(0,a(1,2)) \\ &= a(0,a(0,a(1,1))) = a(0,a(0,a(0,a(1,0)))) \\ &= a(0,a(0,a(0,a(0,1)))) = a(0,a(0,a(0,2))) \\ &= a(0,a(0,3)) = a(0,4) = 5 \end{aligned}$$

Beweis dass Ackermannfunktion nicht primitiv rekursiv
Definiere zu jeder primitiv rekursiven Funktion f eine Funktion $m(n,f) = \max\left\{f(\vec{x}) \mid \sum \vec{x} \le n\right\}$
Zeige, dass $\exists k \in \mathbb{N}$ für jede primitiv rekursive Funktion f, so dass $\forall n \ge k \ m(n,f) < a(k,n)$ 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 $n \ge k$
Insbesondere gilt also für $n = k$:

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

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

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 $A \subset \Sigma^*$
$A \subset \Sigma^*$ ist entscheidbar $\Leftrightarrow$ A und $\bar{A}$ sind semi-entscheidbar
A entscheidbar $\Leftrightarrow$ $\bar{A}$ entscheidbar
$\bar{A} = \Sigma^* \backslash A = \{\omega \in \Sigma^* \mid \omega \notin A\}$ (Komplement von A)

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 $x \in \Sigma^* \backslash A$ 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 $\to$ 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 $x \in \Sigma^* \backslash A$ 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 $A \preccurlyeq B$ $\Rightarrow A \subset \Sigma^*$ reduzierbar auf $B \subset \Gamma^*$
Gilt wenn $\exists\, T : \Sigma^* \to \Gamma^*$ so dass: $\forall \omega \in \Sigma^*$ $\omega \in A \Leftrightarrow F(\omega) \in B$
T := total Turing-berechenbare Funktion
Transitiv: $A \preccurlyeq B$ und $B \preccurlyeq C \to A \preccurlyeq C$
$A \preccurlyeq B \Rightarrow$ Entscheidbarkeit von B gleich wie von A

Zeige dass $A \preccurlyeq B$
$P_1(x)$:= x Primzahl? $P_2(x,y)$:= x kleinster Primfaktor von y?
$P_1(x) \preccurlyeq P_2(x,y)$ 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 $H := \{\omega\#x \in \{0,1,\#\}^* \mid T_\omega$ angesetzt auf x hält$\}$
Leeres $H_0 := \{\omega \in \{0,1\}^* \mid T_\omega$ angesetzt auf das leere Band hält $\}$
Spezielles $H_S := \{\omega \in \{0,1\}^* \mid T_\omega$ angesetzt auf ω hält $\}$
$H_0 \preccurlyeq H_S \preccurlyeq H$
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 NEA = DEA = Reguläre Sprache (z.B. 1^n0^m) < NKA < DKA = KFG = Kontextfreie Sprache (z.B. 1^n0^m) < NTM = DTM = Rekursiv abzählbare Sprachen (z.B. $a^nb^nc^n$)
primitiv rekursive Funktionen < Turing-berechenbare Funktionen

Komplexitätstheorie

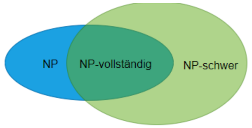
Quantitative Gesetze und Grenzen
- Zeitkomplexität: Laufzeit des besten Programms
- Platzkomplexität: Speicherplatz des besten Programms
- Beschreibungskomplexität: Länge des kürzesten Programms

Zeitbedarf von M auf Eingaben der Länge $n \in \mathbb{N}$
im schlechtesten Fall: $\text{Time}_M(n) = \max\left\{\text{Time}_M(\omega) | |\omega| = n\right\}$
Sei M eine TM, die immer hält und sei die Eingabe $\omega \in \Sigma^*$
Zeitbedarf von M auf ω: Time $_M(\omega)$ = Anzahl Konfigurationsübergänge in der Berechnung von M auf ω

P vs NP Klassifizierung von Problemen
Ein Problem U heisst in Polynomzeit lösbar, wenn es eine obere Schranke $O(n^c)$ gibt für eine Konstante $c \ge 1$.
- $P \doteq$ 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, falls für alle Sprachen $L' \in NP$ gilt, dass $L' \preccurlyeq_p L$
Eine Sprache L heisst NP-vollständig, falls $L \in NP$ und L ist NP-schwer.



$P \subset NP$ gilt, aber $P \neq \mathbb{N}P$ noch nicht bewiesen

Polynomzeit-Verifizierer Überprüft die Eingaben in einem Problem
Zeuge: Informationen einer gültigen Eingabe

Asymptotische Komplexitätsmessung O-Notation
- $f \in O(g)$: $f(n) \le c \cdot g(n)$
  - f wächst asymptotisch nicht schneller als g
- $f \in \Omega(g)$: $f(n) \ge \frac{1}{d} \cdot g(n)$
  - 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 $\text{Time}_\text{M}(n) \in \Omega(g(n))$

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

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

good luck <3

1 - Alphabete, Wörter, Sprachen

Multiple-Choice $\surd$ = richtig, $\boxtimes$ = falsch

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

2 - Reguläre Ausdrücke

Reguläre Ausdrücke und Sprachen

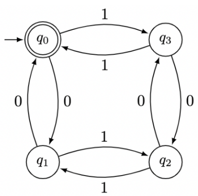
- ϵ = Sprache aller Binärzahlen Länge 0
- $(1(0 \mid 1) * 1) \mid 1$ = Sprache aller positiver ungerader Binärzahlen
- $(0 \mid 0001)*$ = Sprache aller Wörter die vor jeder 1 mind. 3 Nullen haben
- $(0 \mid 1) * 0(0 \mid 1) * 0(0 \mid 1) * 0(0 \mid 1)*$ = Sprache aller Binärzahlen die mind. 4 Nullen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

regulär/nicht regulär:

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

3 - Endliche Automaten

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



Klasse $[q_0] = \{w \in \{0,1\}^* \mid |w|_0 = 2i$ und $|w|_1 = 2j$ für $i, j \in \mathbb{N}\}$
Klasse $[q_1] = \{w \in \{0,1\}^* \mid |w|_0 = 2i + 1$ und $|w|_1 = 2j$ für $i, j \in \mathbb{N}\}$
Klasse $[q_2] = \{w \in \{0,1\}^* \mid |w|_0 = 2i + 1$ und $|w|_1 = 2j + 1$ für $i, j \in \mathbb{N}\}$
Klasse $[q_3] = \{w \in \{0,1\}^* \mid |w|_0 = 2i$ und $|w|_1 = 2j + 1$ für $i, j \in \mathbb{N}\}$

$0 \Rightarrow i,\ 1 \Rightarrow j$

Widerspruchsbeweis dass jeder deterministische endliche Automat

für die Sprache: $L = \left\{ w \in \{a, b\}^* \mid w \text{ hat den Suffix ab } \right\}$ mindestens 3 Zustände braucht.
Für den Widerspruchsbeweis wählen wir die folgenden Wörter: $x_1 = [\varepsilon], x_2 = a, x_3 = ab$
Widerspruch für alle Wortpaare:

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

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

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

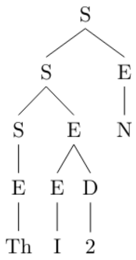
Sprachen in EA umwandelbar

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

KFG

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 $P = \{$
$S \rightarrow SE \mid E,$
$E \rightarrow ED|Th|I \mid N,$
$D \rightarrow 2 \quad \}$



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 $\Sigma = \{0, 1\}$

$$K_1 = \{\{S\}, \Sigma, P_1, S\} \text{ mit } P_1 = \{S \rightarrow 0S1, S \rightarrow 01, S \rightarrow \epsilon\}$$
$$K_2 = \{\{S, A\}, \Sigma, P_2, S\} \text{ mit } P_2 = \{S \rightarrow 0A, A \rightarrow A1 \mid \epsilon\}$$

Entscheiden Sie, für welche Sprachen die Grammatikaussagen zustimmen: [K1] erzeugt die Sprache von Wörtern, in denen auf $\boldsymbol{n}$ Nullen $\boldsymbol{n}$ 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:
$S_1 = \{w \in \{0,1\}^\star \mid w = 0^n 1^n \wedge n \in \mathbb{N}\}$[K1]
$S_2 = \{w \in \{0,1\}^\star \mid w = 01^n \wedge n \in \mathbb{N}\}$[K2]
$S_3 = \{w \in \{0,1\}^\star ||w|_1 > |w|_0\}$[Keines]
$S_4 = \{w \in \{0,1\}^\star ||w|_1 < |w|_0\}$[Keines]
$S_5 = \{w \in \{0,1\}^\star ||w|_0 = 0 \wedge |w|_1 = 0\}$[K1]
$S_6 = \{01\}$[K1 und K2]

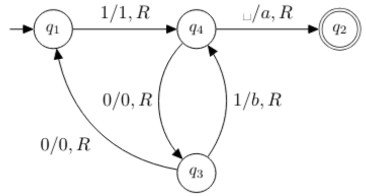
Kellerautomaten und Turingmaschinen

Multiple-Choice $\surd$ = richtig, $\boxtimes$ = falsch

- $\surd$ Kellerautomat kann mit leerem Keller akzeptieren
- $\boxtimes$ Eingabealphabet können, müssen aber nicht identisch sein
- $\surd$ Eine korrekt programmierte Turing-Maschine hält in jedem Fall an, wegen der Bedingung der endlich viele Einsen auf einer Seite
- $\surd$ Die Turing Machine wird immer zuerst die Seite mit den endlich vielen Einsen bestimmen. Ansonsten wird sie nicht terminieren
- $\boxtimes$ 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
- $\surd$ Die Turing-Maschine kann bestimmen, ob eine Seite unendlich viele Einsen enthält, wenn eine Seite endlich viele Einsen enthält

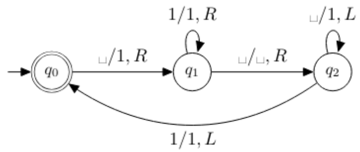
TM $M = (\{q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, \cup, a, b\}, \delta, q_1, \cup, \{q_2\})$

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



Busy-Beaver

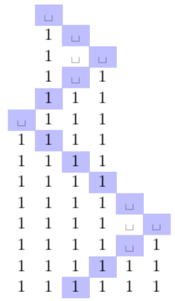
terminierende TM, versucht grösstmögliche Ausgabe zu generieren (fixe Anzahl Zustände/-Symbolen auf Band)



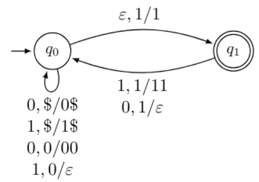
Keller:

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

Band:



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



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

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

Zeuge: konkrete Menge $a_1, ..., a_n \in \mathbb{N}$
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: $w = (a_1 1 a_2 1 ... 1 a_n \# b)$ mit a_i und b unär codiert (z.B. $a_i = 5$ als 00000)

Informell: m ist die Länge der Eingabe insgesamt $\rightarrow a_1 1 a_2 1 \ldots 1 a_n \# b$ (Zahlen $a_1, a_2, \ldots, a_n$ 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 "Probeprüfung" 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, *log2(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.