Grundzüge der Theoretischen Informatik

17. November 2021

Markus Bläser

Universität des Saarlandes

Kapitel 7: WHILE- und FOR-Programme

Die WHILE-Progammiersprache

Variablen: $x_0, x_1, x_2, \ldots$

Konstanten: $0, 1, 2, \ldots$

Schlüsselwörter: **while**, **do**, **od**

Sonstige Zeichen: $:=, \neq, \; ; \,, +, -, [,]$

Definition (7.1)

WHILE-Programme sind induktiv definiert wie folgt:

1. Einfache Anweisungen haben die Form

   $$x_i := x_j + x_k \quad \textit{oder} \quad x_i := x_j - x_k \quad \textit{oder} \quad x_i := c,$$

   wobei $i, j, k \in \mathbb{N}$ und $c \in \mathbb{N}$.
2. Ein WHILE-Program P ist entweder eine einfache Anweisung oder lässt sich schreiben als:

   2.1 **while** $x_i \neq 0$ **do** P_1 **od** oder

   2.2 $[P_1;\, P_2]$

   für ein $i \in \mathbb{N}$ und WHILE-Programme P_1 und P_2.

Die WHILE-Progammiersprache (2)

$$
\begin{aligned}
\mathcal{W}_0 &= \text{Menge aller einfachen Anweisungen} \\
\mathcal{W}_n &= \mathcal{W}_{n-1} \cup \{P \mid \exists P_1 \in \mathcal{W}_{n-1},\ x_i \in X, \\
&\qquad\qquad\qquad \text{so dass } P = \textbf{while } x_i \neq 0 \textbf{ do } P_1 \textbf{ od} \text{ oder} \\
&\qquad\qquad\qquad \exists P_1 \in W_j,\ P_2 \in W_k \text{ mi } j + k \leq n - 1, \\
&\qquad\qquad\qquad \text{so dass } P = [P_1; P_2]\}
\end{aligned}
$$

Bemerkung

Ein WHILE-Programm P ist in $\mathcal{W}_n$, genau dann wenn es aus einfachen Anwendungen durch höchstes n Anwendungen einer Regel aus 2. in Definition 7.1 gebaut werden kann.

Die FOR-Programmiersprache

Definition (7.1)

FOR-Programme sind induktiv definiert wie folgt:

1. Einfache Anweisungen haben die Form

   $$x_i := x_j + x_k \quad \textit{oder} \quad x_i := x_j - x_k \quad \textit{oder} \quad x_i := c,$$

   wobei $i, j, k \in \mathbb{N}$ und $c \in \mathbb{N}$.
2. Ein FOR-Program P ist entweder eine einfache Anweisung oder lässt sich schreiben als:
   2.1 **for** x_i **do** P_1 **od** oder $\hookrightarrow$ while $x_i \neq 0$ do P_1 od
   2.2 $[P_1;\ P_2]$

   für ein $i \in \mathbb{N}$ und FOR-Programme P_1 und P_2.

Semantik

- Eingabe eines Programms P: $\alpha_0, \ldots, \alpha_{s-1} \in \mathbb{N}$.
- Gespeichert in $x_0, \ldots, x_{s-1}$.
- Ausgabe: Inhalt von x_0 nach Ausführung von P.

- Die Menge $X = \{x_0, x_1, x_2, \ldots\}$ aller Variablen ist unendlich.
- Aber jedes Programm P kann nur endlich viele Variablen benutzen.
- $\ell = \ell(P)$ größte Index einer Variablen in P.
- Annahme: $\ell \geq s - 1$.

$\ell(x_i := x_j + x_k)$
$= \max\{i, j, k\}$

Zustand eines Programms P: Element aus $\mathbb{N}^{\underline{\ell(P)+1}}$.
Besser: Unendliche Folge $\mathbb{N} \to \mathbb{N}$ mit endlichem Träger $\leftrightarrow \exists n_0 \in \mathbb{N}\ \forall n \geq n_0\ s(n) = 0$

$s(0) = x_0$ $\qquad$ $s: (x_0, x_1, \ldots, x_{\ell(P)}, 0, 0, 0, \ldots)$

Semantik (2)

Ein Programm bildet Zustände auf Zustände ab
$\longrightarrow$ Partielle Funktion Φ_P

P muss nicht terminieren!

1. Falls P eine einfache Anweisung ist, dann ist

i-te Stelle

$(\sigma_0, \sigma_1, \ldots)$

$\shortparallel$

$$\Phi_P(S) = \begin{cases} (\sigma_0, \ldots, \sigma_{i-1}, \sigma_j + \sigma_k, \sigma_{i+1}, \ldots) \\ \qquad \text{falls } P \text{ gleich } x_i := x_j + x_k \text{ ist} \\ (\sigma_0, \ldots, \sigma_{i-1}, \max\{\sigma_j - \sigma_k, 0\}, \sigma_{i+1}, \ldots) \\ \qquad \text{falls } P \text{ gleich } x_i := x_j - x_k \text{ ist} \\ (\sigma_0, \ldots, \sigma_{i-1}, c, \sigma_{i+1}, \ldots) \\ \qquad \text{falls } P \text{ gleich } x_i := c \text{ ist} \end{cases}$$

Semantik (3)

$\Phi_{P_1}^{(0)}(S) = S$

$\Phi_{P_1}^{(r+1)}(S) = \Phi_{P_1}(\Phi_{P_1}^{(r)}(S))$

2. 1 Sei P gleich **while** $x_i \neq 0$ **do** P_1 **od**.
Sei r das kleinste $r \in \mathbb{N}$, so dass $\Phi_{P_1}^{(r)}(S)$ entweder undefiniert oder die i-te Komponente in $\Phi_{P_1}^{(r)}(S)$ gleich 0 ist.
Dann ist

$$\Phi_P(S) = \begin{cases} \Phi_{P_1}^{(r)}(S) & \text{falls } r \text{ existiert und } \Phi_{P_1}^{(r)}(S) \text{ definiert ist} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

2 Falls P gleich $[P_1; P_2]$ für WHILE-Programme P_1 und P_2 ist, dann ist

Var. nach P_1

$$\Phi_P(S) = \begin{cases} \Phi_{P_2}(\Phi_{P_1}(S)) & \text{falls } \Phi_{P_1}(S) \text{ und} \\ & \Phi_{P_2}(\Phi_{P_1}(S)) \text{ definiert sind} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Semantik (4)

Lemma (7.3)

Für jedes WHILE-Programm ist Φ_P *wohldefiniert.*

explizit Wert zugewiesen

eindeutig zugewiesen

Lemma (7.4)

Für WHILE-Programme P_1, P_2 *und* P_3 *gilt*

$$\Phi_{[P_1;[P_2;P_3]]} = \Phi_{[[P_1;P_2];P_3]}.$$

Lemma 7.3

IA. $P \in W_0$ $\Phi_P(S)$ ist für jedes S genau einmal def.

$\Rightarrow \Phi_P$ ist wohldef.

IV. Für $P \in W_n$ ist Φ_P wohldef. für ein $n \geq 0$

IS. $P \in W_{n+1} \setminus W_n$

$P =$ while $x_i \neq 0$ do P_1 od

$P_1 \in W_n$

$\Rightarrow \Phi_{P_1}$ ist wohldef.

$\Rightarrow \Phi_P$ wohldef. nach Regel 2a

$P = [P_1; P_2]$

$P_1, P_2 \in W_n$

$\Rightarrow \Phi_{P_1}, \Phi_{P_2}$ ist wohldef.

$\Rightarrow \Phi_P$ wohldef. nach Regel 2b

Lemma 7.4

$$\bar{\Phi}_{[P_1;[P_2;P_3]]}(S) = \begin{cases} \bar{\Phi}_{[P_2;P_3]}(\bar{\Phi}_{P_1}(S)) & \text{falls } \bar{\Phi}_{P_1}(S) \text{ und } \bar{\Phi}_{[P_2;P_3]}(\bar{\Phi}_{P_1}(S)) \text{ def. sind} \\ \text{undef.} & \text{sonst} \end{cases}$$

$$\bar{\Phi}_{[P_2;P_3]}(T) = \begin{cases} \bar{\Phi}_{P_3}(\bar{\Phi}_{P_2}(T)) & \text{falls } \_ \text{ und } \_ \text{ def. sind} \\ \text{undef.} & \text{sonst} \end{cases}$$

Mit $T = \bar{\Phi}_{P_1}(S)$

$$\bar{\Phi}_{[P_1;[P_2;P_3]]}(S) = \begin{cases} \bar{\Phi}_{P_3}(\bar{\Phi}_{P_2}(\bar{\Phi}_{P_1}(S))) & \text{falls } \_, \_ \text{ und } \_ \text{ def. sind} \\ \text{undef} & \text{sonst} \end{cases}$$

$$= \bar{\Phi}_{[[P_1;P_2];P_3]}(S)$$

selbst nachrechnen

$\square$

Semantik von FOR-Programmen

einfache Statements: identisch zu WHILE

2. 1 Sei P gleich **for** x_i **do** P_1 **od** für ein FOR-Programm P_1. Dann ist

$$\Phi_P(S) = \Phi_{P_1}^{(\sigma_i)}(S).$$

$(\sigma_0, \sigma_1, \ldots)$

2 Sei P gleich $[P_1; P_2]$ für FOR-Programme P_1 and P_2. Dann ist

$$\Phi_P(S) = \Phi_{P_2}(\Phi_{P_1}(S)).$$

$x_0 := 1;$
for x_0 do
$x_0 := x_0 + x_0$
od

terminiert nach 1 Schleifen durchläufen

WHILE-berechenbare Funktionen

Definition (7.5)

Sei P ein WHILE- oder FOR-Programm mit s Eingaben. Die Funktion $\varphi_P^s : \mathbb{N}^s \to \mathbb{N}$ ist definiert durch: $x_0, \dots x_{s-1}$

kann weggelassen werden, wenn klar aus Kontext

$$\varphi_P^s(\alpha_1, \dots, \alpha_s) = \begin{cases} \text{erster Eintrag von } \Phi_P((\alpha_1, \dots, \alpha_s, 0, \dots)) & \\ \qquad \text{falls } \Phi_P((\alpha_1, \dots, \alpha_s, 0, \dots)) \text{ definiert} & \\ \text{undefiniert} \qquad \text{sonst} & \end{cases}$$

x_0 $x_0 \dots x_{s-1}$

R $\hat{=}$ rekursiv

PR $\hat{=}$ primitiv rekursiv

PR $\subsetneq$ R

Definition (7.6)

1. Eine partielle Funktion $f : \mathbb{N}^s \to \mathbb{N}$ ist *WHILE-berechenbar*, falls es ein WHILE-Programm P gibt, so dass $f = \varphi_P$.
2. f ist *FOR-berechenbar* falls $f = \varphi_P$ für ein FOR-Programm P.
3. Die Menge aller WHILE-berechenbaren Funktionen heißt R.
4. Die Menge aller FOR-berechenbaren Funktionen heißt PR.

Entscheidbare Sprachen

WHILE $\downarrow$ DEAs $\downarrow$

$N \hookrightarrow \Sigma^*$

- $L \subseteq \mathbb{N}$ nennen wir auch Sprache.
- *Characteristische Funktion* von L

$L = \{2n \mid n \in N\}$

$\chi_L(x) = \begin{cases} 1 & \text{wenn } x \text{ gerade} \\ 0 & \text{sonst} \end{cases}$

$$\chi_L : \mathbb{N} \to \{0,1\} \subseteq N$$
$$x \mapsto \begin{cases} 1 & \text{falls } x \in L, \\ 0 & \text{sonst.} \end{cases}$$

$L = \{x \in N \mid \chi_L(x) = 1\}$

- χ_L kann als Funktion $\mathbb{N} \to \mathbb{N}$ aufgefaßt werden.

Definition (7.7)

1. $L \subseteq \mathbb{N}$ heißt *rekursiv* oder *entscheidbar* falls $\chi_L \in R$.
2. Die Menge aller entscheidbaren Sprachen heißt REC.

$\neq$ REG

Kapitel 8: Syntaktischer Zucker

Syntaktischer Zucker

Variablennamen

- beliebige Namen
- werden wieder ersetzt durch x_i

Zuweisungen $x_i := x_j$
wird simuliert durch:

1: $x_k := 0$;
2: $x_i := x_j + x_k$ $=0$

x_k unbenutzte Variable

Funktionsaufrufe (nicht rekursiv)

- Sei h WHILE- oder FOR-berechenbare Funktion $\mathbb{N}^t \to \mathbb{N}$.
- Sei P ein WHILE- oder FOR-Programm for h.

Neues Statement: $x_i := h(x_{j_1}, \ldots, x_{j_t})$

$P \quad \hat{P}$

$x_0 \leftrightarrow x_{\ell+1}$

$x_1 \leftrightarrow x_{\ell+2}$

$\vdots$

$$\left.\begin{array}{ll} 1: & x_{\ell+1} := x_{j_1}; \\ 2: & \vdots \\ 3: & x_{\ell+t} := x_{j_t}; \end{array}\right\} \text{Eingaben für } h$$

$$\left.\begin{array}{ll} 4: & x_{\ell+t+1} := 0; \\ 5: & \vdots \\ 6: & x_{\ell+m+1} := 0; \end{array}\right\} \text{restliche verwendete Variablen}$$

7: $\hat{P}$;

8: $x_i := x_{\ell+1}$ ← Ergebnis

- ℓ die der größte Index einer Variablen im momentanen Programm.
- m die der größte Index einer Variablen in P
- $\hat{P}$ ergibt sich aus P durch Ersetzen von jedem x_i durch $x_{i+\ell+1}$

Arithmetische Operationen

Z.B. Multiplikation: $x_i := x_j \cdot x_k$:

$x_i := 0$;
for x_j **do**
    $x_i := x_i + x_k$
od

Wir nehmen an, dass $i \neq j, k$. $\rightarrow$ $x_{l+1} := x_j \cdot x_k$;
$x_i := x_{l+1}$

Bemerkung

FOR-Schleifen können durch WHILE-Schleifen simuliert werden.

$\Rightarrow PR \leq R$

Paarfunktionen → Arrays

$\mathbb{N}\times\mathbb{N} \to \mathbb{N}$

Lemma (8.3)

$(a,b) \mapsto \langle a,b \rangle$

Es gibt FOR-berechenbare Funktionen $\langle .,. \rangle : \mathbb{N}^2 \to \mathbb{N}$ *und* $\pi_i : \mathbb{N} \to \mathbb{N}$, $i = 1, 2$, *so dass*

$$\pi_i(\langle x_1, x_2 \rangle) = x_i$$

→ ÜB

für alle $x_1, x_2 \in \mathbb{N}$ *und* $i = 1, 2$.

$\Rightarrow \langle .,. \rangle$ ist injektiv

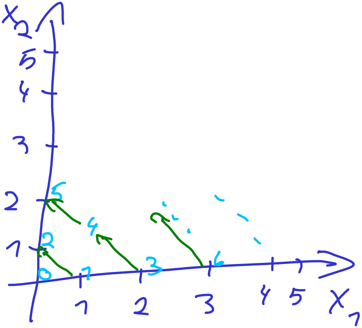
Bemerkung (8.4)

$\langle .,. \rangle$ ist auch surjektiv. → $\langle .,. \rangle$ ist bijektiv

Paarfunktionen (2)

$$\langle x_1, x_2\rangle = \frac{1}{2}\overset{\text{ungerade}}{\underset{\text{gerade}}{(x_1+x_2)}}\overset{\text{gerade}}{\underset{\text{ungerade}}{(x_1+x_2+1)}} + x_2 \overset{?}{\in} \mathbb{N}$$

$$= \sum_{i=0}^{x_1+x_2} i + x_2$$



$\langle 0,0\rangle = 0$

$\langle 1,0\rangle = 1$

$\langle 0,1\rangle = 2$

$\langle 2,0\rangle = 3$

$\langle 1,1\rangle = 4$

$\langle 0,2\rangle = 5$

Stacks

Datenstruktur Stack

push(S, x)	legt x auf den Stack S
pop(S)	entfernt das oberste Element von S keine Auswirkung, falls S leer ist.
top(S)	gibt das oberste Element von S zurück beliebige Ausgabe, falls S leer ist
isempty(S)	liefert 1, falls S leer ist und 0 sonst

Realisierung in WHILE:

- $S = \langle n, Y \rangle$. $S = \langle 3, \langle 5, \langle 7, \langle 42, 0 \rangle\rangle\rangle\rangle$
- n ist die Anzahl der Elemente, Y der Inhalt.
- Der leere Stack ist $\langle 0, 0 \rangle$.
- Falls $a_1, \ldots, a_n$ in S gespeichert sind, dann ist $Y = \langle a_n, \langle a_{n-1}, \ldots \langle a_1, 0 \rangle \ldots \rangle\rangle$.

Stack: Implementierung

push(S, x)

Umkehrfkt von $\langle .,. \rangle$

$Y := \langle x, \pi_2(S) \rangle$;
$S := \langle \underbrace{\pi_1(S) + 1}, Y \rangle$

Anzahl erhöhen

pop(S)

$n := \pi_1(S)$;
if $n \neq 0$ **then**
 $S := \langle n - 1, \underbrace{\pi_2(\overbrace{\pi_2(S)}^{Y})} \rangle$
fi

Y ohne 1. Element

$x = \text{top}(S)$

$x := \pi_1(\overbrace{\pi_2(S)}^{Y})$

$b = \text{isempty}(S)$

$n := \pi_1(S)$;
if $n > 0$ **then** → Präsblatt
 $b := 0$
else
 $b := 1$
fi

Arrays

Datenstruktur Array

- Ein Array A speichert m Elemente $A[0], \ldots, A[m-1]$.
- Direkter Zugriff auf $A[i]$.
- Zu Beginn ist jeder Eintrag mit 0 initialisiert.

Realisierung in WHILE:

- Realisiert durch einen Stack A (und einen Hilfsstack)
- erlaubt sogar dynamische Arrays

Array: Implementierung

B   A

$a_1, a_2, a_3 \dots a_m$

Ausgabe des
iten Elements von A

0-indiziert

Output: $A[i]$ is returned in x

$B := A$;
for i **do**
  $\text{pop}(B)$
od;
$x := \text{top}(B)$

Simulation von $A[i] := b$??

$B = \emptyset$;
for i **do**
  $\text{push}(B, \text{top}(A))$;
  $\text{pop}(A)$
od;
$\text{pop}(A)$;
$\text{push}(A, b)$;
for i **do**
  $\text{push}(A, \text{top}(B))$;
  $\text{pop}(B)$
od;