Einführung in die Theoretische Informatik
TUM Sommersemester 2012
Dozent: Helmut Seidl

Janosch Maier

1. August 2012

Inhaltsverzeichnis

0 Einführung

0.1 Organisatorisches

Wöchentliche Hausaufgaben mit Notenbonus (40% in beiden Teilen nötig)
Vorlesung wird aufgenommen (http://ttt.in.tum.de/lectures/index_ss12.php)

1 Formale Sprachen und Automaten

1.1 Grundbegriffe

- Alphabet Σ ist endliche Menge
- Wort/String aus Σ ist endliche Folge von Zeichen
- Leere Folge ist ϵ
- uv ist Konkatenation von u und v
- w sei Wort: $w^0 = e$, $w^{n+1} = ww^n$
- $|w|$ ist Länge von w
- Σ^*: Menge aller Wörter über Σ
- $\Sigma^+ = \Sigma\Sigma^*$: Menge aller nicht-leeren Wörter über Σ
- Teilmenge $L \subseteq \Sigma^*$ ist **(formale) Sprache**
- Σ^* ist abzählbar, falls Σ endlich
- Sprache heißt entscheidbar, wenn eine Funktion bestimmen kann, ob ein Wort enthalten ist.
- $A^R := \{w^R | w \in A\}$ ist Umkehrung / Spiegelung der Sprache A, w^R ist w rückwärts

1.2 Rechenregeln

- $\emptyset A = \emptyset$
- $\epsilon A = A$
- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$
- $A^*A^* = A^*$

2 Reguläre Sprachen

2.1 Deterministische Endliche Automaten (DFA)

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Q Endliche Menge von Zuständen

Σ Endliches Eingabealphabet

δ Übergangsfunktion $Q \times \Sigma \to Q$

q_0 Startzustand $q_0 \in Q$

F Endzusdände $F \subseteq Q$

Eine Sprache ist regulär, gdw sie von einem DFA akzeptiert wird.
$\hat{\delta}(q, aw) = \hat{\delta}(\delta(q, a), w)$

2.2 Nichtdeterministische Endliche Automaten (NFA)

- Veralgemeinerung: $\delta : Q \times \Sigma \to \mathcal{P}(Q)$
- $\mathcal{P}(Q) = 2^Q$ = Potenzmenge von Q = Menge aller Teilmengen
- Wort wird akzeptiert, wenn ein Weg zu einem Endzustand führt.
- ϵ-NFA erlaubt Übergang ohne Eingabezeichen

2.3 Reguläre Ausdrücke

Bindungsstärke für Reguläre Ausdrücke: $*$ > Konkatenation > |

Reguläre Ausdrücke

- $\emptyset$
- ϵ
- a, für $a \in \Sigma$
- $ab, a|b, a^*$, wenn a, b reguläre Ausdrückes

Erweiterte Reguläre Ausdrücke unter Unix

- $. = a_1|...|a_n$, mit $\Sigma = \{a_1, ..., a_n\}$
- $[a_1, ..., a_n] = a_1|...|a_n$
- $[\hat{}a_1, ..., a_n] = [b_1, ..., b_n], \{b_1, ..., b_n\} = \Sigma \backslash \{a_1, ..., a_n\}$
- $a+ = aa^*$
- $a\{n\} = a...a$ (n mal)

Rechenregeln

- $\alpha \equiv \beta \Leftrightarrow L(\alpha) = L(\beta)$

- $\epsilon \equiv \emptyset^*$, aber $\epsilon \neq \emptyset^*$
- $\emptyset|\alpha \equiv \alpha|\emptyset \equiv \alpha$
- $\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$
- $\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha$
- $\emptyset^* \equiv \epsilon \equiv \epsilon^*$
- $(\alpha|\beta)|\gamma \equiv \alpha|(\beta|\gamma)$ (Assoz.)
- $(\alpha\beta)\gamma \equiv \alpha(\beta\gamma)$ (Assoz.)
- $\alpha|\beta \equiv \beta|\alpha$ (Kommut.)
- $\alpha(\beta|\gamma) \equiv \alpha\beta|\alpha\gamma$ (Dist.)
- $(\alpha|\beta)\gamma \equiv \alpha\gamma|\beta\gamma$ (Dist.)
- $\alpha|\alpha \equiv \alpha$ (Idempotenz)
- $\epsilon|\alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$
- $\epsilon|\alpha^* \equiv \alpha^*$

2.4 Algorithmen zur Umwandlung von RegEx/NFA/DFA

Siehe Folien auf http://carlos-camino.de

2.5 Pumping Lemma für Reguläre Sprachen

Pumping Lemma (Zyklen im DFA von Sprachen mit undendlichen Wörtern) hinreichend, aber nicht notwendig um Nicht-Regularität zu zeigen.

Für Reguläre Sprachen L gilt:

- $\exists n; |z| > n; z = uvw$
- $|u| + |v| \leq n$
- $|v| > 0$
- $uv^i w \in L$

Pumping Lemma unzutreffend $\Rightarrow$ Keine Reguläre Sprache

2.6 Entscheidbarkeit

D ist Beschreibung einer Sprache.

2.6.1 Wortproblem

$w \in L(D)$?

- DFA: $\mathcal{O}(|w|)$
- NFA: $\mathcal{O}(|Q|^2|w|)$

2.6.2 Leerheitsproblem

$L(D) = \emptyset$?

- DFA: $\mathcal{O}(|Q||\Sigma|)$
- NFA: $\mathcal{O}(|Q|^2|\Sigma|)$

2.6.3 Endlichkeitsproblem

$L(D)$ endlich?

$L(M) = \infty$, wenn nicht-leere Schleife erreichbar, die Endzustand erreichen kann.

2.6.4 Äquivalenzproblem

- DFA: $\mathcal{O}(|Q_1||Q_2|)$
- NFA: $\mathcal{O}(2^{|Q_1|+|Q_2|})$

2.6.5 Fazit

Repräsentation der Sprache u.U. wichtig für Komplexität des Problems

2.7 Äquivalenz Regulärer Ausdrücke

- $L(\alpha \sqcap \beta) = L(\alpha) \cap L(\beta)$
- Substitution: $\sigma := V \to RE$ (Variablen werden auf Reguläre Ausdrücke abgebildet)

Äquivalenztest: Betrachte Variablen als Konstanten

- $L[M_1 M_2] = L[M_1]L[M_2]$
- $L[M_1 \cup M_2] = L[M_1] \cup L[M_2]$
- $L[M^*] = (L[M])^*$
- $L[\sigma(L(E))] = L(\sigma(E))$

2.8 Automaten und Gleichungssysteme

- Ardens Lemma (Sprachen): $X = AX \cup B \Rightarrow X = A^*B, \epsilon \notin A$
- Für RegEx gilt: $X \equiv \alpha X|\beta \Rightarrow X \equiv \alpha^*\beta, \epsilon \notin L(\alpha)$
- DFA to RegEx: LGS lösen (Folie auf http://carlos-camino.de)

2.9 Minimerung endlicher Automaten

Nur bei DFAs! Siehe Folie auf http://carlos-camino.de.

Quotientenautomat ist minimal, wenn Ursprungsautomat keine unerreichbaren Zustände hatte.

- Eine Sprache ist regulär gdw. Quotientenautomat endlich viele Äquivalenzklassen besitzt

3 Kontextfreie Sprachen

3.1 Kontextfreie Grammatiken

$$G = (V, \Sigma, P, S)$$

- V: Nichtterminalzeichen / Variablen
- Σ: Terminalzeichen
- $P \subseteq V \times (V \cup \Sigma)^*$: Produktionen
- $S \in V$: Startsymbol

3.2 Kontextfreie Sprache

Sprache L ist kontextfrei gdw. Kontextfreie Grammatik G existiert, mit $L = L(G)$.

3.2.1 Linearität

Rechtslinear: Nur Produktionen der Form $A \to aB$ oder $A \to \epsilon$
Linkslinear: Nur Produktionen der Form $A \to Ba$ oder $A \to \epsilon$

Erzeugen genau regulären Sprachen (echte Teilklasse der kontextfreien Sprachen)

3.2.2 Beweise mit Kontextfreien Sprachen

$w \in L(S) \Rightarrow P(w)$ (Wort in Sprache, dann gilt Bedingung): Schmantische mit Induktion über Erzeugung von w.
$P(w) \Rightarrow w \in L(S)$ (Wenn Bedingung gilt, dann Wort in Sprache): Meist Induktion über $|w|$. Meist Kreativität benötigt

3.2.3 Mehrdeutigkeit

- Grammatik mehrdeutig, wenn kein eindeutiger Syntaxbaum möglich
- Sprache inehärent Mehrdeutig, wenn mehrere mehrdeutige Grammatiken die Sprache beschreiben

3.3 Chomsky-Normalform

Alle Produktionen haben die Form:

- $A \to a$
- $A \to BC$
- $S \to \epsilon$ (ϵ-Produktion)

3.3.1 Überführung in Chomsky-Normalform

- Eleminieren von ϵ-Produktionen
- Eliminieren von Kettenproduktionen
- Erstelle Nichtterminal A für jedes Terminal a, füge Produktinon $A \to a$ hinzu, und ersetze in allen anderen Produktionen a durch A
- Ersetze $A \to B_1 B_2 ... B_k, (k \geq 3)$ durch $A \to B_1 C_2, C_2 \to B_2 C_3, ...$

3.3.2 Greibach-Normalform

Jede Produktion hat die Form $A \to aA_1 ... A_n$

Es gibt zu jeder kontextfreien Grammatik eine kontextfreie Grammatik in Greibach-Normalform mit $L(G') = L(G) \backslash \{\epsilon\}$

3.4 Pumping-Lemma für kontextfreie Sprachen

Für $n \geq 1$ gilt für $z \in L$ mit $|z| \geq n$: $z = xvwxy$ mit

- $vx \neq \epsilon$
- $vwx \leq n$
- $\forall i \in \mathbb{N}. uv^i wx^i y \in L$

3.5 Abschlusseigenschaften

Kontextfreie Grammatiken sind abgeschlossen unter

- Vereinigung
- Konkatenation
- Stern
- Spigelung

3.6 Algorithmen für kontextfreie Grammatiken

Symbol X ist:

- Nützlich: Ausgehend von Startsymbol wird Ableitung genutzt X
- Erzeugend: X ist Determinante einer Ableitung
- Erreichbar: X kann von Startsymbol erreicht werden

Nützliche Ableitungen sind erzeugend und erreichbar.

Entfernen von erzeugenden, dann unerreichbaren Symbolen $\rightarrow$ Nur noch nützliche Symbole

Menge der erzeugenden, erreichbaren Symbole ist berechenbar

3.7 Cocke-Younger-Kasami-Algorithmus

Wortproblem für kontextfreie Grammatiken. Berechnet V_{ij} rekursiv nach wachsendem $j - i$.

Tabelle über welche Produktionen Symbole erstellt werden können, ausgehend von Chomsky-Normalform

3.8 Kellerautomaten

- Syntaxanalyse von Programmiersprachen
- Analyse von Programmen mit Rekursion

3.8.1 (Nichtdeterministischer) Kellerautomat

(N)PDA = (Nondeterministic) Pushdown Automat

- Q: Zustandsmenge
- Σ: Eingabealphabet
- Γ: Kelleralphabet
- $q_0 \in Q$: Anfangszustand
- $Z_0 \in \Gamma$: Anfangskellerinhalt
- $\delta \subseteq Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$: Zustandsübergangsfunktion
- $F \subseteq Q$: Endzusdände
- Kellerautomaten akzeptieren genau kontextfreie Sprachen
- PDA mächtiger als DPDA

3.9 Chomsky-Hierachie

Für jede Produktion $(\alpha \to \beta)$ gilt:

- Typ 0 – Chomsky-Grammatik, Turingmaschine
- Typ 1: $|\alpha| \leq |\beta|$ – Kontextsensitvie Grammatik, Linear beschränkter Automat
- Typ 2: $1 + \alpha \in V$ – Kontextfreie Grammatik, Kellerautomat
- Typ 3: $2 + \beta \in \Sigma \cup \Sigma V$ – Rechtslineare Grammatik, Endlicher Automat

3.10 Tabelle

TODO

4 Berechenbarkeit und Entscheidbarkeit

4.1 Berechenbarkeit

$f : \mathbb{N}^k \to \mathbb{N}$ berechenbar, wenn Algorithmus (endliche Wörter) nach endlichen Schritten terminiert, bzw. nicht terminiert, wenn Eingabe nicht definiert.

Funktion ist $f : A \to B$:

- total, gdw $f(a)$ für alle $a \in A$ definiert
- partiell, gdw $f(a)$ undefiniert sein kann
- echt partiell, gdw nicht total

4.1.1 Nicht berechenbare Funktionen

Es gibt nicht-berechenbare Funktionen in $\mathbb{N} \to \{0, 1\}$

Abzählbar viele Algorithmen, überabzählbar viele Funktionen

4.1.2 Formalismen

Folgende Formalismen sind gleich mächtig:

- Turingmaschinen
- λ-Kalkül
- μ-rekursive Funktionen
- Markov-Algorithmen
- ...

Church-Turing These: Begriff der Berechenbarkeit durch Turingmaschinen stimmt mit intuitiver Berechenbarkeit überein.

4.2 Turingmaschine

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

Q Zustände

Σ Eingabealphabet

Γ Bandalphabet; $\Sigma \subset \Gamma$

δ (partielle) Übergangsfunktion; $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$

q_0 Startzustand; $q_0 \in Q$

$\square$ Leerzeichen; $\Gamma \backslash \Sigma$

F Endzusdände; $F \subseteq Q$

$\delta(q, a) = (q', b, d)$: Zustand a, liest a auf Band $\to$ Neuer Zustand q', a mit b überschrieben, Lesekopf in Richtung d

Nichtdeterministische Turingmaschine $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$

Konfiguration $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$ bedeutet:

- Band: $\square\alpha\beta\square$
- Zustand: q
- Kopf: Erstes Zeichen von β

Startkonfiguration bei Eingabe $w \in \Sigma^*$ ist: (ϵ, q_0, w)

Berechnung

$$(\alpha, q, \beta) \to_M \begin{cases} (a, q', c \text{ rest}(\beta)) & d = N \\ (ac, q', \text{rest}(\beta)) & d = R \\ (\text{butlast}(\alpha), q', \text{last}(\alpha)\ c \text{ rest}(\beta)) & d = L \end{cases}$$

4.2.1 Turing-Berechenbarkeit

Turingmaschine M akzeptiert Srpache $L(M) = \{w \in \Sigma^* | \exists q \in F, \alpha, \beta \in \Gamma^*.(\epsilon, q_0, w) \to_M^* (\alpha, q, \beta)\}$

Eine Sprache heißt Turing-Berechenbar, wenn es eine Turing-Maschine gibt, $f(n_1, ..., n_k) = m$, die für jede binäre Eingabe $(n_1, ..., n_k)$ genau m liefert.

4.2.2 Terminieren von Turingmaschinen

Turingmaschine hält sofort, wenn ein Endzustand erreicht ist.

Ist δ partiell, so kann eine Turingmaschine auch halten, bevor Endzustand erreicht ist.

$\delta(q, a) = \emptyset$, wenn $q \in F$

4.2.3 Deterministische Turingmaschinen

Zu jeder nichtdeterministischen Turingmaschine N gibt es eine deterministische Turingmaschinen M, mit $L(N) = L(M)$.

4.2.4 Turingmaschinen und Chomsky Hierachie

Von Turingmaschinen akzeptierte Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierachie.

4.2.5 K-Band Turingmaschinen

Jede k-Band-Turingmaschine lösst sich durch eine 1-Band-Turingmaschine simulieren.

Idee: Festhalten von Tupeln, die den aktuellen Stand aller Bänder und Leseköpfe der k-Band-Turingmaschine ausdrücken

4.3 Programmieren von Mehrband-Turingmaschinen

- Band i := Band $i + 1$
- Band i := Band $i - 1$
- Band $i := 0$
- Band i := Band j

Imperatives Programmieren über Fallunterscheidung & Hintereinanterschaltung von Turingmaschinen

- :=
- ;
- *if*
- *while*

4.4 LOOP-, WHILE- und GOTO-Berechenbarkeit

- LOOP, WHILE ≡ for/while Programme
- GOTO ≡ Assembler

4.4.1 LOOP

Variablen: $X \in \{x_0, x_1, ...\}$, Konstanten: $C \in \{0, 1, ...\}$. Eingabe in $x_1, x_2, ...$ Ausgabe in x_0.

$$
\begin{aligned}
P \rightarrow\ & X := X + C \\
| \ & X := X - C \\
| \ & P; P \\
| \ & \text{LOOP } X \text{ DO } P \text{ END}
\end{aligned}
$$

Abkürzungen

- $x_i := x_j$
- $x_i := n$
- $x_i := x_k + x_k$
- $x_i := x_j * x_k$
- DIV, MOD, …
- $x :=$ komplexer Ausdruck
- IF $x = 0$ THEN P END
- IF $x = 0$ THEN P ELSE X END

4.4.2 WHILE

Erweiterung von LOOP Programmen

$$P \rightarrow \text{ WHILE } X \neq 0 \text{ DO } P \text{ END}$$

- WHILE-Schleifen, können LOOP-Schleifen simulieren.
- Nicht umgekehrt möglich

$f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist WHILE-berechenbar, gdw ein WHILE Programm existiert, das

- mit $x_0 = f(x_1, ..., x_k)$ terminiert, wenn $f(x_1, ..., x_k)$ definiert
- nicht terminiert, wenn $f(x_1, ..., x_k)$ nicht definiert

WHILE Berechenbarkeit

- Turingmaschinen können WHILE-Programme simulieren (WHILE-Berechenbarkeit → Turing-Berechenbarkeit)
- Jedes WHILE Programm ist zu einen WHILE Programm mit genau einer WHILE Schleife äquivalent

4.4.3 GOTO

Sequenz von markierten Anweisungen $M_1 : A_1; ...M_k : A_k$

- $x_i := x_j + n$
- $x_i := x_j - n$
- GOTO M_i
- IF $x_i = n$ GOTO M_j
- HALT

GOTO Berechenbarkeit

- WHILE- und GOTO-Berechenbarkeit sind äquivalent
- GOTO Programme können Turingmaschinen simulieren (Turing-Berechenbarkeit → GOTO-Berechenbarkeit)

4.5 Primitiv rekursive Funktionen

$f : \mathbb{N}^k \to \mathbb{N}, k \geq 0$. Ist primitiv rekursiv, wenn induktiv erzeugbar über Primitive Rekursion, Komposition von primitiv rekursiven Funktionen. Basisfunktonen sind primitiv rekursiv. Primitiv rekursive Funktionen sind total.

4.5.1 Basisfunktionen

- $f(x) = 0$
- $s(n) = n + 1$
- $\pi_i^k : \mathbb{N}^k \to \mathbb{N}, 1 \leq i \leq k : \pi_i^k(x_i, ..., x_k) = x_i$

4.5.2 Funktionskomposition

$f(\overline{x}) = g(h_1(\overline{x}), ..., h_k(\overline{x}))$

4.5.3 Fixe Art der Rekursion

- $f(x, \overline{x}) = g(\overline{x})$
- $f(m + 1, \overline{x}) = h(f(m, \overline{x}), m, \overline{x})$

4.5.4 Abkürzungen

- *add* (+), *mult* (*)
- Erweiterte Komposition der Funktionenreihe g, wenn nur aus Funktionen von g und Variablen zusammengesetz. Bsp: $f(x, y) = g_1(x, g_x(y, g_3(x)))$. Macht folgendes möglich:
  - $f(x, \overline{x}) = t_0$
  - $f(m + 1, \overline{x}) = t$
- Vorgänger: *pred*
- Modifizierte Differenz (≥ 0): $\dot{-}$

4.5.5 Funktionen

- Prädikate $P(x)$
- beschränkter max-Operator $q(n)$ (größtes Element $x < n$, auf das $P(x)$ zutrifft)
- beschränkter Existenzquantor $\hat{Q}(n)$ (Existiert ein Element bis Grenze n, auf das P zutrifft)

4.6 PR = LOOP

LOOP → PR: Kodierung aller Variablen des LOOP Programms in einer Zahl.

4.6.1 Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist Bijektion zwischen $\mathbb{N}^2$ und $\mathbb{N}$ $p_1(n), p_2(n)$ sind Umkehfunktionen von c, welche x, y ergeben.

4.6.2 PR = LOOP

PR-Funktionen sind genau die LOOP-berechenbaren Funktionen.

4.6.3 Reversible Kodierung von Zahlenfolgen als Zahlen

- $k = max(\{i_1, ..., i_n\}) \Rightarrow (i_1, ..., i_n)$ kodiert als Zahl $i_1 ... i_n$ zur Basis k
- $\mathbb{N}^*$ als Polynom aus Primzahlen

4.7 μ-rekursive Funktionen

Unbeschränkte Suche $0, ...$ nicht mit primitiv rekursiven Funktionen möglich. μ-Operator formalisiert diese Suche ⇒ Alle berechenbaren Funktionen.

$$f(n) = \perp \Rightarrow f(n) \text{ ist undefiniert}$$

4.7.1 μ-Operator

Für $f : \mathbb{N}^{k+1} \to \mathbb{N}$ gilt:

$$\mu f : \mathbb{N}^k \to \mathbb{N} : \mu f(\overline{x}) = \begin{cases} \min\{n \in \mathbb{N} | f(n, \overline{x}) = 0\} & \text{wenn } f(m, \overline{x}) \neq \perp, \forall m \leq \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\overline{x}) = find(0, \overline{x})$
$find(n, \overline{x}) =$ if $f(n, \overline{x}) = 0$ then n else $find(n+1, \overline{x})$

4.7.2 μ-rekursive Funktionen

Kleinste Menge an Funktione, die durch primitive Rekursion oder mu-Operator entstehen.

4.7.3 μ-rekursiv = WHILE

μ-rekursiven sind WHILE-berechenbare Funktionen.

4.7.4 Kleene

n-stellige μ-rekursive Funktion f besitzt zwei $n+1$-stellige primitv rekursive Funktionen h, h', mit:

$$f(\overline{x}) = h(\mu h'(\overline{x}, \overline{x})$$

4.8 Ackermann-Funktion

$$\begin{aligned} a(0,n) &= n+1 \\ a(m+1,0) &= a(m,1) \\ a(m+1,n+1) &= a(m,a(m+1,n)) \end{aligned} \tag{1}$$

Nicht primitiv rekursiv, aber berechenbar, total.

4.8.1 Ackermann-Familie

$A_m(n) = a(m,n)$

$$A_0(n) = s(n)A_{m+1}(n) = A_m^{n+1}(1) = A_m(\ldots A_m(1)\ldots) \tag{2}$$

Primitive Rekursion auf höherer Ebene (iter).

4.8.2 Lexikographische Ordnung

$$(m,n) > (m',n') \Leftrightarrow m > m' \vee (m = m' \wedge n > n')$$

terminiert auf $\mathbb{N} \times \mathbb{N}$

4.8.3 Berechenbare Funktionen

LOOP = PR < total (Ackermann) < berechenbar = TM = WHILE = GOTO = μR

4.9 Entscheidbarkeit und das Halteproblem

Unentscheidbar, ob ein Programm terminiert

4.9.1 Entscheidbarkeit

Menge A entscheidbar, wenn charakteristische Funktion

$$\mathcal{X}_A(x) := \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \tag{3}$$

Problem $P(x)$ entscheidbar, gdw $\{x|P(x)\}$ entscheidbar.

4.9.2 Rechenregeln, Definitionen

- A entscheidbar $\Rightarrow \overline{A}$ entscheidbar.
- $M[w]$: Maschine M mit Eingabe w
- $M[w]\downarrow$: $M[w]$ terminiert

4.9.3 Kodierung einer TM als Wort

$TM \to \{0,1\}^*$ nicht surjektiv. Nicht jedes Wort kodiert eine TM.

4.9.4 Gödelisierung

$\hat{M}$ beliebige feste TM. Zu einem Wort w gehörige TM M_w ist

$$M_w := \begin{cases} M, & w \text{ Kodierung von } M \\ \hat{M} & \text{sonst} \end{cases}$$

4.9.5 Spezielles Halteproblem

Hält M_w bei Eingabe w?

$$K := \{w \in \{0,1\}^* | M_w[w] \downarrow\} \tag{4}$$

Nicht entscheidbar. (Beweisidee: Wenn $M_w[w]$ nicht hält, dann kann TM, die dies berechnen soll nicht terminieren.)

4.9.6 Allgemeines Halteproblem

$$H := \{w\#x | M_w[x] \downarrow\}$$

Nicht entscheidbar. (Sonst müsste auch K entscheidbar sein)

4.9.7 Reduktion

$A \subseteq \Sigma^*$ reduzierbar auf $B \subseteq \Gamma^*$ ($A \leq B$), gdw $f : \Sigma^* \to \Gamma^*$ total und berechenbar mit

$$\forall w \in \Sigma^*.w \in A \to f(w) \in B$$

- B mindestens so schwer zu lösen, wie A
- A untentscheidbar $\Rightarrow$ B untentscheidbar
- B entscheidbar $\Rightarrow$ A entscheidbar

4.9.8 Halteproblem auf leerem Band

$$H_0 := \{w \in \{0,1\}^* | M_w[\epsilon] \downarrow\}$$

Nicht entscheidbar. (Überschreibe $f(w)$ überschreibe Eingabe mit w, führe M_w aus. Reduktion auf unentscheidbare Menge)

4.9.9 Fazit

- Keine allgemeine algorithmische Methode um zu entscheiden, ob ein Programm terminiert
- Unentscheidbarkeit lässt sich häufig über Reduktion auf Hatleproblem lösen
- Nicht alle unentscheidbaren Probleme sind gleich schwer
- Äquivalenzproblem schwerer, als Halteproblem

4.9.10 Hilberts 10. Problem

Es ist unentscheidbar, ob ein Polynom in n Variablen mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat.

4.10 Semi-Entscheidbarkeit

A ist semi-entscheidbar gdw,

$$\mathcal{X}'_A(x) := \begin{cases} 1, & x \in A \\ \perp, & x \notin A \end{cases}$$

berechenbar ist.

A ist entscheidbar, gdw A und $\overline{A}$ semi-entscheidbar.

Semi-Entscheidbarkeit nicht abgeschlossen unter Komplement. Z.B. $\overline{K}$ ist nicht semi-entscheidbar.

4.10.1 Rekursive Aufzählbarkeit

A heißt rekursiv aufzählbar, gdw $A = \emptyset$ oder $F : \mathbb{N} \to A$ existiert mit

$$A = \{f(0), f(1), f(2), ...\}$$

- Elemente dürfen mehrfach auftreten ($f(i) = f(j), i \neq j$
- Reihenfolge ist beliebig
- Rekursiv Aufzählbar $\Rightarrow$ abzählbar (Aber keine Äquivalenz)
- Rekursiv Aufzählbar $\Leftrightarrow$ semi-entscheidbar

4.10.2 Äquivalenz

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- $\mathcal{X}'_A$ ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion
- A ist Wertebereich einer berechenbaren Funktion

4.10.3 Universelle Turingmaschine

$K = \{w | M_w[w] \downarrow\}$ ist semi-entscheidbar. (Selbe TM, nur Ausgabe 1) Interpreter für Turingsmaschinen als Turingmaschine programmierbar.

4.11 Sätze von Rice und Shapiro

Die von TM M_w berechnete Funktion heißt φ_w

4.11.1 Satz von Rice

F eine Menge von berechenbaren Funktionen, nicht trivial $\emptyset \subset F \subset$ Alle berechenbaren Funktionen. Dann ist unentscheidbar, ob $\varphi_w \in F$.

Nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

4.11.2 Satz von Rice-Shapiro

F Menge von berechenbaren Funktionen. Ist $C_F := \{w|\varphi_w \in F\}$ semi-entscheidbar, so gilt für alle berechenbaren f: $f \in F \Leftrightarrow$ es gibt eine Endliche Teilfunktion $g \subseteq f$ mit $g \in F$

4.11.3 Termination

Ein Programm heißt terminierend, gdw. es für alle Eingaben hält

- Menge der terminierenden Programme ist nicht semi-entscheidbar
- Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar

4.12 Postkutsche[1] Korrespondenzproblem (PCP)

Endliche Folge $(x_1, y_1), ..., (x_k, y_k)$, $x_i, y_i \in \Sigma^+$. Gibt es eine folge von Indiezes $i_1 ..., i_n \in \{1, ..., k\}, n > 0$, so dass $x_{i_1} ... x_{i_n} = y_{i_1} ... y_{i_n}$?

Zähle mögliche Lösungen auf, und prüfe Lösung $\Rightarrow$ semi-entscheidbar.

4.12.1 Modifiziertes PCP

Lösung für $i_1 = 1$?

$$MPCP \leq PCP$$

$$H \leq MCPC$$

$\Rightarrow$ PCP (auch für $\Sigma = \{0, 1\}$) unentscheidbar.

4.12.2 Bemerkungen

- PCP $k \leq 2$ entscheidbar, $2 < k < 7$ unbekannt, $7 \leq k$ unentscheidbar
- PCP $|\Sigma| = 1$ entscheidbar.

4.13 Unentscheidbare CFG-Probleme

- Für DFAs fast alles entscheidbar $L(A) = \emptyset, L(A) = L(B)$
- Für TMs fast nichts entscheidbar $L(M) = \emptyset, L(M_1) = L(M_2)$
- Für CFGs (Kontextfreie Grammatiken) manches entscheidbar ($L(G) = \emptyset$)

Unentscheidbar für CFGs:

- $L(G_1) \cap L(G_2) = \emptyset$?
- $|L(G_1) \cap L(G_2)| = \infty$?
- $L(G_1) \cap L(G_2)$ kontextfrei ?

[1] Dieser Verschreiber hat sich in die erste Version der Zusammenfassung eingeschlichen. Ein Tutor möchte sehen wieviele in der Klausur PCP als Postkutsche Korrespondenzproblem bezeichnen. Deshalb bleibt die Überschrift so bestehen. Das Problem, welches von einem Herrn Post beschrieben wurde ist allerdings als Postsches Korrespondenzproblem bekannt

- $L(G_1) \subseteq L(G_2)$?
- $L(G_1) = L(G_2)$?
- G mehrdeutig ?
- $L(G)$ regulär ?
- $L(G)$ deterministisch (DCFL) ?
- $L(G) = L(\alpha)$? (α ist Regulärer Ausdruck)

5 Komplexitätstheorie

- P = von DTM (Deterministische Mehrband-TM) in polynomieller Zeit lösbaren Probleme
- NP = von NTM (Nichtdeterministische Mehrband-TM) in polynomieller Zeit lösbaren Probleme (Exponentielle Zeit bei DTM)
- P = NP? Such- & Optimierungsproblen in NP $\Rightarrow$ Alle gut lösbar, wenn eines gut lösbar (in P)

5.1 Komplexitätsklasse P

- $time_M(w)$ = Anzahl der Schritte ($\mathbb{N} \cup \{\infty\}$) bis DTM $M[w]$ hält
- Klasse der in $f(n)$ entscheidbaren Sprachen:

  $\text{TIME}(f(n)) = \{A \in \Sigma^* | \exists \text{DTM } M. A = L(M) \wedge \forall w \in \Sigma^*. time_M(w) \leq f(|w|)\}$

- $P = \bigcup_{p \text{ Polynom}} \text{TIME}(p(n))$
- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$
- $A \notin P$ meist schwer zu beweisen

5.2 Komplexitätsklasse NP

- $ntime_M(w)$ = minimale Schritte bis NTM $M[w]$ akzeptiert bzw. 0, falls $w \notin L(M)$
- $f : \mathbb{N} \to \mathbb{N}$ total

  $\text{NTIME}(f(n)) = \{A \subseteq \Sigma^* | \exists \text{NTM } M. A = L(M) \wedge \forall w \in \Sigma^*. ntime_M(w) \leq f(|w|)\}$

- $NP = \bigcup_{p \text{ Polynom}} \text{NTIME}(p(n))$

5.3 Verifikator

Lösung schwer zu finden, aber Lösungsvorschlag leicht zu überprüfen. M sei DTM mit $L(M) \subseteq \{w\#c | w \in \Sigma^*, c \in \Delta^*\}$

- $w\#c \in L(M) \rightarrow c$ ist Zertifikat für w
- M ist polynomiell beschränkter Verifikator für Sprache $\{w \in \Sigma^* | \exists c \in \Sigma^*.w\#c \in L(M)\}$, wenn Polynom p existiert mit $time_M(w\#c) \leq p(|w|)$
- $A \in NP$ gdw ein polynomiell beschränkter Verifikator für A existiert

5.4 P, NP und LOOP

- wenn $A \in$ TIME(f) und f LOOP-berechenbar $\Rightarrow A$ LOOP-entscheidbar
- Alle Sprachen in P und NP sind LOOP-entscheidbar

5.5 NP-Vollständigkeit

5.5.1 Polynomielle Reduzierbarkeit

$A \subseteq \Sigma^*, B \subseteq \Gamma^*$, A polynomiell reduzierbar auf B, $A \leq_p B$, gdw totale Funktion $f : \Sigma^* \rightarrow \Gamma^*$, von DTM in polynomieller Zeit berechenbar, wenn für alle $w \in \Sigma^*$ gilt:

$$w \in A \Leftrightarrow f(w) \in B$$

- $\leq_p$ ist transitiv
- P und NP nach unten abgeschlossen $A \leq_p B \in P/NP \Rightarrow A \in P/NP$

5.5.2 NP-hart

Ein Problem ist NP-hart, wenn es mindestens so schwer ist, wie alles in NP. L NP-hart gdw $A \leq_p L. \forall A \in NP$.

5.5.3 NP-vollständig

L ist NP-vollständig, gdw L NP-hart und $L \in$ NP

5.6 SAT (satisfiability)

- Ist eine Aussagenlogische Formel F erfüllbar?
- $F_1 \equiv F_2$ gdw $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar
- SAT NP-vollständig (Satz von Cook)
- 3 Färbarkeit (3COL – Färbung von Knoten, keine benachbarten Knoten selbe Farbe) 3COL $\leq_p$ SAT

5.6.1 Weitere NP-vollständige Probleme

Zeige, dass B NP-vollständig:

- $B \in$ NP
- $A \leq_p B$ für A ist NP-vollständig

5.6.2 3KNF

- Konjunktive Normalform $K_1 \wedge \ldots \wedge K_n$
- Klauseln $K_i = L_1 \vee \ldots \vee L_m$
- Literale L_j (negierte) Variable
- 3KNF gdw Klauseln ≤ 3 Literale
- SAT $\leq_p$ KNF-SAT
- 2KNF $\in$ P

5.6.3 MÜ

- Mengenüberdeckung
- Deckt eine bestimmte Anzahl an Teilmengen eine Menge ab?

Minimierungsproblem

- Binäre Suche: Finde kleinstes k, welches MÜ erfüllt.

5.6.4 CLIQUE

- Besitzt Ungerichteter Graph einen Teilgraph mit bestimmter Größe, so dass alle Knoten benachbart sind?
- 3KNF-SAT $\leq_p$ CLIQUE

5.6.5 RUCKSACK

- Auswahl von Werten, so dass Summe eine bestimmte Zahl ergibt
- 3KNF-SAT $\leq_p$ RUCKSACK

5.6.6 PARTITION

- Zahl als Summe von Zahlen
- RUCKSACK leq_p PARTITION

5.6.7 BIN PACKING

- Aufteilung von Objekten auf Behälter, dass keiner überläuft
- PARTITION $\leq_p$ BIN PACKING

5.6.8 HAMILTON

- Enthält Graph Hamilton-Kreis (Geschlossener Pfad, der jeden Knoten genau einmal enhält)

5.6.9 TRAVELLING SALESMAN - TSP

- Hamilton Kreis mit Länge $< k$
- HAMILTON $\leq_p$ TSP

5.6.10 FÄRBBARKEIT COL

- Färbung von Graph mit k Farben, keine zwei benachbarten Knoten selbe
- 3KNF-SAT $\leq_p$ 3COL
- 2COL $\in$ P

5.7 Unvollständigkeit der Arithmetik

- Kein korrektes & vollständige Beweissistem für die Arithmetik (Gödel)