Berechenbarkeitstheorie

Prof. Dr. Christoph Walther / Nathan Wasser
Technische Universität Darmstadt — Sommersemester 2011

Lösungsvorschlag zu Übung 1

Lösungsvorschlag

Aufgabe 1.1 intuitive Berechenbarkeit

Sind die folgenden Funktionen im intuitiven Sinne berechenbar? Für welche kann man sogar einen Algorithmus explizit angeben? Dabei seien $g : \mathbb{N} \to \mathbb{N}$ eine totale und berechenbare Funktion und $h : \mathbb{N} \mapsto \mathbb{N}$ eine nicht berechenbare Funktion.

1. $f_1(n) = \begin{cases} 1 & \text{, falls } n \text{ eine Primzahl ist} \\ 0 & \text{, sonst} \end{cases}$

2. $f_2(n) = \begin{cases} 1 & \text{, falls es eine Primzahl } p \geq n \text{ gibt} \\ 0 & \text{, sonst} \end{cases}$

3. $f_3(n) = \begin{cases} 1 & \text{, falls es einen Primzahlzwilling } (p, p+2) \text{ mit } p \geq n \text{ gibt} \\ 0 & \text{, sonst} \end{cases}$

   Hinweis: $(p, p+2)$ heißt *Primzahlzwilling*, falls sowohl p als auch $p+2$ Primzahlen sind. Es ist nicht bekannt, ob es unendlich viele Primzahlzwillinge gibt.

4. $f_4(n) = \begin{cases} 1 & \text{, falls } g(n) = n \\ 0 & \text{, sonst} \end{cases}$

5. $f_5(n) = \begin{cases} 0 & \text{, falls } n = 0 \\ h(n) & \text{, sonst} \end{cases}$

6. $f_6(n) = \begin{cases} 0 & \text{, falls } n = 0 \\ h(f_6(n-1)) & \text{sonst} \end{cases}$

7. $f_7(n) = \begin{cases} n & \text{, falls es ein } m \text{ gibt mit } h(m) \leq m \\ 0 & \text{, sonst} \end{cases}$

Lösungsvorschlag

1. f_1 ist berechenbar. Es existieren Verfahren, die testen, ob ein gegebenes $n \in \mathbb{N}$ eine Primzahl ist (z. B. Sieb des Eratosthenes). Somit kann man einen entsprechenden Algorithmus explizit angeben.

2. Es gibt unendlich viele Primzahlen, deshalb ist $f_2(n) = 1$ für alle $n \in \mathbb{N}$. Für diese konstante Funktion lässt sich sicher ein Algorithmus angeben. Damit ist f_2 auch berechenbar.

3. f_3 ist berechenbar. Zurzeit können wir allerdings keinen entsprechenden Algorithmus für f_3 angeben. Wir müssen zwei Fälle unterscheiden:

   Fall 1: Es gibt unendlich viele Primzahlzwillinge. Dann ist $f_3(n) = 1$ für alle $n \in \mathbb{N}$, d. h. f_3 ist berechenbar.

   Fall 2: Es gibt nur endlich viele Primzahlzwillinge. Dann existiert ein größter Primzahlzwilling $(p_0, p_0 + 2)$. Damit ist f_3 berechenbar, denn

   $$f_3(n) = \begin{cases} 1 & \text{falls } n \leq p_0 \\ 0 & \text{falls } n > p_0 \end{cases}.$$

4. f_4 ist berechenbar. Ein Algorithmus zur Berechnung von f_4 könnte zunächst $g(n)$ bestimmen und das Ergebnis mit n vergleichen. Da g berechenbar ist, existiert ein Algorithmus für g. Aufgrund der Totalität von g liefert $g(n)$ immer auch einen definierten Wert.

5. f_5 ist nicht berechenbar. Wäre f_5 berechenbar, dann wäre auch h berechenbar. Ein Algorithmus für h könnte wie folgt aussehen:

   Zunächst überprüft man, ob $n = 0$ ist. Falls $h(0)$ undefiniert ist, geht man bei $n = 0$ in eine Endlosschleife. Ist hingegen $h(0) = m$ für ein $m \in \mathbb{N}$, dann liefert man bei $n = 0$ den Wert m zurück. Im Fall $n \neq 0$ ruft man $f_5(n)$ auf. Dieser Algorithmus würde h berechnen, was nach Voraussetzung nicht sein kann.

6. Die Berechenbarkeit von f_6 hängt von der Funktion h ab. In jedem Fall gilt $f_6(n) = h^n(0)$.

   f_6 kann durch einen expliziten Algorithmus berechenbar sein: Wenn beispielsweise $h(0) = 0$ gilt, dann ist $f_6(n) = 0$ für alle $n \in \mathbb{N}$. Zum Beispiel ist f_5 eine nicht berechenbare Funktion mit dieser Eigenschaft. Für die Nullfunktion gibt es offensichtlich einen expliziten Algorithmus.

   f_6 kann berechenbar sein, ohne dass wir einen expliziten Algorithmus angeben können: Wenn beispielsweise $h(0) = h(1) \in \{0, 1\}$ gilt, so dass $h(0) = 0$ genau dann, wenn es unendlich viele Primzahlzwillinge gibt, dann ist f_6 berechenbar, denn f_6 ist entweder konstant 0 oder konstant 1.

   f_6 kann nicht-berechenbar sein: Ein Beispiel für eine Funktion h, so dass f_6 nicht berechenbar ist, werden wir später konkret konstruieren können. Offenbar darf die Menge $\{h^n(0) \mid n \in \mathbb{N}\}$ dafür nicht endlich sein.

7. f_7 ist berechenbar. Entweder gilt $f_7(n) = n$ für alle $n \in \mathbb{N}$ oder $f_7(n) = 0$ für alle $n \in \mathbb{N}$. Ob sich ein Algorithmus explizit angeben lässt, hängt von der Funktion h ab. Ist beispielsweise h die Funktion f_5, dann gilt $f_5(0) \leq 0$ und damit $f_7(n) = n$ für alle $n \in \mathbb{N}$.

Aufgabe 1.2 Programmiersprache $\mathcal{P}$

Schreiben Sie ein $\mathcal{P}$-Programm zur Berechnung von $n!$ für beliebige natürliche Zahlen n. Verwenden Sie die erweiterte Syntax gemäß den Definitionen 2.2 und 2.3.

Lösungsvorschlag

```
procedure PLUS(x,y) <=
begin var n, m;
 n := x; m := y;
 while n do
  n := PRED(n);
  m := SUCC(m);
 end_while
 return(m)
end
```

```
procedure TIMES(x,y) <=
begin var n, res;
 n = x; res = 0;
 while n do
  res := PLUS(res, y);
  n := PRED(n)
 end_while
 return(res)
end
```

```
procedure FAC(y) <=
begin var n, res;
 n := y; res := 1;
 while n do
  res := TIMES(res,n)
  n := PRED(n)
 end_while
 return(res)
end
```

Aufgabe 1.3 Semantik von $\mathcal{P}$

1. Bestimmen Sie $value(M_P, \texttt{E})$ für die folgenden Ausdrücke `E` unter einer beliebigen Speicherbelegung M_P in der Form wie in Definition 2.6 angegeben. Hierfür werden Sie Definition 2.7 und die Abkürzungen in Definitionen 2.2 und 2.3 benötigen.
   a) $\texttt{E} = \neg\texttt{EXPR}$
   b) $\texttt{E} = \texttt{EXPR1} \vee \texttt{EXPR2}$
   c) $\texttt{E} = \texttt{EXPR1} \wedge \texttt{EXPR2}$
2. Bestimmen Sie $eval(M_P, \texttt{S})$ für die folgenden Anweisungen `S` unter einer beliebigen Speicherbelegung M_P in der Form wie in Definition 2.7 angegeben. Auch hier benötigen Sie Definition 2.3.
   d) $\texttt{S} = \texttt{if EXPR then STA end\_if}$
   e) $\texttt{S} = \texttt{repeat STA until EXPR}$

Lösungsvorschlag

a) Zur Demonstration des Lösungsweges hier der erste Ausdruck in kleinsten Schritten:
Sei $\texttt{P} = \texttt{x := begin var y; if EXPR then y := 0 else y := 1 end\_if; return(y) end}$

$$
\begin{aligned}
& value(M_P, \neg\texttt{EXPR}) \\
= {} & value(eval(M_P, \texttt{P}), \texttt{x}) \\
= {} & value(eval(M_P, \texttt{if EXPR then y := 0 else y := 1 end\_if; x := y}), \texttt{x}) \\
= {} & value(eval(eval(M_P, \texttt{if EXPR then y := 0 else y := 1 end\_if}), \texttt{x := y})), \texttt{x}) \\
= {} & \begin{cases} value(eval(eval(M_P, \texttt{y := 0}), \texttt{x := y}), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ value(eval(eval(M_P, \texttt{y := 1}), \texttt{x := y})), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases} \\
= {} & \begin{cases} value(eval(M_P[\texttt{y} \leftarrow value(M_P, 0)], \texttt{x := y}), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ value(eval(M_P[\texttt{y} \leftarrow value(M_P, 1)], \texttt{x := y}), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases} \\
= {} & \begin{cases} value(eval(M_P[\texttt{y} \leftarrow 0], \texttt{x := y}), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ value(eval(M_P[\texttt{y} \leftarrow 1], \texttt{x := y}), \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases} \\
= {} & \begin{cases} value(M_P[\texttt{y} \leftarrow 0][\texttt{x} \leftarrow value(M_P[\texttt{y} \leftarrow 0], \texttt{y})], \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ value(M_P[\texttt{y} \leftarrow 1][\texttt{x} \leftarrow value(M_P[\texttt{y} \leftarrow 1], \texttt{y})], \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases} \\
= {} & \begin{cases} value(M_P[\texttt{y} \leftarrow 0][\texttt{x} \leftarrow 0], \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ value(M_P[\texttt{y} \leftarrow 1][\texttt{x} \leftarrow 1], \texttt{x}) & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases} \\
= {} & \begin{cases} 0 & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ 1 & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases}
\end{aligned}
$$

Im Folgenden sind nur noch die Ergebnisse angegeben.

b)

$$value(M_P, \texttt{EXPR1} \vee \texttt{EXPR2}) = \begin{cases} 1 & \text{, falls } value(M_P, \texttt{EXPR1}) > 0 \\ value(M_P, \texttt{EXPR2}) & \text{, falls } value(M_P, \texttt{EXPR1}) = 0 \end{cases}$$

c)

$$value(M_P, \texttt{EXPR1} \wedge \texttt{EXPR2}) = \begin{cases} value(M_P, \texttt{EXPR2}) & \text{, falls } value(M_P, \texttt{EXPR1}) > 0 \\ 0 & \text{, falls } value(M_P, \texttt{EXPR1}) = 0 \end{cases}$$

d)

$$eval(M_P, \texttt{if EXPR then STA end\_if}) = \begin{cases} eval(M_P, \texttt{STA}) & \text{, falls } value(M_P, \texttt{EXPR}) > 0 \\ M_P & \text{, falls } value(M_P, \texttt{EXPR}) = 0 \end{cases}$$

e)

$$
\begin{aligned}
& eval(M_P, \texttt{repeat STA until EXPR}) \\
= {} & eval(eval(M_P, \texttt{STA}), \texttt{while } \neg\texttt{EXPR do STA end\_while})
\end{aligned}
$$