

Turing-Maschinen

Unentscheidbarkeit

- Codierung von Turing-Maschinen
- Das Halteproblem
- Reduktion

Unentscheidbare Probleme

Codierung von Turing-Maschinen

Konventionen für alle DTM:

- ▶ $\Sigma = \{0, 1\}$
- ▶ $Q = \{q_1, q_2, \dots, q_k\}$ für ein k
- ▶ Anfangszustand ist q_1 , Endzustände $F = \{q_2\}$
- ▶ $\Gamma = \{a_1, a_2, \dots, a_m\}$ für ein m
- ▶ $a_1 = 0$, $a_2 = 1$ und $a_3 = \square$
- ▶ $D_1 := L$ und $D_2 := R$

Codierung von Turing-Maschinen

Jeder Übergang $\delta(q_i, a_j) = (q_k, a_\ell, D_m)$ wird codiert durch den String

$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m.$$

Sind C_1, C_2, \dots, C_n die Codes der Übergänge von M , dann ist der Code für M der String

$$C_1 11 C_2 11 \dots 11 C_n$$

Ist $e \in \Sigma^*$, so bezeichnet M_e die DTM, deren Code e ist.

Die Konvention, dass jede DTM nur einen Endzustand hat, ist keine echte Einschränkung: jede DTM kann leicht so modifiziert werden, dass sie diese Bedingung erfüllt, ohne dass sich an ihrem Verhalten sonst etwas ändert.

Ist ein Wort e nicht von der Form, die durch Codierung einer DTM entsteht, z.B. wenn es drei oder mehr 1 hintereinander enthält, definieren wir M_e als eine beliebige, aber fest gewählte Turing-Maschine, z.B. die triviale Maschine, die nur einen Zustand und eine nirgends definierte Übergangsfunktion hat.

Es gibt eine **universelle Turing-Maschine** U mit:

- ▶ U hält bei Eingabe (e, w) genau dann, wenn M_e bei Eingabe w hält.
- ▶ U erreicht bei (e, w) den Endzustand, genau dann, wenn M_e bei w den Endzustand erreicht.
- ▶ U berechnet bei (e, w) die selbe Ausgabe wie M_e bei w .

(Das Paar (e, w) wird codiert durch $e111w$)

Die universelle Maschine ist für den Informatiker etwas ganz natürliches, nämlich ein *Interpreter* für Turing-Maschinen, geschrieben als Turing-Maschine, so wie es Interpreter für Programmiersprachen gibt, die in der jeweiligen Sprache selbst geschrieben sind.

Aus Kardinalitätsgründen gibt es unentscheidbare Sprachen:

- ▶ es gibt nur **abzählbar** viele Turing-Maschinen,
- ▶ aber es gibt **überabzählbar** viele Sprachen $L \subseteq \{0, 1\}^*$.

Im Folgenden: eine **konkrete** Sprache, die unentscheidbar ist.

Zur Erinnerung: eine unendliche Menge M ist abzählbar, wenn sie gleichmächtig mit \mathbb{N} ist, d.h. wenn es eine bijektive Abbildung von \mathbb{N} auf M gibt. In diesem Fall lassen sich die Elemente von M aufzählen als $M = \{m_0, m_1, m_2, \dots\}$

Da $\{0, 1\}^*$ gleichmächtig mit \mathbb{N} ist (eine Bijektion ist z.B. die *dyadische Codierung*), und sich DTM in Wörter über $\{0, 1\}$ codieren lassen, gibt es also nur abzählbar viele DTM.

Sprachen dagegen sind wegen der Bijektion in 1-1-Korrespondenz mit Mengen von natürlichen Zahlen, von denen es überabzählbar viele gibt. Hier zur Erinnerung und wegen der Analogie zum Folgenden der Beweis mittels der Diagonalisierungsmethode von Cantor:

Angenommen, es gäbe nur abzählbar viele Teilmengen von \mathbb{N} , aufgezählt als N_0, N_1, N_2, \dots

Dann definiere die Menge $D := \{i \in \mathbb{N}; i \notin N_i\}$. Die Menge D müsste in der Aufzählung vorkommen, also $D = N_d$ für ein $d \in \mathbb{N}$. Dann ist aber

$$d \in D \quad \text{gdw.} \quad d \notin N_d \quad \text{gdw.} \quad d \notin D$$

ein Widerspruch! Also ist die Annahme falsch, und es gibt mehr als abzählbar viele Teilmengen von \mathbb{N} .

Das **Halteproblem** ist die Sprache

$$H := \{ (e, w) ; M_e \text{ hält bei Eingabe } w \}$$

Theorem

Das Halteproblem H ist unentscheidbar.

Die Unentscheidbarkeit des Halteproblems hat eigentlich nichts mit Turing-Maschinen zu tun, sie lässt sich nur so am einfachsten formulieren und beweisen. Die selbe Aussage gilt aber - mit im wesentlichen dem selben Beweis - auch für jeden anderen Berechnungsformalismus, also auch für jede Programmiersprache.

Beweis der Unentscheidbarkeit von H

Turing-Maschinen

Unentscheidbarkeit

Codierung von

Turing-Maschinen

Das Halteproblem

Reduktion

Unentscheidbare
Probleme

Angenommen, M_H sei eine DTM, die H entscheidet.

Konstruiere eine neue DTM D wie folgt:

Bei Eingabe w schreibe (w, w) auf Band; lasse M_H laufen.

Endet M_H im Endzustand, starte Endlosschleife.

Andernfalls halte im Endzustand.

Sei d Code dieser Maschine, also $D = M_d$. Dann gilt:

D hält bei Eingabe d

$\leftrightarrow M_d$ hält bei Eingabe d

$\leftrightarrow (d, d) \in H$

$\leftrightarrow M_H$ erreicht Endzustand bei Eingabe (d, d)

$\leftrightarrow D$ hält nicht bei Eingabe d

Widerspruch! Also kann es M_H nicht geben.

Beachten Sie die strukturelle Ähnlichkeit mit dem Diagonalisierungsbeweis für die Überabzählbarkeit der Potenzmenge von \mathbb{N} .

Die Maschine D betrachtet die Eingabe w as Code einer Turing-Maschine, und fragt (mittels der angenommenen Maschine für H), ob diese bei ihrem eigenen Code als Eingabe hält – und verhält sich dann genau umgekehrt: wenn M_w bei Eingabe w hält, dann betritt sie eine Endlosschleife, hält also gerade nicht, andernfalls hält sie.

Wegen der Churchschen These glauben wir, dass sich die informell beschriebene Maschine D tatsächlich als Turingmaschine realisieren lässt. Eine direkte Angabe von D als Turingmaschine wäre aber auch möglich, wenn auch aufwendig.

Der Widerspruch entsteht dann, wenn wir fragen, ob diese Maschine D ihren eigenen Code akzeptiert.

Seien $A, B \subseteq \Sigma^*$.

Eine **Reduktion** von A auf B ist eine berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ mit der Eigenschaft:

$$w \in A \Leftrightarrow f(w) \in B \quad \text{für alle } w \in \Sigma^*$$

Notation: $A \leq B$ falls es eine Reduktion von A auf B gibt.

Eigenschaft:

- ▶ Ist B entscheidbar und $A \leq B$, dann ist auch A entscheidbar.
- ▶ Ist B semi-entscheidbar und $A \leq B$, dann ist auch A semi-entscheidbar.

Ist ein Problem A reduzierbar auf B , also $A \leq B$, dann heißt das anschaulich, dass A höchstens so schwierig ist wie B , weil man A mit Hilfe von B entscheiden kann.

Der Algorithmus, der A entscheidet, bildet eine Eingabe w mittels der Reduktion ab zu $f(w)$. Da B entscheidbar ist, gibt es einen Algorithmus, der $f(w) \in B$ entscheidet. Wegen der Eigenschaft der Reduktion f kann die Antwort unverändert übernommen werden, da $w \in A$ ist genau dann, wenn $f(w) \in B$ ist.

Das Leerheitsproblem

Das **Leerheitsproblem** ist die Sprache

$$E := \{ e ; M_e \text{ akzeptiert die leere Sprache } \emptyset \}$$

Turing-Maschinen

Unentscheidbarkeit

Codierung von

Turing-Maschinen

Das Halteproblem

ReduktionUnentscheidbare
Probleme

Theorem

Das Leerheitsproblem E ist unentscheidbar.

Beweis: Reduktion von H auf \bar{E} :

$f(e, w)$ ist der Code der folgenden DTM:

lösche die Eingabe und schreibe (e, w) auf das Band
lasse U laufen, d.h., simuliere M_e mit Eingabe w .

Nun gilt:

Ist $(e, w) \in H$, dann hält $M_{f(e,w)}$ immer, also ist $f(e, w) \notin E$.

Ist $(e, w) \notin H$, dann hält $M_{f(e,w)}$ nie, also ist $f(e, w) \in E$.

Das Leerheitsproblem ist also die Frage, ob eine eingegebene Turingmaschine für jeden Input nicht terminiert.

Die Anwendung zeigt, wie Reduktionen typischerweise zum Beweis der Unentscheidbarkeit benutzt wird: wir wissen bereits, dass H unentscheidbar ist. Da $H \leq E$ ist, würde aus der Entscheidbarkeit von E auch die Entscheidbarkeit von H folgen, also muss E auch unentscheidbar sein.

Theorem

Sind A und \bar{A} beide semi-entscheidbar, dann ist A entscheidbar.

Seien M und \bar{M} DTM, die A und \bar{A} akzeptieren.

Idee: Lasse M und \bar{M} parallel laufen!

Bei jedem Input w muss eine von beiden halten:

M hält bei $w \rightsquigarrow w \in A$

\bar{M} hält bei $w \rightsquigarrow w \notin A$.

Realisierung durch **präemptives Multitasking**:

Solange M und \bar{M} nicht gehalten haben

simuliere 10 Schritte von M

simuliere 10 Schritte von \bar{M}

Da Turingmaschinen nicht wirklich parallel laufen können, muss dies durch einen seriellen Ablauf simuliert werden, ähnlich wie Multitasking auf einem seriellen Rechner realisiert wird: durch Vergabe von Zeitscheiben, die die verschiedenen Tasks abwechselnd zugewiesen bekommen.

Nochmal das Leerheitsproblem

Theorem

Das Komplement des Leerheitsproblems \bar{E} ist semi-entscheidbar.

Beweis: mittels *dovetailing*.

Korollar

Das Leerheitsproblem E ist nicht semi-entscheidbar.

Beweis: Da \bar{E} semi-entscheidbar ist, wäre E sonst entscheidbar.

Dovetailing funktioniert ähnlich wie das parallele Abarbeiten von zwei Aufgaben oben, nur dass hier unendlich viele Tasks parallel abgearbeitet werden.

Ein Wort e ist in \bar{E} , wenn M_e für mindestens einen Input w hält. Das Semi-Entscheidungsverfahren für \bar{E} muss also ein w suchen, für das M_e bei input w hält. Dies wird dadurch realisiert, dass für immer mehr Inputs immer längere Zeitscheiben zugewiesen werden.

Sei w_0, w_1, w_2, \dots eine Aufzählung von $\{0, 1\}^*$. Dann ist der folgende Algorithmus ein Semi-Entscheidungsverfahren für \bar{E} , d.h. er terminiert bei Eingabe e genau dann, wenn $e \in \bar{E}$ ist.

```
input e ;
t := 1 ;
s := 0 ;
while s = 0 do
  t := t + 1 ;
  for i := 0 to t do
    if  $M_e$  hält bei input  $w_i$  nach höchstens  $t$  Schritten
      then s := 1 ;
  end;
end;
```