Theoretische Informatik - ZHAW 2024

Alphabete, Wörter und Sprachen

Alphabet	endliche, nichtleere Menge von Symbolen (Bsp.: Σ = {a, b, c})
Wort (String)	endliche Folge von Symbolen über einem Alphabets (Bsp.: 101 über Σ = {1, 0}) **leeres Wort (ε)** enthält keine Symbole —> ist Wort über jedem Alphabet und hat Länge 0! —> eindeutige Folgen brauchen kein Komma, w = aaa schon! (aa, a oder a, a, a oder a, aa)
Länge (Wort)	Anzahl Symbole der Folge (Bsp.: |a b c| = 5, Leerzeichen sind Symbole!)
Häufigkeit	Anzahl, welche ein Symbol in einem Wort vorkommt (Bsp.: $\|1231123\|_1 = 3$)
Spiegelwort	Wort mit umgekehrter Symbolreihenfolge (Bsp.: $(abc)^R$ = cba , $w = w^R$ —> Palindrom)
Infix (Teilwort)	**v** ist Infix von w, wenn w = x**v**y (x, y beliebige Wörter) —> wenn v ≠ w —> **echtes Teilwort** (x oder y nicht leer)
Präfix (~vor)	**v** ist Präfix von w, wenn w = **v**y (y beliebiges Wort) —> wenn v ≠ w —> **echtes Präfix** (y nicht leer)
Suffix (~nach)	**v** ist Suffix von w, wenn w = x**v** (x beliebiges Wort) —> wenn v ≠ w —> **echtes Suffix** (x nicht leer)
Wörter mit |w| = k	Die Menge aller Wörter der Länge k wird mit Σ^k bezeichnet (Σ^0 ist immer {ε}) Bsp.: Σ = {0, 1} ist Σ^3 = {000, 001, 010, 011, 100, … }
Menge aller Wörter	Kleenesche Hülle = Σ^*, $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ —> positive Hülle (alle nichtleere Wörter) Bsp.: $\{0, 1\}^*$ = {ε, 0, 1, 00, 01, 10, 11, 000, 001, …} —> Binärwörter
Konkatenation	Verkettung von beliebigen Wörtern x ∘ y = xy (assoziativ aber nicht kommutativ!) Bsp.: x = 101, y = 00 —> xy = 10100 Verkettung von Sprachen auch möglich! AB = {uv | u ∈ A und v ∈ B} Sprache AB -> Wörter mit Präfix aus A und Suffix aus B.
Wortpotenzen	Wiederholung von Symbolen ($x^{n+1} := x^n \circ x = x^n x$) Bsp.: $x^0 := \varepsilon$, a^3 = aaa
Sprache	Teilmenge von Wörtern über einem Alphabet (Bsp.: Java ist eine Sprache über ASCII) - Wenn Σ1 ⊆ Σ2 gilt und L eine Sprache über Σ1 ist, dann ist L auch eine Sprache über Σ2 - Σ^* ist eine Sprache über jedem Alphabet Σ - {} = ∅ ist die leere Sprache für jedes Alphabet - {ε} ist die Sprache, die aus dem leeren Wort ε besteht für jedes Alphabet Σ Anmerkung: ∅ ≠ {ε} - Sprachen können unendlich viele Wörter haben (Wörter sind aber endlich lang) - Alphabet muss endlich sein
Entscheidungs-problem	Input: Sprache L und Wort x Output: JA, wenn x ∈ L und NEIN, wenn x ∉ L

Reguläre Ausdrücke (RA)

Operatoren: -> Sprache über dem Alphabet {∅, ε, *, (,), |}

- A|B -> A oder B -> " * " vor Konkatenation vor " | "
- A* -> beliebig viele A

-> Erweiterungen: R+ = R(R*) , R? = (R|ε) , $[R_1, \ldots, R_k] = R_1 \mid R_2 \mid \ldots \mid R_k$

- $L(R|S) = L(S|R)$
- $L(R(ST)) = L((RS)T)$
- $L(R|(S|T)) = L((R|S)|T)$
- $L(R(S|T)) = L(RS|RT)$
- $L((R^*)^*) = L(R^*)$
- $L(R|R) = L(R)$
- $L(\varnothing) = \emptyset$
- $L(\epsilon) = \{\varepsilon\}$
- $L(a) = \{a\}$ für $a \in \Sigma$
- $L(R^*) = L(R)^*$
- $L(R|S) = L(R) \cup L(S)$
- $L(RS) = L(R)L(S)$

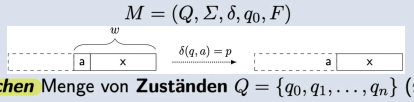
Endliche Automaten (DEA und NEA)

Definition (Endlicher Automat)

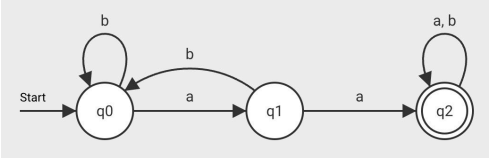
Ein **(deterministischer) endlicher Automat** (EA) ist ein Quintupel

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

mit



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



L(M) = {xaay} | x, y ∈ {a, b, c}*}
= {w ∈ {a, b, c}* | w enthält das Teilwort aa}

Bsp.-Übergang: $\delta(q_0, a) = q_1$

=> wenn auf q_0 und a lesen (im Input-Wort) dann geh nach q_1

-> "Abfall"-Zustände dürfen weggelassen werden

M = ({q0, q1, q2}, {a, b}, {$\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$, $\delta(q_1, a) = q_2$, etc.}, {q_0}, {q_2})

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

Berechnung ->

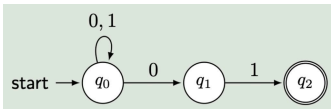
$$(q_a, w_1 w_2 \ldots w_n) \vdash_M (q_b, w_2 \ldots w_n) \vdash_M \ldots \vdash_M (q_e, w_j \ldots w_n)$$

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

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

Nichtdeterminismus

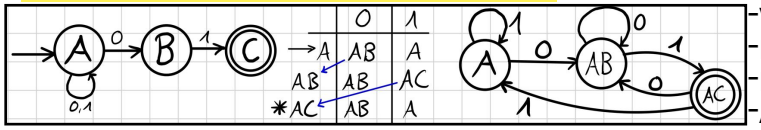
- keine, eine oder mehrere Folgekonfigurationen
- solange **mindestens eine akzeptierte Berechnung** existiert!
- Bei Berechnung von NEA: jede mögliche Route aufschreiben
- Für jeden nicht eindeutigen Übergang wird eine neue Route erstellt



Zustand	Eingabe 0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

$L(M_3) = \{x01 \mid x \in \{0,1\}^*\}$

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



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

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

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

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

Satz: ein EA und x, y zwei beliebige Wörter aus Σ*, -> x,y ∈ [p]. => alle Wörter z ∈ Σ*: xz ∈ L(M) <=> yz ∈ L(M)

Untere Schranke für die Grösse endlicher Automaten:

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

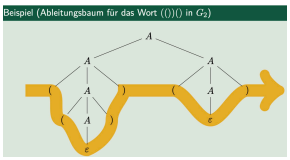
1. Überlegen, wie die n Zustandsklassen für den zu L gehörigen EA aussehen könnten.
2. n unterschiedliche Wörter finden, von denen keine zwei zur gleichen Zustandsklasse gehören.
3. Für je zwei dieser Wörter mit Hilfe von (*) und einem Widerspruchsbeweis zeigen, dass der EA zwischen diesen beiden Wörtern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

Beispiel

Zeige: Jeder EA für die Sprache $L(M_9) = \{w \in \{0,1\}^* \mid |w|_0 \bmod 3 = 1\}$ 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: $z_{12} = \varepsilon \Rightarrow x_1 z_{12} = \varepsilon \notin L$, $x_2 z_{12} = 0 \in L$
   - 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: $z_{23} = \varepsilon \Rightarrow x_2 z_{23} = 0 \in L$, $x_3 z_{23} = 00 \notin L$
4. Jeder EA für $L(M_9)$ muss zwischen mindestens drei Zuständen unterscheiden. Der EA hat mind. 3 Zustände. □



Kontextfreie Grammatiken (KFG)

auch: Parsebaum ⮍

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

Für die Sprache $\{0^n1^n \mid n \in \mathbb{N}\}$:
G = ({A}, {0, 1}, {A -> 0A1, A -> ε}, A}
-> A kann mit 0A1 oder nichts ersetzt werden:
A -> 0**A**1 -> 00**A**11 -> 000**A**111 -> 000111 ($0^3 1^3$)
Ableitung: Folge von Ableitungsschritten -> Satzform α in Wort *w* abzuleiten.
A =>* w = A erzeugt/generiert w (=>* : endliche Anzahl an Ableitungen)
-links-/rechtsseitige Ableitung:bei jedem Schritt:Nichtterminal ganz links/rechts ableiten
-erzeugte Sprache: ALLE Wörter die aus A ableitbar sind.

Wenn L = L(G) -> kontextfreie Sprache

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

Kellerautomaten (DKA und NKA)

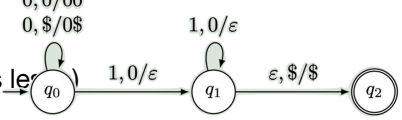
- ist ein endlicher Automat mit zusätzlichem unbegrenzten **Stack** (first-in-first-out!) Beispiel: -> $ heisst, dass Stack leer ist

Definition (deterministischer Kellerautomat)

Ein **deterministischer Kellerautomat (KA)** M ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, \$, F)$, wobei

- Q ist eine endliche Menge von Zuständen.
- 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.

M = ({q_0, q_1, q_2}, {0, 1}, {0, $}, δ*, q_0, $, {$q_2$}) (Start-Stack: $)
δ* = {δ(q_0, 0, 0) = (q_0, 00), δ(q_0, 0, $) = ($q_0$, 0$), etc.}
Berechnungsschritt: δ(q, b, c) = (p, w)
1.Ist im Zustand **q**
2.liest das Symbol **b** (falls b = ε: nichts lesen)
3.entfernt oberstes Stack-Symbol **c**
4.schreibt das Wort **w** auf den Stack (von hinten nach vorne!)
5.wechselt in den Zustand **p**



Wenn deterministisch: wenn (q, b, x): (q, **ε**, x) darf nicht auch noch dort existieren!

Nichtdeterminismus funktioniert wie bei den Automaten

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

Turingmaschinen (TM)

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

Definition (Turing-Maschine)

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

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

mit einer bzw. einem:

- *endlichen* Menge von **Zuständen** $Q = \{q_0, q_1, \ldots, q_n\}$ $(n \in \mathbb{N})$,
- **Eingabealphabet** $\Sigma = \{a_1, a_2, \ldots, a_m\}$ $(m \in \mathbb{N})$,
- **Übergangsfunktion** $\delta\colon Q \times \Gamma \to Q \times \Gamma \times D$, $D = \{L, R\}$,
- **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 $\sqcup \notin \Sigma$.

M = ({q_0, q_1, q_f}, {0, 1}, δ^*, q_0, {␣, 0, 1, #}, ␣)

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

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

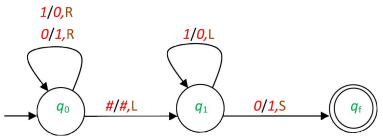
Startzustand: Lesekopf über erstem Zeichen der Eingabe

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

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

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

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



Akzeptiert=TM endet in akzeptiertem Zustand -> Sprache von TM = “rekursiv aufzählbar”
Jede Erweiterung einer TM (Speicher, mehrere Bänder, Köpfe, einseitiges Band, etc.) ist gleich mächtig, wie die normale TM.

Universelle TM (Gödelnummer - Binäre Codierung für TM) -> Bsp.: (0100100010100011001 … 010111 1100)

1. Schritt: Zustände von TM:	**2. Schritt: Bandsymbole**	**3. Schritt: Richtung**	**4 + 5. Schritt: Zusammensetzen**
q_1 = Startzustand q_2 = akzeptierender Zustand q_3, q_4, etc. = Rest Zustände Codierung: q_a = a Nullen Bsp.: q_2 = 00	X_1 = Symbol 0 X_2 = Symbol 1 X_3 = Symbol ␣ (blank) X_4, X_5, … = Rest Codierung: X_a = a Nullen	D_1 = Symbol 0 D_2 = Symbol 1 Codierung: D_a = a Nullen	- Zwischen alle Elemente **eine 1** (Trennzeichen) - Zwischen alle Übergänge **zwei 1** - Nach allen Übergängen **drei 1** danach Input (Gödelnummer kann auch dezimal codiert sein: -> in binär umwandeln und führende 1 entfernen)

Berechnungsmodelle

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

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

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

Primitiv Rekursiv	**LOOP (primitiv rekursiv)**	**WHILE (Turing**	**GOTO (Turing vollständig)**

$\forall n \in \mathbb{N}$ und jede Konstante $k \in \mathbb{N}$ die 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 $\eta(x) = x + 1$
$\forall n \in \mathbb{N}, 1 < k < n$ die n-stellige Projektion auf die k-te Komponente:
$\pi_k^n : \mathbb{N}^n \to \mathbb{N}$ mit $\pi_k^n(x_1, ..., x_k, ..., x_n) = k$
n = Anzahl der Argumente, k = Position des Arguments
$Add(0, y) = \pi_1^1(y)$
$Add(x + 1, y) = \eta(\pi_1^3(Add(x, y), x, y))$
	Zuweisungen:
	Variabel = Variabel ± Konstante
	Sequenzen verbinden: P ; Q
	Schleifen: Loop x Do P End
	
	Loop x1 Do
	x2 = x2 + 1
	End
	x0 = x2 + 0;
	x0 = x0 + 1
		Erweiterung von LOOP:
		While Bedingung Do
		
		While x1 > 0 Do
		x1 = x1 - 1;
		Loop x2 Do
		x0 = x0 + 1
		End
		End
			Zuweisungen: wie LOOP
			(Un)bedingte Sprunganweisung:
			Goto Mr / If xi = c Then Goto Mr
			Stoppanweisung: Halt
			
			M1: x0 = x3 + 0
			M2: IF x1 = 0 THEN GOTO M4
			M3: x0 = x2 + 0
			M4: HALT

Entscheidbarkeit

- Entscheidbar: Es existiert ein Algorithmus, welcher für jede Eingabe eine Antwort liefern kann (Ja/Nein)
- Semi-entscheidbar: Es existiert ein Algo., welcher für jede Eingabe (∈ A) “Ja” zurück gibt. Wenn Eingabe ∉ A, terminiert der Algo nie.
- A als auch Ā sind semi entscheidbar <=> A ist entscheidbar (Ā = Σ* \ A = { ω ∈ Σ* | ω ∉ A})
- Sind A, B (semi-) entscheidbare Sprachen, dann sind auch A ∪ B und A ∩ B (semi-) entscheidbare Sprachen.

Entscheidbarkeit mit Turingmaschinen

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

- Bandinhalt $x \in A$ T hält mit Bandinhalt «1» (Ja) an
- Bandinhalt $x \in \Sigma^* \backslash A$ 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 → 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 ≤ B = A ist reduzierbar auf B

-> Es gibt eine Funktion F(ω) so dass: ∀ω ∈ Σ* ω ∈ A <=> F(ω) ∈ B

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

Funktion: F(ω) = ω × 2: -> Wenn Zahl durch drei, dann ist Zahl mal zwei, durch 6 teilbar.

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

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

Satz: A ≤ B und B ≤ C => A ≤ C / ist B (semi / nicht)entscheidbar und A ≤ B, dann ist A auch (semi / nicht)entscheidbar!

Halteproblem: H := {ω#x ∈ {0, 1, #}* | T_ω angesetzt auf x hält} (# = Trennung zwischen codierter TM und Input)
- leeres Halteproblem H_0:
  —> H_0 := { ω ∈ {0, 1}* | T_ω angesetzt auf das leere Band hält }
- spezielles Halteproblem H_S:
  —> H_S := { ω ∈ {0, 1}* | T_ω angesetztauf ω hält}

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

Beweisidee H_S nicht entscheidbar:
Eine Maschinen, welche bei Input folgendes macht: hält eine TM auf dem Input? Ja: laufe weiter, Nein: halte an
Wenn man nun aber diese Maschine mit ihrem eigenen Code füttert entsteht ein Paradox: *würde sie halten, laufe weiter… aber Moment mal… sie sollte doch anhalten… okay.. dann läuft sie wohl weiter… aber Moment mal, wenn sie weiterläuft, sollte die Maschine stoppen. ähh?*

Reduzieren auf H: Funktion: F(x) = x#x —> H ist auch nicht entscheidbar!
H reduzieren auf H_0: F() = x (wir schreiben einfach was aufs Band und verhalten uns dann wie in H)

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

Komplexitätsthoerie
- Zeitkomplexität: Laufzeit des besten Programms, welches das Problem löst
- Platzkomplexität: Speicherplatzbedarf des besten Programms
- Beschreibungskomplexität: Länge des kürzesten Programms.

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

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

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

Rechenregeln für *O*-Notation:
- konstante Vorfaktoren ignorieren: c · f(n) ∈ O(f(n))
- Für Konstanten c gilt: c ∈ O(1)
- Bei Polynomen: höchste Potenz entscheidet : $a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0 \in O(n^k)$
- Transitivität: f(n) ∈ O(g(n)) und g(n) ∈ O(h(n)) —> f(n) ∈ O(h(n))

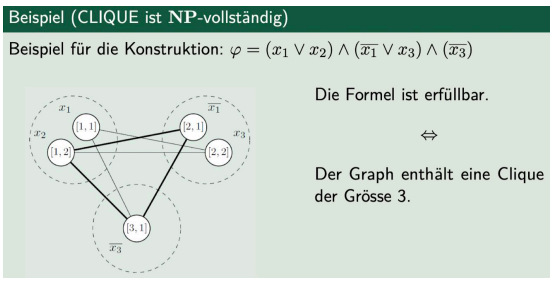
Klasse P: Alle Probleme, die von einer **deterministischen TM in Polynomzeit** gelöst werden können
Klasse NP: Alle Probleme die von einer **NICHT-deterministischen TM in Polynomzeit** gelöst werden können (≠ nicht polynominell!)

Polynomzeit-Verifizierer
“Zeuge”: mögliche Lösung für ein Problem. Wenn die Lösung in polynomineller Zeit verifiziert werden kann: Polynomzeit-Verifizierer
=> P ≙ Lösung finden in Polynomzeit, NP ≙ Lösung verifizieren in Polynomzeit
P = NP ist noch nicht geklärt

Eine Sprache L heisst NP-schwer, falls für alle Sprachen L′ ∈ NP gilt dass L′ $\preccurlyeq_p$ L
—> *D.h. L ist bezüglich der Lösbarkeit in polynomieller Zeit **mindestens so schwer** wie jedes einzelne Problem in NP.*

Eine Sprache L heisst NP-vollständig, falls L ∈ NP und L ist NP-schwer.
-> D.h Wenn P_1 NP-schwer und P_2 in NP enthalten ist und eine polynomielle Reduktion $P_1 \preccurlyeq_p P_2$ existiert, dann ist P_2 NP-vollständig.
—> Wenn für ein NP-vollständiges Problem gezeigt wird, dass es in P liegt, dann gilt: N = NP

Bsp.: NP-vollständig:



Laufzeiten-Reihenfolge:

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