051013 VO Theoretische Informatik

Berechenbarkeit

Ekaterina Fokina

Berechenbarkeitstheorie

Ziele der Berechenbarkeitstheorie (auch Rekursionstheorie):

- Den "Algorithmus" Begriff schärfen.
- Die theoretischen Grenzen des mit einem Computer machbaren erkunden.
- Eine Methodik bereitstellen, die es erlaubt Probleme, die nicht mit einem Computer gelöst werden können, als solche zu klassifizieren.

Literatur

Jens Gallenbacher

Abenteuer Informatik. Kapitel 12, Allmächtiger Computer!?

Springer Spektrum; ISBN 978-3-8274-2965-0

John E. Hopcroft, Rajeev Motwani,Jeffrey D. Ullman

Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit. Kapitel 9, Unentscheidbarkeit.

Springer Spektrum; ISBN 978-3-8274-2965-0

Uwe Schöning

Theoretische Informatik - kurz gefasst. Kapitel 2, Berechenbarkeitstheorie

Spektrum Akademischer Verlag; ISBN: 978-3-8274-1824-1

Gottfried Vossen, Kurt-Ulrich Witt

Grundkurs Theoretische Informatik: Eine Anwendungsbezogene Einführung. [Kapitel 9]

Pearson Studium; ISBN 978-3-8689-4082-4

Subsection 1

Berechenbarkeit - Informelle Einleitung

Berechenbarkeit

Beim Programmieren begegnen uns **Probleme** die wir

- mit einem effizienten Programm/Algorithmus lösen können,
- mit weniger effizienten Programmen/Algorithmus lösen können, oder
- nicht lösen können.

Effizienz: Die Laufzeit wächst nur langsam mit der Größe der Eingabe.

Fragen:

- Kann der Computer das Problem nicht lösen oder haben wir das richtige Programm noch nicht gefunden?
- Gibt es Probleme, die mit keinem Computer gelöst werden können?
- Wie können wir unlösbare Probleme erkennen?

Berechenbarkeit - Entscheidbarkeit

Berechenbarkeit: Wir betrachten eine Fragestellung, bei der es zu jeder Eingabe eine wohldefinierte Ausgabe gibt. Gibt es ein Programm, das für jede Eingabe nach endlich vielen Schritten terminiert und die richtige Ausgabe hat, sagen wir die Fragestellung (das Problem) ist berechenbar, andernfalls nicht berechenbar.

Entscheidbarkeit: Betrachtet man eine Fragestellung, die mit ja/nein beantwortet werden kann, spricht man von einem Entscheidungsproblem. Gibt es ein Programm, das immer die richtige Antwort berechnet, sagen wir das Problem ist entscheidbar, andernfalls unentscheidbar.

Entscheidbarkeit ist ein Spezialfall der Berechenbarkeit.

Gibt es Probleme die Computer nicht lösen können?

- Von Computerprogrammen erwarten wir, dass sie nach endlich vielen Schritten terminieren (und nicht in einer endlosen Schleife festhängen).

Programm 1:

```
i=1;
while(i==1) {
   i++;
}
```

Programm 2:

```
i=1;
while(i>0) {
   i++;
}
```

- Es wäre praktisch ein Programm zu haben, das testet, ob ein beliebiges Programm terminiert.

Halteproblem

Gibt es ein Computerprogramm, das für jedes beliebige Programm entscheidet, ob es terminiert?

Gibt es Probleme, die Computer nicht lösen können?

Halteproblem

Gibt es ein Computerprogramm, das für jedes beliebige Programm entscheidet, ob es terminiert?

Wir zeigen, dass es kein solches Programm geben kann.

Beweisskizze: Beweis durch Widerspruch:

Nehmen wir an, wir hätten ein Programm `HALT(Programm S)`, das

- für jedes beliebige Programm `S`
- die Antwort `ja` liefert, wenn `S` terminiert, und
- die Antwort `nein` liefert, wenn `S` nicht terminiert.

Wir haben also

- `HALT(Programm 1)= ja`
- `HALT(Programm 2)= nein`

Gibt es Probleme, die Computer nicht lösen können?

Halteproblem

Gibt es ein Computerprogramm, das für jedes beliebige Programm entscheidet, ob es terminiert?

Wir betrachten das folgende Programm:
Programm 3:

```
while(HALT(Programm 3)) {
    print "THI";
}
```

Es gibt zwei Möglichkeiten:

- **Programm 3 terminiert**: Dann ist aber die while-Bedigung erfüllt und das Programm terminiert nicht. ↯.
- **Programm 3 terminiert nicht**: Dann ist die while-Bedigung nicht erfüllt und das Programm terminiert. ↯.

Es kann also keine solche Funktion `HALT(Programm S)` geben.
↪ Das Halteproblem ist unentscheidbar.

Gibt es Probleme, die Computer nicht lösen können?

- Das Halteproblem ist eines der bekanntesten unentscheidbaren Probleme.
- Es gibt unentscheidbare Probleme.
  $\hookrightarrow$ Computer können nicht alles.
- Es gibt kein Programm, das für alle Programme entscheiden kann, ob diese terminieren.
  - Terminierung ist eine sehr fundamentale Programmeigenschaft.
    $\hookrightarrow$ schlechte Nachricht für die automatisierte Verifikation von Programmeigenschaften.
  - Es gibt aber Systeme, die für viele Programme entscheiden können, ob diese terminieren.

Subsection 2

Berechenbarkeitstheorie

Berechenbarkeitstheorie

Um zu zeigen, dass ein Problem entscheidbar ist, benötigen wir zunächst ein formales Modell für Computer / Berechenbarkeit.

Wir haben bereits einige solche Modelle kennen gelernt:

- endliche Automaten
- Kellerautomaten
- Linear beschränkte Turingmaschine
- Turingmaschinen

Da die Turingmaschine das mächtigste von diesen Modellen ist, fällt unsere Wahl auf diese.

Turing-Berechenbarkeit

Wir verwenden **Turing-Maschinen** als formales Modell für Computer / Berechenbarkeit:

Definition

Eine Funktion $f : \Sigma^* \Rightarrow \Sigma^*$ heißt **(Turing-)berechenbar**, wenn es eine Turingmaschine $M = (Q, \Sigma, \Gamma, b, \delta, q_0, F)$ gibt, die f berechnet, d.h. für die $f(x) = f_M(x)$ für alle $x \in \Sigma^*$ gilt.

- Die Maschine muss für **alle** Eingaben einen Endzustand erreichen.
- Wenn die Maschine terminiert, steht $f(x)$ auf dem Band.

Zu jeder nichtdet. TM gibt es eine äquivalente det. TM.

$\hookrightarrow$ Wir können uns auf deterministische Maschinen beschränken.

Andere Berechenbarkeitsmodelle

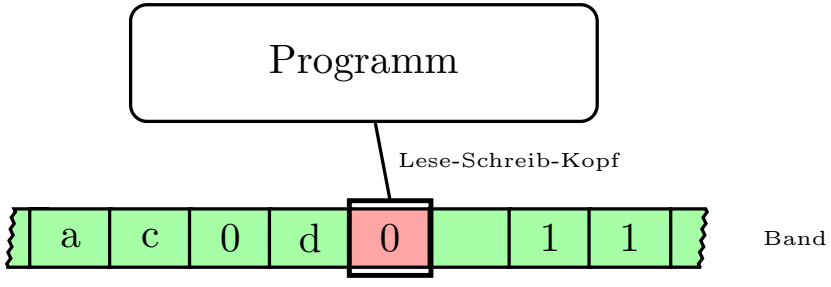
Mehrband-Turingmaschine

Die **Mehrband-Turingmaschine** ist eine Erweiterung der klassischen Turingmaschine und

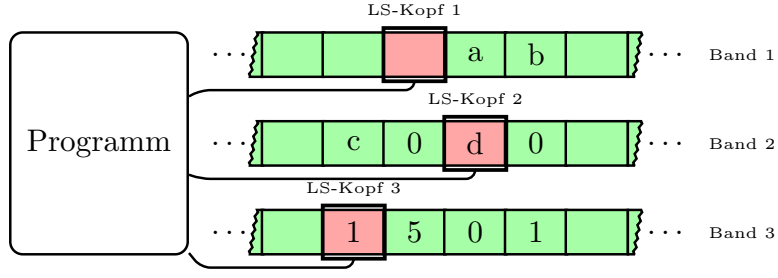
- verfügt über mehrere Speicherbänder
- die jeweils einen eigenen Lese- und Schreibkopf haben
- die unabhängig voneinander bewegt werden können.

Ansonsten verhält sich eine Mehrband-Turingmaschine genau so wie eine klassische Turingmaschine.

Klassische Turingmaschine



Mehrband-Turingmaschine



Andere Berechenbarkeitsmodelle

While-Programme

Wir betrachten eine simple Programmiersprache mit

- Variablen $x_0, x_1, x_2, \ldots$, und
- Konstanten $0, 1, 2, \ldots$.

While-Programme:

- Wertzuweisungen $x_i := x_j + c$ und $x_i := x_j - c$ sind While-Programme
- Wenn P_1, P_2 While-Programme sind dann auch
  - $P_1; P_2$
  - `LOOP` x_i `DO` P_1 `END`
  - `WHILE` $x_i \neq 0$ `DO` P_1 `END`

Andere Berechenbarkeitsmodelle

GOTO-Programme

GOTO Programme bestehen aus einer Sequenz von Anweisungen A_i, die jeweils durch eine Marke M_i eingeleitet werden.

$$M_1 : A_1; M_2 : A_2; \ldots M_k : A_k$$

Mögliche Anweisungen A_k:

- Wertzuweisungen $x_i := x_j + c$ und $x_i := x_j - c$
- unbedingter Sprung: `GOTO` M_i
- bedingter Sprung: `IF` $x_i = c$ `GOTO` M_i
- Stop: `HALT`

Andere Berechenbarkeitsmodelle

Weitere

Viele weitere Berechenbarkeitsmodelle

- μ-Rekursion
- Zählermaschinen mit mind. 2 Zählern
- Zweikellerautomat
- ...

Diese Modelle sind äquivalent zu Turingmaschinen:

- Jede Turing-berechenbare Funktion ist auch in den anderen Modellen berechenbar.
- Jede Funktion, die in einem der anderen Modelle berechenbar ist, ist auch Turing-berechenbar.

Turing-Berechenbarkeit

Turing-Berechenbarkeit stimmt mit anderen Berechenbarkeitsmodellen überein

- Eine Funktion ist Turing-berechenbar genau dann, wenn sie in den anderen Modellen berechenbar ist.

Church-Turing's These

Die Menge der Turing-berechenbaren Funktionen ist genau die Menge der intuitiv berechenbaren Funktionen.

Turing-Vollständigkeit

Eine Programmiersprache ist Turing-vollständig, wenn sie alle Turing-berechenbaren Funktionen implementieren kann.

Gängige Programmiersprachen sind Turing-vollständig: C, C++, Java, Prolog, ...

Entscheidbarkeit

- **Berechenbarkeit von Problemen**: Berechenbarkeit der Lösung
- **Berechenbarkeit von Mengen**: Berechenbarkeit, ob ein Element Teil einer Menge ist – man spricht dann von Entscheidbarkeit.

Die Entscheidbarkeit von Mengen kann auf die Berechenbarkeit von Funktionen zurückgeführt werden.

Definition

Sei Σ ein Alphabet. Eine Menge $L \subseteq \Sigma^*$ heißt **entscheidbar**, falls die charakteristische Funktion von L, $\chi_L : \Sigma^* \to \{0, 1\}$, definiert durch

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

berechenbar ist.

Nicht berechenbare Funktionen

Frage: Gibt es (wohldefinierte) Probleme, die nicht berechenbar sind, d.h. Probleme, die mit keinem Computer gelöst werden können?

Antwort: Ja.

Ein Kardinalitätsargument

- Die Menge aller (Turing-)berechenbarer Funktionen über $\mathbb{N}$ ist abzählbar unendlich.
  - Jede Turingmaschine mit $\Sigma = \{1\}$ ist äquivalent zu einer Turingmaschine mit $\Sigma = \{1\}$ und $\Gamma = \{0, 1\}$.
  - Fixiert man Γ und Σ dann ist die Menge aller Turingmaschinen mit Γ und Σ abzählbar unendlich.
- Es gibt überabzählbar viele Funktionen von $\mathbb{N}$ nach $\mathbb{N}$.
- Es gibt überabzählbar viele Teilmengen der natürlichen Zahlen.

$\Rightarrow$ Es gibt nicht-berechenbare Funktionen und unentscheidbare Mengen.

Subsection 3

Turing's Halteproblem

Turing's Halteproblem

Wir kennen aber auch konkrete unentscheidbare Probleme:

Turing's Halteproblem

Gegeben: Eine beliebige Turingmaschine mit einer beliebigen Eingabe.
Frage: Hält die Maschine mit dieser Eingabe?

Turing hat gezeigt: Es gibt keinen Algorithmus, der das Halteproblem löst, d.h. das Halteproblem ist unentscheidbar.

Praktische Konsequenz: Es ist nicht entscheidbar, ob ein beliebiges Programm mit einer beliebigen Eingabe terminiert oder nicht!
(Für Spezialfälle ist eine Entscheidbarkeit durchaus möglich.)

Turing's Halteproblem

Wir wollen beweisen, dass es keine Turingmaschine gibt, die das Halteproblem löst.

1.Schritt Wir kodieren Turingmaschinen als Wörter über $\{0, 1\}$.

Wir betrachten Turingmaschinen mit

- Zuständen $Q = \{q_1, \ldots, q_k\}$ für ein k;
- Startzustand q_1;
- einem einzigen Endzustand q_2 ;
- Eingabealphabet $\Sigma = \{0, 1\}$;
- Blank b;
- Bandalphabet $\Gamma = \{X_1, X_2, \ldots, X_m\}$ für ein m;
- $X_1 = 0$, $X_2 = 1$ und $X_3 = b$
- Bewegungsrichtungen des LS-Kopfs: $D_1 = L$, $D_2 = R$, $D_3 = N$

Turing's Halteproblem - Binärkodierung von TM

Mit den Annahmen über die Turingmaschine genügt es die Überführungsfunktion δ zu kodieren.

- Zustände und Bandsymbole, die nicht in δ vorkommen, können vernachlässigt werden

Ein Übergang $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ wird als

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

kodiert.

Die Funktion δ wird durch die Hintereinanderreihung der Kodierungen ihre Übergänge kodiert, wobei die Kodierungen durch "11" getrennt sind.

Wenn $C_1 \ldots C_n$ die Kodierungen der einzelnen Übergänge von δ sind, dann wird δ wie folgt kodiert.

$$C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$$

Turing's Halteproblem - Binärkodierung von TM

Beispiel

Betrachte die Turingmaschine

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, b\}, b, \delta, q_1, \{q_2\})$$

mit $\delta(q_1, 1) = (q_3, 0, R)$, $\delta(q_3, 0) = (q_1, 1, R)$,
$\delta(q_3, 1) = (q_2, 0, R)$, $\delta(q_3, b) = (q_3, 1, L)$.

- $\delta(q_1, 1) = (q_3, 0, R)$ wird als 0100100010100 kodiert
- $\delta(q_3, 0) = (q_1, 1, R)$ wird als 0001010100100 kodiert
- $\delta(q_3, 1) = (q_2, 0, R)$ wird als 00010010010100 kodiert
- $\delta(q_3, b) = (q_3, 1, L)$ wird als 0001000100010010 kodiert

δ kann man wie folgt kodieren:

0100100010100110001010100100110001001001010011000100010010

Auch in einer anderen Reihenfolge:

0001000100010010110100100010100110001010100100110001001001010

Turing's Halteproblem - Binärkodierung von TM

- Wir können also Turingmaschinen als Binärzahlen kodieren.
- Die Kodierung ist aber nicht eindeutig.
- Nicht jede Binärzahl ist die Kodierung einer Turingmaschine.

Wir definieren die Turingmaschine

$$\hat{M} = (\{q_1, q_2\}, \{0, 1\}, \{0, 1, b\}, b, \delta, q_1, \{q_2\})$$

mit $\delta(q_1, 0) = (q_2, 0, N)$, $\delta(q_1, 1) = (q_2, 1, N)$ und $\delta(q_1, b) = (q_2, b, N)$.

Wir ordnen jeder Binärzahl w eine TM M_w zu:

- Falls eine Binärzahl w Kodierung einer TM M ist: $M_w = M$
- Falls eine Binärzahl w keine Kodierung einer TM ist: $M_w = \hat{M}$

Turing's Halteproblem - Das spezielle Halteproblem

2.Schritt: Betrachte das spezielle Halteproblem

Definition

Unter dem speziellen Halteproblem verstehen wir die Sprache / Menge

$$K = \{w \in \{0,1\}^* \mid M_w \text{ angewendet auf } w \text{ hält}\}$$

3.Schritt: Beweisen, dass das spezielle Halteproblem unentscheidbar ist.

Theorem

Das spezielle Halteproblem ist unentscheidbar.

Turing's Halteproblem - Das spezielle Halteproblem

Theorem

Das spezielle Halteproblem ist unentscheidbar.

Beweis durch Widerspruch:
Wenn K entscheidbar wäre, dann gebe es eine TM M die K entscheidet.

Wir können M umbauen zu einer TM M' die

- Zuerst M ausführt
- Wenn M in den Endzustand wechselt, beginnt M' eine neue Berechung.
  - Wenn am Band 0 ($w \notin K$) steht geht M' in ihren Endzustand.
  - Wenn am Band 1 ($w \in K$) steht startet M' eine Endlosschleife.

  Der Endzustand von M ist kein Endzustand von M'.

Sei w' das Codewort von M' sodass $M_{w'} = M'$.

Turing's Halteproblem - Das spezielle Halteproblem

Theorem

Das spezielle Halteproblem ist unentscheidbar.

Beweis durch Widerspruch Teil 2:
Sei w' das Codewort von M' sodass $M_{w'} = M'$.

Wir wenden M' auf w' an:

$$\begin{aligned} M' \text{ mit Eingabe } w' \text{ hält} &\Leftrightarrow M \text{ mit Eingabe } w' \text{ gibt 0 aus} \\ &\Leftrightarrow w' \notin K \\ &\Leftrightarrow M_{w'} \text{ mit Eingabe } w' \text{ hält nicht} \\ &\Leftrightarrow M' \text{ mit Eingabe } w' \text{ hält nicht } \lightning \end{aligned}$$

Da wir einen Widerspruch erhalten haben, kann es keine solche TM M geben.
$\hookrightarrow$ Das spezielle Halteproblem ist unentscheidbar.

Subsection 4

Unentscheidbare Probleme & Reduktionen

Anmerkung zur Entscheidbarkeit

- **Eindruck:** Die meisten Probleme erscheinen entscheidbar.
- **Gründe:**
  - Selektive Sicht.
  - Man betrachtet zumeist einfache, gut strukturierte Probleme, und diese sind tatsächlich häufig entscheidbar.
- In "Wirklichkeit" sind die meisten Probleme unentscheidbar (überabzählbar viele).
- Auch viele interessante Probleme sind unentscheidbar.

Post's Korrespondenzproblem PKP

Benannt nach Emil L. Post (1897-1954).

Gegeben:

- Alphabet Σ
- zwei Listen von Zeichenketten über Σ:

$$A = (x_1, \ldots, x_k)$$
$$B = (y_1, \ldots, y_k)$$

Es gilt also $x_i, y_i \in \Sigma^+$.

Frage: Gibt es eine nicht-leere Folge von Indizes $i_1, \ldots, i_m$, sodass

$$x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}?$$

Beispiel

$A = (1, 10111, 10)$ und $B = (111, 10, 0)$.
Für $m = 4$ gibt es eine Lösung mit Indexfolge $(2, 1, 1, 3)$ so dass
$x_2 x_1 x_1 x_3 = y_2 y_1 y_1 y_3 = 101111110$.

Post's Korrespondenzproblem PKP

Beispiel

Für $A = (10, 011, 101)$ und $B = (101, 11, 011)$ gibt es keine Lösung.

Beispiel

Die kürzeste Lösung für $A = (001, 01, 01, 10)$ und $B = (0, 011, 101, 001)$ hat $m = 66$.
(Das beweist man indem man alle Folgen für $m \leq 66$ testet)

Satz

Post's Korrespondenzproblem ist unentscheidbar.

Das Post's Korrespondenzproblem ist sehr hilfreich, um die Unentscheidbarkeit anderer Probleme zu beweisen.

Methodik zum Beweisen der Unentscheidbarkeit

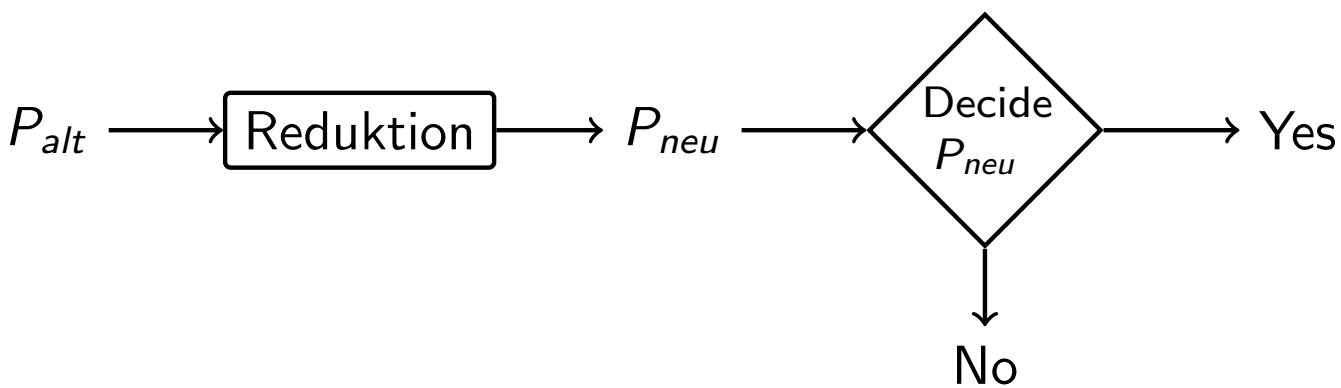
Frage: Ist ein neues Problem P_{neu} entscheidbar?

Bekannt: Es gibt ein Problem P_{alt}, das nicht entscheidbar ist und ähnlich dem Problem P_{neu} ist.

Beweisführung (Idee)

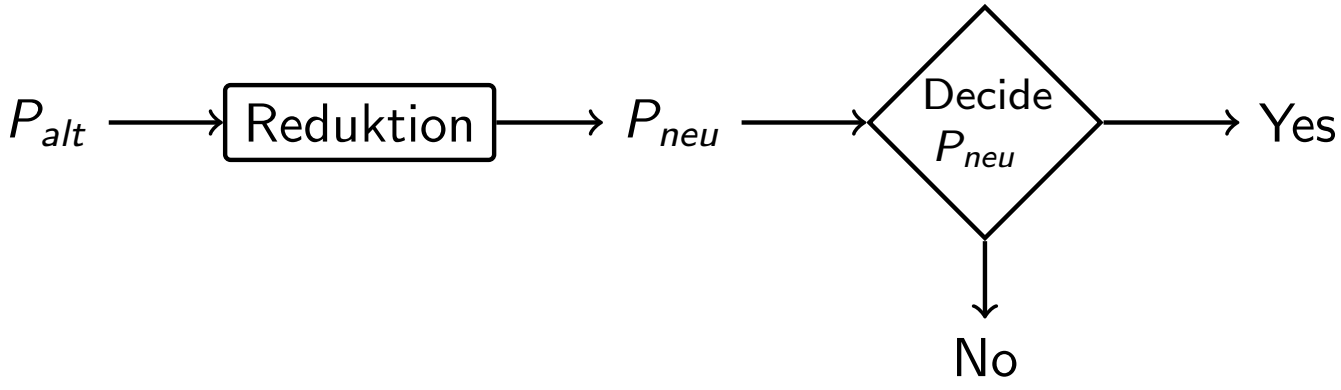
Reduktion und Widerspruch:

- Hat man eine Reduktion des Problems P_{alt} in das Problem P_{neu} und
- nimmt an dass ein Algorithmus Decide P_{neu} entscheidet,
- dann kann man einen Algorithmus für P_{alt} wie folgt konstruieren:



Widerspruch zu P_{alt} unentscheidbar.

Methodik zum Beweisen der Unentscheidbarkeit



Eine **Reduktion** ist eine Funktion $R(.)$.

1. Jede Instanz I von P_{alt} wird auf eine Instanz $R(I)$ von P_{neu} abgebildet.
2. I yes/no Instanz von $P_{alt} \Leftrightarrow R(I)$ yes/no Instanz von P_{neu}

   oder

   I yes/no Instanz von $P_{alt} \Leftrightarrow R(I)$ no/yes Instanz von P_{neu}
3. $R(.)$ ist berechenbar

Die Unentscheidbarkeit der Halteproblems

Das **allgemeine Halteproblem** kann man wie folgt formalisieren:

- $H = \{(w, x) \mid w, x \in \{0,1\}^*, M_w \text{ angewendet auf } x \text{ hält}\}$

Zur Erinnerung das **spezielle Halteproblem**:

- $K = \{w \in \{0,1\}^* \mid M_w \text{ angewendet auf } w \text{ hält}\}$

Die Unentscheidbarkeit des allg. Halteproblems zeigen wir, indem wir das spezielle Halteproblem auf das allg. Halteproblems reduzieren.

Reduktion:

Als Reduktion wählen wir $R(w) = (w, w)$.

1. Jede Binärzahl wird auf ein Paar von Binärzahlen abgebildet ✓
2. $w \in K \Leftrightarrow (w, w) \in H$ ✓
3. $R(w)$ ist berechenbar ✓

Wir haben (1) K auf H reduziert und (2) K ist unentscheidbar, daher ist auch H unentscheidbar.

Das Halteproblem auf leerem Band

Definition

Unter dem Halteproblem auf leerem Band verstehen wir die Sprache

$$H_0 = \{w \in \{0,1\}^* \mid M_w \text{ angewendet auf das leere Band hält}\}$$

Theorem

Das Halteproblem auf leerem Band ist unentscheidbar.

Wir reduzieren das allg. Halteproblem H auf H_0.

Reduktion

Mann kann für jedes Paar (w, x) wie folgt eine TM M konstruieren

- Auf leerem Band gestartet schreibt M zuerst x auf das Band und
- bewegt den LS-Kopf auf das erste Zeichen von x.
- Danach verhält sich M wie M_w

Die Reduktion $R(w, x)$ gibt den Code der Maschine M zurück.

Das Halteproblem auf leerem Band

Reduktion

Mann kann für jedes Paar (w, x) wie folgt eine TM M konstruieren

- Auf leerem Band gestartet schreibt M zuerst x auf das Band und
- bewegt den LS-Kopf auf das erste Zeichen von x.
- Danach verhält sich M wie M_w

Die Reduktion $R(w, x)$ gibt den Code der Maschine M zurück.

Wir müssen noch zeigen, dass die 3 Eigenschaften einer Reduktion erfüllt sind:

1. Jedes Paar (w, x) wird auf eine Binärzahl abgebildet ✓
2. $$\begin{aligned}(w, x) \in H &\Leftrightarrow M_w \text{ mit Eingabe } x \text{ hält} \\ &\Leftrightarrow M \text{ mit leerer Eingabe hält} \\ &\Leftrightarrow R(w, x) \in H_0 \quad \checkmark\end{aligned}$$
3. $R(w, x)$ ist berechenbar (kann man zeigen) ✓

01-Post's Korrespondenzproblem

Satz

Post's Korrespondenzproblem ist unentscheidbar für $|\Sigma| = 2$.

Wir zeigen die Unentscheidbarkeit mit eine Reduktion des allg. PKP auf den Spezialfall.

Reduktion

Wir müssen eine Methode angeben, um eine beliebige Instanz (Σ, A, B) von PKP in eine Äquivalente mit $|\Sigma'| = \{0, 1\}$ umzuwandeln.

Sei $\Sigma = \{a_1, \ldots a_n\}$, dann können wir a_j als 01^j kodieren.

Wir bilden A', B', indem wir in A, B jedes a_j durch 01^j ersetzen.

01-Post's Korrespondenzproblem

Beispiel

$\Sigma = \{a, b, c\}$ mit $A = (ab, ccb, a)$ und $B = (a, bcc, ba)$.

Wir kodieren a als 01, b als 011 und c als 0111.

$$A' = (01011, 01110111011, 01)$$
$$B' = (01, 01101110111, 01101)$$

Wir überprüfen wieder die 3 Eigenschaften:

1. Jede PKP Instanz wird auf eine 01-PKP Instanz ($\Sigma = \{0, 1\}$) abgebildet. ✓
2. Eine Folge ist Lösung von (Σ, A, B) genau dann wenn sie Lösung von $(\{0, 1\}, A', B')$ ist. ✓
3. Die Kodierung ist berechenbar. ✓

Wir haben also eine Reduktion für die Unentscheidbarkeit von 01-PKP gefunden.

Einige Unentscheidbare Probleme

Grammatiken

- **Äquivalenzproblem**: Seien G_1 und G_2 kontextfreie Grammatiken. Es ist nicht entscheidbar, ob $L(G1) = L(G2)$.
- **Mehrdeutigkeit**: Es ist nicht entscheidbar ob eine kontextfreie Grammatik G mehrdeutig ist.

Turingmaschinen

- **Programmäquivalenz**: M, M' Turingmaschinen. Berechnen M und M' die selbe Funktion?
- **Turing's Halteproblem**: Hält eine Turingmaschine M für eine gegebene Eingabe I?

Prädikatenlogik

- **Erfüllbarkeitsproblem** der Prädikatenlogik: Ist diese Prädikatenlogische Formel erfüllbar?
- **Hilbert's Entscheidungsproblem**: Folgt diese Prädikatenlogische Formel aus den gegebenen Formeln?

Zusammenfassung & Ausblick

Heute haben wir Folgendes behandelt:

- Berechenbarkeit
- Entscheidbarkeit
- Halteproblem
- Unentscheidbare Problem
- Reduktionen

Nicht jedes theoretisch berechenbare Problem ist auch tatsächlich praktisch lösbar (aufgrund des hohen Berechnungsaufwands).

Verschiedene berechenbare Probleme haben ganz unterschiedliche Anforderungen an

- Rechenzeit
- Speicher
- ...

Wir wollen die berechenbaren Probleme anhand dieser Anforderungen weiter unterscheiden.

Weiter geht es mit: **Komplexitätstheorie**