Kapitel 9

Berechenbarkeit

Auf den ersten Blick sieht es nicht so aus, dass man mit IMP sehr viel berechnen kann. Dieser Eindruck ist falsch. In der Tat kann man mit IMP alles berechnen, was überhaupt berechenbar ist. Das liegt im Wesentlichen daran, dass IMP mit beliebig großen Zahlen rechnen kann und dass Zahlen genügen, um den Zustand beliebiger Datenstrukturen zu codieren.

Mithilfe von IMP können wir die grundlegenden Konzepte der Berechbarkeitstheorie formal definieren. Wir zeigen, dass das Halteproblem für IMP unentscheidbar ist und dass die Menge der gültigen Formeln von ASSN nicht testbar ist (Gödels Unvollständigkeitssatz).

Lesematerial

[Winskel, Anhang A]

9.1 Gödelisierung

Wir zeigen jetzt, wie man Paare von Zahlen als Zahlen codieren kann. Dazu geben wir 4 Funktionen

$$\begin{aligned}
\mathit{pair} &\in \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}^+ \\
\mathit{first} &\in \mathbb{Z} \to \mathbb{Z} \\
\mathit{second} &\in \mathbb{Z} \to \mathbb{Z} \\
\mathit{ispair} &\in \mathbb{Z} \to \mathbb{B}
\end{aligned}$$

an, die die folgenden Eigenschaften erfüllen:

(1) $\forall n_1, n_2 \in \mathbb{Z}\colon\ \mathit{first}(\mathit{pair}(n_1, n_2)) = n_1$
(2) $\forall n_1, n_2 \in \mathbb{Z}\colon\ \mathit{second}(\mathit{pair}(n_1, n_2)) = n_2$
(3) $\forall n \in \mathbb{Z}\colon\ \mathit{ispair}(n) = 1 \iff (\exists n_1, n_2 \in \mathbb{Z}\colon\ n = \mathit{pair}(n_1, n_2))$

Wir definieren *pair* wie folgt:

$$\mathit{pair}(n_1, n_2) = 2^{sg(n_1)} \cdot 3^{|n_1|} \cdot 5^{sg(n_2)} \cdot 7^{|n_2|}$$

wobei

$$sg(n) = \text{if } n \geq 0 \text{ then } 1 \text{ else } 0$$

Wegen der Eindeutigkeit der Primzahlzerlegung können wir jetzt die restlichen 3 Funktionen wie gewünscht definieren. Man überzeugt sich leicht, dass alle 4 Funktionen mit IMP berechnet werden können.

Mit derselben Technik können wir auch Tupel mit mehr als zwei Komponenten codieren. Auch Listen lassen sich als Zahlen codieren. Dazu ordnen wir der leeren Liste die Zahl 0 zu und erinnern uns daran, dass nichtleere Listen Paare sind.

Die gerade vorgestellte Codierungstechnik bezeichnet man als *Gödelisierung* (nach dem Logiker Kurt Gödel, der diese Technik entdeckt hat). Die einem Objekt o durch die Codierung zugeordnete Zahl bezeichnen wir mit $\#o$ und nennen sie die *Gödelnummer* von o.

In Kapitel 3 haben wir syntaktische Objekte durch Tupel repräsentiert, die man ausgehend von Zahlen und den Elementen von vorgegeben Mengen wie *Var* oder *Loc* erhalten kann. Wenn wir $\mathit{Loc} = \mathbb{N}$ festlegen, sind alle syntaktischen Objekte von IMP und ASSN Tupel, die man ausgehend von Zahlen konstruieren kann. Das bedeutet, dass wir alle syntaktischen Objekte von IMP und ASSN mithilfe von Gödelisierung als Zahlen codieren können. Wir verwenden die folgenden Notationen:

$$\#\mathit{Com} = \{\, \#c \mid c \in \mathit{Com} \,\}$$
$$\#\mathit{Assn} = \{\, \#A \mid A \in \mathit{Assn} \,\}$$

9.2 Berechenbare Funktionen

Da man immer gödelisieren kann, genügt es, Programme (das heißt Kommandos von IMP) zu betrachten, die zu einer Eingabezahl eine Ausgabezahl berechnen (oder divergieren). Damit wir diese Idee formal fassen können, legen wir eine Lokation

$$X_0 \in Loc$$

fest und nehmen an, dass $\perp$ so gewählt wurde, dass $\perp \notin \mathbb{Z}$. Wir definieren

$$\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$$
$$\sigma_0 = \lambda\, x \in Loc\,.\, 0$$

$$\mathcal{F} \in Com \to \mathbb{Z} \to \mathbb{Z}_\perp$$
$$\mathcal{F}[\![c]\!]n = \text{if } \mathcal{C}[\![c]\!](\sigma_0[n/X_0]) \neq \perp \text{ then } (\mathcal{C}[\![c]\!](\sigma_0[n/X_0]))X_0 \text{ else } \perp$$

Definition 9.2.1 *Eine Funktion $f \in \mathbb{Z} \to \mathbb{Z}_\perp$ heißt* berechenbar *genau dann, wenn es ein $c \in Com$ gibt mit $f = \mathcal{F}[\![c]\!]$.*

Satz 9.2.2 (Universelles Kommando) *Es gibt ein Kommando $U \in Com$, so dass:*

$$\forall c \in Com\ \forall n \in \mathbb{Z}\colon\ \mathcal{F}[\![U]\!](\#(\#c, n)) = \mathcal{F}[\![c]\!]n$$

Beweis Bei U handelt es sich offensichtlich um einen Interpreter für IMP, der in IMP geschrieben ist. Die Konstruktion von IMP gelingt mit Standardtechniken aus der Implementierung von Programmiersprachen. □

Beachten Sie, dass das universelle Kommando wie jedes Kommando nur endlich viele Lokationen enthält. Das bedeutet, dass die Menge der berechenbaren Funktionen nicht kleiner wird, wenn man für *Loc* eine hinreichend große aber endliche Teilmenge von $\mathbb{N}$ verwendet.

Das Standardmodell für die Formalisierung von Berechenbarkeit sind Turing-Maschinen. Alle bisher bekannten Berechnungsmodelle lassen sich durch Turing-Maschinen simulieren. Das gilt auch für Berechnugsmodelle, die realen Computern entsprechen. Daher geht man davon aus, dass alles was berechenbar ist, durch Turing-Maschinen berechenbar ist (sogenannte *Churchsche These*). Berechnungsmodelle, die beliebige Turing-Maschinen simulieren können, nennt man *Turing-vollständig*.

Wir überzeugen uns jetzt davon, dass IMP Turing-vollständig ist. Sei also eine beliebige Turing-Maschine gegeben. Wir können annehmen, dass die Symbole der Turing-Maschine Zahlen sind, da es für die Berechnung einer Turing-Maschine keine Rolle spielt, was für mathematische Objekte die Symbole sind. Das Band repräsentieren wir durch zwei Listen xs und ys, so dass $rev(xs)@ys$ dem Band entspricht und der Kopf auf das erste Element von *xs* zeigt. Mithilfe von Gödelisierung ist es nun einfach, ein Kommando von IMP anzugeben, dass die Turing-Maschine simuliert. Damit haben wir:

Satz 9.2.3 *Es gibt eine endliche Menge $M \subseteq \mathbb{Z}$, so dass IMP mit $Loc = M$ Turing-vollständig ist.*

9.3 Abzählbare Mengen

Seien X und Y Mengen. Eine Funktion $f \in X \to Y$ heißt

- *injektiv* genau dann, wenn für alle $x_1, x_2 \in X$ gilt:

$$x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

- *surjektiv* genau dann, wenn für alle $y \in Y$ ein $x \in X$ existiert mit $f(x) = y$.

Eine Menge M heißt *abzählbar* genau dann, wenn es eine surjektive Funktion $\alpha \in \mathbb{N} \to M$ gibt. Eine abzählbare Menge enthält also mindestens ein Element. Offensichtlich ist jede nichtleere Teilmenge von $\mathbb{Z}$ abzählbar.

Proposition 9.3.1 *Eine nichtleere Menge M ist genau dann abzählbar, wenn es eine injektive Funktion in $M \to \mathbb{N}$ gibt.*

Mithilfe von Gödelisierung bekommen wir injektive Funktionen in $Com \to \mathbb{N}$ und $Assn \to \mathbb{N}$. Folglich sind *Com* und *Assn* abzählbar. Da *Com* abzählbar ist, ist auch die Menge der berechenbaren Funktionen abzählbar.

Proposition 9.3.2 *Die Menge der berechenbaren Funktionen ist abzählbar.*

Proposition 9.3.3 *Die Menge $\mathbb{Z} \to \mathbb{Z}_\perp$ ist nicht abzählbar.*

Beweis Durch Widerspruch. Sei $\alpha \in \mathbb{N} \to (\mathbb{Z} \to \mathbb{Z}_\perp)$ surjektiv. Wir definieren

$$f \in \mathbb{Z} \to \mathbb{Z}_\perp$$
$$f(n) = \text{if } (\alpha n)n = 0 \text{ then } 1 \text{ else } 0$$

Offensichtlich gilt

$$\forall n \in \mathbb{N}\colon\ (\alpha n)n \neq f(n)$$

Das ist ein Widerspruch zu der Annahme, dass α surjektiv ist. □

Die gerade verwendete Beweistechnik wird als *Cantors Diagonalargument* bezeichnet.

Wir wissen jetzt, dass es unendlich viele Funktionen in $\mathbb{Z} \to \mathbb{Z}_\perp$ gibt, die nicht berechenbar sind.

9.4 Entscheidbare und testbare Mengen

Sei $M \subseteq \mathbb{Z}$. Ein Kommando $c \in Com$ heißt

- *Entscheider für* M genau dann, wenn gilt:

  $$\mathcal{F}[\![c]\!] = \lambda\, n \in \mathbb{Z}\,\boldsymbol{.}\ \text{if } n \in M \text{ then } 1 \text{ else } 0$$

- *Tester für* M genau dann, wenn gilt:

  $$M = \{\, n \in \mathbb{Z} \mid \mathcal{F}[\![c]\!]n \neq \perp \,\}$$

Eine Menge $M \subseteq \mathbb{Z}$ heißt

1. *entscheidbar* genau dann, wenn es einen Entscheider für M gibt.
2. *unentscheidbar* genau dann, wenn es keinen Entscheider für M gibt.
3. *testbar* genau dann, wenn es einen Tester für M gibt. Statt testbar sagt man auch *rekursiv aufzählbar*.

Für eine unentscheidbare Menge M können wir kein Programm schreiben, dass für beliebige $N \in \mathbb{Z}$ terminiert und genau dann 1 liefert, wenn $n \in M$. Für eine nicht testbare Menge M können wir kein Programm schreiben, dass für beliebige $N \in \mathbb{Z}$ genau dann terminiert, wenn $n \in M$. Wir werden zeigen, dass es testbare Mengen gibt, die unentscheidbar sind.

Proposition 9.4.1 *Die Mengen #Com und #Assn sind entscheidbar.*

Proposition 9.4.2 *$\mathcal{P}(\mathbb{Z})$ ist nicht abzählbar.*

Proposition 9.4.3 *Es gibt Teilmengen von $\mathbb{Z}$, die nicht testbar sind.*

Proposition 9.4.4 *Seien M_1 und M_2 entscheidbare Mengen. Dann sind die Mengen $M_1 \cup M_2$, $M_1 \cap M_2$ und $M_1 - M_2$ entscheidbar.*

Ein *steuerbarer Tester* für eine Menge $M \subseteq \mathbb{Z}$ ist ein Kommando $c \in Com$ wie folgt:

(1) $\forall n \in \mathbb{Z}\colon\; \mathcal{F}[\![c]\!]n \neq \bot$
(2) $\forall n \in \mathbb{Z}\colon\; n \in M \iff \exists k \in \mathbb{N}\colon\; \mathcal{F}[\![c]\!](\#(n,k)) = 1$
(3) $\forall n \in \mathbb{Z}\; \forall k \in \mathbb{N}\colon\; \mathcal{F}[\![c]\!](\#(n,k)) = 1 \Rightarrow \forall m \geq k\colon\; \mathcal{F}[\![c]\!](\#(n,m)) = 1$

Proposition 9.4.5 *Sei $c \in Com$ ein Tester für M. Dann kann man aus c einen steuerbaren Tester für M konstruieren.*

Beweis Den steuerbaren Tester erhält man dadurch, dass man einen Zähler einbaut, der die Anzahl der Schleifendurchläufe zählt. Zusätzlich werden die Schleifenbedingungen so verstärkt, dass nach Überschreiten der durch das zweite Argument vorgegebenen Maximalanzahl keine weiteren Schleifendurchläufe mehr möglich sind.

Seien Z und S zwei Lokationen, die nicht in c vorkommen. Sei c' das Kommando, das man aus c wie folgt erhält:

1. Ersetze jede Schleifenbedingung b durch $b \wedge Z \leq S$.
2. Ersetze jeden Schleifenrumpf c'' durch $Z := Z + 1\;;\; c''$.

Der steuerbare Tester sieht jetzt wie folgt aus:

$\texttt{if}\; \mathit{ispair}(X_0)\; \texttt{then}\; S := \mathit{second}(X_0);\;\; X_0 := \mathit{first}(X_0)\; \texttt{else}\; S := -1;$
$Z := 0;$
$c';$
$\texttt{if}\; Z \leq S\; \texttt{then}\; X_0 := 1\; \texttt{else}\; X_0 := 0$ □

Proposition 9.4.6 *Seien M_1 und M_2 testbare Mengen. Dann sind die Mengen $M_1 \cup M_2$ und $M_1 \cap M_2$ testbar.*

Beweis Aus steuerbaren Testern für M_1 und M_2 kann man Tester für $M_1 \cup M_2$ und $M_1 \cap M_2$ konstruieren. Dabei erhöht man mit einer Schleife schrittweise die maximale Anzahl von möglichen Schleifendurchläufen und führt jeweils beide steuerbaren Tester aus. □

Das Komplement einer Menge $M \subseteq \mathbb{Z}$ bezüglich $\mathbb{Z}$ bezeichnen wir mit

$$\overline{M} \stackrel{\text{def}}{=} \mathbb{Z} - M$$

Proposition 9.4.7 *Eine Menge $M \subseteq \mathbb{Z}$ ist genau dann entscheidbar, wenn M und $\overline{M}$ testbar sind.*

Proposition 9.4.8 *Sei $M \subseteq E \subseteq \mathbb{Z}$ und sei E entscheidbar. Dann:*

$$\overline{M} \text{ testbar} \iff \overline{M} \cap E \text{ testbar}$$

Beweis Gilt da $\overline{M} = \overline{E} \cup (\overline{M} \cap E)$. □

9.5 Unentscheidbarkeit des Halteproblems

Wir definieren zwei Mengen:

$$\#H \stackrel{\text{def}}{=} \{\, \#c \mid c \in Com \wedge \mathcal{F}[\![c]\!](\#c) \neq \bot \,\}$$
$$\#H_0 \stackrel{\text{def}}{=} \{\, \#c \mid c \in Com \wedge \mathcal{F}[\![c]\!](0) \neq \bot \,\}$$

Wir werden zeigen, dass beide Menge unentscheidbar sind. Diese Tatsache bezeichnet man als die Unentscheidbarkeit des Halteproblems.

Proposition 9.5.1 *Die Mengen $\#H$ und $\#H_0$ sind testbar.*

Beweis Folgt aus Proposition 9.4.1 und Satz 9.2.2. □

Proposition 9.5.2 *Die Menge $\overline{\#H}$ ist nicht testbar.*

Beweis Durch Widerspruch. Sei T ein Tester für $\overline{\#H}$. Dann haben wir den folgenden Widerspruch:

$$\begin{array}{lll} \mathcal{F}[\![T]\!](\#T) = \bot & \iff \#T \notin \#H & \text{Definition von } \#H \\ & \iff \#T \in \overline{\#H} & \\ & \iff \mathcal{F}[\![T]\!](\#T) \neq \bot & T \text{ Tester für } \overline{\#H} \end{array}$$

□

Satz 9.5.3 (Halteproblem) *Die Menge $\overline{\#H_0}$ ist nicht testbar.*

Beweis Durch Widerspruch. Sei T ein Tester für $\overline{\#H_0}$. Wir definieren:

$$g \in \mathbb{Z} \to \mathbb{Z}_\bot$$
$$g(n) = \begin{cases} \#(X_0 := \#c \;;\; c) & \text{falls } c \in Com \text{ und } n = \#c \\ \bot & \text{sonst} \end{cases}$$

Sei G ein Kommando, dass die Funktion g berechnet. Wir wählen G so, dass bei Termination alle von X_0 verschiedenen Lokationen, die in T vorkommen, auf 0 gesetzt werden.

Für alle $c \in Com$ gilt:

$$\begin{array}{lll}
\#c \in \overline{\#H} & \iff \mathcal{F}[\![c]\!](\#c) = \bot & \text{Definition von } \#H \\
& \iff \mathcal{F}[\![X_0 := \#c\ ;\ c]\!]0 = \bot & \\
& \iff \#(X_0 := \#c\ ;\ c) \in \overline{\#H_0} & \text{Definition von } \#H_0 \\
& \iff \mathcal{F}[\![T]\!](\#(X_0 := \#c\ ;\ c)) \neq \bot & T \text{ Tester für } \overline{\#H_0} \\
& \iff \mathcal{F}[\![T]\!](\mathcal{F}[\![G]\!](\#c)) \neq \bot & \text{Definition von } G \\
& \iff \mathcal{F}[\![G; T]\!](\#c) \neq \bot &
\end{array}$$

Also ist $G; T$ ein Tester für $\overline{\#H} \cap \#Com$. Also ist $\overline{\#H}$ wegen Proposition 9.4.8 testbar. Das widerspricht Proposition 9.5.2. □

9.6 Gödels Unvollständigkeitssatz

Sei $M \subseteq \mathbb{Z}$ eine testbare Menge. Dann ist ein Tester für M eine *endliche* Repräsentation von M. Diese Eigenschaft ist interessant, wenn M eine unendliche Menge ist. Am Anfang des zwanzigsten Jahrhunderts vertraten viele Logiker die Ansicht, dass die Menge

$$\#G \stackrel{\text{def}}{=} \{\,\#A \mid A \in Assn \wedge \models A\,\}$$

testbar und damit endlich repräsentierbar ist. Der Logiker Kurt Gödel zeigte um 1930, dass dies nicht der Fall ist. Dieses für die Logik fundamentale Ergebnis ist als Gödels Unvollständigkeitssatz bekannt. Aus Gödels Unvollständigkeitssatz folgt, dass kein Computerprogramm das Wissen über die Zahlen, das man mit den gültigen Formeln von ASSN formulieren kann, vollständig repräsentieren kann.

Satz 9.6.1 (Gödels Unvollständigkeitssatz) *Die Menge $\#G$ ist nicht testbar.*

Beweis Durch Widerspruch. Sei $T \in Com$ ein Tester für $\#G$. Weiter sei w eine SVB-Funktion für IMP. Wir definieren:

$$z \in Com \to Com$$
$$z(c) = \text{let } \{Y_1, \dots, Y_n\} = LC(c) \text{ in } (Y_1 := 0;\ \dots\ ;\ Y_n = 0)$$

Dabei liefert $LC(c)$ alle Lokationen, die in c vorkommen. Dann gilt für alle $c \in Com$:

$$\begin{array}{lll}
\#c \in \overline{\#H_0} & \iff \mathcal{F}[\![c]\!]0 = \bot & \text{Definition von } \overline{\#H_0} \\
& \iff \forall \sigma \in \Sigma\colon\ \mathcal{C}[\![z(c); c]\!]\sigma = \bot & \text{Definition von } \mathcal{F} \\
& \iff \models w(z(c); c)(\texttt{false}) & w \text{ ist SVB-Funktion} \\
& \iff \mathcal{F}[\![T]\!](\#(w(z(c); c)(\texttt{false}))) \neq \bot &
\end{array}$$

In Kapitel 8 haben wir gezeigt, dass man für jedes Paar (c, B) eine schwächste Vorbedingung konstruieren kann (in dem Sinne, dass sich die Konstruktion beispielsweise mit einem in Standard ML geschriebenen Programm ausführen lässt). Daraus folgt, dass man $\#(w(z(c);\, c)(\texttt{false}))$ im formalen Sinne aus $\#c$ berechnen kann. Damit haben wir einen Tester für $\overline{\#H_0} \cap \#Com$. Also ist $\overline{\#H_0}$ wegen Proposition 9.4.8 testbar. Das widerspricht Satz 9.5.3. □

Korollar 9.6.2 *Die Menge $\overline{\#G}$ ist nicht testbar.*

Beweis Durch Widerspruch. Sei T ein Tester für $\overline{\#G}$. Dann gilt für alle $A \in Assn$:

$$
\begin{array}{lll}
\#A \in \#G & \iff \models A & \text{Definition von } \#G \\
 & \iff \models \forall A & \\
 & \iff \not\models \neg\forall A & \\
 & \iff \#(\neg\forall A) \in \overline{\#G} & \text{Definition von } \#G \\
 & \iff \mathcal{F}[\![T]\!](\#(\neg\forall A)) \neq \bot & T \text{ Tester für } \overline{\#G}
\end{array}
$$

Dabei bezeichnet $\forall A$ eine Formel $\forall X_1 \ldots \forall X_n \colon A$ mit $FV(A) = \{X_1, \ldots, X_n\}$. Die obigen Äquivalenzen geben uns einen Tester für $\#G \cap \#Assn$. Also ist $\#G$ wegen Proposition 9.4.8 testbar. Das widerspricht Satz 9.6.1. □