Theorie der Informatik

Malte Helmert und Gabriele Röger

FS 2014

14 Primitive Rekursion und μ-Rekursion

Vorbemerkung: Die Kodierungs- und Dekodierungsfunktionen *encode*, decode_1 und decode_2 von den Vorlesungsfolien für Paare von natürlichen Zahlen lassen sich leicht auf beliebige r-Tupel (für alle $r \geq 2$) erweitern. Hierbei soll $\mathit{encode}^r : \mathbb{N}_0^r \to \mathbb{N}_0$ ein r-Tupel bijektiv in eine einzelne Zahl kodieren und $\mathit{decode}_i^r : \mathbb{N}_0 \to \mathbb{N}_0$ für $1 \leq i \leq r$ dann die i-te Komponente des kodierten k-Tupels extrahieren, also

$$\mathit{decode}_i^k(\mathit{encode}^k(x_1, \ldots, x_k)) = x_i$$

für alle $k \geq 2$, $1 \leq i \leq k$, $x_1, \ldots, x_k \in \mathbb{N}_0$.

Für den Fall $r = 2$ verwenden wir die bereits bekannten Funktionen aus der Vorlesung:

$$\begin{aligned}\mathit{encode}^2(x_1, x_2) &:= \mathit{encode}(x_1, x_2)\\ \mathit{decode}_1^2(z) &:= \mathit{decode}_1(z)\\ \mathit{decode}_2^2(z) &:= \mathit{decode}_2(z)\end{aligned}$$

und setzen dann für alle $r \geq 2$:

$$\begin{aligned}\mathit{encode}^{r+1}(x_1, \ldots, x_{r+1}) &:= \mathit{encode}(\mathit{encode}^r(x_1, \ldots, x_r), x_{r+1})\\ \mathit{decode}_j^{r+1}(z) &:= \mathit{decode}_j^r(\mathit{decode}_1(z)) \qquad \text{für alle } 1 \leq j \leq r\\ \mathit{decode}_{r+1}^{r+1}(z) &:= \mathit{decode}_2(z)\end{aligned}$$

In Worten: um ein $(r+1)$-Tupel mit encode^{r+1} zu kodieren, kodieren wir zunächst die ersten r Komponenten mit encode^r als einzelne Zahl. Anschliessend kodieren wir dann diese Zahl und die $(r+1)$-te Komponente mit *encode* in eine einzelne Zahl.

Die Dekodierungsfunktionen machen diese Operation rückgängig: decode_i^{r+1} extrahiert so die i-te Komponente des kodierten Tupels.

Satz 14.1 (Zusammenhang primitive/μ-Rekursion vs. LOOP-/WHILE-Programme)

1. *Jede primitiv rekursive Funktion ist LOOP-berechenbar.*
2. *Jede LOOP-berechenbare Funktion ist primitiv rekursiv.*
3. *Jede μ-rekursive Funktion ist WHILE-berechenbar.*
4. *Jede WHILE-berechenbare Funktion ist μ-rekursiv.*

Beweis. zu 1.: Um den folgenden Beweis zu erleichtern, überlegen wir uns zunächst, dass wir zu jedem LOOP-Programm P ein „sauberes“ LOOP-Programm $\tilde{P}$ konstruieren können, das dieselbe Funktion berechnet wie P (d.h. die Variable x_0 mit demselben Wert hinterlässt wie P) und alle anderen Programmvariablen mit demselben Wert hinterlässt wie vor Ausführung von $\tilde{P}$. Saubere Programme haben also keine unerwünschten Seiteneffekte.

Konkret können wir aus P das saubere Programm $\tilde{P}$ wie folgt erzeugen. Seien $x_1, \ldots, x_m$ alle Variablen ausser x_0, die P verwendet. Seien $z_1, \ldots, z_m$ frische Variablen. Dann ist $\tilde{P}$ das folgende Programm:

$z_1 := x_1; \quad \ldots \quad z_m := x_m;$
$P;$
$x_1 := z_1; \quad \ldots \quad x_m := z_m;$
$z_1 := 0; \quad \ldots \quad z_m := 0$

Wir zeigen nun zunächst, dass die Basisfunktionen LOOP-berechenbar sind. Anschliessend zeigen wir, dass Funktionen, die mit dem Einsetzungsschema und mit dem primitiven Rekursionsschema konstruiert werden können, LOOP-berechenbar sind, wenn die Funktionen, aus denen sie konstruiert werden, LOOP-berechenbar sind. Dies zeigt, dass alle PRFs LOOP-berechenbar sind.

Basisfunktionen:

- *Nullfunktion: null* : $\mathbb{N}_0 \to \mathbb{N}_0$ mit *null*$(x) = 0$ für alle $x \in \mathbb{N}_0$

  Die Nullfunktion wird berechnet durch das LOOP-Programm:

  $x_0 := 0$

- *Nachfolgerfunktion: succ* : $\mathbb{N}_0 \to \mathbb{N}_0$ mit *succ*$(x) = x + 1$ für alle $x \in \mathbb{N}_0$

  Die Nachfolgerfunktion wird berechnet durch das LOOP-Programm:

  $x_0 := x_1 + 1$

- *Projektionsfunktionen:* $\pi^i_j : \mathbb{N}^i_0 \to \mathbb{N}_0$ mit $\pi^i_j(x_1, \ldots, x_i) = x_j$ für alle $x_1, \ldots, x_i \in \mathbb{N}_0$

  Die Projektionsfunktion π^i_j wird berechnet durch das LOOP-Programm:

  $x_0 := x_j$

Einsetzungsschema:
Sei $f : \mathbb{N}^k_0 \to \mathbb{N}_0$ eine PRF, die durch das Einsetzungsschema entsteht, d.h. es gibt Funktionen $h : \mathbb{N}^i_0 \to \mathbb{N}_0$ und $g_1, \ldots, g_i : \mathbb{N}^k_0 \to \mathbb{N}_0$ mit $f(x_1, \ldots, x_k) = h(g_1(x_1, \ldots, x_k), \ldots, g_i(x_1, \ldots, x_k))$ für alle $x_1, \ldots, x_k \in \mathbb{N}_0$. Wir müssen zeigen: wenn h, g_1, …, g_i alle LOOP-berechenbar sind, dann ist f LOOP-berechenbar.
Zu $u \in \{h, g_1, \ldots, g_i\}$ sei P_u ein *sauberes* LOOP-Programm, das u berechnet. Seien $y_1, \ldots, y_i$ frische Variablen. Dann berechnet das folgende LOOP-Programm die Funktion f:

$P_{g_1}; \quad y_1 := x_0; \quad x_0 := 0;$
$P_{g_2}; \quad y_2 := x_0; \quad x_0 := 0;$
$\ldots$
$P_{g_i}; \quad y_i := x_0; \quad x_0 := 0;$
$x_1 := 0; \quad x_2 := 0; \quad \ldots \quad x_k := 0;$
$x_1 := y_1; \quad x_2 := y_2; \quad \ldots \quad x_i := y_i;$
P_h

Idee des Programms: Berechne zuerst alle Werte $g_j(x_1, \ldots, x_k)$ für $1 \leq j \leq i$ und speichere sie in den Variablen y_j. Berechne dann $h(y_1, \ldots, y_i)$.

primitives Rekursionsschema:
Sei $f : \mathbb{N}^{k+1}_0 \to \mathbb{N}_0$ eine PRF, die durch das primitive Rekursionsschema entsteht, d.h. es gibt Funktionen $g : \mathbb{N}^k_0 \to \mathbb{N}_0$ und $h : \mathbb{N}^{k+2}_0 \to \mathbb{N}_0$, so dass

$$f(0, x_1, \ldots, x_k) = g(x_1, \ldots, x_k)$$
$$f(n+1, x_1, \ldots, x_k) = h(f(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k)$$

für alle $n, x_1, \ldots, x_k \in \mathbb{N}_0$. Wir müssen zeigen: wenn g und h LOOP-berechenbar sind, dann ist f LOOP-berechenbar.

Seien hierzu P_g und P_h *saubere* LOOP-Programme, die g bzw. h berechnen. Seien ferner $x'_1, \ldots, x'_{k+1}$, *result* und *counter* frische Variablen.

Dann berechnet das folgende LOOP-Programm f:

$x'_1 := x_1; \quad x'_2 := x_2; \quad \ldots \quad x'_{k+1} := x_{k+1};$
$x_1 := x'_2; \quad x_2 := x'_3; \quad \ldots \quad x_k := x'_{k+1}; \quad x_{k+1} := 0;$
$P_g;$
result $:= x_0;$
counter $:= 0;$
LOOP x'_1 **DO**
    $x_0 := 0; \quad x_1 :=$ *result*; $\quad x_2 :=$ *counter*;
    $x_3 := x'_2; \quad x_4 := x'_3; \quad \ldots \quad x_{k+2} := x'_{k+1};$
    $P_h;$
    result $:= x_0;$
    counter $:=$ *counter* $+ 1$
END;
$x_0 :=$ *result*

Idee des Programms: Bei Eingabe $\langle n, a_1, \ldots, a_k \rangle$, setze zuerst *result* $:= g(a_1, \ldots, a_k)$. Wiederhole dann n-mal die Berechnung *result* $:= h($*result*, *counter*$, a_1, \ldots, a_k)$, wobei *counter* die Werte $0, \ldots, n-1$ durchläuft. Man kann leicht prüfen, dass diese iterative Berechnung denselben Wert errechnet wie die rekursive Definition über das primitive Rekursionsschema.

zu 2.: Sei P ein beliebiges LOOP-Programm, das die Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ berechnet. Wir müssen zeigen, dass f eine PRF ist. Seien hierzu $x_0, \ldots, x_m$ die von P verwendeten Variablen. Ohne Einschränkung fordern wir $m \geq k$. (Es ist für den Beweis kein Problem, so zu tun, als würde das LOOP-Programm eine Variable verwenden, selbst wenn es diese nicht verwendet.)

Wir konstruieren für jedes Teilprogramm Q von P (inklusive P selbst) eine PRF f_Q, die kodiert, welchen Effekt das Programm Q auf den Variablen $x_0, \ldots, x_m$ hat. Die Grundidee hierbei ist, dass f_Q die Werte der Programmvariablen vor Ausführung von Q als Eingabe hat und die Werte der Programmvariablen nach Ausführung zurückliefert. Hierbei ergibt sich die Schwierigkeit, dass PRFs nur *einen* Wert berechnen können, wir aber die neuen Wert von $m+1$ Variablen zurückliefern müssen. Wir lösen diese Schwierigkeit, indem wir $(m+1)$-Tupel von Zahlen in einer einzelnen Zahl kodieren.

Formaler: wir definieren zu jedem Teilprogramm Q von P eine Funktion $f_Q : \mathbb{N}_0 \to \mathbb{N}_0$, so dass gilt:

$$f_Q(\mathit{encode}^{m+1}(a_0, \ldots, a_m)) = \mathit{encode}^{m+1}(b_0, \ldots, b_m)$$

genau dann, wenn Q ausgeführt in einer Situation mit Variableninhalten $x_0 = a_0, \ldots, x_m = a_m$ in einer Situation mit Variableninhalten $x_0 = b_0, \ldots, x_m = b_m$ endet.

Nach der Semantik von LOOP-Programmen berechnet P den Funktionswert, der nach Ausführung von P in Variable x_0 steht, wobei anfänglich die Variablen $x_1, \ldots, x_k$ die Eingabewerte und alle andere Variablen den Wert 0 beinhalten. Somit gilt:

$$f(a_1, \ldots, a_k) = \mathit{decode}_1^{m+1}(f_P(\mathit{encode}^{m+1}(0, a_1, \ldots, a_k, \underbrace{0, \ldots, 0}_{(m-k) \text{ mal}}))).$$

Diese Funktion ist primitiv rekursiv, wenn wir zeigen können, dass die Funktion f_P primitiv rekursiv ist. Dies zeigen wir, indem wir zeigen, dass f_Q für *alle* Teilprogramme Q von P (also auch P selbst) primitiv

rekursiv ist. Der Beweis erfolgt per Induktion über den Aufbau des LOOP-Programms P. Hierbei verwenden wir, dass es bei LOOP-Programmen ausreicht, die Operationen „$x_i := x_i + 1$“, „$x_i := x_i - 1$“, *Komposition* und *LOOP-Schleife* zuzulassen.

- Q ist „$x_i := x_i + 1$“:

  Definiere $f_Q : \mathbb{N}_0 \to \mathbb{N}_0$ mit

  $$f_Q(z) := \mathit{encode}^{m+1}(\mathit{decode}_1^{m+1}(z) + c_0, \ldots, \mathit{decode}_{m+1}^{m+1}(z) + c_m),$$

  wobei die Konstanten c_j definiert sind als $c_i = 1$ und $c_j = 0$ für alle $j \neq i$.

  Dies ist eine PRF, da alle Bestandteile PRFs sind.

- Q ist „$x_i := x_i - 1$“:

  Analog mit $\cdots \ominus c_j$ anstelle von $\cdots + c_j$.

- Q ist „$Q_1; Q_2$“:

  Induktiv gibt es PRFs f_{Q_1} und f_{Q_2}, die die Effekte der Teilprogramme auf den Variableninhalten kodieren.

  Definiere $f_Q : \mathbb{N}_0 \to \mathbb{N}_0$ mit $f_Q(z) := f_{Q_2}(f_{Q_1}(z))$.

- Q ist „**LOOP** x_i **DO** R **END**“:

  Induktiv gibt es eine PRF f_R, die kodiert, wie der Schleifenrumpf R (bei einmaliger Ausführung) die Variableninhalte verändert. Die Funktion f_Q muss kodieren, wie sich die Variableninhalte verändern, wenn man n-mal hintereinander R ausführt, wobei n der Wert der Variablen x_i zu Beginn der Ausführung von Q ist. Hierfür definieren wir zunächst eine Hilfsfunktion $\tilde{f}_Q : \mathbb{N}_0^2 \to \mathbb{N}_0$ mit folgender Semantik: $\tilde{f}_Q(n, z)$ kodiert die Variableninhalte nach n-maliger Ausführung von Q auf den Anfangswerten, die durch z kodiert sind. Dann gilt:

  $$\tilde{f}_Q(0, z) = z$$
  $$\tilde{f}_Q(n + 1, z) = f_R(\tilde{f}_Q(n, z))$$

  Dies ist eine Anwendung des primitiven Rekursionsschemas (mit $g(a) := a$ und $h(a, b, c) := f_R(a)$). Damit ist $\tilde{f}_Q$ eine PRF. Die gesuchte Funktion f_Q ist dann definiert durch

  $$f_Q(z) = \tilde{f}_Q(\mathit{decode}_{i+1}^{n+1}(z), z).$$

  Da die Funktion f_Q durch Einsetzung aus PRFs entsteht, ist sie ebenfalls eine PRF.

zu 3. und 4.: ohne Beweis ∎