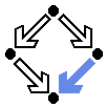


Berechenbarkeit und Komplexität

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>



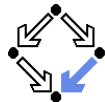
Entscheidungsprobleme



- (Entscheidungs)problem P :
 - (Entscheidungs)frage P mit formalen Parameters n_1, \dots, n_k .
Ist die natürliche Zahl n eine Quadratzahl?
- Eine Instanz $P(a_1, \dots, a_k)$ des Problems:
 - Frage P mit konkreten Argumenten a_1, \dots, a_k .
Ist die natürliche Zahl 15 eine Quadratzahl?
- Die Sprache L_P des Problems:
 - $L_P := \{(a_1, \dots, a_k) \mid \text{Die Antwort auf } P(a_1, \dots, a_k) \text{ ist "ja"}\}$
Die Menge aller Quadratzahlen.

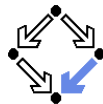
Wir beschäftigen uns im Folgenden mit Entscheidungsproblemen.

Die Entscheidbarkeit von Problemen



- Problem P heißt **entscheidbar**, wenn L_P rekursiv ist.
 - Es existiert eine Turing-Maschine (Algorithmus) M , die für jede Instanz $P(a_1, \dots, a_k)$ terminiert und “ja” oder “nein” antwortet:
 - Sowohl L_P als auch $\overline{L_P}$ sind rekursiv aufzählbar.
 - M kann also alle Argumente (a_1, \dots, a_k) aufzählen, für die die Antwort auf $P(a_1, \dots, a_k)$ “ja” ist und (gleichzeitig) auch alle Argumente, für die die Antwort “nein” ist.
 - Für die gegebenen Argumente (a_1, \dots, 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 $P(a_1, \dots, a_k)$ entweder “ja” oder aber gar nicht antwortet:
 - M kann alle Argumente (a_1, \dots, a_k) aufzählen, für die die Antwort auf $P(a_1, \dots, a_k)$ “ja” ist.
 - Für die gegebenen Argumente (a_1, \dots, 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 $M = (\{q_1, \dots, q_n\}, \{0, 1\}, \{0, 1, \sqcup\}, q_1, \{q_2\}, \delta)$.

■ Die **Codierung** $\langle M \rangle$ von M :

■ Eine Folge von $0en$ und $1en$ der Form:

111 code₁ 11 code₂ 11 ... 11 code_r 111

■ code₁, ..., code_r sind die Codierungen der r Operationen von M .

■ Jede Operation ist bestimmt durch eine Abbildung

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

■ $X_1 = 0, X_2 = 1, X_3 = \sqcup, D_1 = L, D_2 = R$

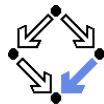
“Richtung” S kann durch Folge RL ersetzt werden.

■ $code = 0^i 10^j 10^k 10^l 10^m$

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

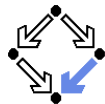


- Wir können alle Turing-Maschinen als M_1, M_2, M_3, \dots reihen.
 - Sortierung der Codierungen nach Länge und dann bit-alphabetisch.
- Wir können alle möglichen Eingabewörter w_1, w_2, w_3, \dots reihen.
 - Sortierung der Wörter nach Länge und dann bit-alphabetisch.
- Wir können die folgende unendliche Matrix konstruieren:

	M_1	M_2	M_3	\dots
w_1	a_{11}	a_{12}	a_{13}	\dots
w_2	a_{21}	a_{22}	a_{23}	\dots
w_3	a_{31}	a_{32}	a_{33}	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

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

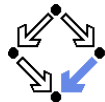
Die Diagonalsprache einer Turing-Maschine



- Die **Diagonalsprache** $L_d := \{w_j \mid a_{jj} = 1\}$:
 - $w_j \in L_d \Leftrightarrow a_{jj} = 1 \Leftrightarrow w_j \notin L(M_j)$
 - 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.

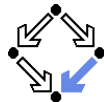
Das Akzeptierungsproblem



- 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 , die 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

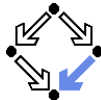


Ein scheinbar etwas einfacheres Problem.

- 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 , die 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:

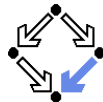
```
test(Programm)
  wenn haltetest(Programm, Programm) = Ja dann
    laufe in einer leeren Endlosschleife
```

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

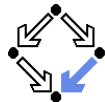


Welche Eigenschaften von Turing-Maschinen (d.h. rekursiv aufzählbarer Sprachen) sind überhaupt entscheidbar?

- 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 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.

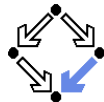
Alle “interessanten” Eigenschaften von Turing-Maschinen (d.h. der von ihnen akzeptierten Sprachen) sind unentscheidbar.

Der Satz von Rice



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

Weitere unentscheidbare Probleme

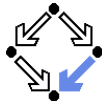


Anwendungen des Satzes von Rice.

- 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.
 - $L_{u,\epsilon} = L_{S_A}$, für eine nicht-triviale Eigenschaft S_A .
- Das **eingeschränkte 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.

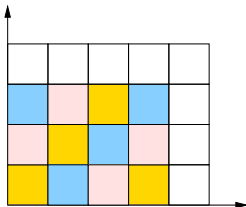
Das Pflasterungsproblem

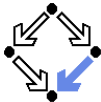


Auch manche mathematische Probleme können auf Probleme über Turing-Maschinen zurückgeführt werden.

- 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.



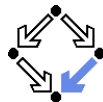


Das Pflasterungsproblem

- 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} :
 - $f : \mathbb{N} \times \mathbb{N} \rightarrow D$
 - $f(0, 0) = d_0$
 - $\forall n, m \in \mathbb{N} : (f(n, m), f(n + 1, m)) \in H$
 - $\forall n, m \in \mathbb{N} : (f(n, m), f(n, m + 1)) \in V$
- Das **Pflasterungsproblem**:

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

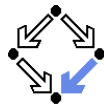
Das Pflasterungsproblem



- **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 $\mathcal{D}_M = (D, d_0, H, V)$, 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:
$$\delta : Q \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\}):$$
 - $\delta(q, \gamma) = (p, \eta)$: 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).
 - $\delta(q, \gamma) = (p, R)$: 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 $\begin{array}{c} \gamma \\ \square \\ \gamma \end{array}$

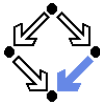
- Für $\delta(q, \gamma) = (p, \eta)$ den Typ $\begin{array}{c} (p, \eta) \\ \square \\ (q, \gamma) \end{array}$

- Für $\delta(q, \gamma) = (p, R)$ und $\eta \in \Gamma$ die Typen $\begin{array}{c} \gamma \\ \square \\ (q, \gamma) \end{array} \xrightarrow{p}$ und $\xrightarrow{p} \begin{array}{c} (p, \eta) \\ \square \\ \eta \end{array}$

- Für $\delta(q, \gamma) = (p, L)$ und $\eta \in \Gamma$ die Typen $\begin{array}{c} (p, \eta) \\ \square \\ \eta \end{array} \xleftarrow{p}$ und $\xleftarrow{p} \begin{array}{c} \gamma \\ \square \\ (q, \gamma) \end{array}$

- Die Typen $\begin{array}{c} (q_0, \sqcup) \\ \square \\ \sqcup \end{array}$ (Anfangsstein) und $\sqcup \begin{array}{c} \square \\ \sqcup \end{array} \sqcup$

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



Beispiel

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

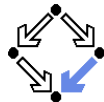
$$\delta(q_0, \sqcup) = (q_1, R), \delta(q_1, \sqcup) = (q_0, L).$$

Pflasterung zu \mathcal{D}_M :

\sqcup	(q_1, \sqcup)	\sqcup	\sqcup	
\sqcup $\xrightarrow{q_1}$ (q_0, \sqcup)	(q_1, \sqcup) $\xrightarrow{q_1}$ \sqcup	\sqcup \sqcup	\sqcup \sqcup	\sqcup
(q_0, \sqcup) $\xleftarrow{q_0}$ \sqcup	\sqcup $\xleftarrow{q_0}$ (q_1, \sqcup)	\sqcup \sqcup	\sqcup \sqcup	\sqcup
\sqcup $\xrightarrow{q_1}$ (q_0, \sqcup)	(q_1, \sqcup) $\xrightarrow{q_1}$ \sqcup	\sqcup \sqcup	\sqcup \sqcup	\sqcup
(q_0, \sqcup) \sqcup	\sqcup \sqcup	\sqcup \sqcup	\sqcup \sqcup	\sqcup

Turing-Maschine kommt nie über zweite Bandzelle hinaus.

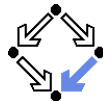
Das Korrespondenzproblem von Post



- Das **Korrespondenzproblem von Post**:
 - Gegeben sind zwei Listen von Wörtern über einem Alphabet Σ
 w_1, \dots, w_k und x_1, \dots, x_k .
 - Gibt es eine Folge von Zahlen $i_1, \dots, i_m \in \{1, \dots, k\}$, $m \geq 1$, sodass
 $w_{i_1} \dots w_{i_m} = x_{i_1} \dots x_{i_m}$?
- 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.

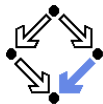
Orakel-Turingmaschinen und Sprachen



- 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

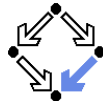


Was wäre, wenn wir ein Orakel für das Emptiness-Problem hätten?

- 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.

Hierarchie von Orakeln



- Wir können so eine **Hierarchie von Orakeln** konstruieren:

$$S_0 := \emptyset$$

$$S_1 := \{\langle M \rangle \mid L(M^{S_0}) = \emptyset\}$$

$$S_2 := \{\langle M \rangle \mid L(M^{S_1}) = \emptyset\}$$

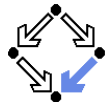
...

$$S_{i+1} := \{\langle M \rangle \mid L(M^{S_i}) = \emptyset\}$$

...

- $S_0 = \emptyset$; 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 **Hierarchie 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, \dots klassifizieren.

- **Satz:** Das Akzeptierungsproblem $w \in L(M)$ ist äquivalent zu S_1 .
 - Beweis siehe Skript.
- **Satz:** Das Problem $L(M) = \Sigma^*$ ist äquivalent zu S_2 .

Akzeptiert die Turing-Maschine M alle Eingaben?

 - Beweis, dass dieses Problem rekursiv bezüglich S_2 ist:
 - Wir konstruieren $M_3^{S_2}$ mit $L(M_3^{S_2}) = \{\langle M \rangle \mid L(M) = \Sigma^*\}$.
 - $M_3^{S_2}$ 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 $x \notin L(M)$ findet, 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 bezüglich des Problems ist, siehe Skript.

Manche unentscheidbare Probleme sind also "schwieriger" als andere.