Formale Grundlagen der Informatik 2
Entscheidbarkeit und Unentscheidbarkeit

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at

Entscheidungsprobleme

- (Entscheidungs)problem P:
  - (Entscheidungs)frage P mit formalen Parameters $n_1, \ldots, n_k$.

    Ist die natürliche Zahl n eine Quadratzahl?
- Eine Instanz $P(a_1, \ldots, a_k)$ des Problems:
  - Frage P mit konkreten Argumenten $a_1, \ldots, a_k$.

    Ist die natürliche Zahl 15 eine Quadratzahl?
- Die Sprache L_p des Problems:
  - $L_p := \{(a_1, \ldots, a_k) \mid \text{Die Antwort auf } P(a_1, \ldots, a_k) \text{ ist "ja"}\}$

    Die Menge aller Quadratzahlen.

Wir beschäftigen uns im Folgenden mit Entscheidungsproblemen.

Die Entscheidbarkeit von Problemen

- Problem P heißt entscheidbar, wenn L_p rekursiv ist.
  - Es existiert eine Turing-Maschine (Algorithmus) M, die für jede Instanz $P(a_1, \ldots, a_k)$ terminiert und "ja" oder "nein" antwortet:
    - Sowohl L_p als auch $\overline{L_p}$ sind rekursiv aufzählbar.
    - M kann also alle Argumente $(a_1, \ldots, a_k)$ aufzählen, für die die Antwort auf $P(a_1, \ldots, a_k)$ "ja" ist und (gleichzeitig) auch alle Argumente, für die die Antwort "nein" ist.
    - Für die gegebenen Argumente $(a_1, \ldots, a_k)$ wartet M, bis diese in einer der beiden Aufzählungen auftauchen und gibt dementsprechend die Antwort "ja" oder "nein".
- Problem P heißt semi-entscheidbar, wenn L_p rekursiv aufzählbar ist.
  - Es existiert eine Turing-Maschine (*Semi-Algorithmus*), die für jede Instanz $P(a_1, \ldots, a_k)$ entweder "ja" oder aber gar nicht antwortet:
    - M kann alle Argumente $(a_1, \ldots, a_k)$ aufzählen, für die die Antwort auf $P(a_1, \ldots, a_k)$ "ja" ist.
    - Für die gegebenen Argumente $(a_1, \ldots, a_k)$ wartet M, bis diese in der Aufzählung auftauchen und antwortet dann "ja"; tauchen sie aber nie auf, wartet M unendlich lange und antwortet daher nie.

Die Codierung einer Turing-Maschine

Turing-Maschine $M = (\{q_1, \ldots, q_n\}, \{0,1\}, \{0,1,\sqcup\}, q_1, \{q_2\}, \delta)$.

- Die Codierung $\langle M \rangle$ von M:
  - Eine Folge von 0*en* und 1*en* der Form:

    $111 \text{ code}_1 \ 11 \text{ code}_2 \ 11 \ \ldots \ 11 \text{ code}_r \ 111$
    - $code_1, \ldots, code_r$ sind die Codierungen der r Operationen von M.
  - Jede Operation ist bestimmt durch eine Abbildung

    $\delta(q_i, X_j) = (q_k, X_l, D_m)$
    - $X_1 = 0, X_2 = 1, X_3 = \sqcup, D_1 = L, D_2 = R$

      "Richtung" S kann durch Folge RL ersetzt werden.
    - $code = 0^i 1 0^j 1 0^k 1 0^l 1 0^m$

      Codierung des Tupels (i, j, k, l, m)

Eine Turing-Maschine lässt sich als Bit-Folge codieren; es gibt also abzählbar viele Turing-Maschinen.

Die Auflistung aller Turing-Maschinen

- Wir können alle Turing-Maschinen als $M_1, M_2, M_3, \ldots$ reihen.
  - (Bit)-alphabetische Sortierung ihrer Codierungen.
- Wir können alle möglichen Eingabewörter $w_1, w_2, w_3, \ldots$ reihen.
  - (Bit)-alphabetische Sortierung.
- Wir können die folgende unendliche Matrix konstruieren:

$$\begin{array}{c|cccc}
 & M_1 & M_2 & M_3 & \ldots \\
\hline
w_1 & \mathbf{a_{11}} & a_{12} & a_{13} & \ldots \\
w_2 & a_{21} & \mathbf{a_{22}} & a_{23} & \ldots \\
w_3 & a_{31} & a_{32} & \mathbf{a_{33}} & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

$$a_{ij} := \begin{cases} 0, & \text{wenn } w_i \in L(M_j) \\ 1, & \text{sonst} \end{cases}$$

Die Diagonalsprache einer Turing-Maschine

- Die Diagonalspache $L_d := \{w_j \mid a_{jj} = 1\}$:
  - $w_j \in L_d \Leftrightarrow a_{jj} = 1 \Leftrightarrow w_j \notin L(M_j)$
  - w_j ist ein Wort der Diagonalsprache, wenn es *nicht* von der Turing-Maschine M_j akzeptiert wird.
- Satz: Die Diagonalsprache ist nicht rekursiv aufzählbar.
  - Angenommen, L_d wäre rekursiv aufzählbar, dann gäbe es eine Turing-Maschine M_j mit $L_d = L(M_j)$. Es gilt nun $w_j \in L_d \Leftrightarrow a_{j,j} = 1 \Leftrightarrow w_j \notin L(M_j)$ und daher $L_d \neq L(M_j)$.

Die Diagonalsprache ist daher auch nicht rekursiv.

Das Akzeptierungsproblem

- Das Akzeptierungsproblem für Turingmaschinen:
  Akzeptiert die Turing-Maschine mit Codierung M das Wort w?
  - Die universelle Sprache L_u ist die Sprache des Akzeptierungsproblems:
    $L_u = \{(\langle M \rangle, w) \mid w \in L(M)\}$
    - Turing-Maschine M akzeptiert $w \Leftrightarrow (\langle M \rangle, w) \in L_u$
  - Eine universelle Turing-Maschine akzeptiert L_u.
    - Existiert, da L_u rekursiv aufzählbar ist (Beweis siehe Skriptum).
    - Ist ein Interpreter für Turing-Maschinen.
- Satz: das Akzeptierungsproblem ist unentscheidbar
  (d.h. L_u ist nicht rekursiv).
  - Angenommen, es gäbe M_u, das L_u akzeptiert und für jede Berechnung terminiert. Dann könnten wir M' mit $L(M') = L_d$ konstruieren:
  - M' bestimmt für die Eingabe w den Index i sodass $w = w_i$.
  - M' bestimmt $\langle M_i \rangle$ und übergibt die Eingabe $(\langle M_i \rangle, w_i)$ an M_u.
    - M_u akzeptiert die Eingabe: M' akzeptiert w nicht.
    - M_u akzeptiert die Eingabe nicht: M' akzeptiert w.

Eine universelle Turing-Maschine terminiert nicht für alle Eingaben.

Das Halteproblem

Ein scheinbar etwas einfacheres Problem.

- Das Halteproblem für Turing-Maschinen:+
  Endet die Berechnung der Turing-Maschine M für Eingabe w?
- Die Sprache dieses Problems:
  $L_h = \{(\langle M \rangle, w) \mid M \text{ hält bei Eingabe } w \text{ an}\}$
- Satz: das Halteproblem ist unentscheidbar
  (d.h. L_h ist nicht rekursiv).
  - Angenommen, es gäbe M_h, das L_h akzeptiert und für jede Berechnung terminiert. Dann könnten wir M' mit $L(M') = L_u$ konstruieren:
  - M' leitet seine Eingabe $(\langle M \rangle, w)$ an M_h weiter.
    - M_h akzeptiert die Eingabe nicht: M' akzeptiert die Eingabe nicht.
    - M_h akzeptiert die Eingabe: M' übergibt w an M und wartet auf das Ende der Berechnung. M' akzeptiert $(\langle M \rangle, w)$ genau dann wenn M das Wort w akzeptiert.
  - M' terminiert immer; wir wissen aber, dass L_u nicht rekursiv ist!

Es kann auch kein Algorithmus zur Lösung des Halteproblems existieren.

Das Halteproblem

Volkstümliche Version (nach Wikipedia).

Angenommen, es gibt eine Funktion *haltetest*:

```
haltetest(Programm, Eingabe)
  wenn Programm(Eingabe) terminiert
    dann return Ja
    sonst return Nein
```

Dann lässt sich diese im folgenden Programm verwenden:

```
test(Programm)
  wenn haltetest(Programm, Programm) = Ja dann
    laufe in einer leeren Endlosschleife
```

Wenn man nun der Prozedur *test* sich selbst als Eingabe übergibt, kann diese kein richtiges Ergebnis liefern:

```
test(test);
```

Dieser Aufruf terminiert genau dann, wenn er nicht terminiert.

Der Satz von Rice

Welche Eigenschaften von Turing-Maschinen (d.h. rekursiv aufzählbarer Sprachen) sind überhaupt entscheidbar?

- Eine Eigenschaft rekursiv aufzählbarer Sprachen ist eine Menge solcher Sprachen.
- Eine Eigenschaft $\mathcal{S}$ heißt trivial wenn $\mathcal{S}$ leer ist oder alle rekursiv aufzählbaren Sprachen enthält.
- Eine Eigenschaft $\mathcal{S}$ heißt entscheidbar, wenn die Sprache
  $L_{\mathcal{S}} := \{\langle M\rangle \mid L(M) \in \mathcal{S}\}$
  rekursiv ist (d.h. wenn es für jede Turing-Maschine M entschieden werden kann, ob die von ihr akzeptierte Sprache in $\mathcal{S}$ enthalten ist).
- Satz von Rice (1953): Keine nicht-triviale Eigenschaft rekursiv aufzählbarer Sprachen ist entscheidbar.

Alle "interessanten" Eigenschaften von Turing-Maschinen (d.h. der von ihnen akzeptierten Sprachen) sind unentscheidbar.

Der Satz von Rice

- Beweis: Sei $\mathcal{S}$ eine nicht-triviale Eigenschaft r.a. Sprachen.
  Annahme: $\emptyset \notin \mathcal{S}$ (andernfalls betrachten wir $\overline{\mathcal{S}}$).
  - Sei $L \in \mathcal{S}$ eine Sprache und M_L eine Turing-Maschine mit $L(M_L) = L$.
  - Wir konstruieren eine Turing-Maschine A, die aus Eingabe $(\langle M\rangle, w)$ die Ausgabe $\langle M'\rangle$ produziert, sodass
    $L(M') \in \mathcal{S} \Leftrightarrow w \in L(M)$
    - M' simuliert M auf w. Akzeptert M das Wort nicht, akzeptiert es auch M' nicht. Ansonsten simuliert M' das Verhalten von M_L auf w und akzeptiert w genau dann, wenn es von M_L akzeptiert wird.
    - Also $L(M') = \begin{cases} \emptyset, & \text{wenn } w \notin L(M) \\ L, & \text{wenn } w \in L(M) \end{cases}$
  - Angenommen $\mathcal{S}$ wäre durch eine Turing-Maschine $M_{\mathcal{S}}$ entscheidbar. Dann könnten wir auch M_u zur Entscheidung von L_u konstruieren:
    - Wende A auf die Eingabe $(\langle M\rangle, w)$ an und erzeuge $\langle M'\rangle$.
    - M_u akzeptiert $(\langle M\rangle, w)$ gdw. $M_{\mathcal{S}}$ die Eingabe $\langle M'\rangle$ akzeptiert.
  - Wir wissen aber bereits, dass L_u nicht rekursiv ist.

Weitere unentscheidbare Probleme

Anwendungen des Satzes von Rice.

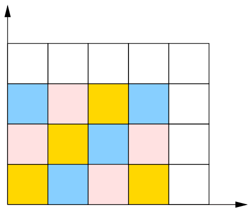
- Das eingeschränkte Akzeptierungsproblem ist unentscheidbar.
  Akzeptiert die Turing-Maschine mit Codierung M die Eingabe ϵ?
  - $L_{u,\epsilon} := \{\langle M\rangle \mid \epsilon \in L(M)\}$ ist nicht rekursiv.
  - $L_{u,\epsilon} = L_{\mathcal{S}_A}$, für eine nicht-triviale Eigenschaft $\mathcal{S}_A$.
- Das eigeschränkte Halteproblem ist unentscheidbar.
  Endet die Berechnung der Turing-Maschine M für die Eingabe ϵ?
  - $L_{h,\epsilon} := \{\langle M\rangle \mid M \text{ hält bei Eingabe } \epsilon \text{ an}\}$ ist nicht rekursiv.
  - Wäre $L_{h,\epsilon}$ rekursiv, wäre auch $L_{u,\epsilon}$ rekursiv.
    - Beweis analog zu Beweis für allgemeines Halteproblem.
- Das Problem $L(M_1) = L(M_2)$? ist unentscheidbar.
- Das Problem $L(M_1) \subseteq L(M_2)$? ist unentscheidbar.
- Das Problem $L(M) = L'$? ist unentscheidbar.

Es können keine Algorithmen (höchstens Semi-Algorithmen) zur Lösung dieser Probleme existieren.

Das Pflasterungsproblem

Auch manche mathematische Probleme können auf Probleme über Turing-Maschinen zurückgeführt werden.

- Das Pflasterungsproblem:
  - Eine endliche Menge von Typen von Pflastersteinen der Größe 1×1.
  - Ein "Anfangsstein" und eine Menge von "Nachbarschaftsregeln".
    - Welche Steintypen dürfen horizontal bzw. vertikal benachbart sein?
  - Gesucht ist eine Pflasterung des rechten oberen Quadranten der Ebene beginnend mit dem Anfangsstein an der linken unteren Ecke, sodass die Nachbarschaftsregeln berücksichtigt werden.



Das Pflasterungsproblem

- Ein Pflasterungssystem $\mathcal{D} = (D, d_0, H, V)$:
  - D ... eine endliche Menge von Pflastersteintypen.
  - $d_0 \in D$ ... der Typ des Anfangssteins.
  - $H, V \subseteq D \times D$ ... die Mengen der horizontal bzw. vertikal erlaubten Paare von benachbarten Typen.
- f ist eine Pflasterung zum Pflasterungssystem $\mathcal{D}$:
  - $f : \mathbb{N} \times \mathbb{N} \to D$
  - $f(0,0) = d_0$
  - $\forall n, m \in \mathbb{N} : (f(n,m), f(n+1,m)) \in H$
  - $\forall n, m \in \mathbb{N} : (f(n,m), f(n,m+1)) \in V$
- Das Pflasterungsproblem:

  Gibt es eine Pflasterung zum Pflasterungssystem $\mathcal{D}$?
- Satz: Das Pflasterungsproblem ist unentscheidbar.

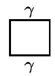
Das Pflasterungsproblem

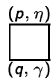
- Beweis: wir führen das eingeschränkte Halteproblem von Turingmaschinen auf das Pflasterungsproblem zurück.
  - Wir konstruieren zur Turing-Maschine $M = (Q, \Sigma, \Gamma, q_0, F, \delta)$ ein Pflasterungssystem $\mathcal{D}_M = (D, d0, H, V)$, sodass es genau dann eine Pflasterung zu $\mathcal{D}_M$ gibt, wenn M bei Eingabe ϵ nicht anhält.
    - Wäre das Pflasterungsproblem entscheidbar, dann wäre auch das eingeschränkte Halteproblem entscheidbar.
  - Wir nehmen dabei eine Übergangsfunktion der folgenden Form an:

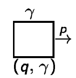
    $\delta : Q \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$:
    - $\delta(q, \gamma) = (p, \eta)$: M liest im Zustand q das Symbol γ, geht in Zustand p über und schreibt Symbol η (L/S-Kopf bleibt stationär).
    - $\delta(q, \gamma) = (p, L)$: M liest im Zustand q das Symbol γ, geht in Zustand p über, und bewegt den L/S-Kopf nach links (ohne zu schreiben).
    - $\delta(q, \gamma) = (p, L)$: M liest im Zustand q das Symbol γ, geht in Zustand p über, und bewegt den L/S-Kopf nach rechts (ohne zu schreiben).

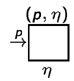
  (keine Einschränkung der Mächtigkeit von Turing-Maschinen)

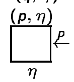
Das Pflasterungsproblem

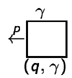
- Beweis: D enthält die folgenden Pflastersteintypen:
  - Für $\gamma \in \Gamma$ den Typ 

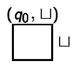
  - Für $\delta(q, \gamma) = (p, \eta)$ den Typ 

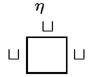
  - Für $\delta(q, \gamma) = (p, R)$ und $\eta \in \Gamma$ die Typen 

 und 

  - Für $\delta(q, \gamma) = (p, L)$ und $\eta \in \Gamma$ die Typen 

 und 

  - Die Typen 

 (Anfangsstein) und 

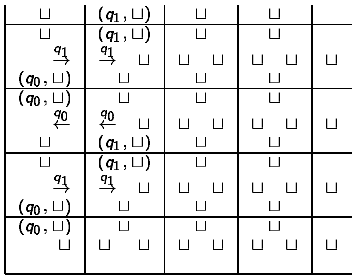
- Benachbarte Typen: müssen an Rändern zusammenpassen.
- Pflasterung zu $\mathcal{D}_M$: unendliche Berechnung von M bei Eingabe ϵ.

Beispiel

Turing-Maschine $M_4 = (\{q_0, q_1\}, \emptyset, \{\sqcup\}, \emptyset, \delta)$:

$$\delta(q_0, \sqcup) = (q_1, R), \ \delta(q_1, \sqcup) = (q_0, L).$$

Pflasterung zu $\mathcal{D}_M$:



Turing-Maschine kommt nie über zweite Bandzelle hinaus.

Das Korrespondenzproblem von Post

- Das Korrespondenzproblem von Post:
  - Gegeben sind zwei Listen von Wörtern über einem Alphabet Σ
    $w_1, \ldots, w_k$ und $x_1, \ldots, x_k$.
  - Gibt es eine Folge von Zahlen $i_1, \ldots, i_m \in \{1, \ldots, k\}, m \geq 1$, sodass
    $w_{i_1} \ldots w_{i_m} = x_{i_1} \ldots x_{i_m}$?
- Beispiel: sei $\Sigma = \{0, 1\}$ und
  $w_1 = 1, w_2 = 10111, w_3 = 10$ und $x_1 = 111, x_2 = 10, x_3 = 0$
  Es gilt
  $w_2 w_1 w_1 w_3 = 101111110 = x_2 x_1 x_1 x_3$
  also ist $i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$ eine Lösung.
- Satz: Das Korrespondenzproblem von Post ist unentscheidbar.
  - Sogar, wenn $i_1 = 1$ festgelegt wird.
  - Beweis siehe Skriptum.

Als Konsequenz ist zum Beispiel auch unentscheidbar, ob eine kontextfreie Grammatik mehrdeutig ist.

Das Emptiness-Problem

- Das Non-Emptiness-Problem:
  Akzeptiert die Turing-Maschine mit Codierung M ein Wort?
  - Die Sprache L_{ne} dieses Problems:
    $L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$
  - Satz: L_{ne} ist rekursiv aufzählbar aber nicht rekursiv.
    Beweis siehe Skriptum.
  - Das Problem ist nur semi-entscheidbar.
- Das Emptiness-Problem:
  Akzeptiert die Turing-Maschine mit Codierung M kein Wort?
  - Die Sprache L_e dieses Problems:
    $L_e = \{\langle M \rangle \mid L(M) = \emptyset\}$
  - Satz: L_e ist nicht rekursiv aufzählbar.
    Beweis siehe Skriptum.
  - Das Problem ist nicht einmal semi-entscheidbar.

Weitere unentscheidbare Probleme.

Orakel-Turingmaschinen

Was wäre, wenn wir gewisse unentscheidbare Probleme (mit einem stärkeren Berechnungsmodell?) doch entscheiden könnten?

- Orakel-Turingmaschine M^A mit Orakel für A:
  - Sprache $A \subseteq \Sigma^*$.
  - Turing-Maschine mit drei ausgezeichneten Zuständen $q_?$, q_j und q_n.
    - Ist M^A in Zustand $q_?$, wird an das Orakel die Frage gestellt:
      Ist das Wort rechts vom L/S-Kopf (bis zum ersten Leersymbol) in A?
    - Ist Antwort "ja", geht M^A in den Zustand q_j.
    - Ist Antwort "nein", geht M^A in den Zustand q_n.
- Ist A nicht rekursiv, so kann M^A durch keine Turing-Maschine (ohne Orakel) simuliert werden.
  - $L(M^A)$ ist möglicherweise nicht rekursiv aufzählbar.

Das Konzept der Orakel-Turingmaschinen ist nützlich zur Klassifikation unentscheidbarer Probleme.

Orakel-Turingmaschinen und Sprachen

- Eine Sprache B ist rekursiv aufzählbar bezüglich A:
  - Es gibt eine Orakel-Turingmaschine M^A mit $B = L(M^A)$.
- Eine Sprache B ist rekursiv bezüglich A:
  - Es gibt eine Orakel-Turingmaschine M^A mit $B = L(M^A)$, deren Berechnungen für jede Eingabe enden.
- Zwei Sprachen sind äquivalent:
  - Jede Sprache ist rekursiv bezüglich der anderen.

Äquivalente unentscheidbare Probleme sind also "gleich schwierig" (nicht) zu lösen.

Orakel für das Emptiness-Problem

Was wäre, wenn wir ein Orakel für das Emptiness-Problem hätten?

- Nicht jede Sprache ist rekursiv bezüglich L_e:
  - Es gibt überabzählbar viele Sprachen aber nur abzählbar viele Turing-Maschinen.
  - Also gibt es Probleme, die sich nicht mit einer Orakel-Turingmaschine M^{L_e} entscheiden lassen.
  - Wir können für solches Problem P ein Orakel annehmen und damit die Probleme lösen, deren Sprachen rekursiv bezüglich $L(P)$ sind.
  - Dieser Prozess lässt sich beliebig fortsetzen.

Idee für den Aufbau einer Hierarchie von Orakeln.

Hierachie von Orakeln

- Wir können so eine Hierachie von Orakeln konstruieren:

  $S_0 := \emptyset$
  $S_1 := \{\langle M\rangle \mid L(M^{S_0}) = \emptyset\}$
  $S_2 := \{\langle M\rangle \mid L(M^{S_1}) = \emptyset\}$
  ...
  $S_{i+1} := \{\langle M\rangle \mid L(M^{S_i}) = \emptyset\}$
  ...
  - $S_0 = \emptyset$; das entsprechende Orakel ist trivial.
    - M^{S_0} entspricht einer Turing-Maschine ohne Orakel.
  - $S_1 = L_e$; das entsprechende Orakel löst das Emptiness-Problem für Turing-Maschinen ohne Orakel.
  - Das Orakel für S_{i+1} löst das Emptiness-Problem für die Orakel-Turingmaschinen M^{S^i}.
- Wir erhalten damit eine Hierachie von Sprachen über $\{0,1\}$:
  - Entscheidbar mit Orakel für S_0: rekursive Sprachen.
  - Entscheidbar mit Orakel für S_1.
  - Entscheidbar mit Orakel für S_2.
  - ...

Klassifikation unentscheidbarer Probleme

Man kann einige (nicht alle) unentscheidbare Probleme nach ihrer Äquivalenz zu Elementen der Folge $S_0, S_1, S_2, \ldots$ klassifizieren.

- Das Problem $L(M) = \Sigma^*$.

  Akzeptiert die Turing-Maschine M alle Eingaben?
  - $\Sigma = \{0,1\}$ das Eingabealphabet von M.
- Satz: Das Problem $L(M) = \Sigma^*$ ist äquivalent zu S_2.
  - Beweis: dieses Problem ist rekursiv bezüglich S_2.
    - Wir konstruieren $M_3^{S_2}$ mit $L(M_3^{S_2}) = \{M \mid L(M) = \Sigma^*\}$.
    - $M_3^{S_2}$ konstruiert $\widehat{M}^{S_1}$, das alle Wörter $x \in \Sigma^*$ aufzählt und für jedes x das Orakel S_1 befragt, ob $x \in L(M)$. $\widehat{M}^{S_1}$ akzeptiert seine Eingabe, wenn es ein x mit $x \notin L(M)$, d.h.
      $$L(\widehat{M}^{S_1}) = \begin{cases} \emptyset, & \text{wenn} L(M) = \Sigma^* \\ \Sigma^* & \text{sonst} \end{cases}$$
    - $M_3^{S_2}$ befragt Orakel S_2 ob $L(\widehat{M}^{S_1}) = \emptyset$. Wenn ja, akzeptiert $M_3^{S_2}$ seine Eingabe, ansonsten nicht.
  - Beweis, dass S_2 rekursiv zum Problem ist, ist im Skriptum skizziert.

Manche unentscheidbare Probleme sind also "schwieriger" als andere.