

Grundlagen Theoretischer Informatik 3

SoSe 2010 in Trier

Henning Fernau

Universität Trier

fernau@uni-trier.de

Organisatorische Ergänzungen

Die Tutorensprechstunde montags wird auf denselben Ort und dieselbe Zeit dienstags verlegt.

Vorlesungsumlegungen:

Wie besprochen, findet VL4 am MI, 5.5., 10.05-11.35 statt, und zwar im H 6.

Aus persönlichen Gründen würde ich ebenfalls die VL5 verlegen, und zwar auf MI, 12.5., 10.05-11.35 im H 12.

Für die Abschlussklausur gibt es jetzt einen festen Termin und Ort:
DI, 20.7., 9:00-11:00 im HZ 204.

Realistischere While-Programme : Syntax

Das Grundalphabet A besteht aus:

Schlüsselwörtern: $S = \{\mathbf{skip}, \leftarrow, ;, \mathbf{if}, \mathbf{then}, \mathbf{else}, \mathbf{fi}, \mathbf{while}, \mathbf{do}, \mathbf{od}\}$

Booleschen Variablen: $V_{\mathbb{B}}$

Booleschen Operatoren: $O_{\mathbb{B}} = \{\neg, \wedge, \vee, \mathbf{true}, \mathbf{false}, \dots\}$

Zahlvariablen: $V_{\mathbb{N}}$

Zahlenoperatoren: $O_{\mathbb{N}} = \{0, 1, +, -, *, /, \dots\}$

Zahlenprädikaten: $P_{\mathbb{N}} = \{=, <, >, \leq, \geq, \dots\}$

Klammern: $K = \{(,)\}$.

Mit Hilfe einer kontextfreien Grammatik könnte man die Sprachen

$WBA \subseteq (V_{\mathbb{B}} \cup O_{\mathbb{B}} \cup K)^*$ der *wohlgeformten Booleschen Ausdrücke* und

$WAA \subseteq (V_{\mathbb{N}} \cup O_{\mathbb{N}} \cup K)^*$ der *wohlgeformten arithmetischen Ausdrücke* definieren.

Mögliche rekursive Definition von WBAs

Hinweis: Jedem Operator o kann man seine *Stelligkeit* $\alpha(o)$ zuordnen. Wir betrachten im Folgenden nur nullstellige Operatoren (*Konstanten*, hier **true** und **false**), einstellige Operatoren (hier: die Negation \neg) und zweistellige Operatoren.

1. Boolesche Variablen und Konstanten sind WBAs.
2. Ist w ein WBA, so auch $\neg w$.
3. Sind w_1 und w_2 WBAs, so auch $(w_1 \wedge w_2)$ und $(w_1 \vee w_2)$.
4. Nichts anderes sind WBAs.

Ganz entsprechend kann man WAAs definieren.

Wohlgeformte Boolesche Terme WBT

1. WBAs sind WBTs.
2. Sind u_1 und u_2 WAAs, so sind $(u_1 = u_2)$, $(u_1 < u_2)$, $(u_1 > u_2)$, $(u_1 \leq u_2)$, $(u_1 \geq u_2)$ WBTs.
3. Ist w ein WBT, so auch $\neg w$.
4. Sind w_1 und w_2 WBTs, so auch $(w_1 \wedge w_2)$ und $(w_1 \vee w_2)$.
5. Nichts anderes sind WBTs.

Semantik von Ausdrücken

Ist w ein WBA mit (höchstens) den Booleschen Variablen x_1, \dots, x_n , so definiert w eine Boolesche Funktion $f_w : \mathbb{B}^n \rightarrow \mathbb{B}$, $\mathbb{B} = \{0, 1\}$, definiert “entlang” der rek. Def. von WBAs.

1. Boolesche Variablen v und Konstanten c sind WBAs mit natürlicher Semantik f_v bzw. f_c .
2. Ist w ein WBA, so auch $\neg w$, wobei $f_{\neg w} = 1 - f_w$.
3. Sind w_1 und w_2 WBAs, so auch $A = (w_1 \wedge w_2)$ und $B = (w_1 \vee w_2)$ mit $f_A = \min\{f_{w_1}, f_{w_2}\}$ und $f_B = \max\{f_{w_1}, f_{w_2}\}$.

Entsprechend lassen sich WAAs w mit Zahlvariablen y_1, \dots, y_m als Zahlfunktionen $f_w : \mathbb{N}^m \rightarrow \mathbb{N}$ deuten und WBTs w mit Zahlvariablen y_1, \dots, y_m und Booleschen Variablen x_1, \dots, x_n als Funktionen $f_w : \mathbb{N}^m \times \mathbb{B}^n \rightarrow \mathbb{B}$.

Syntax von WHILE-Programmen definiert formale Sprache $L_{\text{while}} \subseteq A^*$:

(1) *Leere Anweisung*: **skip** $\in L_{\text{while}}$.

(2) *Wertzuweisung*: Gilt $v \in V_{\mathbb{B}}$ und ist e ein WBT, so gilt: $v \leftarrow e \in L_{\text{while}}$.

Ist $x \in V_{\mathbb{N}}$ und ist w ein WAA, so gilt: $x \leftarrow w \in L_{\text{while}}$.

(3) *Anweisungsfolge*: Mit S_1 und S_2 aus L_{while} folgt auch $S_1; S_2 \in L_{\text{while}}$.

(4) *Verzweigung*: Mit WBT e und S_1 und S_2 aus L_{while} folgt auch

if e **then** S_1 **else** S_2 **fi** $\in L_{\text{while}}$.

(5) *Schleife*: Mit WBT e und $S \in L_{\text{while}}$ folgt auch **while** e **do** S **od** $\in L_{\text{while}}$.

Nur Wörter über A , die wie in (1) bis (5) beschrieben konstruiert werden können, gehören zu L_{while} .

Ist $p \in L_{\text{while}}$ ein WHILE-Programm mit vorkommenden Booleschen Variablen $V_{\mathbb{B},p}$ und vorkommenden Zahlvariablen $V_{\mathbb{N},p}$, so lässt sich der Berechnungsstand in einer *Konfiguration* $(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$ festhalten, mit $S \in L_{\text{while}} \cup \{\lambda\}$, $\sigma_{\mathbb{B}} \in (\mathbb{B} \cup \{\uparrow\})^{|V_{\mathbb{B},p}|}$, $\sigma_{\mathbb{N}} \in (\mathbb{N} \cup \{\uparrow\})^{|V_{\mathbb{N},p}|}$, \uparrow meint undef.

Konfigurationsübergänge definieren die Semantik von WHILE-Programmen

Hinweis: Dies ist ein einfacher Spezialfall für die Definition einer *operationellen Semantik*.

Mehr zu diesem Thema bieten Spezialvorlesungen.

Es seien $(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$ und $(T, \tau_{\mathbb{B}}, \tau_{\mathbb{N}})$ zwei Konfigurationen.

$(S, \sigma_{\mathbb{B}}, \sigma_{\mathbb{N}}) \Rightarrow (T, \tau_{\mathbb{B}}, \tau_{\mathbb{N}})$ gelte, wenn eine der nachstehenden Bedingungen erfüllt ist.

Ziel: Daraus kann man sofort die Semantik f_S eines WHILE-Programms S definieren als Ausgabewert $\rho = (\rho_{\mathbb{B}}, \rho_{\mathbb{N}})$ bei Eingabe von $\eta = (\eta_{\mathbb{B}}, \eta_{\mathbb{N}})$, falls nämlich $(S, \eta) \Rightarrow^* (\lambda, \rho)$ gilt. Hinweis: Ähnlichkeit mit Automaten und Grammatiken!

(1) $(S = \mathbf{skip}; T$ oder $(S = \mathbf{skip}$ und $T = \lambda))$ und $\tau = \sigma$

Konfigurationsübergänge definieren die Semantik von WHILE-Programmen (Forts.)

Ist w ein Ausdruck, so liefert f_w seine Semantik als Funktion.

Ist $\sigma = (\sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$ eine konkrete Variablenbelegung, so liefert $f_w(\sigma)$ einen konkreten Wert (evtl. \uparrow).

Vereinfachende Notationen:

$\sigma(x)$ für Variable x liest den richtigen Wert im Tupel $(\sigma_{\mathbb{B}}, \sigma_{\mathbb{N}})$ aus.

$\sigma[x/f_w(\sigma)]$ ersetzt (nur) den Wert von x in σ durch den Wert $f_w(\sigma)$.

(2) $S = x \leftarrow t; T$ oder $(S = x \leftarrow t \text{ und } T = \lambda)$ und $\tau = \sigma[x/f_t(\sigma)]$.

Hierbei ist t WAA oder WBT und x vom passenden Typ \mathbb{B} bzw. \mathbb{N} .

Konfigurationsübergänge definieren die Semantik von WHILE-Programmen (Forts.)

(3a) $S = \mathbf{if\ } e \mathbf{\ then\ } S_1 \mathbf{\ else\ } S_2 \mathbf{\ fi}; S_3$ mit $\tau = \sigma$ und

$$T = \begin{cases} S_1; S_3 & \text{falls } f_e(\sigma) = 1 \\ S_2; S_3 & \text{falls } f_e(\sigma) = 0 \end{cases}$$

(3b) $S = \mathbf{if\ } e \mathbf{\ then\ } S_1 \mathbf{\ else\ } S_2 \mathbf{\ fi}$ mit $\tau = \sigma$ und

$$T = \begin{cases} S_1 & \text{falls } f_e(\sigma) = 1 \\ S_2 & \text{falls } f_e(\sigma) = 0 \end{cases}$$

Konfigurationsübergänge definieren die Semantik von WHILE-Programmen (Forts.)

(4a) $S = \mathbf{while\ } e \mathbf{\ do\ } S_1 \mathbf{\ od}; S_2$ mit $\tau = \sigma$ und

$$T = \begin{cases} S_1; S & \text{falls } f_e(\sigma) = 1 \\ S_2 & \text{falls } f_e(\sigma) = 0 \end{cases}$$

(4b) $S = \mathbf{while\ } e \mathbf{\ do\ } S_1 \mathbf{\ od}$ mit $\tau = \sigma$ und

$$T = \begin{cases} S_1; S & \text{falls } f_e(\sigma) = 1 \\ \lambda & \text{falls } f_e(\sigma) = 0 \end{cases}$$

Beispiel einer Konfigurationsfolge

Betrachte das folgende Programm:

$a \leftarrow 0;$

$b \leftarrow x;$

while $b \geq y$ **do**

$b \leftarrow b - y;$

$a \leftarrow a + 1$

od

$a, b, x, y \in V_{\mathbb{N}}$.

Betrachte $\sigma(x) = 14, \sigma(y) = 5$.

} $=: S_1$

$(S, \sigma) = (a \leftarrow 0; b \leftarrow x; S_1, \sigma = (\uparrow, \uparrow, 14, 5))$

$\Rightarrow (b \leftarrow x; S_1, (0, \uparrow, 14, 5))$

$\Rightarrow (S_1, (0, 14, 14, 5))$

$\Rightarrow (b \leftarrow b - y; a \leftarrow a + 1; S_1, (0, 14, 14, 5))$

$\Rightarrow (a \leftarrow a + 1; S_1, (0, 9, 14, 5))$

$\Rightarrow (S_1, (1, 9, 14, 5))$

$\Rightarrow (b \leftarrow b - y; a \leftarrow a + 1; S_1, (1, 9, 14, 5))$

$\Rightarrow (a \leftarrow a + 1; S_1, (1, 4, 14, 5))$

$\Rightarrow (S_1, (2, 4, 14, 5))$

$\Rightarrow (\lambda, (2, 4, 14, 5))$

Was aber liefert das Programm als Ausgabe ((a, b) -Werte) für bel. (x, y) -Eingaben??

Funktionale Interpretation von WHILE-Programmen

Mit der Schreibweise $M \langle S \rangle$ anstelle von f_S für WHILE-Programme S können wir formulieren:

Satz: Für alle Variablenbelegungen σ gilt:

$$(1) M \langle \mathbf{skip} \rangle (\sigma) = \sigma.$$

$$(2) M \langle x \leftarrow t \rangle (\sigma) = \sigma[x/f_t(\sigma)] \text{ für alle WBT oder WAA } t.$$

$$(3) M \langle S; T \rangle (\sigma) = M \langle T \rangle (M \langle S \rangle (\sigma)).$$

$$(4) M \langle \mathbf{if } e \mathbf{ then } S_1 \mathbf{ else } S_2 \mathbf{ fi} \rangle (\sigma) = \begin{cases} M \langle S_1 \rangle (\sigma) & \text{falls } f_e(\sigma) = 1 \\ M \langle S_2 \rangle (\sigma) & \text{falls } f_e(\sigma) = 0 \end{cases}$$

$$(5) M \langle \mathbf{while } e \mathbf{ do } S \mathbf{ od} \rangle (\sigma) = \begin{cases} M \langle \mathbf{while } e \mathbf{ do } S \mathbf{ od} \rangle (M \langle S \rangle (\sigma)) & \text{falls } f_e(\sigma) = 1 \\ \sigma & \text{falls } f_e(\sigma) = 0 \end{cases}$$

Anmerkungen

- M betrachtet Programme als Variablenbelegungstransformation.
- Dies entspricht im Wesentlichen unserer alten Deutung von WHILE-berechenbaren Funktionen.
- Dadurch sollte auch der Bezug zu Registermaschinen klarer sein, da die Variablenbelegungen (Registerinhalte) immer explizit benannt werden.
- Die in den Konfigurationen genannten Programmstückchen entsprechen dabei dem Befehlszähler.
- Konfigurationsfolgen definieren eine Art Interpreter für WHILE-Programme.
- Unsere Darstellung vereinfacht manche abstrakteren Darstellungen aus der Literatur, da wir uns auf das “Universum” \mathbb{BUN} beschränken und auch die unterliegende Logik deutlich einschränken. Das erschwert auch die formale Unterscheidung zwischen syntaktischem und Modell-basiertem Ableitungsbegriff in der Logik.

Die Hoaresche Logik

Ein *Hoarescher Ausdruck* ist eine Zeichenkette der Form

$$\{p\} S \{q\}$$

Dabei ist p ein WBT (die *Vorbedingung* von S), $S \in L_{\text{while}}$ und q ein WBT (die *Nachbedingung* von S). Das Paar (p, q) heißt auch *Spezifikation*.

Bedeutung: Für alle Variablenbelegungen σ soll gelten:

Ist $f_p(\sigma) = 1$ oder terminiert S auf σ nicht, so gilt auch $f_q(\mathcal{M}\langle S\rangle(\sigma)) = 1$.

Wir schreiben dann auch: $\mathcal{M}\langle\{p\}S\{q\}\rangle(\sigma) = 1$.

Die Hoaresche Logik Beispiele

Gilt $M \langle \{x \geq 5\} x \leftarrow 2 * x \{x \geq 20\} \rangle = 1$?

Dann müsste für alle $x \in \mathbb{N}$ gelten:

Aus $x \geq 5$ folgt $2 * x \geq 20$.

Das ist logisch äquivalent zu $(x < 5) \vee (x \geq 10)$ und lässt somit eine “Lücke” für Gegenbeispiele, z.B. $x = 5$.

Hingegen gilt: $M \langle \{x \geq 10\} x \leftarrow 2 * x \{x \geq 20\} \rangle = 1$,

denn das ist logisch äquivalent zur Tautologie $(x < 10) \vee (x \geq 10)$.

Formaler ist zu zeigen: Für alle Variablenbelegungen σ mit $\sigma(x) \geq 10$ gilt:

$f_{x \geq 20}(M \langle x \leftarrow 2 * x \rangle (\sigma)) = 1$ (da Terminierung klar).

Die letzte Bedingung gilt nach vorigem Satz gdw.:

$f_{x \geq 20}(\sigma[x/f_{2*x}(\sigma)]) = f_{x \geq 20}(\sigma[x/2 * \sigma(x)]) = 1$.

Da σ beliebig aber fest, können wir in den WBT $x \geq 20$ einsetzen und erhalten als äquivalente Bedingung:

$2 * \sigma(x) \geq 20$, was aus der Bedingung $\sigma(x) \geq 10$ folgt.

Die Hoaresche Logik Beispiele

Es gilt: $\{\mathbf{true}\} x \leftarrow y + 1 \{x > y\}$.

Formal zu zeigen ist (da Terminierung klar):

$f_{x>y}(M \langle x \leftarrow y + 1 \rangle (\sigma)) = 1$ für alle Variablenbelegungen σ .

Nach vorigem Satz ist diese Bedingung gleichwertig mit:

$$f_{x>y}(\sigma[x/f_{y+1}(\sigma)]) = f_{x>y}(\sigma[x/(\sigma(y) + 1)]) = 1.$$

Da σ beliebig aber fest, können wir in den WBT $x > y$ einsetzen und erhalten als äquivalente Bedingung:

$\sigma(y) + 1 > \sigma(y)$. Dies gilt für beliebige Zahlen $\sigma(y)$.

Die Hoaresche Logik Beispiele

Es gilt: $\{\mathbf{true}\} a \leftarrow 0; b \leftarrow x \{a = 0 \wedge b = x\}$.

Formal ist zu zeigen für beliebige Variablenbelegungen σ :

$$f_{a=0 \wedge b=x}(M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma)) = 1.$$

Nach dem Satz gilt:

$$\begin{aligned} M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma) &= M \langle b \leftarrow x \rangle (M \langle a \leftarrow 0 \rangle (\sigma)) \\ &= M \langle b \leftarrow x \rangle (\sigma[a/0]) \\ &= \sigma[a/0][b/\sigma[a/0](x)] \\ &= \sigma[a/0][b/\sigma(x)] \end{aligned}$$

Da die Variablenbelegungsänderungen einander nicht beeinflussen, gilt für die sich ergebene Belegung $\sigma[a/0][b/\sigma(x)]$ die behauptete Eigenschaft

$$\sigma[a/0][b/\sigma(x)](a) = 0 \wedge \sigma[a/0][b/\sigma(x)](b) = \sigma[a/0][b/\sigma(x)](x)$$

(für jede Bel. σ !), was zu zeigen war.

Die Hoaresche Logik Beispiele

Es gilt: $b := M \langle \{\mathbf{true}\} \mathbf{while} \ x \neq 10 \ \mathbf{do} \ x \leftarrow x + 1 \ \mathbf{od} \ \{x = 10\} \rangle = 1.$

Für eine bel. Variablenbelegung σ gilt anfangs sicher $\sigma(x) \leq 10$ oder $\sigma(x) > 10$. Im zweiten Fall terminiert die Schleife nicht, da die Bedeutung der Hoareschen Logik aber nur die *partielle Korrektheit* einfordert, gilt $b = 1$.

Im ersten Fall zeigt ein einfacher Induktionsbeweis die Behauptung. Dazu behaupten wir: Gilt $\sigma(x) = 10 - i$ für ein $i \in \mathbb{N}$ für eine Anfangsbelegung σ , so wird der *Schleifenrumpf* $x \leftarrow x + 1$ genau i -mal ausgeführt, und beim Verlassen der Schleife gilt: $x = 10$.

Induktionsanfang: Für $i = 0$ gilt: $\sigma(x) = 10$, also zu zeigen:

$f_{x=10}(M \langle \mathbf{while} \ x \neq 10 \ \mathbf{do} \ x \leftarrow x + 1 \ \mathbf{od} \rangle (\sigma)) = 1$ für obiges σ .

Natürlich ist $f_{x \neq 10}(\sigma) = 0$, wenn $\sigma(x) = 10$ gilt, denn $\sigma(x) \neq 10$ ist falsch.

Offensichtlich wird der Schleifenrumpf gar nicht ausgeführt.

Induktionsschritt: Es sei die Beh. bewiesen für $i = 0, \dots, I, I \geq 0$. Betrachte sie für $i = I + 1$.

Jetzt gilt: $f_{x \neq 10}(\sigma) = 1$ gdw. $\sigma(x) \neq 10$ gdw. $10 - (I + 1) \neq 10$ gdw. $I + 1 \neq 0$.

Die letzte Bedingung ist klar wegen der Voraussetzung $I \geq 0$.

Also wird der Schleifenrumpf wenigstens einmal ausgeführt, und es gilt:

$f_{x=10}(M \langle \mathbf{while} \ x \neq 10 \ \mathbf{do} \ x \leftarrow x + 1 \ \mathbf{od} \rangle (\sigma)) = 1$ für obiges σ gdw.

$f_{x=10}(M \langle \mathbf{while} \ x \neq 10 \ \mathbf{do} \ x \leftarrow x + 1 \ \mathbf{od} \rangle (M \langle x \leftarrow x + 1 \rangle (\sigma))) = 1$. (*)

Es gilt: $(x \leftarrow x + 1, 10 - (I + 1)) \Rightarrow (\lambda, 10 - (I + 1) + 1) = (\lambda, 10 - I)$

(Erinnere Konfigurationsübergangsnotation).

Also lässt sich auf $\sigma' = \sigma[x/(\sigma(x) + 1)]$ die Induktionsvoraussetzung anwenden.

(*) ist gleichwertig mit: $f_{x=10}(M \langle \mathbf{while} \ x \neq 10 \ \mathbf{do} \ x \leftarrow x + 1 \ \mathbf{od} \rangle (\sigma')) = 1$.

Dies ist nach IV richtig, und außerdem wurde der Schleifenrumpf $I + 1$ -mal ausgeführt.

Die Überlegungen aus dem letzten Beispiel lassen sich verallgemeinern zu:

Lemma: Es sei e WBT, $S \in L_{\text{while}}$, $f = M \langle \mathbf{while} \ e \ \mathbf{do} \ S \ \mathbf{od} \rangle$ und $g = M \langle S \rangle$.

Dann gilt für beliebige Belegungen σ :

1. $f(\sigma) = g^k(\sigma)$, falls es ein $k \geq 0$ gibt mit $f_e(g^k(\sigma)) = 0$ und $f_e(g^i(\sigma)) = 1$ für alle $0 \leq i < k$.
2. $f(\sigma)$ ist ansonsten undefiniert.

Die Hoaresche Logik Ein letztes Beispiel

Betrachte das folgende Programm:

$a \leftarrow 0;$

$b \leftarrow x;$

while $b \geq y$ **do**

$b \leftarrow b - y;$

$a \leftarrow a + 1$

od

$a, b, x, y \in V_{\mathbb{N}}$.

Wir wissen bereits:

$\{\mathbf{true}\} a \leftarrow 0; b \leftarrow x \{a = 0 \wedge b = x\}$.

} $\equiv S_1$

Gilt $\sigma(x) < \sigma(y)$, so auch $(M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma))(b) < \sigma(y) = (M \langle a \leftarrow 0; b \leftarrow x \rangle (\sigma))(y)$, also wird der Schleifenrumpf nie betreten, und die "Ausgabe" liefert für das Variablenpaar (a, b) : $(0, \sigma(x))$.

Wie oben sieht man durch Induktion über die Anzahl i von Schleifenrumpfdurchläufen für eine Belegung σ_i am Ende des i -ten Durchlaufs:

1. $\sigma_i(a) = i$ und
2. $\sigma_i(b) = \sigma(x) - i * \sigma(y)$.

Also gilt das auch bei Schleifenabbruch, wobei dann zusätzlich $\sigma_i(b) < \sigma_i(y) = \sigma(y)$ gilt. Das gewährt, das schließlich das Variablenpaar (a, b) den Wert

$(\sigma(x) \text{ DIV } \sigma(y), \sigma(x) \text{ mod } \sigma(y))$

enthält.