Das Konzept der Berechenbarkeit

Wir gehen von einem Alphabet A aus und einer Sprache $L \subset A^*$, deren Elemente wir Programme nennen. Zu jedem Programm $u \in L$ gehört eine Menge $\Delta(u)$ auf der das Programm u operiert. Z. B. für

$$\begin{array}{rcl} u & := & \text{(Deklaration: } x \text{ integer;} \\ & & \text{Anweisung:} \\ & & 1: \ \text{while } x > 0 \text{ do } x := x - 1; \\ & & 2: \ \text{end),} \end{array}$$

ist $\Delta(u) = \mathbb{Z} \times \{1, 2\}$. $\Delta(u)$ wird durch den Deklarationsteil von u definiert. Der Anweisungsteil definiert auf $\Delta(u)$ eine Abbildung $\alpha(u)$, wenn das Programm „korrekt" ist. Wir haben also

$$\alpha(u) : \Delta(u) \longrightarrow \Delta(u).$$

Die zu u gehörige Berechnung besteht in der Iteration von $\alpha(u)$. Ist also $\xi_0 \in \Delta(u)$, dann ist

$$\xi_0, \xi_1 := \alpha(u)(\xi_0), \ldots, \xi_{i+1} := \alpha(u)(\xi_i), \ldots$$

die zu (ξ_0, u) gehörige Berechnung.

Ist für ein $t \in \mathbb{N}$

$$\xi_t = \xi_{t+1}, \text{ dann ist } \xi_t = \xi_{t+l} \text{ für } l \in \mathbb{N}.$$

In diesem Fall sagen wir, daß die Berechnung **abbricht** oder **terminiert**. Ist t minimal mit dieser Eigenschaft, dann heißt t die **Länge** der Berechnung. ξ_t heißt das **Resultat** der Berechnung.

Wir schreiben in diesen Fällen

$$T(u, \xi_0) := t, \; Res(u, \xi_0) := \xi_t.$$

Die durch u definierte Funktion F_u wird definiert durch:

$$\begin{aligned} \Omega(F_u) &:= \{\xi_0 \in \Delta(u) \mid \text{die zu } (u, \xi_0) \\ &\qquad \text{gehörige Berechnung terminiert}\} \\ F_u(\xi_0) &:= \xi_t. \end{aligned}$$

Also u definiert die Abbildung

$$F_u : \Omega(F_u) \longrightarrow \Omega(F_u).$$

Im allgemeinen ist der Raum $\Delta(u)$ wesentlich größer als der Definitionsbereich von F_u, da $\Delta(u)$ gewöhnlich Komponenten enthält,die nur für die Organisation der Berechnung wichtig sind. Hierauf nehmen wir Rücksicht, indem wir zwei **einfache** Funktionen **in** $=$ „Eingabe“ und **out** $=$ „Ausgabe“ hinzufügen, so daß wir L-berechenbar wie folgt definieren

$$
\begin{array}{ccccc}
 & M & \xrightarrow{f} & M' & \\
\text{in} & \downarrow & & \uparrow & \text{out} \\
 & \Delta(u) & & \Delta(u) & \\
 & \cup & & \cup & \\
 & \Omega(u) & \xrightarrow{F_u} & \Omega(u) &
\end{array}
$$

kommutativ ist.

Dieser Berechenbarkeitsbegriff enthält die folgende Schwierigkeit: Starten wir die Berechnung mit ξ_0, dann wissen wir nicht, ob die Berechnung abbricht. Bricht die Berechnung ab, dann haben wir Glück gehabt, bricht sie nicht ab, dann warten wir ∞-lange vergeblich. Es stellt sich daher die Frage, ob man das nicht vorher wissen kann. Wir fragen ja gar nicht nach dem Resultat, sondern nur danach, ob wir eines erhalten werden.

Haben wir uns für die Sprache L entschieden, dann lautet diese Frage: Gibt es ein Programm $u \in L$, so daß dieses Programm für alle Programme $v \in L$ und $\xi_0 \in \Delta(v)$ entscheiden kann, ob die zu (v, ξ_0) gehörige Berechnung abbricht. Dazu benötigen wir offensichtlich eine Kodierung von (v, ξ_0) durch ein Element aus $\Delta(u)$. Wir wollen uns über die Konstruktion von γ nicht den Kopf zerbrechen, bevor wir wissen, ob es eine solche Kodierung überhaupt geben kann.

Wir nehmen an, daß $L \subset \{0,1\}^*$ ist. Wir können das ohne Einschränkung der Allgemeinheit tun, da sich X^* für jedes X, das höchstens abzählbar-unendlich groß ist, isomorph auf ein Untermonoid von $\{0,1\}^*$ abbilden läßt. Ist $X = \{x_1, \ldots, x_n\}$, dann definiert

$$h(X_i) := 1\underbrace{0\ldots0}_{i}$$

einen solchen Isomorphismus.

Alle Daten werden in unseren Computern binär abgelegt, so daß wir auch annehmen dürfen, daß

$$\Delta(v) = X^* \quad \text{für alle } v \in L$$

und

$$\alpha(v) : X^* \longrightarrow X^*$$

die durch v definierte Abbildung ist, deren Iteration die Berechnung beschreibt.

Für gewisse Argumente mag die Berechnung abbrechen, für andere nicht. Wir fragen nun: Gibt es eine Kodierung

$$\gamma : L \times X^* \longrightarrow X^*$$

und ein Programm u mit den Eigenschaften (1) und (2)?

(1) $F_u(\gamma(v,w)) = 1 \iff \gamma(v,w) \notin \Omega(v)$

(2) $\gamma(v,w) \in \Omega(v) \iff \gamma(v,w) \notin \Omega(u)$.

Das Programm u schreibt also eine 1 aus, wenn v auf w angesetzt nicht terminiert. In allen anderen Fällen terminiert u nicht. Wir werden zeigen

Lemma: $u \notin L$.

Beweis: Annahme $u \in L$.

$$\Rightarrow \quad F_u(\gamma(u,u)) = 1 \Leftrightarrow \gamma(u,u) \notin \Omega_u.$$

Widerspruch!

Beantwortet die Konstruktion von u die gestellte Frage? Wenn wir ein Programm für unsere Maschine haben, das entscheidet, ob $v \in L$ auf w angesetzt hält oder nicht, dann muß sie in jedem Falle anhalten, da wir die gewünschte Information im Falle des Nichtanhaltens nicht bekommen. Die Maschine muß also auch anhalten, wenn die Eingabe keine Kodierung $\gamma(v, w)$ ist. Verlangen wir nun von der Programmiersprache L, daß sie in dem unten definierten Sinne vollständig ist, dann können wir u so abwandeln, daß u die Voraussetzung des Lemmas erfüllt.

Definition: L heißt **schwach vollständig** $:\iff$ Zu jedem Programm $v \in L$ und $c \in X^*$ gibt es ein Programm $v' \in L$ mit folgenden Eigenschaften:

- $F_v(w) = c \quad \Rightarrow \quad w \notin \Omega(v')$,
- $F_v(w) \neq c \quad \Rightarrow \quad F_{v'}(w) = F_v(w)$.

Die Annahme, daß L schwach vollständig ist, ist eine plausible Annahme, da sie nur voraussetzt, daß man in dem Fall, daß ein Programm mit bestimmtem Wert terminiert, das Programm durch eine Prozedur erweitern kann, die im Falle dieser Terminierung z. B. die alternierende Folge $0, 1, 0, 1, \ldots$ als Ausgabe erzeugt.

Haben wir also ein Programm $\tilde{u}$, das auf $\gamma(v, w)$ angesetzt eine 1 ausgibt, falls v auf w angesetzt nicht anhält, und in allen anderen Fällen 0, dann gibt es auch ein Programm $u \in L$, das im Falle der Ausgabe 0 nicht anhält und sonst den gleichen Wert wie $\tilde{u}$ liefert. Also gilt

Satz: Schwach vollständige Programmiersprachen L enthalten kein Programm, das das Halteproblem für Programme v aus L entscheidet.

Es kann zu Sprachen L_0 Programme u_0 geben, die das Halteproblem für Programme aus L_0 entscheiden. Erweitert man diese Sprachen, indem man u_0 zu L_0 hinzunimmt, so erhält man eine Sprache L_1, die außer u_0 noch weitere zusätzliche Programme enthält, wenn L_0 z. B. ein Prozedurkonzept einschließt. Indem wir nun den Satz auf L_1 anwenden, kommen wir in dem Fall, daß ein Programm u_1 existiert, das das Halteproblem für L_1 löst, zu einer Sprache L_2 und schließlich zu einer ganzen Hierarchie

$$L_0 \subsetneq L_1 \subsetneq L_2 \subsetneq \ldots$$

von Sprachen, die auch ∞-groß werden kann.

Geht man davon aus, daß man eine Sprache L_u vorliegen hat, die universell in dem Sinne ist, daß sich jede berechenbare Funktion in ihr programmieren läßt, dann ist also die Funktion, die das Halteproblem für L_u löst, keine berechenbare Funktion.

Wir erläutern die Idee der Konstruktion einer Funktion, die nicht in L liegt, etwas näher.

Sei M eine Menge, $O \subset 2^M$ und

$$F \subset \{f : M_1 \longrightarrow \{0,1\} \mid M_1 \in O\}$$

Es sei $\tilde{M} \subset M$ und

$$\gamma : \tilde{M} \longrightarrow F$$

eine Abbildung. Ist $m \in \tilde{M}$ und $\gamma(m) = f$, dann sehen wir m als eine Bezeichnung für f an.

Nun betrachten wir die Abbildung $u : \tilde{M} \longrightarrow \{0,1\}$, die durch

$$u(m) := \begin{cases} 1 & \text{für } m \notin Q(\gamma(m)) \text{ oder } \gamma(m)(m) \neq 1 \\ 0 & \text{für } m \in Q(\gamma(m)) \text{ und } \gamma(m)(m) = 1 \end{cases}$$

definiert wird.

Lemma: $u \notin F$ oder γ ist nicht surjektiv.

Beweis: Annahme: $u \in F$ und γ ist surjektiv. Ist $u \in F$, dann gibt es ein $m_0 \in \tilde{M}$ mit $\gamma(m_0) = u$, und also $Q(\gamma(m_0)) = \tilde{M}$.

- Ist $u(m_0) = 1$, dann ist $m_0 \notin \tilde{M} = Q(\gamma(m_0))$,oder es ist $u(m_0) \neq 1$ im Widerspruch zur Annahme.

- Ist $u(m_0) = 0$, dann ist $m_0 \in \tilde{M}$ und $\gamma(m_0)(m_0) = 1$ im Widerspruch zur Annahme.

Also ist die Annahme

$$u \in F \text{ und } \gamma \text{ surjektiv}$$

nicht erfüllt.

Wir betrachten eine zweite Version dieses Lemmas.

Lemma: Sei

$$M_F := \{m \in M \mid m \notin Q(\gamma(m))\}$$

und

$$u_F : M_F \longrightarrow \{1\}.$$

Dann gilt γ ist nicht surjektiv oder $u_F \notin F$.

Beweis: Wir nehmen an: γ ist surjektiv. Ist $u_F \in F$, dann finden wir ein $m_0 \in \tilde{M}$ mit $\gamma(m_0) = u_F$.

Falldiskussion:

(a) $m_0 \in M_F \Rightarrow m_0 \notin (\gamma(m_0)) = M_F$. Widerspruch.

(b) $m_0 \notin M_F \Rightarrow m_0 \in Q(\gamma(m_0)) = M_F$. Widerspruch.

Also ist mindestens eine unserer beiden Annahmen falsch.

Die beiden Konstruktionen enthalten als wesentliches Element

- Die Bezeichnung γ der Abbildungen aus F durch Elemente aus M.

- Die Anwendung von $\gamma(m)$ auf m.

Beide Lemmata geben also Verfahren an, Abbildungen zu konstruieren, die nicht in F liegen. Dazu muß es nur möglich sein, eine Teilmenge $\tilde{M} \subset M$ surjektiv auf F abzubilden.

Die Konstruktion der „Haltefunktion“ u zur Sprache L ist eine Anwendung des ersteren der beiden Lemmata. Zur Erläuterung geben wir noch zwei Anwendungen.

Anwendung 1: Die Menge der reellen Zahlen ist nicht aufzählbar.

Beweis: Wähle $M = \mathbb{N}_0$ und

$\gamma : \mathbb{N}_0 \longrightarrow F := \{a : \mathbb{N}_0 \to \{0,1\} \mid a \text{ Abbildung}\}$.

Wir können F vermöge der Definition

$$\overline{a} := \sum_{i \in \mathbb{N}_0} \frac{a(i)}{2^{i+1}}, \quad a \in F$$

als die Menge der reellen Zahlen aus $[0,1]$ ansehen.

Wir setzen nun wie in dem ersten Lemma

$$u(i) \neq \gamma(i)(i) \quad \text{für } i \in \mathbb{N}_0.$$

Offensichtlich ist $u \in F$. Also ist, wie das Lemma besagt, γ nicht surjektiv. Also ist $[0,1]$ nicht aufzählbar.

Das Lemma ergibt hier also das **Cantor'sche Diagonalisierungsverfahren**.

Anwendung 2: Die Antinomie des Barbiers

Nach griechischer Überlieferung ist ein Barbier ein Mann, der die Athener barbiert, die sich nicht selbst barbieren.

Sei M die Menge der vollständigen Namen der Athener. $\gamma(m)$ ordnet m seine Barbierfunktion zu.

Es sei

$$\gamma(m)(m') = 1 \iff m \text{ barbiert den } m!$$

$\gamma(m)$ ist also für alle Athener definiert und ist surjektiv.

Wir definieren nun den Barbier u :

$$u(m) = 1 :\iff \gamma(m)(m) = 0$$

Unser zweites Lemma ergibt nun $u \notin F$, da γ surjektiv ist.

Wir sehen, daß die Idee der hier verwendeten Konstruktion etwa 2000 Jahre alt ist.

Wir werden nun in den folgenden Abschnitten verschiedene Konzepte der Berechenbarkeit entwickeln und sie miteinander vergleichen. Dabei werden wir auch wiederholt auf die hier geschilderten Grundrezepte, das Konzept der Iteration einer Abbildung und der Diagonalisierung zur Unterscheidung von Klassen von Abbildungen zurückbekommen. Wissen wir nämlich, daß $L \subset L'$ gilt, und gilt für die „Haltefunktion“ u zu L, $u \in L'$, dann ist $L \subsetneq L'$.