

Einführung in die Theoretische Informatik

Johannes Köbler



Institut für Informatik
Humboldt-Universität zu Berlin

WS 2013/14

Definition

- Eine NTM M **hält** bei Eingabe x (kurz: $M(x) = \downarrow$), falls alle Rechnungen von $M(x)$ eine endliche Länge haben.
- Falls $M(x)$ nicht hält, schreiben wir auch kurz $M(x) = \uparrow$.
- Eine NTM M **entscheidet** eine Eingabe x , falls $M(x)$ hält oder eine Konfiguration mit einem Endzustand erreichen kann.
- Eine Sprache $L \subseteq \Sigma^*$ heißt **entscheidbar**, falls eine DTM M mit $L(M) = L$ existiert, die jede Eingabe $x \in \Sigma^*$ entscheidet.
- Jede von einer DTM M erkannte Sprache heißt **semi-entscheidbar**.

Bemerkung

- Die von M akzeptierte Sprache $L(M)$ heißt semi-entscheidbar, da M zwar alle Eingaben $x \in L$ entscheidet (aber eventuell nicht alle $x \in \bar{L}$).
- Später werden wir sehen, dass genau die Typ-0 Sprachen semi-entscheidbar sind.

Definition

- Eine k -DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine Funktion $f : \Sigma^* \rightarrow \Gamma^*$, falls M bei jeder Eingabe $x \in \Sigma^*$ in einer Konfiguration

$$K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k) \text{ mit } u_k = f(x)$$

hält (d.h. $K_x \vdash^* K$ und K hat keine Folgekonfiguration).

- Hierfür sagen wir auch, M gibt bei Eingabe x das Wort $f(x)$ aus und schreiben $M(x) = f(x)$.
- f heißt **Turing-berechenbar** (oder einfach **berechenbar**), falls es eine k -DTM M mit $M(x) = f(x)$ für alle $x \in \Sigma^*$ gibt.
- Aus historischen Gründen werden berechenbare Funktionen auch **rekursiv** (engl. *recursive*) genannt.

Definition

- Eine **partielle Funktion** hat die Form $f : \Sigma^* \rightarrow \Gamma^* \cup \{\uparrow\}$.
- Für $f(x) = \uparrow$ sagen wir auch $f(x)$ ist **undefiniert**.
- Der **Definitionsbereich** (engl. *domain*) von f ist

$$\text{dom}(f) = \{x \in \Sigma^* \mid f(x) \neq \uparrow\}.$$

- Das **Bild** (engl. *image*) von f ist

$$\text{img}(f) = \{f(x) \mid x \in \text{dom}(f)\}.$$

- f heißt **total**, falls $\text{dom}(f) = \Sigma^*$ ist.
- Eine k -DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** f , falls $M(x)$ für alle $x \in \text{dom}(f)$ das Wort $f(x)$ ausgibt und für alle $x \notin \text{dom}(f)$ keine Ausgabe berechnet (d.h. $M(x) = \uparrow$).

Wir fassen die entscheidbaren Sprachen und die (partiellen) berechenbaren Funktionen in folgenden Klassen zusammen:

$REC = \{L(M) \mid M \text{ ist eine DTM, die jede Eingabe entscheidet}\},$

$FREC = \{f \mid f \text{ ist eine berechenbare (totale) Funktion}\},$

$FREC_p = \{f \mid f \text{ ist eine berechenbare partielle Funktion}\}.$

Dann gilt:

- $FREC \not\subseteq FREC_p$ und
- $REG \not\subseteq DCFL \not\subseteq CFL \not\subseteq DCSL \subseteq CSL \not\subseteq REC \not\subseteq RE.$

Beispiel

- Bezeichne x^+ den **lexikografischen Nachfolger** von $x \in \Sigma^*$.
- Für $\Sigma = \{0, 1\}$ ergeben sich beispielsweise folgende Werte:

x	ε	0	1	00	01	10	11	000	...
x^+	0	1	00	01	10	11	000	001	...

- Betrachte die auf Σ^* definierten partiellen Funktionen f_1, f_2, f_3, f_4 mit

$$\begin{aligned}
 f_1(x) &= 0, \\
 f_2(x) &= x, \\
 f_3(x) &= x^+
 \end{aligned}
 \quad \text{und} \quad
 f_4(x) = \begin{cases} \uparrow, & x = \varepsilon, \\ y, & x = y^+. \end{cases}$$

- Da f_1, f_2, f_3, f_4 berechenbar sind, gehören die totalen Funktionen f_1, f_2, f_3 zu FREC und die partielle Funktion f_4 zu FREC_p . \triangleleft

Satz

- Eine Sprache $A \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn ihre **charakteristische Funktion** $\chi_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist:

$$\chi_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

- Eine Sprache $A \subseteq \Sigma^*$ ist genau dann semi-entscheidbar, falls ihre **partielle charakteristische Funktion** $\hat{\chi}_A : \Sigma^* \rightarrow \{1, \uparrow\}$ berechenbar ist:

$$\hat{\chi}_A(x) = \begin{cases} 1, & x \in A, \\ \uparrow, & x \notin A. \end{cases}$$

Beweis

Siehe Übungen.

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **rekursiv aufzählbar**, falls $A = \emptyset$ oder das Bild $\text{img}(f)$ einer berechenbaren Funktion $f : \Gamma^* \rightarrow \Sigma^*$ ist.

Satz

Folgende Eigenschaften sind äquivalent:

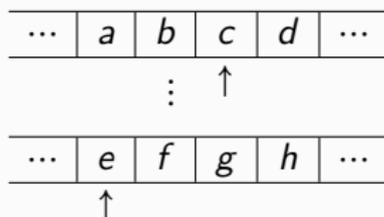
- 1 A ist semi-entscheidbar (d.h. A wird von einer DTM akzeptiert),
- 2 A wird von einer 1-DTM akzeptiert,
- 3 A wird von einer 1-NTM akzeptiert,
- 4 A ist vom Typ 0,
- 5 A wird von einer NTM akzeptiert,
- 6 A ist rekursiv aufzählbar.

Beweis

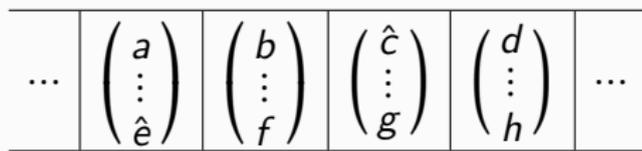
Die Implikation 2 \Rightarrow 3 ist klar. Die Implikationen 3 \Rightarrow 4 \Rightarrow 5 werden in den Übungen gezeigt. Hier zeigen wir 1 \Rightarrow 2 und 5 \Rightarrow 6 \Rightarrow 1.

Beweis von ① \Rightarrow ②: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k -DTM mit $L(M) = A$.
- Wir konstruieren eine 1-DTM $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$ für A .
- M' simuliert M , indem sie jede Konfiguration K von M der Form



durch eine Konfiguration K' folgender Form nachbildet:



Simulation einer k -DTM durch eine 1-DTM

Beweis von ① \Rightarrow ②: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Das heißt, M' arbeitet mit dem Alphabet

$$\Gamma' = \Gamma \cup (\Gamma \cup \{\hat{a} \mid a \in \Gamma\})^k$$

- und erzeugt bei Eingabe $x = x_1 \dots x_n \in \Sigma^*$ zuerst die der Startkonfiguration

$$K_x = (q_0, \varepsilon, x_1, x_2 \dots x_n, \varepsilon, \sqcup, \varepsilon, \dots, \varepsilon, \sqcup, \varepsilon)$$

von M bei Eingabe x entsprechende Konfiguration

$$K'_x = q'_0 \begin{pmatrix} \hat{x}_1 \\ \hat{\sqcup} \\ \vdots \\ \hat{\sqcup} \end{pmatrix} \begin{pmatrix} x_2 \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix} \dots \begin{pmatrix} x_n \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix}.$$

Beweis von ① \Rightarrow ②: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Dann simuliert M' jeweils einen Schritt von M durch folgende Sequenz von Rechenschritten:
 - Zuerst geht M' solange nach rechts, bis sie alle mit \wedge markierten Zeichen (z.B. $\hat{a}_1, \dots, \hat{a}_k$) gefunden hat.
 - Diese Zeichen speichert M' in ihrem Zustand.
 - Anschließend geht M' wieder nach links und realisiert dabei die durch $\delta(q, a_1, \dots, a_k)$ vorgegebene Anweisung von M .
 - Den aktuellen Zustand q von M speichert M' ebenfalls in ihrem Zustand.
- Sobald M in einen Endzustand übergeht, wechselt M' ebenfalls in einen Endzustand und hält.
- Nun ist leicht zu sehen, dass $L(M') = L(M)$ ist. □

Beweis von ⑤ \Rightarrow ⑥: $\{L(M) \mid M \text{ ist eine NTM}\} \subseteq \{L \mid L \text{ ist rek. aufzählbar}\}$

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k -NTM und sei $A = L(M) \neq \emptyset$.
- Sei $\tilde{\Gamma}$ das Alphabet $Z \cup \Gamma \cup \{\#\}$.
- Wir kodieren eine Konfiguration $K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$ durch das Wort

$$\text{code}(K) = \#q\#u_1\#a_1\#v_1\#\dots\#u_k\#a_k\#v_k\#$$

und eine Rechnung $K_0 \vdash \dots \vdash K_t$ durch $\text{code}(K_0) \dots \text{code}(K_t)$.

- Dann lassen sich die Wörter von A durch folgende Funktion $f : \tilde{\Gamma}^* \rightarrow \Sigma^*$ aufzählen (dabei ist x_0 ein beliebiges Wort in A):

$$f(x) = \begin{cases} y, & x \text{ kodiert eine Rechnung } K_0 \vdash \dots \vdash K_t \text{ von } M \text{ mit} \\ & K_0 = K_y \text{ und } K_t \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k \\ x_0, & \text{sonst.} \end{cases}$$

- Da f berechenbar ist, ist $A = \text{img}(f)$ rekursiv aufzählbar. □

Beweis von ⑥ \Rightarrow ①: $\{L \mid L \text{ ist rek. aufzählbar}\} \subseteq \{L(M) \mid M \text{ ist eine DTM}\}$

- Sei $f : \Gamma^* \rightarrow \Sigma^*$ eine Funktion mit $A = \text{img}(f)$ und sei M eine k -DTM, die f berechnet.
- Betrachte folgende $(k + 1)$ -DTM M' , die bei Eingabe x
 - auf dem 2. Band der Reihe nach alle Wörter y in Γ^* erzeugt,
 - für jedes y den Wert $f(y)$ durch Simulation von $M(y)$ berechnet, und
 - ihre Eingabe x akzeptiert, sobald $f(y) = x$ ist. □

Satz

A ist genau dann entscheidbar, wenn A und \bar{A} semi-entscheidbar sind, d.h. $\text{REC} = \text{RE} \cap \text{co-RE}$.

Beweis.

- Falls A entscheidbar ist, ist auch \bar{A} entscheidbar, d.h. A und \bar{A} sind dann auch semi-entscheidbar.
- Für die Rückrichtung seien $f_1, f_2 : \Gamma^* \rightarrow \Sigma^*$ Turing-berechenbare Funktionen mit $\text{img}(f_1) = A$ und $\text{img}(f_2) = \bar{A}$.
- Wir betrachten folgende DTM M , die bei Eingabe x für jedes $y \in \Gamma^*$ die beiden Werte $f_1(y)$ und $f_2(y)$ bestimmt und im Fall
 - $f_1(y) = x$ in einem Endzustand hält,
 - $f_2(y) = x$ in einem Nichtendzustand hält.
- Da jede Eingabe x entweder in $\text{img}(f_1) = A$ oder in $\text{img}(f_2) = \bar{A}$ enthalten ist, hält M bei allen Eingaben. □

Kodierung (Gödelisierung) von Turingmaschinen

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine 1-DTM mit
 - Zustandsmenge $Z = \{q_0, \dots, q_m\}$ (o.B.d.A. sei $E = \{q_m\}$),
 - Eingabealphabet $\Sigma = \{0, 1, \#\}$ und
 - Arbeitsalphabet $\Gamma = \{a_0, \dots, a_l\}$, wobei wir o.B.d.A. $a_0 = \sqcup$, $a_1 = 0$, $a_2 = 1$ und $a_3 = \#$ annehmen.
- Dann können wir jede Anweisung der Form $q_i a_j \rightarrow q_{i'} a_{j'} D$ durch das Wort

$$\# \text{bin}(i) \# \text{bin}(j) \# \text{bin}(i') \# \text{bin}(j') \# b_D \#$$

kodieren.

- Dabei ist $\text{bin}(n)$ die Binärdarstellung von n und

$$b_D = \begin{cases} 0, & D = N, \\ 1, & D = L, \\ 10, & D = R. \end{cases}$$

Kodierung von Turingmaschinen

- M lässt sich nun als ein Wort über dem Alphabet $\{0, 1, \#\}$ kodieren, indem wir die Anweisungen von M in kodierter Form auflisten.
- Kodieren wir die Zeichen $0, 1, \#$ binär (z.B. $0 \mapsto 00, 1 \mapsto 11, \# \mapsto 10$), so gelangen wir zu einer Binärkodierung w_M von M .
- Die Zahl $(w_M)_2$ wird auch die **Gödel-Nummer** von M genannt.
- M_w ist durch Angabe von w bis auf die Benennung ihrer Zustände und Arbeitszeichen eindeutig bestimmt.
- Ganz analog lassen sich auch DTMs mit einer beliebigen Anzahl von Bändern (sowie NTMs, Konfigurationen oder Rechnungen von TMs) binär kodieren.
- Umgekehrt können wir jedem Binärstring $w \in \{0, 1\}^*$ eine DTM M_w wie folgt zuordnen:

$$M_w = \begin{cases} M, & \text{falls eine DTM } M \text{ mit } w_M = w \text{ existiert,} \\ M_0, & \text{sonst (dabei sei } M_0 \text{ eine beliebige DTM).} \end{cases}$$

Unentscheidbarkeit des Halteproblems

Definition

- Das **Halteproblem** ist die Sprache

$$H = \left\{ w \# x \mid \begin{array}{l} w, x \in \{0, 1\}^* \text{ und} \\ \text{die DTM } M_w \text{ h\"alt} \\ \text{bei Eingabe } x \end{array} \right\}$$

- Das **spezielle Halteproblem** ist

$$K = \left\{ w \in \{0, 1\}^* \mid \begin{array}{l} \text{die DTM } M_w \\ \text{h\"alt bei Eingabe } w \end{array} \right\}$$

χ_H	x_1	x_2	x_3	\dots
w_1	0	1	0	\dots
w_2	0	1	1	\dots
w_3	1	1	0	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

χ_K				
w_1	0			
w_2		1		
w_3			0	
\vdots				\ddots

Satz

$K \in \text{RE} \setminus \text{co-RE}$.

Beweis von $K \in RE$

- Sei w_0 die Kodierung einer DTM, die bei jeder Eingabe (sofort) hält und betrachte die Funktion

$$f(x) = \begin{cases} w, & x \text{ ist Kodierung einer haltenden Rechnung einer} \\ & \text{DTM } M_w \text{ bei Eingabe } w, \\ w_0, & \text{sonst.} \end{cases}$$

- Da f berechenbar und $\text{img}(f) = K$ ist, folgt $K \in RE$. □

Bemerkung

Ganz ähnlich lässt sich $H \in RE$ zeigen.

Beweisidee

- Da $K \in \text{RE}$ ist, gibt es in der Matrixdarstellung von χ_H eine Zeile w (bzw. eine DTM $M = M_w$), die mit der Diagonalen der Matrix übereinstimmt.
- Beispielsweise können wir für M eine DTM wählen, die die partielle charakteristische Funktion $\hat{\chi}_K$ von K berechnet.
- Wäre auch $\bar{K} \in \text{RE}$, so würde eine DTM M' existieren, deren Zeile invers zur Zeile von M und damit zur Diagonalen ist.
- Unter der Annahme $\bar{K} \in \text{RE}$ können wir nämlich für M' eine DTM wählen, die die partielle charakteristische Funktion $\hat{\chi}_{\bar{K}}$ von \bar{K} berechnet.
- Da in keiner Matrix eine Zeile existieren kann, die invers zur Diagonalen ist, führt dies auf den gewünschten Widerspruch.

Unentscheidbarkeit des speziellen Halteproblems

Beweis von $\bar{K} \notin \text{RE}$

- Angenommen, die Sprache

$$\bar{K} = \{w \mid M_w(w) \text{ h\"alt nicht}\} \quad (*)$$

w\"are semi-entscheidbar.

- Dann existiert eine DTM $M_{w'}$, die die partielle charakteristische Funktion $\hat{\chi}_{\bar{K}}$ von \bar{K} berechnet, d.h. es gilt

$$M_{w'}(w) \text{ h\"alt} \Leftrightarrow w \in \bar{K} \quad (**)$$

- F\"ur die Kodierung w' von $M_{w'}$ folgt dann aber

$$w' \in \bar{K} \stackrel{(*)}{\Leftrightarrow} M_{w'}(w') \text{ h\"alt nicht} \stackrel{(**)}{\Leftrightarrow} w' \notin \bar{K} \quad \downarrow \text{ (Widerspruch!)} \quad \square$$

χ_H	x_1	x_2	x_3	\dots
w_1	0	1	0	\dots
w_2	0	1	1	\dots
w_3	1	1	0	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

w'	1	0	1	\dots
------	---	---	---	---------

Korollar

$\text{REC} \not\subseteq \text{RE}$.

Beweis

Klar, da $K \in \text{RE} - \text{REC}$. □

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt auf $B \subseteq \Gamma^*$ **reduzierbar** (kurz: $A \leq B$), falls eine berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ ex., so dass gilt:

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B.$$

Beispiel

Es gilt $K \leq H$ mittels $f : w \mapsto w\#w$, da für alle $w \in \{0,1\}^*$ gilt:

$$\begin{aligned} w \in K &\Leftrightarrow M_w \text{ ist eine DTM, die bei Eingabe } w \text{ hält} \\ &\Leftrightarrow w\#w \in H. \end{aligned}$$



Abschluss von REC unter \leq

Definition

- Eine Sprachklasse \mathcal{C} heißt **unter \leq abgeschlossen**, wenn für alle Sprachen A, B gilt:

$$A \leq B \wedge B \in \mathcal{C} \Rightarrow A \in \mathcal{C}.$$

Satz

Die Klassen REC und RE sind unter \leq abgeschlossen.

Beweis

- Gelte $A \leq B$ mittels f und sei $B \in \text{REC}$.
- Dann ex. eine DTM M , die χ_B berechnet.
- Betrachte folgende DTM M' :
 - M' berechnet bei Eingabe x zuerst den Wert $f(x)$ und
 - simuliert dann M bei Eingabe $f(x)$.

Abschluss von REC und RE unter \leq

Satz

Die Klasse REC ist unter \leq abgeschlossen.

Beweis.

- Gelte $A \leq B$ mittels f und sei $B \in \text{REC}$.
- Dann ex. eine DTM M , die χ_B berechnet.
- Betrachte folgende DTM M' :
 - M' berechnet bei Eingabe x zuerst den Wert $f(x)$ und
 - simuliert dann M bei Eingabe $f(x)$.
- Wegen $x \in A \Leftrightarrow f(x) \in B$ ist $\chi_A(x) = \chi_B(f(x))$ und daher folgt
$$M'(x) = M(f(x)) = \chi_B(f(x)) = \chi_A(x).$$
- Also berechnet M' die Funktion χ_A , d.h. $A \in \text{REC}$. □

Bemerkung

Der Abschluss von RE unter \leq folgt analog (siehe Übungen).

Der Vollständigkeitsbegriff

Definition

- Eine Sprache B heißt **hart** für eine Sprachklasse \mathcal{C} (kurz: **\mathcal{C} -hart** oder **\mathcal{C} -schwer**), falls jede Sprache $A \in \mathcal{C}$ auf B reduzierbar ist:

$$\forall A \in \mathcal{C} : A \leq B.$$

- Eine \mathcal{C} -harte Sprache B , die zu \mathcal{C} gehört, heißt **\mathcal{C} -vollständig**.

Beispiel

Das Halteproblem H ist RE-vollständig. Es gilt nämlich

- $H \in \text{RE}$ und
- $\forall A \in \text{RE} : A \leq H$

mittels der Reduktionsfunktion $x \mapsto w\#x$, wobei w die Kodierung einer DTM M_w ist, die $\hat{\chi}_A$ berechnet.



Bemerkung

Auch das spezielle Halteproblem K ist RE-vollständig (siehe Übungen).

H ist nicht entscheidbar

Korollar

- $A \leq B \wedge A \notin \text{REC} \Rightarrow B \notin \text{REC}$,
- $A \leq B \wedge A \notin \text{RE} \Rightarrow B \notin \text{RE}$.

Beweis

Aus der Annahme, dass B entscheidbar (bzw. semi-entscheidbar) ist, folgt wegen $A \leq B$, dass dies auch auf A zutrifft (Widerspruch). \square

Bemerkung

Wegen $K \leq H$ überträgt sich somit die Unentscheidbarkeit von K auf H .

Korollar

$H \notin \text{REC}$.

Das Halteproblem bei leerem Band

Definition

Das **Halteproblem bei leerem Band** ist die Sprache

$$H_0 = \left\{ w \in \{0, 1\}^* \mid \begin{array}{l} \text{die DTM } M_w \\ \text{hält bei Eingabe } \varepsilon \end{array} \right\}$$

Satz

H_0 ist RE-vollständig.

Beweis

- $H_0 \in \text{RE}$ folgt wegen $H_0 \leq H \in \text{RE}$ mittels der Reduktionsfunktion $w \mapsto w\#\varepsilon$.

χ_H	x_1	x_2	x_3	\dots
w_1	0	1	0	\dots
w_2	0	1	1	\dots
w_3	1	1	0	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

χ_{H_0}	$x_1 (= \varepsilon)$
w_1	0
w_2	0
w_3	1
\vdots	\vdots

H_0 ist RE-vollständig

Beweis

- $H_0 \in \text{RE}$ folgt wegen $H_0 \leq H \in \text{RE}$ mittels der Reduktionsfunktion $w \mapsto w\#\epsilon$.
- Sei $A \in \text{RE}$ und sei w die Kodierung einer DTM, die $\hat{\chi}_A$ berechnet. Um A auf H_0 zu reduzieren, transformieren wir $x \in \{0,1\}^*$ in die Kodierung w_x einer DTM M , die zunächst ihre Eingabe durch x ersetzt und dann $M_w(x)$ simuliert. Dann gilt

$$x \in A \iff w_x \in H_0$$

und somit $A \leq H_0$ mittels der Reduktionsfunktion $x \mapsto w_x$.

Korollar

$H_0 \notin \text{REC}$.

Frage

- Kann man einer beliebig vorgegebenen DTM ansehen, ob die von ihr berechnete Funktion eine gewisse Eigenschaft hat?
- Kann man beispielsweise entscheiden, ob eine gegebene DTM eine totale Funktion berechnet?

Antwort

Nein (es sei denn, die fragliche Eigenschaft ist trivial, d.h. keine oder jede DTM berechnet eine Funktion mit dieser Eigenschaft).

Definition

- Zu einer Klasse \mathcal{F} von Funktionen definieren wir die Sprache

$$L_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \text{die DTM } M_w \text{ berechnet eine Funktion in } \mathcal{F}\}.$$

- Die Eigenschaft \mathcal{F} heißt **trivial**, wenn $L_{\mathcal{F}} = \emptyset$ oder $L_{\mathcal{F}} = \{0,1\}^*$ ist.

Der Satz von Rice besagt, dass $L_{\mathcal{F}}$ nur für triviale Eigenschaften entscheidbar ist.

Satz (Satz von Rice)

Für jede nicht triviale Eigenschaft \mathcal{F} ist $L_{\mathcal{F}}$ unentscheidbar.

Beispiel

- Die Sprache

$$L = \{w \in \{0, 1\}^* \mid M_w(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

ist unentscheidbar.

- Dies folgt aus dem Satz von Rice, da die Eigenschaft

$$\mathcal{F} = \{f \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

nicht trivial und $L = L_{\mathcal{F}}$ ist.

- \mathcal{F} ist nicht trivial, da z.B. die berechenbare partielle Funktion

$$f(x) = \begin{cases} 0^{n+1}, & x = 0^n \text{ für ein } n \geq 0 \\ \uparrow, & \text{sonst} \end{cases}$$

in \mathcal{F} und die konstante Funktion $g(x) = 0$ nicht in \mathcal{F} enthalten ist.

Der Satz von Rice

Satz (Satz von Rice)

Für jede nicht triviale Eigenschaft \mathcal{F} ist die Sprache $L_{\mathcal{F}}$ unentscheidbar.

Beweis

- Die Idee besteht darin, H_0 auf $L_{\mathcal{F}}$ (oder auf $\bar{L}_{\mathcal{F}}$) zu reduzieren, indem wir für eine gegebene DTM M_w eine DTM $M_{w'}$ konstruieren mit

$$w \in H_0 \Leftrightarrow M_{w'} \text{ berechnet (k)eine Funktion in } \mathcal{F}.$$
- Hierzu lassen wir $M_{w'}$ bei Eingabe x zunächst einmal die DTM M_w bei Eingabe ε simulieren.
- Falls $w \notin H_0$ ist, berechnet $M_{w'}$ also die überall undefinierte Funktion u mit $u(x) = \uparrow$ für alle $x \in \{0, 1, \#\}^*$.
- Damit die Reduktion gelingt, müssen wir nur noch dafür sorgen, dass $M_{w'}$ im Fall $w \in H_0$ eine Funktion f berechnet, die genau dann die Eigenschaft \mathcal{F} hat, wenn u sie nicht hat.
- Da \mathcal{F} nicht trivial ist, gibt es eine DTM M , die eine solche Funktion f berechnet.

Beweis (Schluss)

- Da \mathcal{F} nicht trivial ist, gibt es eine DTM M , die eine solche Funktion f berechnet.
- Betrachte die Reduktionsfunktion
$$h(w) = w', \text{ wobei } w' \text{ die Kodierung einer DTM ist, die bei Eingabe } x \text{ zun\u00e4chst die DTM } M_w(\varepsilon) \text{ simuliert und im Fall, dass } M_w(\varepsilon) \text{ h\u00e4lt, mit der Simulation von } M(x) \text{ fortf\u00e4hrt.}$$
- Dann ist $h : w \mapsto w'$ eine totale berechenbare Funktion und es gilt
$$w \in H_0 \Rightarrow M_{w'} \text{ berechnet } f$$
$$w \notin H_0 \Rightarrow M_{w'} \text{ berechnet } u.$$
- Dies zeigt, dass h das Problem H_0 auf $L_{\mathcal{F}}$ (oder auf $\bar{L}_{\mathcal{F}}$) reduziert, und da H_0 unentscheidbar ist, muss auch $L_{\mathcal{F}}$ unentscheidbar sein. \square

Der Satz von Rice gilt auch für Eigenschaften, die das Akzeptanzverhalten einer gegebenen Turingmaschine betreffen.

Satz (Satz von Rice für Spracheigenschaften)

Für eine beliebige Sprachklasse \mathcal{S} sei

$$L_{\mathcal{S}} = \{w \in \{0, 1\}^* \mid L(M_w) \in \mathcal{S}\}.$$

Dann ist $L_{\mathcal{S}}$ unentscheidbar, außer wenn $L_{\mathcal{S}} \in \{\emptyset, \{0, 1\}^*\}$ ist.

Beweis

Siehe Übungen.

Das Postsche Korrespondenzproblem (PCP)

Definition

- Sei Σ ein beliebiges Alphabet mit $\# \notin \Sigma$.
- Das **Postsche Korrespondenzproblem über Σ** (kurz **PCP $_{\Sigma}$**) ist:
 gegeben: k Paare $(x_1, y_1), \dots, (x_k, y_k)$ von Wörtern in Σ^* .
 gefragt: Gibt es eine Folge $\alpha = (i_1, \dots, i_n)$, $n \geq 1$, von Indizes $i_j \in \{1, \dots, k\}$ mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?
- Das **modifizierte PCP über Σ** (kurz **MPCP $_{\Sigma}$**) fragt nach einer Lösung $\alpha = (i_1, \dots, i_n)$ mit $i_1 = 1$.
- Wir notieren eine PCP-Instanz meist in Form einer Matrix $\begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$ und kodieren sie durch das Wort $x_1 \# y_1 \# \dots \# x_k \# y_k$.

Beispiel

Die Instanz $I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & ab & caa \\ aca & bc & aa \end{pmatrix}$ besitzt wegen

$$x_1 x_3 x_2 x_3 = acaabcaa$$

$$y_1 y_3 y_2 y_3 = acaabcaa$$

die PCP-Lösung $\alpha = (1, 3, 2, 3)$, die auch eine MPCP-Lösung ist.

Das Postsche Korrespondenzproblem

Lemma

Für jedes Alphabet Σ gilt $\text{PCP}_\Sigma \leq \text{PCP}_{\{a,b\}}$.

Beweis

- Sei $\Sigma = \{a_1, \dots, a_m\}$. Für ein Zeichen $a_i \in \Sigma$ sei $c(a_i) = ab^{i-1}$ und für ein Wort $w = w_1 \dots w_n \in \Sigma^*$ mit $w_i \in \Sigma$ sei $c(w) = c(w_1) \dots c(w_n)$.
- Dann folgt $\text{PCP}_\Sigma \leq \text{PCP}_{\{a,b\}}$ mittels der Reduktionsfunktion

$$f : \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix} \mapsto \begin{pmatrix} c(x_1) \dots c(x_k) \\ c(y_1) \dots c(y_k) \end{pmatrix}.$$

□

Beispiel

Sei $\Sigma = \{0, 1, 2\}$. Dann ist $c(0) = a$, $c(1) = ab$ und $c(2) = abb$. Somit ist

$$f \left(\begin{pmatrix} 0 & 01 & 200 \\ 020 & 12 & 00 \end{pmatrix} \right) = \begin{pmatrix} a & aab & abbbaa \\ aabba & ababb & aa \end{pmatrix}.$$

Das Postsche Korrespondenzproblem

Im Folgenden lassen wir im Fall $\Sigma = \{a, b\}$ den Index weg und schreiben einfach PCP (bzw. MPCP).

Satz

$\text{MPCP} \leq \text{PCP}$.

Beweis

- Wir zeigen $\text{MPCP} \leq \text{PCP}_\Sigma$ für $\Sigma = \{a, b, \langle, |, \rangle\}$.
- Für ein Wort $w = w_1 \dots w_n$ sei

$$\begin{array}{cccc}
 \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} \\
 \langle w_1 | \dots | w_n \rangle & \langle w_1 | \dots | w_n \rangle & | w_1 | \dots | w_n & w_1 | \dots | w_n \rangle
 \end{array}$$

Beweis von MPCP \leq PCP

- Für ein Wort $w = w_1 \dots w_n$ sei

$$\begin{array}{cccc} \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} \\ \hline \langle |w_1| \dots |w_n| & w_1 | \dots |w_n| & \langle |w_1| \dots |w_n| & |w_1| \dots |w_n| \end{array}$$

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ :

$$f : \begin{pmatrix} x_1 & \dots & x_k \\ y_1 & \dots & y_k \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \dots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots & \overleftarrow{y_k} & | \rangle \end{pmatrix}$$

- Dabei nehmen wir an, dass $(x_i, y_i) \neq (\varepsilon, \varepsilon)$ ist, da wir diese Paare im Fall $i > 1$ einfach weglassen und im Fall $i = 1$ $f(I) = \begin{pmatrix} a \\ a \end{pmatrix}$ setzen können.

Beispiel

$$f : \begin{pmatrix} aa & b & bab & bb \\ aab & bb & a & b \end{pmatrix} \mapsto \begin{pmatrix} \langle |a|a| & a|a| & b| & b|a|b| & b|b| & \rangle \\ \langle |a|a|b| & |a|a|b| & |b|b| & |a| & |b| & | \rangle \end{pmatrix}$$

- Für ein Wort $w = w_1 \dots w_n$ sei

$$\begin{array}{cccc}
 \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} & \overleftarrow{w} \\
 \hline
 \langle |w_1| \dots |w_n| & w_1 \dots w_n & \langle |w_1| \dots |w_n| & |w_1| \dots |w_n|
 \end{array}$$

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ :

$$f : \begin{pmatrix} x_1 & \dots & x_k \\ y_1 & \dots & y_k \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \dots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots & \overleftarrow{y_k} & | \rangle \end{pmatrix}$$

- Da jede MPCP-Lösung $\alpha = (1, i_2, \dots, i_n)$ für I auf eine PCP-Lösung $\alpha' = (1, i_2 + 1, \dots, i_n + 1, k + 2)$ für $f(I)$ führt, folgt

$$I \in \text{MPCP} \Rightarrow f(I) \in \text{PCP}_\Sigma.$$

Beweis von $\text{MPCP} \leq \text{PCP}$

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ :

$$f : \begin{pmatrix} x_1 & \dots & x_k \\ y_1 & \dots & y_k \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \dots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots & \overleftarrow{y_k} & | \rangle \end{pmatrix}$$

- Für die umgekehrte Implikation sei $\alpha' = (i_1, \dots, i_n)$ eine PCP-Lösung für

$$f(I) = \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \dots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots & \overleftarrow{y_k} & | \rangle \end{pmatrix}.$$

- Dann muss $i_1 = 1$ und $i_n = k + 2$ sein, da das Lösungswort mit \langle beginnen und mit \rangle enden muss (und $f(I)$ nicht das Paar $(\varepsilon, \varepsilon)$ enthält).
- Wählen wir α' von minimaler Länge, so ist $i_j \in \{2, \dots, k + 1\}$ für $j = 2, \dots, n - 1$.
- Dann ist aber $\alpha = (i_1, i_2 - 1, \dots, i_{n-1} - 1)$ eine MPCP-Lösung für I . \square

Das Postsche Korrespondenzproblem

Satz

PCP ist RE-vollständig und damit unentscheidbar.

Beweis.

- Es ist leicht zu sehen, dass $PCP \in RE$ ist.
- Um zu zeigen, dass PCP RE-hart ist, sei A eine beliebige Sprache in RE und sei $G = (V, \Sigma, P, S)$ eine Typ-0 Grammatik für A .
- Wir zeigen $A \leq MPCP_{\Gamma}$ für $\Gamma = V \cup \Sigma \cup \{\langle, |, \rangle\}$.
- Wegen $MPCP_{\Gamma} \leq PCP$ folgt hieraus $A \leq PCP$.

Beweisidee für die Reduktion $A \leq MPCP_{\Gamma}$:

Transformiere eine Eingabe $w \in \Sigma^*$ in eine Instanz $f(w) = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$, so dass $\alpha = (i_1, \dots, i_n)$ genau dann eine MPCP-Lösung für $f(w)$ ist, wenn das zugehörige **Lösungswort** $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$ eine Ableitung $S = \alpha_0 \Rightarrow \dots \Rightarrow \alpha_m = w$ von w kodiert.

Beweis von $A \leq \text{MPCP}_\Gamma$

- Wir bilden $f(w)$ aus folgenden Wortpaaren:

- $(\langle, \langle | S)$,
- für jede Regel $l \rightarrow r$ in P : (l, r) ,
- für alle $a \in V \cup \Sigma \cup \{|\}$: (a, a) ,
- sowie das Paar $(w | \rangle, \rangle)$

„Startpaar“

„Ableitungspaare“

„Kopierpaare“

„Abschlusspaar“

- Nun lässt sich leicht aus einer Ableitung $S = \alpha_0 \Rightarrow \dots \Rightarrow \alpha_m = w$ von w in G eine MPCP-Lösung mit dem Lösungswort

$$\langle |\alpha_0 | \alpha_1 | \dots | \alpha_m | \rangle$$

angeben.

- Umgekehrt lässt sich aus jeder MPCP-Lösung auch eine Ableitung von w in G gewinnen, womit

$$w \in L(M) \Leftrightarrow f(w) \in \text{MPCP}_\Gamma$$

gezeigt ist.

Das Postsche Korrespondenzproblem

Beispiel

- Sei $G = (\{S\}, \{a, b\}, \{S \rightarrow aSbS, \varepsilon\}, S)$ und $w = aabb$.
- Die MPCP-Instanz $f(aabb)$ enthält dann die acht Wortpaare

$$f(aabb) = \left(\begin{array}{c} \langle \quad S \quad S \quad S \quad a \quad b \quad | \quad aabb| \rangle \\ \langle |S \quad aSbS \quad \varepsilon \quad S \quad a \quad b \quad | \quad \rangle \end{array} \right).$$

- Der Ableitung $\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSb\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$ entspricht dann das MPCP-Lösungswort

$$\begin{array}{l} \langle |S|aSbS|aaSbSbS|aaSbbsS|aabbS|aabb| \rangle \\ \langle |S|aSbS|aaSbSbS|aaSbbsS|aabbS|aabb| \rangle \end{array}$$

- Das kürzeste MPCP-Lösungswort für $f(aabb)$ ist

$$\begin{array}{l} \langle |S|aSbS|aaSbSb|aabb| \rangle \\ \langle |S|aSbS|aaSbSb|aabb| \rangle \end{array}$$

- Dieses entspricht der „parallelisierten“ Ableitung

$$\underline{S} \Rightarrow a\underline{S}b\underline{S} \Rightarrow^2 aa\underline{S}b\underline{S}b \Rightarrow^2 aabb$$

Das Schnittproblem für CFL ist unentscheidbar

Das Schnittproblem für kontextfreie Grammatiken

Gegeben: Zwei kontextfreie Grammatiken G_1 und G_2 .

Gefragt: Ist $L(G_1) \cap L(G_2) \neq \emptyset$?

Satz

Das Schnittproblem für kontextfreie Grammatiken ist RE-vollständig.

Beweis

- Es ist leicht zu sehen, dass das Problem semi-entscheidbar ist.
- Um PCP auf dieses Problem zu reduzieren, betrachte für eine Folge $s = (x_1, \dots, x_k)$ von Strings $x_i \in \{a, b\}^*$ die Sprache

$$L_s = \{i_n \dots i_1 \# x_{i_1} \dots x_{i_n} \mid n \geq 1, 1 \leq i_1, \dots, i_n \leq k\}.$$

- L_s wird von der Grammatik $G_s = (\{A\}, \{1, \dots, k, \#, a, b\}, P_s, A)$ erzeugt mit

$$P_s: A \rightarrow 1Ax_1, \dots, kAx_k, 1\#x_1, \dots, k\#x_k$$

Reduktion von PCP auf das Schnittproblem für CFL

- Zu einer PCP-Instanz $I = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$ bilden wir das Paar (G_s, G_t) , wobei $s = (x_1, \dots, x_k)$ und $t = (y_1, \dots, y_k)$ ist.
- Dann ist $L(G_s) \cap L(G_t)$ die Sprache

$$\{i_n \dots i_1 \# x_{i_1} \dots x_{i_n} \mid 1 \leq n, x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}\}.$$

- Folglich ist $\alpha = (i_1, \dots, i_n)$ genau dann eine Lösung für I , wenn $i_n \dots i_1 \# x_{i_1} \dots x_{i_n} \in L(G_s) \cap L(G_t)$ ist.
- Also vermittelt $f : I \mapsto (G_s, G_t)$ eine Reduktion von PCP auf das Schnittproblem für CFL.



Beispiel

- Die PCP-Instanz

$$I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & aab & abbaa \\ aabba & ababb & aa \end{pmatrix}$$

wird auf das Grammatikpaar (G_s, G_t) mit folgenden Regeln reduziert:

$$P_s: A \rightarrow 1Aa, 2Aaab, 3Aabbaa,$$

$$1\#a, 2\#aab, 3\#abbaa,$$

$$P_t: A \rightarrow 1Aaabba, 2Aababb, 3Aaa,$$

$$1\#aabba, 2\#ababb, 3\#aa.$$

- Der PCP-Lösung $\alpha = (1, 3, 2, 3)$ entspricht dann das Wort

$$3231\#x_1x_3x_2x_3 = 3231\#aabbaaaababbaa$$

$$= 3231\#aabbaaaababbaa = 3231\#y_1y_3y_2y_3$$

im Schnitt $L(G_s) \cap L(G_t)$.

Das Inklusionsproblem für DPDAs

Gegeben: Zwei DPDAs M_1 und M_2 .

Gefragt: Gilt $L(M_1) \subseteq L(M_2)$?

Korollar

- 1 Das Schnittproblem für DPDAs ist RE-vollständig.
- 2 Das Inklusionsproblem für DPDAs ist co-RE-vollständig.

Beweisidee.

Die kontextfreie Grammatik G_s in obigem Beweis lässt sich leicht in DPDAs M_s und \overline{M}_s überführen mit $L(M_s) = L(G_s)$ und $L(\overline{M}_s) = \overline{L(G_s)}$ (siehe Übungen).

Korollar

Für kontextfreie Grammatiken sind folgende Probleme unentscheidbar:

- 1 Ist $L(G) = \Sigma^*$? (Ausschöpfungsproblem)
- 2 Ist $L(G_1) = L(G_2)$? (Äquivalenzproblem)
- 3 Ist G mehrdeutig? (Mehrdeutigkeitsproblem)

1 Das Ausschöpfungsproblem für CFL ist co-RE-vollständig

Wir reduzieren das Komplement von PCP auf das Ausschöpfungsproblem für kontextfreie Grammatiken. Es gilt

$$I \notin \text{PCP} \Leftrightarrow L_s \cap L_t = \emptyset \Leftrightarrow \bar{L}_s \cup \bar{L}_t = \Sigma^*.$$

Daher vermittelt die Funktion $f : I \mapsto G$, wobei G eine kontextfreie Grammatik mit

$$L(G) = \bar{L}_s \cup \bar{L}_t$$

ist, die gewünschte Reduktion. □

Korollar

Für kontextfreie Grammatiken sind folgende Probleme unentscheidbar:

- 1 Ist $L(G) = \Sigma^*$? (Ausschöpfungsproblem)
- 2 Ist $L(G_1) = L(G_2)$? (Äquivalenzproblem)
- 3 Ist G mehrdeutig? (Mehrdeutigkeitsproblem)

2 Das Äquivalenzproblem für CFL ist co-RE-vollständig

Wir reduzieren das Ausschöpfungsproblem für CFL auf das Äquivalenzproblem für CFL.

Dies leistet beispielsweise die Reduktionsfunktion

$$f : G \mapsto (G, G_{all}),$$

wobei G_{all} eine kontextfreie Grammatik mit $L(G_{all}) = \Sigma^*$ ist. □

③ Das Mehrdeutigkeitsproblem ist RE-vollständig

- Wir reduzieren PCP auf dieses Problem.
- Hierzu überführen wir eine PCP-Instanz $I = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$ in die Grammatik

$$G = (\{S, A, B\}, \{a, b, 1, \dots, k\}, P \cup \{S \rightarrow A, S \rightarrow B\}, S)$$

mit den Regeln

$$P: A \rightarrow 1Ax_1, \dots, kAx_k, 1\#x_1, \dots, k\#x_k,$$

$$B \rightarrow 1By_1, \dots, kBy_k, 1\#y_1, \dots, k\#y_k.$$

- Da alle von A oder B ausgehenden Ableitungen eindeutig sind, ist G genau dann mehrdeutig, wenn es ein Wort $w \in L(G)$ gibt mit

$$S \Rightarrow A \Rightarrow^* w \quad \text{und} \quad S \Rightarrow B \Rightarrow^* w.$$

- Wie wir gesehen haben, ist dies genau dann der Fall, wenn I eine PCP-Lösung hat.

Ein Unentscheidbarkeitsresultat für DCSL

Das Leerheitsproblem für DLBAs

Gegeben: Ein DLBA M .

Gefragt: Ist $L(M) = \emptyset$?

Satz

Das Leerheitsproblem für DLBAs ist co-RE-vollständig.

Beweis.

- Wir reduzieren $\overline{\text{PCP}}$ auf das Leerheitsproblem für DLBAs.
- Hierzu transformieren wir eine PCP-Instanz $I = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$ in einen DLBA M für die Sprache $L_s \cap L_t$, wobei $s = (x_1, \dots, x_k)$ und $t = (y_1, \dots, y_k)$ ist (siehe Übungen).
- Dann gilt $I \notin \text{PCP} \Leftrightarrow L(M) = \emptyset$. □

Entscheidbare Probleme

Dagegen ist es nicht schwer,

- für eine kontextfreie Grammatik G zu entscheiden, ob mindestens ein Wort in G ableitbar ist (Leerheitsproblem für CFL), und
- für eine kontextsensitive Grammatik G und ein Wort x zu entscheiden, ob x in G ableitbar ist (Wortproblem für CSL).

Satz

- Das Leerheitsproblem für CFL ist entscheidbar.
- Das Wortproblem für CSL ist entscheidbar.

Beweis.

Siehe Übungen.

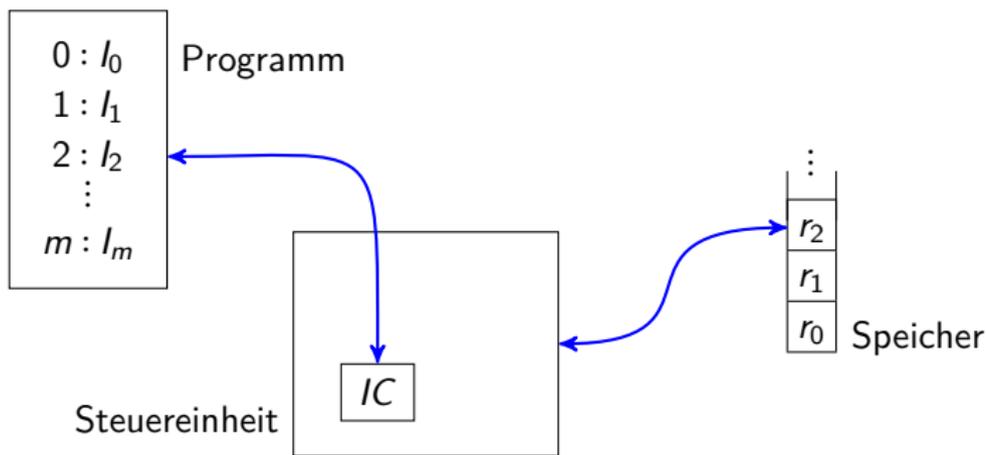


In folgender Tabelle fassen wir nochmals zusammen, wie schwierig die betrachteten Entscheidungsprobleme für die verschiedenen Stufen der Chomsky-Hierarchie sind.

	Wort- problem $x \in L?$	Leerheits- problem $L = \emptyset?$	Aus- schöpfung $L = \Sigma^*?$	Äquivalenz- problem $L_1 = L_2?$	Inklusions- problem $L_1 \subseteq L_2$	Schnitt- problem $L_1 \cap L_2 \neq \emptyset?$
REG	ja	ja	ja	ja	ja	ja
DCFL	ja	ja	ja	ja ^a	nein	nein
CFL	ja	ja	nein	nein	nein	nein
DCSL	ja	nein	nein	nein	nein	nein
CSL	ja	nein	nein	nein	nein	nein
RE	nein	nein	nein	nein	nein	nein

^aBewiesen in 1997 von Géraud Sénizergues (Univ. Bordeaux).

Die Registermaschine (random access machine, RAM) 324



- führt ein Programm $P = l_0, \dots, l_m$ aus, das aus einer endlichen Folge von Befehlen (**instructions**) l_i besteht,
- hat einen Befehlszähler (**instruction counter**) IC , der die Nummer des nächsten Befehls angibt (zu Beginn ist $IC = 0$),
- verfügt über einen frei adressierbaren Speicher (**random access memory**) mit unendlich vielen Speicherzellen (Registern) r_i , die beliebig große natürliche Zahlen aufnehmen können.

Eine Programmiersprache für RAMs

In **GOTO-Programmen** sind folgende Befehle zulässig ($i, j, c \in \mathbb{N}$):

Befehl	Semantik
$r_i := r_j + c$	setzt Register r_i auf den Wert $r_j + c$
$r_i := r_j \dot{-} c$	setzt Register r_i auf den Wert $\max(0, r_j - c)$
GOTO j	setzt den Befehlszähler IC auf den Wert j
IF $r_i = c$ THEN GOTO j	setzt IC auf j , falls r_i den Wert c hat
HALT	beendet die Programmausführung

Falls nichts anderes angegeben ist, wird zudem IC auf den Wert $IC + 1$ gesetzt.

GOTO-Berechenbarkeit

Definition

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$ heißt **GOTO-berechenbar**, falls es ein GOTO-Programm $P = (I_0, \dots, I_m)$ mit folgender Eigenschaft gibt:

Wird P auf einer RAM mit den Werten $r_i = n_i$ für $i = 1, \dots, k$, sowie $IC = 0$ und $r_i = 0$ für $i = 0, k + 1, k + 2, \dots$ gestartet, so gilt:

- P hält genau dann, wenn $(n_1, \dots, n_k) \in \text{dom}(f)$ ist, und
- falls P hält, hat r_0 nach Beendigung von P den Wert $f(n_1, \dots, n_k)$.

Beispiel

Folgendes GOTO-Programm berechnet die Funktion $f(x, y) = xy$:

0	IF $r_1 = 0$ THEN GOTO 4	5	$r_3 := r_2$
1	$r_1 := r_1 \div 1$	6	IF $r_3 = 0$ THEN GOTO 3
2	$r_0 := r_0 + r_2$ GOTO 5	7	$r_3 := r_3 \div 1$
3	GOTO 0	8	$r_0 := r_0 + 1$
4	HALT	9	GOTO 6



DTMs und GOTO-Programme sind gleichmächtig

- Da DTMs auf Wörtern und GOTO-Programme auf Zahlen operieren, müssen wir Wörter durch Zahlen (und umgekehrt) kodieren.
- Sei $\Sigma = \{a_0, \dots, a_{m-1}\}$ ein Alphabet. Dann können wir jedes Wort $x = a_{i_1} \dots a_{i_n} \in \Sigma^*$ durch eine natürliche Zahl $num_{\Sigma}(w)$ kodieren:

$$num_{\Sigma}(x) = \sum_{j=0}^{n-1} m^j + \sum_{j=1}^n i_j m^{n-j} = \frac{m^n - 1}{m - 1} + (i_1 \dots i_n)_m.$$

- Da die Abbildung $num_{\Sigma} : \Sigma^* \rightarrow \mathbb{N}$ bijektiv ist, können wir umgekehrt jede natürliche Zahl n durch das Wort $str_{\Sigma}(n) = num_{\Sigma}^{-1}(n)$ kodieren.

Beispiel

Für das Alphabet $\Sigma = \{a, b, c\}$ erhalten wir folgende Kodierung:

w	ε	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc	aaa	\dots
$num_{\Sigma}(w)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	\dots

- Ist $\Sigma = \{0, 1\}$, so lassen wir den Index weg und schreiben einfach num und str anstelle von num_{Σ} und str_{Σ} .
- Zudem erweitern wir die Kodierungsfunktion $str : \mathbb{N} \rightarrow \{0, 1\}^*$ zu einer Kodierungsfunktion $str_k : \mathbb{N}^k \rightarrow \{0, 1, \#\}^*$ wie folgt:

$$str_k(n_1, \dots, n_k) = str(n_1)\# \dots \#str(n_k).$$

- Nun können wir eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$ durch folgende partielle Wortfunktion $\hat{f} : \{0, 1, \#\}^* \rightarrow \{0, 1\}^* \cup \{\uparrow\}$ repräsentieren:

$$\hat{f}(w) = \begin{cases} str(n), & w = str_k(n_1, \dots, n_k) \text{ und } f(n_1, \dots, n_k) = n \in \mathbb{N}, \\ \uparrow, & \text{sonst.} \end{cases}$$

- Es ist klar, dass f durch \hat{f} eindeutig bestimmt ist. Wir nennen f die **numerische Repräsentation** von \hat{f} .

Satz

Sei f die numerische Repräsentation einer partiellen Funktion \hat{f} .
Dann ist f genau dann GOTO-berechenbar, wenn \hat{f} berechenbar ist.

- Sei \hat{f} eine partielle Funktion, deren numerische Repräsentation f von einem GOTO-Programm P auf einer RAM R berechnet wird.
- Dann existiert eine Zahl k' , so dass P nur Register r_i mit $i \leq k'$ benutzt.
- Daher lässt sich eine Konfiguration von R durch Angabe der Inhalte des Befehlszählers IC und der Register $r_0, \dots, r_{k'}$ beschreiben.
- Wir konstruieren eine $(k' + 2)$ -DTM M , die
 - den Inhalt von IC in ihrem Zustand,
 - die Registerwerte $r_1, \dots, r_{k'}$ auf den Bändern $1, \dots, k'$ und
 - den Wert von r_0 auf dem Ausgabeband $k' + 2$ speichert.
- Ein Registerwert r_i wird hierbei in der Form $str(r_i)$ gespeichert.
- Band $k' + 1$ wird zur Ausführung von Hilfsberechnungen benutzt.

- Die Aufgabe von M ist es, bei Eingabe $w = str_k(n_1, \dots, n_k)$ das Wort $str(f(n_1, \dots, n_k))$ auszugeben, wenn $(n_1, \dots, n_k) \in dom(f)$ ist, und andernfalls nicht zu halten.
- Zuerst kopiert M die Teilwörter $str(n_i)$ für $i = 2, \dots, k$ auf das i -te Band und löscht auf dem 1. Band alle Eingabezeichen bis auf $str(n_1)$.
- Da das leere Wort den Wert $num(\varepsilon) = 0$ kodiert, sind nun auf den Bändern $1, \dots, k'$ und auf Band $k' + 2$ die der Startkonfiguration von R bei Eingabe (n_1, \dots, n_k) entsprechenden Registerinhalte gespeichert.
- Danach führt M das Programm P Befehl für Befehl aus.
- Es ist leicht zu sehen, dass sich jeder Befehl l in P durch eine Folge von Anweisungen realisieren lässt, die die auf den Bändern gespeicherten Registerinhalte bzw. den im Zustand von M gespeicherten Wert von IC entsprechend modifizieren.
- Sobald P stoppt, hält auch M und gibt das Wort $str(r_0) = str(f(n_1, \dots, n_k))$ aus.

- Sei $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$ eine partielle Funktion und sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine DTM mit Eingabealphabet $\Sigma = \{0, 1, \#\}$, die die zugehörige Wortfunktion $\hat{f} : \Sigma^* \rightarrow \{0, 1\}^* \cup \{\uparrow\}$ berechnet.
- M gibt also bei Eingabe $str_k(n_1, \dots, n_k)$ das Wort $str(f(n_1, \dots, n_k))$ aus, falls $f(n_1, \dots, n_k)$ definiert ist, und hält andernfalls nicht.
- Wir konstruieren ein GOTO-Programm P , das bei Eingabe (n_1, \dots, n_k) die DTM M bei Eingabe $w = str_k(n_1, \dots, n_k)$ simuliert und im Fall, dass $M(w)$ hält, den Wert $num(M(w)) = f(n_1, \dots, n_k)$ berechnet.
- Wir können annehmen, dass M eine 1-DTM ist.
- Sei $Z = \{q_0, \dots, q_r\}$ und $\Gamma = \{a_0, \dots, a_{m-1}\}$, wobei wir annehmen, dass $a_0 = \sqcup$, $a_1 = 0$, $a_2 = 1$ und $a_3 = \#$ ist.

- Eine Konfiguration $K = uq_i v$ von M wird wie folgt in den Registern r_0, r_1, r_2 gespeichert. Sei $u = a_{i_1} \dots a_{i_n}$ und $v = a_{j_1} \dots a_{j_{n'}}$.
 - $r_0 = (i_1 \dots i_n)_m$,
 - $r_1 = i$,
 - $r_2 = (j_{n'} \dots j_1)_m$.
- P besteht aus 3 Programmteilen $P = P_1, P_2, P_3$.
- P_1 berechnet in Register r_2 die Zahl $(j_n \dots j_1)_m$, wobei (j_1, \dots, j_n) die Indexfolge der Zeichen von $str_k(n_1, \dots, n_k) = a_{j_1} \dots a_{j_n}$ ist.
- Die übrigen Register setzt P_1 auf den Wert 0.
- P_1 stellt also die Startkonfiguration $K_w = q_0 w$ von M bei Eingabe $w = str_k(n_1, \dots, n_k)$ in den Registern r_0, r_1, r_2 her.
- Anschließend führt P_2 eine schrittweise Simulation von M aus.
- Hierzu überführt P_2 solange die in r_0, r_1, r_2 gespeicherte Konfiguration von M in die zugehörige Nachfolgekonfiguration, bis M hält.

Simulation einer DTM durch ein GOTO-Programm

- Das Programmstück P_2 hat die Form

$$M_2 \quad r_3 := r_2 \text{ MOD } m$$

$$\text{IF } r_1 = 0 \wedge r_3 = 0 \text{ THEN GOTO } M_{0,0}$$

$$\vdots$$

$$\text{IF } r_1 = r \wedge r_3 = m - 1 \text{ THEN GOTO } M_{r,m-1}$$

- Die Befehle ab Position $M_{i,j}$ hängen von $\delta(q_i, a_j)$ ab. Wir betrachten exemplarisch den Fall $\delta(q_i, a_j) = \{(q_{i'}, a_{j'}, L)\}$:

$$M_{i,j} \quad r_1 := i'$$

$$r_2 := r_2 \text{ DIV } m$$

$$r_2 := r_2 m + j'$$

$$r_2 := r_2 m + (r_0 \text{ MOD } m)$$

$$r_0 := r_0 \text{ DIV } m$$

$$\text{GOTO } M_2$$

- Im Fall $\delta(q_i, a_j) = \emptyset$ erfolgt ein Sprung an den Beginn von P_3 .
- P_3 transformiert den Inhalt $r_0 = (j_1 \dots j_n)_m$ von Register r_0 in die Zahl $\text{num}(a_{j_1} \dots a_{j_n})$ und hält.

WHILE- und LOOP-Programme

- Die Syntax von **WHILE-Programmen** ist induktiv wie folgt definiert ($i, j, c \in \mathbb{N}$):
 - Jede Wertzuweisung der Form $x_i := x_j + c$ oder $x_i := x_j \div c$ ist ein WHILE-Programm.
 - Falls P und Q WHILE-Programme sind, so auch
 - $P; Q$ und
 - **IF $x_i = c$ THEN P ELSE Q END**
 - **WHILE $x_i \neq c$ DO P END**
- Die Syntax von **LOOP-Programmen** ist genauso definiert, nur dass Schleifen der Form **LOOP x_i DO P END** an die Stelle von WHILE-Schleifen treten.
- Die Semantik von WHILE-Programmen ist selbsterklärend.
- Eine LOOP-Schleife **LOOP x_i DO P END** wird sooft ausgeführt, wie der Wert von x_i zu Beginn der Schleife angibt.

WHILE- und LOOP-Berechenbarkeit

- Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$ heißt **WHILE-berechenbar**, falls es ein WHILE-Programm P mit folgender Eigenschaft gibt:
Wird P mit den Werten $x_i = n_i$ für $i = 1, \dots, k$ gestartet, so gilt:
 - P hält genau dann, wenn $(n_1, \dots, n_k) \in \text{dom}(f)$ ist, und
 - falls P hält, hat x_0 nach Beendigung von P den Wert $f(n_1, \dots, n_k)$.
- Die **LOOP-Berechenbarkeit** von f ist entsprechend definiert.

Beispiel

Die Funktion $f(x_1, x_2) = x_1 x_2$ wird von dem WHILE-Programm

```
WHILE  $x_1 \neq 0$  DO
```

```
   $x_0 := x_0 + x_2$ ;
```

```
   $x_1 := x_1 \div 1$ 
```

```
END
```

sowie von folgendem LOOP-Programm berechnet:

```
LOOP  $x_1$  DO  $x_0 := x_0 + x_2$  END
```

Satz

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$ ist genau dann GOTO-berechenbar, wenn sie WHILE-berechenbar ist.

- Sei P ein WHILE-Programm, das f berechnet.
- Wir übersetzen P wie folgt in ein äquivalentes GOTO-Programm P' .
- P' speichert den Variablenwert x_i im Register r_i .
- Damit lassen sich alle Wertzuweisungen von P direkt in entsprechende Befehle von P' transformieren.
- Eine Schleife der Form **WHILE** $x_i \neq c$ **DO** Q **END** simulieren wir durch folgendes GOTO-Programmstück:

```
 $M_1$  IF  $r_i = c$  THEN GOTO  $M_2$   
     $Q'$   
    GOTO  $M_1$ 
```

```
 $M_2$   $\vdots$ 
```

- Ähnlich lässt sich die Verzweigung **IF** $x_i = c$ **THEN** Q_1 **ELSE** Q_2 **END** in ein GOTO-Programmstück transformieren.
- Zudem fügen wir ans Ende von P' den HALT-Befehl an.

- Sei $P = (l_0, \dots, l_m)$ ein GOTO-Programm, das f berechnet, und sei r_z , $z > k$, ein Register, das in P nicht benutzt wird.
- Wir übersetzen P wie folgt in ein äquivalentes WHILE-Programm P' :


```

 $x_z := 0;$ 
WHILE  $x_z \neq m + 1$  DO
  IF  $x_z = 0$  THEN  $P'_0$  END;
   $\vdots$ 
  IF  $x_z = m$  THEN  $P'_m$  END
END

```
- Dabei ist P'_j abhängig vom Befehl l_j folgendes WHILE-Programm:
 - $r_i := r_k + c$: $x_i := x_k + c$; $x_z := x_z + 1$,
 - $r_i := r_k \dot{-} c$: $x_i := x_k \dot{-} c$; $x_z := x_z + 1$,
 - **GOTO** k : $x_z := k$,
 - **IF** $r_i = c$ **THEN GOTO** k :


```

IF  $x_i = c$  THEN  $x_z := k$  ELSE  $x_z := x_z + 1$  END

```
 - **HALT**: $x_z := m + 1$.
- Man beachte, dass P' nur eine WHILE-Schleife enthält.

- Es ist leicht zu sehen, dass sich jedes LOOP-Programm durch ein WHILE-Programm simulieren lässt.
- Offensichtlich können LOOP-Programme nur totale Funktionen berechnen.
- Daher kann nicht jedes WHILE-Programm durch ein LOOP-Programm simuliert werden.
- Mittels Diagonalisierung lässt sich eine totale WHILE-berechenbare Funktion f angeben, die nicht LOOP-berechenbar ist.
- Ein bekanntes Beispiel einer totalen WHILE-berechenbaren Funktion, die nicht LOOP-berechenbar ist, ist die **Ackermannfunktion** $a(x, y)$, die induktiv wie folgt definiert ist:

$$a(x, y) = \begin{cases} y + 1, & x = 0, \\ a(x - 1, 1), & x \geq 1, y = 0, \\ a(x - 1, a(x, y - 1)), & x, y \geq 1. \end{cases}$$