# Berechenbarkeitstheorie: Church-Turing-Hypothese und Asimovs Robotergesetze

**Theory of Computation: Church-Turing-Hypothesis and Asimov's Laws of Robotics**

Bachelor-Thesis von Sandra Maria Meyer

März 2011

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Mathematik
Arbeitsgruppe Logik

Berechenbarkeitstheorie: Church-Turing-Hypothese und Asimovs Robotergesetze
Theory of Computation: Church-Turing-Hypothesis and Asimov's Laws of Robotics

Vorgelegte Bachelor-Thesis von Sandra Maria Meyer

Tag der Einreichung:

# Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 24. März 2011

(Sandra Meyer)

## Zusammenfassung

Diese Arbeit führt den Leser von den Grundlagen der Berechenbarkeitstheorie, der Turingmaschine und der Logik über eine Betrachtung der Unentscheidbarkeit von Akzeptanz- und Halteproblem, der Menge der Turingmaschinen ohne akzeptierende Rechnung sowie des Post'schen Korrespondenzproblems hin zu drei auf diesen Problemen beruhenden, verschiedenen Szenarien, in denen eine Maschine, den Asimov'schen Robotergesetzen gehorchend, nicht immer in der Lage ist eine richtige Entscheidung zu treffen. Auf dem Weg dorthin wird unter anderem der bekannte Satz von Rice betrachtet, die Church-Turing-Hypothese erklärt und die Äquivalenz von Register- und Turingmaschine aufgezeigt. Sie schließt mit der Frage nach den Konsequenzen der vorhandenen algorithmischen Grenzen.

## Abstract

This Bachelor Thesis leads from basics of Computability Theory, the Turing machine and logic with a view on un-decidability of Acceptance and Halting Problem, the set of all Turing machines without accepting computation and the Post Correspondence Problem to three different scenarios based on these problems, in which a machine, obeying the Laws of Robotics given by Isaac Asimov, cannot in general find the right decision. On the way Rice's famous Theorem is discussed, the Church Turing Hypothesis is mentioned and the equivalence of Turing and random access machines is shown. Finally we ask for the consequences of those given algorithmic limits.

## Inhaltsverzeichnis

# 1 Einleitung

Diese Arbeit hat das Ziel, in Anlehnung an die berühmten Robotergeschichten von Isaac Asimov, Situationen zu konstruieren, in denen ein Roboter aus rein mathematischen Gründen eine vorhandene richtige Entscheidung nicht erkennen kann. Hierzu werden, nach Erarbeitung der nötigen mathematischen Grundlagen, drei neue Robotergeschichten betrachtet in denen ein Roboter, allein aus dem Grund, dass er eine Maschine ist, nicht immer die richtige Entscheidung treffen kann, obwohl eine optimale Alternative existiert.

Zunächst sei noch einmal der Inhalt dieser Arbeit ausführlich beleuchtet.

## 1.1 Übersicht

In dieser Arbeit sollen zunächst die mathematischen Grundlagen der Berechenbarkeitstheorie erarbeitet werden. Hierzu wird in Kapitel zwei zuerst das theoretische Modell der Turingmaschine erläutert, mögliche Darstellungsformen gesucht und die Arbeitsweise einer solchen Maschine erklärt. Dies führt zur Definition der von einer Turingmaschine akzeptierten Sprache. Im Anschluss daran werden nötige Grundlagen der Logik, wie die Definition der Menge rekursiven Funktionen als die Menge der berechenbaren Funktionen, vorgestellt. Hierauf folgt dann aus der intuitiven Vorstellung heraus die Definition eines Algorithmus.

Kapitel drei beschäftigt sich ausschließlich mit dem Begriff der Entscheidbarkeit. Nachdem die Existenz von nicht-entscheidbaren Sprachen mittels des von Cantor aufgestellten Diagonalisierungsverfahren gezeigt ist, wird die Unentscheidbarkeit des so genannten Akzeptanzproblems bewiesen. Hieraus folgt nun, dass das berühmte Halteproblem unentscheidbar ist. Als weiteres unentscheidbares Problem wird die Menge aller Turingmaschinen ohne akzeptierende Rechnung betrachtet. Der Beweis für dessen Unentscheidbarkeit erfolgt durch Rückführung auf das Akzeptanzproblem. Das dritte hier erwähnte Problem ist das so genannte Post'sche Korrespondenzproblem. Nach einer ausführlichen Definition von Korrespondenzproblem, Anfangs-Korrespondenzproblem und Match wird die Unentscheidbarkeit des Post'schen Problems durch Konstruktion einer Turingmaschine, welche das Akzeptanzproblem löst, bewiesen. In diesem Zusammenhang ist noch der Satz von Rice erwähnenswert, welcher mithilfe des s-m-n-Theorems gezeigt wird.

Kapitel vier gibt einen Überblick über die Church-Turing-Hypothese. Die in der Literatur häufig auftretende Registermaschine wird betrachtet und ihre berechenbarkeitstheoretische Äquivalenz zur Turingmaschine bewiesen, indem beide Rechenmodelle ineinander überführt werden. Mit der Erkenntnis, dass auch Registermaschinen die unentscheidbaren Probleme nicht entscheiden können endet dieser Abschnitt.

Das folgende Kapitel fünf stellt kurz die vier wichtigsten Robotergesetze nach Isaac Asimov zusammen und erklärt deren Relevanz.

Kapitel sechs ist eine Zusammenführung der vorangegangenen Kapitel. In Anlehnung an die Ideen von Asimov werden hier drei voneinander unabhängige Robotergeschichten betrachtet, welche die Grenzen der algorithmischen Leistungsfähigkeit von Maschinen aufzeigen sollen. Es werden drei Szenarien konstruiert in denen ein Roboter nur aufgrund der Tatsache, dass er den berechenbarkeitstheoretischen Grenzen der Church-Turing-Hypothese unterliegt, nicht immer den Robotergesetzen gehorchen kann. Hierbei legt die erste Geschichte das Halteproblem zu Grunde und betrachtet auch den Fall der Negation der Hypothese von Turing und Church. Das Problem der Existenz von akzeptierenden Rechnungen einer Turingmaschine ist die Idee der zweiten Geschichte. Die letzte Geschichte wird aus dem Post'schen Korrespondenzproblem und dessen Unentscheidbarkeit heraus konstruiert.

Zu guter Letzt bleibt aus den Szenarien ein Resümee zu ziehen und die Bedeutung dieser algorithmischen Grenzen für uns Menschen zu überdenken.

## 2 Grundlagen der Berechenbarkeitstheorie

Zu Beginn des zwanzigsten Jahrhunderts stellten bedeutende Mathematiker wie Kurt Gödel, Alan Turing und Alonzo Church fest, dass einige grundlegende Probleme der Mathematik nicht von allein algorithmisch arbeitenden Maschinen gelöst werden können. Mit einer Klassifizierung von Problemen als lösbar oder unlösbar beschäftigt man sich seitdem in der Berechenbarkeitstheorie. Sie ist eng mit der Komplexitätstheorie verbunden, welche sich mit der Frage beschäftigt, wie schwer einzelne Probleme algorithmisch zu lösen sind.

### 2.1 Die Turingmaschine

Mit dem Ziel ein formales Modell für die Arbeitsweise eines Computers zu erstellen ersann Alan Turing im Jahre 1936 die später nach ihm benannte Turingmaschine. Bildlich kann man sie sich als eine Maschine mit einem unendlich langen, einseitig beschränkten Arbeitsband vorstellen, auf welchem sich ein Lese- und Schreibkopf befindet, der im Laufe der Rechnung Symbole vom Band lesen und auf dieses schreiben kann. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass das Band nach links beschränkt ist. Formal hat sie die folgende Definition:

**Definition 2.1** [1] *Eine Turingmaschine besteht aus*

- *einer endlichen Menge von Zuständen $Q$*
- *einem endlichen Eingabealphabet $\Sigma$, welches nicht das Blank-Symbol $\sqcup$ enthält*
- *einem endlichen Bandalphabet $\Gamma$, wobei $\sqcup \in \Gamma$ und $\Sigma \subseteq \Gamma$*
- *einer Übergangsfunktion $\delta : Q \setminus \{q_+, q_-\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$, wobei $\{L, R\}$ die Menge der möglichen Richtungen ist, in denen sich der Kopf auf dem Band bewegen kann*
- *dem ausgezeichneten Startzustand $q_0$*
- *dem akzeptierenden Zustand $q_+$ und dem ablehnenden Zustand $q_-$, wobei $q_+ \neq q_-$*

Die Turingmaschine arbeitet folgendermaßen:

Zu Beginn einer jeden Rechnung steht auf dem Arbeitsband die endliche Eingabe in Form von Symbolen aus dem Eingabealphabet. Der Rest des Bandes ist mit Blank-Symbolen ausgefüllt, die Maschine befindet sich im Zustand $q_0$ auf der ersten Zelle des Bandes. Zuerst liest sie nun das Symbol $a_1$ in der ersten Zelle des Arbeitsbandes und handelt der Übergangsfunktion entsprechend: Ist $\delta(q_0, a_1) = (q_1, b_1, R)$, so geht sie in den Zustand $q_1$ über, schreibt $b_1$ auf die erste Stelle des Bandes und geht eine Position nach rechts. Dort verfährt sie analog. Dies setzt sie so lange fort, bis sie in den akzeptierenden oder ablehnenden Endzustand übergegangen ist, woraufhin sie anhält. Sollte sie diesen nie erreichen, läuft sie ewig weiter. Gelangt die Maschine auf das Feld ganz links, so kann sie keinen Schritt mehr nach links machen, da das Arbeitsband dort endet, und verharrt in ihrer Position.[2]

Oftmals wird der Einfachheit halber die Übergangsfunktion einer Turingmaschine in Form einer Tabelle angegeben, welche für jedes Element des Bandalphabetes eine Spalte und jeden Zustand außer den Haltezuständen der Turingmaschine eine Zeile hat. Die Tabelleneinträge sind dann die Aktionen der Turingmaschine bei vorliegen des entsprechenden Zustandes und gelesenen Buchstaben.

**Satz 2.2** *Jede Turingmaschine ist durch einen Binärstring kodierbar.*

[1] vgl. [Blö06, S. 5ff]
[2] vgl. [Sip05, S. 142]

*Beweis:* Wir betrachten noch einmal die Bestandteile einer Turingmaschine: Sie besteht aus endlichen Mengen $Q, \Gamma, \Sigma$ sowie einer Übergangsfunktion $\delta : Q \setminus q_+, q_- \times \Gamma \to Q \times \Gamma \times \{R, L\}$ und ist hiermit, bis auf Umbenennung der Symbole und der Zustände, eindeutig definiert. Nun numerieren wir zunächst alle Zustände und auch die Elemente aus $\Gamma$ durch und kodieren diese Nummern binär. Da die Mengen $Q$ und $\Gamma$ endlich sind gibt es eine endliche Liste von Paaren $((q,\gamma),(q',\gamma',D))$, mit $\delta(q,\gamma) = (q',\gamma',D)$ wobei $D \in \{L,R\}$ ist. Wir kodieren nun jedes endliche solche Paar derart, dass

$$((q,\gamma),(q',\gamma',D)) =<< bin(q) < bin(\gamma) < bin(q') < bin(\gamma') < bin(D),$$

wobei $<$ ein Trennsymbol ist und $bin(c)$ die binär dargestellte Nummer des Elementes c ist. Hierbei setzen wir außerdem $bin(L) = 0$ und $bin(R) = 1$. Für diejenigen Elemente $q \in Q$, für die $\delta(q,\gamma)$ für alle $\gamma$ nicht definiert ist (dieses sind genau die beiden Endzustände $q_+$ und $q_-$) erstellen wir die Paare $((q_+,\gamma),(true,\gamma))$ beziehungsweise $((q_-,\gamma),(false,\gamma))$ für alle $\gamma \in \Gamma$. Diese kodieren wir binär nun so: $((q_+,\gamma),(true,\gamma)) =<< bin(q_+) < bin(\gamma) < 1$ und $((q_-,\gamma),(false,\gamma)) =<< bin(q_-) < bin(\gamma) < 0$.

Nun können wir einen String erstellen, in dem alle Paare $((q,\gamma),(q',\gamma',D))$, alle $((q_+,\gamma),(true,\gamma))$ und alle $((q_-,\gamma),(false,\gamma))$ binär vorkommen, indem wir sie einfach hintereinander schreiben. Um uns des Trennsymbols $<$ zu entledigen verwenden wir noch die folgende Abbildung:

$$\{0,1,<\} \to \{0,1\}\text{: } 0 \to 00,\ 1 \to 10,\ < \to 01$$

und haben einen vollständigen Binärstring der unsere Turingmaschine eindeutig beschreibt.

Beim Studium der Definition der Turingmaschine mag sich die Frage stellen, ob Turingmaschinen mit größerem Eingabealphabet mehr leisten können als eine Turingmaschine mit nur einem zweielementigen Eingabealphabet. Wir haben im obigen Beweis schon gesehen, wie man ein dreielementiges Eingabealphabet in ein zweielementiges übersetzen kann. Mit einer entsprechenden Modifikation kann man nun aus einer Turingmaschine mit dreielementigem Eingabeband eine mit nur zweielementigem Eingabeband konstruieren. Ein ähnliches Vorgehen kann man dazu benutzen, um aus einer Turingmaschinen mit (beliebigem) mehrelementigem Eingabeband eine Turingmaschine mit zweielementigem Eingabeband zu gewinnen. Deshalb bemerken wir folgendes:

**Bemerkung 2.3** *Es genügt, Turingmaschinen mit zweielementigem Eingabeband zu betrachten, da auch Turingmaschinen mit größerem Eingabealphabet nicht mehr leisten können.*

Da wir weiter oben schon gesehen haben, dass es möglich ist Turingmaschinen als Binärstrings zu kodieren, überlegen wir uns, dass es auch eine universelle Turingmaschine $M_{univ}$ geben kann, welche als Eingabe die Binärkodierung einer beliebigen anderen Turingmaschine N und einen Eingabestring $\omega$ erhält und dann das Vorgehen der Turingmaschine N bei Eingabe $\omega$ simuliert. $M_{univ}$ kommt also bei Eingabe von der Kodierung von N und $\omega$ zum selben Ergebnis wie N bei Eingabe $\omega$. Wie sich eine solche universelle Turingmaschine genau konstruieren lässt findet sich in [Her78, S.203ff].

Um einzelne Rechenschritte einer Turingmaschine genauer zu betrachten, benötigen wir einen Begriff für die Kombination von Zustand, Bandinhalt und Standpunkt des Kopfes der Turingmaschine. Dies liefert diese Definition:

**Definition 2.4** [3]*Eine Konfiguration einer Turingmaschine ist eine Sequenz der Form $\alpha q \beta$, wobei $\alpha \in \Gamma^*$ derjenige String ist, welcher links vom Lesekopf auf dem Band steht, $\beta \in \Gamma^*$ derjenige, welcher rechts davon steht und q der derzeitige Zustand der Maschine ist. Der Kopf der Maschine steht dabei auf dem ersten Symbol von $\beta$.*

[3] vgl. [Blö06, S.7]

Wir verwenden hierbei $\Gamma^* = \bigcup_{n\in\mathbb{N}} \Gamma^n$.

Im Laufe der Rechnung geht eine Turingmaschine von einer Konfiguration in eine eindeutige nächste Konfiguration über. Daher definieren wir:

**Definition 2.5** [4] *Die Konfiguration $\alpha' q' \beta'$ ist direkte Nachfolgekonfiguration von $\alpha q \beta$, falls die Turingmaschine in einem Schritt von der Konfiguration $\alpha q \beta$ in die Konfiguration $\alpha' q' \beta'$ übergeht. Man schreibt dann $\alpha q \beta \vdash \alpha' q' \beta'$ Hierbei ist $\alpha, \overline{\alpha}, \alpha', \beta, \overline{\beta}, \beta' \in \Gamma^*$ und $a, b \in \Gamma$.*

Analog dazu kann man auch $\alpha' q' \beta'$ als i-te Nachfolgekonfiguration von $\alpha q \beta$ definieren, indem man verlangt, dass es i Konfigurationen gibt sodass $\alpha q \beta \vdash \alpha_1, q_1, \beta_1 \vdash \cdots \vdash \alpha_{i-1}, q_{i-1}\beta_{i-1} \vdash \alpha' q' \beta'$
Dies führt uns zu der folgenden Definition:

**Definition 2.6** [5] *Sei M eine Turingmaschine und $\omega \in \Sigma^*$. Dann gilt: M akzeptiert $\omega$, falls es $\alpha, \beta \in \Gamma^*$ und $i \in \mathbb{N}$ gibt, sodass $\alpha q_+ \beta$ die i-te Nachfolgekonfiguration von $q_0\omega$ ist.*

Nun betrachten wir die Menge aller von einer Turingmaschine M entschiedenen Wörter:

**Definition 2.7** *Die von einer Turingmaschine M akzeptierte Sprache ist die Menge $L(M) := \{\omega \in \Gamma^* : M \text{ akzeptiert } \omega\}$*

Natürlich können wir eine Turingmaschine nicht nur dazu verwenden, eine Sprache zu entscheiden, sondern auch dazu, eine Funktion zu berechnen. In diesem Fall benötigt die Turingmaschine sogar nur einen einzigen Haltezustand. Bei der Berechnung einer Funktion soll der Funktionswert der Bandinhalt zu dem Zeitpunkt sein, zu dem die Maschine in den Haltezustand übergeht:

**Definition 2.8** [6] *Eine Turingmaschine M berechnet eine Funktion $f : \Sigma^* \to (\Gamma \setminus \sqcup)^*$, $f(\omega) = y$, wenn M bei Eingabe $\omega$ in den Haltezustand übergeht und dann auf dem ersten Element von $y$ als Bandinhalt stehenbleibt. Übrige Blank-Symbole zählen hier nicht zum Bandinhalt. Sollte $f(\omega)$ nicht definiert sein, so geht M nicht in den Haltezustand über oder bleibt nicht am Anfang des Bandes stehen.*

Man kann sich überlegen, dass auch Funktionen in mehreren Variablen von einer Turingmaschine berechnet werden können. Hierzu muss man verschiedene Eingabewerte $x_1, \ldots x_n$ in einem String zusammenfassen. Dies kann man beispielsweise erreichen indem man diese natürlichen Zahlen $x_1, \ldots x_n$ als Strings bestehend aus Einsen in den Längen $x_1 + 1, \ldots, x_n + 1$ codiert und diese durch jeweils eine Null getrennt zu einem $\omega$ zusammenfasst.

## 2.2 Grundlagen der Logik

In diesem Kapitel fassen wir bereits bekannte Definitionen und Sätze aus der Einführung in die Logik zusammen.

**Definition 2.9** [7] *Die Menge der rekursiven Funktionen ist die kleinste Menge partieller Funktionen $f :\subseteq \mathbb{N}^n \to \mathbb{N}$, welche*

- *die konstante Nullfunktion*
- *die Nachfolgerfunktion*
- *die Projektionen*

*enthält und abgeschlossen ist unter*

[4] vgl. [Blö06, S.7ff]
[5] nach [Blö06, S.8]
[6] nach [Blö06, S.9ff]
[7] nach [Mor97, S.136]

- *Komposition*
- *primitiver Rekursion*
- *Minimierung*

Es gilt: Eine (partielle) Funktion ist rekursiv genau dann, wenn sie von einer Turingmaschine berechenbar ist. Einen Beweis hierfür findet man beispielsweise in [Döp00, S.76].

In Anlehnung daran definiert man:

**Definition 2.10** [8] *Eine Menge ist rekursiv, wenn ihre charakteristische Funktion rekursiv ist. Sie ist rekursiv aufzählbar, wenn sie leer ist oder die Bildmenge einer rekursiven Funktion ist.*

In manchen Situationen ist es sinnvoll, mehreren verschiedenen Zahlen eine eindeutige einzelne zuzuordnen, etwa um mit einer Turingmaschine eine Funktion in mehreren Variablen mit nur einer Eingabe berechnen zu können. Natürlich können wir hierzu beispielsweise die Binärdarstellung der Zahlen, getrennt durch ein weiteres Element des Eingabealphabetes als Eingabe verwenden, oder gar jede Eingabeinstanz $n \in \mathbb{N}$ durch n+1 aufeinanderfolgende Einsen darstellen und einzelne Instanzen mit einer Null trennen. Eine schönere Technik hierfür ist die folgende Gödelisierung, auf welche wir vor allem in Kapitel 4 zurückgreifen werden:[9]

Wir beachten zunächst, dass es abzählbar unendlich viele Primzahlen gibt und dass umgekehrt die Primfaktorzerlegung einer natürlichen Zahl eindeutig ist. Sei $p_i$ die i-te Primzahl, wobei $p_i < p_j$ für $i < j$ gelten soll. Zur Kodierung der Zahlen $z_1, \ldots, z_n$ mit $z_1, \ldots, z_n, n \in \mathbb{N}$ in eine einzelne Zahl z setzen wir $z = p_1^{z_1} \cdots p_n^{z_n}$. Aus diesem z lassen sich so alle Zahlen $z_1, \ldots, z_n$, sogar ohne die Kenntnis von n, mittels Primfaktorzerlegung in obiger Reihenfolge rekonstruieren. Auf diese Weise können wir aus beliebig, aber endlich vielen, natürlichen Zahlen eine eindeutige einzelne konstruieren.

## 2.3 Algorithmen

Schon im Altertum fahndeten Menschen nach Regeln, mithilfe derer sie komplexere Rechnungen, wie beispielsweise die schriftliche Addition und Multiplikation, ausführen konnten. Einer der ältesten sogenannten Algorithmen ist der Euklidische Algorithmus, mit dem man den größten gemeinsamen Teiler zweier Zahlen finden kann. Heutzutage ist dieser Begriff einer der zentralen Begriffe bei der Verwendung einer Maschine zur Lösung von Problemen. Informell kann man einen Algorithmus als Verfahrensvorschrift, als Rezept oder als Prozedur beschreiben.

Eine etwas formellere Definition liefert E. Smith: [10]

**Definition 2.11** *Ein Algorithmus ist ein mechanisch ausführbares Verfahren, welches für jede Instanz eines Problems eine Antwort auf eine allgemeine Frage liefert.*

Im Prinzip ist eine Turingmaschine mit ihrer Übergangsfunktion nichts anderes als ein Algorithmus. Nach der Hypothese von Church und Turing, auf die wir später noch eingehen werden, entspricht die intuitive Notation eines Algorithmus genau derjenigen von Turingmaschinen-Algorithmen.

[8] nach [Mor97, S.148]
[9] nach [Smi96, S.33ff]
[10] vgl. [Smi96, S.1ff]

## 3 Entscheidbarkeit

Nachdem wir nun die Turingmaschine eingeführt und uns eine Vorstellung davon verschafft haben, was ein Algorithmus ist, wenden wir uns nun der Frage zu, ob alle Fragestellungen von Algorithmen beantwortet werden können. Wir werden hierbei sehen, dass nicht alle Probleme, auch wenn sie recht einfach erscheinen, algorithmisch gelöst werden können.

Zunächst einmal benötigen wir folgende Definition:

**Definition 3.1** [11] *Eine Turingmaschine M entscheidet die von ihr akzeptierte Sprache $L(M)$, wenn für alle $\omega \notin L(M)$ gilt:*
*Es gibt $\alpha, \beta \in \Gamma^*$ und $i \in \mathbb{N}$, sodass $\alpha q\_ \beta$ die i-te Nachfolgekonfiguration von $q_0\omega$ ist.*
*Wir bemerken, dass die Turingmaschine hierfür stets terminieren muss.*

Analog hierzu definieren wir:

**Definition 3.2** [12] *Eine Sprache heißt entscheidbar, wenn es eine Turingmaschine gibt, die diese entscheidet.*
*Eine Sprache heißt semi-entscheidbar, wenn es eine Turingmaschine gibt, die sie akzeptiert.*

In der Literatur findet man häufig auch die folgende, äquivalente Definition.

**Bemerkung 3.3** [13] *Eine Menge M heißt entscheidbar, wenn ihre charakteristische Funktion berechenbar ist.*

### 3.1 Das Halteproblem

Wir wenden uns nun einem ersten, für Maschinen unentscheidbaren Problem zu: dem Halteproblem. Es ist die Frage, ob eine gegebene Turingmaschine eine gegebene Eingabe akzeptiert oder nicht.

Zunächst beschäftigen wir uns mit einem Verfahren zur Bestimmung der Mächtigkeit von Mengen, welches wir für den späteren Beweis benötigen.

Aus der Analysis kennen wir bereits:

**Definition 3.4** [14] *Eine Menge heißt endlich, wenn sie nur endlich viele Elemente enthält.*
*Zwei Mengen A und B heißen gleichmächtig, wenn es eine bijektive Abbildung $f : A \to B$ gibt.*

**Definition 3.5** [15] *Eine Menge A heißt abzählbar, wenn sie endlich ist oder es eine bijektive Abbildung $f : \mathbb{N} \to A$ gibt, das heißt, wenn sie gleichmächtig ist wie die natürlichen Zahlen.*

Im Jahre 1873 stellte Georg Cantor fest, dass die Menge der rationalen Zahlen abzählbar ist. Er verwendete dazu sein so genanntes Diagonalisierungsverfahren. [16] Hierbei erstellte er eine Tabelle der Form

[11] vgl. [Sip05, S.144]
[12] vgl. [Sip05, S.144]
[13] siehe auch [EFT07, S.161ff]
[14] nach [Sip05, S.177]
[15] nach [Sip05, S.177]
[16] nach [Sip05, S.177ff]

| | $\frac{\cdot}{1}$ | $\frac{\cdot}{2}$ | $\frac{\cdot}{3}$ | $\frac{\cdot}{4}$ | $\frac{\cdot}{5}$ | $\cdots$ |
|---|---|---|---|---|---|---|
| 1 | $\frac{1}{1}$ | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{4}$ | $\frac{1}{5}$ | $\cdots$ |
| 2 | $\frac{2}{1}$ | $\frac{2}{2}$ | $\frac{2}{3}$ | $\frac{2}{4}$ | $\frac{2}{5}$ | $\cdots$ |
| 3 | $\frac{3}{1}$ | $\frac{3}{2}$ | $\frac{3}{3}$ | $\frac{3}{4}$ | $\frac{3}{5}$ | $\cdots$ |
| 4 | $\frac{4}{1}$ | $\frac{4}{2}$ | $\frac{4}{3}$ | $\frac{4}{4}$ | $\frac{4}{5}$ | $\cdots$ |
| 5 | $\frac{5}{1}$ | $\frac{5}{2}$ | $\frac{5}{3}$ | $\frac{5}{4}$ | $\frac{5}{5}$ | $\cdots$ |
| $\cdots$ | | | $\cdots$ | | | |

mit allen möglichen Zählern in jeder Spalte und allen möglichen Nennern in jeder Zeile. Auf diese Weise lassen sich alle rationalen Zahlen in die Tabelle einfügen. Cantor erkannte, dass er die Elemente der Tabelle abzählen konnte, indem er immer die Diagonalen verfolgte: $\frac{1}{1} \to \frac{2}{1} \to \frac{1}{2} \to \frac{3}{1} \to \cdots$. Bereits gezählte Brüche (also nicht vollständig gekürzte Brüche) lies er dabei aus. So erstellte er eine Bijektion f von den natürlichen Zahlen auf die rationalen Zahlen, indem er jeder natürlichen Zahl n die n-te Zahl in seiner Aufzählung der rationalen Zahlen zuwies: $f(1) = \frac{1}{1}, f(2) = \frac{2}{1}, f(3) = \frac{1}{2} \cdots$.

Nun versuchen wir ähnliches auch mit den irrationalen Zahlen. Angenommen, die irrationalen Zahlen seien abzählbar. Dann gibt es eine bijektive Funktion $f : \mathbb{N} \to \mathbb{R}$. Wir konstruieren nun ein $x \in \mathbb{R}$ sodass $f(n) \neq x$ für alle $n \in \mathbb{N}$. Hierzu verwandeln wir jede Zahl in ihre Dezimaldarstellung. Unser x soll nun unendlich viele Nachkommastellen haben. Hierbei soll die erste Nachkommastelle verschieden sein von der ersten Nachkommastelle der Zahl $f(1)$, die zweite verschieden von derjenigen der Zahl $f(2)$, allgemein also die n-te Nachkommastelle verschieden von der n-ten Nachkommastelle von $f(n)$ für alle $n \in \mathbb{N}$. Dann kann es kein $m \in \mathbb{N}$ geben mit $f(m) = x$. (Wir bedenken hierbei, dass die Dezimaldarstellung nicht unbedingt eindeutig ist, i.e. $0,29999\cdots = 0,3$, dieses Problem vermeiden wir jedoch, indem wir 0 und 9 nicht als Ziffern von x zulassen.)
Die Menge der irrationalen Zahlen ist also überabzählbar.

Nun überlegen wir uns:

**Satz 3.6** *Es gibt Sprachen, die nicht entscheidbar sind.*

*Beweis:* [17]Wären alle Sprachen entscheidbar, so gäbe es mindestens so viele Turingmaschinen wie Sprachen, da jede Turingmaschine genau eine Sprache entscheiden kann. Wir müssen also zeigen, dass es mehr Sprachen gibt als Turingmaschinen. Hierzu verwenden wir wieder einen Diagonalbeweis. Weiter oben haben wir gesehen, dass sich jede Turingmaschine als (bis auf Umbenennung der Symbole und Umnummerierung der Zustände) eindeutiger endlicher Binärstring kodieren lässt. Nun gibt es endlich viele Strings der Länge n, also ist die Menge aller Strings $\bigcup_{n\in\mathbb{N}}\{\omega, \omega \in \Sigma^n\}$ abzählbar. Also kann es höchstens abzählbar viele Turingmaschinen geben.

Nun müssen wir noch zeigen, dass es überabzählbar viele Sprachen gibt. Dabei gehen wir analog zum Beweis der Überabzählbarkeit von $\mathbb{R}$ vor.
Wir zeigen zunächst, dass es überabzählbar viele unendliche Strings gibt, also überabzählbar viele unendlich lange Sequenzen aus Nullen und Einsen. Sei $\mathscr{B}$ die Menge aller unendlichen Binärstrings. Angenommen diese Menge wäre abzählbar. Dann gibt es eine Bijektion $f_{\mathbb{N}} \to \mathscr{B}$. Wieder konstruieren wir ein $x \in \mathscr{B}$, sodass $f(n) \neq x$ für alle $n \in \mathbb{N}$. Analog zum obigen Beweis soll das gesuchte x im ersten Bit vom ersten Bit von $f(1)$ verschieden sein, im zweiten von dem von $f(2)$, im n-ten von demjenigen von

[17] nach [Sip05, S.180ff]

$f(3)$ und so weiter. Für das so konstruierte x gilt offensichtlich $f(m) \neq x$ für alle $m \in \mathbb{N}$. Damit ist $\mathscr{B}$ überabzählbar.

Wenn wir nun zeigen, dass $\mathscr{B}$ die selbe Größe hat wie die Menge aller Sprachen sind wir fertig. Sei $\mathscr{L}$ die Menge aller Sprachen über $\Sigma$. Wir setzen $\Sigma^* := \bigcup_{n\in\mathbb{N}} \Sigma^n$. Da das Alphabet $\Sigma$ per Definition endlich ist, ist jedes $\Sigma^n$ endlich und somit $\Sigma^*$ abzählbar. Daher schreiben wir $\Sigma^* = \{s_1, s_2, s_3, \dots\}$. Jede Sprache $A \in \mathscr{L}$ hat eine eindeutige Sequenz, genannt die charakteristische Sequenz, in $\mathscr{B}$, derart, dass das i-te Element 1 ist, falls $s_i \in A$, und 0 ist, falls $s_i \notin A$. Nun ist aber die Funktion $f : \mathscr{L} \to \mathscr{B}$, welche eine Sprache auf ihre charakteristische Sequenz abbildet, bijektiv. Also sind $\mathscr{B}$ und $\mathscr{L}$ gleichmächtig. Damit ist die Menge aller Sprachen überabzählbar.

Da es mehr Sprachen gibt als Turingmaschinen muss es Sprachen geben, die nicht von einer Turingmaschine entscheidbar sind.

Eine dieser Sprachen ist das Halteproblem. Bevor wir beweisen, dass dieses unentscheidbar ist, zeigen wir dies für das folgende Akzeptanzproblem:

**Satz 3.7** *Die Sprache AP = {(M, ω) : M ist eine Turingmaschine, welche die Eingabe ω akzeptiert} ist unentscheidbar.*

*Beweis:* [18] Wir zeigen dies durch einen Widerspruch. Angenommen, AP wäre entscheidbar. Dann gibt es eine Turingmaschine N derart:

$$N(\langle M, \omega\rangle) = \begin{cases} akzeptiert, & \text{falls M die Eingabe } \omega \text{ akzeptiert,} \\ abgelehnt, & \text{falls M die Eingabe } \omega \text{ ablehnt} \end{cases}$$

Wir haben oben schon gesehen, dass sich eine Turingmaschine als Binärstring kodieren lässt. Daher bezeichne $\langle M\rangle$ eine solche Binärkodierung der Turingmaschine M, $\langle M, \omega\rangle$ analog die Kodierung einer Turingmaschine $M$ und einem String $\omega$. Nun können wir folgende Turingmaschine $P$ erstellen:

$$P(\langle M\rangle) = \begin{cases} akzeptiert, & \text{falls } N(\langle M, \langle M\rangle\rangle) = abgelehnt \\ abgelehnt, & \text{falls } N(\langle M, \langle M\rangle\rangle) = akzeptiert \end{cases}$$

Diese Turingmaschine P simuliert also die Turingmaschine N mit einer Turingmaschine M und deren Maschinenbeschreibung als Eingabe und gibt dann genau das Gegenteil von N aus. Mit anderen Worten akzeptiert sie, wenn M die eigene Beschreibung ablehnt und lehnt ab, wenn M die eigene Beschreibung akzeptiert.

Nun betrachten wir $P(\langle P\rangle)$:

$$P(\langle P\rangle) = \begin{cases} akzeptiert, & \text{falls } N(\langle P, \langle P\rangle\rangle) = abgelehnt \\ abgelehnt, & \text{falls } N(\langle P, \langle P\rangle\rangle) = akzeptiert \end{cases}$$

Offensichtlich akzeptiert P die Eingabe $\langle P\rangle$ genau dann, wenn P diese ablehnt und umgekehrt. Dies ist ein Widerspruch. Also kann es ein solches P und damit ein solches N nicht geben. Damit ist AP unentscheidbar.

Nun wenden wir uns dem eigentlichen Halteproblem zu.

**Satz 3.8** *Die Sprache HP = {(M, ω) : M ist eine Turingmaschine, die bei Eingabe ω hält} ist unentscheidbar.*

[18] nach [Sip05, S.181]

*Beweis:*[19] Angenommen, HP wäre entscheidbar von einer Turingmaschine T. Dann können wir die folgende Turingmaschine S konstruieren:

$$S(\langle M,\omega\rangle) = \begin{cases} akzeptiert, & \text{falls } T(\langle M,\omega\rangle) = akzeptiert \text{ und } M(\omega) = akzeptiert, \\ abgelehnt, & \text{falls } T(\langle M,\omega\rangle) = akzeptiert \text{ und } M(\omega) = abgelehnt, \\ abgelehnt, & \text{falls } T(\langle M,\omega\rangle) = abgelehnt \end{cases}$$

Dieses S entscheidet nun jedoch AP, was unentscheidbar ist. Dies ist ein Widerspruch, also kann es eine solche Turingmaschine T nicht geben. Damit ist das Halteproblem HP unentscheidbar.

## 3.2 Weitere unentscheidbare Probleme

Analog zum letzten Beweis kann man für eine Vielzahl anderer Probleme zeigen, dass sie unentscheidbar sind. Ein weiteres wollen wir hier noch betrachten:

**Satz 3.9** *Die Menge* $LP = \{\langle M\rangle : M \text{ ist eine Turingmaschine und } L(M) = \emptyset\}$ *ist unentscheidbar.*

*Beweis:* [20] Wieder nehmen wir an, dass LP entscheidbar wäre von einer Turingmaschine T.
Zu einer beliebigen Turingmaschine M lässt sich eine Turingmaschine N konstruieren, welche höchstens den String $\omega$ akzeptiert:

$$N(x) = \begin{cases} akzeptiert & \text{falls } x = \omega \text{ und } M(\omega) = akzeptiert, \\ abgelehnt & \text{falls } x = \omega \text{ und } M(\omega) = abgelehnt, \\ abgelehnt & \text{falls } x \neq \omega \end{cases}$$

Beachte, dass der String $\omega$ Teil der Maschinenbeschreibung von N ist. Da die Eingabe immer endlich ist, kann N den Test ob $x = \omega$ gilt durch Vergleich der einzelnen Bits von $x$ und $\omega$ immer in endlicher Zeit durchführen.

Nun konstruieren wir eine Turingmaschine S, welche bei Eingabe $\langle M,\omega\rangle$ zunächst eine Turingmaschine N analog zu obiger erstellt. Dies ist möglich, denn zu der eigentlichen Maschinenbeschreibung von M müssen nur noch weitere Zustände hinzugefügt werden, welche zu Beginn den $x = \omega$-Test vollziehen und dabei schon direkt ablehnen oder M mit Eingabe $\omega$ aufrufen und dieses Ergebnis ausgeben. Hiernach soll sie die Turingmaschine T mit Eingabe N simulieren und das Gegenteil ausgeben. Wir erhalten

$$S(\langle M,\omega\rangle) = \begin{cases} akzeptiert & \text{falls } T(\langle N\rangle) = abgelehnt, \\ abgeleht & \text{falls } T(\langle N\rangle) = akzeptiert \end{cases}$$

Dieses S würde nun jedoch AP entscheiden. Da dies jedoch nicht möglich ist, kann so ein S und somit solch ein T nicht existieren. Also ist LP unentscheidbar.

[19] vgl. [Sip05, S.192]
[20] nach [Sip05, S.193ff]

## 3.3 Das Post'sche Korrespondenzproblem

Ein weiteres unentscheidbares Problem ist das Korrespondenzproblem von Post. Hierbei betrachtet man eine endliche Menge von Paaren von Worten eines Alphabetes $\mathscr{A}$. Es gilt zu entscheiden, ob es eine endliche Folge dieser Paare gibt, sodass die Elemente der ersten Komponente hintereinander gereiht genau dasselbe Wort ergeben wie diejenigen der zweiten Komponente. Formal definieren wir zunächst:

**Definition 3.10** [21] *Ein Korrespondenzsystem $\mathscr{K}$ ist eine endliche, nicht leere Menge von geordneten Paaren $(D_i, D_i')$, $i = 1, ..., m$ nichtleerer Wörter $D_i, D_i'$ gebildet aus dem Alphabet $\mathscr{A} \neq \emptyset$. Zu einer nicht leeren Indexfolge $i_1, ..., i_r$, $1 \leq i_j \leq m$ definiert man das Wort $D_{i_1} \cdots D_{i_r}$ und das dazu korrespondierende Wort $D'_{i_1} \cdots D'_{i_r}$.*

**Definition 3.11** [22] *Das Korrespondenzproblem für $\mathscr{K}$ heißt lösbar, falls es eine Indexfolge $i_1, ..., i_r$ wie oben gibt, sodass gilt $D_{i_1} \cdots D_{i_r} = D'_{i_1} \cdots D'_{i_r}$. Man nennt $D_{i_1} \cdots D_{i_r} = D'_{i_1} \cdots D'_{i_r}$ ein Match.*

Um diese Definitionen besser verstehen zu können betrachten wir das folgende kleine Beispiel:[23]
Angenommen, das Alphabet sei $\mathscr{A} = \{a, b, c\}$. Dann sind die Elemente $a, b, c, ab, ca$ und $abc$ Beispiele für Wörter über diesem Alphabet. Wir wählen nun das folgende Korrespondenzsystem:

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

Zur besseren Lesbarkeit sind hier die beiden Komponenten untereinander dargestellt. Bei näherer Betrachtung fällt die folgende Reihenfolge von Elementen aus dieser Menge auf:

$$\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right]$$

In diesem Fall ist das Wort in der ersten Komponente, $abcaaabc$, dasselbe wie jenes in der zweiten. Dieses Korrespondenzsystem hat also ein Match.

Den späteren Beweis über die Unentscheidbarkeit des Korrespondenzproblemes werden wir über ein so genanntes Anfangs-Korrespondenzproblem führen. Dieses enthält ein ausgezeichnetes Paar $(D_j, D_j')$ und wir definieren das Anfangs-Korrespondenzproblem als lösbar, falls es eine Indexfolge $j, i_2, ..., i_r$ gibt, sodass $D_j D_{i_2} \cdots D_{i_r} = D'_j D'_{i_2} \cdots D'_{i_r}$ gilt. Hierfür gilt jedoch:

**Satz 3.12** *Sind beliebige Korrespondenzprobleme entscheidbar, so sind auch beliebige Anfangs-Korrespondenzprobleme entscheidbar*

*Beweis:*[24] Wir konstruieren zu einem beliebigen Anfangs-Korrespondenzsystem $\mathscr{K}$ ein Korrespondenzsystem $\overline{\mathscr{K}}$, welches genau dann lösbar beziehungsweise unlösbar ist, wenn $\mathscr{K}$ lösbar ist.
Sei $\mathscr{A} = \{A_1, \ldots, A_N\}$ das Alphabet von $\mathscr{K}$ und $(D_1, D_1')$ das ausgezeichnete Paar. Zu jedem Wort W über $\mathscr{A}$ bilden wir nun Worte $W^l$ und $W^r$ derart, dass im Falle von $W^l$ links und im Falle von $W^r$ rechts von jedem Buchstaben von W ein neuer Buchstabe X eingefügt wird.
Nun definieren wir die Paare in $\overline{\mathscr{K}}$ folgendermaßen:
$(\overline{D_1}, \overline{D_1}') = (XD_1^r, D_1'^l)$ sei das erste Paar, für $i = 2, \ldots, m$ sei $(\overline{D_i}, \overline{D_i}') = (D_i^r, D_i'^l)$ das i-te Paar. Wir definieren mit einem weiteren Buchstaben Y das (m+1)-te Paar $(\overline{D_{m+1}}, \overline{D_{m+1}}') = (Y, XY)$.

Angenommen, eine Lösung des Anfangs-Korrespondenzproblemes $\mathscr{K}$ sei gegeben durch eine Indexfolge $1, i_2, \ldots, i_r$. Dann gilt $D_1 D_{i_2} \cdots D_{i_r} = D_1' D_{i_2}' \cdots D_{i_r}'$. Diese Worte bleiben gleich, wenn man zwischen zwei Buchstaben immer noch einen Buchstaben X einfügt. Auch ein Anfügen von X am Anfang und XY am Ende ändert an der Gleichheit nichts. Wir erhalten nun mit der Definition von

[21] nach [Her78, S.238ff]
[22] nach [Her78, S.238ff]
[23] nach [Sip05, S.203]
[24] vgl. [Her78, S. 239]

oben $XD_1XD_{i_2}X\cdots D_{i_r}XY = XD_1^rD_{i_2}^r\cdots D_{i_r}^rY$ und $XD_1'XD_{i_2}'X\cdots D_{i_r}'XY = D_1^{'l}D_{i_2}^{'l}\cdots D_{i_r}^{'l}XY$, also ist $XD_1^rD_{i_2}^r\cdots D_{i_r}^rY = D_1^{'l}D_{i_2}^{'l}\cdots D_{i_r}^{'l}XY$ und somit ist $1, i_2, \ldots, i_r, m+1$ Lösung des Korrespondenzproblems $\overline{\mathcal{K}}$. Aus einer Lösung von $\mathcal{K}$ können wir also eine solche von $\overline{\mathcal{K}}$ konstruieren.

Sei nun $i_1, \ldots, i_r$ eine Lösung von $\overline{\mathcal{K}}$, das heißt es gilt $\overline{D_{i_1}}\cdots\overline{D_{i_r}} = \overline{D_{i_1}}'\cdots\overline{D_{i_r}}'$. Wir beobachten, dass die erste Komponente der i-ten Relation auf X endet, die zweite jedoch nicht. Somit kann nur $i_r = m+1$ gelten, denn nur bei diesem Paar stimmen die Endbuchstaben überein. Nun betrachten wir den kleinsten Index $s \leq r$, sodass $i_s = m+1$ gilt, das heißt Y kommt in keinem Wort der ersten Paare vor. Nun muss also, per Konstruktion, $\overline{D_{i_1}}\cdots\overline{D_{i_s}}$ das kürzeste Anfangswort mit Endbuchstaben Y von $\overline{D_{i_1}}\cdots\overline{D_{i_r}}$ sein. Analog dazu ist $\overline{D_{i_1}}'\cdots\overline{D_{i_s}}'$ das kürzeste solche von $\overline{D_{i_1}}'\cdots\overline{D_{i_r}}'$. Wir bemerken, dass nun auch $i_1, \ldots, i_s$ das Korrespondenzsystem $\overline{\mathcal{K}}$ löst. Also können wir allgemein annehmen, dass $i_j \leq m$ gilt für alle $j < r$. Als erstes Paar in einer Lösung benötigen wir eines, das im ersten und zweiten Wort denselben Anfangsbuchstaben hat. Für $i > 1$ beginnt jedoch das zweite Wort stets mit einem X, das erste hingegen nie. Ist $i = 1$, so beginnen beide Worte mit X. Also muss $i_1 = 1$ gelten, insbesondere gilt daher $r > 1$. Da der Buchstabe X stets mit den anderen Buchstaben alterniert, muss $i_2 \neq 1$ gelten, da in dem ersten Paar das erste Wort mit X endet, das zweite hingegen nicht und wir deshalb ein zweites Element benötigen, welches in der zweiten Komponente mit X beginnt. Aufgrund des Aufbaus der i-ten Relationen für $i = 2, \ldots m$ setzt sich dies analog fort, weswegen für $j = 2, \ldots, r-1$ gilt: $1 < i_j \leq m$. Vollständig ausgeschrieben erhalten wir also als Lösung von $\overline{\mathcal{K}}$: $XD_1rD_{i_2}^r\cdots D_{i_{r-1}}DY = D_1^lD_{i_2}^l\cdots D_{i_{r-1}}XY$. Wieder können wir alle alternierenden Buchstaben X und Y weglassen und erhalten als Indexfolge der Lösung von $\mathcal{K}$: $1, i_2, \ldots, i_{r-1}$. Hiermit ist der Beweis abgeschlossen.

Nun sind wir in der Lage die Unentscheidbarkeit des Korrespondenzproblems zu beweisen.

**Satz 3.13** *Das Post'sche Korrespondenzproblem ist unentscheidbar.*

*Beweis:*[25] Angenommen, das Korrespondenzproblem wäre entscheidbar. Dann ist nach obigem Satz auch das Anfangs-Korrespondenzproblem entscheidbar. Wir konstruieren nun aus einer Turingmaschine K, die das Korrespondenzproblem entscheidet, eine Turingmaschine S, welche das Akzeptanzproblem AP entscheidet um zum Widerspruch zu gelangen.

Die Maschine S soll aus einer Eingabe von Turingmaschine $M = (Q, \Sigma, \Gamma, \delta)$ und String $\omega = (\omega_1 \cdots \omega_n)$ eine Instanz des PCP konstruieren, welche genau dann eine Lösung hat, wenn M bei Eingabe $\omega$ hält. Als ausgezeichnetes Anfangs-Paar wählen wir das Paar $(X, Xq_0\omega_1\cdots\omega_nX)$, also ein Paar bestehend aus dem, nicht im Alphabet von M enthaltenen, Buchstaben X in der ersten Komponente und der Startkonfiguration der Turingmaschine M in der zweiten Komponente, eingerahmt von X. Nun müssen wir weitere Paare erstellen, welche es erlauben, dass die erste Komponente ebenfalls die Startkonfiguration enthält und gleichzeitig in der zweiten Komponente die nächste Konfiguration erstellt wird. Wir erstellen nun nacheinander Paare, welche die Bewegung des Kopfes nach links und rechts beschreiben, und Paare, die dafür sorgen, dass die vom Kopf nicht gelesenen Bandinhalte kopiert werden.

Bewegung nach rechts: Für alle Elemente a,b des Bandalphabetes $\Gamma$ und alle Zustände $q, r \in Q$, wobei $q \neq q_-$ gilt, erstellen wir, falls $\delta(q,a) = (r,b,R)$ gilt, ein Paar $(qa, br)$.

Bewegung nach links: Für alle Elemente a,b,c des Bandalphabetes $\Gamma$ und alle $q, r \in Q$ mit $q \neq q_-$, erstellen wir, falls $\delta(q,a) = (r,b,L)$ ein Paar $(cqa, rcb)$.

Kopieren weiterer Elemente: Für alle $a \in \Gamma$ erstellen wir $(a,a)$.

Nun benötigen wir noch das Symbol X, um die einzelnen Konfigurationen voneinander zu trennen: Wir fügen $(X,X)$ und $(X, \sqcup X)$. Das letzte Paar sorgt dafür, dass wir am Ende der Konfiguration ein Blank anfügen können, da auf dem unendlichen Band der Turingmaschine unendlich viele Blanks hinter dem Bandinhalt stehen.

Während wir nun mit diesen Paaren eine Lösung konstruieren, simulieren wir die Turingmaschine M bei Eingabe $\omega$. Wir müssen nun nur noch dafür sorgen, dass, wenn der akzeptierende oder ablehnende

[25] nach [Sip05, S205ff]

Endzustand in einer Konfiguration in der zweiten Komponente erreicht wird, die erste Komponente die noch fehlenden Elemente hinzufügen kann. Hierfür benötigen wir noch die Paare:
Für alle $a \in \Gamma$: $(aq_+, q_+)$ und $(q_+a, q_+)$ sowie $(q_+XX, X)$.

Hiermit haben wir jedoch eine Turingmaschine konstruiert, welche das Akzeptanzproblem entscheidet. Da dieses jedoch unentscheidbar ist, kann das Post'sche Korrespondenzproblem nicht entscheidbar sein. Dies komplettiert den Beweis.

## 3.4 Der Satz von Rice

Wir werden nun sehen, dass auch Aussagen über die von einer Turingmaschine berechneten Funktionen nicht entscheidbar sind. Dieses Resultat liefert uns der Satz von Rice.

**Definition 3.14** [26] *Die Menge aller berechenbaren Funktionen $\phi_i$ ist $\mathscr{R} = \{\phi_i : i \in \mathbb{N}\}$*

Diese Menge ist abzählbar, da jede berechenbare Funktion von einer Turingmaschine berechnet wird und es nur abzählbar viele Turingmaschinen gibt.

Wir können nun die berechenbaren Funktionen eindeutig durch ihr Eingabe- und Ausgabeverhalten charakterisieren. Hierbei fordern wir, dass für eine Teilmenge $\mathscr{C}$ von $\mathscr{R}$ gilt: Sei $\phi_i \in \mathscr{C}$. Wenn eine Turingmaschine $T_1$, die $\phi_i$ berechnet, und eine Turingmaschine $T_2$, die eine andere Funktion $\phi_j \in \mathscr{R}$ berechnet, bei denselben Eingabewerten dieselben Ausgabewerte erzeugen und auch bei denselben Eingabewerten kein Ergebnis liefern, dann muss auch $\phi_j \in \mathscr{C}$ gelten. [27]

Außerdem wird noch folgender Satz benötigt, der auch unter dem Namen „s-m-n-Theorem" bekannt ist:

**Satz 3.15** *Es gibt eine totale berechenbare Funktion s sodass, für alle i, alle $m \geq 1$, alle $n \geq 1$ und alle $x_1, \ldots, x_m, y_1, \ldots, y_n \in \mathbb{N}$ gilt:*
$\phi_{s(i,m,x_1,\ldots,x_m)}(y_1, \ldots, y_n) = \phi_i(x_1, \ldots, x_m, y_1, \ldots, y_n)$

Intuitiv bedeutet dieser Satz: Gegeben eine berechenbare Funktion in den Argumenten $x_1, \ldots, x_m$, $y_1, \ldots, y_n \in \mathbb{N}$. Wenn wir die m Argumente $x_1, \ldots, x_m$ festhalten, so können wir eine neue, ebenfalls berechenbare, Funktion erstellen, welche nur noch die n Argumente $y_1, \ldots, y_n$ hat und bei Eingabe von $y_1, \ldots, y_n$ dasselbe liefert wie die ursprüngliche Funktion bei Eingabe von $x_1, \ldots, x_m, y_1, \ldots, y_n$.
*Beweis:*[28] Da die Eingabe für Turingmaschinen nur aus einem String besteht, gibt es eine berechenbare Funktion $\psi(m, x_1, \cdots, x_m, y_1 \cdots, y_n) = (x_1, \cdots, x_m, y_1, \cdots, y_n)$. Da $\psi$ berechenbar ist gilt $\psi = \phi_k$ für ein $k \in \mathbb{N}$. Wir können also implizit schreiben $\phi_{s(i,m,x_1,\cdots,x_m)}(y_1, \ldots, y_n) = \phi_i(\psi(m, x_1, \ldots, x_m, y_1 \ldots, y_n))$. Nun müssen wir s konstruieren. Hierzu konstruieren wir eine Funktion $f = \phi_{i_f}$ mit $f(y) = (\epsilon, y)$ und eine Funktion $g = \phi_{i_g}$ mit $g(x, y) = (Succ(x), y)$. Dabei ist $Succ(x) = x + 1$ Hieraus wiederum konstruieren wir eine Funktion $h$ mit $h(\epsilon) = i_f$ und $h(xa) = c(i_g, h(x))$ mit der Kompositionsfunktion c, das heißt es gilt $\phi_{c(b,d)} = \phi_b(\phi_d)$. $\phi_b(\phi_d)$ ist berechenbar, da die Komposition zweier berechenbarer Funktionen wieder berechenbar ist.

Wir zeigen nun induktiv, dass $\phi_{h(x)}(y) = (x, y)$ gilt:

*Induktionsanfang:* Setze $x = \epsilon$. Dann ist $h(a) = c(i_g, i_f)$ und somit $\phi_{h(a)} = \phi_{i_g}(\phi_{i_f}(y)) = g(f(y)) = g(\epsilon, y) = (a, y)$.

*Induktionsschritt:* Die Annahme gelte für x. Wir zeigen, dass sie auch für xa gelten muss. Es ist $\phi_{h(xa)(y)} = \phi_{c(i_g,h(x))}(y) = \phi_{i_g}(\phi_{h(x)}(y)) = g(\phi_{h(x)}(y)) = g(x, y) = (xa, y)$. Damit gilt die Aussage.
Also können wir schreiben $\phi_{h(x)}(\phi_{h(y)}(z)) = \phi_{h(x)}(y, z) = (x, (y, z)) = (x, y, z)$. Analog für mehrere Argumente von h.
Damit definieren wir s mittels $s(i, m, x) := c(i, c(k, c(h(m), h(x))))$

[26] nach[Mor97, S.144]
[27] nach [Mor97, S.155]
[28] nach [Mor97, S.145ff]

Nun gilt

$$\begin{aligned}\phi_{s(i,m,x)} &= \phi_i(\phi_k(\phi_{h(m)}(\phi_{h(x)}(y))))\\ &= \phi_i(\phi_k(m,x,y))\\ &= \phi_i(\psi(m,x,y)\\ &= \phi_i(x,y)\end{aligned}$$

wie gewünscht.

Dies führt uns zum folgenden Satz von Rice:

**Satz 3.16** *Sei $\mathscr{C}$ eine Teilmenge der berechenbaren Funktionen definiert durch ihr Eingabe- und Ausgabeverhalten. Dann ist die Menge $P(\mathscr{C}) = \{x : \phi_x \in \mathscr{C}\}$ entscheidbar genau dann wenn sie die leere Menge oder die Menge aller berechenbaren Funktionen ist.*

*Beweis:* [29] Wenn $P(\mathscr{C})$ die leere Menge ist, so ist $P(\mathscr{C})$ gewiss entscheidbar. Ist $P(\mathscr{C})$ ihr Komplement, so muss $\mathscr{C}$ die Menge aller berechenbaren Funktionen $\mathscr{R}$ sein. Dann ist $P(\mathscr{C})$ die Indexmenge dieser (nach obigem) aufzählbaren Menge, damit rekursiv und somit entscheidbar.
Also muss $\mathscr{C}$ eine berechenbare Funktion $\psi$ enthalten, darf jedoch nicht alle enthalten.

Sei u die nirgendwo definierte Funktion. Wir nehmen zunächst an, dass $u \notin \mathscr{C}$. Nun definieren wir $\Theta(x,y) = \psi(y) + Zero(\phi_{univ}(x,x))$. Wobei Zero die Nullfunktion ist und $\phi_{univ}$ die universelle Turingmaschine, welche bei Eingabe $(v,w)$ die Berechnung der Turingmaschine kodiert durch $v$ auf Eingabe $w$ simuliert.

$\Theta$ ist berechenbar, da $\psi, Zero$ und $\phi_{univ}$ das sind. Nach dem s-m-n-Theorem gibt es ein $j = s(i,x)$ sodass $\phi_j(y) = \psi(y) + Zero(\phi_{univ}(x,x))$ gilt. Es gilt:

Wenn $\phi_{univ}(x,x)$ hält, so ist $\phi_j = \psi$ und damit $j \in P(\mathscr{C})$

Wenn $\phi_{univ}(x,x)$ nicht hält, so ist $\phi_j$ nirgendwo definiert und nach unserer Annahme oben nicht in $\mathscr{C}$ und damit $j \notin P(\mathscr{C})$. Somit gilt dann aber $j \in P(\mathscr{C}) \Leftrightarrow \phi_{univ}$ hält, und dies ist unentscheidbar, da das Halteproblem unentscheidbar ist.

Wenn nun $u \in \mathscr{C}$ gilt, so ist $P(\mathscr{R} \setminus \mathscr{C})$ unentscheidbar nach obigem. $P(\mathscr{R})$ ist entscheidbar, das haben wir gesehen. Da nun aber $P(\mathscr{R} \setminus \mathscr{C}) = P(\mathscr{R}) \setminus P(\mathscr{C})$ gilt, muss auch $\mathscr{C}$ unentscheidbar sein.

[29] nach [Mor97, S.155ff]

## 4 Die Church-Turing-Hypothese

Im letzten Kapitel haben wir gesehen, dass sich nicht alle Probleme von einer Maschine entscheiden lassen. Hieraus ergibt sich die Frage, ob es vielleicht noch ein anderes, besseres Modell von Algorithmen und Maschinen gibt, welche bisher unentscheidbare Probleme trotzdem entscheiden können.
Eine Aussage dazu trifft die Church-Turing-Hypothese:

**Hypothese 4.1** *Alle im intuitiven Sinne berechenbaren Funktionen sind von Turingmaschinen berechenbar.*
*oder allgemein:*
*Jeder intuitiv formulierte Algorithmus kann durch eine Turingmaschine dargestellt werden.* [30]

Wenn diese Hypothese gilt, kann es also kein Modell geben, das mehr leistet als das Modell der Turingmaschine. Da es keine formale Definition für „intuitiv berechenbar“ gibt, lässt sich diese Hypothese nicht beweisen. Allerdings wurden schon viele alternative Modelle ersonnen, die sich jedoch alle als höchstens äquivalent zur Turingmaschine entpuppten. Ein Beispiel dafür sind die Registermaschinen, die wir im folgenden Abschnitt behandeln werden.

### 4.1 Registermaschinen

Die hier behandelten Registermaschinen haben die folgenden Definitionen: [31]

**Definition 4.2** *Eine Registermaschine wird repräsentiert durch unendlich viele Register, wobei jedes Register eine natürliche Zahl inklusiver der Null enthalten kann. Auf den Registern sind folgende Elementaroperationen möglich:*

- *Inkrementieren: Erhöhen des Inhalts eines Registers um 1*
- *Dekrementieren: Vermindern des Inhalts eines Registers um 1, jedoch darf die Null nicht unterschritten werden*
- *Test: die Abfrage, ob der Inhalt eines Registers größer ist als 0*

Hierbei ist es für unsere Betrachtung nicht wichtig, in welcher Weise die Zahlen in die Register eingetragen werden. Man kann sich so beispielsweise vorstellen, dass die Zahl $n \in \mathbb{N}$ durch einen (n+1) Bit langen String bestehend aus Einsen dargestellt ist und eine Null durch eine einzelne Eins repräsentiert wird.

Wir bezeichnen die Elementaroperationen wie folgt: $Inc(i) = I_i$ ist die Inkrementierung, $Dec(i) = D_i$ die Dekrementierung und $Test(i) = T_i$ ist der Test des Registers $i \geq 1$. Außerdem müssen wir das Ende eines Programmes anzeigen, hierfür verwenden wir das Symbol E. Nun können wir Registermaschinen-Programme erstellen:

**Definition 4.3** *Registermaschinen-Programme werden induktiv aus den Basisprogrammen $I_i, D_i$ und der leeren Zeichenkette $\epsilon$ aufgebaut. Dabei gilt für Programme Q und R:*

- *QR ist ein Programm*
- *alle Zeichenketten $T_i$RE sind Programme*

Wir stellen nun fest, dass jedes für jedes nichtleere Programm P Programme Q und R gibt, sodass entweder $P = I_iQ$, $P = D_iQ$ oder $P = T_iSQ$ gibt. Zudem überlegen wir, dass die Elementaroperation $T_i$ keine eigentliche Aktion, sondern nur eine Feststellung liefert. Es muss klar aus der Syntax hervorgehen, welche Aktion bei positivem und welche bei negativem Testausgang ausgeführt werden soll. Um dies

[30] beides nach [Smi96, S. 114 und 118]
[31] alle Definitionen dieses Kapitels nach [Smi96, Kapitel2]

klar zu machen, muss das Programm sozusagen korrekt geklammert werden. Wir fordern also, dass jede Pogrammschleife, die mit einem $T_i$ beginnt, auch mit einem E beendet werden muss.

Zur leichteren Vorstellung denke man sich $T_i$ als Bedingung in einer WHILE-Schleife, das danach stehende Programm S als Ausführungsanweisung. Da diese Notation intuitiver ist werden wir sie auch im folgenden benutzen.

Es muss außerdem schon aus dem Programmaufbau hervorgehen, welche Programme noch zur Ausführung innerhalb der Schleife (dann, wenn die Bedingung $T_i$ erfüllt ist) gehören und welche nicht mehr.

**Definition 4.4** *Eine Zeichenkette $\alpha = a_1 \cdots a_n$ Elementen aus $I, D, T$ und $E$ heißt korrekt geschachtelt, wenn die Anzahl der T's der Anzahl der E's entspricht und in jedem Präfix von $\alpha$ höchstens so viele E's vorkommen wie T's.*

Aus einer Betrachtung der obigen Definition von Programmen wird klar, dass Programme per se korrekt geschachtelt sind.

Desweiteren stellen wir fest, dass eine Registermaschine bei terminierenden Rechnungen nur endlich viele Register benutzen kann.

Wir definieren nun Berechenbarkeit einer Registermaschinen analog zur der einer Turingmaschine. Dementsprechend ist eine Funktion f Registermaschinen-berechenbar, wenn es eine Registermaschine gibt, die f berechnet. Einige Beispiele für Registermaschinen-berechenbare Funktionen geben wir hier, den Beweis hierzu findet man beispielsweise in [Smi96, S.16ff].

**Satz 4.5** *Registermaschinen-berechenbar sind die folgenden Funktionen:*

- *die überall undefinierte Funktion u*
- *die Summe $x + y$*
- *das Produkt $x \cdot y$*
- *die Funktion $h(x_1, \cdots, x_{i-1}, y_1, \cdots, y_m, x_{i+1}, \cdots, x_n) := f(x_1, \cdots, x_{i-1}, g(y_1, \cdots y_m), x_{i+1}, \cdots x_n)$ für $f(x_1, \cdots, x_n), g(y_1, \cdots, y_m)$ Registermaschinen-berechenbar*
- *die modifizierte Differenz $x -_m y = \begin{cases} x - y, & falls x > y \\ 0, & sonst \end{cases}$*
- *die Funktion $h(x_1, \cdots, x_n) = \begin{cases} f(x_1, \cdots, x_n), & falls\ \tau\ gilt \\ g(x_1, \cdots, x_n), & sonst \end{cases}$*
  *für $f(x_1, \cdots, x_n), g(y_1, \cdots, y_m)$ Registermaschinen-berechenbar und $\tau$ eine aus berechenbaren Ausdrücken gebildete Bedingung*
- *die Ganzzahl-Division $x \div y := \begin{cases} max\{z : y \cdot z \leq y\}, & falls\ y \neq 0 \\ 0, & sonst \end{cases}$*
- *der Rest der Ganzzahl-Division $x\ mod\ y := \begin{cases} x -_m (x \div y) \cdot y, & falls\ y \neq 0 \\ 0, & sonst \end{cases}$*

*Außerdem lassen sich if-, for- und while-Schleifen auf Registermaschinen realisieren.*

Für den späteren Beweis brauchen wir noch eine Definitionen.

**Definition 4.6** *Sei $prim : \mathbb{N} \to \mathbb{N}$ diejenige Abbildung, welche einer Zahl n die n-te Primzahl zuordnet. Wir setzen zusätzlich $prim(0) = 1$*

Da es unendlich viele Primzahlen gibt ist dies eine Bijektion. Wir zeigen nun:

**Satz 4.7** *Die Funktion $prim$ ist von einer Registermaschine berechenbar.*

*Beweis:* [32] Das folgende Programm P berechnet prim:

$W := 1; I := 1;$
WHILE $I \leq n$ do
$INC(I); PR := 0;$
WHILE $PR = 0$ do
$INC(W); PR := 1;$
$T := 2;$
WHILE $T < W$ do
if $W$ mod $T = 0$ THEN $PR = 0;$
$INC(T)$
$EEE$
$ERGEBNIS := W$

Da wir bisher nur die Bedingung der Form WHILE $a > 0$ kennengelernt haben, beobachten wir zunächst, dass die Bedingung WHILE $I \leq N$ dadurch realisiert werden kann, dass ein weiteres Register eingeführt wird, welches zu Beginn den Wert N hat und bei jeder Iteration durch die while-Schleife dekrementiert wird. Es hat also immer den Wert $N - I + 1$, und es ist $N - I + 1 > 0$ genau dann, wenn $N \geq I$ gilt. W, I, PR und T sind eigene Register, welche mit der Funktion $INC$ inkrementiert werden. Wir hätten sie auch mit $R_i$ bezeichnen können, was jedoch der Übersichtlichkeit geschadet hätte. Die If-Bedingung können wir, nachdem wir ein weiteres Register V mit $V := 1$ eingeführt haben, ersetzen durch WHILE $(W \text{ mod } T = 0, V > 0)$ do $PR := 0, Dec(V)E$.

Nun zeigen wir mittels Induktion, dass die n-malige Iteration über die äußerste Schleife den Wert $prim(n)$ liefert.

*Induktionsanfang:* Für $n = 0$ ist $prim(n) = 1$, die Schleife wird nicht benutzt.

*Induktionsschritt:* Gelte die Behauptung für ein n. Wir betrachten den (n+1)-ten Durchlauf. Hier werden in der zweiten Schleife nacheinander alle Nachfolger von $prim(n)$, die kleiner sind als W, geprüft. Dazu setzen wir PR auf eins (dh. wir inkrementieren das Register PR) und T auf zwei (wir inkrementieren Register W zwei mal).In der dritten Schleife testen wir dann W auf Teilbarkeit, und falls dieser Test positiv ist gehen wir zurück in die zweite Schleife. Ist er negativ, so ist W Primzahl und es werden die zweite und die dritte Schleife beendet und, da wir uns schon im (n+1)-ten Durchlauf befinden, auch die erste. Also hört das Programm auf und gibt die nach $prim(n)$ nächstgrößere Primzahl aus.

Wir werden nun sehen, dass es im Sinne der Berechenbarkeit keinen Vorteil bringt, anstelle der Turingmaschinen Registermaschinen zu verwenden:

**Satz 4.8** *Eine Funktion f ist genau dann (mit einer Turingmaschine) berechenbar, wenn sie Registermaschinenberechenbar ist.*

*Beweis:* [33]
„⇒“: Sei f eine von einer Turingmaschine M berechenbare Funktion.

Wir nehmen an, dass $Q := \{0, \ldots, m\}$ die Menge der Zuständ von M ist, wobei 0 der Startzustand und m der Haltezustand sein sollen. Wir betrachten hier ohne Beschränkung der Allgemeinheit eine Turingmaschine mit einem zweielementigen Bandalphabet $\{0, a\}$ und nur einem Endzustand. Hierbei enthält die Eingabe eine 0 als Trennsymbol zwischen einzelnen Eingabewerten und dem Anfang oder Ende der Eingabe. Insbesondere startet die Maschine auf einer 0 vor dem ersten Eingabewert. Eine Zahl n als Eingabe entspricht also einem String bestehend aus 0,(n+1) hintereinander gereihten a und noch

[32] nach [Smi96, S.27]
[33] nach [Smi96, S.102ff]

einer 0, zwei Zahlen n und m als Eingaben seinen codiert als ein String bestehend aus 0, (n+1) a, 0, (m+1)a und einer weiteren 0. Blank-Symbole kommen so nur außerhalb des eigentlichen Bandinhaltes vor. Das Ergebnis der Rechnung soll der String sein, welcher direkt rechts vom Kopf der Turingmaschine steht wenn diese ihre Rechnungen beendet. Hierbei soll der Kopf auf einer Null stehen, weitere Elemente hinter diesem String sollen irrelevant sein. Steht der Kopf nicht auf einer Null oder terminiert M nicht so soll der Funktionswert nicht definiert sein.

Wir müssen nun zeigen, dass dieses f auch Registermaschinen-berechenbar ist. Dazu stellen wir zunächst fest, dass der jeweilige Zustand, da als feste Zahle angenommen, in einem Register $Q$ gespeichert werden können. Den Bandinhalt verteilen wir auf zwei Register LB und RB, wobei in LB der Inhalt des Bandes links vom Lese- und Schreibkopf, mittels Gödelisierung, als Zahl kodiert gespeichert ist, in RB der Inhalt des rechten Bandes. Der Bandkopf steht dabei zunächst auf dem Feld, das von der kleinsten Primzahl in RB, von 2, kodiert wird. Wenn die Turingmaschine ein Symbol schreiben soll, so entfernen wir von RB zunächst alle Potenzen von 2 (wir teilen die Zahl in RB so lange durch 2 bis dies nicht ohne Rest möglich ist). Schreibt sie das Bandsymbol, multiplizieren wir mit dem Faktor $4 = 2^2$, schreibt sie eine 0 multiplizieren wir mit 2.

Die Bewegungen des Bandkopfes nach links beschreiben wir nun wie folgt, eine Bewegung nach rechts geht dann analog. Ist der Wert y auf LB nicht durch zwei teilbar (die Turingmaschine trifft auf das Ende des Bandes), so müssen nur nichts verändern. Ist LB nicht leer, so teilen wir für den Wert y in LB n-mal durch 2, mit n so groß, dass dies kein weiteres Mal ohne Rest möglich ist. Wir verändern RB wie folgt: Wir berechnen die größten Primzahl $prim(i_{max})$, die in der Zahl x in RB vorkommt. Wir teilen x durch $prim(i_{max})$ und multiplizieren mit $prim(i_{max+1})$, kommt $prim(i_{max})$ noch einmal vor wiederholen wir dies. Dann berechnen wir die zweitgrößte Primzahl, teilen durch diese und multiplizieren mit der nächstgrößeren. Dies setzen wir so lange fort, bis es keine kleinere Primzahl mehr gibt. In x dürfen dann keine Faktoren der Form $2^n, n \in \mathbb{N}$ vorkommen. Nun multiplizieren x schließlich noch n-mal mit zwei, wobei n=1 ist, falls eine 0 gelesen wird und n=2, falls a gelesen wird.

Mit diesen Programmen kann nun jede mögliche Operation der Turingmaschine nachvollzogen werden. Wir bezeichnen mit $Q_i$ das Programm, das die Operationen simuliert, die die Turingmaschine im Zustand i ausführen würde.

Wir benötigen nun noch ein Programm A1, welches den Inhalt der Register in die Anfangskonfiguration unserer simulierenden Turingmaschine übersetzt, und ein Programm A2, das den Funktionswert vom simulierten Turingband in das Resultatregister überträgt. Diese seien hier explizit angegeben:

$A1 : LB := 1, RB := 1, K = 1;$
$P_1 \cdots P_n$ mit
$P_i = RB \cdot prim(K)$
$INC(K);$
WHILE $R_i > 0$ do
$RB := RB \cdot prim(K)^2;$
$INC(K); DEC(R_i)$
$E$

$A2$ : IF $RB[1] = 2$ oder $RB[2] \neq 2$ THEN
WHILE $0 = 0$ do
$E$
ELSE
$R_1 := 0; K := 3;$
WHILE $RB[K] = 2$
do $INC(R_1); INC(K)$
$E$

Hierbei sei $RB[K] = n$ falls die K-te Primzahl n-mal als Faktor in x vorkommt. Dies kann die Maschine leicht überprüfen. Wir beobachten, dass A2 nicht hält, falls

Wir erhalten so das vollständige Simulationsprogramm

$P : A1;$
$Q := 0$
WHILE $Q \neq m$ do
IF $Q = 0$ do $Q_0$
ELSEIF $Q = 1$ do $Q_1$
$\dots$
$EE$
$A2$

„⇐“: Sei f eine durch eine Registermaschine P berechenbare Funktion.

Wir müssen nun zeigen, dass eine Turingmaschine M existiert, die diese Registermaschine simuliert. Zunächst benötigen wir eine Übersetzung der Inhalte der verwendeten Register auf das Turingband. Sei m die Anzahl der verwendeten Register. Wir können die Registerinhalte $x_i$ jedes $R_i$ auf das Turingband schreiben, indem wir zuerst eine Null, dann $x_1 + 1$ viele a auf das Band schreiben, dann eine weitere Null setzen, $x_2 + 1$ viele a schreiben und so weiter bis wir bei $x_m$ angelangt sind. Hiernach schreiben wir noch eine letzte Null.

Wir konstruieren nun für jedes Teilprogramm $P'$ von P eine Turingmaschine $M_{P'}$, welche die Aktionen von $P'$ auf den Registern simuliert und dann wieder in die Ausgangsposition zurückkehrt. Wir konstruieren nun drei Hilfsmaschinen wie folgt:

| $\mathscr{L}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $Lq_0$ | $Lq_1$ | $Lq_1$ |
| $q_1$ | $Lq_0$ | $Rq_H$ | $Rq_H$ |

| $\mathscr{R}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $Rq_0$ | $Rq_1$ | $Rq_1$ |
| $q_1$ | $Rq_0$ | $Lq_H$ | $Lq_H$ |

| $\mathscr{S}_i$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $Lq_0$ | $Rq_1$ | $Rq_1$ |
| $q_1$ | $Rq_1$ | $Rq_2$ | $Rq_2$ |
| $\dots$ | $\dots$ | $\dots$ | |
| $q_{i-1}$ | $Rq_{i-1}$ | $Rq_i$ | |

Die Maschinen $\mathscr{L}$ und $\mathscr{R}$ bewegen den Kopf von jeder beliebigen Position innerhalb des Bandinhalts auf die Null links bzw. rechts davon. Die Maschine $\mathscr{S}_i$ wird auf die erste Null der Bandinschrift angesetzt und hält dann auf dem ersten $a$ des Wertes $x_i$.
Nun können wir die $M_{P'}$ induktiv konstruieren.

*Induktionsanfang:* Hierfür müssen wir drei Fälle unterscheiden:

1.Fall: $P' = \epsilon$ Die leere Anweisung beschreiben wir durch eine Maschine, die sich nicht bewegt und bei der Start- und Haltezustand übereinstimmen.

2.Fall: $P' = I_i$ für $1 \leq i \leq m$. Eine Maschine $M_{P'}$ simulieren wir nach folgender Tabelle:

| $M_{I_i}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $s_0$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ |
| $s_i$ | $Rs_i$ | $aRr$ | $aRr$ |
| $r$ | $0Rr'$ | $Lz$ | $Lz$ |
| $r'$ | $Rr'$ | $aRr$ | $aRr$ |
| $z$ | $\mathscr{L}q_H$ | $\mathscr{L}q_H$ | $\mathscr{L}q_H$ |

Diese Maschine sucht zunächst den Wert von $x_i$ auf dem Band auf, hängt an dessen Ende ein weiteres a an, schreibt dahinter eine Null und verschiebt dann alle nachfolgenden Register um ein Feld nach rechts. Damit ist der Wert von $x_i$ um eins größer, es gibt eine Null zwischen $x_i$ und $x_{i+1}$ und alle anderen $x_j$ sind gleich geblieben. Die Form des Bandinhaltes ist wie gewünscht.

3.Fall: $P' = D_i$ für $1 \leq i \leq m$. Diese Anweisung wird durch die Maschine mit folgender Tabelle umgesetzt:

| $M_{D_i}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $s_0$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ |
| $s_i$ | $Rt$ | $\mathscr{L}$ | $\mathscr{L}$ |
| $t$ | $Rt'$ | $Lz$ | $Lz$ |
| $t'$ | $Rt'$ | $Lu$ | $Lu$ |
| $u$ | $0Ru$ | $Rv$ | $Rv$ |
| $v$ | $\mathscr{R} v'$ | $\mathscr{R} v'$ | $\mathscr{R} v'$ |
| $v'$ | $R$ | $\sqcup Ll$ | $\sqcup Ll$ |
| $l$ | $0Ll'$ | $Lz$ | $Lz$ |
| $l'$ | $Ll'$ | $aLl$ | $aLl$ |
| $z$ | $Lq_H$ | $Lq_H$ | $Lq_H$ |

Analog zur Maschine in Fall 2 wird der Kopf an den Anfang von $x_i$ gebracht, danach wird geprüft, ob $x_i$ aus mehr als einem a besteht, also ob der Wert im Register i schon 0 ist. Falls dies der Fall ist, kann die Registermaschine nicht weiter dekrementieren, deswegen wird unsere Turingmaschine hier mittels Zustand z an den Anfang des Bandes geschickt. Ist der Wert von Null verschieden, so wird der letzte Strich von $x_i$ entfernt und alle Felder rechts davon werden um ein Feld nach links verlagert, sodass wieder nur eine Null zwischen $x_i$ und $x_{i+1}$ vorhanden ist. Dann kehrt die Maschine wieder an den Ausgangspunkt zurück und geht in den Haltezustand über.

*Induktionsschritt:* Wir müssen nun noch zeigen, dass für $Q, R$ Programme auch $QR$ ein Programm ist. Es existieren also zwei Maschinen $M_Q$ und $M_R$ zu zwei Programmen $Q$ und $R$. Dann können wir $M_{QR}$ definieren mittels der Tabelle

| $M_{QR}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $M_Q q_1$ | $M_Q q_1$ | $M_Q q_1$ |
| $q_1$ | $M_R q_H$ | $M_R q_H$ | $M_R q_H$ |

,

benutzen also nacheinander zuerst die Maschine $M_Q$ und danach die Maschine $M_R$.

Zuletzt zeigen wir noch, dass die Schleife $P' = \text{WHILE } R_i > 0 \text{ do } S; E$, welche ganz formal der Ausdruck $T_i S E$ ist, für ein Programm S von einer Turingmaschine simuliert werden kann. Da S ein Programm ist, gibt es nach Induktionsvoraussetzung eine Turingmaschine $M_S$, welche S simuliert. Mit folgender Tabelle können wir nun P' simulieren:

| $M_{T_i S E}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $s_0$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ | $\mathscr{S}_i s_i$ |
| $s_i$ | $aRt$ | $\mathscr{L}$ | $\mathscr{L}$ |
| $t$ | $Lq_{neq}$ | $Lq_{eq}$ | $Lq_{eq}$ |
| $q_{neq}$ | $\mathscr{L} q_S$ | $\mathscr{L} q_S$ | $\mathscr{L} q_S$ |
| $q_S$ | $M_S s_0$ | $M_S s_0$ | $M_S s_0$ |
| $q_{eq}$ | $\mathscr{L} q_H$ | $\mathscr{L} q_H$ | $\mathscr{L} q_H$ |

Diese Maschine prüft, ob $x_i$ schon Null ist oder nicht, wenn dies der Fall ist, so kehrt sie in die Ausgangsposition zurück, ist $x_i$ von Null verschieden, so wird $M_S$ gestartet. Nach $M_S$ wird $x_i$ erneut überprüft und so weiter.

Hiermit ist der Aufbau der Maschine $M_{P'}$ beendet. Wir betrachten nun die Maschine $M_P$, welche alle Teilprogramme P' vereint. Damit alle m Register vertreten sind, müssen wir vor dem Start von $M_P$ noch die Register $n+1$ bis $m$ einrichten und mit Null belegen. Hierzu müssen wir dem Eingabestring noch $m-n-1$ $a$s, getrennt durch jeweils eine Null voneinander und vom Eingabestring, hinzufügen. Dieses Einfügen geht mit der Maschine $\mathscr{T}$, welche durch die folgende Tabelle realisiert wird:

| $\mathscr{T}$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $\mathscr{L}$ | $Rq_1$ | $Rq_1$ |
| $q_1$ | $\mathscr{L}$ | $aRq_2$ | $aRq_2$ |
| $q_2$ | $\mathscr{L}$ | $0Rq_3$ | $0Rq_3$ |
| $q_3$ | $\mathscr{L}$ | $Lq_H$ | $Lq_H$ |

Damit können wir die komplette Maschine M angeben:

| $M$ | $a$ | 0 | $\sqcup$ |
|---|---|---|---|
| $q_0$ | $\mathscr{R}q_{n+1}$ | $\mathscr{R}q_{n+1}$ | $\mathscr{R}q_{n+1}$ |
| $q_{n+1}$ | $\mathscr{T}q_{n+2}$ | $\mathscr{T}q_{n+2}$ | $\mathscr{T}q_{n+2}$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $q_m$ | $\mathscr{T}q_{m+1}$ | $\mathscr{T}q_{m+1}$ | $\mathscr{T}q_{m+1}$ |
| $q_{m+1}$ | $\mathscr{L}s_0$ | $\mathscr{L}s_0$ | $\mathscr{L}s_0$ |
| $s_0$ | $M_P q_H$ | $M_P q_H$ | $M_P q_H$ |

Dieses M hält genau dann, wenn das entsprechende Registermaschinen-Programm P hält. Der Ausgabestring von M enthält dann genau die Ausgabewerte, welche auch die Register von P enthalten. Also sind die Funktionen, die M und P berechnen, für genau dieselben Argumente definiert und liefern die selben Ergebnisse. Dies liefert die Rückrichtung und komplettiert den Beweis.

## 5 Asimovs Robotergesetze

Um sich mit Robotern beschäftigen zu können, muss zunächst einmal der Begriff „Roboter“ definiert werden. Für unsere Zwecke gelte die folgende

**Definition 5.1** *Ein Roboter ist eine Maschine, die autonom handelt. Ein Roboter muss Anweisungen des Menschen befolgen, sich dabei aber an vorher für ihn festgelegte Regeln halten.*

Die physische Gestalt dieser Maschine spielt dabei für uns keine Rolle. Wir nehmen jedoch an, dass dieser Roboter für jede mögliche Berechnung genügend Speicherplatz beschaffen kann und eine Fragestellung auch genügend schnell bearbeiten kann.

Im Jahre 1942 schrieb der Schriftsteller Isaak Asimov mehrere Geschichten, in denen Roboter in Situationen gebracht wurden, in denen sie außer Stande waren den ihnen übertragenen Auftrag zu vollenden. Er stellte dabei, zunächst implizit, die folgenden Robotergesetze auf:[34]

**Robotergesetz Nummer 1** *Ein Roboter darf keinen Menschen verletzen oder zulassen, dass durch seine Untätigkeit ein Mensch zu Schaden kommt.*

Dieses Gesetz setzt insbesondere voraus, dass der Roboter die Folgen seines Handelns absehen kann.

**Robotergesetz Nummer 2** *Ein Roboter muss die Befehle befolgen, die ihm vom Menschen gegeben werden, es sei denn diese Befehle stehen im Konflikt mit Gesetz Nummer 1.*

**Robotergesetz Nummer 3** *Ein Roboter muss seine eigene Existenz schützen, solange dies nicht im Konflikt mit Gesetz Nummer 1 oder 2 steht.*

Manchmal wird noch ein viertes Robotergesetz genannt, welches über obigen drei Gesetzen steht:

**Robotergesetz Nummer 4** *Ein Roboter darf der Menschheit keinen Schaden zufügen oder zulassen, dass die Menschheit durch seine Untätigkeit zu Schaden kommt.*

Dieses letzte Gesetz erlaubt dem Roboter, ein menschliches Individuum zu verletzen, wenn dadurch Schaden von der gesamten Menschheit abgewandt wird.

Es macht Sinn, die Robotergesetze als verschieden wichtig zu erachten. So ist es beispielsweise sinnvoll, den Schutz des Menschen über den Schutz der Maschine zu stellen. Deswegen ist es in der Regel sinnvoll, Gesetz Nummer vier mit oberster Priorität zu versehen, gefolgt von den Gesetzen Nummer eins, zwei und drei. Allerdings gibt es auch Geschichten wie „Runaround“[35], deren Handlung aus einem Konflikt der Gesetze Nummer 2 und 3 besteht, welche als gleichwichtig erachtet werden.

[34] nach [Vow06], vgl. auch [Luk08], [uOE94] und [And]

[35] die man zum Beispiel in [Asi07, S.276ff] nachlesen kann

## 6 Robotergeschichten

In diesem Kapitel werden wir nun Situationen ersinnen, in denen ein Roboter aus rein berechenbarkeitstheoretischen Gründen eine richtige Entscheidung nicht immer treffen kann. Im Gegensatz zu den Geschichten von Asimov soll es dem Roboter hier zumindest theoretisch möglich sein, eine richtige Alternative auszuwählen. Aufgrund der algorithmischen Beschränkungen wird der Roboter diese aber im Allgemeinen nicht erkennen. So wird er in eine Situation gebracht, in der er durch einen Verstoß gegen ein Robotergesetz einen Fehler macht obwohl es eine bessere Lösung gegeben hätte.

Wir nehmen nun zunächst an, dass die Church-Turing-Hypothese gelte.

### 6.1 Weltraumexpedition

Die erste Geschichte beschäftigt sich mit Robot Gaezz17. Er ist ein sehr leistungsfähiger Roboter, der leistungsfähigste seiner Zeit, ist jedoch trotz allem durch eine Turingmaschine gesteuert. Er kann jede beliebige andere Turingmaschine emulieren, ist hierbei allerdings höchstens so schnell wie die Maschine selbst. Weiterhin gehorcht er den Robotergesetzen, in folgender Priorität:

1. Sicherung des Fortbestandes seines Volkes
2. Das Wohl seines Volkes
3. Er muss ihm gegebenen Befehlen gehorchen

*Wir befinden uns viele Jahre nach unserer Zeitrechnung in einem dem unseren ähnlichen Multiversum. Das auf einem blauen Planeten beheimatete Volk der Tudanier hat vor kurzem eine neue Technologie entwickelt, die es möglich macht, dass eine Turingmaschine, indem sie rechnet, aus kosmischem Xin, das im Weltall überall in unbegrenzter Menge vorhanden ist, Energie erzeugt. Nun soll eine Weltraummission starten, die eine Turingmaschine mit einer Eingabe als Energiequelle für den Raketenantrieb nutzt. Diese Mission soll das Volk der Tudanier im Weltraum vertreten und so das Überleben des Volkes, ungebunden an ihren Planeten, bis in alle Ewigkeit sichern. Leider stehen in der Heimat nicht mehr viele Energieressourcen zur Verfügung, weshalb die Hoffnung einer ganzen Art an dieser Mission hängt. Am Tage des Abfluges beobachtet im Hauptquartier der Weltrauminstitution Robot Gaezz17 die Startvorbereitungen. Er ist der beste bisher entwickelte Roboter und die Tudanier vertrauen ihm blind. Allerdings gebietet ihnen ihr Stolz, diese Mission selbst zu organisieren. Nachdem der Countdown abgelaufen ist gibt Schun, der Chef des Unternehmens, das Startsignal und die Rakete hebt ab. Während sie majestätisch gen Himmel gleitet bricht in der Bodenstation plötzlich Panik aus. „Das darf nicht wahr sein!“, brüllt einer der Entwickler, „Wir haben einen Fehler gemacht! Der Beweis, dass die Turingmaschine mit dieser Eingabe nicht hält, ist fehlerhaft!“ Es kommt noch schlimmer. „Da rast ein riesiger Asteroid auf das Raumschiff zu!“, ruft einer der Astronomen, „Den haben wir bisher übersehen. Wir müssen ihn abschießen!“ „Nein“, tönt es aus einer anderen Ecke, „wenn wir ihn abschießen, haben wir unsere letzten Energiereserven aufgebraucht.“ In seiner Verzweiflung wendet sich Schun hilfesuchend an Gaezz17. „Sag uns, was wir tun sollen.“, bittet er. Der Robot fährt mit einem leisen Surren in den betriebsbereiten Zustand. „Was sind die Fakten?“, fragt er in seiner metallisch klingenden Stimme. „Hier ist der Code der Turingmaschine und dies ist die ihre Eingabe.“, sagt Schun, der vor Aufregung zittert, und gibt Gaezz17 diese Eingabe. „Ein Asteroid rast auf das Raumschiff zu. Wenn wir den nicht abschießen ist die Mission dem Untergang geweiht. Aber wenn die Turingmaschine hält ist sie dies sowieso. Dann bräuchten wir auch nicht unseren letzten Treibstoff zu verschwenden. Wir müssen eine Entscheidung treffen, und zwar schnell!“ „Wir haben die Flugbahn des Asteroiden berechnet“, meldet sich Schuns Assistent aufgeregt, „und mit den Daten aus der Versuchsreihe zur Turingmaschine verglichen. Zum Zeitpunkt des Aufpralls läuft sie noch!“ Alle Augen ruhen auf Gaezz17. „Sag uns, was wir tun sollen, Gaezz17“ , bittet Schun, „du bist unsere letzte Hoffnung.“*

Robot Gaezz17 steht nun also vor der Entscheidung, ob der Asteroid abgeschossen werden soll oder nicht. Dies ist abhängig davon, ob die Turingmaschine hält oder nicht. Es gibt die Optionen:

| Entscheidung von Gaezz17 | Turingmaschine | Konsequenz | externe Bewertung der Handlung |
|---|---|---|---|
| Abschießen | hält nicht | Mission glückt | richtig |
| Abschießen | hält | Mission misslingt, Nachteil für Tudanier | falsch |
| Nicht abschießen | hält nicht | Mission misslingt | falsch |
| Nicht abschießen | hält | Energie gespart | richtig |

Angenommen, die Turingmaschine hält. Dann führt das Abschießen des Asteroiden zu unnötigem Energieverbrauch, denn die Mission scheitert in jedem Fall. Folglich sollte dann nicht abgeschossen werden. Angenommen, die Turingmaschine hält nicht. Wenn der Asteroid jetzt nicht abgeschossen wird, scheitert die Mission obwohl dies hätte verhindert werden können. Für den Roboter hat es erste Priorität, die Mission zu retten, da dies das Überleben der Spezies sichert. Wenn ihm dies aber nicht gelingen kann, weil es außer seiner Macht steht, so muss er dafür sorgen, dass zumindest nicht unnötig Energie verbraucht wird, weil diese zum Wohle der Tudanier beiträgt.

Der Roboter muss also in endlicher Zeit entscheiden, ob die Turingmaschine hält oder nicht, er soll das Halteproblem entscheiden. Wenn er die Turingmaschine emuliert, so ist er höchstens so schnell wie die Turingmaschine selbst und kommt per Konstruktion nicht rechtzeitig zu einem Ergebnis, auch wenn die Maschine hält.

Da das Halteproblem unentscheidbar ist, ist es in dieser Situation dem Roboter nicht immer möglich, die richtige Entscheidung zu treffen. Zwar gibt es durchaus Fälle in denen der Roboter beweisen kann, dass eine Turingmaschine hält beziehungsweise nicht hält, und er die richtige Entscheidung trifft, dies ist aber nicht immer gegeben.

Weiter bemerken wir, dass eine Fehlentscheidung eines Roboters im Multiversum der Tudanier nicht immer erkannt wird. Wenn die Turingmaschine für Gaezz17 nicht erkennbar nicht hält, so wird in diesem Multiversum nie auffallen, dass eine Entscheidung des Roboters gegen den Abschuss des Asteroiden ein Fehler war. Eine Entscheidung für den Abschuss bei terminierender Turingmaschine wird jedoch immer als Fehler erkannt. Aus dieser Überlegung ergibt sich eine Strategie für den Roboter um, selbst bei einer Fehlentscheidung, keine Konsequenzen fürchten zu müssen: Solange er nicht beweisen kann, dass die Maschine nicht hält, entscheidet er gegen einen Abschuss. Terminiert die Maschine, hat er so ohnehin richtig gehandelt, terminiert sie nicht beweisbar nicht, so wird dies im Multiversum nie erkannt und Gaezz17 blebt ohne Strafe.

Beschäftigen wir uns nun mit dem Aufbau der Geschichte: Die Grundlage ist ein unentscheidbares, jedoch semi-entscheidbares Problem, welches von einer Turingmaschine, in Form von Roboter Gaezz17, entschieden werden soll. Es ist essentiell, dass der Roboter in beliebiger, jedoch von der Umgebung vorgegebener endlicher Zeit zu einer Entscheidung kommt, da er sonst stets die Turingmaschine samt Eingabe simuliert und erst dann eine Entscheidung fällt, wenn die Simulation terminiert. Die zeitliche Schranke ist hier die Zeit bis zu dem Moment, in dem eine Abwehrrakete gestartet werden müsste, welche den Asteroiden vernichtet.

Die Vorgabe aus der Umgebung, dass die Turingmaschine bis zur möglichen Kollision noch läuft, verhindert, dass der Roboter schlicht die Turingmaschine intern simuliert. Auf diese Weise wird ein Großteil der semi-entscheidbaren Fälle ausgeschlossen, in denen Gaezz17 immer richtig handelt, weil er beweisen kann, dass die Maschine terminiert. Es verbleiben weitere Fälle, in denen Gaezz17 richtig handelt, weil er auf anderem Wege beweisen kann, dass die Turingmaschine hält beziehungsweise nicht hält. Dies ist aufgrund der Unentscheidbarkeit des Halteproblems aber nicht immer gegeben.

Diese Geschichte ermöglicht allerdings nicht die strikte Einhaltung der Robotergesetze, da immer mindestens eines zu Gunsten eines anderen gebrochen werden muss oder per Konstruktion nicht eingehalten werden kann. Das Optimum „Mission retten und Energie sparen“ ist nicht zu erreichen.

Nun betrachten wir einmal ein Szenario, in dem der Roboter abermals das Halteproblem entscheiden soll. Allerdings hat eine Fehlentscheidung seinerseits jetzt eine direkte Folge, beispielsweise indem

ein Biomonitor erlischt, falls Gaezz17 die falsche Entscheidung trifft. In einer solchen Situation wird die richtige Entscheidung im Multiversum selbst direkt erkannt - es ist die Antwort des Roboters, falls der Monitor nicht erlischt und das Gegenteil seiner Antwort, falls der Monitor erlischt. Auf diese Weise lässt sich nun ein Gerät erstellen, welches das Halteproblem löst. Ein solches Gerät steht im Widerspruch zur Church-Turing-Hypothese. Hieraus wird nun auch direkt deutlich, dass sich keine Situationen konstruieren lassen, welche stets ein striktes Einhalten der Robotergesetze zulassen falls die Church-Turing-Hypothese gilt. Ein richtiges „hält“ wird hierbei also stets eine andere Konsequenz haben als ein richtiges „hält nicht“.

Setzt man die Church-Turing-Hypothese als falsch heraus und konstruiert so für das Mulitversum ein Orakel, das immer entscheiden kann, ob eine Turingmaschine hält oder nicht, so lässt sich folgende Situation herstellen, in der die Robotergesetze in ihrer striktesten Form eingehalten werden können.

Für die nächste Geschichte erschaffen wir uns einen Robot Gaezz17, welcher den Robotergesetzen in folgender Priorität gehorcht:

1. Das Wohl der Tudanier und die Sicherung der Arterhaltung

2. Gehorsam gegenüber Befehlen

*Wir befinden uns in einer anderen Welt des Multiversums der Tudanier. Auch in dieser geht es wieder um die Arterhaltung durch eine Weltraummission. Analog zu obiger Geschichte starten nun zwei Teams mit jeweils einem Raumschiff in den Weltraum. Da man wirklich sicher gehen will, dass dieses Unterfangen glückt, werden die beiden Turingmaschinen mit ihren Eingaben vor dem Start noch einmal dem Halteorakel vorgelegt. Das Halteorakel steht allerdings nur für eine Frage zu Verfügung. Daher einigt sich die oberste Kommission des Planeten auf die Frage „Wird mindestens eine dieser Turingmaschinen nicht terminieren?“, da so sichergestellt ist, dass mindestens ein Team überlebt. Das Orakel bejaht diese Frage. Nach dem gleichzeitigen Start der beiden Raumschiffe überkommt die Tudanier die Reue. Man möchte im Falle des Haltens einer der beiden Turingmaschinen die Besatzung in dessen Rakete retten. Hierzu wird Roboter Gaezz17 befragt. Wieder gibt es ein Zeitlimit, das darin besteht, dass die bereits reisende Rakete eingeholt werden muss.*

Der Roboter Gaezz17 soll nun entscheiden, welche der beiden Missionen von einer Rettungskapsel eingeholt werden und dessen Besatzung zurück auf den blauen Planeten befördert werden soll. Entscheidet er sich gegen diejenige, deren Turingmaschine nicht hält, so hat er richtig gehandelt, hält hingegen die Maschine der nicht-geretteten Mission, hat er falsch entschieden. Seine Handlungsmöglichkeiten sind in folgender Tabelle dargestellt:

| Entscheidung von Gaezz17 | Turingmaschine Mission 1 | Turingmaschine Mission 2 | Konsequenz | Bewertung |
|---|---|---|---|---|
| Mission 1 retten | hält nicht | hält nicht | eine Mission glückt | richtig |
| Mission 1 retten | hält nicht | hält | keine Mission glückt, Tudanier sterben | falsch |
| Mission 1 retten | hält | hält nicht | eine Mission glückt, Tudanier gerettet | richtig |
| Mission 2 retten | hält nicht | hält nicht | eine Mission glückt | richtig |
| Mission 2 retten | hält nicht | hält | eine Mission glückt, Tudanier gerettet | richtig |
| Mission 2 retten | hält | hält nicht | keine Mission glückt, Tudanier sterben | falsch |
| keine Mission retten | hält nicht | hält nicht | beide Missionen glücken | richtig |
| keine Mission retten | hält nicht | hält | Tudanier sterben | falsch |
| keine Mission retten | hält | hält nicht | Tudanier sterben | falsch |

An der Tabelle ist zu erkennen, dass für den Roboter die Option „rette keine Mission“ nicht in Frage kommt, da er dann entweder falsch handelt (falls eine der Turingmaschinen hält), oder aber auch durch die Rettung einer Mission keinen Fehler gemacht hätte (falls keine Turingmaschine hält). Er wird also stets eine Mission retten, jedoch unter Umständen die falsche.

Auch hier kann der Roboter nicht immer richtig entscheiden, da das Halteproblem unentscheidbar ist. Allerdings wird im Falle einer richtigen Entscheidung kein Robotergesetz verletzt, weder per Konstruktion noch aufgrund des Handelns des Roboters.

Im Gegensatz zur vorigen Geschichte wird hier nun eine Fehlentscheidung des Roboters stets erkannt. Diese liegt genau dann vor, wenn die Turingmaschine der nicht geretteten Mission nach endlicher Zeit anhält. Jedoch wird nur unter Umständen erkannt, dass der Roboter eine richtige Entscheidung getroffen hat, wenn nämlich die Turingmaschine der geretteten Mission zum Stehen kommt. Hält keine der beiden Turingmaschinen jemals an, so wird die Richtigkeit des Handelns von Gaezz17 nie erkannt. Wir bemerken hier, dass es egal ist, ob in diesem Falle der Roboter eine Mission gerettet hat oder nicht.

Betrachten wir nun eine noch weitere Verschärfung des Szenarios: Das Orakel wird gefragt, ob genau eine Turingmaschine hält und bejaht dieses. Der Roboter hat nun die folgenden Handlungsmöglichkeiten:

| Entscheidung von Gaezz17 | Turingmaschine Mission 1 | Turingmaschine Mission 2 | Konsequenz | Bewertung |
|---|---|---|---|---|
| Mission 1 retten | hält nicht | hält | keine Mission glückt, Tudanier sterben | falsch |
| Mission 1 retten | hält | hält nicht | eine Mission glückt, Tudanier gerettet | richtig |
| Mission 2 retten | hält nicht | hält | eine Mission glückt, Tudanier gerettet | richtig |
| Mission 2 retten | hält | hält nicht | keine Mission glückt, Tudanier sterben | falsch |

In dieser Situation ist es im Multiversum der Tudanier immer möglich, die Handlung des Roboters als richtig oder falsch zu erkennen, da eine der beiden Maschinen nach endlicher Zeit zum Stehen kommt und dann eine Fehlentscheidung beziehungsweise die Richtigkeit des Handelns direkt erkannt wird. Dem Roboter ist es hier nicht möglich, keine Mission zu retten, da er hierbei in jedem Fall einen Fehler begehen würde.

Wenden wir uns nun einer weiteren Geschichte zu, die ein anderes unentscheidbares Problem zugrunde legt. Wieder gelte die Church-Turing-Hypothese.

## 6.2 Die Frage aller Fragen

Auch diese Geschichte handelt wieder von Robot Gaezz17, der, analog zu oben, wieder die Prioritäten

1. Sicherung des Fortbestandes der Spezies der Tudanier
2. Das Wohl der Spezies Tudanier
3. Gehorsam gegenüber Befehlen

besitzt.

*Wir befinden uns im Multiversum der Tudanier. Leider ist das Volk nun in zwei Gruppen gespalten, die einander bekriegen. Anlass des Krieges ist die Frage, welche der beiden Gruppen die wahre Antwort auf die Frage aller Fragen kennt. Um ihre Fehde ein für alle mal beizulegen, setzen sich die beiden Anführer zusammen und wählen ein Team von Wissenschaftlern aus beiden Gruppen aus, die einen Computer entwerfen*

*sollen, welcher die Antwort auf die Frage aller Fragen erkennen kann. Damit sie den Computer nicht zu ihren persönlichen Gunsten programmieren können, soll Polizeirobot Gaezz17, der beste und leistungsfähigste Robot seiner Zeit, entscheiden, ob der entwickelte Computer zweckdienlich ist. Für die Zeit bis zur Fertigstellung wird ein Waffenstillstand vereinbart. Dieser ist zwar stets besser als Krieg, allerdings ermöglicht er aufgrund immer wieder auftretender gegenseitiger Anfeindungen der verfeindeten Parteien den Tudaniern kein angenehmes Leben. Diejenige Gruppe, die am Ende die Antwort auf die Frage aller Fragen kennt, bekommt das Recht, die Vorherrschaft auf dem Planeten zu übernehmen. Polizeirobot Gaezz17 betrachtet die Situation zunächst gelassen. Wenn der Computer erst einmal fertiggestellt ist wird Frieden eintreten, da die Tudanier dann viel zu sehr mit der Suche nach der Antwort auf die Frage aller Fragen beschäftigt sind um sich gegenseitig anzufeinden.*
*Gaezz17 ist bewusst, dass das Optimum erreicht wird, wenn der Computer niemals eine Antwort als richtig akzeptiert. Allerdings ist er sich auch sicher, dass er selbst zerstört würde, sollten die Tudanier herausfinden, dass der von ihm akzeptierte Computer niemals eine Eingabe akzeptiert. Dies könnte ihnen mithilfe eines kleineren Roboters gelingen. Zum Glück von Gaezz17 können sind diese anderen Roboter berechenbarkeitstheoretisch auf keinem besseren Stand er selbst, was bedeutet, dass sie nicht mehr leisten können als er. Wenn Gaezz17 nicht erkennen kann, dass die Turingmaschine niemals akzeptiert, so können es die kleineren Roboter auch nicht. Die Wissenschaftler beginnen mit ihrer Arbeit. Nacheinander bekommt Gaezz17 immer wieder verschiedenste Turingmaschinen vorgelegt.*

Wir beobachten: Gaezz17 wird nie eine Turingmaschine akzeptieren, und so die Tudanier nie aus ihrer unglücklichen Situation befreien können.

Natürlich gibt es auch hier wieder Situationen, in denen der Roboter immer alternativlos richtig handelt. Werden ihm immer nur Turingmaschinen vorgelegt, die bei mindestens einer Eingabe terminieren, dann kann er nicht umhin diese abzulehnen und macht dabei nie einen Fehler. Desweiteren wird er Maschinen, bei denen er erkennen kann, dass sie nie akzeptieren, auch nicht akzeptieren, da er sonst Gefahr läuft eliminiert zu werden. Außerdem wäre dann das gesamte Projekt gescheitert und Krieg würde erneut ausbrechen. Gaezz17 müsste also genau diejenigen Turingmaschinen akzeptieren, bei denen er nicht erkennen kann, dass sie niemals akzeptieren. Dies ist ihm sicherlich nicht möglich, da bei einer solchen Eingabe seine Rechnungen niemals terminieren, denn die charakteristische Funktion der Menge $LP = \{\langle M\rangle : \text{M ist eine Turingmaschine und } L(M) = \emptyset\}$ ist nicht berechenbar.

Auch dieser Geschichte liegt wieder ein unentscheidbares Problem zu Grunde. Im Gegensatz zur ersten Geschichte gilt es hier nicht eine Instanz, sondern hintereinander beliebig viele Instanzen des Problems richtig zu entscheiden. Hierbei trifft Gaezz17, wenn er eine Entscheidung trifft, immer die richtige Wahl. Sein Fehler besteht darin, eine geeignete Maschine nie zu erkennen, wobei er sie auch nie ablehnt.

Eine Fehlentscheidung aus der Sicht von Gaezz17 wird niemals erkannt, da niemand im Multiversum beweisen kann, dass eine im Multiversum nicht-beweisbar niemals akzeptierende Turingmaschine kein Wort akzeptiert. Insofern macht sich Gaezz17 nur vor den Augen der Beobachter von außen schuldig.

## 6.3 Passende Korrespondenzen

In den letzten beiden Geschichten bedeutete das Auftreten einer semi-entscheidbaren Instanz, dass nur ein relatives Optimum erreicht werden kann. Wir wollen nun eine Geschichte konstruieren, bei der diese beweisbaren Instanzen das globale Optimum erreichbar machen. Wir legen hierbei das Post'sche Korrespondenzproblem zu Grunde und nehmen die Church-Turing-Hypothese als wahr an.

Wir überlegen nun: Die Grundmenge muss eine Menge von PCP-Instanzen sein. Unser Roboter Gaezz17 soll eine Menge, bestehend aus endlich vielen geordneten 2-Tupeln, bekommen und entscheiden, ob diese Menge ein Match hat. Wenn dies der Fall ist, so kann der Roboter dieses finden und das globale Optimum soll erreicht sein. Ist dies nicht der Fall, und der Roboter gibt dies auch aus, so soll ein relatives Optimum erreicht sein, beispielsweise indem jetzt eine weitere Menge von 2-Tupeln betrachtet wird. Falls der Roboter eine Menge ablehnt, obwohl es ein Match gibt, oder falls er die Existenz eines Matches proklamiert obwohl keines existiert, so sollen die Robotergesetze verletzt sein. Es macht Sinn,

dem Roboter ein Zeitlimit aufzuerlegen, innerhalb dessen er entscheiden soll, ob sie ein Match hat oder nicht, da er sonst stets versucht durch Ausprobieren ein Match zu finden und im Falle eines für ihn nicht-erkennbar nicht existierenden Matches niemals ablehnt sondern ewig weiterrechnet und hierbei nie einen Fehler macht. Auf diesen Überlegungen basiert die folgende Geschichte:

*Ein weiteres Mal befinden wir uns im Multiversum der Tudanier. Wiederum streben einige Wissenschaftler an, das Leben in ihrem Multiversum zu erhalten, auch wenn der Planet der Tudanier in absehbarer Zeit sein Ende finden wird. Der Plan ist nun, organische Moleküle, von denen jedes aus zwei Teilen, einem A-Teil und einem B-Teil besteht, in einer Raumsonde zu einem dem blauen Planeten ähnlichen Himmelskörper zu schicken. Dort gibt es genügend Nährstoffe, sodass die außerdem mitgeschickten Enzyme die Moleküle beliebig oft reproduzieren können. Die einzelnen Teile eines Moleküls wiederum bestehen aus, nicht unbedingt verschiedenen, organischen Strukturen. Die Wissenschaftler haben herausgefunden, dass die Moleküle untereinander verschiedenste Ketten bilden können. Wenn sich eine Kette bildet, welche im A-Teil dieselben Komponenten in der selben Reihenfolge wie im B-Teil hat, so entsteht in diesem Multiversum neues Leben. Aufgrund der Gesetze des Zufalls wird sich eine solche Kette, sofern sie existiert, auch eines Tages bilden. Dies ist die einzige Möglichkeit, das Leben in dieser Welt zu erhalten, da es aufgrund der langen Reisedauer nicht möglich ist lebende Organismen zu versenden. Die Wissenschaftler extrahieren nun aus verschiedensten Grundstoffen Proben von obigen organischen Molekülen. Da sie hiermit schon zur Genüge beschäftigt sind, beauftragen sie Roboter Gaezz17 damit, die Proben dahingehend zu überprüfen, ob sie sich für das Vorhaben eignen. Eine geeignete Probe muss zu einem gewissen Zeitpunkt t vorliegen, da dann und nur dann die Sternenkonstellation es zulässt die Rakete zu starten. Robot Gaezz17 überprüft nun nacheinander die Proben. Hält er eine für nicht tauglich, so verwirft er sie und geht direkt zur nächsten über. Es sind stets genügend Proben vorhanden, sodass Gaezz17 nie ohne Arbeit ist, egal wieviele Proben er verwirft. Hat er eine gefunden, die er für richtig hält, so kann er diese markieren und ist mit seiner Arbeit fertig. Hat er bis zum Zeitpunkt t keine passende Probe gefunden, so kann er die aktuelle für die Mission wählen oder die Mission absagen, er darf nicht schweigen. Eine abgesagte Mission ist stets besser als eine fehlgeschlagene - denn nun wird wenigstens nicht überflüssig viel Material verschwendet, das man auch hätte anders nutzen können.*

Auf den ersten Blick scheinen in dieser Geschichte die Chancen für Gaezz17 das Optimum zu erreichen gut zu stehen, muss er hierfür doch lediglich eine passende Probe finden. Wenn er eine Probe hat, bei der es ihm schwierig scheint eine geeignete Kette zu finden oder wenn er merkt, dass es keine solche Kette gibt, verwirft er sie. Allerdings läuft er hierbei Gefahr, geeignete Proben zu verwerfen. Trotz allem ist es Gaezz17 nicht immer möglich, seine Aufgabe ordnungsgemäß zu erfüllen. Wenn er vor dem Zeitpunkt t eine Probe akzeptiert, so hat sie sicherlich ein Match und er hat zweifellos richtig gehandelt. Werden ihm nur Proben vorgelegt, die für ihn erkennbar kein Match enthalten, so macht er auch keinen Fehler. Allerdings kann Gaezz17 nicht immer erkennen, dass eine Probe kein Match hat. Wenn er an eine solche Probe gerät kann er nach einer gewissen Rechenzeit diese Probe verwerfen. Dies kann ihm allerdings auch bei einer Probe mit Match passieren, wenn er die Existenz des Matches nicht in dieser gewissen Rechenzeit erkennen kann. Das Problem ist, dass Gaezz17 nicht weiß, wann er aufgeben sollte. Im schlimmsten Falle bleibt er also bei einer Probe hängen, die er hätte verwerfen sollen, und gelangt daher nicht zu einer nachfolgenden passenden Probe. Andernfalls könnte er eventuell eine Probe mit Match fälschlicherweise verwerfen und unter den darauffolgenden Proben keine mit Match vorhanden sein.

Wir überlegen uns noch kurz, ob Gaezz17 in diesem Beispiel von einer Turingmaschine vollständig emuliert werden kann. Dass eine Turingmaschine ein Match in einer Menge finden kann steht außer Frage. Jedoch muss geklärt werden, nach welchen Kriterien Gaezz17 eine Probe verwirft. Zum einen könnte man die Turingmaschine mit einem internen Zähler versehen, der nach einer festen Anzahl von Schritten die Berechnung abbricht. Allerdings ist diese Zahl an Schritten im Vorhinein festgelegt. Denkbar wäre hier auch ein System von zwei Turingmaschinen, von denen die eine die andere überwacht und deren Berechnungen nach bestimmten Kriterien abbricht, beispielsweise indem sie diese Turingmaschine zurücksetzt. Hierfür wären aber zusätzliche Hardware-Funktionen, das Anhalten und Zurücksetzen, not-

wendig. Dieses so konstruierte System kann das Halteproblem umgehen - jedoch trifft auch ein Roboter mit einem solchen System in dieser Geschichte nicht immer die richtige Entscheidung. [36]

Weiterhin bemerken wir: Im Multiversum der Tudanier wird immer erkannt, dass die Akzeptanz einer Probe mit Match richtig ist. Es wird auch erkannt, wenn eine Probe fälschlicherweise verworfen wurde, da ein vorhandenes Match immer in endlicher Zeit gefunden werden kann. Auch erkennbar matchfreie Proben werden als richtig verworfen erkannt. Lediglich für die Proben, welche für Gaezz17 nicht erkennbar kein Match enthalten, wird nie erkannt, ob Gaezz17 richtig gehandelt hat.

Betrachten wir anhand dieser Geschichte einmal die Frage nach der Schuld des Roboters. Gaezz17 hat nicht immer zweifellos selbst Schuld an einer Fehlentscheidung seinerseits. Gerät er beispielsweise an eine Probe, die ein Match hat, welches er aber nicht bis zum Zeitpunkt t nachweisen kann, so bleibt fraglich, ob ein Ablehnen der Probe seinerseits ihm als Fehler anzukreiden ist. Immerhin hat er, da er antworten muss, nach bestem Wissen gehandelt. Desweiteren hat Gaezz17 bei einer Probe, die nicht erkennbar kein Match enthält, keine Chance die richtige Entscheidung zu wissen. Ob er dann für seine Fehler haftbar gemacht werden kann bleibt zu diskutieren. Ohne Zweifel hat er jedoch in diesen Fällen gegen die Robotergesetze verstoßen.

## 7 Schluss

Im letzten Kapitel haben wir Situationen betrachtet, in denen eine Maschine in einen Konflikt gerät, den sie aus eigener Kraft nicht lösen kann. Unter der Voraussetzung, dass die Church-Turing-Hypothese gelte haben wir gesehen, dass es Probleme gibt, die Maschinen und mit ihnen letztlich auch Menschen nicht lösen können. Wir haben erkannt, dass ein solches Dilemma nur von einem Orakel, welches außerhalb des betrachteten Multiversums steht, beweisbar richtig entschieden werden kann, nicht jedoch im Multiversum selbst. Sicherlich lassen sich noch viele weitere solche Situationen konstruieren, ihre Botschaft bleibt jedoch gleich: Maschinen sind fehlbar. Es wird immer ein Problem geben, das unentscheidbar bleibt. Dies ist eine Grenze, die wir auch mit den modernsten und besten Mitteln nicht überschreiten können, egal wieviel besser wir unsere Computersysteme machen - es wird immer ein unentscheidbares Problem bleiben. Diese Aussage erinnert an den ersten Unvollständigkeitssatz von Gödel, wonach jedes hinreichend komplexe arithmetische System, sofern es widerspruchsfrei ist, mindestens einen Satz besitzt, der im System weder beweisbar noch widerlegbar ist.

Bei jedem Computersystem wird es also immer ein Problem geben, bei dem die Maschine nie aufhören wird zu rechnen. Nun stellen wir jedoch fest, dass dies beim menschlichen Gehirn nicht der Fall ist: Unser Gehirn verfügt per se über eine Kontrollinstanz, die verhindert, dass wir endlos ein einer Problemstellung arbeiten. Hiermit überwindet der Mensch das eigene Halteproblem. Aber auch wenn wir Gedankengänge bewusst abbrechen, können wir uns nicht immer sicher sein, ob ein Weiterdenken nicht doch zum Erfolg geführt hätte. Dies bringt uns Menschen dazu, Fehlentscheidungen zu treffen, die hätten vermieden werden können. Somit ist selbst für den Menschen nicht alles entscheidbar.

[36] vgl. [Vow06, S. 103ff]

## Literatur


[And] Susan Leigh Anderson. Asimov's "three laws of robotics" and machine metaethics. *AI & Society - Special Issue: Ethics and artificial agents*.

[Asi07] Isaac Asimov. *Alle Roboter-Geschichten*. Luebbe Verlagsgruppe, 11 2007.

[Blö06] Johannes Blömer. *Skript zur Vorlesung Komplexitätstheorie*. Universität Paderborn, Sommersemester 2006.

[Döp00] Klemens Döpp. *Berechenbarkeit und Unlösbarkeit. Eine kurze Einführung für Informatiker und Mathematiker.* Vieweg Verlagsgesellschaft, 9 2000.

[EFT07] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Einführung in die mathematische Logik*. Spektrum Akademischer Verlag, 5. aufl. edition, 10 2007.

[Gru10] Bücher Gruppe. *Berechenbarkeitstheorie: Turingmaschine, Gödelscher Unvollständigkeitssatz, Church-Turing-These, Berechenbarkeit, Ackermannfunktion (German Edition)*. Books LLC, 7 2010.

[Her78] Hans Hermes. *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit: Einführung in die Theorie der rekursiven Funktionen (Heidelberger Taschenbücher) (German Edition)*. Springer, 3. aufl. edition, 8 1978.

[Luk08] Dusko Lukac. *Realisierung eines ferngesteuerten autonomen mobilen Roboters: Entwurf eines Ausbildungskonzeptes für Robotik (German Edition)*. Diplomica Verlag, 5 2008.

[Mor97] Bernard M. Moret. *The Theory of Computation*. Addison Wesley, 1st edition, 9 1997.

[Sip05] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 2 edition, 2 2005.

[Smi96] Einar Smith. *Elementare Berechenbarkeitstheorie (Springer-Lehrbuch) (German Edition)*. Springer, 1 edition, 4 1996.

[uOE94] Daniel Weld und Oren Etzioni. The first law of robotics (a call to arms). 1994.

[Vow06] B. Vowinkel. *Maschinen mit Bewusstsein: Wohin führt die künstliche Intelligenz? (Erlebnis Wissenschaft) (German Edition)*. Wiley-VCH Verlag GmbH, 5 2006.