Automaten und Formale Sprache & Berechenbarkeit und Komplexität ¬ TLDR

Thomas Schimper

<2017-10-16 Mo>

Inhaltsverzeichnis

1 Automaten und Formale Sprachen

1.1 Was ist eine Sprache?

Eine Menge Σ heißt Alphabet, falls gilt: $|\Sigma| < \infty$ und $\Sigma \neq \emptyset$. Die Elemente eines Alphabetes heißen Buchstaben oder Zeichen.

Eine Sprache L über (Σ) ist eine Teilmenge des von der Menge Σ frei erzeugeten Monoids ($L \subseteq \Sigma^*$). Die Elemente einer Sprache heißen auch Wörter.

1.2 Halbgruppe

Eine Halbgruppe ist eine Struktur $(H, \circ)$, wobei $\circ$ eine assoziative Verknüpfung auf H ist. Ein Element $\in H$ heißt neutrales Element $\Leftrightarrow x \circ e = e \circ x = x$ für alle $x \in H$.

1.2.1 Nachweis

- Definition der Operation
- Abgeschlossenheit
- Assoziativität

1.3 Monoid

Eine Struktur $(M, \circ, e)$ heißt Monoid, wenn $(M, \circ)$ eine Halbgruppe ist und $e \in M$ das neutrale Element dieser Halbgruppe.

1.4 Eigenschaften zweistelliger Operationen

- Die Konkatenation auf X^+ ist im Allgemeinen nicht kommutativ
- Die Konkatenation ist im Allgemeinen nicht idempotent

1.5 Morphismen

ist eine Strukturerhaltende Abbildung. Sind $(H, \circ)$ und $(G, \circ)$ Halbgruppen, so ist eine Abbildung $h : H \to G$ ein Morphismus

$$\forall x, y \in H : h(x \circ y) = h(x) \square h(y) \tag{1}$$

1.5.1 Satz

Sind $(H, \circ)$ und $(G, \square)$ Halbgruppen und $h : H \to G$ ein Morphismus, so ist

$$(\{h(x) | x \in H\}, \square) \tag{2}$$

eine Halbgruppe. Besitzt $(H, \circ)$ darüber hinaus ein neutrales Element $e \in H$, so ist $h(e)$ neutrales Element von $(\{h(x) | x \in H\}, \square)$

1.6 Reguläre Sprachen

Eine Sprache $L \subseteq \Sigma^*$ heißt regulär $\Leftrightarrow$ es gibt ein endliches Monoid $(M, \circ, e)$ einen Monoidmorphismus $h : (\Sigma, \cdot, \lambda) \to (M, \circ, e)$, sowie eine endliche Menge $F \subseteq M$ gibt mit:

$$L = \{w \in \Sigma | h(w) \in F\} \tag{3}$$

1.7 DEA

$$A = (Q, \Sigma, \delta, q_0, F) \tag{4}$$

$$Q : \text{endliche Menge von Zuständen} \tag{5}$$

$$\Sigma : \text{Eingabezeichen} \tag{6}$$

$$\delta : Q \times \Sigma \to Q \text{ totale Überführungsfunktion} \tag{7}$$

$$q_0 \in Q : \text{Anfangszustand} \tag{8}$$

$$F \subseteq Q : \text{Endzustände} \tag{9}$$

1.7.1 Konfigurationen

Ein Element aus $C = Q \times \Sigma^*$ heißt Konfiguration von A. Definiere eine Binärrelation $\vdash_A$ auf C durch

$$(q, w) \vdash_A (q', w') \Leftrightarrow a \in \Sigma : w = aw' \text{ und } q' = \delta(q, a) \tag{10}$$

- $\vdash_A^n$ beschreibt n Schritte von A"
- $L(A) = \{w \in \Sigma^* | q \in F : (q_0, w) \vdash_A^* (q, \lambda)\}$

Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA

$$\hat{\delta} : Q \times \Sigma^* \rightarrow, (q, w) \begin{cases} q, & w = \lambda \\ \hat{\delta}(\delta(q, a), w'), & w = aw', a \in \Sigma \end{cases} \tag{11}$$

1. Lemma Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Seien $p, q \in Q$ und $w \in \Sigma^*$ beliebig. Dann gilt:

$$(p, w) \vdash_A^* (q, \lambda) \Leftrightarrow \hat{\delta}(p, w) = q \tag{12}$$

1.8 Schlingenlemma

Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Es sei $q \in Q$ und sei $X = \{a \in \Sigma : (q, a) \vdash_A (q, \lambda)\}$. Dann gilt: $X^* \subseteq \{w \in \Sigma^* | (q, w) \vdash_A^* (q, \lambda)\}$

1.9 Regularität

- Wird $L \subseteq \Sigma^*$ von einem DEA erkannt, so ist L regulär.
- Ist $L \subseteq \Sigma^*$ regulär, so wird L von einem DEA erkannt.

1.10 Eigenschaften

- DEA-Sprachen sind komplementabgeschlossen

1.11 NEA

Ein nichtdeterministischer Endlicher Automat oder NEA wird beschrieben durch ein Quintupel

$$A = (Q, \Sigma, \delta, Q_0, F) \tag{13}$$
$$Q: \text{ endliche Menge von Zuständen} \tag{14}$$
$$\Sigma: \text{ endliche Menge von Eingabezeichen} \tag{15}$$
$$\delta \subseteq Q \times \Sigma \times Q: \text{ Überführungsfunktion} \tag{16}$$
$$Q_0 \subseteq Q: \text{ Anfangszustände} \tag{17}$$
$$F \subseteq Q: \text{ Endzustände} \tag{18}$$

1.11.1 Konfigurationen

Ein Element aus $C = Q \times \Sigma^*$ heißt Konfiguration von A. Definiere eine Binärrelation $\vdash_A$ auf C durch

$$(q, w) \vdash_A (q', w') \Leftrightarrow a \in \Sigma : w = aw' \text{ und } \delta \ni (q, a, q') \tag{19}$$

$\vdash_A^n$ n Schritte von A

$$L(A) = \{w \in \Sigma^* | q_0 \in Q_0, q \in F : (q_0, w) \vdash_A^* (q, \lambda)\} \tag{20}$$

1.11.2 Eigenschaften

- Jede endliche Sprache ist NEA-Sprache
- NEA ist unter Vereinigung abgeschlossen
- NEA = DEA
- Zu jedem NEA mit λ -Übergängen gibt es einen äquivalenten NEA mit λ -Übergängen, der nur Anfgangs- und nur einen Endzustand besitzt; der Anfangszustand hat nur ausgehende Kanten und der Endzustand nur eingehende Kanten.

1.12 Eigenschaften Regulärer Sprachen

- REG ist gegen Komplementbildung abgeschlossen.

1.13 Monoidkonstruktionen

Es seien (M, o, e) und $(N, \Box, 1)$ Monoide. Dann kann man die Menge $M \times N$ zu einem Monoid machen durch komponentenweises Anwenden der Operationen; definiere daher

$$(m, n)[\circ, \Box](m', n') := (m \circ m', n \Box n') \tag{21}$$

- $(M \times N, [\circ, \Box], (e, 1))$ ist ein Monoid, das Produktmonoid
- Sind $h_M : (X, \Delta, I) \to (M, \circ, e)$ und $h_N : (X, \Delta, I) \to (N, \Box, 1)$ Monoidmorphismen, so auch der Produktmorphismus

$$h_M \times h_N : X \to M \times N, x \mapsto (h_M(x), h_N(x)) \tag{22}$$

1.14 Mengenoperationen

- Ist f eine k -stellige Mengenoperation, so ist REG gegen f abgeschlossen
- REG ist gegen Konkatenation abgeschlossen

1.15 Potenzen in Kleene-Sternen

Ist $(M, \circ, e)$ ein Monoid, so können wir induktiv die n -te Potenz eines Elementes $x \in M$ rekursiv festlegen durch : $x^0 = e$ sowie $x^{n+1} = x^n \circ x$ für $n \in \mathbb{N}$ Dann kann man $A^+ = \bigcup_{n \geq 1} A^n$ und $A^* = \bigcup_{n \geq 0} A^n$ Somit ist auch L^+ und L^* (*Kleene-Stern*) für $L \subseteq \Sigma^*$ definiert.

$L^+(L^*)$ ist die (das) durch L bezüglich der Konkatenation erzeugte Halbgruppe (Monoid)

1.15.1 Eigenschaften

- REG ist gegen Kleene-Stern abgeschlossen

1.16 Reguläre Ausdrücke

1. $\emptyset$ und a sind RA (über Σ) für jedes $a \in \Sigma$
2. Ist R ein RA (über Σ), so auch $(R)*$.
3. Sind R_1 und R_2 RAs (über Σ), so auch $R_1 R_2$ und $(R_1 \cup R_2)$
4. $L(\emptyset) = \emptyset; L(a) = \{a\}$

5. Ist R ein RA, setzte $L((R)*) = (L(R))^*$
6. Sind R_1 und R_2 RA, setze $L(R_1R_2) = L(R_1) \cdot L(R_2)$ und $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
7. Jede RA-Sprache ist regulär
8. Jede reguläre Sprache ist durch einen RA beschreibbar

1.17 Pumping-Lemma

Zu jeder regulären Sprache L gibt es eine Zahl $n > 0$, sodass jedes Wort $w \in L$ mit $\ell(w) \geq n$ als Konkatenation $w = xyz$ dargestellt werden kann mit geeigneten x, y, z mit folgenden Eigenschaften:

1. $\ell(y) > 0$
2. $\ell(xy) \leq n$
3. $\forall i \geq 0 : xy^iz \in L$

Wichtig: Die Umkehrung gilt nicht!

1.17.1 Ablauf

1. Wir vermuten, eine Sprache L ist nicht regulär.
2. Im Widerspruch zu unserer Annahme nehmen wir an, L wäre regulär. Dann gibt es eine Pumping-Konstante n
3. Wir wählen ein geeignetes, hinreichend langes Wort $w \in L \quad (\ell(w) \geq n)$ Da wir vermuten, das L nicht regulär ist, kann L auch unendlich lang sein und wir finden immer ein Wort $w \in L$ mit $\ell(w) \geq n$.
4. Wir diskutieren alle möglichen Zerlegungen $w = xyz$ mit $\ell(y) > 0$ und $\ell(xy) \leq n$ und zeigen für jede solche Zerlegung, dass es ein $i \geq 0$ gibt, sodass $xy^iz \notin L$ gilt.

1.18 Spiegeloperationen

$$\lambda^R = \lambda \tag{23}$$

$$\text{für } w = va \text{ mit } v \in \Sigma^*, a \in \Sigma \tag{24}$$

$$\text{definiere } w^R := a(v^R) \tag{25}$$

- Die regulären Sprachen sind unter Spiegelung abgeschlossen.

1.19 Äquivalenzrelationen

1.19.1 Definition

1. $R^0 = \Delta_X \subseteq R$ (Reflexivität)
2. $R^2 = R \circ R \subseteq R$ (Transitivität)
3. Mit $R^{-1} = \{(y,x)|(x,y) \in R\}$ gilt $R^{-1} \subseteq R$ (Symmetrie)

Eine ÄR auf X induziert eine Partition von X in ÄK $[x]_R = \{y \in X|xRy\}$

1.19.2 Beispiele

1. Es sei $h : (\Sigma^*, \cdot, \lambda) \to (M, \circ, e)$ ein Monoidmorphismus. Dann ist definiert $x \equiv_h y \Leftrightarrow h(x) = h(y)$. $x \equiv_h y$ ist eine ÄR auf Σ^*
2. Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DEA. Definiere u\equiv$_A$ v$\Leftrightarrow$q$\in$ Q:((q$_{0,u}$)$\vdash_A^*$(q,λ))$\wedge$((q$_{0,v}$)$\vdash_A^*$ (q,λ))

$u \equiv_A v$ ist eine ÄR auf Σ^*

- Es sei $L \subseteq \Sigma^*$. L trennt zwei Wörter $x, y \in \Sigma^* \Leftrightarrow |\{x, y\} \cap L| = 1$

Zwei Wörter u und v heißen kongruent modulo L ($u \equiv_L v$), wenn für jedes beliegige Wort w aus Σ^* die Sprache L die Wörter uw und vw nicht trennt, d.h wenn gilt:

$$(\forall w \in \Sigma^*)(uw \in L \Leftrightarrow vw \in L) \tag{26}$$

Es sei $L \subseteq \Sigma^*.u, v \in \Sigma^*$ heißen syntaktisch kongruent modulo L ($u \equiv_L^{synt} v$ gdw. $\forall x, y \in \Sigma^* : (xuy \in L \Leftrightarrow xvy \in L)$

- Für jede Sprache L ist $u \equiv_L^{synt} v$ eine Kongruenzrelation

1.20 Myhill-Nerode Äquivalenz

Für jede Sprache $L \subseteq \Sigma^*$ ist $\equiv_L$ eine ÄR

- $\equiv_L$ heißt auch Myhill-Nerode Äquivalenz.
- Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn es nur endlich viele ÄK bezüglich $\equiv_L$

1.20.1 Eigenschaften

- $\equiv_L$ ist sogar Rechtskongruenz, d.h. aus $u \equiv_L v$ folgt für bel. Wörter $x \in \Sigma^* : ux \equiv_L vx$
- Es sei $L \subseteq \Sigma^*$ regulär, d.h. L ist durch ein endliches Monoid $(M, \circ, e)$ einen Monoidmorphismus $h : \Sigma^* \Leftrightarrow M$ und eine endliche Menge $F \subseteq M$ beschreiben. Dann gilt: Falls $u \equiv_h v$, so $u \equiv_L v$.
- Ist L regulär, so hat $\equiv_L$ nur endlich viele ÄK.

1.21 Minimalautomat

Ist L regulär, so ist $A_{min}(L)$ der L akzeptierende DEA mit der kleinsten Anzahl von Zuständen. Der Minimalautomat ist "bis auf Isomorphie“ eindeutig bestimmt.

1.22 Automatenmorphismen

Es seien $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ DEAs. Eine Funktion $f : Q_1 \to Q_2$ heißt Automantenhomomorphismus von A_1 nach A_2 gdw:

- Für alle $a \in \Sigma$ und für alle $q \in Q_1$ gilt $f(\delta_1(q, a)) = \delta_2(f(q), a)$
- $f(q_{01}) = q_{02}$
- Für alle $q \in Q_1$ gilt: $q \in F_1 \Leftrightarrow f(q) \in F_2$

1.23 Logische Formeln

- $(F \vee G)$ steht für $\neg(\neg F \wedge \neg G)$ Disjunktion von F und G
- $(F \to G)$ steht für $(\neg F \vee G)$ Implikation
- $(F \leftrightarrow G)$ steht für $((F \to G) \wedge (G \to F))$ Äquivalenz.
- Für jede Formel F gilt: $F \equiv (F \wedge F)$
- (Absorption) $F \equiv (F \wedge (F \vee G))$

1.24 Syntax von Büchis Logik erster Stufe über Alphabet Σ

- atomare Formeln sind Formeln erster Stufe
- Sind ϕ und ψ Formeln erster Stufe, so auch $(\neg\phi), (\phi \wedge \psi), (\phi, \vee\psi)$
- Ist ϕ eine Formel erster Stufe und ist x eine Variable, so sind auch $(x\phi)$ und $(\forall x\phi)$ Formeln erster Stufe

1.25 Wann ist ein DEA A nicht minimal?

- Wenn es nicht-erreichbare Zustände gibt
- Wenn es Zustände $q \neq q'$ gibt mit

$$\forall w \exists p, p' \in Q : |\{, p'\} \cap F| \neq 1 \Rightarrow ((q, w) \vdash_A^* (p, \lambda) \Leftrightarrow (q', w) \vdash_A^* (p', \lambda)) \tag{27}$$

d.h. q und q' sind nicht trennbar, sondern äquivalent.

1.26 Konstruktion des Minimalautomaten

Definiere zu $A = (Q, \Sigma, \delta, q_0, F)$ neuen Automaten

$$A_{[]} = (Q_{[]}, \Sigma, \delta_{[]}, [q_0], F_{[]}) \text{ mit} \tag{28}$$

- Anfangszustand $[q_0]$
- Endzuständen $F_{[]} := \{[q] | q \in F\}$
- Übergangsfunktion $\delta_{[]}([q], a) := [\delta(q, a)]$

f ist Automatenmorphismus; und damit gilt $L(A) = L(A_{[]})$. $A_{[]}$ isomorph zum Minimalautomaten von A,also $A_{min}(L(A))$

1.27 Leerheitsproblem

Die vom Automaten beschriebene Sprache ist leer gdw. E keine Endzustände enthält.

1.28 Teilmengen- und Äquivalenzproblem

$$L(A) \subseteq L(A') \Leftrightarrow L(A) \setminus L(A') = \emptyset \tag{29}$$

1.29 Endlichkeitsproblem

Ist $L(A)$ unendlich, so gibt es einen Zustand q, einen (evtl. leeren) Weg vom Anfangszustand q_0 nach q, einen nicht-leeren Weg von q nach q und einen (evtl. leeren) Weg von q zu einem Endzustand. Die Umkehrung gilt sogar trivialerweise. Bezeichnet R_A die 1-Schritt-Zustandserreichbarkeitsrelation, so berechne E': die Menge der Zustände, die sowohl erreichbar als auch co-erreichbar sind. Dann gilt: $L(A)$ ist unendlich gdw. $\exists q \in E' : (q, q) \in R_A^+$

1.30 Mustersuche

$t \in \Sigma^*$ enthält das Muster $p \Leftrightarrow t \in \Sigma^* p \Sigma^*$

1.31 Hilfsbegriffe

- U heißt Teilwort von $x \in \Sigma^* \Leftrightarrow x \in \Sigma^* \{u\} \Sigma^*$
- u heißt Präfix von $x \in \Sigma^* \Leftrightarrow x \in \{u\} \Sigma^*$
- u heißt Suffix oder Endwort von $x \in \Sigma^* \Leftrightarrow x \in \Sigma^* u$
- Ein Teilwort/Präfix/Suffix u von x heißt echt gdw. $\ell(u) < \ell(x)$
- Ein echtes Teilwort u von x, das sowohl Präfix als auch Suffix von x ist, heißt Rand (der Breite $\ell(u)$) von x

1.32 kontextfreie Grammatik

- Σ ist das Terminalalphabet
- N ist das Nonterminalalphabet mit $N \cap \Sigma = \emptyset$
- $R \subset N \times (\Sigma \cup N)^*$ ist das Alphabet der Regeln
- $S \in N$ ist das Startsymbol oder Anfangszeichen

Ein Wort über dem Gesamtalphabet $(\Sigma \cup N)$ heißt auch Satzform

1.33 Ableitungsmechanismus

1-Schritt-Ableitungsrelation $\longrightarrow_G$ zwischen zwei Satzformen u, v einer kfG $G : u \longrightarrow_G v$ gdw. es gibt Regel $A \to y$, sodass u und v wie folgt zerlegt werden können: $u = xAz$ und $v = xyz$ hierbei sind x und z wiederum Satzformen.

Die von einer kfG G erzeugten oder abgeleitete Sprache ist gegeben durch

$$L(G) := \{w \in \Sigma^* | S \longrightarrow^* w\} \tag{30}$$

1.34 Rechtslinear und regulär

Eine kfG heißt rechtslinear gdw. alle rechten Regelseiten haben die Form zA oder z, wobei z ein Terminalwort ist und A ein Nichtterminalzeichen

- L ist regulär gdw. L ist durch rechslineare kfG erzeugbar.

1.35 Der Satz von Doner, Thatcher und Wright

Es sei $L \subseteq \Sigma^*$ eine Sprache. L ist kontextfrei gdw. $L = yield(B(A))$ für einen endlichen Baumautomaten A.

1.36 Weitere Eigenschaften

- Zu der kfG G gibt es eine kfG G' mit $L(G) = L(G')$ bei der Regeln, die Terminalzeichen enthalten, alleinig von der Form $A \to a$ sind mit $a \in \Sigma$
- Zu jeder kfG G gibt es eine kfG G' ohne λ -Regeln mit $L(G) \setminus \{\lambda\} = L(G')$

1.37 Algorithmus zur Generierung G' aus G

1.37.1 Problem

Berechne $N^\lambda = \{A \in N | A \Rightarrow_G^* \lambda\}$

$$N_0^\lambda : = \{A \in N | A \to_G \lambda\} \tag{31}$$

$$N_k^\lambda : = \{A \to_G X \wedge X \in (N_{k-1}^\lambda)^*\} \text{ für } k > 0 \tag{32}$$

$$A \in N^\lambda \Leftrightarrow A \in \bigcup_{k=0}^{\infty} N_k^\lambda =: N_*^\lambda \Leftrightarrow A \in N_{|N|}^\lambda \tag{33}$$

1.37.2 Algorithmus für N^λ

- $E := N_0^\lambda$
- `WHILE` $(\exists A \to B_1 \cdots B_n \in R : A \notin E \wedge (\forall i \in \{1, \cdots, n\} : B_i \in E))$ `DO`: $E := E \cup \{A\}$

Der Algorithmus findet Wege in einem Graphen mit Knotenmenge 2^N.

1.37.3 Algorithmus für λ -Elimination

- Berechne N^{λ}; dies definiert den Morphismus h
- Setze $R^{'} := \emptyset$
- `FORALL` $A \to w \in R$ `DO`: Bestimme $R^{\lambda}(A \to w) = \{A \to u | u \in h(w) \wedge u \neq \lambda\}$

1.38 Elimination von Kettenregeln

Zu jeder kfG G gibt es eine kfG G' ohne Kettenregeln mit $L(G) = L(G')$

1.38.1 Algorihtmus für $N(A)$

Wegeproblem im zu H gehörigen Graphen

$$N(A) = \{B \in N | (B, A) \in (R \cap N \times N)^*\} \tag{34}$$

1.38.2 Algorithmus für Elimination von Kettenregeln

- Berechne $N(A)$ für alle A
- Setze $R^{'} := \emptyset$
- `FORALL` $A \to w \in R \setminus (N \times N)$ `DO` Bestimme $R(A) = \{B \to w | B \in N(A)\}$ $R' := R' \cup R(A)$

1.39 kontextfreie Grammatik in Chomsky-Normalform

ein Quadrupel $G = (\Sigma, N, R, S)$:

- Σ ist das Terminalalphabet
- N ist das Nonterminalalphabet
- $R \subset (N \times N^2) \cup (N \times \Sigma)$ ist das Alphabet der Regeln oder Produktionen.
- $S \in N$ ist das Startsymbol oder Anfangszeichen

1.40 Chomsky-Normalform

Zu jeder kfg G gibt es eine kfG G' in Chomsky-Normalform mit $L(G) \setminus \{\lambda\} = L(G')$.

1.41 Algorithmus zur Vermeidung zu langer rechter Regelseiten

1.41.1 Grundidee

Einfügen von SZwischenzuständen"beim "Buchstabieren"

1.41.2 Eingabezeichen

Regelmenge R

1.41.3 Der Algorithmus

`WHILE` $(A \to w \in R : \ell(w) > 2)$ `DO`

- Stelle $w = Bu$ dar.
- Erzeuge neues Nichtterminalzeichen X
- Ersetze $A \to w$ durch $A \to BX$ und $X \to u$ in R.

Warum terminiert das Verfahren?

1.42 Eigenschaften

- Es gibt einen Algorithmus, der zu jeder vorgelegte kfG $G = (\Sigma, N, R, S)$ und jedem $W \in \Sigma^*$ entscheidet, ob $w \in L(G)$ gilt.

Cocke, Younger und Kasami haben gezeigt, wie man die Komplexität des beschriebenen Verfahrens durch dynamisches Programmieren erheblich vermindern kann.

1.43 CYK-Algorithmus

Ist eine kfG G in CNF fixiert, so lässt sich die Frage "$w \in L(G)$ïn einer Zeit beantworten, die sich durch ein kubisches Polynom in $\ell(w)$ abschätzen lässt.

1.44 Eigenschaften

- KF ist unter Vereinigung abgeschlossen.
- KF ist unter Konkatenation abgeschlossen.
- KF ist unter Kleene-Stern abgeschlossen.

- KF ist unter Durchschnitt mit regulären Sprache abgeschlossen.
- KF ist nicht unter Durchschnitt abgeschlossen.
- KF ist nicht uner Komplementbildung abgeschlossen.

1.45 Binärer Wurzelbaum

ist gegeben durch ein Tripel $B = (V, \phi, r)$ mit ausgezeichneter Wurzel $r \in V$ und einer Vater-Abbildung $\phi : V \setminus \rightarrow V$ mit der Eigenschaft

$$\forall v \in V : \#\{u \in V | \phi(u) = v\} \leq 2 \tag{35}$$

1.45.1 Lemma

Der Ableitungsbaum eines jeden von einer kontextfreien Grammatik in CNF akzeptierten Wortes kann als binärer Wurzelbaum aufgefasst werden.

1.45.2 Höhe des Baums

ist gegeben durch

$$h(B) = max_{v \in V \setminus \{r\}}\{k \in \mathbb{N} | \phi^k(v) = r\} \tag{36}$$

1. Lemma
   - Hat ein binärer Wurzelbaum B mehr als 2^h, so gilt: $h(B) > h$
   - Ist $G = (\Sigma, N, R, S)$ eine kfG in CNF und ist $w \in L(G)$ mit $\ell(w) > 2^{\#N}$ so gilt für jeden Ableitungsbaum von w bzgl. G, dass es einen Weg von S zu einem Blatt gibt, auf dem mehr als \N viele Nichtterminalzeichen ersetzt werden.
   - Auf besagtem Weg von der Wurzel zum Blatt im Ableitungsbaum von w finden wir also nach dem Schubfachprinzip zwei Regelanwendungen $A \rightarrow v$ und $A \rightarrow u$ mit gleicher linker Seite

1.46 Erweiterte CNF

Jede $L \in KF$ lässt sich beschreiben durch eine kfG $G = (\Sigma, N, R, S)$ mit Regeln der Form $N \times ((NN) \cup (\Sigma))$. Zusätzlich darf eine Regel $S \rightarrow \lambda$ existieren, wobei dann gefordert ist, dass S in keiner rechten Regelseite vorkommt.

1.47 Pumping-Lemma für KF

Zu jeder kfS L gibt es eine Konstante $n > 0$, sodass jedes Wort $w \in L$ mit $\ell(w) \geq n$ als Konkatenation $w = uvxyz$ dargestellt werden kann mit geeigneten u, v, x, y, z mit folgenden Eigenschaften

- $\ell(v) > 0$ oder $\ell(y) > 0$
- $\ell(vxy) \leq n$
- $\forall i \geq 0 : uv^i xy^i z \in L$

1.48 Entscheidbarkeitsfragen

- Das Leerheitsproblem ist für kfG entscheidbar.
- Das Endlichkeitsproblem ist für kfG entscheidbar.
- Zu jeder Sprache $L \in KF$ gibt es eine Konstante $n > 0$, sodass gilt: L ist unendlich gdw. es gibt ein Wort $t \in L$ mit $n \leq \ell(t) < 2n$
- Zu jeder Sprache $L \in KF$ gibt es eine Konstante $n > 0$, sodass gilt: L ist unendlich gdw. es gibt ein Wort $t \in L$ mit $n \leq \ell(t) < 2n$

1.49 Phasenstruktur-Grammatik

ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit

- Σ ist ein Terminalalphabet
- N ist das Nonterminalalphabet
- $R \subset (\Sigma \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ ist das Alphabet der Regeln, wobei $A \in N$ und $x, y, v \in (\Sigma \cup N)^*$ auch linke Seite, bzw. rechte Seite der Regel heißen.
- $S \in N$ ist das Startsymbol oder Anfangszeichen

Ein Wort über dem Gesamtalphabet $(\Sigma \cup N)$ heißt auch Satzform

1.50 Ableitungsmechanismus

1.50.1 1-Schritt-Ableitungsrelation $\longrightarrow_G$

zwischen zwei Satzformen u, v einer Grammatik G: $u \longrightarrow_G v$ gdw. es gibt Regel $\alpha \to y$, sodass u und v wie folgt zerlegt werden können: $u = x\alpha z$ und $v = xyz$. Hierbei sind x und z wiederum Satzformen.

1. formal

$$\forall u, v \in (\Sigma \cup N)^* : u \longrightarrow_G v \Leftrightarrow (\exists x, z \in (\Sigma \cup N)^* \exists (\alpha \to y) \in R : u = x\alpha z \wedge v = xyz) \tag{37}$$

2. abgeleitete Sprache

   Die von einer Grammatik G erzeugte oder erzeugte Sprache ist gegeben durch

$$L(G) := \{w \in \Sigma^* | S \longrightarrow^* w\} \tag{38}$$

1.51 Monotonie

Eine Grammatik heißt monoton oder nichtverkürzend gdw. für alle Regeln gilt, dass die rechte Regelseite nicht kürzer als die linke ist.

Eine monotone Grammatik heißt kontextsensitiv, wenn für alle Regeln von der Form $\zeta A\nu \to \zeta w\nu$ sind für irgendein $A \in N$

Bei kontextfreien Regeln gilt

$$\zeta = \nu = \lambda \tag{39}$$

1.52 Sprachgleichheit

Für $L_k := \{w | a^k Y_a b^k c^k \Rightarrow^{k+2} w\}$ gilt:

$$L_k = \{a^{k+1} b^{k+1} Y_a c^{k+1} b^{k+1} c^{k+1}\} \text{ für jedes } k \geq 1 \tag{40}$$

- Aus $a^k b^k Y_a c^k$ ergibt sich nach k Ableitungsschritten $a^k Y_a b^k c^k$ (und keine andere Satzform)
- Die einzigen in den beiden Lemmata beobachteten Satzformen, die nur aus Terminalzeichen bestehen, sind explizit im ersten Lemma angegeben.

1.53 Eine Normalform für monotone Grammatiken

Zu jeder monotonen Grammatik gibt es eine äquivalente, bei der alle Regeln mit Terminalzeichen a von der Form $A \to a$ sind.

1.54 wichtige Sätze

- $KF = MON$
- $KF \subsetneq KS$

1.55 Abschlusseigenschaften bei Typ-0/1 Sprachen

1.55.1 Vereinigung/Konkatenation

Standardkonstruktion wie Typ-2 Achtung: Disjunkte Pseudoterminalalphabete bei Konkatenation

1.55.2 Durchschnitt

NF -Grammatiken G_1 und G_2 werden simuliert auf Produktalphabet"$N_1 \times N_2$. Nur für Paare (X_a, X_a) gibt es terminierende Regeln $(X_a, X_a) \to a$.

1.55.3 Komplement

Typ-1 Sprachen sind abgeschlossen; die entsprechende Technik des induktiven Zählens ist Gegenstand der Komplexitätstheorie. Typ-0 Sprachen sind nicht von abgeschlossen (Satz von Post)

1.56 Die Chomsky-Hierarchy

Grammatik	Regeln	Sprachen	Entscheidbarkeit	Automanten	Abgeschlossenheit
Typ-0 Beliebig formale Grammatik	$\alpha \to B$	rekursiv aufzählbar	-	Turingmaschine	$\circ, \cap\cup, *$

Fortsetzung nächste Seite

Fortsetzung von vorheriger Seite

Grammatik	Regeln	Sprachen	Entscheidbarkeit	Automanten	Abgeschlossenheit
Typ-1 Kontext-sensitive Grammatik	$\alpha A\beta \rightarrow \alpha\gamma\beta$	kontextsensitiv	Wortproblem	linear platzbeschränkte nichtdeterministische Turingmaschine	$\complement, \circ, \cap, \cup, *$
Typ-2 Kontextfreie Grammatik	$A \rightarrow \gamma$	kontextfrei	Wortproblem, Leerheitsproblem, Endlichkeitsproblem	nichtdeterministischer Kellerautomat	$\circ, \cup, *$
Typ-3 Reguläre Grammatik	$A \rightarrow aB$	regulär	Wortproblem, Leerheitsproblem, Endlichkeitsproblem, Äquivalenzproblem	Endlicher Automat	$\complement, \circ, \cap, \cup, *$

1.56.1 Satz

$REG \subsetneq KF \subsetneq KS \subsetneq RA$

1.57 Turingmaschine

Eine Turingmaschine ist durch ein 7-Tupel beschrieben:

$$TM = (S, E, A, \delta, s_0, \square, F) \tag{41}$$

Dabei bedeuten:

- $S = \{s_0, s_1, \cdots, s_n\}$ ist die Menge der Zustände
- $E = \{e_1, e_2, \cdots, e_n\}$ ist das endliche Eingabealphabet
- $A = \{a_0, a_1, \cdots, a_m\}$ das endliche Arbeitsalphabet (auch Bandalphabet genannt), es sei dabei $E \subset A$.
- s_0 der Startzustand

- $a_0 = \square$ das Blank-Symbol, das zwar dem Arbeitsalphabet , aber nicht dem Eingabealphabet angehört
- $F \subseteq S$ die Menge der Endzustände
- δ sei die Überführungsfunktion/-relation

$$\delta : (S \setminus F) \times A \to S \times A \times \{L, R, N\} \quad \text{deterministischer Fall} \tag{42}$$
$$\delta \subseteq ((S \setminus F) \times A) \times (S \times A \times \{L, R, N\}) \quad \text{nichtdeterministische Fall} \tag{43}$$

1.57.1 Konfiguration

Eine Konfiguration einer Turingmaschine $TM = (S, E, A, \delta, s_0, \square, F)$ ist ein Tripel (u, s, v) aus $A^* \times S \times A^+$:

- uv ist aktuelle Bandinschrift
- s ist der aktuelle Zustand
- Schreib-Lesekopf über erstem Zeichen von v, daher $v \neq \varepsilon$
- Start der Maschine: $v \in E^* \cup \{\square\}$ (Eingabe), $s = s_0, u = \varepsilon$

1.57.2 Initialkonfiguraiton, Finalkonfiguration, akzeptierte Sprache

- Anfangskonfiguration beim Start der TM mit Eingabe $w \in E^*$ ist $s_0 w$ (bzw. $s_0\square$, falls $w = e$)
- Endkonfiguration sind alle Konfigurationen $us_f v$ mit $s_f \in F$. Hier kan die Berechnung nicht fortgesetzt werden.
- Weiter ist

$$L(TM) := \{w \in E^* | s_o w \vdash^* us_f v, s_f \in F, u, v \in A^*\} \tag{44}$$

die von der Turingmaschine akzeptierte Sprache

1.57.3 Beobachtungen

Simulation endlicher Automat durch TM z.B. wie folgt

- Schreib-Lesekopf hat ausschließlich lesende Funktion
- Lesekopf zeichenweise lesend nach rechts
- Zuständsübergänge des endlichen Automaten übernommen
- Akzeptieren bei Erreichen von $\square$ im 'Automaten-Endzustand'

1.58 These von Church/Turing

Turingmaschinen können 'alles', was überhaupt jemals von Computern"gemacht werden kann.

1.59 Linear beschränkte Automaten

Eine TM heißt linear beschränkter Automat (`LBA`), wenn sie keine Blankzeichen überschreiben darf.

1.59.1 Satz

L ist Typ-1 gdw. L wird von LBA akzeptiert.

1.60 Kellerautomat

Ein Kellerautomat (Pushdown Automaton (PDA)) ist ein Sextupel

$$A = (Q, \Sigma, \Gamma, q_0, \Delta, F) \tag{45}$$

- Q ist das Zustandsalphabet
- Σ ist das Eingabealphabet
- Γ ist das Kelleralphabet
- $q_0 \in Q$ ist der Startzustand
- $\Delta \subset (Q \times (\Sigma \cup \{\lambda\} \times \Gamma^*) \times (Q \times \Gamma^*)$ ist die endliche Übergangsrelation
- $F \subseteq Q$ ist die Endzustandsmenge

1.60.1 PDA erzeugen kontextfreie Sprachen

Sei $G = (\Sigma, N, R, S)$ eine kfG. Betrachte folgende Transitionen:

- $((s, \lambda, \lambda), (f, S))$
- für jede Regel $C \to w : ((f, \lambda, C), (f, w))$
- für jedes Terminalzeichen $a : ((f, a, a), (f, \lambda))$

f ist der einzige Endzustand und s der Startzustand

1. Satz $KF \subseteq PDA$ Hinweis : Es gilt sogar die Gleichheit

1.61 Compiler-Aufbau

- Scanner: lexikalische Analyse
- Parser: syntaktische Analyse
- semantische Analyse
- Codegenerierung
- Optimierung

1.62 Parsing

- Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum
- Rechtsableitung: analog.

Ein Linksparser für Grammatik $G = (\Sigma, Q, R, s)$ liefert zu einem Wort $w \in L(G)$ eine Linksableitung von w, beschrieben durch ein Wort über R, und `NEIN`, falls $w \notin L(G)$. Analog: Rechtsparser liefert Rechtsableitung.

Der Linksparser simuliert im Wesentlichen den beschreibenden Kellerautomat. (allgemein nichtdeterministisch) Simuliert der Kellerautomat die kfG, so gibt der Parser die entsprechende Regel aus. (Produktionsschritt)

Schritte eder Form $((f, a, a,), (f, \lambda))$ heißen Leseschritte (Shifts) und führen zu keiner Ausgabe.

Im Wesentlichen kann der Linksparser die Zustandsinformation des Kellerautomaten ignorieren.

1.62.1 Konflikte beim Linksparsen

Solange ein Terminalzeichen auf dem Keller liegt, muss ein Linksparser einen Leseschritt durchführen. Stimmt das oberste Kellerzeichen nicht mit dem aktuellen Eingabezeichen überein, so `NEIN`)

Bei den Produktionsschritten kann es jedoch zu Konflikten kommen: Welche Produktion (Regel) ist anzuwenden (zu simulieren) und damit auszugeben?

1.63 Greibach-Normalform

Eine kfG $G = (\Sigma, N, R, S)$ ist in Greibach-Normalform gdw.

$$R \subseteq (N \times \Sigma(N \setminus \{S\})^*) \cup (S \times \{\lambda\}) \tag{46}$$

Jede kontextfreie Sprache besitzt eine sie erzeugende kfG in Greibach-NF.

Eine kfG $G = (\Sigma, N, R, S)$ mit $R \subseteq (N \times \Sigma(N \cup \Sigma)^*)$ heißt simpel oder s-Grammatik gdw.

$$\forall A \in N \forall a \in \Sigma : |\{\beta \in (N \cup \Sigma)^* | A \to a\beta \in R\}| \leq 1 \tag{47}$$

1.63.1 Lemma

Linksparser für s-Grammatiken arbeiten deterministisch

1.64 Parser

1.64.1 Linksparser

heißen auch Top-Down-Parser. Der Ableitungsbaum wird von oben nach unten durchforstet.

1. Problem Regelanwendungskonflikte
2. Lösung Verzeichne zu jeder Regel $A \to w$ die Menge der Eingabezeichen aus Σ, die als Präfix der Länge k für ein aus w ableitbares Terminalwort vorkommen können, als Vorschau (Lookahead).

1.64.2 Rechtsparser

Sei $G = (\Sigma, N, R, S)$ eine kfG.

- für jede Regel $C \to w : ((f, \lambda, w), (f, C))$ (Reduktionsschritt)

- für jedes Terminalzeichen $a : ((f, a, \lambda), (f, a))$ (Leseschritt)

Der Kellerautomat akzeptiert bei leerer Eingabe und nur S auf dem Keller

1. Konflikte Es gibt zwei Arten von Konflikten:
   - Welche Produktion (Regel) unter mehreren ist anzuwenden (reduziernd zu simulieren) und damit auszugeben?
   - Reduktionsschritt oder Leseschritt?

2 Berechenbarkeit und Komplexität

2.1 Berechenbarkeit

2.1.1 Berechenbarkeit

- Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist berechenbar, wenn es einen Algorithmus (mechanisches Rechenverfahren) gibt, der f berechnet, also nach Eingabe von $(n_1, \cdots, n_k) \in \mathbb{N}^k$ in endlich vielen Schritten terminiert mit Ausgabe $f(n_1, \cdots, n_k)$

2.1.2 Churches These

Die durch den formalen Begriff der *Turingberechenbarkeit* (`WHILE,GOTO,µ-Rekursivität`) erfasste Klasse von Funktionen stammt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

2.1.3 Turing-Berechenbarkeit

- Der Berechenbarkeitsbegriff sollte mit Hilfe einer sehr einfachen Rechenmaschine beschrieben werden (**Turing-Maschine**) erfasst werden. - Eine (nichtdeterministische) Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, E)$, wobei Q eine endliche Menge von Zuständen, Σ eine endliche Menge von Eingabesymbolen, $\Gamma \subset \Sigma$ endliche Menge von Bandsymbolen

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\} \text{ deterministische} \tag{48}$$

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}} \text{ nichtdeterministische} \tag{49}$$

$q_0 \in Q$ Startzustand,$\square \in \Gamma - \Sigma$ leeres Bandsymbolen $E \subseteq Q$ endliche Menge von Endzuständen

1. Konfiguration Eine Konfiguration einer TM M ist ein Wort $k \in \Gamma^* Q \Gamma^*$: $k = \alpha q \beta$ bedeutet:

   - $\alpha\beta$ ist der nichtleere Teil der Bandinschrift
   - q ist der Zustand TM.
   - Das erste Zeichen von β ist das Zentrum des Lese-/Schreibkopfes

2. formale Definition Eine Funktion $f : \mathbb{N}^* \to \mathbb{N}$ heißt *Turing-berechenbar*, falls es eine (deterministische) TM M gibt, so dass für alle $n_1, \cdots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \cdots, n_k) = m \Leftrightarrow q_0 bin(n_1) \# bin(n_2) \# \cdots \# bin(n_k) \vdash^* q_e bin(m) \tag{50}$$

   Eine Funktion $f : \Sigma^* \to \Sigma$ heißt Turing-berechenbar, falls es eine (deterministische) TM M gibt, so dass für alle $x, y \in \Sigma^*$ gilt:

$$f(x) = y \Leftrightarrow q_0 x \vdash^* q_e y \tag{51}$$

   Damit ist ausgedrückt, dass im Falle $f(x) = undef$ M eine unendliche Schleife geht.

3. Mehrbanddefinition Zu jeder Mehrband-TM M gibt es eine 1-Band TM M' die dieselbe Funktion berechnet wie M.

   (a) formaler

$$\Gamma' := (\Gamma \cup \{*\})^{2k} \tag{52}$$

   M' simuliert M wie folgt: Gestartet mit Eingabe $x_1, \cdots, x_n \in \Gamma^*$ erzeugt M' die Darstellung der Startkonfiguration von M in Spuren-Darstellung.

   - Sei M eine 1-Band TM: $M(i,k); i \leq n$ bezeichne die k -Band TM, die aus M dadurch entsteht, dass alle Aktionen auf Band i ablaufen.

2.1.4 `LOOP-`, `WHILE-`, und `GOTO-` Berechenbarkeit

1. `LOOP`

   - Variablen: $x_0, x_1, \cdots$

- Konstanten: 0, 1, 2
- Trennsymbole: ;, :=
- Operationszeichen: +,-
- Schlüsselwörter `LOOP`, `DO`, `END`
- Wertzuweisung:
  $x_i := x_j + c$ $x_i := x_j - c$ ist ein `LOOP`-Programm
- Sind P_1, P_2 `LOOP`-Programme, dann auch $P_1; P_2$
- ist P ein `LOOP`-Programm, x_1 Variable, dann auch `LOOP` x_1 `DO` P `END`;

(a) formale Definition Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt `LOOP`-berechenbar, falls es ein `LOOP`-Programm P gibt, das gestartet mit $n_1, \cdots, n_k$ in den Variable $x_1, \cdots, x_k$ mit dem Wert $f(n_1, \cdots, n_k)$ in x_0 stoppt.

i. Bemerkung
- Alle `LOOP`-berechenbaren Funktionen sind total definiert.
- Erweiterung der `LOOP`-Programme durch `WHILE`-Schleife $\Rightarrow$ `WHILE`-Programme

2. `WHILE`

(a) formale Definition Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt `WHILE`-berechenbar, falls es ein `WHILE`-Programm P gibt, das gestartet mit $n_1, \cdots, n_k$ in $x_1, \cdots, x_k$ (0 sonst) mit dem Wert $f(n_1, \cdots, n_k)$ in x_0 stoppt. Sonst stoppt P nicht.

i. Bemerkung
- TM können `WHILE`-Programme simulieren. D.h. jede `WHILE`-berechenbare Funktion ist auch TM-berechenbar.

3. `GOTO`

(a) formale Definition Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt `GOTO`-berechenbar, falls es ein `GOTO`-Programm P gibt, das gestartet mit $n_1, \cdots, n_k$ in $x_1, \cdots, x_k$ (0 sonst) mit dem Wert $f(n_1, \cdots, n_k)$ in x_0 stoppt. Sonst stoppt P nicht

i. Bemerkung
- Jedes `WHILE`-Programm kann durch ein `GOTO`-Programm simuliert werden.

- Jedes `GOTO`-Programm kann durch ein `WHILE`-Programm (mit nur einer `WHILE`-Schleife) simuliert werden.
- Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist `GOTO`-berechenbar, falls f `WHILE`-berechenbar ist.
- Eine Funktion ist TM-berechenbar $\Leftrightarrow$ `GOTO-berechenbar` $\Leftrightarrow$ `WHILE`-berechenbar

2.1.5 Primitiv rekursive und µ-rekursive Funktionen

1. Definition Die Klasse der primitiv rekursiven Funktione ist induktiv wie folgt definiert:

   (a) Alle konstanten Funktionen sind primitiv rekursive

   (b) Projektionsabbildungen sind primitiv rekursive, d.h $pr_i : \mathbb{N}^k \to \mathbb{N}$ $pr_i(n_1, \cdots, n_k) = n_i$

   (c) Die Nachfolgerfunktion ist primitiv rekursiv, d.h. $s : \mathbb{N}^k \to \mathbb{N}$ $i \mapsto i + 1$

   (d) Jede Funktion, die durch Einsetzung (Komposition) aus primitiv rekursiven Funktionen entsteht ist primitiv rekursiv.

   (e) Jede Funktion, die durch primitive Rekursion aus primitiv rekursiven Funktionen entsteht, ist primitiv rekursiv.

   (a) Bemerkung
   - primitiv rekursive Funktionen sind offenbar berechenbar
   - primitiv rekursive Funktionen sind total berechenbar
   - Es gilt nicht : primitiv rekursive = total und berechenbar

2. Zusammenhang zur `LOOP`-berechenbaren Funktionen

   Die Klasse der primitiv rekursiven Funktionen stimmt mit der Klasse der `LOOP`-berechenbaren Funktionen überein.

3. µ-Operator Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ Durch Anwendung des µ-Operators entsteht aus f die Funktion $g : \mathbb{N}^K \to \mathbb{N}$ mit $g(x_1, \cdots, x_k) = min\{n : f(n, x_1, \cdots, x_k) = 0$ und für alle $m < n$ ist $f(m, x_1, \cdots, x_k) def.\}$ mit $\min \emptyset = 0$ undefiniert.

   Durch die Anwendung des µ-Operators können partielle Funktionen entstehen.

4. formale Definition Die Klasse der µ-rekursiven Funktionen ist die kleinste Klasse von Funktionen, die die Basisfunktionen (konstante Funktionen, Projektionen, Nachfolgerfunktion) enthält und abgeschlossen ist bzgl. der primitiven Rekursion und der Anwendung des µ-Operators. Die Klasse des µ-rekursiven Funktionen stimmt genau mit der Klasse der `WHILE-` (`GOTO-` , `TM-`) berechenbaren Funktionen überein.

5. Kleene Für jede n -stellige µ-rekursive Funktion f gibt es zwei $n+1$ -stellige, primitiv rekursive Funktionen p und q, so dass sich f darstellen lässt als

$$f(x_1, \cdots, x_n) = p(x_1, \cdots, x_n, q(x_1, \cdots, x_n)). \tag{53}$$

2.1.6 Ackermann-Funktion

1. Definition Die Ackermann-Funktion ist eine Funktion, die intuitive berechenbar (`WHILE` berechenbar), aber nicht primitiv rekursive (`LOOP` berechenbar) ist.

$$ack(0, y) = y + 1 \tag{54}$$
$$ack(x, 0) = ack(x - 1, 1) \tag{55}$$
$$ack(x, y) = ack\,(x - 1, ack\,(x, y - 1)) \tag{56}$$
$$ack(x, y) = ack\underbrace{(x - 1, ack\,(x - 1, \cdots ack\,(x - 1, 1) \cdots))}_{y-mal} \tag{57}$$

2. Lemma Für jedes `LOOP`-Programm P gibt es eine Konstante k mit

$$f_p(n) < a(k, n) \text{ für alle } n \geq k \tag{58}$$

3. Eigenschaften

   - Die Ackermann-Funktion a ist nicht `LOOP`-berechenbar.
   - Die Ackermann-Funktion ist `WHILE`-berechenbar.

2.1.7 Entscheidbarkeit und Semi-Entscheidbarkeit

1. Definition

$A \subseteq \Sigma^*$ heißt entscheidbar, falls die charakteristische Funktion

$$\chi_A : \Sigma^* \to \{0,1\} \tag{59}$$

$$\chi_A(\omega) = \begin{cases} 1 & \text{falls } \omega \in A \\ 0 & \text{sonst} \end{cases} \tag{60}$$

berechenbar ist. $A \subseteq \Sigma^*$ heißt semi-entscheidbar, falls die charakteristische Funktion

$$\chi_A^{'}(\omega) = \begin{cases} 1 & \text{falls } \omega \in A \\ \text{undefiniert} & \textit{sonst.} \end{cases} \tag{61}$$

berechenbar ist.

(a) Man kann an Stelle von $A \subseteq \Sigma^*$ auch $A \subseteq \mathbb{N}$ betrachten.

(b) Das Entscheidungsproblem für A ist die Frage nach einem stoppenden Algorithmus mit $\omega \to (ja, nein)$

(c) Das Semi-Entscheidungsproblem für A ist die Frage nach einem Algorithmus mit $\omega \to (ja, ???)$. Hat der Algorithmus noch nicht gestoppt, dann ist unklar ob $\omega \in A$ oder nicht.

2. Satz

- A ist entscheidbar $\Leftrightarrow$ sowohl A als auch $\bar{A}$ sind semi-entscheidbar.
- $A \subseteq \Sigma^*$ heißt rekursive aufzählbar, falls $A = \emptyset$ oder es eine totale und berechenbare Funktion $f : \mathbb{N} \to \Sigma^*$ gibt mit

$$A = \{f(0), f(1), f(2), \cdots\} \tag{62}$$

"f zählt A auf"

- Eine Sprache ist rekursive aufzählbar, genau dann wenn sie semi-entscheidbar ist.
- Eine Sprache A ist entscheidbar, genau dann wenn A und $\bar{A}$ rekursive aufzählbar sind.
- A heißt abzählbar, falls $A = \emptyset$ oder es gibt eine totale Funktion f

$$A = \{f(0), f(1), f(2), \cdots\} \tag{63}$$

- A ist rekursive aufzählbar, falls A durch eine totale rekursive Funktion abzählbar ist.

2.1.8 Das Halte-Problem und die Reduzierbarkeit

1. Definition Die folgende Sprache

$$K = \{\omega \in \{0,1\} | M_\omega \text{ angesetzt auf } \omega \text{ hält } \} \tag{64}$$

heißt spezielles Halte-Problem

2. Eigenschaften

   (a) Das spezielle Halte-Problem ist nicht entscheidbar.

   (b) Seien $A, B \subseteq \Sigma^*$. A heißt auf B reduzierbar ($A \leq B$), falls es eine totale berechenbare Funktion. $f : \Sigma^* \to \Sigma^*$ gibt mit:

$$x \in A \Leftrightarrow f(x) \in B \tag{65}$$

   (c) Gilt $A \leq B$ und ist B entscheidbar, so ist auch A entscheidbar.

   (d) Gilt $A \leq B$ und ist B semientscheidbar, so ist auch A semientscheidbar.

   (e) $A \leq B$ und A ist nicht entscheidbar $\Rightarrow B$ ist nicht entscheidbar

   (f) Die Sprache $H = \{\omega \# x | M_\omega$ angesetzt auf x hält $\}$ heißt (allgemeines) Halte-Problem

   (g) Das Halte-Problem ist nicht entscheidbar

   (h) Die Sprache $H_0 = \{\omega | M_\omega$ angesetzt auf leeren Band hält$\}$ heißt Halte-Problem auf leeren Band.

   (i) Das Halte-Problem auf dem leeren Band H_0 ist nicht entscheidbar.

3. Satz von Rice Sei $\mathcal{R}$ die Klasse alelr TM-berechenbaren Funktionen. Sei $\mathcal{S} \subset \mathcal{R}, \mathcal{S} \neq \emptyset$, Dann ist die Sprache

$$\mathcal{C}(\mathcal{S}) := \{\omega | \text{ die von } M_\omega \text{ berechnete Funktion liegt in } \mathcal{S}\} \tag{66}$$

unentscheidbar.

2.1.9 Das Postsche Korrespondenz-Problem

1. Definition Das nachfolgend beschriebene Problem heißt Postsches Korrespondez-Problem (PCP)

gegeben :
Eine endliche Folge $(\mathrm{x}_1, y_1), \cdots, (x_k, y_k)$ von Wortpaaren mit $x_i, y_i \in A^+$
(A endliche Alphabet)
gefragt :
Gibt es eine Folge von Indizes $\mathrm{i}_1, \cdots, i_n \in \{1, \cdots, k\}, n \geq 1,$ mit x_{i_1}

2. Eigenschaften

   - Das Postsche Korrespondenz-Problem ist nicht entscheidbar
   - $MPCP \leq PCP$
   - $H \leq MPCP$
   - PCP bleibt unentscheidbar, falls für das Alphabet A gilt: $A = \{0, 1\}$. ("01-PCP")
   - Sei PCP_k die Variante des PCPs deren Eingabe aus genau k Wortpaaren besteht.
     - PCP_k ist unentscheidbar für $k \geq 9$
     - PCP_k ist entscheidbar für $k \leq 2$
   - Das Halte-Problem H für TM ist semi-entscheidbar.

2.1.10 Universelle TM

1. Definition Nachobiger Folgerung gibt es eine TM U, die sich bei Eingabe von $\omega\#x$ so verhält wie M_ω bei Eingabe von x. (Zunächst nur im Bezug auf Halten, bzw. Nicht-Halten).

2. Eigenschaften

   - Das Schnittproblem für kontextfreie Sprachen (G_1, G_2 kontextfrei Sprachen gilt: $L(G_1) \cup L(G_2) = \emptyset$?) ist unentscheidbar.
   - Das Schnittproblem für deterministische kontextfreie Sprachen ist unentscheidbar.
   - Das Äquivalenzproblem für kontextfreie Sprachen ist unentscheidbar.
     - Das Äquivalenzproblem ist für folgende Probleme unentscheidbar:
       * nichtdeterministische Kellerautomaten
       * BNF
       * EBNF
       * Syntaxdiagramme
       * LBA
       * kontextsensitiven Grammatiken
       * TM
   - Das Leerheitsproblem für kontextsensitive Sprachen ist unentscheidbar.

2.1.11 Der Gödelsche Unvollständigkeitssatz

1. Definition Arithmetische Terme sind induktiv wie folgt definiert:

   - Jedes $n \in \mathbb{N}$ und jede Variabel $x_i, i \in \mathbb{N}$ ist ein Term
   - sind t_1, t_2 Terme, so ist auch $(t_1 + t_2)$ und $(t_1 \cdot t_2)$ Terme.

   Arithmetische Formeln sind wie folgt definiert:

   - Sind t_1, t_2 Terme, dann ist $t_1 = t_2$ eine Formel.
   - Sind F, G Formeln, dann auch $\neg F, F \vee G$ und $F \wedge G$
   - Ist x eine Variable und F eine Formeln, dann sind $\exists x F$ und $\forall x F$ Formeln.
   - Eube Variable heißt gebunden, wenn sie in Wirkungsbereich eines Quantoren steht. Sonst heißt eine Variable frei.
   - Eine Funktion \$f:N$^{\mathrm{k}}$ $\rightarrow$N ist arithmetisch repräsentierbar, falls es eine arithmetische Formel $F(x_1, \cdots, x_k, y)$ gibt, so dass für alle $n_1, \cdots, n_k, m \in \mathbb{N}$ gilt:

     $$f(n_1, \cdots, n_k) = m \Leftrightarrow F(x_1/n_1, , x_k/n_k, y/m) \text{ ist wahr.} \tag{67}$$

   - Jede `WHILE`-berechenbare Funktion ist arithmetisch repräsentierbar.
   - Die Menge der arithmetischen Formeln ist nicht rekursiv aufzählbar.

2. Zusatz Jedes Beweissystem für WA ist notwendigerweise unvollständig (d.h es bleiben stets Formeln, die nicht beweisbar sind.)

2.1.12 Beweissystem

1. Definition Ein Beweissystem für eine Menge $A \subseteq \Gamma^*$ ist ein Paar (B, F) mit folgenden Eigenschaften:

   (a) $B \subseteq \Gamma^*$ ist entscheidbar.

   (b) $F : B \rightarrow A$ ist total und berechenbar.

2.2 Komplexität

2.2.1 TIME-Komplexität

Die Komplexitätsklasse TIME $(f(n))$, wobei $f(n)$ nach oben durch LOOP-Programme beschränkt weden kann, ist enthalten in der Klasse der primitiv rekursiven Sprachen (bzw. der LOOP-berechenbaren Sprachen)

1. Korollar =TIME=$(n^k(k \in \mathbb{N}), \texttt{TIME}(2^n), \texttt{TIME}(2^{2^{\cdots 2}})$ enthalten nur primitiv rekursiven Mengen.

2.2.2 Die Komplexitätsklassen P und NP

1. Definition Ein Polynom $p : \mathbb{N} \to \mathbb{N}$ ist eine Funktion der Form

$$p(n) = a_k n^k + \cdots + a_1 n + a_0 \qquad a_i \in \mathbb{N}, k \in \mathbb{N} \tag{68}$$

2. Komplexitätsklasse P

$$P = \bigcup_{P \text{ Polynom}} TIME(p(n)) \tag{69}$$

$$= \{A : TMM \text{ mit } L(M) = A \text{ und } time_M(x) \leq p(|x|) \text{ für ein Polynom } p\} \tag{70}$$

$$\Rightarrow P = \bigcup_{k \geq 1} TIME(O(n^k)) \tag{71}$$

   - P = Klasse der effizienten Algorithmen (d.h. der praktisch realisierbaren Algorithmen)
   - Algorithmen mit exponentieller Laufzeit sind nicht effizient.
   - P könnte auch als WHILE-Programm mit logarithmischen Kostenmaß definiert werden.
   - \$P$\subseteq TIME(2^n)\subseteq$\Primitiv rekursive Sprache\
   - P enthält alle Probleme, für die sich in polynomieller Zeit ein Beweis finden lässt.

3. Komplexitätsklasse NP Sei M eine nichtdeterministische Mehrband-

TM

$$time_M(x) = \begin{cases} min & \{ \text{Länge einer akzeptierenden Berechnung von } M \text{ auf } x\}, x \in L(M) \\ 0, & x \notin L(M) \end{cases} \tag{72}$$

$$f : \mathbb{N} \to \mathbb{N} \tag{73}$$

$$NTIME(f(n)) = \{A : NTMM \text{ mit } L(M) = A \text{ und } time_M(x) \leq f(|x|) \forall x \in \Sigma^*\} \tag{74}$$

$$NP := \bigcup_{pPolynom} NTIMEp(n) \tag{75}$$

$$= \bigcup_{k \geq 1} TIME(O(n^k)) \tag{76}$$

Offenbar gilt: $P \subseteq NP$. Die Umkehrung ist unklar.

4. NP -Vollständig

   (a) Definition
   Seien $A, B \subseteq \Sigma^*$. A heißt polynomiell auf B reduzierbar ($A \leq_p B$), falls es eine totale und mit polynomialer Komplexität berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt:

$$x \in A \Leftrightarrow f(x) \in B \tag{77}$$

   (b) Lemma
   - $\leq_p$ ist transitiv
   - $A \leq_p B, B \in P \to A \in P$
   - $A \leq_p B, B \in NP \to A \in NP$
   - Eine Sprache A heißt $NP - hart$, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$.
   - Eine Sprache A heißt NP -vollständig, falls A NP -hart ist und ein $A \in NP$ gilt.
   - NP -vollständige Sprachen sind ßchwereSSprachen in NP.
   - Sei A NP -vollständig. Dann gilt:

$$A \in P \Leftrightarrow P = NP \tag{78}$$

5. Erfüllbarkeitsproblem der Aussagenlogik `SAT` Das folgende Problem heißt SAT:

   - Eine Formel F der Aussagenlogik mit n Variablen, $n \in \mathbb{N}$
     - Ist F erfüllbar? (d.h. Belegung $a \in \{0,1\}^n$ mit $F(a) = 1$?)

6. Theorem von Cook Das Erfüllbarkeitsproblem der Aussagenlogik `SAT` ist NP -vollständig.

7. weitere NP -vollständige Probleme

   (a) Definition Boolsche Formel F in KNF mit höchstens 3 Literalen pro Klausel.

   (b) Eigenschaft `3SAT` ist NP -vollständig.

8. `CLIQUE`

   (a) Definition Ein ungerichteter Graph $G = (V, E), k \in \mathbb{N}$

   - Besitzt G eine Clique der Größe k? Wobei Clique ein vollständiger Teilgraph $G' = (V', E')$ ist, mit $(u, v) \in E \quad \forall u, v \in V', u \neq v$

   (b) Satz `CLIQUE` ist NP -vollständig

9. Hamilton-Kreis

   (a) Definition Ein Hamilton-Kreis ist eine Permutation der Knotenindizies $(\nu_{\pi(1)}, \cdots, \nu_{\pi(n)}$, so dass $(\nu_{\pi(i)}, \nu_{\pi(i+1)}) \in E \forall i = 1, \cdots, n-1$ und $(\nu_{\pi(n)}, \nu_{\pi(1)}) \in E$

   (b) Satz

   - Ein gerichteter Hamilton-Kreis ist NP -vollständig
   - F ist erfüllbar $\Leftrightarrow$ G hat Hamilton-Kreis.
   - Ein ungerichteter Hamilton-Kreis ist NP -vollständig.
   - Eulerkreis ist in P Königsberger Brückenproblem.

10. Traveling Salesperson $n \times n$ Matrix $(M_{i,j})$ von "Entfernungen" zwischen n Städten. Gesucht Permutation ("Rundreise") mit

$$\Sigma_{i=1}^{n-1} M_{\pi(i),\pi(i+1)} + M_{\pi(n),\pi(1)} \leq k \tag{79}$$

    (a) Satz Das Traveling Salesperson Problem ist NP -vollständig.