Universelle Rechner und Church'sche These

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

Homepage: http://www.ruhr-uni-bochum.de/lmi

Partiell definierte Funktionen

Wir betrachten im Folgenden partiell definierte Funktionen der Form

$$f : \mathbb{N}_0^k \to \mathbb{N}_0 \text{ bzw. } f : \Sigma^* \to \Sigma^* \ .$$

Für x außerhalb des Definitionsbereiches gilt dann

$$f(x) = \text{„undefiniert“} \ .$$

Intuition: Rechenprogramme mit Eingaben aus $\mathbb{N}_0^k$ bzw. Σ^* werden

entweder nach endlich vielen Schritten mit einem (durch eine Ausgabekonvention festgelegten) Ergebnis stoppen,

oder in eine unendliche Schleife geraten.

Die von einem Programm berechnete Funktion ist also i.A. nur partiell definiert.

Zentrale Frage

Welche Funktionen sind berechenbar ?

- Bei berechenbaren Funktionen genügt ein intuitiver Berechenbarkeitsbegriff.
- Aber zum Nachweis der Unberechenbarkeit benötigen wir eine exakte Definition.

Intuitive Berechenbarkeit

Informelle Definition: Eine Funktion

$$f : \mathbb{N}_0^k \to \mathbb{N}_0 \text{ bzw. } f : \Sigma^* \to \Sigma^*$$

heißt „(intuitiv) berechenbar“, wenn es eine „mechanisch anwendbare“ Rechenvorschrift gibt, die bei Eingabe x

- nach „endlich vielen Schritten“ zur Ausgabe $f(x)$ führt, falls $f(x)$ definiert ist,
- in eine „unendliche Schleife“ führt, falls $f(x)$ undefiniert ist.

Ausblick: Formale Definitionen der Berechenbarkeit

- durch Turing–Programme berechenbar
- durch WHILE–Programme berechenbar
- durch GOTO–Programme berechenbar

Alle diese Vorschläge (von Turing, Church und anderen Mathematikern Mitte der 1930er unterbreitet) haben sich als äquivalent erwiesen. Zudem wurde bislang keine intuitiv berechenbare Funktion gefunden, die nicht auch Turing–berechenbar wäre. Dies führte zur (formal nicht beweisbaren)

Church'schen These: Die Klasse der intuitiv berechenbaren Funktionen stimmt überein mit der Klasse der durch Turing–berechenbaren (bzw. WHILE–berechenbaren, GOTO–berechenbaren, ...) Funktionen.

Turing–Berechenbarkeit

Eine Funktion

$$f : \Sigma^* \to \Sigma^*$$

heißt Turing–berechenbar **gdw** eine DTM M existiert mit folgenden Eigenschaften:

- Falls $f(x) = y$, dann gilt $z_0 x \vdash^* zy$ für eine Endkonfiguration zy.
- Falls $f(x) =$ „undefiniert“, dann erreicht M bei der Rechnung auf Eingabe x keine Stoppkonfiguration (Endlosrechnung).

Turing–Berechenbarkeit (fortgesetzt)

Die Turing–Berechenbarkeit von Funktionen der Form

$$f : \mathbb{N}_0^k \to \mathbb{N}_0$$

ist analog definiert, wobei wir im Falle von

$$f(x) = y \text{ mit } x = (n_1, n_2, \ldots, n_k)$$

anstelle von $z_0 x \vdash^* zy$ nun

$$z_0 \text{bin}(n_1)\#\text{bin}(n_2)\#\cdots\#\text{bin}(n_k) \vdash^* z\text{bin}(y)$$

fordern. Hierbei bezeichnet bin$(\cdot)$ die Binärdarstellung ohne führende Nullen.

Beispiele

Die Nachfolgerfunktion

$$s(n) = n + 1$$

ist Turing–berechenbar:

Siehe unsere frühere Implementierung eines Binärzählers.

Beispiele (fortgesetzt)

Die total undefinierte Funktion

$$\forall n \in \mathbb{N}_0 : \ \Omega(n) = \text{„undefiniert“}$$

ist Turing–berechenbar durch die DTM mit

$$\forall A \in \Gamma : \delta(z_0, A) = (z_0, A, R) \ ,$$

die auf jeder Eingabe eine unendliche Schleife durchläuft.

Beispiele (fortgesetzt)

Zu einer Sprache L vom Typ 0 betrachte die Funktion

$$\chi'_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ \text{„undefiniert“} & \text{falls } w \notin L \end{cases}$$

Die Turing–Berechenbarkeit von χ'_L kann folgendermaßen eingesehen werden:

- Es gibt eine Grammatik G vom Typ 0, welche L generiert.
- Es gibt eine NTM M, welche L erkennt, indem Ableitungen $S \Rightarrow_G^* w$ geraten werden. M kann so implementiert werden, dass
  - nach Auffinden einer Ableitung Ausgabe 1 produziert und gestoppt wird,
  - bei Nicht–Auffinden einer Ableitung eine unendliche Schleife betreten wird.
- Dann wird χ'_L berechnet durch die deterministische Simulation M' von M.

Mehrspurenmaschinen

Zu einem gegebenen Alphabet Γ können wir „Supersymbole“ aus Γ^k betrachten. Wenn das Arbeitsalphabet einer TM ein Supersymbol $(A_1, \ldots, A_k)$ enthält, dann ist es anschaulich sich vorzustellen, dass

- das Band in k „Spuren“ zerlegt werden kann,
- und beim Abspeichern von $(A_1, \ldots, A_k)$ in einer Zelle, das Symbol A_i in der i–ten Spur der Zelle steht.

Beachte: Mehrspurenmaschinen haben zwar ein unkonventionelles Arbeitsalphabet (welches k-Tupel enthält), entsprechen aber unserer Standarddefinition einer TM (**kein** neues Modell).

Mehrbandmaschinen

Definition: Unter einer k-Band TM verstehen wir eine TM mit k Bändern und einem Kopf pro Band. Die insgesamt k Köpfe können sich in einem Rechenschritt in verschiedene Richtungen bewegen. Die Überführungsfunktion δ hat nun die Form

$$\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{R, L, N\}^k$$

mit der offensichtlichen Interpretation.

Mehrbandmaschinen sind nicht mächtiger als das Standardmodell wie der folgende sogenannte Bandreduktionssatz zeigt:

Satz: Eine k-Band TM M kann von einer 1-Band TM M' simuliert werden. Ist dabei M eine DTM, so auch M'.

Beweis

- M' besitzt für jeden Zustand z von M einen entsprechenden Zustand z' (und weitere Zustände).
- M' simuliert
  - einen Schritt von M mit Zustandswechsel von z_1 nach z_2
  - durch eine Folge von Schritten, welche im Zustand z'_1 startet und im Zustand z'_2 endet

Nach diesem Schema verlaufende Simulationen heißen „Schritt für Schritt Simulation“.

Beweis (fortgesetzt)

Die wesentliche Schwierigkeit besteht darin, die Beschriftung der k-Band TM M auf einem einzigen Band unterzubringen. M' benutzt dazu ein Band mit k Spuren. Dabei soll stets gelten:

(1) Spur i des Bandes von M' enthält die Beschriftung von Band i von M $(1 \leq i \leq k)$.

(2) Zelle 1 von M' enthält genau die k Symbole, auf denen die k Köpfe von M positioniert sind.

(3) Zu Beginn der Simulation des nächsten Rechenschrittes von M befindet sich der Kopf von M' auf Zelle 1.

Bedingungen (2) und (3) sorgen dafür, daß M' die von M gelesenen k Symbole kennt.

Beweis (fortgesetzt)

Um einen Schritt von M zu simulieren, geht M' vor wie folgt:

- Wenn M Symbole $a_1, \ldots, a_k$ durch $b_1, \ldots, b_k$ ersetzt, ersetzt M' in Zelle 1 $(a_1, \ldots, a_k)$ durch $(b_1, \ldots, b_k)$.
- Wenn M Kopf i nach rechts (bzw. links) bewegt, so verschiebt M' die Inschrift von Spur i um eine Position in die entgegengesetzte Richtung (positioniert aber im Anschluss den Kopf wieder auf Zelle 1).
- Wenn M in Zustand z übergeht, geht M' in Zustand z' über.

Hierdurch bleiben Bedingungen (1), (2) und (3) erhalten und die Simulation ist korrekt.

Offensichtlich arbeitet M' deterministisch, falls M deterministisch arbeitet.

Zusätzliche Beobachtung

Wenn M auf Eingaben der Länge n

- maximal $S(n)$ Zellen ihrer Bänder besucht
- und maximal $T(n)$ Schritte rechnet,

dann

- besucht M' ebenfalls maximal $S(n)$ Zellen
- und rechnet maximal $O(S(n) \cdot T(n))$ Schritte (da jeder Schritt von M in $O(S(n))$ Schritten von M' simuliert werden kann).

Ein „Baukastensystem“ für Turing–Maschinen

Ziel: Entwurf von DTMs zur Ausführung von Befehlen einer „höheren Programmiersprache“ (mit bedingten Anweisungen, while-Schleifen etc.)

Methode: Baukastensystem

Veränderung des Inhaltes von einem der Bänder

- Zu einer 1–Band–TM M bezeichne $M(i,k)$, oder einfach $M(i)$, die k–Band TM, die das „Programm“ von M auf ihrem i–ten Band simuliert (und auf den anderen Bändern keine Modifikationen vornimmt).
- „Band := Band +1“ bezeichne die früher bereits besprochene 1–Band DTM zur Berechnung der Funktion $s(n) = n + 1$.
- Statt „Band := Band+1“(i) schreiben wir „Band i:= Band i +1“.
- Definiere die „modifizierte Differenz“ wie folgt:

$$n \dot{-} m = \max\{0, n - m\} \ .$$

Die Notationen

„Band i := Band i $\dot{-}$1“

„Band i := Band j

„Band i := 0

sind dann analog zu verstehen.

Komposition von TMs

Die Komposition zweier TMs

$$M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0i}, \Box, E_i), i = 1, 2, \ Z_1 \cap Z_2 = \emptyset$$

ist definiert als die TM

$$M = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{01}, \Box, E_2) \ ,$$

wobei

$$\delta(z, A) = \begin{cases} \delta_1(z, A) & \text{falls } z \in Z_1 \setminus E_1 \\ (z_{02}, A, N) & \text{falls } z \in E_1 \\ \delta_2(z, A) & \text{falls } z \in Z_2 \end{cases} .$$

M führt also zuerst das Programm von M_1 aus und (falls M_1 einen Endzustand erreicht) dann das Programm von M_2.

Notation als „Flussdiagramm“: start $\rightarrow M_1 \rightarrow M_2 \rightarrow$ stop.

Notation wie bei Programmiersprachen: $M_1; M_2$

Beispiel

Die DTM

$$\begin{aligned} \text{start} &\rightarrow \text{„Band} := \text{Band} + 1\text{“} \\ &\rightarrow \text{„Band} := \text{Band} + 1\text{“} \\ &\rightarrow \text{„Band} := \text{Band} + 1\text{“} \rightarrow \text{stop} \end{aligned}$$

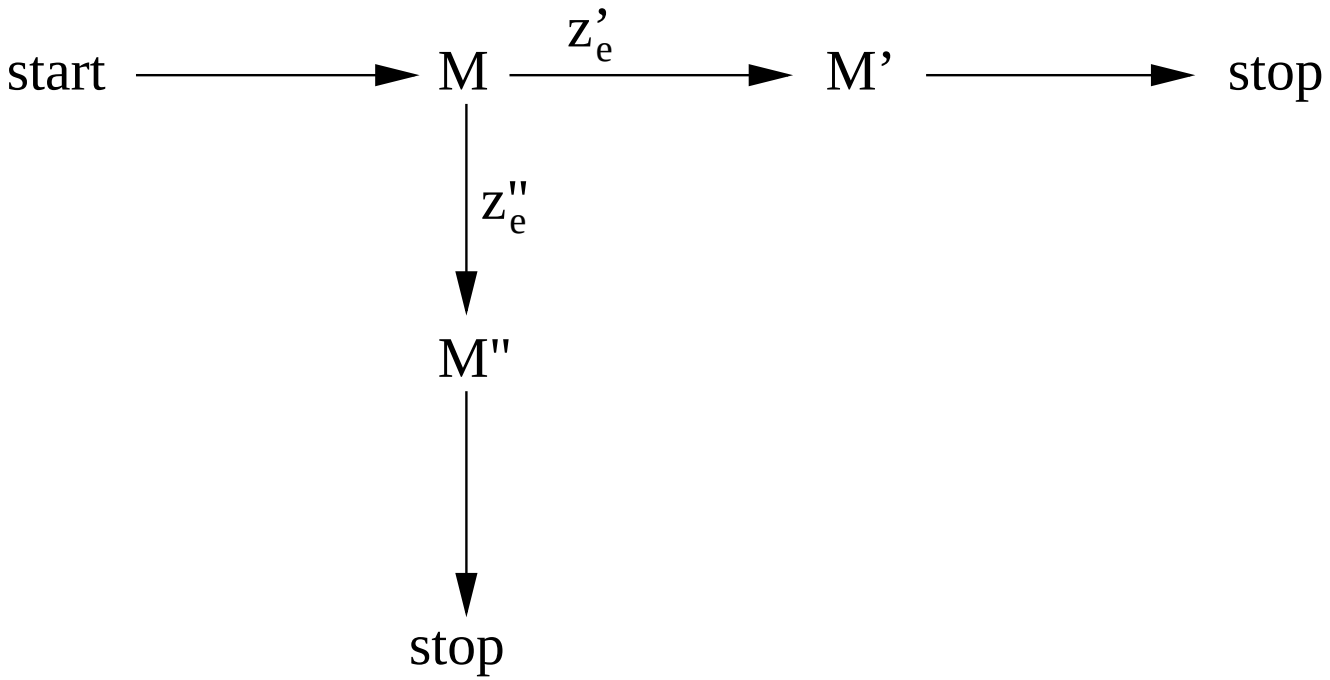
addiert zu einer gegebenen natürlichen Zahl die Konstante 3 hinzu.

Bedingte Komposition von TMs

Eine TM, welche

- zunächst das Programm einer TM M ausführt,
- hernach das Programm von M', falls M im Endzustand z'_e stoppt,
- bzw. das Programm M'', falls M im Endzustand z''_e stoppt,

notieren wir in der Form



Die Abfrage–Maschine

„Band=0?“ bezeichnet eine DTM mit folgenden Eigenschaften:

- Sie hat vier Zustände $z_0, z_1, \mathrm{JA}, \mathrm{NEIN}$ mit $\mathrm{JA}, \mathrm{NEIN}$ als Endzuständen.
- Sie verändert den Bandinhalt nicht. Zu Beginn und am Ende der Rechnung ist der Kopf auf dem ersten Zeichen der Eingabe positioniert.
- Ihre Hauptaufgabe ist zu testen, ob die Eingabe nur aus dem Zeichen 0 besteht. Falls dem so ist stoppt sie im Endzustand JA; andernfalls stoppt sie im Endzustand NEIN.

Eine solche DTM ist einfach zu entwerfen:

$$\delta(z_0, a) = \begin{cases} (z_1, a, R) & \text{falls } a = 0 \\ (\mathrm{NEIN}, a, N) & \text{sonst} \end{cases}$$

$$\delta(z_1, a) = \begin{cases} (\mathrm{JA}, a, L) & \text{falls } a = \square \\ (\mathrm{NEIN}, a, L) & \text{sonst} \end{cases}$$

Einbettung einer TM in eine WHILE–Schleife

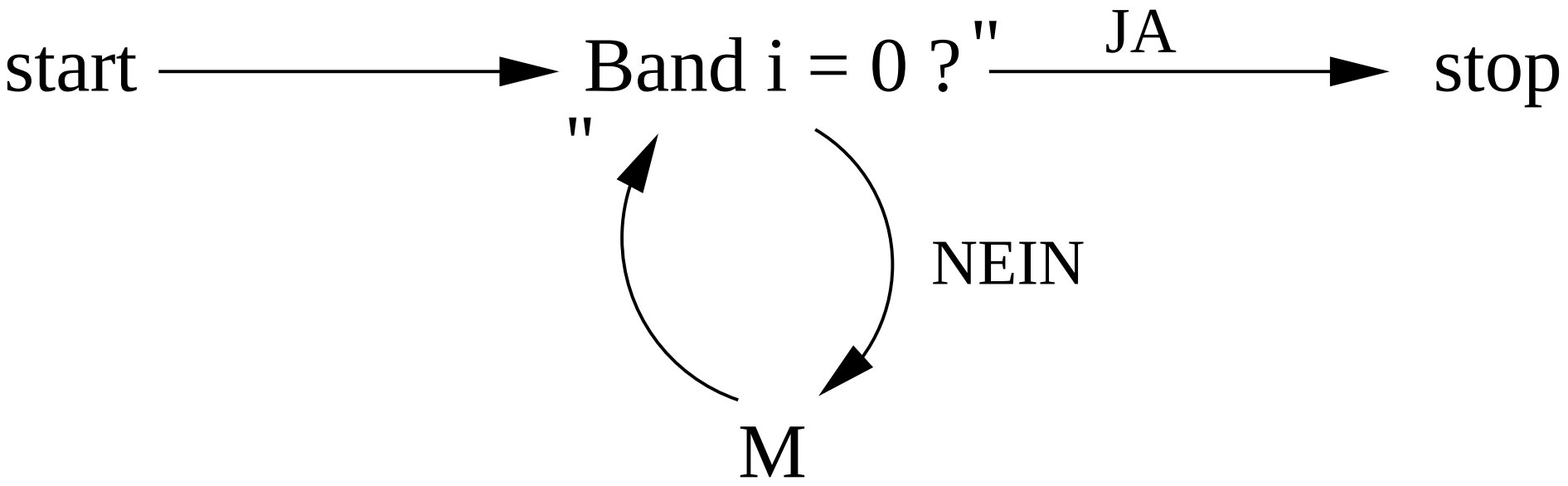
Statt „Band=0?“(i) schreiben wir einfach

„Band i = 0 ?“

Zu einer gegebenen TM M bezeichne

„WHILE Band i $\neq$ 0 DO M“

die durch folgendes Flussdiagramm gegebene TM:



Résumée

- Mit dem Baukastensystem lassen sich aus elementaren TMs komplexere TMs zusammensetzen, die Strukturen höherer Programmiersprachen wie zum Beipiel
  - bedingte Anweisungen
  - WHILE–Schleifen
  - Prozedurkonzept

  (ansatzweise) realisieren.
- Die Realisierung macht Gebrauch von Mehrband–TMs. Wie wir wissen lässt sich aber jede Mehrband–TM durch eine Einband–TM simulieren.

Zeichenvorrat für LOOP-Programme

Variablen:	x_0 x_1 x_2 $\cdots$
Konstanten:	0 1 2 $\cdots$
Trennsymbole:	; :=
Operationszeichen:	+ −
Schlüsselwörter:	LOOP DO END

Syntax von LOOP-Programmen

Induktive Definition:

1. Jede Wertzuweisung der Form

   $$x_i := x_j + c \text{ oder } x_i := x_j - c$$

   (für eine Konstante c) ist ein LOOP-Programm.

2. Die Hintereinanderschaltung

   $$P_1 \ ; \ P_2$$

   von LOOP-Programmen P_1, P_2 ist ein LOOP-Programm.

3. Das iterierte Durchlaufen

   $$\text{LOOP } x_i \text{ DO } P \text{ END}$$

   eines LOOP-Programmes P ist ein LOOP-Programm.

Semantik von LOOP-Programmen

Kanonisch definiert bis auf:

- „$a - b$“ wird interpretiert als „modifizierte Differenz“ $a \dot{-} b := \max\{a - b, 0\}$.
- Bei einem LOOP-Programm der Form LOOP x_i DO P END wird P so oft ausgeführt wie der Wert der Variablen x_i zu *Beginn* angibt. (Änderung des Wertes von x_i im Innern von P haben auf die Anzahl der Wiederholungen also keinen Einfluss.)
  Folgerung: LOOP-Programme terminieren stets.

Konventionen beim Berechnen von $f : \mathbb{N}_0^k \to \mathbb{N}_0$ durch ein LOOP-Programm:

- Eingabewerte $n_1, \ldots, n_k$ anfangs in $x_1, \ldots, x_k$.
  Restliche Variable initialisiert auf 0.
- Ausgabewert $f(n_1, \ldots, n_k)$ am Ende in x_0.

LOOP-Programme berechnen nur *totale* (= total definierte) Funktionen.

LOOP-Simulierbare Konstrukte

neues Konstrukt	**Simulation**
$x_i := x_j$ $x_i := c$	$x_i := x_j + 0$ $x_i := y + c$ (für ein y mit Wert 0)
IF $x = 0$ THEN A END	$y := 1$; LOOP x DO $y := 0$ END; LOOP y DO A END
$x_i := x_j + x_k$	$x_i := x_j$; LOOP x_k DO $x_i := x_i + 1$ END
$x_i := x_j * x_k$	$x_i := 0$; LOOP x_k DO $x_i := x_i + x_j$ END
$x_i := x_j$ DIV x_k $x_i := x_j$ MOD x_k	s. Übung (evtl.) s. Übung (evtl.)

WHILE-Programme

Syntax wie bei LOOP-Programmen, außer dass die WHILE-Schleife an die Stelle der LOOP-Schleife tritt:

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

Semantik der WHILE-Schleife: P wird iteriert solange ausgeführt wie x_i (mit ihrem aktuellen Wert!) ungleich Null ist. Endlosschleife ist möglich.

Konventionen zum Berechnen von Funktionen wie bei LOOP-Programmen.
Berechnung partieller (= partiell definierter) Funktionen ist möglich.

WHILE-simulierbare LOOP-Schleife:

$$\text{LOOP } x \text{ DO } P \text{ END}$$

kann simuliert werden durch

$$y := x;\ \text{WHILE } y \neq 0 \text{ DO } y := y - 1; P \text{ END}\ .$$

Wechselseitige Simulationen

Da die LOOP-Schleife durch die WHILE-Schleife simulierbar ist, gilt der

Satz: Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

Weiter gilt:

Satz: (Beweis mündlich in der Vorlesung)
Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Wir werden (nach Einführung der GOTO-Programme) noch zeigen:
– Jede Turing-berechenbare Funktion ist auch GOTO-berechenbar.
– Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

Folgerung: Turing-Maschinen, WHILE-Programme und GOTO-Programme berechnen dieselbe Klasse von Funktionen.

GOTO-Programme

Syntax: GOTO-Programme haben (bis auf Fehlen von redundanten Marken) die Form

$$M_1 : A_1 \ ; \ M_2 : A_2 \ ; \ \cdots \ ; \ M_\ell : A_\ell \ .$$

Dabei ist A_i eine „Anweisung“ und M_i eine sogenannte „Marke“ (eindeutige Adresse für die Anweisung A_i). Als Anweisungen sind zugelassen:

Wertzuweisungen:	$x_i := x_j \pm c$
unbedingter Sprung:	GOTO M_i
bedingter Sprung:	IF $x_i = c$ THEN GOTO M_j
Stoppanweisung:	HALT

Semantik: — offensichtlich (oder?) —

Konventionen beim Berechnen von Funktionen:
analog zu LOOP- oder WHILE-Programmen.

Simulation von GOTO durch WHILE

Satz: Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

$$M_1 : A_1 \ ; \ M_2 : A_2 \ ; \ \cdots \ ; \ M_\ell : A_\ell$$

kann simuliert werden durch

$y := 1;$
WHILE $y \neq 0$ DO
  IF $y = 1$ THEN A'_1 END;
  IF $y = 2$ THEN A'_2 END;
  $\cdots$
  IF $y = \ell$ THEN A'_ℓ END
END

Idee: Identifiziere M_i mit Nummer i.
Wert von y = Nummer der aktuellen Marke (bzw. 0 nach Erreichen von HALT).
A'_i realisiert A_i und aktualisiert y.

Simulation von GOTO durch WHILE (fortgesetzt)

A_i	A'_i
$x_k := x_l \pm c$	$x_k := x_l \pm c$; y:=y+1
GOTO M_j	$y := j$
IF $x_k = c$ THEN GOTO M_j	IF $x_k = c$ THEN $y := j$ ELSE $y := y + 1$ END
HALT	$y := 0$

Es kann leicht gezeigt werden, dass die zur Simulation des bedingten Sprunges benötigte if-then-else Anweisung LOOP-simulierbar ist.

Beobachtung: Die Simulation benötigt nur **eine** WHILE-Schleife (sowie if-then-else Anweisungen oder, alternativ, LOOP-Anweisungen) .

Simulation von WHILE durch GOTO (fortgesetzt)

Satz: Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

$$\text{WHILE } x_i \neq 0 \; DO \; P \; \text{END}; \ldots$$

kann simuliert werden durch:

M_1: IF $x_i = 0$ THEN GOTO M_2;
P;
GOTO M_1;
M_2: ...

Folgerung (Kleene-Normalform für WHILE-Programme):
Jede WHILE-berechenbare Funktion kann durch ein WHILE-Programm mit lediglich einer WHILE-Schleife berechnet werden (wobei allerdings if-then-else Anweisungen oder, alternativ, LOOP-Anweisungen zum Einsatz kommen müssen).

Exkurs: DIV und MOD

DIV (ganzzahliger Quotient) und MOD (kleinster Rest) sind die folgenden Operationen:

$$x \text{ DIV } y = \left\lfloor \frac{x}{y} \right\rfloor .$$

$$x \text{ MOD } y = x - y \left\lfloor \frac{x}{y} \right\rfloor$$

Zum Beispiel:

$$75 \text{ DIV } 20 = \left\lfloor \frac{75}{20} \right\rfloor = \lfloor 3.75 \rfloor = 3 .$$

$$75 \text{ MOD } 20 = 75 - 20 \left\lfloor \frac{75}{20} \right\rfloor = 75 - 20 \cdot 3 = 15 .$$

DIV und MOD (fortgesetzt)

CUT und PASTE (Abschneiden und Ankleben) von Ziffern kann mit Hilfe von DIV, MOD und $*, +$ implementiert werden:

CUT und PASTE	Ergebnis	Simulation mit DIV,MOD,+, *
CUT(1984)	$198\|4$	$198 = 1984$ DIV $10; 4 = 1984$ MOD 10
PASTE(198|5)	1985	$1985 = 198 * 10 + 5$

Verallgemeinerung auf b-näre Zahlendarstellungen (Ziffern aus $\{0, 1, \ldots, b-1\}$):

CUT und PASTE	Ergebnis	Simulation mit DIV,MOD,+, *
$\mathrm{CUT}(\overbrace{i_1 \cdots i_{p-1} i_p}^{x})$	$\overbrace{i_1 \cdots i_{p-1}}^{x'} \| i_p$	$x' = x$ DIV $b; i_p = x$ MOD b
$\mathrm{PASTE}(\underbrace{i_1 \cdots i_{p-1}}_{x'} \| j)$	$\underbrace{i_1 \cdots i_{p-1} j}_{\hat{x}}$	$\hat{x} = x' * b + j$

Exkurs: Konfiguration als Zahlentripel

- Zustandsmenge $Z = \{z_1, \ldots, z_s\}$: z_l hat „Nummer“ l.
- Bandalphabet $\Gamma = \{a_1, \ldots, a_m\}$: a_i hat „Nummer“ i

Setze $b := |\Gamma| + 1$. Eine Konfiguration

$$\overbrace{a_{i_1} \cdots a_{i_p}}^{x} z_l \overbrace{a_{j_1} \cdots a_{j_q}}^{y}$$

(mit z_l als aktuellem Zustand, $a_{i_1} \cdots a_{i_p}$ als Bandinschrift links vom Kopf und $a_{j_1} \cdots a_{j_q}$ als Bandinschrift ab Kopfposition) kann als b-näres Zahlentripel (x, y, z) kodiert werden (wobei das leere Wort als Zahl 0 dargestellt würde):

$$z = l \ , \ x = \sum_{\rho=1}^{p} i_\rho b^{p-\rho} \ , \ y = \sum_{\rho=1}^{q} j_\rho b^{\rho-1}$$

(Nummern der Symbole sind gleichsam die Ziffern der Zahlendarstellung.)

Simulation von Turing-Maschine durch GOTO

Satz Jede Turing-berechenbare Funktion ist auch GOTO-berechenbar.

DTM M berechne $f : \mathbb{N}_0^k \to \mathbb{N}_0$.

Aufbau der Simulation:

Phase 1 (Vorbereitung): Berechne aus den Werten $n_1, \ldots, n_k$ der Eingabevariablen $x_1, \ldots, x_k$ das Zahlentripel (x, y, z), welches die Startkonfiguration $z_0\text{bin}(n_1)\# \cdots \#\text{bin}(n_k)$ von M repräsentiert.

Phase 2 (Schritt-für-Schritt Simulation): Solange M nicht stoppt, berechne aus dem Zahlentripel (x, y, z) der aktuellen Konfiguration das Zahlentripel für die direkte Folgekonfiguration.

Phase 3 (Nachbereitung) Extrahiere aus dem Zahlentripel (x, y, z), das eine Endkonfiguration $z_e\text{bin}(f(n_1, \ldots, n_k))$ von M repräsentiert, den Ausgabewert $f(n_1, \ldots, n_k)$ und belege damit die Ausgabevariable x_0.

Details zur Phase 2

„(l,j)-Aktualisierung“ von (x,y,z) bezeichne die Aktualisierung, die erforderlich ist, wenn M im Zustand z_l Symbol a_j liest (und die durch $\delta(z_l,a_j)$ beschriebene Aktion ausführt).

Das GOTO-„Unterprogramm“ (plus zugehöriger Marke), das die (l,j)-Aktualisierung durchführt (und i.A. aus mehreren Anweisungen besteht) bezeichnen wir mit

$$M_{l,j} : A_{l,j} \quad .$$

Wir präsentieren im Folgenden:

- Das Grundgerüst eines Teilprogrammes P_2, das die Verzweigung zum richtigen Unterprogramm gewährleistet,
- ein Beispiel-Unterprogramm.

Verzweigung zum richtigen Unterprogramm

Programmstück $M_2 : P_2$ für Phase 2 hat folgende Form:

M_2: $a := y$ MOD b; (CUT-Operation liefert Symbol unterm Lesekopf)
IF $z = 1$ AND $a = 1$ THEN GOTO $M_{1,1}$;
IF $z = 1$ AND $a = 2$ THEN GOTO $M_{1,2}$;
usw.
— alle sm Zustands/Symbolkombinationen —
usw.
IF $z = s$ AND $a = m$ THEN GOTO $M_{s,m}$;

$M_{1,1}$: $A_{1,1}$; GOTO M_2;

$M_{1,2}$: $A_{1,2}$; GOTO M_2;
usw.
— alle sm Zustands/Symbolkombinationen —
usw.

$M_{s,m}$: $A_{s,m}$; GOTO M_2;

Ein Beispiel-Unterprogramm

Programmzeile

$$\delta(z_l, a_j) = (z_{l'}, a_{j'}, L)$$

würde durch folgendes Unterprogramm realisiert:

$M_{l,j}$: $z := l'$; (Aktualisiere z mit Nummer des neuen Zustands)
$y := y$ DIV b (CUT)
$y := b * y + j'$ (PASTE)
Kommentar: führt $y = \langle j\ j_2\ \cdots\ j_q\rangle_b$ in $y = \langle j'\ j_2\ \cdots\ j_q\rangle_b$ über
$y := b * y + (x$ MOD $b)$ (PASTE)
$x := x$ DIV b (CUT)
Kommentar: führt $x = \langle i_1\ \cdots\ i_{p-1}\ i_p\rangle_b$, $y = \langle j'\ j_2\ \cdots\ j_q\rangle_b$
in $x = \langle i_1\ \cdots\ i_{p-1}\rangle_b$, $y = \langle i_p\ j'\ j_2\ \cdots\ j_q\rangle$ über

Falls $z_{l'}$ ein Endzustand wäre, dann würde dieses Unterprogramm noch um die Anweisung „GOTO M_3“ (Eintritt in Phase 3) erweitert.

Entscheidbare und unentscheidbare Probleme

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

Homepage: http://www.ruhr-uni-bochum.de/lmi

Das Grundvokabular der Theorie entscheidbarer und unentscheidbarer Sprachen

Entscheidbarkeit und Semi–Entscheidbarkeit

Eine Sprache $L \subseteq \Sigma^*$ heißt entscheidbar **gdw** die charakteristische Funktion

$$\chi_L(w) := \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{falls } w \notin L \end{cases}$$

von L berechenbar ist.

Eine Sprache $L \subseteq \Sigma^*$ heißt semi–entscheidbar **gdw** die „halbe“ charakteristische Funktion

$$\chi'_L(w) := \begin{cases} 1 & \text{falls } w \in L \\ \text{„undefiniert“} & \text{falls } w \notin L \end{cases}$$

von L berechenbar ist.

Erinnerung: Sprache und Haltebereich einer DTM

Wir erinnern an die Definition der Sprache $T(M)$ und des Haltebereiches $H(M)$ einer DTM M:

$$\begin{aligned} T(M) &= \{w \in \Sigma^* | \ \exists z_e \in E, \alpha, \beta \in \Gamma^* : z_0 w \vdash^* \alpha z_e \beta\} \\ H(M) &= \{w \in \Sigma^* | \\ &\quad \exists z \in Z, A \in \Gamma, \alpha, \beta \in \Gamma^* : z_0 w \vdash^* \alpha z A \beta, \delta(z, A) = \text{„undefiniert“}\} \end{aligned}$$

Hierbei setzen wir folgendes voraus:

- $\delta(z_e, A) =$„undefiniert“ für alle $z_e \in E$.
- $\delta(z, A)$ mit $z \in Z \setminus E$ darf undefiniert sein (muss aber nicht).

Somit gilt

$$T(M) \subseteq H(M) \subseteq \Sigma^* \ .$$

Entscheidbarkeit (fortgesetzt)

Ein nicht-akzeptierender Stoppzustand ist ein Zustand $z \in Z \setminus E$ mit $\delta(z, A) =$„undefiniert“ für alle $A \in \Gamma$.

Satz: L ist entscheidbar **gdw** es eine DTM M gibt mit

$$T(M) = L \text{ und } H(M) = \Sigma^* \ .$$

Beweis: Es wird ausgenutzt, dass Produzieren der Ausgabe 1 ersetzt werden kann durch Übergang in einen (akzeptierenden) Endzustand — und umgekehrt. Produzieren der Ausgabe 0 kann ersetzt werden durch Übergang in einen (nicht-akzeptierenden) Stoppzustand außerhalb von E — und umgekehrt.

Semi–Entscheidbarkeit (fortgesetzt)

Satz: L ist semi–entscheidbar **gdw** es eine DTM M gibt mit $T(M) = L$.

Beweis: Es wird wieder ausgenutzt, dass Produzieren der Ausgabe „1“ ersetzt werden kann durch Übergang in einen (akzeptierenden) Endzustand — und umgekehrt.

Folgerung: Jede entscheidbare Sprache ist (erst recht) semi-entscheidbar.

Sprachen und Entscheidungsprobleme

(Binäre) Entscheidungsprobleme sind Probleme, welche nur die Antworten JA oder NEIN zulassen.

Sprachen und Entscheidungsprobleme sind zwei Seiten der gleichen Münze:

- Ein Entscheidungsproblem kann auch aufgefasst werden als die Sprache aller Eingabeinstanzen, welche zur Antwort JA führen.
- Eine Sprache kann auch als das Problem aufgefasst werden zu entscheiden, ob eine Eingabeinstanz zur Sprache gehört (Wortproblem).

Wir werden daher im Folgenden „Sprache“ und „Problem“ zuweilen synonym verwenden.

Erinnerung: Abzählbarkeit

Erinnerung: Eine nichtleere Menge A ist abzählbar **gdw** wenn wir ihre Elemente durchnummerieren können, d.h., wenn die Elemente sich bijektiv (1–zu–1) auf $\mathbb{N}_0$ oder (falls A endlich ist) auf eine endliche Teilmenge von $\mathbb{N}_0$ abbilden lassen.

Äquivalent hierzu können wir fordern, dass eine Abbildung $f : \mathbb{N}_0 \to A$ existiert, so dass

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

Beachte, dass $f(i) = f(j)$ für $i \neq j$ zulässig ist (sonst würden endliche Mengen A ausgeschlossen).

- Jede Teilmenge einer abzählbaren Menge ist ebenfalls abzählbar.
- Da die Wortmenge Σ^* über einem endlichen Alphabet Σ abzählbar ist, ist jede formale Sprache $L \subseteq \Sigma^*$ eine abzählbare Menge.

Aufzählbarkeit

intuitiv: „Aufzählbarkeit“ = „algorithmisch durchführbare Abzählbarkeit“.

Formale Definition: Eine Sprache L heißt (rekursiv) aufzählbar **gdw** $L = \emptyset$ oder es gibt eine total berechenbare (= total definierte und berechenbare) Abbildung f mit $L = \{f(0), f(1), f(2), \ldots\}$.

Wir wenden diese Definition sinngemäß auch auf Mengen an, deren Elemente „auf natürliche Weise“ als Strings kodiert werden können (so dass die Menge als formale Sprache auffassbar ist).

Eine DTM zur Berechnung von f nennen wir im Folgenden eine „Abzählmaschine“ für L.

- Σ^* ist aufzählbar.
- Es gibt nicht aufzählbare formale Sprachen (Beispiele hierfür später).
- Die Teilmenge einer aufzählbaren Menge ist nicht notwendig aufzählbar.
- $\mathbb{N}_0 \times \mathbb{N}_0$ ist aufzählbar

Eine Abzählmaschine für $\{0,1\}^*$

Betrachte die Abbildung $f_* : \mathbb{N}_0 \to \{0,1\}^*$, die n abbildet auf den n–ten Binärstring der unendlichen Liste

$$\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots \quad . \tag{1}$$

Also

$$f_*(0) = \varepsilon, f_*(1) = 0, f_*(2) = 1, f_*(3) = 00, f_*(4) = 01, f_*(5) = 10, f_*(6) = 11, \ldots$$

Die Berechnung von $f_*(n)$ ist (ohne auf Effizienz zu achten) offensichtlich:

- Zähle auf Band A einen Binärzähler hoch.
- Zähle parallel dazu auf Band B einen Zähler hoch, der die Binärstrings gemäß (1) durchläuft.
- Wenn der Zähler von Band A auf n steht, dann enthält Band B den Binärstring $f^*(n)$.

f_* ist bijektiv und ihre Umkehrabbildung ist nach dem gleichen Schema berechenbar.

Eine Abzählmaschine für $\mathbb{N}_0 \times \mathbb{N}_0$

Betrachte die bijektive Abbildung $d : \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0$, die n abbildet auf das n-te Zahlenpaar der unendlichen Liste

$$\underbrace{(0,0)}_{x+y=0}, \underbrace{(0,1),(1,0)}_{x+y=1}, \underbrace{(2,0),(1,1),(0,2)}_{x+y=2}, \underbrace{(3,0),(2,1),(1,2),(0,3)}_{x+y=3}, \ldots \quad . \qquad (2)$$

Also

$$d(0) = (0,0), d(1) = (1,0), d(2) = (0,1), d(3) = (2,0), d(4) = (1,1), \ldots \quad .$$

Die Umkehrabbildung $c : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ ordnet dann einem Zahlenpaar $(x,y) \in \mathbb{N}_0 \times \mathbb{N}_0$ eine eindeutige Nummer $c(x,y) \in \mathbb{N}_0$ zu.

Mit Hilfe eines Zählers zum Auflisten der Paare gemäß (2) und eines parallel dazu inkrementierten Binärzählers sind d und c (ohne auf Efizienz zu achten) offensichtlich berechenbar.

Veranschaulichung der Nummerierung von Zahlenpaaren

Die Nummern $c(x,y)$ ergeben sich auch aus folgender Tabelle:

	y=0	y=1	y=2	y=3	y=4	$\cdots$
x=0	0	1	3	6	10	$\cdots$
x=1	2	4	7	11	16	$\cdots$
x=2	5	8	12	17	23	$\cdots$
x=3	9	13	18	24	31	$\cdots$
x=4	14	19	25	32	40	$\cdots$

$\cdots$

Es gilt (Denksportaufgabe!):

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

Äquivalenz von Aufzählbarkeit und Semi–Entscheidbarkeit

Satz Eine Sprache L ist aufzählbar **gdw** L semi–entscheidbar ist.

Wir haben zwei Beweisrichtungen:

1. Transformation einer Abzählmaschine für L in eine DTM zur Berechnung von χ'_L.
2. Transformation einer DTM zur Berechnung von χ'_L in eine Abzählmaschine für L (vorausgesetzt $L \neq \emptyset$).

1. Beweisrichtung

Gegeben: eine Abzählmaschine M für L, die eine Funktion f mit $f(\mathbb{N}_0) = L$ berechnet

Gesucht: eine DTM M' zur Berechnung von $\chi'(w)$

Methode: Für $i = 0, 1, 2, \ldots$ mache folgendes:

1. Berechne $f(i)$ mit Hilfe von M.
2. Falls $f(i) = w$, dann gib 1 aus und stoppe.

- Ein Eingabewort $w \in L$ taucht für mindestens einen Index i als $w = f(i)$ in der Abzählung auf und führt zur Ausgabe 1.
- Für $w \notin L$ gerät M' in eine Endlosschleife.

2. Beweisrichtung

Gegeben: eine DTM M' zur Berechnung von χ'_L

Gesucht: eine Abzählmaschine M für L, die eine Funktion $f(n)$ mit $f(\mathbb{N}_0) = L$ berechnet

Erinnerung: total berechenbare bijektive Abbildungen $f_* : \mathbb{N}_0 \to \Sigma^*$, $d : \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0$

Methode: 1. Zu Eingabe n berechne $d(n) = (n_1, n_2) \in \mathbb{N}_0 \times \mathbb{N}_0$.

2. Simuliere n_2 Schritte von M' angesetzt auf Eingabe $f_*(n_1)$.
3. Falls M' in dieser Zeit Ausgabe „1“ produziert, dann gib $f(n) := f_*(n_1)$ aus; andernfalls gib einen „Default–String“ $w \in L$ aus.

Korrektheit: Offensichtlich gilt $f(n) \in L$ für alle $n \in \mathbb{N}_0$, d.h., es werden wirklich nur Wörter aus L aufgezählt.
Zudem ist f surjektiv, d.h., jedes Wort $x \in L$ kommt in der Aufzählung vor. Wieso ? (Begründung in der Vorlesung)

Überblick zur Semi–Entscheidbarkeit

Folgende Aussagen zu einer Sprache $L \subseteq \Sigma^*$ sind äquivalent:

- L ist aufzählbar.
- L ist semi–entscheidbar.
- L hat eine DTM als Akzeptor.
- L hat eine NTM als Akzeptor.
- L ist vom Typ 0.

Abschluss–Eigenschaften

Satz 1: Die Klasse der entscheidbaren Sprachen ist abgeschlossen unter den Operationen „$\cup, \cap, \neg, \cdot, *$“.

Satz 2: Die Klasse der semi–entscheidbaren Sprachen ist abgeschlossen unter den Operationen „$\cup, \cap, \cdot, *$“, aber (wie wir später noch zeigen werden) nicht unter der Operation „$\neg$“.

- Für semi-entscheidbare Sprachen (= Typ 0 Sprachen) sind uns diese Abschlusseigenschaften von den Typ 0 Sprachen her bereits bekannt.
- Der Nachweis der Abschluss–Eigenschaften kann (relativ leicht) geführt werden, indem zwei gegebene DTMs für L_1 und L_2 benutzt werden, um DTMs für

$$L_1 \cup L_2, L_1 \cap L_2, \bar{L}_1, L_1 \cdot L_2, L_1^*$$

zusammenzubasteln (Syntheseprobleme).

Zweimal „halb“ macht „ganz“

Satz: Eine Sprache L ist entscheidbar **gdw** L und $\bar{L}$ semi–entscheidbar sind.

$\Rightarrow$:

Wenn L entscheidbar ist, dann ist auch $\bar{L}$ entscheidbar.

Da jede entscheidbare Sprache erst recht semi–entscheidbar ist, sind folgerichtig dann L und $\bar{L}$ semi–entscheidbar.

$\Leftarrow$:

Wenn L und $\bar{L}$ semi–entscheidbar sind, sagen wir $\chi'_{\bar{L}}$ und χ'_L werden durch DTMs M_0 und M_1 berechnet, dann ist χ_L nach folgendem Muster berechenbar:

Für $t = 0, 1, 2, \ldots$ mache folgendes:

1. Simuliere M_0 und M_1 jeweils auf Eingabe w für t Schritte.
2. Sowie eine der DTMs, sagen wir M_i, eine 1 ausgibt und stoppt, dann gib i aus und stoppe ebenfalls.

Eine binäre Kodierung von Turing–Maschinen

Arbeitsalphabet und Zustandsmenge können stets so gewählt werden, dass jedes Symbol und jeder Zustand eine Nummer erhält:

$$\begin{aligned} \Gamma &= \{A_0, \ldots, A_{r-1}\} \\ Z &= \{z_0, \ldots, z_{s-1}\} \end{aligned}$$

Dabei ist z_0 der Startzustand und oBdA gibt es nur einen Endzustand, und das ist z_{s-1}.

Ebenso können die Richtungsangaben nummeriert werden:

$$d_0 = L, d_1 = R, d_2 = N$$

Ein Eintrag $\delta(z_i, A_j) = (z_{i'}, A_{j'}, d_k)$ der Turing–Tafel kann dann durch den String

$$\#\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(k)$$

kodiert werden.

Die Turing–Tafel ist dann kodiert durch die Konkatenation der Codewörter ihrer Einträge (wobei diese, sagen wir, zeilenweise durchlaufen werden).

Die komplette TM ist schließlich kodiert durch die Konkatention von:

- Präambel bin(r)#bin(s)
- Kodierung der Turing-Tafel
- Endmarkierung ###, welche dafür sorgt, dass kein Codewort Präfix eines anderen Codewortes ist.

Schließlich erhalten wir ein binäres Codewort durch die Substitutionen

$$0 \mapsto 00, 1 \mapsto 01, \# \mapsto 11 \ .$$

Binäre Kodierung von Turing–Maschinen (fortgesetzt)

Es ist nicht schwer zu zeigen, dass

$$G := \{w \in \{0,1\}^* | \ w \text{ ist Codewort einer DTM}\}$$

entscheidbar ist.

- Falls $w \in G$, dann bezeichne M_w die von w kodierte DTM.
- Falls $w \notin G$, dann bezeichne M_w eine (beliebig aber fest ausgewählte) „Default–DTM“.

Damit haben wir erreicht, dass jedes Binärwort als DTM interpretierbar ist!

Universelle Turing–Maschine

Die universelle Sprache ist definiert wie folgt:

$$U := \{w\#x |\ x \in T(M_w)\}$$

Eine DTM heißt Universelle Turing–Maschine (UTM) **gdw** sie ein Akzeptor von U ist.

Eine UTM ist eine Art „General Purpose Computer“, der auf Eingaben der Form $w\#x$ vorgeht wie folgt:

- Simuliere M_w auf x.
- Akzeptiere $w\#x$ **gdw** M_w ihre Eingabe x akzeptiert.

Konstruktion einer UTM

Satz: Es gibt eine universelle Turing-Maschine (UTM).

Folgerung: U ist semi–entscheidbar.

Beweis des Satzes: Wir konstruieren eine UTM mit 3 Bändern, die wie folgt eingesetzt werden:

- Band 1 dient als „Programmspeicher“. Es enthält dauerhaft den String w, der als Code der DTM M_w interpretiert wird.
- Band 2 enthält die Binärkodierung $\mathrm{bin}(i)$ des aktuellen Zustands z_i von M_w.
- Band 3 dient als „Rechenspeicher“. Es enthält die aktuelle Bandinschrift der 1-Band DTM M_w in kodierter Form. Dabei wird das Symbol $A_j \in \Gamma$ kodiert als String $\mathrm{bin}(j)\$\ldots\$ \in \{0,1,\$\}^\ell$ mit $\ell = \lceil \log r \rceil$, wobei r die Größe des Bandalphabets Γ von M_w bezeichnet. Durch die „Auspolsterung“ mit $\$\ldots\$$ wird erreicht, dass alle Codewörter für Bandalphabetssymbole die selbe Länge ℓ haben.

Konstruktion einer UTM (fortgesetzt)

Die UTM vollzieht eine Schritt–für–Schritt Simulation von M_w. Ein Rechenschritt von M_w gemäß $\delta(z_i, A_j) = (z_{i'}, A_{j'}, R)$ (analog für die anderen Richtungswechsel des Kopfes) verändert die Konfiguration der Rechnung von M_w wie folgt:

$$\cdots z_i A_j A_k \cdots \vdash_{M_w} \cdots A_{j'} z_i A_k \cdots \quad .$$

Die UTM (mit bin(i) auf Band 2 und dem ersten Symbol von bin(j) unter dem Lesekopf von Band 3) simuliert diesen Schritt wie folgt:

1. Finde im „Programm“ w auf Band 1 den Teilstring für den Eintrag $\delta(z_i, A_j) = (z_{i'}, A_{j'}, R)$ der Turing-Tafel.
2. Ersetze auf Band 2 „bin(i)“ durch „bin(i')“.
3. Ersetze auf Band 3 „bin$(j)\$\ldots\$$“ durch „bin$(j')\$\ldots\$$“ und bewege den Kopf auf das erste Bit dahinter (also das erste Bit von „bin$(k)\$\ldots\$$“).

Auf diese Weise kann die Schritt–für–Schritt Simulation in Gang gehalten werden.

Erste Beispiele unentscheidbarer Sprachen

Eine nicht semi–entscheidbare Sprache

Die sogenannte Diagonalsprache ist definiert wie folgt:

$$D := \{w \in \{0,1\}^* | \ w \notin T(M_w)\}$$

In Worten: D besteht aus allen (Kodierungen von) DTMs, die ihre eigene Beschreibung (durch ein Codewort) nicht akzeptieren.

Satz: D ist nicht semi–entscheidbar.

Beweis durch Widerspruch: Wir machen die (heuchlerische) Annahme, es gäbe eine DTM M_0 mit $T(M_0) = D$. Betrachte das Codewort w_0 von M_0. Die folgenden Aussagen sind äquivalent:

(1) $w_0 \in T(M_0)$ (2) $w_0 \in D$. (3) $w_0 \notin T(M_0)$.

- Zur Äquivalenz von (1) und (2) nutze aus, dass $T(M_0) = D$.
- Zur Äquivalenz von (2) und (3) nutze die Definition von D aus.
- Die Äquivalenz von (1) und (3) ist ein WIDERSPRUCH.

Reduzierbarkeit

Definition: Betrachte zwei Sprachen $L_1, L_2 \subseteq \Sigma^*$.
L_1 heißt reduzierbar auf L_2 **gdw** eine total berechenbare Abbildung

$$f : \Sigma^* \to \Sigma^*$$

existiert mit der Eigenschaft

$$\forall w \in \Sigma^* : w \in L_1 \Leftrightarrow f(w) \in L_2 \ .$$

f nennen wir in diesem Zusammenhang eine Reduktionsabbildung.

Notation: $L_1 \leq L_2$.

Eigenschaften dieser Relation

Reflexivität: $L \leq L$.

Transitivität: Aus $L_1 \leq L_2$ und $L_2 \leq L_3$ folgt $L_1 \leq L_3$.

- Zum Nachweis der Reflexivität benutze die identische Reduktionsabbildung $f(w) = w$.
- Zum Nachweis der Transitivität setze die Reduktionsabbildungen f_1 und f_2 für die Reduktionen $L_1 \leq L_2$ und $L_2 \leq L_3$ zu einer Reduktionsabbildung $f(w) := f_2(f_1(w))$ für die Reduktion $L_1 \leq L_3$ zusammen:

$$w \in L_1 \Leftrightarrow f_1(w) \in L_2 \Leftrightarrow f_2(f_1(w)) \in L_3$$

Eigenschaften (fortgesetzt)

Voraussetzung: $L_1 \leq L_2$ (Reduktionsabbildung f).

Behauptungen: 1. Falls L_2 (semi–)entscheidbar ist, dann ist auch L_1 (semi–)entscheidbar.

2. Falls L_1 nicht (semi–)entscheidbar ist, dann ist auch L_2 nicht (semi–)entscheidbar.

Beweis: 1. Wegen $w \in L_1 \Leftrightarrow f(w) \in L_2$ gilt

$$\chi'_{L_1}(w) = \chi'_{L_2}(f(w)) \ .$$

Abbildung $\chi'_{L_1}(w)$ kann also berechnet werden, indem zunächst $f(w)$ und anschließend $\chi'_{L_2}(f(w))$ berechnet wird. Eine analoge Bemerkung gilt für die Funktion $\chi_{L_1}(w)$.

2. Die zweite Behauptung ist logisch äquivalent zur ersten.
(Umkehrschluss: $A \Rightarrow B$ ist logisch äquivalent zu $\neg B \Rightarrow \neg A$.)

Verbreitung „guter und schlechter Nachrichten“

Betrachte eine Reduktionskette

$$L_1 \leq L_2 \leq \cdots \leq L_{k-1} \leq L_k \ .$$

- Wenn L_k (semi–)entscheidbar ist, so auch $L_{k-1}, \ldots, L_2, L_1$.
- Wenn L_1 nicht (semi–)entscheidbar ist, so auch $L_2, \ldots, L_{k-1}, L_k$.

Salopp formuliert:

- „Gute Nachrichten“ verbreiten sich entlang von Reduktionsketten von rechts nach links.
- „Schlechte Nachrichten“ verbreiten sich entlang von Reduktionsketten von links nach rechts.

Wir werden diese Denkweise ausnutzen, um aus der Diagonalsprache D weitere nicht entscheidbare bzw. nicht semi–entscheidbare Sprachen abzuleiten.

Eine kleine Sammlung unentscheidbarer Sprachen

Neben dem Komplement der Diagonalsprache

$$\bar{D} = \{w \in \{0,1\}^* |\ w \in T(M_w)\}$$

und der universellen Sprache

$$U = \{w\#x |\ x \in T(M_w)\}$$

betrachten wir noch die folgenden Sprachen:

$$\begin{array}{rcll} H & := & \{w\#x |\ x \in H(M_w)\} & \text{(Halteproblem)} \\ H_0 & := & \{w |\ \varepsilon \in H(M_w)\} & \text{(Halteproblem auf leerem Band)} \\ K & := & \{w |\ w \in H(M_w)\} & \text{(spezielles Halteproblem)} \end{array}$$

Kleine Sammlung (fortgesetzt)

Bei diesen Sprachen geht es also um die folgenden Fragen:

$\bar{D}$: Akzeptiert eine DTM ihre eigene Beschreibung ?

U: Akzeptiert eine DTM ihre Eingabe ?

H: Stoppt eine auf ihre Eingabe angesetzte DTM nach endlich vielen Schritten ?

H_0: Stoppt eine auf das leere Band angesetzte DTM nach endlich vielen Schritten ?

K: Stoppt eine auf ihre eigene Beschreibung angesetzte DTM nach endlich vielen Schritten ?

Alle diese Fragen werden sich als unentscheidbar erweisen.

Kleine Sammlung (fortgesetzt)

Satz: $\bar{D}$ ist unentscheidbar.

Beweis: Wäre $\bar{D}$ entscheidbar, so wäre auch D entscheidbar.
D ist aber noch nicht einmal semi–entscheidbar.

Satz: $K = \{w|\ w \in H(M_w)\}$ ist semi–entscheidbar.

Beweis: $\chi'_K(w)$ kann berechnet werden wie folgt:

1. Gegeben Eingabe w, berechne $w\#w$.
2. Setze die UTM auf $w\#w$ an und gib „1“ aus, sowie die UTM in eine Stoppkonfiguration gerät.

Wegen

$$w \in K \Leftrightarrow w \in H(M_w) \Leftrightarrow w\#w \in H(UTM)\ .$$

führt diese Vorgehensweise zu einer korrekten Berechnung von $\chi'_K(w)$.

Kleine Sammlung (fortgesetzt)

Wir werden die folgende Reduktionskette nachweisen:

$$\bar{D} \leq U \leq H \leq H_0 \leq K$$

Wegen der Art, wie sich gute und schlechte Nachrichten verbreiten, erhalten wir die

Folgerung
$\bar{D}, U, H, H_0, K$ sind zwar semi-entscheidbar aber unentscheidbar.

Bleibt der Nachweis der obigen Reduktionskette.

Reduktion 1

Lemma: $\bar{D} \leq U$.

Verwende Reduktionsabbildung

$$w \mapsto w\#w \ .$$

Offensichtlich gilt:

$$w \in \bar{D} \Leftrightarrow w \in T(M_w) \Leftrightarrow w\#w \in U \ .$$

Reduktion 2

Lemma: $U \leq H$.

Verwende Reduktionsabbildung

$$w\#x \mapsto w'\#x \ .$$

Ziel: Transformiere M_w in eine DTM $M_{w'}$, so dass

$$x \in T(M_w) \Leftrightarrow x \in H(M_{w'}) \ .$$

Ändere dazu das „Programm“ w von M_w zu einem neuen „Programm“ w' einer DTM $M_{w'}$ ab:

- $M_{w'}$ simuliert M_w Schritt-für-Schritt,
- außer dass $M_{w'}$ sich in eine Endlosschleife begibt, falls M_w nicht–akzeptierend stoppt.

Reduktion 3

Lemma: $H \leq H_0$.

Verwende Reduktionsabbildung

$$w\#x \mapsto w' \ .$$

Ziel: Transformiere $w\#x$ in das „Programm“ w' einer DTM M', so dass

$$x \in H(M_w) \Leftrightarrow \varepsilon \in H(M_{w'}) \ .$$

Die gesuchte DTM $M_{w'}$ arbeitet wie folgt:

- Sie löscht ihre Eingabe (sofern diese nicht ohnehin leer ist) und schreibt dann x auf ihr Band.
- Sie simuliert anschließend Schritt-für-Schritt M_w auf Eingabe x.

Reduktion 4

Lemma: $H_0 \leq K$.

Verwende Reduktionsabbildung

$$w \mapsto w' \ .$$

Ziel: Transformiere M_w in eine DTM $M_{w'}$, so dass

$$\varepsilon \in H(M_w) \Leftrightarrow w' \in H(M_{w'}) \ .$$

Ändere dabei das „Programm“ w von M_w zu einem neuen Programm w' einer DTM $M_{w'}$ ab:

- $M_{w'}$ löscht zunächst ihre Eingabe (zum Beipiel auch die Eingabe w')
- und simuliert dann Schritt-für-Schritt M_w angesetzt auf das leere Band.

Das Post'sche Korrespondenzproblem

PKP und MPKP

Postsches Korrespondenzproblem (PKP)

Entscheide zu einer gegebenen Folge

$$K = [(x_1, y_1), \ldots, (x_k, y_k)]$$

von Wortpaaren über einem endlichen Alphabet Σ, ob es eine Folge

$$i_1, \ldots, i_n \in [1 : k]$$

von Indizes, genannt „Lösung“, gibt, so dass

$$x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n} \ .$$

Modifiziertes Postsches Korrespondenzproblem (MPKP)

Wie PKP, außer dass die Indexfolge mit $i_1 = 1$ beginnen muss.

Beispiel 1

Zu

$$K = [(1, 111), (10111, 10), (10, 0)]$$

ist $(2, 1, 1, 3)$ eine passende Indexfolge:

$$\overbrace{10111}^{x_2}\overbrace{1}^{x_1}\overbrace{1}^{x_1}\overbrace{10}^{x_3} = 101111110 = \overbrace{10}^{y_2}\overbrace{111}^{y_1}\overbrace{111}^{y_1}\overbrace{0}^{y_3}$$

Beispiel 2

Zu

$$K = [(10, 101), (011, 11), (101, 011)]$$

gibt es keine passende Indexfolge (Zugzwangargument):

1. Jede potenzielle Lösung müßte beginnen mit $i_1 = 1$:

$$x_1 = 10, \ y_1 = 101$$

2. Wann immer die y-Sequenz eine 1 Vorsprung hat, ist die einzig aussichtsreiche Fortsetzung

$$\begin{array}{lll} x\text{-Sequenz}: & \ldots & \overbrace{101}^{x_3} \\ y\text{-Sequenz}: & \ldots & 1\underbrace{011}_{y_3} \end{array},$$

was den Vorsprung von der y-Sequenz auf ewig reproduziert.

Beispiel 3

Zu

$$K = [(001, 0), (01, 011), (01, 101), (10, 001)]$$

gibt es eine passende Indexfolge $i_1, \ldots, i_n$, aber erst ab $n = 66$.

Wer findet die Lösung ?

Hauptresultat

Wir werden die Reduktionskette

$$H \leq MPKP \leq PKP$$

nachweisen.

Folgerung MPKP und PKP sind unentscheidbar.

Bemerkungen:

1. Die Unentscheidbarkeit ergibt sich bereits für binäres Alphabet (wie sich zeigen wird).
2. Bei unärem Alphabet hingegen sind MPKP und PKP entscheidbar. (Der Beweis hierfür wäre eine gute Übungsaufgabe.)

Reduktion von MPKP auf PKP

Die Eingabeinstanz von MPKP über Alphabet Σ sei

$$K = [(x_1, y_1), \ldots, (x_k, y_k)] \ .$$

Seien $\#, \$ \notin \Sigma$ zwei neue Symbole. Dann soll

$$f(K) = [(x'_0, y'_0), (x'_1, y'_1), \ldots, (x'_k, y'_k), (x'_{k+1}, y'_{k+1})]$$

die folgende Eingabe von PKP sein:

- Für $i = 1, \ldots, k$ entsteht x'_i, indem **hinter** jedem Buchstaben von x_i Symbol # eingefügt wird; y'_i entsteht aus y_i, indem **vor** jedem Buchstaben von y_i Symbol # eingefügt wird.
- $x'_0 = \#x'_1,\ x'_{k+1} = \$,\ y'_0 = y'_1,\ y'_{k+1} = \#\$.$

Es folgt ein Beispiel.

x_1	$=$	10111	x'_1	$=$	1#0#1#1#1#	x'_0	$=$	#1#0#1#1#1#
y_1	$=$	10	y'_1	$=$	#1#0	y'_0	$=$	#1#0
x_2	$=$	1	x'_2	$=$	1#			
y_2	$=$	111	y'_2	$=$	#1#1#1			
x_3	$=$	10	x'_3	$=$	1#0#	x'_4	$=$	\$
y_3	$=$	0	y'_3	$=$	#0	y'_4	$=$	#\$

$(1, 2, 2, 3)$ ist eine passende Indexfolge für $K = [(x_1, y_1), (x_2, y_2), (x_3, y_3)]$:

$$\overbrace{10111}^{x_1}\overbrace{1}^{x_2}\overbrace{1}^{x_2}\overbrace{10}^{x_3} = 101111110 = \overbrace{10}^{y_1}\overbrace{111}^{y_2}\overbrace{111}^{y_2}\overbrace{0}^{y_3} .$$

$(0, 2, 2, 3, 4)$ ist eine passende Indexfolge für $f(K)$, die folgenden „gepolsterten“ Lösungsstring liefert:

$$\overbrace{\#1\#0\#1\#1\#1\#}^{x'_0}\overbrace{1\#}^{x'_2}\overbrace{1\#}^{x'_2}\overbrace{1\#0\#}^{x'_3}\overbrace{\$}^{x'_4} = \overbrace{\#1\#0}^{y'_0}\overbrace{\#1\#1\#1}^{y'_2}\overbrace{\#1\#1\#1}^{y'_2}\overbrace{\#0}^{y'_3}\overbrace{\#\$}^{y'_4}$$

Reduktion von MPKP auf PKP (fortgesetzt)

Aus dem Design von $f(K)$ ergibt sich leicht:

1. Für alle n und alle $i_2, \ldots, i_n \in [1:k]$:

   $$1, i_2, \ldots, i_n \text{ Lösung für } K \Leftrightarrow 0, i_2, \ldots, i_n, k+1 \text{ Lösung für } f(K) \ .$$

2. Jede Lösung (= passende Indexfolge) kürzester Länge für $f(K)$ startet mit Index 0, endet mit Index $k+1$ und verwendet dazwischen nur Indizes aus $\{1, \ldots, k\}$.

Es folgt:
K besitzt eine passende Indexfolge **gdw** $f(K)$ besitzt eine passende Indexfolge.
Da f außerdem berechenbar ist, ergibt sich MPKP$\leq$PKP.

Reduktion von H auf MPKP

Ziel:

Entwurf einer Reduktionsabbildung f, die Eingaben von H der Form $w\#x$, so auf Eingaben von MPKP abbildet, dass gilt:

$$x \in H(M_w) \Leftrightarrow f(w\#x) \text{ hat eine Lösung } . \tag{3}$$

Notation: Im Folgenden schreiben wir einfach M statt M_w und benutzen die üblichen Symbole für die Komponenten von M.

Idee:

Um (3) zu erzwingen, werden die x- und y-Sequenzen Konfigurationsfolgen von M entsprechen, wobei die y-Sequenz immer eine Konfiguration Vorsprung hat. Der x-Sequenz erlauben wir erst nach Stoppen von M diesen Vorsprung einzuholen.

Normierung der Turing-Maschine M

Indem wir (falls nötig) die TM M leicht normieren (ohne ihr Stoppverhalten auf Eingabe x zu verändern), können wir o.E. voraussetzen:

1. M hat ein einseitig unendliches Band.
2. M bewegt in jedem Schritt den Kopf.
3. M druckt niemals ihr Leerzeichen $\Box$.
4. Es gibt eine ausgezeichnete Menge $S \subseteq Z$ von „Stoppzuständen“, so dass M genau dann stoppt, wenn sie einen der Zustände aus S erreicht.

Diese Normierung vereinfacht die folgende Konstruktion der MPKP-Eingabeinstanz $f(w\#x)$.

Die MPKP-Eingabeinstanz K=f(w#x)

Die Stringpaare von K zerfallen in fünf Gruppen:

- das Anfangspaar $(\#, \#z_0x\#)$
- Kopierpaare (X, X) für alle $X \in \Gamma \cup \{\#\}$
- Überführungspaare

$$
\begin{aligned}
&(zY, Y'z') && \text{, falls } \delta(z, Y) = (z', Y', R)\\
&(XzY, z'XY') && \text{, falls } \delta(z, Y) = (z', Y', L)\\
&(z\#, Y'z'\#) && \text{, falls } \delta(z, \square) = (z', Y', R)\\
&(Xz\#, z'XY'\#) && \text{, falls } \delta(z, \square) = (z', Y', L)
\end{aligned}
$$

  für alle $z \in Z \setminus S$, $z' \in Z$, $X, Y, Y' \in \Gamma \setminus \{\square\}$.

- Löschpaare (XsY, s), $(Xs\#, s\#)$, $(\#sY, \#s)$
  für alle $s \in S$, $X, Y \in \Gamma \setminus \{\square\}$.
- Abschlusspaare $(s\#\#, \#)$ für alle $s \in S$.

Intuition hinter diesem Design

- Das Anfangspaar dient dazu, der y-Sequenz eine Konfiguration (hier die Anfangskonfiguration) Vorsprung zu geben.
- Die Kopier- und Überführungspaare dienen dazu, beide Sequenzen um eine Konfiguration zu verlängern.
- Die Lösch- und Abschlußpaare sollen die x-Sequenz den Vorsprung aufholen lassen, sofern einen Zustand aus S erreicht wurde.

Um zu verifizieren, dass dieser Plan aufgeht, benötigen wir das folgende Konzept der *partiellen Lösung* für K.

Partielle Lösung für K=f(w#x)

Stringpaar $(x, y) \in \Sigma^* \times \Sigma^*$ heißt eine *partielle Lösung* für K ist, wenn gilt:

1. x ist Anfangswort von y.
2. Es existiert ein $n \geq 1$ und $i_1, \ldots, i_n$ mit $i_1 = 1$ (das Anfangspaar), so dass x die x-Sequenz und y die y-Sequenz zu $i_1, \ldots, i_n$ ist.

Zentrale Beobachtung:
Falls M aus Startkonfiguration z_0x die Folgekonfigurationen

$$\alpha_1 z_1 \beta_1, \alpha_2 z_2 \beta_2, \ldots, \alpha_k z_k \beta_k \text{ mit } z_0, \ldots, z_{k-1} \notin S$$

produziert, dann besitzt K eine partielle Lösung der Form

$$\begin{aligned}(x, y) \quad = \quad & (\#z_0 w \# \alpha_1 z_1 \beta_1 \# \ldots \# \alpha_{k-1} z_{k-1} \beta_{k-1} \# \ , \\ & \#z_0 w \# \alpha_1 z_1 \beta_1 \# \ldots \# \alpha_{k-1} z_{k-1} \beta_{k-1} \# \alpha_k z_k \beta_k \#)\end{aligned} \tag{4}$$

Induktiver Beweis der zentralen Beobachtung

Der Beweis erfolgt durch Induktion nach k.

k = 0: $(x, y) = (\#, \#z_0 w\#)$ realisiert durch das Anfangspaar.

Schritt von k auf k+1: Sei per Induktionsvoraussetzung eine partielle Lösung der Form (4) gegeben und $z_k \notin S$. Wir können die x-Sequenz um $\alpha_k z_k \beta_k \#$ und die y-Sequenz um $\alpha_{k+1} z_{k+1} \beta_{k+1} \#$ verlängern, indem wir

- die identischen Teile von Konfigurationen k und $k+1$ mit den Kopierpaaren aufbauen,
- die verschiedenen Teile (lokale Umgebung der Zustandssymbole z_k, z_{k+1}, weil dort jeweils der Kopf von M positioniert ist) mit dem eindeutig bestimmten Überführungspaar aufbauen.

Auf diese Weise erhalten wir die partielle Lösung

$$(x', y') = (y, y\alpha_{k+1} z_{k+1} \beta_{k+1} \#) \ .$$

Damit ist der induktive Beweis abgeschlossen.

Illustration des Induktionsschrittes

Es sei $ABCzDEF\#$ die Konfiguration, welche den Vorsprung der y-Sequenz ausmacht. Wir nehmen an, dass die Turing-Tafel von M die Aktion

$$\delta(z, D) = (z', D', L) \tag{5}$$

vorschreibt. Wir machen drei „Schnappschüsse“ der x- und y-Sequenz:

x-Sequenz (Einsatz Kopierpaare):	...	AB
y-Sequenz (Einsatz Kopierpaare):	...	$ABCzDEF\#AB$
x-Sequenz (Einsatz Überführungspaar):	...	$ABCzD$
y-Sequenz (Einsatz Überführungspaar:	...	$ABCzDEF\#ABz'CD'$
x-Sequenz (Einsatz Kopierpaare):	...	$ABCzDEF\#$
y-Sequenz (Einsatz Kopierpaare):	...	$ABCzDEF\#ABz'CD'EF\#$

Nachweis der Reduktionseigenschaften von f

Behauptung 1 Falls $x \in H(M)$, dann besitzt K eine Lösung.

Falls $x \in H(M)$, erhalten wir irgendwann eine partielle Lösung der Form

$$(y, y\alpha s\beta\#) \text{ mit } s \in S, \alpha, \beta \in \Gamma^* \ .$$

Nun können wir die Kopierpaare und die Löschpaare einsetzen, um den Vorsprung $\alpha s\beta\#$ zu vermindern:

- Jede Anwendung eines Löschpaares vermindert den Vorsprung um ein α- oder β-Symbol.
- Irgendwann ist der Vorsprung auf $s\#$ zusammengeschmolzen und die Sequenzen haben die Form

  $$(y', y's\#) \ .$$

  Anwendung des Abschlußpaares für s egalisiert die Sequenzen:

  $$(y's\#\#, y's\#\#)$$

Illustration der „Aufholjagd“

x-Sequenz:	...	
y-Sequenz (Endkonfiguration als Vorsprung):	...	$ABsD\#$
x-Sequenz (Einsatz Kopierpaar):	...	A
y-Sequenz (Einsatz Kopierpaar):	...	$ABsD\#A$
x-Sequenz (Einsatz Löschpaar):	...	$ABsD$
y-Sequenz (Einsatz Löschpaar):	...	$ABsD\#As$
x-Sequenz (Einsatz Kopierpaar):	...	$ABsD\#$
y-Sequenz (Einsatz Kopierpaar):	...	$ABsD\#As\#$
x-Sequenz (Einsatz Löschpaar):	...	$ABsD\#As\#$
x-Sequenz (Einsatz Löschpaar):	...	$ABsD\#As\#s\#$
x-Sequenz (Einsatz Abschlusspaar):	...	$ABsD\#As\#s\#\#$
x-Sequenz (Einsatz Abschlusspaar):	...	$ABsD\#As\#s\#\#$

Nachweis der Reduktionseigenschaften von f (fortgesetzt)

Behauptung 2 Falls K eine Lösung besitzt, dann gilt $x \in H(M)$.

Indirekter Beweis: Wir zeigen, dass K keine Lösung besitzt, falls $x \notin H(M)$.

Da wir bei der Produktion von partiellen Lösungen keine Freiheiten hatten (Anwendung von anderen Paaren als die beschriebenen führt immer direkt in eine Sackgasse), besteht der einzige Lösungsversuch im Produzieren partieller Lösungen der Form (4), wobei (solange kein Zustand aus S erreicht wird), die Löschpaare nicht zum Einsatz kommen und die y-Sequenz daher stets eine Konfiguration Vorsprung hat. Eine Egalisierung der Sequenzen kann nicht stattfinden.

PKP mit binärem Alphabet

01-PKP sei das PKP über dem Alphabet $\Sigma = \{0, 1\}$.

Satz: PKP$\leq$01-PKP.

Beweis: Sei K eine PKP-Eingabeinstanz über einem beliebigen Alphabet der Form $\Sigma = \{a_1, \ldots, a_m\}$. Wähle als $f(K)$ die 01-PKP-Eingabeinstanz, die aus K durch die Substitutionsregel

$$a_j \mapsto 10^j$$

hervorgeht. Die Lösungen (sofern vorhanden) für K und $f(K)$ entsprechen sich (in der offensichtlichen Weise) 1-zu-1. Insbesondere besitzt K eine Lösung **gdw** $f(K)$ eine Lösung besitzt.

Folgerung: 01-PKP ist unentscheidbar.

Unentscheidbare Probleme mit Grammatiken und Automaten

Zu 01-PKP assoziierte kontextfreie Sprachen

Zu einem Wort $w = w_1 \cdots w_n$ bezeichne $\tilde{w} = w_n \cdots w_1$ sein „Spiegelbild“. Beachte: $\tilde{uv} = \tilde{v}\tilde{u}$. Zu der Eingabeinstanz

$$K = [(x_1, y_1), \ldots, (x_k, y_k)]$$

von 01-PKP assoziieren wir die folgenden Sprachen über dem Alphabet $\Sigma = \{0, 1, \$, a_1, \ldots, a_k, \}$:

$$\begin{aligned} L_1[K] &:= \{a_{i_m} \cdots a_{i_1} x_{i_1} \cdots x_{i_m} \$ \tilde{y}_{j_n} \cdots \tilde{y}_{j_1} a_{j_1} \cdots a_{j_n} \mid m, n \geq 1\} \\ L_2[K] &:= \left\{uv\$\tilde{v}\tilde{u} \mid u \in \{a_1, \ldots, a_k\}^+, v \in \{0,1\}^+\right\} \end{aligned}$$

Erinnerung: $i_1, \ldots, i_n$ ist eine Lösung für K **gdw** $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$.

Zentrale Beobachtung: $L_1[K] \cap L_2[K]$ ist identisch zu

$$\{a_{i_m} \cdots a_{i_1} x_{i_1} \cdots x_{i_m} \$ \tilde{y}_{i_m} \cdots \tilde{y}_{i_1} a_{i_1} \cdots a_{i_m} \mid i_1, \ldots, i_n \text{ ist eine Lösung für } K\} \ .$$

(Deterministische) Kontextfreiheit dieser Sprachen

Wir geben hier kfG's $G_1 = G_1[K]$, $G_2 = G_2[K]$ für $L_1 = L_1[K]$, $L_2 = L_2[K]$ an:

1. G_1 enthält die Regeln

$$S_1 \to A\$B \ , \ A \to a_i A x_i \mid a_i x_i \ , \ B \to \tilde{y}_i B a_i \mid \tilde{y}_i a_i$$

   für $i = 1, \ldots, k$.

2. G_2 enthält die Regeln

$$S_2 \to a_i S_2 a_i \mid a_i R a_i \ , \ R \to 0R0 \mid 1R1 \mid 0\$0 \mid 1\$1$$

   für $i = 1, \ldots, k$.

Bemerkung: Grammatiken G_1 und G_2 sind eindeutig und es lassen sich auch DPDA's $M_1 = M_1[K]$ und $M_2 = M_2[K]$ für L_1 und L_2 angeben. (Der Beweis hierfür wäre eine gute Übungsaufgabe.)

Folgerung: L_1, L_2 sind eindeutige deterministisch kontextfreie Sprachen.

Beispiel

Zu $K = [(1, 111), (10111, 10), (10, 0)]$ erhalten wir bei G_1 die Regeln

$$\begin{aligned} S_1 &\rightarrow A\$B \\ A &\rightarrow a_1A1|a_2A10111|a_3A10|a_11|a_210111|a_310 \\ B &\rightarrow 111Ba_1|01Ba_2|0B|a_3|111a_1|01a_2|0 \end{aligned}$$

und bei G_2 die Regeln

$$\begin{aligned} S_2 &\rightarrow a_1S_2a_1|a_2S_2a_2|a_3S_2a_3|a_1Ra_1|a_2Ra_2|a_3Ra_3 \\ R &\rightarrow 0R0|1R1|0\$0|1\$1 \ . \end{aligned}$$

Zwei sehr unterschiedliche Szenarios

K hat eine Lösung	K hat keine Lösung
1. $L_1[K] \cap L_2[K] \neq \emptyset$	1'. $L_1[K] \cap L_2[K] = \emptyset$
2. $\lvert L_1[K] \cap L_2[K] \rvert = \infty$	2'. $\lvert L_1[K] \cap L_2[K] \rvert < \infty$
3. $L_1[K] \cap L_2[K]$ ist nicht kontextfrei	3'. $L_1[K] \cap L_2[K]$ ist kontextfrei
4. $\overline{L_1[K]} \cup \overline{L_2[K]}$ ist nicht regulär	4'. $\overline{L_1[K]} \cup \overline{L_2[K]}$ ist regulär
5. $\overline{L_1[K]} \cup \overline{L_2[K]}$ ist nicht deterministisch kontextfrei	5'. $\overline{L_1[K]} \cup \overline{L_2[K]}$ ist deterministisch kontextfrei

Begründungen

- Unsere „zentrale Beobachtung“ liefert 1. und 1’.
- Aus $L_1 \cap L_2 = \emptyset$ (und somit $\overline{L_1} \cup \overline{L_2} = \overline{L_1 \cap L_2} = \Sigma^*$) ergeben sich unmittelbar die Aussagen 2’. bis 5’.
- Zu Aussage 2. nutze aus, dass K unendlich viele Lösungen hat, sofern es mindestens eine Lösung hat.
- Aussage 3. ergibt sich leicht mit dem Pumping–Lemma. (Der Beweis hierfür wäre eine gute Übungsaufgabe.)
- Aussage 5. ergibt sich mit einem Beweis durch Widerspruch:
  Wäre $\overline{L_1} \cup \overline{L_2}$ deterministisch kontextfrei, dann müsste auch $L_1 \cap L_2$ deterministisch kontextfrei sein im Widerspruch zu 3.
- Aussage 4. ergibt sich direkt aus 5.

Folgerungen

Die folgenden Probleme sind unentscheidbar (Reduktion von PKP bzw. Komplement von PKP auf das betreffende Problem):

1. Ist der Durchschnitt zweier (durch DPDAs oder kfGs gegebener) deterministisch kontextfreier Sprachen leer (Schnittproblem für deterministisch kontextfreie Sprachen) ?
2. Ist der Durchschnitt zweier (durch DPDAs oder kfGs gegebener) deterministisch kontextfreier Sprachen endlich ?
3. Ist der Durchschnitt zweier (durch DPDAs oder kfGs gegebener) deterministisch kontextfreier Sprachen kontextfrei (Kontextfreiheit des Durchschnittes) ?
4. Ist eine (durch eine kfG gegebene) kontextfreie Sprache regulär ?
5. Ist eine (durch eine kfG gegebene) kontextfreie Sprache deterministisch kontextfrei ?

Zugehörige Reduktionsabbildungen

- Für die ersten drei Reduktionen verwende die Reduktionsabbildung

$$K \mapsto (M_1[K], M_2[K]) \text{ bzw. } K \mapsto (G_1[K], G_2[K]) \ .$$

- Für die letzten zwei Reduktionen verwende die Reduktionsabbildung $K \mapsto G'[K]$, wobei $G'[K]$ eine (aus K berechenbare !) kfG für die (kontextfreie !) Sprache $\overline{L_1[K]} \cup \overline{L_2[K]}$ ist.

Da die kfGs $G_1[K], G_2[K]$ eindeutig sind, bleiben die ersten drei Probleme der obigen Liste unentscheidbar, selbst wenn die beteiligten kontextfreien Sprachen durch eindeutige kfGs gegeben sind.

Weitere unentscheidbare Probleme

Inklusionsproblem: Gilt für zwei (durch DPDAs oder kfGs gegebene) deterministisch kontextfreie Sprachen L_1, L_2 die Beziehung $L_1 \subseteq L_2$?

Nachweis der Unentscheidbarkeit: Das Schnittproblem für deterministisch kontextfreie Sprachen

$$N(M_1) \cap N(M_2) = \emptyset \ ?$$

ist wegen

$$N(M_1) \cap N(M_2) = \emptyset \Leftrightarrow N(M_1) \subseteq \overline{N(M_2)}$$

auf das Inklusionsproblem reduzierbar. Als Reduktionsabbildung verwende

$$(M_1, M_2) \mapsto (M_1, M_2') \ ,$$

wobei M_2' ein (aus M_2 berechenbarer) DPDA für $\overline{N(M_2)}$ ist.

Weitere unentscheidbare Probleme (fortgesetzt)

Eindeutigkeitsproblem: Ist eine gegebene kfG eindeutig ?

Nachweis der Unentscheidbarkeit: Das Schnittproblem für eindeutige kontextfreie Sprachen

$$L(G_1) \cap L(G_2) = \emptyset \text{ } (G_1, G_2 \text{ eindeutige kfGs}) \text{ ?}$$

ist auf das Eindeutigkeitsproblem reduzierbar. Als Reduktionsabbildung verwende

$$(G_1, G_2) \mapsto G_3 \ ,$$

wobei G_3 die (aus G_1, G_2 mit der Standardtechnik berechenbare) kfG für die Sprache $L(G_1) \cup L(G_2)$ bezeichnet. Da G_1 und G_2 eindeutige kfGs sind, gibt es ein Wort mit zwei verschiedenen Syntaxbäumen über G_3 **gdw** $L(G_1) \cap L(G_2) \neq \emptyset$.

Weitere unentscheidbare Probleme (fortgesetzt)

Kontextfreiheit des Komplementes: Ist das Komplement einer (durch eine kfG gegebenen) kontextfreien Sprache ebenfalls kontextfrei ?

Nachweis der Unentscheidbarkeit: Das Problem der Kontextfreiheit des Durchschnittes

$$N(M_1) \cap N(M_2) \text{ kontextfrei } (M_1, M_2 \text{ DPDAs}) \text{ ?}$$

ist auf das Problem der Kontextfreiheit des Komplementes reduzierbar. Verwende die Reduktionsabbildung

$$(M_1, M_2) \mapsto G_3' \ ,$$

wobei G_3' die (aus M_1, M_2 berechenbare) kfG für

$$\overline{N(M_1) \cap N(M_2)} = \overline{N(M_1)} \cup \overline{N(M_2)}$$

bezeichne.

Weitere unentscheidbare Probleme (fortgesetzt)

Leerheit des Komplementes: Stimmt eine (durch eine kfG gegebene) kontextfreie Sprache mit Σ^* überein (d.h., sind alle Terminalstrings ableitbar) ?

Nachweis der Unentscheidbarkeit: Dieselbe Abbildung

$$(M_1, M_2) \mapsto G_3' \quad , \quad L(G_3') = \overline{N(M_1)} \cup \overline{N(M_2)} \ ,$$

wie in der vorangegangenen Reduktion reduziert das Schnittproblem für deterministisch kontextfreie Sprachen

$$N(M_1) \cap N(M_2) = \emptyset \ ?$$

auf das Problem der Leerheit des Komplementes.

Weitere unentscheidbare Probleme (fortgesetzt)

Äquivalenzproblem: Sind zwei gegebene kfGs äquivalent (d.h., sind die von ihnen erzeugten Sprachen identisch) ?

Nachweis der Unentscheidbarkeit: Das Problem der Leerheit des Komplementes

$$L(G) = \Sigma^* \ ?$$

ist auf das Äquivalenzproblem reduzierbar. Wenn G_* eine (leicht zu berechnende) kfG für Σ^* bezeichnet, dann kann als Reduktionsabbildung

$$G \mapsto (G, G_*)$$

verwendet werden.

Unentscheidbare Probleme für kontextsensitive Sprachen

„Meta–Satz“: Jedes Problem, das für den Durchschnitt zweier kontextfreier Sprachen unentscheidbar ist, ist auch für eine einzelne kontextsensitive Sprache unentscheidbar.

Beweis:

- Die gegebenen kontextfreien Grammatiken G_1, G_2 sind (erst recht) kontextsensitive Grammatiken.
- Daher sind sie (mit der Methode der Vorlesung) in äquivalente LBAs M_1, M_2 transformierbar.
- Aus M_1, M_2 lässt sich leicht ein LBA M_3 für

$$T(M_3) = T(M_1) \cap T(M_2) = L(G_1) \cap L(G_2)$$

zusammenbasteln.

- Daher hat die zu $L(G_1) \cap L(G_2)$ gestellte Frage (wie immer sie lautet) die selbe Antwort wie die zu $T(M_3)$ gestellte Frage (Reduktionsabbildung $(G_1, G_2 \mapsto M_3)$.

Folgerungen

Die folgenden Probleme zu einer (durch einen LBA gegebenen) kontextsensitiven Sprache sind unentscheidbar:

- Ist die Sprache leer ?
- Ist die Sprache endlich ?
- Ist die Sprache kontextfrei ?