1 Endliche Automaten

geornetes Quintupel A = $(\Phi, \Sigma, \delta, S, F)$

- Φ Zustandsmenge
- Σ Eingabealphabet
- δ Übergangsfunktion $\Phi \times \Sigma \rightarrow \Phi$
- S Startzustand
- F Menge der ausgezeichneten Zustände

- L(A)$\subseteq \Sigma^*$: Sprache des Automaten
  Menge aller Worte zu denen der Automat JA sagt
- Menge aller Wort über Σ: Σ^* (Σ^+ = $\Sigma^* \backslash \{\epsilon\}$)
- leeres Wort ϵ
- l(w): Länge des Wortes
- semantische Äquivalenz : L(A_1) = L(A_2), strenge Äquivalenz falls $\Sigma_1 = \Sigma_2$
- Wortverknüpfungen sind Konkatenationen (u,v Worte): uv
- u Teilwort von w: w = $u_1 u u_2$; echtes Teilwort, falls u_1 oder $u_2 \neq \epsilon$
- Verknüpfungen von Sprachen:
  A = $(\Phi, \Sigma, \delta, S, F)$
  $\Phi = \Phi_1 \times \Phi_2 = \{(T_1, T_2), T_1 \in \Phi_1, T_2 \in \Phi_2\}$
  $\delta : \Phi \times \Sigma \rightarrow \Phi$
  $\delta((T_1, T_2), z) = (\delta_1(T_1, z), \delta_2(T_2, z))$
  S = (S_1, S_2)
  F = $\{(T_1, T_2) \mid T_1 \in F_1$ oder (und) $T_2 \in F_2\} \Rightarrow L(A) = L(A_1) \cup (\cap) L(A_2)$
- L $\subseteq \Sigma'^*$, L endlich (endl. Menge von Passwörtern) Existiert A mit L(A) = L?
  Ja, Kombination von |L| Automaten, die alle genau ein Wort akzeptieren
- L = $\{\}$ $\Rightarrow$ Ja, F = $\varnothing$
- L = Σ^* $\Rightarrow$ Ja, F = Φ (Sprache unendlich)
- L unendlich? $\Rightarrow$ Nein (nicht für alle unendlichen Sprachen)

- Schubfachprinzip

  L(A) = $\{a^n b^n \mid n \in \mathbb{N}\}$ Widerspruch

  n + 1 Dinge in n Schubfächer $\Rightarrow$ in einem Schubfach min. 2 Dinge

  aber: Obermenge von $a^n b^n$ ist darstellbar $\Rightarrow$ L = Σ^*

- Pumping Lemma

  A mit m Zustaänden und akzeptiert w mit l(w) $\geq$ m, so akzeptiert A unendliche viele Worte

- Nondeterministischer Automat

  Übergangsfunktion kann auf Mengen von Folgezuständen abbilden und es müssen von einem Zustand nicht alle möglichen Pfeile abgehen!

- Kann NEA als DEA simuliert werden?

  Ja, mit Potenzmenge $\Phi_D = 2^{\Phi_N}$: Menge aller Teilmengen der ursprünglichen Zustandsmenge.

  Somit kann δ_D alle nondeterministischen Zustandsübergange simulieren

2 Endliche Maschinen

sind ein 6-Tupel M = $(\Phi, \Sigma_1, \Sigma_2, \delta, \lambda, S)$

- Σ_1 ist Eingabealphabet, Σ_2 ist Ausgabealphabet
- Übergangsfunktion: $\delta : \Phi \times \Sigma_1 \rightarrow \Phi$
- Ausgabefunktion: $\lambda : \Phi \times \Sigma_1 \rightarrow \Sigma_2$
- Präzisierung und Synchronisation von EINGABE und AUSGABE
  $\hat{\delta} : \Sigma_1^* \rightarrow \Phi$
  mit $\hat{\delta}(\epsilon)$ = S und $\hat{\delta}(wz) = \delta(\hat{\delta}(w), z)$
  $\hat{\lambda} : \Sigma_1^* \rightarrow \Sigma_2^*$
  mit $\hat{\lambda}(\epsilon) = \epsilon$ und $\hat{\lambda}(wz) = \hat{\lambda}(w)\lambda(\hat{\delta}(w), z)$
- Leistungsgrenze

  Es gibt keine endliche Maschine, welche das Produkt $a * b$ zweier bel. natürlicher Zahler a, b immer korrekt berechnet (Widerspruchsbeweis mit Längeninvarianz, mit t = 2^n und t * t)
- serielle Kopplung von Maschinen ist möglich mit $\Sigma_2^{(1)} \leq \Sigma_1^{(2)}$

  Generisches Verfahren: kartesisches Produkt der Zustände

  Preis:: $|\Phi| = |\Phi_1| * |\Phi_2|$
- Rückkopplung $\Rightarrow$ Nein
- Bilanz für Maschinen

  Ausgabezeichen hängt nur von dem gelesenen Eingabezeichen und dem Zustand, den wir nach lesen des Präfix w_1 erreicht haben $\Rightarrow$ es spielt nicht die gesamte Vorgeschichte sondern nur der aktuelle Zustand eine Rolle ("Markov-Eigenschaft")

  Verarbeitung hängt nicht vom Suffix ab

  festes Verhältnis von Ein- und Ausgabe (Maschinen haben kein Gedächtnis)
- semantische Äquivalenz: $\Sigma_1 = \Sigma_1'$ und $\hat{\lambda}(w) = \hat{\lambda}'(w)$ $\forall$ w $\in \Sigma_1^*$
- Moore Maschine
  $M_{oo} = (\Phi, \Sigma_1, \Sigma_2, \delta, \mu, S)$
  Markierungsfunktion: $\mu : \Phi \rightarrow \Sigma_2$
  $\hat{\mu} : \Sigma_1^* \rightarrow \Sigma_2^*$
  mit $\hat{\mu}(\epsilon) = \mu$(S) und $\hat{\mu}$(wz) = $\hat{\mu}$(w)$\mu(\hat{\delta}(wz))$
  Jede Moore-Maschine lässt sich durch eine endliche Maschine simulieren.
  Zu jeder endlichen Maschine existiert Moore-Maschine
  (aktuelle Eingabezeichen in Zustandsmenge codiert)

3 Grammatiken, Regelsprachen, Markov-Algorithmen

Eine Regelgrammatik ist ein 4-Tupel G = (Σ', Σ, R, S) mit

- Σ' Gesamtalaphabet
- Σ Terminalalphabet
- S Startwort $\in \Sigma' - \Sigma$
- R endliche Menge von Regeln R = $(r_1, ..., r_n)$

  mit $r_i = (u_i, v_i)$ mit $u_i \in \Sigma'^* - \Sigma^*$, d.h. u_i enthält mindestens ein Nichtterminalzeichen
- y′ kann aus y direkt abgeleitet werden, falls passende Regel gibt
- Folge $(w_1, w_2, ..., w_k)$ heißt Ableitung der Länge k-1
- Jedes aus dem Startwort S ableitbare Wort w $\in \Sigma'^*$ heißt <u>Satzform</u> von G und die Menge aller Worte in Σ^* heißt <u>(Regel-) Sprache von G: L(G)</u>
- Beweis G hat Sprache L:

  L $\subseteq$ L(G): alle Worte in Grammatik

  L(G) $\subseteq$ L: keine anderen Worte erzeugbar aus L
- <u>kontextsensitive</u> Regel: r = $(w_1 A w_2, w_1 w w_2)$ mit A $\in \Sigma' - \Sigma$ und w $\neq \epsilon$

  falls $w_1, w_2 \neq \epsilon \Rightarrow$ echt kontextsensitiv
- <u>kontextfreie</u> Regel r = (A,v) mit A $\in \Sigma' - \Sigma$, v $\in \Sigma'^*$

  Eine Regelgrammatik heißt kontextfrei, falls alle Regeln kontextfrei sind (links nur (ein) Nichtterminalzeichen)
- <u>rechtreguläre</u> Regel r = (u,v) u $\in \Sigma' - \Sigma$

  $$\text{v} \left\{ \begin{array}{ll} aA & , a \in \Sigma, A \in \Sigma' - \Sigma \\ a & , a \in \Sigma \\ \epsilon & \end{array} \right\}$$
- <u>abschließende</u> Regel r = (u,v), falls v $\in \Sigma^*$
- <u>lineare</u> Regel r = (T,v), T $\in \Sigma' - \Sigma$, v = wAw′, w,w′ $\in \Sigma^*$, A $\in \Sigma' - \Sigma$

  Lineare Grammatik: jede ableitbare Satzform enthält höchstens ein Nichtterminalzeichen

  (nur lineare und abschließende Regeln)
- Top-Down: man versucht von S aus das Ableitungswort zu finden
- Bottom-Up: Rückverfolgung des Ableitungsprozesses

- Kontextfreie Sprache: es ex. min. eine kontextfreie Regelgrammtik mit L(G) = L
- echt kontextsensitive Sprache: keine kontextfreie, aber -sensitive G mit L(G) = L
- Typisierung von Regelgrammatiken (Chomsky)
  TYP-0-Grammatik: G = (Σ', Σ, R, S) (keine Einschränkungen)
  TYP-1-Grammatik: r = (u,v) $\Rightarrow$ l(u) $\leq$ l(v) (nicht verkürzend, beschränkt)
  TYP-2-Grammatik: r = (u,v) u $\in \Sigma' - \Sigma$, v $\in \Sigma'^*$ (kontextfrei)
  TYP-3-Grammatik: r = (u,v) u $\in \Sigma' - \Sigma$ (reguläre Sprache)
  $$\text{v} = \left\{ \begin{array}{ll} aA & ,a \in \Sigma, A \in \Sigma' - \Sigma \\ a & ,a \in \Sigma \\ \epsilon & \end{array} \right\}$$
- TYP-3-Grammatiken leisten genausoviel wie endliche Automaten
  $\Sigma' - \Sigma = \Phi$, $\Sigma = \Sigma$, T' = δ(T,a), S = S, r(T,aT'), r = (T,a), falls δ(T,a) $\in$ F
  Regelgrammatiken leisten sogar mehr als EA
- Markov-Algorithmen: MA = (Σ', Σ, t, R), t Trennzeichen
  Substitutionsalgorithmus: Input (w $\in \Sigma'^*$ beliebig) $\Rightarrow$ Output
- Markov-Automat
  Markov-Algorithmus der Wort aktzeptiert, falls mit leerem Band terminiert wird
- bei Markov-Algorithmen ist Reihenfolge nicht egal
- kontextfreie Grammatik G = (Σ', Σ, R, S) mit L = $\{a^n b^n\}$
  Σ' = {S,a,b}, Σ = {a,b}
  r_1 = (S,ab)
  r_2 = (S,aSb)
- G für L = $\{a^n b^n c^m | n, m \in \mathbb{N}^+\}$
  Σ' = {S,T,U,a,b,c}, Σ = {a,b,c}
  r_1 = (S,aTbU), r_2 = (T,aTb), r_3 = (T,ϵ), r_4 = (U,c), r_5 = (U,cU)
- MA für $n_1 \dot{-} n_2 = \left\{ \begin{array}{ll} x - y & x \geq y \\ 0 & x < y \end{array} \right\}$ (asymmetrische Differenz)
  Σ = {I}, Σ' = {I,t} mit R = (r_1, r_2, r_3)
  r_1 = (ItI,t)
  r_2 = (tI,t)
  r_3 = (t,ϵ)

4 Notationen, mathematischen Vokabular

- f = (X,Y,R) ist Korrespondenz mit binärer Relation R
- f heißt total, falls Def(f) = X
- f heißt surjektiv, falls Bild(f) = Y
- rechtseindeutig oder partielle Abbildung falls $\forall x \in X$ und $\forall y \in Y$ aus (x,y) $\in$ R und (x,y′) $\in$ R $\Rightarrow$ y = y′

  ($\Rightarrow$ nicht bei jeder Eingabe gibt es Output)

  f: X $\dashrightarrow$ Y (wenn wir treffen, dann nur 1 x)
- linkseindeutig oder injektiv, falls $\forall x \in X$ und $\forall y \in Y$ aus (x,y) $\in$ R und (x′,y) $\in$ R $\Rightarrow$ x = x′
- rechtseindeutig und total: f: X $\rightarrow$ Y (totale Fkt.)
- f(X,Y,R) ist bijektiv, falls total, rechtseindeutig, linkseindeutig und surjektiv
- π_i^{k+1} heißt Projektion auf die i-te Komponente (i = 0,1,...,k)
- Y^X = {f | f: X $\rightarrow$ Y} (Menge aller totalen Fkt. von X in Y)

  X = $\varnothing$: $Y^{\varnothing}$ = {( )} (leere Tupel)

  Y = $\varnothing$: $\varnothing^X$ = $f_\perp$ (f bottom, ist die nirgends definierte Funktion)

  Sinnvoll?

  z.B. MA: r_1 = (I,t), r_2 = (t,I)

  Output: $\left\{ \begin{array}{ll} \perp & \text{, falls w} \neq \epsilon \\ \epsilon & \text{, falls w} = \epsilon \end{array} \right\}$

5 Unbeschränkte Registermaschinen (URM)

- normierte URM = (Σ, M, L)
- $\Sigma = \{|\}$ M = $\mathbb{N}$ (M Menge der Register)
- L = $\{a_i, s_i, t_i | i = 0, 1, 2, ...\}$ (Befehle)
- A_i Addition einer 1 im Register i
- S_i Subtraktion einer 1 im Register i, falls w ≥ 1
- T_i Test auf 0 im i-ten Register, Ausgabe 1, falls 0 ($\perp$, falls z_i = 0)
- Z Zustandsmenge: Menge aller denkbaren Registerbelegungen
  Z = $(z_0, z_1, ...)$

- normierte Kommunikationsmaschine K = (C,L,α_k, w_m)
- C ist Rechenmaschine mit (Σ, M, L) (normierte URM)
- L nichtleere Wortmenge: L $\subseteq \Sigma_a^*$ (L = $\mathbb{N}$)
- $\alpha_k : L^k \to z$ Eingabefunktion mit Eingabeparameter k
  $\alpha_k(n_1, ..., n_k)$ = (0,$n_1, ..., n_k$,0,0,...)
- $w_m : z \to L^m$ Ausgabefunktion mit Ausgabeparameter m
  $w_m(z_0, z_1, ..., z_m, z_{m+1}$,...) = $(z_1, z_2, ...z_m)$

- Ablauf: Eingabe gemäß α_k - Rechnen - Ausgabe gemäß w_m
- Rechnen: (Syntax und Semantik von URM-Programmen)
  - A_i und S_i sind n.URM-Programm für jedes i = 0,1,2,...
  - M_1 und $M_2 \Rightarrow M_1; M_2$ URM (Verkettung)
  - M URM-Programm $\Rightarrow$ (M)i URM, wiederhole sooft bis im i-ten Register 0 auftritt (bedingte Wiederholung)
- Regelgrammtik zum Erstellen der Syntax
  - G = (Σ', Σ, R, P), $\Sigma' - \Sigma = \{P\}$,
  - R: $r_1 : P \to A$, $r_2 : P \to S$, $r_3 : P \to PI$, $r_4 : P \to P; P$, $r_5 : P \to (P)$
- durch diese Regeln wird URM-Programm eindeutig eine partielle Abbildung zugeordnet (Beweisidee: vollst. Induktion nach Anzahl der Klammerpaare bzw. Semicolons)

- L(G) = $\underline{P_{URM}}$ Menge aller Programme:
  berechnen alles was intuitiv berechenbar ist
- Semantik $||P_{URM}|| : L^k \dashrightarrow L^m$; $||P_{URM}|| = w_k \circ |P_{URM}| \circ \alpha_k$
- Eine partielle Fkt heißt URM-berechenbar, wenn min. ein P $\in P_{URM}$ mit f = $|P|$
- Bsp: f(x,y) = x + y $f_+ = w_1 \circ ||(A1; S2)2|| \circ \alpha_2$
- Semantische Äquivalenz
  - $||P|| = w_m \circ |P| \circ \alpha_k = w_m \circ |P'| \circ \alpha_k = ||P'||$
  - streng, wenn $||P||$ = $||P'||$ für alle k $\in \mathbb{N}$, m $\in \mathbb{N}_+$
- Hilfskonstrukte
  - MOV(i,j) = (Sj)j;(Aj;Si)i (Verschieben von Reg i nach Reg j)
  - ADD(i,j,k) = (Sk)k;(Aj;Ak;Si)i;(Ai;Sk)k; Addieren von Register i auf Register j mit Hilfe von k
  - DUPL(i,j,k) = (Sj)j;ADD(i,j,k); Dupliziren von Register i in Register j mit Hilfe von k
- berechenbare Funktionen: x + y, x * y,...
- Funktion f(i) = $\begin{pmatrix} 1 & \text{,falls } ||P_i||(i) = 0 \\ 0 & \text{sonst.} \end{pmatrix}$ nicht URM-berechenbar

  mit $P_0, ..., P_i, ...$ beliebige Abzählung von URM-Programmen

  Beweis per Widerspruch:

  Sei P_{i0} URM-Programm, welches f berechnet:

  $||P_{i0}||$(i0) = f(i0) = $\begin{pmatrix} 1 & \text{,falls } ||P_{i0}||(i0) = 0 \\ 0 & \text{sonst.} \end{pmatrix}$

  $\Rightarrow$ vollständiger Widerspruch (Selbstanwendungsproblem)
- Sprache
  - entscheidbar:
    $\exists$ MA, der $\forall w \in \Sigma^*$ und nach endlich vielen Schritten abbricht und w genau dann akzeptiert, wenn w $\in$ L
  - semi-entscheidbar:
    $\exists$ MA, der bei allen Worten aus L abbricht und w akzeptiert, alle anderen Worte w $\in \Sigma^*$-L aber nicht akzeptiert
- Programm für f(x) = $\begin{pmatrix} 0 & \text{falls x = 0} \\ f_\perp & \text{sonst.} \end{pmatrix}$

  (A1;S1)1 (ist unbeschränkte Minimalisierung von f(x,y) = x + y)

6 Turingmaschinen, Turingprogramme, Turingautomaten

- Bandbewegungen (Lesekopfbewegt sich)
  - keine Bewegung: 0
  - Bewegung um eine Zelle nach Rechts: R
  - Bewegung um eine Zelle nach Links: L
- Konfiguration K = (q,w_o, w_1) (Momentaufnahme)
  - q $\in$ Q (Zustandsmenge)
  - $w_0 \in \Sigma'^*$: wesentlicher linker Bandinhalt
  - $w_1 \in \Sigma'^*$: wesentlicher rechter Bandinhalt
  - $q_s \in$ Q Startzustand
- Startkonfiguration: K = (q_s, ϵ, w)
- Turingmaschine TM = (Σ', Q, q_s) mit $\Sigma' = \Sigma \cup \{\not b\}$ (blank)
- TM werden durch Turingprogramme gesteuert, welche aus endl. vielen Turinginstruktionen bestehen
- Turinginstruktion i = (q,$\sigma, q', \sigma', \delta$) mit Anwendungs- und Ausführungsteil, q,q′ $\in$ Q, $\sigma, \sigma' \in \Sigma'$:

  Lesen von σ im Zustand q $\Rightarrow$ Schreiben von σ', gehen in Zustand q′ und Kopfbewegung δ

  Einzelschrittfunktion: gibt zu Konfiguration Nachfolgekonfiguration an

  Menge aller Instruktionen endlich, da Q und Σ endlich
- Menge aller Turing-Programme
  - NP_{Turing}: Menge aller Turing-Programme
  - P_{Turing}: Menge aller det. TP
  - TP ist deterministisch, falls es nicht zwei Instruktionen mit gleichem Anwendungsteil gibt
- Kommunikation mit Außenwelt
  - $\alpha_k(n_1, n_2, ..., n_k) = (q_s, \epsilon, n_1 \not b n_2 \not b ... \not b n_k)$ (Startkonfiguration)
  - $w_m(q, w_0, w_1) = \begin{pmatrix} (v_1, v_2, ..., v_m) & m \leq l \\ (v_1, v_2, ..., v_l, \epsilon, ..., \epsilon) & m > l \end{pmatrix}$ (Ausgabefunktion)

    mit $w_1 = v_1 \not b v_2 ... \not b v_l$
- partielle Abbildung f heißt Turing-berechenbar $\Leftrightarrow$ es ex. ein TP $\in P_{Turing}$ für eine geeignete TM
- Nullfunktion c_0(n) = 0 $\forall n \in \mathbb{N}$

- semantische Äquivalenz
  - $||TP_1|| = ||TP_2||$ für vorgegebene Kommunikations-Parameter k $\in \mathbb{N}$, m $\in \mathbb{N}_+$
  - streng semantisch äquivalent, wenn $\forall k \in \mathbb{N}$, m $\in \mathbb{N}_+$
- Jede endliche Maschine kann von einer TM simuliert werden

  r_1 = (S,$\not b$,S,$\not b$,R), $r_{T,z}$ = (T,z,$\delta(T,z)$,$\lambda(T,z)$,R)
- Turing-Automaten TA = (TM,F,TP) mit TM = (Σ', Q, q_s), TP $\in P_{Turing}$, F $\subseteq$ Q

  DTA: det TA, NDTA: nondet. TA

  F: Menge ausgezeichneter Zustände

  L(TA) = $\{w \in \Sigma^*$ | TA akzeptiert w$\}$

  TP nichtdeterministisch, so wird ein Wort w akzeptiert g.d.w. es mindestens eine Folge anwendbarer Instruktionen gibt, die mit einer Konfiguration $\hat{K} = (\hat{q}, \hat{w}_0, \hat{w}_1)$ abbricht mit $\hat{q} \in F$
- Zeitkomplexität eines DTA

  $t_{DTA}(n)$ = max$\{$
  - p $\in \mathbb{N}|$ es gibt Wort w $\in \Sigma^*$ der Länge n, so dass bei Eingabe von w (d.h. bei Start mit $K_0 = (q_s, \epsilon, w)$) bis zum Abbruch p Instruktionen ausgeführt werden
  - $\perp$, falls es ein Wort w mit l(w) = n gibt ohne Abbruch

  $\}$
- Speicherplatzkomplexität eines DTA

  s_{DTA}(n) = sup$\{$Speicherplatzbedarf(w), w $\in \Sigma^*$, l(w) = n $\}$

  Speicherplatzbedarf einer Konfiguration K =(q,w_0, w_1)

  s(K) = l(w_0) + l(w_1) + 2 ($\not b$ links und rechts, Delimiter)

  Speicherplatbedarf(w) = max (sup) $\{$s(K) | K wird erreicht bei der Verarbeitung von w $\}$
- Die Klasse P ("deterministisch in Polynomialzeit entscheidbar")

  ist die Menge aller Sprachen L, welche von det. Turingautomaten DTA in polynomialer Zeitkomplexität entscheibar sind, d.h.

  P = $\{$ L $\subseteq \Sigma^*|$ es gibt mindestens einen DTA und ein Polynom p(n) mit den Eigenschaften:

  1) DTA hält für alle Worte w $\in \Sigma^*$ an

  2) L(DTA) = L

  3) t_{DTA}(n) $\leq$ p(n)

  $\}$

- Jeder endliche Automat ist durch einen DTA simulierbar, also L = L(A) $\in$ P
- Die Klasse PSpace

  Die Menge aller Sprachen L $\subseteq \Sigma^*$, die von DTA entschieden werden können in polynomiellen Speicherplatzbedarf
- PTIME $\subseteq$ PSPACE

  denn in polynomieller Zeit können nur polynomiell viele (neue) Zellen beschrieben werden
- Nondeterministische TA

  Zu einer Kofniguration gibt es mehrer anwendbare Instruktionen

  NDTA akzeptiert w $\in \Sigma^*$, falls es mindestens einen Berechnungsweg w gibt, der mit einem akzeptierenden Zustand endet
- Zeitkomplexität eines NDTA $t_{NDTA}(n)$ = max{
  - p $\in \mathbb{N}|$ Wort w $\in \Sigma^*$ mit l(w) = n, so dass der kürzeste akzeptierende Berechnungsweg für w die Länge p hat
  - $\perp$, falls kein Wort der Länge n akzeptiert wird

  }
- Klasse NP

  es ex. min ein NDTA mit

  1) L_{NDTA} = L

  2) t_{NDTA}(n) $\leq$ p(n)
- P $\subseteq$ NP

  ist L $\in$ P, so ex. nach Def. von P ein DTA mit der Eigenschaft

  1. ist w $\in L_{DTA} \Rightarrow$es existiert (genau) ein akzeptierender Berechnungsweg und seine Länge ist kleiner gleich p(l(w))
  2. ist w $\notin L_{DTA}$, so bricht der Berechnungsweg (für w) nach höchsten p(l(w)) Schritten ab in q $\notin$ F

  DTA ist Spezialfall für NDTA

  NP $\subseteq$ P ist offen (es wird P $\neq$ NP angenommen)
- Jede Sprache aus NP kann in exponentieller Zeit deterministisch entschieden werden

  NDTM auf DTM simulieren: O($2^{p(n)}$)

  NDTA: Min. ein Berechnungsweg: jede Verzweigung ein DTA

  $\Rightarrow$ Höhe des Verzweigungsbaums

- NP $\subseteq$ PSPACE
- TA mit L = $\{a^n b^n | a, b \in \Sigma, n \in \mathbb{N}\}$

  Lesekopf fährt immer zwischen Wortanfang und -ende hin und her und löscht jeweils links a, rechts b, akzeptiert, wenn Band leer
- Halteproblem nicht entscheidbar

  Gibt es Programm, das als Eingabe anderes Programm sowie dessen Eingabewerte erhält und entscheiden kann, ob das zweite Programm terminiert , d.h. nicht weiterläuft
- Spezialfall des Halteproblems: Selbstanwendungsproblem nicht Turing-berechenbar

  Aufruf terminiert genau dann wenn er nicht terminiert
- TM, die nie anhält: (S, $\not{b}$,S,I,R)
- TM f(x,y) = x + y und f(x,y) = x*y
- Turing-Automaten
  - DTA
    w $\in \Sigma^*$ akzeptiert
    Automat bricht bei jeder Eingabe nach endlich vielen Schritten ab
  - nondet. TA
    w $\in \Sigma^*$ akzeptiert und eine Folge anwendbarer Instruktionen bricht ab
    w $\notin \Sigma^*$ Abbruch mit q $\notin$ F oder kein Abbruch

7 Vergleich der Ausdrucksmächtigkeit von Turing-Maschinen und unbeschränkten Registermaschinen

- Jede URM-berechnbare arithmetische Funktion f : $\mathbb{N}^k \dashrightarrow \mathbb{N}^m$ ist auch mittels der Einregistermaschine $URM_{\Sigma',1}$ mit $\Sigma' = \{I, b\}$
- Jede $URM_{\Sigma,1}$-berechnbare Funktion ist auch Turing-berechenbar: $F_{URM} \subseteq F_{TURING}$
- Jede Turing-berechnbare arithmetische Funktion f ist auch URM berechnbar:

  $F_{URM} = F_{TURING} = ...$
  - linker wesentlicher Bandinhalt gespiegelt in Register 1 (w_0)
  - rechter Bandinhalt in Register 2 (w_1)
  - gleiche Operationen wie bei TM

8 Funktionale Programmierung, rekursive Funktionen

- Ziel: Definition einer Klasse von Funktionen, die aus "intuitiv" berechnbaren Grundfunktionen und aus einfachen Operatien generierbar ist
- Menge der primitiv rekursiven Grundfunktionen G_{pr}

  1. v: $\mathbb{N} \rightarrow \mathbb{N}$ mit v(n) = n + 1 (Nachfolgerfunktion)
  2. C_0: $\mathbb{N}^0 \rightarrow \mathbb{N}$ mit $C_0^{(0)}(()) = 0$ (Nullstellige Nullfunktion)
  3. $C_0^{(1)}$: $\mathbb{N}^1 \rightarrow \mathbb{N}$ mit $C_0^{(1)}(n) = 0\ \forall n \in \mathbb{N}$ (einstellige Nullfunktion)
  4. $\Pi_i^{(k)}$: $\mathbb{N}^k \rightarrow \mathbb{N}$ mit $\Pi_i^{(k)}(n_1, ..., n_k) = n_i$ für $1 \leq i \leq k$ (i-te Projektion)

- alle Funktionen URM-berechenbar
- Komposition

  r,s $\in \mathbb{N}_+$ mit

  g: $\mathbb{N}^s \dashrightarrow \mathbb{N}$

  h_i: $\mathbb{N}^r \dashrightarrow \mathbb{N}$ für i = 1,...,s

  f = g $\circ(h_1, ..., h_s)$ die Komposition von g und $h_1, ..., h_s$

  mit f$(n_1, ..., n_r)$ = g$(h_1(n_1, ..., n_r), ..., h_s(n_1, ..., n_r))$, falls $h_i \in \mathbb{N}$ und $\in$ Def(g)

- Sind g, $h_1, h_2, ..., h_s$ URM-berechenbar, so auch g $\circ(h_1, ..., h_s)$
- Primitive Rekursion

  Sei k $\in \mathbb{N}$ und seien g: $\mathbb{N}^k \rightarrow \mathbb{N}$ und h: $\mathbb{N}^{k+2} \rightarrow \mathbb{N}$, dann ist

  1. f(x,0) = g(x)
  2. f(x,y+1) = h(x,y,f(x,y)) für alle x $\in \mathbb{N}^k, y \in \mathbb{N}$

  das zu g und h gehörige primitive Rekusionsschema.

  f ist eindeutige und totale Funktion f: $\mathbb{N}^{k+1} \rightarrow \mathbb{N}$

- Eine arithmetische Funktion f: $\mathbb{N}^k \dashrightarrow \mathbb{N}$ heißt primitiv rekursiv $\in F_{pr}$ g.d.w. f entweder eine primitiv rekursive Grundfunktion ist oder in endlich vielen Schritten durch Komposition und / oder primitive Rekursion erzeugbar ist.

  Die Menge aller primitiv rekusriven Funktionen heißt F_{pr}

  Liste primitiv rekursiver Funktionen: x+y, x-y, k*x, x^y, sign(x), max, min

  $G_{pr} \leq F_{pr}$

  jede konstante Funktin ist primitiv rekursiv

  f(x,y) = $\lfloor x/y \rfloor$ ist nicht in F_{pr}, da sie nicht total ist

- Beispiel primitive Rekursion

  $f_+ = \text{pr}(\Pi_1^{(1)}, v \circ \Pi_3^{(3)})$

  f(x,0) = g(x) = x

  f(x,y+1) = h(x,y,f(x,y)) = v ∘ f(x,y) = f(x,y) + 1

  ⇒ f(x,1) = h(x,0,f(x,0)) = v ∘ f(x,0) = 1 + x

  $f_\bullet = \text{pr}(C_0^{(1)}, \Pi_1^{(3)} + \Pi_3^{(3)})$

  f(x,0) = g(x) = 0

  f(x,y+1) = h(x,y,f(x,y)) = x + f(x,y) = x + x*y = x*(y+1)

- Operation (Unbeschränkte Minimalisierung)

  f_+: $\mathbb{N}^{k+1} \dashrightarrow \mathbb{N} \Rightarrow \mu(f) : \mathbb{N}^k \dashrightarrow \mathbb{N}$

  mit $\mu(f_+)(x) = \begin{pmatrix} y & \text{falls f(x,y) = 0 und f(x,i)} \in \mathbb{N} \\ \perp & sonst. \end{pmatrix}$

  $\mu(f)$ heißt die unbeschränkte Minimalisierung. Eine Funktion f: $\mathbb{N}^k \dashrightarrow \mathbb{N}$ heißt μ-rekursiv, falls sie aus primitiv rekursiven Funktionen durch endlich häufige Anwendung der unbeschränkten Minimalisierung erzeugbar ist

- Churchsche These (Klasse der intuitiv berechenbaren Funktionen)

  $F_{pr} \subset F_{\mu r} = F_{URM} = F_{TURING} = F_{MA} = F_{GOTO}$

- $F_{pr} \subseteq F_{URM}$
  1. $G_{pr} \subseteq F_{URM}$ klar
  2. Komposition
  3. primitive Rekursion
     g: $\mathbb{N}^k \to \mathbb{N}$, h: $\mathbb{N}^{k+2} \to \mathbb{N}$ URM-berechenbar, so auch f = pr(g,h)

- $F_{URM} \neq F_{pr}$

  Nein, da F_{pr} nur totale Fkt. enthält, z.B.: ||(A1)1;||(n) = $\begin{Bmatrix} 0 & \text{n = 0} \\ \perp & \text{sonst.} \end{Bmatrix}$

- $F_{pr} \subset (\neq) F_{\mu r}$

  $\mu(f_+) \notin F_{pr}$, aber $\mu(f_+) \in F_{\mu r}$

  in $F_{\mu r}$ gibt es echt partielle Funktionen

- $F_{\mu r} \subseteq F_{URM}$ (einfach)

  z.Z. f: $\mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ URM-berechenbar, so auch μ(f)

  $|\text{P}|(0, x_1, ..., x_k, y, 0, 0, 0, ...) = \begin{Bmatrix} (0, x_1, ..., x_k, y, f(x,y), 0, 0, ...) & \text{falls (x,y)} \in \text{Def(f)} \\ \perp & \text{sonst.} \end{Bmatrix}$

  |P'| = $((Sk+2)k+2; P_i; (Ak+1))k+2; (S_1)1; MOV(k+1, 1);$

  berechne für i = 0,1,... P(x,i) und prüfe ob Ergebnis = 0

9 Nicht-sequentielle Programme, allegemeine Kontrollstrukturen, speziell: GOTO-Programme (Spaghetti-Programme)

- Marken (labels: L)
- Variablen X = $\{x_1, ..., x_n\}$
- elementare Anweisungen A_e mit
  $A_e = \{l, x_i = x_i + 1, l' | l, l' \in L$ und $x_i \in X\} \cup \{l, x_i = x_i - 1, l' | l, l' \in L$ und $x_i \in X\} \cup \{l, if x_i = 0, l', l'' | l, l', l'' \in L$ und $x_i \in X\}$
- s $\in A_e$ und s = l,..., so l die (Anfangs-)Marke von s: l(s) = l
  ist s = l, x_i = $x_i \pm 1$, l', so heißt l' die Folgemarke von s
  ist s = l,if x_i = 0, l', l'' so heißt l', l'' Verzweigungsmarken von s, x_i Variablen von s
- Verbundanweisung a = $s_1; s_2; ...s_q$ (s Anweisung)
- $NP_{GOTO} = \{a$ | a ist Verbund-Anweisung$\}$
- $P_{GOTO} = \{a = s_1; s_2; ...s_q$ | l(s_i) $\neq$ l(s_j) für i $\neq$ j (i,j = 1,...,q) $\}$
- X(p) = X_p Menge aller zugelassenen Variablen
- L_p Menge aller Marken von P
- L_p^a Menge aller Anfangsmarken von P
- Beispielprogramm
  P X_p = $\{x_1, x_2\}$
  a_p = s_1, s_2, s_3, s_4 mit
  s_1: 0,if x_1 = 0,1,2
  s_2: 2,if x_2 = 0,3,4
  s_3: 4,x_1 = x_1-1,5
  s_4: 5,x_2 = x_2-1,0
  L_p = $\{0, 1, 2, 3, 4, 5\}$, L_p^a = $\{0, 2, 4, 5\}$
  Semantik: Prüfe Variablen x_1 und x_2 auf 0, es wird solange vermindert bis eine 0
- $F_{URM} \subseteq F_{GOTO}$ (quasi 1:1 Umsetzung der Programme)
  Ai $\rightarrow X_p = \{x_0, x_i\}$, $a_p = s_1$ mit $s_1 : l, x_i = x_i + 1, l'$ (Si analog)
  M1;M2: binden mittels Marken-Translation
  (M)i: 0,if $x_i = 0, l, 1$
- $F_{GOTO} \subseteq F_{\mu r}$ (am schwierigsten: komponentenweise primitive und μ-Rekursivität)