1 Maschinenmodelle

NFA	+ PDA	+ TM
Programmdefinition und Arbeitsspeicher in einem (Zustände), konstante Größe; Eingabeband sequentiell	unbeschänkt großer, zugriffsbeschränkter Arbeitsspeicher durch Stack	unbeschänkt großer, „zugriffsunbeschränkter“ Arbeitsspeicher durch Tape; Eingabeband faktisch mit unbeschränktem Zugriff (Eingabe auf Tape schreiben)

1.1 Welches Maschinenmodell ist „richtig“?

Beispielszenarien:

(a) Stream (z.B. Netzwerk) als Eingabe; Verarbeitung mit Programm, das in den Arbeitsspeicher passt und den (konstanten) Rest als Arbeitsspeicher verwendet ⇒ endlicher Automat
denken wir wirklich so ?

(b) Modellierung von Prozeduren ⇒ Kellerautomat
schwacher Kompromiss unbeschränkter Speicher vs. wenig Aussagekraft; „high-level-Denken für einfache Systeme“

(c) high-level-Denken für beliebige Systeme ⇒ Turingmaschine
unbeschränkter Speicher ist eine eigentlich unrealistische Annahme

Realismus (Hardwareentwickler) vs. Abstraktion (Softwaredesigner)

Realität führt zu merkwürdigen Phänomenen
z.B. endlicher Speicher ⇒ endlich viele Programme

Allgemein akzeptierte Meinung: Abstraktion ist die bessere Sicht.

2 Kapitel IV: Nicht berechenbare Funktionen – Unentscheidbare Probleme (97-100)

Eine Menge M heißt *höchstens abzählbar*, wenn sie endlich ist oder wenn sie unendlich und abzählbar ist.
Korollar 1.5: Eine unendliche Menge M ist *abzählbar* gdw. $M = \{\beta(0), \beta(1), \dots\}$ für eine Bijektion $\beta : \mathbb{N} \to M$.
$\mathbb{N}$ ist somit die kleinste unendliche Menge, wenn man nur die Elementzahl betrachtet (es gibt echte Teilmengen, die immer noch unendlich groß sind).

Frage: Ist jede Funktion algorithmisch berechenbar?
Antwort: Nein.
Es gibt abzählbare Mengen, z.B. $\mathbb{N}$, $\mathbb{N} \times \mathbb{N}$.
Es gibt echt größere, überabzählbare Mengen, z.B. $\mathcal{P}(\mathbb{N})$, $\mathbb{N} \to \mathbb{N}$.

Beweis für $\mathcal{P}(\mathbb{N})$:
Annahme: $\mathcal{P}(\mathbb{N})$ ist abzählbar. Dann gibt es eine Aufzählung β mit $\mathcal{P}(\mathbb{N}) = \{\beta(0), \beta(1), \dots\}$. Definiere Diagonalmenge $D := \{n \mid n \notin \beta(n), n \in \mathbb{N}\}$. Wegen $D \subseteq \mathbb{N}$ gilt $D \in \mathcal{P}(\mathbb{N})$. Also gibt es ein n_0, sodass $\beta(n_0) = D$. Aber: $n_0 \in D \overset{D}{\Leftrightarrow} n_0 \notin \beta(n_0) \overset{n_0=D}{\Leftrightarrow} n_0 \notin D$, ein Widerspruch.

Beweis für $\mathbb{N} \to \mathbb{N}$ (Lemma 1.7):
Annahme: $\mathbb{N} \to \mathbb{N}$ ist abzählbar. Dann gibt es eine Aufzählung β mit $\mathbb{N} \to \mathbb{N}= \{\beta(0), \beta(1), \dots\}$. Wir nennen $\beta(n) = f_n$. Definiere Diagonalfunktion $d(n) := f_n(n) + 1$. Nach Definition gilt $D \in \mathbb{N} \to \mathbb{N}$. Also gibt es ein n_0, sodass $d = f_{n_0}$. Aber: $f_{n_0}(n_0) \overset{d=f_{n_0}}{=} d(n_0) \overset{d}{=} f_{n_0}(n_0) + 1$, ein Widerspruch.

3 Probleme mit mächtigen Systemen

Sobald ein System eine gewisse Aussagekraft erlangt, treten unweigerlich unentscheidbare Probleme auf. Prominenteste Beispiele:

(a) Jedes System, in dem man mindestens die Arithmetik der natürlichen Zahlen mit Multiplikation ausdrücken kann, enthält Sätze, die weder bewiesen noch widerlegt werden können (Gödelscher Unvollständigkeitssatz, 1931).

(b) Es gibt keinen Algorithmus, der die Terminierung anderer Algorithmen analysieren kann (Halteproblem, Turing 1937).

(c) Literaturempfehlung: *Gödel, Escher, Bach* (Hofstadter, 1979)

3.1 Das spezielle Halteproblem – Voraussetzungen

Wir benötigen eine Eigenschaft, die wir an ein System stellen müssen, bevor das Halteproblem auftritt:
Man muss Selbstbezüglichkeit darstellen können (z.B. kann eine Formel eine Nummer n erhalten und dann über Formel n reden; Programme können ihren eigenen Programmcode lesen).

Als Beispiel betrachten wir eine simple Programmiersprache Imp.
Wir betrachten das Alphabet $B = \{0, 1\}$ und Programme in der imperativen Programmiersprache Imp.

Ein IMP-*Programmcode* ist ein Binärwort nach den formal definierten Regeln für IMP (wir interessieren uns hier nicht dafür). Es ist entscheidbar, ob ein gegebenes Binärwort ein gültiges IMP-Programm kodiert.

Ein IMP-*Programm* ist die Interpretation von IMP-Programmcode nach den semantischen Regeln für IMP-Programme (wir interessieren uns hier nicht dafür).

Jedes IMP-Programm hat eine Repräsentation in IMP-Programmcode. Jeder IMP-Programmcode beschreibt ein IMP-Programm.

Um zwischen Programmen und ihrem Code zu unterscheiden, benutzen wir Variablennamen (z.B. P) für Programme und denselben Variablennamen mit einem Strich darüber (z.B. $\overline{P}$) für den entsprechenden Programmcode.

IMP-Programme können ausgeführt werden. Sie können ein (Binär-)Wort als Eingabe lesen und ein (Binär-)Wort als Ausgabe schreiben. Folglich können IMP-Programme auch andere IMP-Programme in Form von Programmcode als Eingabe lesen.

3.2 Das spezielle Halteproblem – Beweis

Wir zeigen, dass es kein IMP-Programm P_{magic} gibt, welches den folgenden Algorithmus ausführt:

Eingabe: ein IMP-Programmcode $\overline{P} \in B^*$
Ausgabe: 0 oder 1 als Antwort auf: Terminiert P für die Eingabe $\overline{P}$?

Idee: Wir modifizieren P_{magic} leicht und zeigen dann einen Widerspruch.

Angenommen, P_{magic} existiert. Wir definieren ein neues Programm P', welches zuerst P_{magic} ausführt und bei einer Ausgabe von 1 in eine Endlosschleife geht. Sei `input` das Eingabewort.

```
P'(input):

if (P_magic(input) == 1) {
    while (true) {
        skip; // do nothing
    }
}
```

Wir wissen, dass es zu P' einen entsprechenden Programmcode $\overline{P'}$ gibt. Nun

führen wir $P'(\overline{P'})$ aus. Es gilt:

P' terminiert mit der Eingabe $\overline{P'}$

$\overset{(1)}{\Leftrightarrow}$ P_{magic} terminiert mit der Eingabe $\overline{P'}$ mit Ausgabe 0

$\overset{(2)}{\Leftrightarrow}$ P_{magic} beantwortet die Frage „Terminiert P' für die Eingabe $\overline{P'}$?“ mit „nein“

$\overset{(2)}{\Leftrightarrow}$ P' terminiert nicht mit der Eingabe $\overline{P'}$

nach Definition von (1) P' bzw. (2) P_{magic}

3.3 Analyse

Wir haben gezeigt, dass es kein Programm gibt, das für alle Programme eine Terminierungsanalyse durchführen kann.
Probleme waren:

(a) es gibt unendlich viele Programme

(b) es gibt Programme, die andere Programme simulieren können
also auch Programme, die Programme simulieren, die wiederum Programme simulieren, ...

3.4 Das allgemeine Halteproblem

s. `https://www.youtube.com/watch?v=92WHN-pAFCs`
Angenommen es gäbe eine Maschine H, die zwei Eingaben nimmt:

(a) einen Programmcode $\overline{P}$ für ein Programm P

(b) eine Eingabe u für Programm P

H beantwortet die Frage, ob $P(u)$ terminiert, immer korrekt mit 0 oder 1.

Wir können eine Maschine C bauen, die eine Eingabe v dupliziert, also zweimal v ausgibt ($C(v) = (v, v)$). Wir können eine Maschine I bauen, die für Eingabe 1 nicht terminiert und für 0 terminiert.
Wir definieren die Maschine $X(w) := I(H(C(w))) = I(H(w, w))$. Wir rufen X mit dem Code $\overline{X}$ auf, fragen also, ob Maschine X angesetzt auf ihre eigene Kodierung hält (also $I(H(\overline{X}, \overline{X}))$).

(a) Wenn $H(\overline{X}, \overline{X}) = 0$, so terminiert $X(\overline{X})$.

(b) Wenn $H(\overline{X}, \overline{X}) = 1$, so terminiert $X(\overline{X})$ nicht.