TECHNISCHE UNIVERSITÄT DRESDEN

THEORETISCHE INFORMATIK UND LOGIK

5. Vorlesung: Der Satz von Rice und das Postsche Korrespondenzproblem

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 26. April 2021

Wesentliche Ergebnisse bisher

Viele Dinge sind nicht berechenbar:

- Die Busy-Beaver-Funktion
- Das Halteproblem
- Das ϵ-Halteproblem

Dazu gibt es mehrere Beweismethoden:

- Kardinalitätsargumente: Anzahl Algorithmen vs. Anzahl Probleme
- Diagonalisierungen: Nimm Berechenbarkeit an und konstruiere damit (als Subroutine) einen paradoxen Algorithmus
- Reduktionen: Zeige, dass bereits bekannte nicht berechenbare Probleme sich lösen ließen, wenn das Problem berechenbar wäre

Der Satz von Rice

Ein interessantes Resultat von Henry Gordon Rice bewahrt uns davor, noch hunderte andere Probleme im Detail zu betrachten:

Satz von Rice (informelle Version): Jede nicht-triviale Frage über die von einer TM ausgeführte Berechnung ist unentscheidbar.

Der Satz von Rice

Ein interessantes Resultat von Henry Gordon Rice bewahrt uns davor, noch hunderte andere Probleme im Detail zu betrachten:

Satz von Rice (informelle Version): Jede nicht-triviale Frage über die von einer TM ausgeführte Berechnung ist unentscheidbar.

Satz von Rice (formell): Sei E eine Eigenschaft von Sprachen, die für manche Turing-erkennbare Sprachen gilt und für manche Turing-erkennbare Sprachen nicht gilt (=„nicht-triviale Eigenschaft“). Dann ist das folgende Problem unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Alles unentscheidbar

Beispiele für Fragen, die laut Rice unentscheidbar sind:

- „Ist $\mathtt{aba} \in \mathbf{L}(\mathcal{M})$?“
- „Ist $\mathbf{L}(\mathcal{M})$ leer?“
- „Ist $\mathbf{L}(\mathcal{M})$ endlich?“
- „Ist $\mathbf{L}(\mathcal{M})$ regulär?“
- ...

Rice ist dagegen nicht anwendbar auf:

- „Hat $\mathcal{M}$ mindestens zwei Zustände?“
  (keine Eigenschaft von $\mathbf{L}(\mathcal{M})$)
- „Ist $\mathbf{L}(\mathcal{M})$ semi-entscheidbar?“ (trivial)
- ...

Der Satz von Rice lässt sich sinngemäß auf alle Turing-mächtigen Formalismen übertragen.

Der Satz von Rice: Beweis (1)

Satz von Rice: Sei E eine nicht-triviale Eigenschaft von Turing-erkennbaren Sprachen. Dann ist das folgende unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat die Sprache $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis: Sei E eine Eigenschaft wie im Satz. Wir konstruieren eine Many-One-Reduktion vom ϵ-Halteproblem auf „E-Haftigkeit“.

- Sei $\emptyset \notin E$ (o.B.d.A.: wir könnten sonst auch Unentscheidbarkeit von $\overline{E}$ beweisen)
- Sei $\mathcal{M}_{\mathbf{L}}$ eine TM, die eine Sprache $\mathbf{L} \in E$ akzeptiert
  (muss existieren, da E nicht-trivial ist)

Der Satz von Rice: Beweis (2)

Satz von Rice: Sei E eine nicht-triviale Eigenschaft von Turing-erkennbaren Sprachen. Dann ist das folgende unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat die Sprache $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis (Fortsetzung):

Der Satz von Rice: Beweis (2)

Satz von Rice: Sei E eine nicht-triviale Eigenschaft von Turing-erkennbaren Sprachen. Dann ist das folgende unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat die Sprache $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis (Fortsetzung): Für eine beliebige TM $\mathcal{M}$ sei $\mathcal{M}^*$ eine TM, die für eine Eingabe w das folgende tut:

(1) Simuliere $\mathcal{M}$ auf dem leeren Wort ϵ

(2) Falls $\mathcal{M}$ hält, simuliere $\mathcal{M}_{\mathbf{L}}$ auf w

Der Satz von Rice: Beweis (2)

Satz von Rice: Sei E eine nicht-triviale Eigenschaft von Turing-erkennbaren Sprachen. Dann ist das folgende unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat die Sprache $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis (Fortsetzung): Für eine beliebige TM $\mathcal{M}$ sei $\mathcal{M}^*$ eine TM, die für eine Eingabe w das folgende tut:

(1) Simuliere $\mathcal{M}$ auf dem leeren Wort ϵ

(2) Falls $\mathcal{M}$ hält, simuliere $\mathcal{M}_{\mathbf{L}}$ auf w

Damit gilt: falls $\mathcal{M}$ auf ϵ hält, dann $\mathbf{L}(\mathcal{M}^*) = \mathbf{L} \in E$ und
falls $\mathcal{M}$ auf ϵ nicht hält, dann $\mathbf{L}(\mathcal{M}^*) = \emptyset \notin E$

Der Satz von Rice: Beweis (2)

Satz von Rice: Sei E eine nicht-triviale Eigenschaft von Turing-erkennbaren Sprachen. Dann ist das folgende unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat die Sprache $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis (Fortsetzung): Für eine beliebige TM $\mathcal{M}$ sei $\mathcal{M}^*$ eine TM, die für eine Eingabe w das folgende tut:

(1) Simuliere $\mathcal{M}$ auf dem leeren Wort ϵ

(2) Falls $\mathcal{M}$ hält, simuliere $\mathcal{M}_{\mathbf{L}}$ auf w

Damit gilt: falls $\mathcal{M}$ auf ϵ hält, dann $\mathbf{L}(\mathcal{M}^*) = \mathbf{L} \in E$ und
falls $\mathcal{M}$ auf ϵ nicht hält, dann $\mathbf{L}(\mathcal{M}^*) = \emptyset \notin E$

Eine geeignete Many-One-Reduktion f ist demnach:

$$f(v) = \begin{cases} \text{enc}(\mathcal{M}^*) & \text{falls } v = \text{enc}(\mathcal{M}) \text{ für eine TM } \mathcal{M} \\ \# & \text{falls die Eingabe nicht korrekt kodiert ist} \end{cases}$$ □

Semi-Entscheidbarkeit

Das Halteproblem, schon wieder

Wir haben gesehen, dass das Halteproblem unentscheidbar ist, aber es ist dennoch Turing-erkennbar:

Satz: Das Halteproblem ist semi-entscheidbar.

Das Halteproblem, schon wieder

Wir haben gesehen, dass das Halteproblem unentscheidbar ist, aber es ist dennoch Turing-erkennbar:

Satz: Das Halteproblem ist semi-entscheidbar.

Beweis: Eine Turingmaschine, die das Halteproblem erkennt, ist leicht skizziert:

- Wenn die Eingabe die Form enc($\mathcal{M}$)##enc(w) hat
- dann simuliere $\mathcal{M}$ auf Eingabe w.
- Wenn $\mathcal{M}$ hält, dann halte und akzeptiere. □

Im Wesentlichen ist die TM für das Halteproblem also die universelle Turingmaschine.

Komplementierung

Rückblick (Vorlesung Formale Systeme): Für eine Sprache $\mathbf{L}$ bezeichnet $\overline{\mathbf{L}}$ die Komplementsprache:

$$\overline{\mathbf{L}} = \{w \in \Sigma^* \mid w \notin \mathbf{L}\}$$

Komplementierung

Rückblick (Vorlesung Formale Systeme): Für eine Sprache $\mathbf{L}$ bezeichnet $\overline{\mathbf{L}}$ die Komplementsprache:

$$\overline{\mathbf{L}} = \{w \in \Sigma^* \mid w \notin \mathbf{L}\}$$

Satz: Für jede Sprache $\mathbf{L}$ gibt es Turing-Reduktionen $\overline{\mathbf{L}} \leq_T \mathbf{L}$ und $\mathbf{L} \leq_T \overline{\mathbf{L}}$.

Komplementierung

Rückblick (Vorlesung Formale Systeme): Für eine Sprache $\mathbf{L}$ bezeichnet $\overline{\mathbf{L}}$ die Komplementsprache:

$$\overline{\mathbf{L}} = \{w \in \Sigma^* \mid w \notin \mathbf{L}\}$$

Satz: Für jede Sprache $\mathbf{L}$ gibt es Turing-Reduktionen $\overline{\mathbf{L}} \leq_T \mathbf{L}$ und $\mathbf{L} \leq_T \overline{\mathbf{L}}$.

Beweis: Der Algorithmus für die Reduktion $\overline{\mathbf{L}} \leq_T \mathbf{L}$ ist sehr einfach:

- Für Eingabe w,
- entscheide zunächst ob $w \in \mathbf{L}$
- und invertiere das Ergebnis anschließend.

Die Umkehrung $\mathbf{L} \leq_T \overline{\mathbf{L}}$ funktioniert ebenso. □

Komplementierung

Rückblick (Vorlesung Formale Systeme): Für eine Sprache $\mathbf{L}$ bezeichnet $\overline{\mathbf{L}}$ die Komplementsprache:

$$\overline{\mathbf{L}} = \{w \in \Sigma^* \mid w \notin \mathbf{L}\}$$

Satz: Für jede Sprache $\mathbf{L}$ gibt es Turing-Reduktionen $\overline{\mathbf{L}} \leq_T \mathbf{L}$ und $\mathbf{L} \leq_T \overline{\mathbf{L}}$.

Beweis: Der Algorithmus für die Reduktion $\overline{\mathbf{L}} \leq_T \mathbf{L}$ ist sehr einfach:

- Für Eingabe w,
- entscheide zunächst ob $w \in \mathbf{L}$
- und invertiere das Ergebnis anschließend.

Die Umkehrung $\mathbf{L} \leq_T \overline{\mathbf{L}}$ funktioniert ebenso. □

Korollar: $\mathbf{L}$ ist genau dann entscheidbar, wenn $\overline{\mathbf{L}}$ entscheidbar ist.

Zwei halbe Entscheidbarkeiten = eine ganze

Es gibt einen interessanten Zusammenhang von Komplementierung und Semi-Entscheidbarkeit:

Satz: **L** ist genau dann entscheidbar, wenn **L** und $\overline{\mathbf{L}}$ semi-entscheibar sind.

Zwei halbe Entscheidbarkeiten = eine ganze

Es gibt einen interessanten Zusammenhang von Komplementierung und Semi-Entscheidbarkeit:

Satz: **L** ist genau dann entscheidbar, wenn **L** und $\overline{\mathbf{L}}$ semi-entscheibar sind.

Beweis: „⇒“ Angenommen **L** ist entscheidbar.

Zwei halbe Entscheidbarkeiten = eine ganze

Es gibt einen interessanten Zusammenhang von Komplementierung und Semi-Entscheidbarkeit:

Satz: **L** ist genau dann entscheidbar, wenn **L** und $\overline{\mathbf{L}}$ semi-entscheibar sind.

Beweis: „⇒“ Angenommen **L** ist entscheidbar.

- Dann ist **L** per Definition auch semi-entscheidbar.
- Außerdem ist auch $\overline{\mathbf{L}}$ entscheidbar (gerade gezeigt), also ebenfalls semi-entscheidbar.

Zwei halbe Entscheidbarkeiten = eine ganze

Es gibt einen interessanten Zusammenhang von Komplementierung und Semi-Entscheidbarkeit:

Satz: $\mathbf{L}$ ist genau dann entscheidbar, wenn $\mathbf{L}$ und $\overline{\mathbf{L}}$ semi-entscheibar sind.

Beweis: „$\Leftarrow$“ Wenn $\mathbf{L}$ und $\overline{\mathbf{L}}$ semi-entscheibar sind, dann werden sie durch TMs $\mathcal{M}_{\mathbf{L}}$ und $\mathcal{M}_{\overline{\mathbf{L}}}$ erkannt.

Zwei halbe Entscheidbarkeiten = eine ganze

Es gibt einen interessanten Zusammenhang von Komplementierung und Semi-Entscheidbarkeit:

Satz: **L** ist genau dann entscheidbar, wenn **L** und $\overline{\mathbf{L}}$ semi-entscheibar sind.

Beweis: „$\Leftarrow$“ Wenn **L** und $\overline{\mathbf{L}}$ semi-entscheibar sind, dann werden sie durch TMs $\mathcal{M}_{\mathbf{L}}$ und $\mathcal{M}_{\overline{\mathbf{L}}}$ erkannt.

Algorithmus: Für Eingabe w, iteriere über alle $n = 1, 2, 3, \ldots$

- Simuliere $\mathcal{M}_{\mathbf{L}}$ für n Schritte:
  Wenn $\mathcal{M}_{\mathbf{L}}$ akzeptiert, dann halte und akzeptiere
- Simuliere $\mathcal{M}_{\overline{\mathbf{L}}}$ für n Schritte:
  Wenn $\mathcal{M}_{\overline{\mathbf{L}}}$ akzeptiert, dann halte und verwerfe
- Ansonsten fahre mit nächstem n fort.

Dieser Algorithmus ist korrekt und terminiert für jede Eingabe, da immer entweder $\mathcal{M}_{\mathbf{L}}$ oder $\mathcal{M}_{\overline{\mathbf{L}}}$ terminieren muss. □

Co-Semi-Entscheidbarkeit

Wir können unsere Einsichten zusammenfassen:

Korrolar: Wenn **L** unentscheidbar aber semi-entscheidbar ist, dann kann $\overline{\mathbf{L}}$ nicht semi-entscheidbar (und auch nicht entscheidbar) sein.

Co-Semi-Entscheidbarkeit

Wir können unsere Einsichten zusammenfassen:

Korrolar: Wenn $\mathbf{L}$ unentscheidbar aber semi-entscheidbar ist, dann kann $\overline{\mathbf{L}}$ nicht semi-entscheidbar (und auch nicht entscheidbar) sein.

Beispiel: Sei $\overline{\mathbf{P}}_{\text{Halt}}$ das Komplement des Halteproblems $\mathbf{P}_{\text{Halt}}$. Dann ist $\overline{\mathbf{P}}_{\text{Halt}} \leq_T \mathbf{P}_{\text{Halt}}$ und $\mathbf{P}_{\text{Halt}} \leq_T \overline{\mathbf{P}}_{\text{Halt}}$, aber $\overline{\mathbf{P}}_{\text{Halt}}$ ist nicht semi-entscheidbar.

Anmerkung: Wir hatten $\overline{\mathbf{P}}_{\text{Halt}} \leq_T \mathbf{P}_{\text{Halt}}$ in Vorlesung 4 leicht anders definiert, da wir falsch kodierte Eingaben abgelehnt hatten. Die Aussage gilt aber auch für die erste Definition.

Many-One-Reduktionen

Beobachtung: Mit Turing-Reduktionen können wir Entscheidbarkeit oder Unentscheidbarkeit zeigen, aber nicht Semi-Entscheidbarkeit.

Bei Many-One-Reduktionen ist das anders:

Satz: Wenn $\mathbf{P} \leq_m \mathbf{Q}$ und **Q** semi-entscheidbar ist, dann ist auch **P** semi-entscheidbar.

Many-One-Reduktionen

Beobachtung: Mit Turing-Reduktionen können wir Entscheidbarkeit oder Unentscheidbarkeit zeigen, aber nicht Semi-Entscheidbarkeit.

Bei Many-One-Reduktionen ist das anders:

Satz: Wenn $\mathbf{P} \leq_m \mathbf{Q}$ und $\mathbf{Q}$ semi-entscheidbar ist, dann ist auch $\mathbf{P}$ semi-entscheidbar.

Beweis: Die Reduktion liefert einen Semi-Entscheidungsalgorithmus. Die Korrektheit folgt direkt aus den Definitionen. □

Anmerkung 1: Wir haben diese Aussage in der letzten Vorlesung mit „entscheidbar“ anstelle von „semi-entscheidbar“ gezeigt. Die Idee ist genau die gleiche.
Anmerkung 2: Die Aussage gilt analog wenn man „co-semi-entscheidbar“ anstelle von „semi-entscheidbar“ verwendet. Dies folgt schon deshalb, weil eine Many-One-Reduktion $\mathbf{P} \leq_m \mathbf{Q}$ gleichzeitig auch eine Many-One-Reduktion $\overline{\mathbf{P}} \leq_m \overline{\mathbf{Q}}$ ist.
Anmerkung 3: Damit schließen wir einen Beweis aus der letzten Vorlesung ab: Es gibt keine Many-One-Reduktion $\mathbf{P}_{\text{Halt}} \leq_m \overline{\mathbf{P}}_{\text{Halt}}$.

Das Postsche Korrespondenzproblem

Ein neues Problem

Bisher hatten alle unsere unentscheidbaren Probleme direkt mit Turingmaschinen bzw. Programmen zu tun.

$\rightsquigarrow$ Gibt es auch unentscheidbare Probleme ohne direkten Bezug zu Berechnung?

Ein neues Problem

Bisher hatten alle unsere unentscheidbaren Probleme direkt mit Turingmaschinen bzw. Programmen zu tun.

$\rightsquigarrow$ Gibt es auch unentscheidbare Probleme ohne direkten Bezug zu Berechnung?

Ja, z.B. das Postsche Korrespondenzproblem. Das PCP gleicht einem Dominospiel, in dem Dominos mit Wörtern beschriftet sind.

Beispiel: $\left[\begin{matrix}\texttt{AB}\\ \texttt{A}\end{matrix}\right]$ $\left[\begin{matrix}\texttt{AA}\\ \texttt{A}\end{matrix}\right]$ $\left[\begin{matrix}\texttt{B}\\ \texttt{BBAB}\end{matrix}\right]$

Ziel ist es, beliebig viele Dominos jeden Typs so in Reihe zu legen, dass oberes und unteres Wort gleich werden

Ein neues Problem

Bisher hatten alle unsere unentscheidbaren Probleme direkt mit Turingmaschinen bzw. Programmen zu tun.

$\rightsquigarrow$ Gibt es auch unentscheidbare Probleme ohne direkten Bezug zu Berechnung?

Ja, z.B. das Postsche Korrespondenzproblem. Das PCP gleicht einem Dominospiel, in dem Dominos mit Wörtern beschriftet sind.

Beispiel: $\left[\begin{matrix}\texttt{AB}\\ \texttt{A}\end{matrix}\right]\ \left[\begin{matrix}\texttt{AA}\\ \texttt{A}\end{matrix}\right]\ \left[\begin{matrix}\texttt{B}\\ \texttt{BBAB}\end{matrix}\right]$

Ziel ist es, beliebig viele Dominos jeden Typs so in Reihe zu legen, dass oberes und unteres Wort gleich werden, z.B.

$$\left[\begin{matrix}\texttt{AB}\\ \texttt{A}\end{matrix}\right]\left[\begin{matrix}\texttt{B}\\ \texttt{BBAB}\end{matrix}\right]\left[\begin{matrix}\texttt{AB}\\ \texttt{A}\end{matrix}\right]\left[\begin{matrix}\texttt{AA}\\ \texttt{A}\end{matrix}\right]$$

Emil Leon Post

11.2.1897 – 21.4.1954

Tragisches Genie
Wegbereiter der Logik
Stiller Vordenker von Gödel und Turing

Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PCP) besteht in der folgenden Frage.

Gegeben: eine endliche Folge von Wortpaaren

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

über einem Alphabet Σ^*.

Frage: Gibt es eine Folge von Zahlen $i_1, \ldots, i_\ell$, so dass gilt

$$x_{i_1} \cdots x_{i_\ell} = y_{i_1} \cdots y_{i_\ell},$$

wobei $\ell > 0$ ist und $i_j \in \{1, \ldots, k\}$ für alle $j = 1, \ldots, \ell$?

Beispiele

$$\begin{bmatrix} \texttt{AB} \\ \texttt{B} \end{bmatrix} \quad \begin{bmatrix} \texttt{B} \\ \texttt{BBB} \end{bmatrix} \quad \begin{bmatrix} \texttt{BB} \\ \texttt{BA} \end{bmatrix}$$

Beispiele

$$\left[\begin{matrix}\mathtt{AB}\\ \mathtt{B}\end{matrix}\right] \quad \left[\begin{matrix}\mathtt{B}\\ \mathtt{BBB}\end{matrix}\right] \quad \left[\begin{matrix}\mathtt{BB}\\ \mathtt{BA}\end{matrix}\right]$$

Dieses PCP hat eine Lösung mit 10 Schritten (Bonusaufgabe).

Beispiele

$$\left[\begin{matrix}\texttt{AB}\\\texttt{B}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{B}\\\texttt{BBB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{BB}\\\texttt{BA}\end{matrix}\right]$$

Dieses PCP hat eine Lösung mit 10 Schritten (Bonusaufgabe).

$$\left[\begin{matrix}\texttt{ABB}\\\texttt{A}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{BB}\\\texttt{AAB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{A}\\\texttt{BB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{A}\\\texttt{BAA}\end{matrix}\right]$$

Beispiele

$$\left[\begin{matrix}\texttt{AB}\\ \texttt{B}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{B}\\ \texttt{BBB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{BB}\\ \texttt{BA}\end{matrix}\right]$$

Dieses PCP hat eine Lösung mit 10 Schritten (Bonusaufgabe).

$$\left[\begin{matrix}\texttt{ABB}\\ \texttt{A}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{BB}\\ \texttt{AAB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{A}\\ \texttt{BB}\end{matrix}\right] \quad \left[\begin{matrix}\texttt{A}\\ \texttt{BAA}\end{matrix}\right]$$

Dieses PCP hat ebenfalls eine Lösung, aber keine mit weniger als 160 Schritten!

Beispiele

$$\begin{bmatrix} \texttt{AB} \\ \texttt{B} \end{bmatrix} \quad \begin{bmatrix} \texttt{B} \\ \texttt{BBB} \end{bmatrix} \quad \begin{bmatrix} \texttt{BB} \\ \texttt{BA} \end{bmatrix}$$

Dieses PCP hat eine Lösung mit 10 Schritten (Bonusaufgabe).

$$\begin{bmatrix} \texttt{ABB} \\ \texttt{A} \end{bmatrix} \quad \begin{bmatrix} \texttt{BB} \\ \texttt{AAB} \end{bmatrix} \quad \begin{bmatrix} \texttt{A} \\ \texttt{BB} \end{bmatrix} \quad \begin{bmatrix} \texttt{A} \\ \texttt{BAA} \end{bmatrix}$$

Dieses PCP hat ebenfalls eine Lösung, aber keine mit weniger als 160 Schritten!

$$\begin{bmatrix} \texttt{AA} \\ \texttt{B} \end{bmatrix} \quad \begin{bmatrix} \texttt{BA} \\ \texttt{BAA} \end{bmatrix} \quad \begin{bmatrix} \texttt{ABA} \\ \texttt{A} \end{bmatrix}$$

Beispiele

$$\begin{bmatrix} \texttt{AB} \\ \texttt{B} \end{bmatrix} \quad \begin{bmatrix} \texttt{B} \\ \texttt{BBB} \end{bmatrix} \quad \begin{bmatrix} \texttt{BB} \\ \texttt{BA} \end{bmatrix}$$

Dieses PCP hat eine Lösung mit 10 Schritten (Bonusaufgabe).

$$\begin{bmatrix} \texttt{ABB} \\ \texttt{A} \end{bmatrix} \quad \begin{bmatrix} \texttt{BB} \\ \texttt{AAB} \end{bmatrix} \quad \begin{bmatrix} \texttt{A} \\ \texttt{BB} \end{bmatrix} \quad \begin{bmatrix} \texttt{A} \\ \texttt{BAA} \end{bmatrix}$$

Dieses PCP hat ebenfalls eine Lösung, aber keine mit weniger als 160 Schritten!

$$\begin{bmatrix} \texttt{AA} \\ \texttt{B} \end{bmatrix} \quad \begin{bmatrix} \texttt{BA} \\ \texttt{BAA} \end{bmatrix} \quad \begin{bmatrix} \texttt{ABA} \\ \texttt{A} \end{bmatrix}$$

Dieses PCP hat keine Lösung (Bonusaufgabe: Warum?).

Unentscheidbarkeit

Satz: Das PCP ist unentscheidbar.

Satz: Das PCP ist unentscheidbar.

Das zu zeigen ist nicht ganz so einfach, da das PCP auf den ersten Blick nichts mit den uns bisher bekannten unentscheidbaren Problemen zu tun hat.

Wir gehen in zwei Schritten vor:

(1) Wir reduzieren das Halteproblem auf ein modifiziertes PCP

(2) Wir reduzieren das modifizierte PCP auf PCP

Unentscheidbarkeit

Satz: Das PCP ist unentscheidbar.

Das zu zeigen ist nicht ganz so einfach, da das PCP auf den ersten Blick nichts mit den uns bisher bekannten unentscheidbaren Problemen zu tun hat.

Wir gehen in zwei Schritten vor:

(1) Wir reduzieren das Halteproblem auf ein modifiziertes PCP

(2) Wir reduzieren das modifizierte PCP auf PCP

Eine Instanz des Modifizierten PCP (MPCP) ist eine Instanz des PCP (d.h. eine Menge von Wortpaaren), für die ein bestimmtes Startpaar angegeben ist. Die Lösung des MPCP ist eine Lösung des PCP, welche mit dem Startpaar beginnt.

Turingmaschinen simulieren in MPCP (1)

Wir wollen das Halteproblem von DTMs auf das MPCP reduzieren.

Wir entwickeln dazu eine Many-One-Reduktion, die eine Instanz des Halteproblems in eine Instanz des MPCP verwandelt.
$\rightsquigarrow$ Kodiere TM-Berechnungen als Sequenz von Wortpaaren

Turingmaschinen simulieren in MPCP (1)

Wir wollen das Halteproblem von DTMs auf das MPCP reduzieren.

Wir entwickeln dazu eine Many-One-Reduktion, die eine Instanz des Halteproblems in eine Instanz des MPCP verwandelt.
$\rightsquigarrow$ Kodiere TM-Berechnungen als Sequenz von Wortpaaren

Ansatz für die Reduktion:

- Das Wort, welches zur Lösung des MPCP entsteht, kodiert den Lauf einer TM
  - Hält die TM, dann ist er Lauf endlich und es gibt eine Lösung
  - Hält die TM nicht, dann wird es keine Lösung geben
- Eine TM-Konfiguration können wir wie immer als Wort der Form $v\,q\,w$ darstellen
- Wir kodieren einen Lauf als Folge von Konfigurationen, getrennt mit # (kein Alphabetszeichen oder Zustand der TM)

Turingmaschinen simulieren in MPCP (2)

Wie kann man sicherstellen, dass die MPCP-Lösung eine korrekte Folge von TM-Konfigurationen kodiert?

Turingmaschinen simulieren in MPCP (2)

Wie kann man sicherstellen, dass die MPCP-Lösung eine korrekte Folge von TM-Konfigurationen kodiert?

Kernidee:

- Das Lösungswort soll wie folgt beginnen: $\#c_0\#c_1\#c_3\#\ldots$, wobei c_i Konfigurationen kodieren
- Beim PCP entsteht das Lösungswort doppelt, oben und unten
- Wir beginnen mit $\left[\begin{array}{c}\# \\ \#c_0\#\end{array}\right]$, d.h. das obere Wort liegt eine Konfiguration zurück
- Wir definieren die Wortpaare so, dass man oben eine Kopie der unteren Konfiguration nur dann erzeugen kann, wenn man gleichzeitig unten die Nachfolgerkonfiguration anfügt
- Falls die TM hält, dann sorgen wir dafür, dass das obere Wort die fehlende Konfiguration aufholen kann

Turingmaschinen simulieren in MPCP (3)

Überführungsregeln kodieren die Übergänge der DTM:

$$\begin{bmatrix} q\mathtt{a} \\ \mathtt{b}p \end{bmatrix} \quad \text{falls } \delta(q, \mathtt{a}) = \langle p, \mathtt{b}, R \rangle$$

$$\begin{bmatrix} \mathtt{c}q\mathtt{a} \\ p\mathtt{cb} \end{bmatrix} \quad \text{falls } \delta(q, \mathtt{a}) = \langle p, \mathtt{b}, L \rangle \text{ und } \mathtt{c} \in \Gamma \text{ beliebig}$$

$$\begin{bmatrix} q\mathtt{a} \\ p\mathtt{b} \end{bmatrix} \quad \text{falls } \delta(q, \mathtt{a}) = \langle p, \mathtt{b}, N \rangle$$

In diesen Regeln steckt die Kernidee des Beweises. Es sind die wesentlichen Regeln, mit denen man Zustandssymbol $q \in Q$ im oberen Wort replizieren kann.

Turingmaschinen simulieren in MPCP (4)

Es gibt zwei Randfälle:

Am linken Rand soll unsere TM einfach „anstoßen“ (einseitig unendliches Band):

$$\begin{bmatrix} \#q\mathrm{a} \\ \#p\mathrm{b} \end{bmatrix} \quad \text{falls } \delta(q, \mathrm{a}) = \langle p, \mathrm{b}, L \rangle$$

Am rechten Rand kann die TM das Band beliebig erweitern:

$$\begin{bmatrix} q\# \\ q\sqcup\# \end{bmatrix} \quad \text{für jeden Zustand } q \in Q$$

Anmerkung: Diese Umformung ist kein echter Rechenschritt, aber erspart uns die Auflistung von Sonderfällen für jede denkbare Transition am rechten Rand.

Turingmaschinen simulieren in MPCP (5)

Kopierregeln erlauben uns, den Rest der TM-Konfiguration (die Teile, die nicht nah am Lese-/Schreibkopf liegen) vom unteren zum oberen Wort zu kopieren:

$$\left[\begin{array}{c} x \\ x \end{array}\right] \quad \text{für jedes Symbol } x \in \Gamma \cup \{\#\}$$

Anmerkung: Damit kann man keine Zustände kopieren.

Turingmaschinen simulieren in MPCP (5)

Kopierregeln erlauben uns, den Rest der TM-Konfiguration (die Teile, die nicht nah am Lese-/Schreibkopf liegen) vom unteren zum oberen Wort zu kopieren:

$$\left[\begin{array}{c} x \\ x \end{array}\right] \quad \text{für jedes Symbol } x \in \Gamma \cup \{\#\}$$

Anmerkung: Damit kann man keine Zustände kopieren.

Die Startregel schließlich setzt die Berechnung in Gang:

$$\left[\begin{array}{c} \# \\ \#q_0 w\# \end{array}\right] \quad \text{wobei } q_0 \text{ der Startzustand und } w \in \Sigma^* \text{ das Eingabewort ist}$$

Turingmaschinen simulieren in MPCP (6)

Zwischenstand: Angefangen von der Startregel zwingen uns die Regeln, Konfigurationen zu kopieren und dabei entweder einen Berechnungsschritt auszuführen, oder mehr Speicher am rechten Rand zu allozieren.

Turingmaschinen simulieren in MPCP (6)

Zwischenstand: Angefangen von der Startregel zwingen uns die Regeln, Konfigurationen zu kopieren und dabei entweder einen Berechnungsschritt auszuführen, oder mehr Speicher am rechten Rand zu allozieren.

Es fehlt noch ein Abschluss:
(wir verwenden ein weiteres zusätzliches Symbol ●)

$$\left[\begin{matrix} q\mathrm{a} \\ \bullet \end{matrix}\right] \quad \text{falls } \delta(q, \mathrm{a}) \text{ undefiniert und } \mathrm{a} \in \Gamma \text{ (d.h. } \mathrm{a} \neq \#)$$

$$\left[\begin{matrix} \mathrm{a}\bullet \\ \bullet \end{matrix}\right] \quad \text{und} \quad \left[\begin{matrix} \bullet\mathrm{a} \\ \bullet \end{matrix}\right] \quad \text{für alle } \mathrm{a} \in \Gamma$$

$$\left[\begin{matrix} \bullet\#\# \\ \# \end{matrix}\right] \quad \text{der endgültige Abschluss}$$

Turingmaschinen simulieren in MPCP (7)

Satz: Es gibt eine Many-One-Reduktion vom Halteproblem auf das modifizierte PCP.

Beweis: Wir haben die Reduktion gerade angegeben, wobei das Wortpaar mit der Startkonfiguration das Startpaar des MPCP ist.

Korrektheit (Skizze):

- Wenn es einen haltenden Lauf gibt, dann kann man eine Lösung des MPCP finden: gemäß Konstruktion
- Wenn es eine Lösung für das MPCP gibt, dann hält die TM:
  - Wir können die Übergänge als Berechnungsschritte interpretieren, d.h. es entsteht ein Lauf
  - Das obere Wort ist anfangs kürzer und wird in keiner Regel länger als das untere, solange nicht eine haltende Konfiguration erreicht wurde
  - Also kann das MPCP nur dann eine Lösung haben, wenn die TM hält. □

Von PCP zu MPCP (1)

Es fehlt noch eine Reduktion von MPCP auf PCP.

Satz: Es gibt eine Many-One-Reduktion vom modifizierten PCP auf PCP.

Beweis: Wir verwenden zwei zusätzliche Symbole # und ■. Für ein Wort $w = a_1 \cdots a_\ell$ definieren wir:

$$ {}_{\#}w_{\#} = \#a_1\#\cdots\#a_\ell\# \qquad w_{\#} = a_1\#\cdots\#a_\ell\# \qquad {}_{\#}w = \#a_1\#\cdots\#a_\ell $$

Die gesuchte Reduktion bildet jetzt ein MPCP

$$ \left[\begin{matrix} x_1 \\ y_1 \end{matrix}\right] \quad \cdots \quad \left[\begin{matrix} x_k \\ y_k \end{matrix}\right] $$

ab auf das PCP

$$ \left[\begin{matrix} {}_{\#}x_1\# \\ {}_{\#}y_1 \end{matrix}\right] \quad \left[\begin{matrix} x_1\# \\ {}_{\#}y_1 \end{matrix}\right] \quad \cdots \quad \left[\begin{matrix} x_k\# \\ {}_{\#}y_k \end{matrix}\right] \quad \left[\begin{matrix} ■ \\ \#■ \end{matrix}\right] $$

Von PCP zu MPCP (2)

Beweis (Fortsetzung): Wir erhalten also das folgende PCP

$$\left[\begin{matrix}\#x_1\# \\ \#y_1\end{matrix}\right] \quad \left[\begin{matrix}x_1\# \\ \#y_1\end{matrix}\right] \quad \cdots \quad \left[\begin{matrix}x_k\# \\ \#y_k\end{matrix}\right] \quad \left[\begin{matrix}\blacksquare \\ \#\blacksquare\end{matrix}\right]$$

Es ist nicht schwer zu zeigen, dass dies genau dann eine Lösung hat, wenn das ursprüngliche MPCP eine hat:

Von PCP zu MPCP (2)

Beweis (Fortsetzung): Wir erhalten also das folgende PCP

$$\begin{bmatrix} \#x_1\# \\ \#y_1 \end{bmatrix} \quad \begin{bmatrix} x_1\# \\ \#y_1 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} x_k\# \\ \#y_k \end{bmatrix} \quad \begin{bmatrix} \blacksquare \\ \#\blacksquare \end{bmatrix}$$

Es ist nicht schwer zu zeigen, dass dies genau dann eine Lösung hat, wenn das ursprüngliche MPCP eine hat:

- „$\Leftarrow$“ Wenn das MPCP eine Lösung hat, dann erhalten wir leicht eine entsprechende Lösung für das PCP, wobei jedes Symbol zusätzlich von # umgeben ist und das Wort auf ■ endet

Von PCP zu MPCP (2)

Beweis (Fortsetzung): Wir erhalten also das folgende PCP

$$\left[\begin{matrix}\#x_1\# \\ \#y_1\end{matrix}\right] \quad \left[\begin{matrix}x_1\# \\ \#y_1\end{matrix}\right] \quad \cdots \quad \left[\begin{matrix}x_k\# \\ \#y_k\end{matrix}\right] \quad \left[\begin{matrix}\blacksquare \\ \#\blacksquare\end{matrix}\right]$$

Es ist nicht schwer zu zeigen, dass dies genau dann eine Lösung hat, wenn das ursprüngliche MPCP eine hat:

- „$\Leftarrow$“ Wenn das MPCP eine Lösung hat, dann erhalten wir leicht eine entsprechende Lösung für das PCP, wobei jedes Symbol zusätzlich von # umgeben ist und das Wort auf ■ endet
- „$\Rightarrow$“ Wenn das PCP eine Lösung hat, dann muss es mit dem ersten Wortpaar beginnen, da nur dieses Wortpaar gleiche Anfangssymbole hat. Durch Weglassen aller # und ■ entsteht wieder eine Lösung des MPCP. □

Zusammenfassung und Ausblick

Alle interessanten Fragen über Turingmaschinen sind unentscheidbar

Semi-Entscheidbarkeit wird durch Many-One-Reduktionen erhalten, nicht aber durch Turing-Reduktionen

Das Postsche Korrespondenzproblem ist ein unentscheidbares Problem, das nicht (direkt) mit TMs zu tun hat – es ist hilfreich bei vielen Reduktionen

Was erwartet uns als nächstes?

- Unberechenbare Probleme formaler Sprachen
- Abschließende Bemerkungen zu Berechenbarkeit
- Methoden, zur Unterteilung entscheidbarer Probleme: Komplexität

Literatur und Bildrechte

Literatur

- Richard J. Lorentz: **Creating Difficult Instances of the Post Correspondence Problem.** Computers and Games 2000: 214–228
- John J. O'Connor, Edmund F. Robertson: **Emil Leon Post.** MacTutor History of Mathematics archive, University of St Andrews. `http://www-history.mcs.st-andrews.ac.uk/Biographies/Post.html`

Bildrechte

Folie 16: gemeinfrei