Prof. Martin Hofmann
Dr. Ulrich Schöpp

Ludwig-Maximilians-Universität München
Institut für Informatik
SS 2014

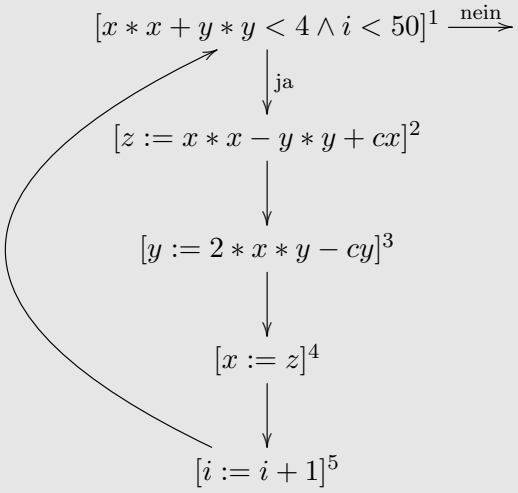
Übungen zur Vorlesung
Formale Spezifikation und Verifikation

Blatt 7

Aufgabe 7-1 Geben Sie für folgendes Programm den Kontrollflussgraphen an und bestimmen Sie die Available Expressions für beide Programme, d.h. berechnen Sie jeweils die größte Lösung der Gleichungen für $AE_{entry}(l)$ und $AE_{exit}(l)$ für alle Programmpunkte l.

$$\texttt{while}\ [x * x + y * y < 4 \wedge i < 50]^1\ \texttt{do}$$
$$\left([z := x * x - y * y + cx]^2;\ [y := 2 * x * y - cy]^3;\ [x := z]^4;\ [i := i + 1]^5\right)$$

Lösung: Kontrollflussgraph



Available Expressions:

$$\begin{array}{lcl} AE_{entry}(\ell) & = & \bigcap_{\ell':\ell' \to \ell} AE_{exit}(\ell') \\ AE_{exit}(\ell) & = & (AE_{entry}(\ell) \setminus kill_{AE}(B^\ell)) \cup gen_{AE}(B^\ell) \end{array}$$

wobei B^ℓ das durch ℓ beschriftete Programmstück ("Block") ist und

$$\begin{array}{lcl} kill_{AE}([x\texttt{:=}a]^\ell) & = & \{a' \mid x \in FV(a')\} \\ kill_{AE}([\texttt{skip}]^\ell) & = & \emptyset \\ kill_{AE}([b]^\ell) & = & \emptyset \\ gen_{AE}([x\texttt{:=}a]^\ell) & = & \{a' \mid a' \text{ Teilausdruck von } a \text{ und } x \notin FV(a')\} \\ gen_{AE}([\texttt{skip}]^\ell) & = & \emptyset \\ gen_{AE}([b]^\ell) & = & \{a' \mid a' \text{ Teilausdruck von } b\} \end{array}$$

In diesem Beispiel erhalten wir:

$$
\begin{aligned}
kill(1) &= \emptyset \\
kill(2) &= \emptyset \\
kill(3) &= \{(y*y), ((2*x)*y), ((x*x)+(y*y)), (((x*x)-(y*y))+cx), \\
&\qquad ((x*x)-(y*y)), (((2*x)*y)-cy)\} \\
kill(4) &= \{(2*x), (x*x), ((2*x)*y), ((x*x)+(y*y)), (((x*x)-(y*y))+cx), \\
&\qquad ((x*x)-(y*y)), (((2*x)*y)-cy)\} \\
kill(5) &= \{(i+1)\} \\
gen(1) &= \{(x*x), (y*y), ((x*x)+(y*y))\} \\
gen(2) &= \{(x*x), (y*y), (((x*x)-(y*y))+cx), ((x*x)-(y*y))\} \\
gen(3) &= \{(2*x)\} \\
gen(4) &= \emptyset \\
gen(5) &= \emptyset \\
AE_{entry}(1) &= \emptyset \\
AE_{entry}(2) &= \{(x*x), (y*y), ((x*x)+(y*y))\} \\
AE_{entry}(3) &= \{(x*x), (y*y), ((x*x)+(y*y)), (((x*x)-(y*y))+cx), ((x*x)-(y*y))\} \\
AE_{entry}(5) &= \emptyset \\
AE_{entry}(4) &= \{(2*x), (x*x)\} \\
AE_{exit}(1) &= \{(x*x), (y*y), ((x*x)+(y*y))\} \\
AE_{exit}(2) &= \{(x*x), (y*y), ((x*x)+(y*y)), (((x*x)-(y*y))+cx), ((x*x)-(y*y))\} \\
AE_{exit}(3) &= \{(2*x), (x*x)\} \\
AE_{exit}(5) &= \emptyset \\
AE_{exit}(4) &= \emptyset
\end{aligned}
$$

Aufgabe 7-2 Eine mögliche Anwendung der Available Expressions ist, die wiederholte Auswertung von Ausdrücken bei der Programmausführung zu vermeiden. Der Wert verfügbarer Ausdrücke kann gespeichert werden und dann später ohne Neuauswertung des Ausdrucks benutzt werden. Möchte man keinen zusätzlichen Speicher verwenden, so kann man sich auf verfügbare Ausdrücke beschränken, deren Wert in einer bestimmten Programmvariable verfügbar ist: Ein Ausdruck a ist an einem Programmpunkt in Variable x verfügbar, falls die Variable x an diesem Programmpunkt den Wert des Ausdrucks a speichert.

Passen Sie die Gleichungen für die Available Expressions (d.h. für $AE_{entry}(l)$ und $AE_{exit}(l)$) so an, dass für jeden Programmpunkt die Menge aller Paare (x, a) berechnet wird, für die an diesem Programmpunkt der Ausdruck a in Variable x verfügbar ist.

Lösung: Es genügt, die Definitionen für gen_{AE} und $kill_{AE}$ anzupassen.

$$\begin{aligned} kill_{AE}([b]^l) &= \emptyset \\ kill_{AE}([skip]^l) &= \emptyset \\ kill_{AE}([x := a]^l) &= \{(y, a') \mid y = x \text{ oder } x \text{ ist freie Variable in } a'\} \end{aligned}$$

(Das Paar (y, a') könnte ungültig werden, wenn sich der Wert der Variablen y ändert oder wenn eine in a' vorkommende Variable geändert wird.)

$$\begin{aligned} gen_{AE}([skip]^l) &= \emptyset \\ gen_{AE}([b]^l) &= \emptyset \\ gen_{AE}([x := a]^l) &= \begin{cases} \{(x, a)\} & \text{falls } x \text{ keine freie Variable von } a \text{ ist} \\ \emptyset & \text{sonst} \end{cases} \end{aligned}$$

Durch die Zuweisung $x := a$ wird im Prinzip das Paar (x, a) generiert. Wenn x in a vorkommt, bezieht sich der Wert dieser Variablen aber auf den Wert von x vor der Zuweisung. Durch die Zuweisung könnte sich der Wert des Ausdrucks a also ändern, wenn x in a vorkommt. Deshalb wird (x, a) nur generiert, wenn x nicht frei in a vorkommt.

Aufgabe 7-3 Gegeben sei ein vollständiger Verband $(L, \sqsubseteq)$.
Dann ist die Menge $L \times L$ ebenfalls ein vollständiger Verband bezüglich der Ordnung

$$(x, y) \sqsubseteq (x', y') \quad \iff \quad x \sqsubseteq x' \text{ und } y \sqsubseteq y'.$$

Seien $F_1 \colon L \times L \to L$ und $F_2 \colon L \times L \to L$ zwei monotone Funktionen, d.h. aus $(x, y) \sqsubseteq (x', y')$ folgt $F_1(x, y) \sqsubseteq F_1(x', y')$ und analog für F_2.
Definiere $F, G \colon L \times L \to L \times L$ wie folgt:

$$\begin{aligned} F(x, y) &= (F_1(x, y), F_2(x, y)) \\ G(x, y) &= (F_1(x, y), F_2(F_1(x, y), y)) \end{aligned}$$

Zeigen Sie:

a) $L \times L$ ist ein vollständiger Verband.

Lösung: Man prüft, dass die definierte Ordnung $\sqsubseteq$ eine partielle Ordnung ist, also dass sie reflexiv, transitiv und antisymmetrisch ist.

Es bleibt zu zeigen, dass jede Teilmenge $U \subseteq L \times L$ ein Supremum $\bigsqcup U$ in $L \times L$ hat. Sei $U \subseteq L \times L$ eine beliebige Teilmenge. Definiere

$$\begin{aligned} U_1 &= \{u_1 \mid \exists u_2 . (u_1, u_2) \in U\} \subseteq L \\ U_2 &= \{u_2 \mid \exists u_1 . (u_1, u_2) \in U\} \subseteq L \end{aligned}$$

Da L ein vollständiger Verband ist, haben diese Mengen Suprema $\bigsqcup U_1$ und $\bigsqcup U_2$ in L. Wir zeigen, dass das Paar $(\bigsqcup U_1, \bigsqcup U_2)$ das Supremum von U in $L \times L$ ist.

Wir müssen zeigen, dass dieses Paar die kleinste obere Schranke von U ist. Dazu ist zuerst zu zeigen, dass es überhaupt eine obere Schranke ist, also dass jedes Element von U kleiner-gleich diesem Paar ist. Sei $(u_1, u_2) \in U$. Es gilt sicher $u_1 \in U_1$ und, da $\bigsqcup U_1$ eine obere Schranke für U_1 ist, dann auch $u_1 \sqsubseteq \bigsqcup U_1$. Analog folgt $u_2 \sqsubseteq \bigsqcup U_2$. Daraus folgt aber $(u_1, u_2) \sqsubseteq (\bigsqcup U_1, \bigsqcup U_2)$, also ist $(\bigsqcup U_1, \bigsqcup U_2)$ eine obere Schranke für U.

Nun bleibt noch zu zeigen, dass $(\bigsqcup U_1, \bigsqcup U_2)$ die kleinste obere Schranke von U ist, d.h. dass diese obere Schranke kleiner-gleich allen anderen oberen Schranken ist. Angenommen $s = (s_1, s_2)$ ist eine beliebige obere Schranke von U. Da $u \sqsubseteq s$ für alle $u \in U$ gilt, muss auch $u_1 \sqsubseteq s_1$ für alle $u_1 \in U_1$ sowie $u_2 \sqsubseteq s_2$ für alle $u_2 \in U_2$ gelten. (Denn $u_1 \in U_1$ bedeutet, dass $(u_1, v) \in U$ für ein v gilt. Dann gilt aber $(u_1, v) \sqsubseteq (s_1, s_2)$, da s eine obere Schranke ist, und es folgt $u_1 \sqsubseteq s_1$ nach Definition.) Das heißt, s_1 und s_2 sind obere Schranken für U_1 und U_2. Daraus folgt $\bigsqcup U_1 \sqsubseteq s_1$ und $\bigsqcup U_2 \sqsubseteq s_2$, da $\bigsqcup U_1$ und $\bigsqcup U_2$ die kleinsten oberen Schranken sind. Es folgt $(\bigsqcup U_1, \bigsqcup U_2) \sqsubseteq s$ nach Definition von $\sqsubseteq$, also ist $(\bigsqcup U_1, \bigsqcup U_2)$ die kleinste obere Schranke.

b) Sowohl F als auch G sind monoton.

Lösung: Wir zeigen den Fall für G. Der Beweis für F geht ganz analog. Angenommen $(x, y) \sqsubseteq (x', y')$. Dann gilt auch $F_1(x, y) \sqsubseteq F_1(x', y')$ (wegen Monotonie von F_1) und $y \sqsubseteq y'$ (Definition von $\sqsubseteq$ für $L \times L$), also auch $(F_1(x, y), y)) \sqsubseteq (F_1(x', y'), y')$ (Definition von $\sqsubseteq$ für $L \times L$) und damit $F_2(F_1(x, y), y) \sqsubseteq F_2(F_1(x', y'), y')$ (Monotonie von F_2). Es folgt $(F_1(x, y), F_2(F_1(x, y), y)) \sqsubseteq (F_1(x', y'), F_2(F_1(x', y'), y'))$ (Definition von $\sqsubseteq$ für $L \times L$), also $G(x, y) \sqsubseteq G(x', y')$, was zu zeigen war.

c) Der kleinste Fixpunkt von F und G ist gleich.

Lösung: Zeige: Jeder Fixpunkt von F ist auch einer von G und umgekehrt. Dann sind die kleinsten Fixpunkte natürlich gleich.

Sei (U, V) ein Fixpunkt von F, d.h. $F(U, V) = (U, V)$. Das bedeutet $U = F_1(U, V)$ und $V = F_2(U, V)$ nach Definition. Dann gilt aber $G(U, V) = (F_1(U, V), F_2(F_1(U, V), V)) = (U, F_2(U, V)) = (U, V)$, also ist (U, V) ein Fixpunkt von G.

In die andere Richtung, sei (U, V) ein Fixpunkt von G, d.h. $U = F_1(U, V)$ und $V = F_2(F_1(U, V), V)$. Dann folgt aber $F_2(F_1(U, V), V) = F_2(U, V)$. Es folgt $U = F_1(U, V)$ und $V = F_2(U, V)$, d.h. (U, V) ist ein Fixpunkt von F.

Aufgabe 7-4 In der Vorlesung wurde das Alternating Bit Protokoll in NuSMV implementiert und seine Korrektheit überprüft.

In der Implementierung des Protokolls wird angenommen, dass Sender und Empfänger über Kanäle kommunizieren, die Nachrichten entweder unverfälscht weitergeben oder unlesbar machen. Es werden zwei Arten von Kanälen verwendet: Ein Zwei-Bit-Kanal `two_bit_channel` überträgt in jedem Zeitschritt zwei Bit vom Sender zum Empfänger und ein Ein-Bit-Kanal `one_bit_channel` überträgt in jedem Zeitschritt ein Bit in die andere Richtung.

```
sen : sender();
rec : receiver();
out_c : two_bit_channel(sen.msg, sen.bit, rec.in0, rec.in1);
ret_c : one_bit_channel(rec.out, sen.in0);
```

Beim Zwei-Bit-Kanal `two_bit_channel` ist die Annahme, dass entweder beide Bits korrekt übertragen werden oder beide verfälscht werden.

Angenommen wir wollen das Alternating Bit Protokoll nur mit Ein-Bit-Kanälen implementieren und zwar so, dass die beiden Bits vom Sender zum Empfänger über zwei parallele Kanäle übertragen werden. Das System wird dann wie folgt zusammengesetzt:

```
sen : sender();
rec : receiver();
out_msg_c : one_bit_channel(sen.msg, rec.in0);
out_bit_c : one_bit_channel(sen.bit, rec.in1);
ret_c : one_bit_channel(rec.out, sen.in0);
```

Eine vollständige SMV-Datei `abp1.smv` mit dieser Änderung finden Sie auf der Vorlesungshomepage.

a) Warum ist `abp1.smv` keine korrekte Implementierung des Alternating Bit Protokolls?

Lösung: NuSMV liefert ein Gegenbeispiel für die Eigenschaft `AG (sen.state = sent -> AX (sen.msg = c_0 -> A [ rec.result = c_X U rec.result = c_0 ] ))`, an dem folgendes Problem der Implementierung ablesen kann:

Der Empfänger prüft nur das Kontrollbit und geht in den Zustand `received` über, auch wenn er gar keine eigentliche Nachricht empfangen hat. Er signalisiert dann dem Sender, dass dieser eine neue Nachricht senden kann. Somit kann es passieren, dass der Sender schon eine neue Nachricht `c_1` sendet, ohne dass der Empfänger die Nachricht `c_0` jemals empfangen hat, und dass der Empfänger dann nur `c_1` empfängt.

b) Passen Sie die Module `sender` und `receiver` in `abp1.smv` so an, dass das Alternating Bit Protokoll korrekt implementiert wird. Ihre Implementierung soll alle in `abp1.smv` gegebenen Spezifikationen erfüllen. Die in Ihrer Implementierung auf den Kanälen gesendeten Nachrichten sollen von der Implementierung aus der Vorlesung nur in dem Fall abweichen, dass einer der Kanäle `out_msg_c` und `out_bit_c` in einem Zeitschritt eine Nachricht verfälscht, der andere aber nicht.

Lösung: Eine Möglichkeit ist, dem Empfänger einen neuen Zustand zu geben, so dass er nach dem Empfangen des Kontrollbits in einen Zustand `receivingmsg` übergeht, in dem er nun auf den tatsächlichen Empfang der Nachricht wartet. Erst wenn beide Bits angekommen sind geht er in den Zustand `received` über. Eine vollständige Beispiellösung finden Sie auf der Vorlesungshomepage.