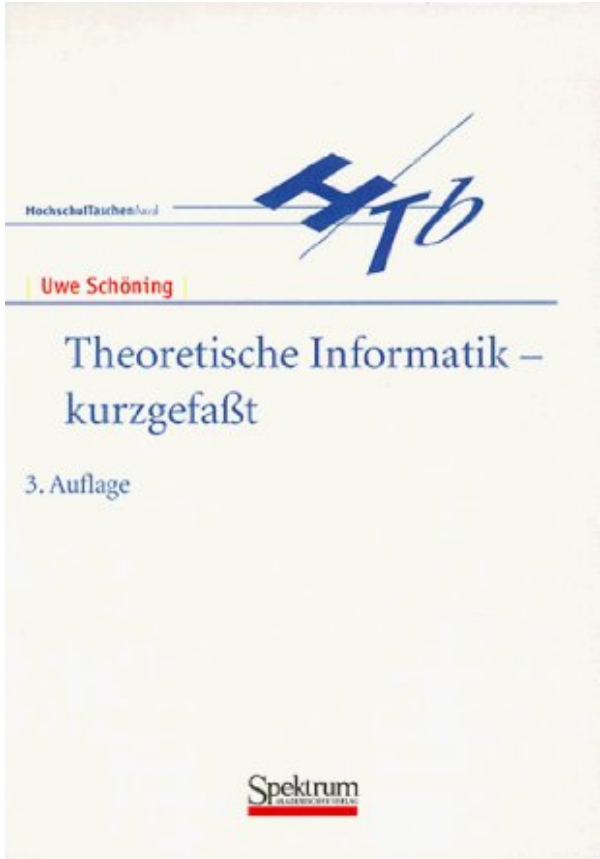
GTI

Hannes Diener

ENC B-0123,
diener@math.uni-siegen.de

18. Juni



Als Literatur zu diesem Thema empfiehlt sich das Buch „Theoretische Informatik – kurzgefasst“ von Uwe Schöning (mittlerweile in der 5. Auflage, von der 4. Auflage sind mehrere in der Unibib ausleihbar).

Kommen wir als nächstes zu unserem zweiten Ansatz zur Berechenbarkeit. Diesmal werden wir Berechenbarkeit über eine idealisierte (und extrem vereinfachte) Programmiersprache definieren.

Definieren wir zunächst die Syntax; d.h. wie unsere Programme formal aufgebaut sein müssen. Unsere Programme sind aus folgenden syntaktischen[1] Komponenten aufgebaut:

- Variablen: $x_0, x_1, \ldots$
- Konstanten: $0, 1, \ldots$
- Trennsymbole: ; $:=$
- Operationssymbole: $+, -$
- Schlüsselwörter: `LOOP`, `DO`, `END`

[1] Prinzipiell ist dies kein Alphabet, selbst wenn wir z.B. `LOOP` als ein Symbol auffassen. Das Problem ist, daß es unendlich viele Variablen und Konstanten gibt. Hier kann man sich mit dem folgendem angedeuteten Trick behelfen: Als Variablen verwenden wir x', x'', ... Auf diese Weise benötigt man nur zwei Symbole $x,'$ zur Darstellung aller Variablen.

LOOP-Programme sind jetzt induktiv definiert[2]:

- Jede Zuweisung der Form

  $$x_i := x_j + c \qquad \text{und} \qquad x_i := x_j - c$$

  ist ein LOOP-Programm.
- Sind P_1 und P_2 LOOP-Programme, so auch

  $$P_1; P_2$$

- Ist x_i eine Variable und P ein LOOP-Programm, so ist

  $$\texttt{LOOP } x_i \texttt{ DO } P \texttt{ END}$$

  ein LOOP-Programm.

[2] In einer Übung werden Sie eine alternative Definition über eine Grammatik definieren.

LOOP-Programme sind bis zu dieser Stelle vollkommen bedeutungslos.[3] D.h. zunächst nur formale Wörter ohne Bedeutung. Ihre *Semantik* müssen wir ihnen als nächstes geben.

[3] Wortspiel

Definition

Eine Speicherbelegung σ ist eine totale Funktion mit Werten in den natürlichen Zahlen definiert auf der Menge aller Variablen $\{x_0, x_1, \ldots\}$. Ist σ nur auf endlich vielen Werten $x_{i_1}, \ldots, x_{i_k}$ nicht null[4]*, so schreiben wir in Zeichen*

$$[x_{i_1} \mapsto \sigma(x_{i_1}), \ldots, x_{i_k} \mapsto \sigma(x_{i_k})] \ .$$

Also z.B.

$$[x_1 \mapsto 5, x_3 \mapsto 42, x_{42} \mapsto 3] \ .$$

Des Weiteren sei $\sigma[x_i \mapsto n]$ die Speicherbelegung, die definiert ist durch

$$\sigma[x_i \mapsto n](x_j) = \begin{cases} \sigma(x_j) & \text{falls } i \neq j \\ n & \text{sonst.} \end{cases}$$

[4] D.h. also daß z.B. $[x_1 \mapsto 3](x_5) = 0$.

Wir können nun induktiv eine Übergangsrelation definieren, welche beschreibt, welchen Effekt LOOP-Programme auf Speicherbelegungen hat. Sei hierfür LOOP die Menge aller LOOP-Programme und $\mathcal{S}$ die Menge aller Speicherbelegungen. Dann ist $\cdot \vdash \cdot \to \cdot \subseteq (\text{LOOP} \times \mathcal{S}) \times \mathcal{S}$ definiert durch

- Ist P eine Zuweisung der Form $x_i := x_j + c$, so ist

$$P \vdash \sigma \to \sigma[x_i \mapsto n] \ ,$$

  wobei $n = \sigma(x_j) + c$.
- Ist P eine Zuweisung der Form $x_i := x_j - c$, so ist

$$P \vdash \sigma \to \sigma[x_i \mapsto n] \ ,$$

  wobei $n = \max\{\sigma(x_j) - c, 0\}$.

- Gilt $P_1 \vdash \sigma \to \sigma'$ und $P_2 \vdash \sigma' \to \sigma''$, so gilt auch

  $$P_1; P_2 \vdash \sigma \to \sigma''$$

- Ist P von der Form `LOOP` x_i `DO` Q `END`, $\sigma(x_i) = n$ und

  $$\underbrace{Q; Q; \ldots Q}_{n \text{ mal}} \vdash \sigma \to \sigma'$$

  so gilt

  $$P \vdash \sigma \to \sigma' \ .$$

Man beachte, daß Q zwar die Variable x_i verändern darf, dies aber keinen Einfluss darauf hat, wie oft Q durchlaufen wird.

Satz

Sei P ein Programm und σ eine Speicherbelegung. Dann gibt es genau eine Speicherbelegung σ', so daß

$$P \vdash \sigma \to \sigma' \ .$$

Insbesondere gilt dies natürlich, wenn σ eine „Start-Speicherbelegung“ ist.

Bevor wir wieder formal einen entsprechenden Berechenbarkeitsbegriff einführen ein

Beispiel

Sei $P = \texttt{LOOP}\ x_1\ \texttt{DO}\ x_2 := x_2 + 1\ \texttt{END}; x_1 := x_2 + 0$. *Des Weiteren sei* $\sigma_1 = [x_1 \mapsto 3, x_2 \mapsto 2]$

Dann gilt, mit obiger Definition:

$$(x_2 := x_2 + 1) \vdash \sigma_1 \to \sigma_1[x_2 \mapsto 3]$$
$$(x_2 := x_2 + 1) \vdash \sigma_1[x_2 \mapsto 3] \to \sigma_1[x_2 \mapsto 4]$$

Zusammen also

$$(x_2 := x_2 + 1); (x_2 := x_2 + 1) \vdash \sigma_1 \to \sigma_1[x_2 \mapsto 4]$$

Etwas weniger ausführlich machen wir weiter:

$$(x_2 := x_2 + 1); (x_2 := x_2 + 1); (x_2 := x_2 + 1) \vdash \sigma_1 \to \sigma_1[x_2 \mapsto 5]$$

Also, da $\sigma_1(x_1) = 3$

$$\texttt{LOOP } x_1 \texttt{ DO } x_2 := x_2 + 1 \texttt{ END}; \vdash \sigma_1 \to \sigma_1[x_2 \mapsto 5]$$

In einem letzten Schritt

$$P \vdash \sigma_1 \to [x_1 \mapsto 5, x_2 \mapsto 5]$$

In der Variable x_1 steht also nach „Ausführen“ des Programms P eine 5, was genau die Summe von 2 und 3 entspricht. Allgemein kann man leicht *beweisen*, daß

$$P \vdash [x_1 \mapsto m, x_2 \mapsto n] \to [x_1 \mapsto m + n, x_2 \mapsto m + n] \ .$$

In gewisser Weise berechnet das Programm P also die Addition!

In gewisser Weise berechnet das Programm P also die Addition! In welcher Weise genau?

Definition

Die von P berechnete (k-stellige) Funktion $[\![P]\!]_k : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch

$$[\![P]\!]_k(n_1, \ldots, n_k) = \sigma'(x_1) \ ,$$

wobei σ' die nach obigen Satz die eindeutige Speicherbelegung ist, für die gilt, daß

$$P \vdash [x_1 \mapsto n_1, \ldots, x_k \mapsto n_k] \to \sigma'$$

Beobachtung

Für ein LOOP-Programm P ist $[\![P]\!]_k$ eine totale Funktion

Dies ist schon ein erster Hinweis, daß der Begriff der LOOP-Berechenbarkeit nicht unserem intuitiven Begriff der Berechenbarkeit entspricht, da wir in der Einführung ja gesagt hatten, daß es auch partielle berechenbare Funktionen gibt.

Natürlich definieren wir

Definition

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar, wenn es ein LOOP-Programm P gibt, so daß $[\![P]\!]_k = f$.

Syntaktischer Zucker:
Normalerweise enthalten imperative Programmiersprachen bedingte Anweisungen. Diese können wir mit einem Trick simulieren (d.h. wir müssen diese nicht extra zu unserer Sprache hinzufügen). Hierzu sei für ein LOOP-Programm P und eine Variable x_i

$$\texttt{IF } x_i \texttt{ THEN } P \texttt{ END}$$

die *abkürzende Schreibweise* für

$$x_j := 1;$$
$$\texttt{LOOP } x_i \texttt{ DO } x_j := 0 \texttt{ END};$$
$$\texttt{LOOP } x_j \texttt{ DO } P \texttt{ END};$$
$$x_j := 0;$$

wobei x_j eine Variable ist, die nicht in P vorkommt.

Syntaktischer Zucker:
Dies hat genau den gewünschten Effekt: Gilt $P \vdash \sigma \to \sigma'$ und ist $\sigma(x_i) = 0$, so kann man leicht zeigen, daß

$$\texttt{IF } x_i = 0 \texttt{ THEN } P \texttt{ END} \vdash \sigma \to \sigma'$$

Für $\sigma(x_i) \neq 0$ kann man analog zeigen, daß

$$\texttt{IF } x_i = 0 \texttt{ THEN } P \texttt{ END} \vdash \sigma \to \sigma$$

Mit diesen syntaktischen Kohlenhydraten gestärkt können wir jetzt leicht zeigen:

Satz

Die Addition, (abgeschnittene) Subtraktion, Differenz, Multiplikation, Division (ohne Rest) und die Mod-Funktion (der Rest einer Division) sind alle `LOOP` berechenbar.

Einschub: λ Notation.

In der Definition „Sei f eine Funktion, definiert durch $f(n) = n^2 + 1$“ ist der Name f der Funktion ja eigentlich irrelevant; insbesondere, wenn wir uns später nicht mehr darauf beziehen. Die Lösung sind anonyme Funktionen in Churchs Lambda-Notation:

$$\lambda x.t(x)$$

wobei x eine Variable und t ein Term ist, und für die Funktion steht, die jedes x auf $t(x)$ abbildet. Also in unserem Beispiel

$$\lambda n.n^2 + 1$$

Einschub: λ Notation.

In der Definition „Sei f eine Funktion, definiert durch $f(n) = n^2 + 1$“ ist der Name f der Funktion ja eigentlich irrelevant; insbesondere, wenn wir uns später nicht mehr darauf beziehen. Die Lösung sind anonyme Funktionen in Churchs Lambda-Notation:

$$\lambda x.t(x)$$

wobei x eine Variable und t ein Term ist, und für die Funktion steht, die jedes x auf $t(x)$ abbildet. Also in unserem Beispiel

$$\lambda n.n^2 + 1$$

Diese Notation macht auch den Unterschied zwischen Termen und Funktionen klarer.

Mehr Syntaktischer Zucker:
Ist P ein Programm, so steht der Ausdruck

$$x_j := P(x_{i_1}, \ldots, x_{i_k})$$

abkürzend für das Programm

$$x_{j_1} := x_1; x_1 := x_{i_1}; \ldots; x_{j_k} := x_k; P; x_j := x_1; x_1 := x_{j_1}; \ldots x_k := x_{j_k}$$

wobei die Variablen $x_{j_1}, \ldots, x_{j_k}$ „frische“ Variablen sind, d.h. nirgendwo sonst verwendet werden.
Sei beispielsweise ADD das oben behandelte erste Beispiel eines LOOP-Programms. So können wir nun auch, mit etwas Streuen von syntaktischen Zucker, auch Befehle wie

$$x_4 := ADD(x_1, x_2)$$

verwenden.

LOOP Funktionen sind ausserdem unter einer Vielzahl von Operationen abgeschlossen. Z.B. Fallunterscheidung:

Lemma

Sind $f, g, h : \mathbb{N} \to \mathbb{N}$ LOOP-berechenbar, so auch die Funktion $F : \mathbb{N} \to \mathbb{N}$ definiert durch

$$F(n) = \begin{cases} g(n) & \text{falls } h(x) = 0 \\ f(n) & \text{falls } h(x) \neq 0 \end{cases}$$

LOOP berechenbar.

Beispiel: Die $3n + 1$ Funktion.

Noch ein weiteres Beispiel:

Beispiel

Die Fakultätsfunktion $\lambda n.n!$ ist LOOP-berechenbar.

Dies ist ein Spezialfall der folgenden *primitiven Rekursion* Ist $h : \mathbb{N}^2 \to \mathbb{N}$ eine Funktion und $g \in \mathbb{N}$ ein Wert, und sei $f : \mathbb{N} \to \mathbb{N}$ definiert durch $f(0) = g$ und $f(n+1) = h(n, f(n))$;

Satz

Ist h `LOOP` berechenbar, so auch die Funktion f.

Mehr noch: wir können alles natürlich noch von k Parametern abhängen lassen. Also

Definition

Sind $h : \mathbb{N}^{k+2}$, $g : \mathbb{N}^k \to \mathbb{N}$ *und* $f : \mathbb{N}^{k+1} \to \mathbb{N}$ *Funktionen, so daß*

$$f(0, y_1, \ldots, y_k) = g(y_1, \ldots, y_k)$$
$$f(n+1, y_1, \ldots, y_k) = h(n, f(n, y_1, \ldots, y_k), y_1, \ldots, y_k) \ .$$

So sagen wir, daß f *durch primitive Rekursion aus* g *und* h *hervorgeht.*

Satz

LOOP-Programme sind unter primitiver Rekursion abgeschlossen; d.h. sind f, g, h *wie oben und* g, h *LOOP-berechenbar, so ist auch* f *LOOP-berechenbar.*

WHILE-Programme

Wie schon angedeutet umfasst der Begriff der LOOP-Berechenbarkeit nicht alle intuitiv berechenbaren Funktionen (konkretes Beispiel später in Form der Ackermann-Funktion). Wir erweitern deshalb unsere Programmiersprache um ein weiteres Konstrukt.

Definition

Die Menge der `WHILE`*-Programme ist genauso definiert wie die* `LOOP`*-Programme, aber zusätzlich ist auch, wenn* P *ein* `WHILE`*-Programm ist und* x_i *eine Variable, so ist auch*

$$\texttt{WHILE } x_i \neq 0 \texttt{ DO } P \texttt{ END}$$

ein `WHILE`*-Programm.*

Die Semantik der WHILE-Programme ist auch definiert wie die der LOOP-Programme, allerdings gilt zusätzlich:

- Sind $\sigma_1, \sigma_2, \ldots, \sigma_n$ Speicherbelegungen so daß $\sigma_j(x_i) \neq 0$ für $j < n$ und $\sigma_n(x_i) = 0$ und $P \vdash \sigma_j \to \sigma_{j+1}$ für $j < n$, so gilt

  $$\texttt{WHILE } x_i \neq 0 \texttt{ DO } P \texttt{ END} \vdash \sigma_1 \to \sigma_n \ .$$

- Ist σ eine Speicherbelegung, so daß $\sigma(x_i) = 0$, so gilt

  $$\texttt{WHILE } x_i \neq 0 \texttt{ DO } P \texttt{ END} \vdash \sigma \to \sigma \ .$$

Beobachtung

`LOOP`-Schleifen können durch `WHILE`-Schleifen ersetzt werden.

Alle Beobachtungen über LOOP-Programme übertragen sich analog auch auf WHILE-Programme. Mit einer Ausnahme:

Beispiel

Es gibt WHILE-Programme P und Speicherbelegungen σ, so daß es keine Speicherbelegung σ' gibt, mit $P \vdash \sigma \to \sigma'$.

Trotzdem gilt noch die Eindeutigkeit.

Satz

Ist P ein WHILE-Programm σ, σ' Speicherbelegungen mit $P \vdash \sigma \to \sigma'$, so ist σ' eindeutig.

Definition

Die von einem WHILE-Programm P berechnete (k-stellige) partielle Funktion $[\![P]\!]_k : \mathbb{N}^k \rightharpoonup \mathbb{N}$ ist definiert durch

$$[\![P]\!]_k(n_1, \ldots, n_k) = \begin{cases} \sigma'(x_1) & \text{falls } P \vdash [x_1 \mapsto n_1, \ldots, x_k \mapsto n_k] \to \sigma' \\ \bot & \text{falls es kein solches } \sigma' \text{ gibt.} \end{cases},$$

Natürlich definieren wir auch hier

Definition

Eine partielle Funktion $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ ist WHILE-berechenbar, wenn es ein WHILE-Programm P gibt, so daß $[\![P]\!]_k = f$.

Beispiel

Die Funktion $\mathrm{sqrt} : \mathbb{N} \rightharpoonup \mathbb{N}$ *definiert durch*

$$\mathrm{sqrt}(n) = \begin{cases} \sqrt{n} & \textit{falls } n \textit{ Qudratzahl ist} \\ \bot & \textit{sonst} \end{cases}$$

ist `WHILE`*-berechenbar.*