Übersicht

Übersicht

Berechenbare Funktionen

Eine (partielle) Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist berechenbar, wenn es einen Algorithmus gibt, der

- bei Eingabe von $(n_1, \ldots, n_k)$ genau dann terminiert, wenn $f(n_1, \ldots n_k)$ definiert ist,
- und dann den Wert $f(n_1, \ldots n_k)$ ausgibt.

Thema dieses zweiten Teils ist die Frage nach den prinzipiellen Grenzen dessen, was mit Computern möglich ist.
Positive Aussagen können dabei informell – durch Angabe eines Algorithmus – belegt werden. Um aber zu zeigen, dass ein gewisses Problem grundsätzlich nicht lösbar ist, braucht es ein formales Modell, über das Aussagen mit mathematischen Methoden bewiesen werden können. Wir werden ein Maschinenmodell kennenlernen, die *Turing-Maschine*, und dann argumentieren, dass dieses den intuitiven Algorithmenbegriff erfasst.
Die Beschränkung auf natürliche Zahlen als Bereich ist eigentlich keine, da alle im Computer verarbeitetet Objekte als Bitstrings, und somit als natürliche Zahlen, codiert werden können.
Beispiele berechenbarer Funktionen sind die nirgends definierte Funktion, sowie die Funktion f mit $f(n) = \begin{cases} 0 & \text{falls die Goldbach-Vermutung wahr ist,} \\ 1 & \text{sonst.} \end{cases}$
Dieses Beispiel zeigt, dass nur die Existenz eines Algorithmus gefordert ist. Für die obige Funktion existiert ein Algorithmus: sie ist entweder die Konstante 0 oder die Konstante 1, auch wenn niemand weiß, welcher von beiden Algorithmen es ist.
Dieses Beispiel ist künstlich, aber es gibt im Bereich der Graphentheorie natürliche Probleme, für die bekannt ist, dass es einen Algorithmus geben muss, obwohl keiner bekannt ist. Dies folgt aus dem sog. Minorensatz für Graphen.

Die Turing-Maschine

Eingeführt von Alan M. Turing in:

On computable numbers, with an application to the Entscheidungsproblem, 1936

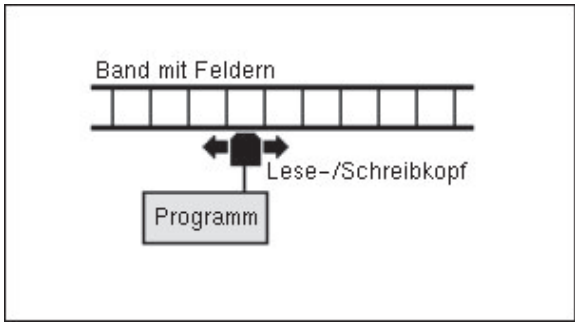

Photo ©National Portrait Gallery, London

Grafik von de.wikipedia.org

Alan M. Turing war ein britischer Mathematiker, nach ihm ist der *Turing-Award* benannt, der bedeutendste Forschungspreis in der Informatik.

Das von ihm entwickelte und heute Turing-Maschine genannte Modell einer Rechenmaschine ist heute in der theoretischen Informatik das Standardmodell. Obwohl es schon vorher ähnliche formale Modelle gab, liegt die Bedeutung von Turings Arbeit darin, dass er zeigen konnte, dass es *eine* solche Maschine gibt, die jede andere simulieren kann - also eine universelle Rechenmaschine, die jede mögliche Berechnung ausführen kann, wie eben ein programmgesteuerter Computer.

Turing arbeitet im zweiten Weltkrieg als Kryptologe und leistete entscheidende Beiträge zum Erfolg der Alliierten, da es ihm geang, den Code des deutschen Verschlüsselungsgerätes Enigma zu brechen. Er starb 1954 im Alter von 43 Jahren vermutlich durch Selbstmord, nachdem er zuvor wegen seiner Homosexualität verurteilt wurde.

Die Turing-Maschine besteht aus einer Steuerung sowie einem beidseitig unendlichen Band mit einem Schreib-/Lesekopf, der jeweils eine Bandzelle lesen und überschreiben kann. Die Steuerung (Programm) entspricht einem endlichen Automaten, die Turing-Maschine kann also als “endlicher Automat mit Speicher” aufgefasst werden.

Turing-Maschine: Definition

Eine deterministische Turing-Maschine (DTM) M mit Alphabet Σ besteht aus:

- Zustände Q
- Bandalphabet $\Gamma \supseteq \Sigma$
- Leerzeichen $\square \in \Gamma \setminus \Sigma$
- Übergangsfunktion $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- Anfangszustand $q_0 \in Q$
- Endzustände $F \subseteq Q$

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

Wie beim DEA ist die Zustandsmenge Q endlich. Das Bandalphabet Γ umfasst das Eingabealphabet Σ, kann aber weitere Symbole enthalten. Insbesondere enthält es das Leerzeichen $\square$. Die Vorstellung ist, dass das Band zwar unendlich ist, aber zu jedem Zeitpunkt ist nur ein endlicher Teil mit Zeichen beschrieben, der Rest enthält Leerzeichen.

Wichtig ist, dass die Übergangsfunktion δ hier partiell ist, dass also $\delta(q, a)$ für manche Zustände $q \in Q$ und Symbole $a \in \Gamma$ undefiniert ist. Ist $\delta(q, a)$ definiert, so ist der Wert ein Tripel (p, b, D) aus dem neuen Zustand $p \in Q$, dem geschriebenen Symbol $b \in \Gamma$ und der Richtung der Kopfbewegung $D = L$ (links) oder $D = R$ (rechts).

Konfigurationen

Eine Konfiguration α von M ist eine Zeichenkette

$$\alpha = a_1 a_2 \dots a_{i-1} q a_i a_{i+1} \dots a_n$$

mit $a_i \in \Gamma$ für $i \leq n$ und $q \in Q$.

Interpretation:

- M ist in Zustand q
- Der Kopf ist auf Bandzelle i, in der a_i steht
- $a_1 a_2 \dots a_{i-1}$ ist der Bandinhalt links vom Kopf
- Links von a_1 sind nur $\square$ auf dem Band
- $a_{i+1} \dots a_n$ ist der Bandinhalt rechts vom Kopf
- Rechts von a_n sind nur $\square$ auf dem Band

Diese Darstellung ist für viele Zwecke bequem, da man sich nicht extra merken muss, an welcher Stelle auf dem Band der Schreib-/Lesekopf steht, dies wird durch den Zustand markiert. Damit die Notation eindeutig ist, müssen Bandsymbole und Zustände verschieden voneinander sein, d.h. $\Gamma \cap Q = \emptyset$.

Vergleichen Sie dies auch mit den Konfigurationen eines PDA: dort wurde sich der neben dem Zustand der noch zu lesende Input und der Stackinhalt gemerkt. Ersteres braucht man sich hier nicht extra merken, weil der Input zu Beginn auf dem Band steht und nicht anders behandelt wird als der Bandinhalt. Dafür brauchte man beim PDA keine Position auf dem Stack zu speichern, weil beim Stack immer nur das oberste Symbol sichtbar ist.

Übergangsrelation

Die Relation $\vdash_M$ zwischen Konfigurationen wird definiert:

- Ist $\delta(q, a_i) = (p, b, L)$, dann gilt:

  $a_1 \dots a_{i-1} q a_i a_{i+1} \dots a_n \vdash_M a_1 \dots a_{i-2} p a_{i-1} b a_{i+1} \dots a_n$

  $q a_1 a_2 \dots a_n \vdash_M p \square b a_2 \dots a_n$ für $i = 1$

  $a_1 \dots a_{n-1} q a_n \vdash_M a_1 \dots a_{n-2} p a_{n-1}$ für $i = n$ und $b = \square$

- Ist $\delta(q, a_i) = (p, b, R)$, dann gilt:

  $a_1 \dots a_{i-1} q a_i a_{i+1} \dots a_n \vdash_M a_1 \dots a_{i-1} b p a_{i+1} \dots a_n$

  $a_1 \dots a_{n-1} q a_n \vdash_M a_1 a_2 \dots a_{n-1} b p \square$ für $i = n$

  $q a_1 a_2 \dots a_n \vdash_M p a_2 \dots a_n$ für $i = 1$ und $b = \square$

Die Übergangsrelation zwischen Konfigurationen beschreibt einen Rechenschritt der Maschine, und wird je nach Wert der Übergangsfunktion $\delta(q, a_i)$ für den Zustand q und das gelesene Symbol in der Ausgangskonfiguration definiert.

Dabei sind zunächst zwei Basisfälle zu unterscheiden, je nach Richtung der Kopfbewegung. In jedem der Fälle steht der Normalfall in der ersten Zeile. Daneben gibt es je zwei Sonderfälle:

- in der jeweils zweiten Zeile ist der Fall beschrieben, dass der Kopf über das bisher gesehene Bandstück hinausgeht. In diesen Fällen muss das Bandstück in der Konfiguration links bzw. rechts um ein Leerzeichen ergänzt werden, auf dem der Kopf dann steht.
- in der jeweils dritten Zeile ist der Fall beschrieben, wo ein Symbol am Rand des Bandes mit einem Leerzeichen überschrieben wird und der Kopf sich dann vom Rand wegbewegt. In diesem Fall verschmilzt dieses Leerzeichen mit dem nicht notierten String von Leerzeichen außerhalb des in der Konfiguration beschriebenen Bereichs, und dieser wird um ein Zeichen kürzer.

Für jede Konfiguration gibt es entweder genau eine Folgekonfiguration, wenn die Übergangsfunktion $\delta(q, a_i)$ definiert ist, oder sonst keine.

Berechnung, Halten

Die Berechnung von M bei Eingabe $w \in \Sigma^*$ ist die eindeutig definierte Folge von Konfigurationen

$$\alpha_1, \alpha_2, \alpha_3, \ldots$$

mit $\alpha_1 = q_0 w$ und $\alpha_i \vdash_M \alpha_{i+1}$ für alle i.

Zwei Möglichkeiten:

- Berechnung endet mit einer Konfiguration $vqav'$, für die $\delta(q, a)$ undefiniert ist.
  $\rightsquigarrow$ M hält bei Eingabe w.
- Berechnung ist unendlich.

In der Anfangskonfiguration α_1 ist die Maschine im Anfangszustand q_0, auf dem Band steht nur das Eingabewort w, und der Kopf steht auf dem ersten Symbol von w.

Hier sieht man, warum die Funktion δ partiell sein muss: sonst würde keine Berechnung jemals halten.

(Semi-)Entscheidbarkeit

Eine DTM M akzeptiert die Sprache $L \subseteq \Sigma^*$, wenn

- M hält bei Eingabe w genau dann, wenn $w \in L$ ist.

$L \subseteq \Sigma^*$ ist semi-entscheidbar, wenn es eine DTM gibt, die L akzeptiert.

Eine DTM M entscheidet die Sprache $L \subseteq \Sigma^*$, wenn

- M hält bei Eingabe w für alle $w \in \Sigma^*$
- für die Endkonfiguration vqv' ist $q \in F$ genau dann, wenn $w \in L$ ist.

$L \subseteq \Sigma^*$ ist entscheidbar, wenn es eine DTM gibt, die L entscheidet.

Entscheidbare Sprachen sind also solche, für die das Wortproblem "*Ist* $w \in L$?" von einem Computer gelöst werden kann. Da Entscheidungsprobleme aus anderen Bereichen durch Codierung in das Wortproblem für eine Sprache übersetzt werden können, spricht man i.A. von entscheidbaren Problemen.

Semi-entscheidbare Probleme sind dagegen nicht wirklich algorithmisch lösbar, für diese gibt es nur einen Algorithmus, der den positiven Fall feststellt, aber im negativen Fall nicht terminiert.

Beispiel

Eine DTM, die $\{0^n1^n\ ;\ n \in \mathbb{N}\}$ entscheidet:

Anfangszustand ist q_0,

Endzustände sind $\{q_4\}$

	0	1	x	y	$\square$
q_0	(q_1, x, R)			(q_3, y, R)	
q_1	$(q_1, 0, R)$	(q_2, y, L)		(q_1, y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, y, L)	
q_3				(q_3, y, R)	$(q_4, \square, R)$
q_4					

Bei Eingabe von 0011 durchläuft die Maschine die folgende Folge von Konfigurationen:

$$q_0 0011 \vdash x q_1 011 \vdash x 0 q_1 11 \vdash x q_2 0 y 1 \vdash q_2 x 0 y 1 \vdash x q_0 0 y 1$$
$$\vdash x x q_1 y 1 \vdash x x y q_1 1 \vdash x x q_2 y y \vdash x q_2 x y y \vdash x x q_0 y y \vdash x x y q_3 y$$
$$\vdash x x y y q_3 \square \vdash x x y y \square q_4 \square$$

Hier hält die Berechnung, und da der Zustand q_4 ein Endzustand ist, liegt das Wort in der Sprache.

Bei Eingabe von 0010 durchläuft die Maschine die folgende Folge von Konfigurationen:

$$q_0 0010 \vdash x q_1 010 \vdash x 0 q_1 10 \vdash x q_2 0 y 0 \vdash q_2 x 0 y 0 \vdash x q_0 0 y 0$$
$$\vdash x x q_1 y 0 \vdash x x y q_1 0 \vdash x x y 0 q_1 \square$$

Hier hält die Berechnung, und da der Zustand q_1 kein Endzustand ist, liegt das Wort nicht in der Sprache.

Turing-Berechenbarkeit

Eine DTM M berechnet die Funktion $f : \Sigma^* \to \Sigma^*$, wenn

- M hält bei Eingabe w genau dann, wenn $f(w)$ definiert ist.
- die Endkonfiguration ist dann qv mit $v = f(w)$.

$f : \mathbb{N}^k \to \mathbb{N}$ ist Turing-berechenbar, wenn es eine DTM gibt, die die Funktion

$$f' : \{1, \#\}^* \to \{1, \#\}^*$$
$$1^{n_1}\#1^{n_2}\#\cdots\#1^{n_k} \mapsto 1^{f(n_1,\ldots,n_k)}$$

berechnet.

Funktionen auf Strings werden berechnet, indem am Anfang die Eingabe auf dem Band steht, und wenn die Maschine hält wird das Ergebnis vom Band abgelesen. Dabei sollte gelten, dass am Ende nur ein String aus Σ^* und Leerzeichen auf dem Band stehen.

Funktionen auf natürlichen Zahlen werden berechnet, indem die Eingaben unär codiert und mit einem Trennzeichen voneinander getrennt eingegeben werden, das Ergebnis wird ebenso unär codiert ausgelesen.

Beispiel

Eine DTM, die die Funktion $(n, m) \mapsto \max(n - m, 0)$ berechnet:

	1	#	□
q_0	$(q_1, \square, R)$	$(q_5, \square, R)$	
q_1	$(q_1, 1, R)$	$(q_2, \#, R)$	
q_2	$(q_3, \#, L)$	$(q_2, \#, R)$	$(q_4, \square, L)$
q_3	$(q_3, 1, L)$	$(q_3, \#, L)$	$(q_0, \square, R)$
q_4	$(q_4, 1, L)$	$(q_4, \square, L)$	$(q_6, 1, R)$
q_5	$(q_5, \square, R)$	$(q_5, \square, R)$	$(q_6, \square, R)$
q_6			

Bei Eingabe von 1111#11 führt die Maschine die folgende Berechnung aus:

$$
\begin{aligned}
& q_0 1111\#11 \vdash q_1 111\#11 \vdash \ldots \vdash 111 q_1 \#11 \vdash 111\# q_2 11 \\
\vdash\ & 111 q_3 \#\#1 \vdash \ldots \vdash q_3 111\#\#1 \vdash q_3 \square 111\#\#1 \\
\vdash\ & q_0 111\#\#1 \vdash \ldots \vdash 11 q_1 \#\#1 \vdash 11\# q_2 \#1 \\
\vdash\ & 11\#\# q_2 1 \vdash \ldots \vdash q_3 \square 11\#\#\# \vdash q_0 11\#\#\# \vdash q_1 1\#\#\# \\
\vdash\ & 1 q_1 \#\#\# \vdash \ldots \vdash 1\#\#\# q_2 \square \vdash 1\#\# q_4 \# \\
\vdash\ & 1\# q_4 \# \vdash \ldots \vdash q_4 1 \vdash q_4 \square 1 \vdash 1 q_6 1
\end{aligned}
$$

An dieser Stelle hält die Berechnung, und auf dem Band steht die Codierung des Ergebnisses 2, was korrekt ist, da $4 - 2 = 2$ ist.

Churchsche These

Weitere Formalisierungen des intuitiven Algorithmen-Begriffs:

- Registermaschinen
- rekursive Funktionen
- Lambda-Kalkül
- ...

Alle sind äquivalent zur Turing-Maschine.

Auch: Church-Turing-These

Die Turing-berechenbaren Funktionen erfassen genau den intuitiven Begriff von Berechenbarkeit.

Die Äquivalenz der verschiedenen Modelle, die z.T. von sehr verschiedener Natur sind, führt zu der Annahme, dass diese Modelle den intuitiven Begriff von algorithmischer Berechenbarkeit tatsächlich genau erfassen, der sog. Churchschen These.

Wegen der Churchschen These werden wir im Folgenden von jedem irgendwie spezifizierten Algorithmus annehmen, dass er als Turing-Maschine implementiert werden kann. In jedem konkreten Fall wäre die Konstruktion einer entsprechenden Maschine aufwändig, aber nicht wirklich schwierig.

Das Bild zeigt Alonzo Church, den Doktorvater von Alan Turing (und von dem im Teil I erwähnten Stephen Kleene). Church hat u.a. den Lambda-Kalkül eingeführt, ein weiteres zur Turing-Maschine äquivalentes Berechnungsmodell. Dieser bildet die mathematische Grundlage funktionaler Programmiersprachen wie LISP und ML, und ist Ihnen vermutlich aus der Vorlesung *Programmierung und Modellierung* bekannt.