Kapitel 3: Berechnungstheorie

Gliederung

- Ziel: Wir wollen zeigen dass …
  - es „interessante“ algorithmische Entscheidungsprobleme gibt, die nicht Turing-berechenbar sind.
  - Unter der Annahme der Church‘schen These kann es dann auch kein Computerprogramm geben, welches das Problem löst.

Entscheidungsprobleme

- Beziehen sich jeweils auf ein Paar (X,Y) von unendlichen Mengen X und Y mit $Y \subseteq X$.
- Für jedes $x \in X$ ist die Frage zu beantworten, ob $x \in Y$ gilt.
- Jedes $x \in X$ sei als Zeichenkette cod(x) eindeutig beschreibbar; damit sind sie algorithmische Probleme.
- Gesucht ist ein Computerprogramm bzw. eine Turing-Maschine mit folgendem Verhalten:
  Für jedes $x \in X$ gibt es/sie bei Eingabe von cod(x) als Ergebnis aus:
  - 1, wenn $x \in Y$,
  - 0, wenn $x \notin Y$.

Wir haben uns schon mit zahlreichen Entscheidungsproblemen (L,Σ^*) befasst:
dem Wortproblem für eine Sprache L über dem Alphabet Σ,
z.B. wenn L durch einen Automaten oder eine Grammatik definiert war.
Dabei war cod(x) = x.

Erinnerung: Problemreduktion

- Seien (X_1,Y_1) und (X_2,Y_2) Entscheidungsprobleme
- und t eine vollständig definierte Funktion von X_1 auf X_2.
- Dann sagt man:

> t reduziert (X_1,Y_1) auf (X_2,Y_2), wenn für alle $x_1 \in X_1$ gilt:
>
> - wenn $x_1 \in Y_1$ gilt, so gilt $t(x_1) \in Y_2$
> - wenn $x_1 \notin Y_1$ gilt, so gilt $t(x_1) \notin Y_2$

- Man nennt (X_1,Y_1) auf (X_2,Y_2) reduzierbar, wenn ein solches t existiert.

Praktisch wichtig: Ist die Reduktionsabbildung t berechenbar?

berechenbare Problemreduktion

- Seien (X_1,Y_1) und (X_2,Y_2) Entscheidungsprobleme
- und t eine vollständig definierte Funktion von X_1 auf X_2.
- Dann sagt man:

> (X_1,Y_1) ist Turing-reduzierbar auf (X_2,Y_2),
> wenn es eine Funktion t gibt, die (X_1,Y_1) auf (X_2,Y_2) reduziert,
> sowie eine Turing-Maschine M, die t berechnet,
> d.h. M leistet für alle $x \in X_1$ folgendes:
>
> - M berechnet bei der Eingabe cod(x) die Ausgabe cod(t(x))

... mit der Funktion cod werden die Elemente aus X_1 und X_2 geeignet als Zeichenketten über einem Alphabet Σ kodiert

Anwendung des Reduktionsbegriffs

Der Reduktionsbegriff erlaubt es uns, die folgenden zwei Fragen miteinander in Beziehung zu setzen:

- Ist (X_2,Y_2) ein lösbares Entscheidungsproblem?
- Ist (X_1,Y_1) ein lösbares Entscheidungsproblem?

- Seien (X_1,Y_1) und (X_2,Y_2) Entscheidungsprobleme, so dass (X_1,Y_1) Turing-reduzierbar auf (X_2,Y_2) ist.

- Dann gilt die folgende Implikation:

> Wenn (X_2,Y_2) Turing-berechenbar ist, dann auch (X_1,Y_1).

- Also gilt auch deren Kontraposition:

> Wenn (X_1,Y_1) nicht Turing-berechenbar ist,
> dann ist auch (X_2,Y_2) nicht Turing-berechenbar.

Begründung

- Sei t eine Funktion, die (X_1,Y_1) auf (X_2,Y_2) reduziert und T eine Turing-Maschine, mit der t berechnet werden kann
- Sei M_2 eine Turing-Maschine, die das Entscheidungsproblem (X_2,Y_2) löst

- Eine Turing-Maschine M_1, die das Entscheidungsproblem (X_1,Y_1) löst, arbeitet wie folgt:

Eingabe: Zeichenkette $x_1 \in X_1$

1) Bestimme $cod(x_1)$
2) Benutzte die Turing-Maschine T, um daraus $cod(t(x_1))$ zu berechnen
3) Starte die Turing-Maschine M_2 mit der Eingabe $cod(t(x_1))$, warte bis M_2 den Endzustand erreicht hat und gib die von M_2 berechnete Ausgabe aus

Denksportaufgabe

- Es seien (X_1,Y_1) und (X_2,Y_2) nicht-triviale Entscheidungsprobleme (d.h. es gilt $\varnothing \neq Y_1 \neq X_1$ sowie $\varnothing \neq Y_2 \neq X_2$, so dass die Entscheidungen keine Konstanten sind.)

(X_1,Y_1)	(X_2,Y_2)	Wann ist (X_1,Y_1) Turing-reduzierbar auf (X_2,Y_2)?
lösbar*	lösbar	immer
lösbar	unlösbar	immer
unlösbar	lösbar	nie
unlösbar	unlösbar	vielleicht

*) „lösbar“: kurz für Turing-berechenbar

-- Wieso? --

Denksportaufgabe – Lösung

- (X_2,Y_2) nicht-trivial $\Rightarrow$ Man kann feste Werte $y_2 \in Y_2$, $z_2 \in X_2 \setminus Y_2$ wählen.
- (X_i,Y_i) wenn T.-entscheidbar, dann z.B. mit T.-Maschine M_i.

	(X_1,Y_1)	(X_2,Y_2)	(X_1,Y_1) T.-red.bar auf (X_2,Y_2)?
a)	lösbar	lösbar	immer
b)	lösbar	unlösbar	immer
c)	unlösbar	lösbar	nie
d)	unlösbar	unlösbar	vielleicht

- Wie funktioniert reduzierende T.-Maschine T von X_1 nach X_2?
  - a,b) Input x. Entscheide (X_1,Y_1) für x mit M_1.
    Wenn Ergebnis=1 ($x \in Y_1$), dann Output: y_2, sonst Output: z_2
  - c) Wäre (X_1,Y_1) T.-red.bar auf (X_2,Y_2), dann wäre ja (X_2,Y_2) lösbar.
  - d) T.-red.bar z.B. wenn $X_1=X_2$ und $Y_1=Y_2$ mittels T: Output:=Input.

Diskussion

- weiteres Vorgehen

  - Auf der Suche nach „interessanten“ unlösbaren Entscheidungsproblemen werden wir
    - für ein Entscheidungsproblem direkt zeigen, dass es unlösbar ist;
    - mittels Reduktion für andere Entscheidungsprobleme nachweisen, dass sie auch unlösbar sind.
  - Unser **erstes** Entscheidungsproblem,
    - dessen Unlösbarkeit wir direkt zeigen,
    - und von dem aus wir auf die Unlösbarkeit vieler anderer Entscheidungsprobleme schließen können,

    wird ein **Halteproblem** sein.

Das Halteproblem

Inhaltlich geht es um folgendes algorithmisches Problem:

gegeben:
- eine Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{ 0,1 \}$
- eine Eingabe $w \in \Sigma^*$

gesucht:
- die Antwort auf die Frage,
  ob M bei Eingabe von w den Endzustand erreicht

Anmerkungen

- Um nachzuweisen, dass das Halteproblem unlösbar ist, genügt es, folgende Variante des Halteproblems zu betrachten:

Das eingeschränkte Halteproblem – Vorschau

- Inhaltlich geht es um folgendes algorithmisches Problem:

gegeben:	• eine Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{ 0,1 \}$ und ein **speziell** für M **ausgewähltes** $w \in \Sigma^*$ (Details zur Wahl von w folgen noch.)
gesucht:	• die Antwort auf die Frage, ob M bei Eingabe dieses w den Endzustand erreicht

Das eingeschränkte Halteproblem (cont.)

- Um zu zeigen, dass das eingeschränkte Halteproblem unlösbar ist, benutzen wir, dass man alle Turing-Maschinen mit dem Eingabe-alphabet $\Sigma = \{ 0,1 \}$ effektiv aufzählen kann:
  - Man verwendet zur Beschreibung aller solcher Turing-Maschinen ein festes Alphabet A und eine Grammatik, produziert z.B. alle Wörter über A in aufsteigender Länge und lässt dabei alle weg, die keine Turing-Maschine beschreiben. Das macht man, bis man die i-te Turing-Maschine hat.
  - Der Aufzählungs-Algorithmus liefert also für die Eingabe i die Ausgabe M_i.
  - M_0, M_1, M_2, ... umfasst alle Turing-Maschinen über A.

Das eingeschränkte Halteproblem (cont.)

- Das eingeschränkte Halteproblem kann man nun folgendermaßen als Entscheidungsproblem (X_H, Y_H) formulieren:
  - X_H ist die Menge aller natürlichen Zahlen einschl. 0.
  - Y_H ist die Menge aller $i \in N_0$ für die gilt, dass die Turing-Maschine M_i bei Eingabe der Zeichenkette bin(i) den Endzustand erreicht.

Das Hauptergebnis

- Es gilt:

> Das eingeschränkte Halteproblem (X_H, Y_H) ist unlösbar.

... folglich ist das Halteproblem ebenfalls eine unlösbares Entscheidungsproblem

Beweisidee

- Wir nehmen an, dass es eine Turing-Maschine M_H gibt, die das eingeschränkte Halteproblem löst.
- Die Turing-Maschine M_H leistet also für alle $i \in N_0$ folgendes:
  - Wenn die Turing-Maschine M_i bei Eingabe von bin(i) den Endzustand **erreicht**, so bestimmt M_H bei Eingabe von bin(i) die Ausgabe **1**;
  - wenn die Turing-Maschine M_i bei Eingabe von bin(i) den Endzustand **nicht erreicht**, so bestimmt M_H bei Eingabe von bin(i) die Ausgabe **0**.

Beweisidee (cont.)

- Wir „bauen“ nun eine Turing-Maschine M, die wie folgt arbeitet:

Eingabe: Binärzahl w

1) Starte die Turing-Maschine M_H mit der Eingabe w und warte bis M_H einen Endzustand erreicht.

   (Dieses Ereignis muss eintreten.)

2) Prüfe, welche Ausgabe M_H berechnet hat.

   a) Falls M_H die Ausgabe 1 berechnet hat, so gehe in eine Endlosschleif e (d.h. M erreicht den Endzustand nicht).

   b) Falls M_H die Ausgabe 0 berechnet hat, so gehe in den Endzustand.

Das ist eine lösbare praktische Programmieraufgabe.

Beweisidee (cont.)

- Da die Turing-Maschine M aber eine der aufgezählten Maschinen ist, hat sie eine Nummer: M = M_j . Und damit gilt:
  - Wenn M_j bei Eingabe von bin(j) den Endzustand erreicht, so bestimmt M_H bei Eingabe von bin(j) die Ausgabe 0,

    … denn so haben wir M gebaut: Befehl 2b!
  - Wenn M_j bei Eingabe von bin(j) den Endzustand erreicht, so bestimmt M_H bei Eingabe von bin(j) die Ausgabe 1,

    … denn das war ja die Aufgabe von M_H!
  - Beides zusammen kann aber nicht sein. Also kann M_j bei Eingabe von bin(j) den Endzustand nicht erreichen, sonst würde ein Widerspruch gelten.

Beweisidee (cont.)

- Da die Turing-Maschine M eine der aufgezählten Maschinen war, hatte sie auch eine Nummer: $M = M_j$.
- Wir haben gerade gefolgert:
  M_j kann bei Eingabe von bin(j) den Endzustand nicht erreichen.
- Daraus folgt nun weiter:
  - M_H bestimmt bei Eingabe von bin(j) die Ausgabe 0,

    … denn das war ja die Aufgabe von M_H!
  - M_H bestimmt bei Eingabe von bin(j) die Ausgabe 1,

    … denn so haben wir M gebaut: Befehl 2a!
- Das kann aber auch nicht sein. Also kann M_j kann bei Eingabe von bin(j) den Endzustand weder erreichen noch nicht erreichen.
- Das kann aber auch nicht sein. Also kann es die eingeschränkte Halteproblem lösende Turing-Maschine M_H nicht geben.

Anmerkungen

Beim Beweis, dass das eingeschränkte Halteproblem unlösbar ist, haben wir folgende Eigenschaften des Berechnungsmodells Turing-Maschine ausgenutzt

- Turing-Maschinen lassen sich endlich beschreiben
- Turing-Maschinen können andere Turing-Maschinen als „Unterprogramme“ benutzen

... diese Eigenschaften hat jedes „leistungsfähigste“ Berechnungsmodell

Plan

- Wir werden jetzt die Tatsache,
  dass das eingeschränkte Halteproblem unlösbar ist,
  dazu verwenden, zu zeigen, dass
  es weitere „interessante“ unlösbare Entscheidungsprobleme gibt.

- Ein weiteres unlösbare Entscheidungsprobleme kennen wir natürlich jetzt schon:
  Auch das (uneingeschränkte) Halteproblem ist unlösbar, denn das verlangt ja viel mehr als das eingeschränkte.

- Aber es gibt noch (unglaublich) viele mehr …

Anmerkungen

- Wir konzentrieren uns im Folgenden auf Entscheidungsprobleme, die Fragestellungen adressieren, die aus der Perspektive eines Softwareentwicklers von Bedeutung sind

- Dabei werden wir verstehen lernen, weshalb uns Entwicklungsumgebungen bei einigen Aufgaben so wenig unterstützen, etwa dabei, zu überprüfen, ob ein von uns geschriebenes Programm
  - für bestimmte Eingaben überhaupt einen bzw. den korrekten Rückgabewert bestimmt
  - ein gewünschtes Eingabe/Ausgabe-Verhalten hat
  - ein anderes Programm „gleichwertig“ ersetzen kann
  - ...

Ein Beispielentscheidungsproblem

Beispiel

- Das folgende Beispiel dient nur zur Illustration der zentralen Beweisidee.
- Sei B eine Turing-Maschine, die eine vollständig definierte Funktion über Σ^* mit $\Sigma = \{ 0,1 \}$ berechnet.
- Wir betrachteten das algorithmische Problem:

gegeben:	• eine Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{ 0,1 \}$
gesucht:	• die Antwort auf die Frage, ob M dieselbe Funktion über Σ^* wie B berechnet

Vorbereitung

- Um zu zeigen, dass unser Beispielproblem unlösbar ist, beschreiben wir es zunächst unter Benutzung der bei der Beschreibung des eingeschränkten Halteproblems verwendeten Begriffe:
- Es geht um das Entscheidungsproblem (X_B, Y_B) mit …
  - X_B ist die Menge aller natürlichen Zahlen
  - Y_B ist die Menge aller $i \in N_0$ für die gilt, dass die Turing-Maschine M_i dieselbe Funktion wie die Turing-Maschine B berechnet

- Die zugehörigen Ergebnisse

  - Es gilt

    Das eingeschränkte Halteproblem (X_H, Y_H) ist Turing-reduzierbar auf unser Beispiel-Entscheidungsproblem (X_B, Y_B) .

  - ... und damit gilt auch

    Das Beispiel-Entscheidungsproblem (X_B, Y_B) ist unlösbar.

  - ... denn wäre das Beispiel-Entscheidungsproblem lösbar, dann wäre auch das eingeschränkte Halteproblem lösbar.

… und diese Schlussweise kann man noch häufiger anwenden.

Beweisidee

- Wir werden eine Reduktionsfunktion t angeben, die jedem $i \in X_H$ ein „passendes“ $j \in X_B$ zuordnet, und begründen, dass diese Funktion t Turing-berechenbar ist
- „passend“ bedeutet, dass wir jeder Turing-Maschine M_i so eine Turing-Maschine M_j zuordnen, dass $t(Y_H) \subseteq (Y_B)$ gilt, d.h. …
  - wenn M_i bei Eingabe von bin(i) den Endzustand erreicht, so berechnet M_j dieselbe Funktion wie die Turing-Maschine B;
  - wenn M_i bei Eingabe von bin(i) nicht den Endzustand erreicht, so berechnet M_j die nirgends definierte Funktion (und damit offenbar eine andere Funktion als die Turing-Maschine B).

Beweisidee (cont.)

- Die der gegebenen Turing-Maschine M_i zugeordnete Turing-Maschine M_j wird so definiert, dass sie wie folgt arbeitet:

Eingabe: Zeichenkette $w \in \Sigma^*$

1) Starte die gegebene Turing-Maschine M_i mit der Eingabe bin(i) und warte bis M_i einen Endzustand erreicht (Dieses Ereignis muss nicht immer eintreten; ggf. endet der Programmlauf also nicht.)
2) Falls M_i bei Eingabe bin(i) den Endzustand erreicht hat, starte die Turing-Maschine B mit der Eingabe w, warte bis B den Endzustand erreicht hat und gibt die von B bestimmte Ausgabe aus.

... offenbar leistet M_j das Gewünschte

Beweisidee (cont.)

- Um uns davon zu überzeugen, dass die Funktion t Turing-berechenbar ist, sind folgende Argumente wichtig:
  - Anhand einer gegebenen Zahl i kann man effektiv bin(i) und das Programm der Turing-Maschine M_i bestimmen.
  - Anhand von bin(i) und des Programms von M_i kann man effektiv ein Programm für die der Turing-Maschine M_i zugeordnete Turing-Maschine M_j bestimmen.
  - Anhand des Programms der Turing-Maschine M_j kann man effektiv die der Zahl i zugeordnete Zahl j bestimmen.

Wir nutzen jeweils aus, dass M_0, M_1, M_2, ... eine effektive Aufzählung aller Turing-Maschinen mit dem Eingabealphabet $\Sigma = \{ 0,1 \}$ ist

Die betrachteten algorithmischen Probleme

- Wir interessieren uns für folgende Entscheidungsprobleme:

gegeben:	• eine Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{ 0,1 \}$ • eine denkbare nicht-triviale Eigenschaft einer Funktion
gesucht:	• die Antwort auf die Frage, ob die Funktion, die M berechnet, diese Eigenschaft hat

nicht-triviale funktionale Eigenschaften

- Nicht-triviale funktionale Eigenschaften von Turing-Maschinen haben etwas mit dem Eingabe/Ausgabe-Verhalten einer Turing-Maschine zu tun, beispielsweise …
  - Gibt M bei Eingabe einer bestimmten Zeichenkette v eine bestimmte Ausgabe f(v)=w aus?
  - Ist die von M berechnete Funktion vollständig definiert?
  - Ist die von M berechnete Funktion überhaupt irgendwo definiert?
  - Kommt bei irgendeinem Input das Ergebnis 0 heraus?
  - Ist die von M berechnete Funktion f streng monoton wachsend, d.h. wenn zwischen zwei Binärzahlen m<n gilt, gilt dann auch f(m)<f(n)?

- nicht-triviale funktionale Eigenschaften (cont.)
  - E ist eine nicht-triviale Eigenschaft, falls
    - es mindestens eine Turing-Maschine mit der Eigenschaft E gibt,
    - es mindestens eine Turing-Maschine gibt, die die Eigenschaft E nicht hat.
  - E ist eine funktionale Eigenschaft, falls für alle Turing-Maschinen M und M‘ gilt:
    - Wenn M und M‘ dasselbe Eingabe/Ausgabe-Verhalten haben, dann haben entweder beide die Eigenschaft E oder beide haben die Eigenschaft E nicht.

Das zentrale Ergebnis (Vorbereitung)

- Sei E eine interessierende nicht-triviale funktionale Eigenschaft von Turing-Maschinen.
- Dann betrachten wir das folgende Entscheidungsproblem:
  - X_E ist die Menge aller natürlichen Zahlen
  - Y_E ist die Menge aller $i \in N_0$, für die gilt, dass die Turing-Maschine M_i die nicht-triviale funktionale Eigenschaft E hat

Das zentrale Ergebnis (Satz von Rice)

- Es gilt mit Blick auf jede nicht-triviale funktionale Eigenschaft E

> Das eingeschränkte Halteproblem (X_H, Y_H) ist Turing-reduzierbar auf das Entscheidungsproblem (X_E, Y_E) .

- ... und damit gilt auch wieder

> Das Entscheidungsproblem (X_E, Y_E) ist unlösbar.

Hart aber wahr:
Keine einzige interessante funktionale Eigenschaft eines Algorithmus ist algorithmisch überprüfbar.

Tröstlich:
Da Programme (meist) mit bestimmten Zielen zweckdienlich geschrieben werden, sind i.d.R. manuelle Beweise möglich.

Beweisskizze für den Satz von Rice

- … zur Vereinfachung nur in dem Fall, dass die Turing-Maschine, die die nirgends definierte Funktion berechnet, nicht die Eigenschaft E hat*.
- Um zu zeigen, dass das eingeschränkte Halteproblem (X_H,Y_H) Turing-reduzierbar auf das Entscheidungsproblem (X_E,Y_E) ist, gehen wir genauso wie bei unserem Beispielentscheidungsproblem vor.
- Man wählt sich eine Turing-Maschine B, die die Eigenschaft E hat, und ordnet jeder Turing-Maschine M_i so eine Turing-Maschine M_j zu (wieder im Stile des Anhängens von B an M_i), dass gilt:
  - Wenn M_i bei Eingabe von bin(i) den Endzustand erreicht, so berechnet M_j dieselbe Funktion wie die Turing-Maschine B;
  - wenn M_i bei Eingabe von bin(i) nicht den Endzustand erreicht, so berechnet M_j die nirgends definierte Funktion.

Genau hier benutzen wir, dass die Turing-Maschine, die die nirgends definierte Funktion berechnet, nicht die Eigenschaft E hat.

Anmerkungen

- Der Satz von Rice gilt mit Blick auf jedes „leistungsfähigste“ Berechnungsmodell.
- Also ist auch das folgende algorithmische Problem unlösbar:

gegeben:
- eine C++-Programm bzw. ein Java-Programm P
- eine nicht-triviale funktionale Eigenschaft E

gesucht:
- die Antwort auf die Frage, ob P eine Funktion berechnet, die die Eigenschaft E hat