DD2457 Program Semantics and Analysis

EXAMINATION PROBLEMS
WITH SOLUTIONS
29 May 2017, 14:00 - 19:00

Dilian Gurov
KTH CSC
tel: 08-790 8198

1 Level E

For passing level E you need 8 (out of 10) points from this section.

Consider the following program:

$$x := 0;\ \textbf{while}\ \neg(y \leq 0)\ \textbf{do}\ x := x + 1;\ y := y - 1$$

which has the purpose to copy (incrementally) the initial value of variable y to the final value of variable x, implicitly assuming that the former is not negative.

1. Specify the program by means of a *Hoare triple*. The specification should make sense on its own, i.e., it should tell the user how to use the program without knowing the implementation. [2p]

   Solution:

   We make explicit the assumption about the non-negative initial value of variable y, introduce a logical variable y_0 to which we bind this value in the pre-condition, and specify in the post-condition that it should be equal to the final value of variable x:

   $$\{y \geq 0 \wedge y = y_0\}\, x := 0;\ \textbf{while}\ \neg(y \leq 0)\ \textbf{do}\ x := x + 1;\ y := y - 1\, \{x = y_0\}$$

2. Suggest a *loop invariant* that is suitable for verifying the program. [1p]

   Solution:

   A good choice turns out to be: $x + y = y_0 \wedge y \geq 0$. The left conjunct is the essential invariant property of the loop body that makes the algorithm work, while the second conjunct is needed to take into account the loop guard.

   Your next task will be to verify the program against your specification and loop invariant by means of *symbolic execution* (see handouts).

3. That is, first, convert the Hoare triple and loop invariant into a *loop-free program* in the **While** language extended with **assume**, **assert** and **havoc** statements. [1p]

   Solution:

$$\begin{array}{rcl}
l_0 & : & \textbf{assume}\ y \geq 0; \\
l_1 & : & x := 0; \\
l_2 & : & \textbf{assert}\ x + y = y_0 \wedge y \geq 0; \\
l_3 & : & \textbf{havoc}\ x, y; \\
l_4 & : & \textbf{assume}\ x + y = y_0 \wedge y \geq 0; \\
l_5 & : & \textbf{if}\ \neg(y \leq 0)\ \textbf{then} \\
l_6 & : & \quad x := x + 1; \\
l_7 & : & \quad y := y - 1; \\
l_8 & : & \quad \textbf{assert}\ x + y = y_0 \wedge y \geq 0; \\
l_9 & : & \quad \textbf{assume false}; \\
 & : & \textbf{else} \\
l_{10} & : & \quad \textbf{skip}; \\
l_{11} & : & \textbf{assert}\ x = y_0; \\
l_F & : &
\end{array}$$

4. Next, explore all paths. 3p

Solution:

Symbolic execution starts with the path:

$$\begin{array}{rl} & \langle l_0,\ [x \mapsto x_0,\ y \mapsto y_0],\ \mathbf{true}\rangle \\ \Rightarrow & \langle l_1,\ [x \mapsto x_0,\ y \mapsto y_0],\ y_0 \geq 0\rangle \\ \Rightarrow & \langle l_2,\ [x \mapsto 0,\ y \mapsto y_0],\ y_0 \geq 0\rangle \\ \overset{(1)}{\Rightarrow} & \langle l_3,\ [x \mapsto 0,\ y \mapsto y_0],\ y_0 \geq 0\rangle \\ \Rightarrow & \langle l_4,\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0\rangle \\ \Rightarrow & \langle l_5,\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0\rangle \end{array}$$

which then splits into two, a *then*-branch:

$$\begin{array}{rl} \Rightarrow & \langle l_6,\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0)\rangle \\ \Rightarrow & \langle l_7,\ [x \mapsto x_1 + 1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0)\rangle \\ \Rightarrow & \langle l_8,\ [x \mapsto x_1 + 1,\ y \mapsto y_1 - 1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0)\rangle \\ \overset{(2)}{\Rightarrow} & \langle l_9,\ [x \mapsto x_1 + 1,\ y \mapsto y_1 - 1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0)\rangle \\ \Rightarrow & \langle l_{11},\ [x \mapsto x_1 + 1,\ y \mapsto y_1 - 1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0) \wedge \mathbf{false}\rangle \end{array}$$

which turns infeasible from here on, and an *else*-branch:

$$\begin{array}{rl} \Rightarrow & \langle l_{10},\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg\neg(y_1 \leq 0)\rangle \\ \Rightarrow & \langle l_{11},\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg\neg(y_1 \leq 0)\rangle \\ \overset{(3)}{\Rightarrow} & \langle l_F,\ [x \mapsto x_1,\ y \mapsto y_1],\ y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg\neg(y_1 \leq 0)\rangle \end{array}$$

which terminates and is feasible.

5. And finally, collect all resulting *verification conditions* and argue semi-formally for their validity. 3p

Note that if you chose too weak a loop invariant, this would show itself only here, resulting in verification conditions that are not valid (the right-hand side of the implication will not be a logical consequence of the left-hand side).

Solution:

The three verification conditions resulting from the above symbolic execution are:

(1) $y_0 \geq 0 \ \Rightarrow\ 0 + y_0 = y_0 \wedge y_0 \geq 0$
Valid, since $0 + y_0 = y_0$ is a theorem of Integer Arithmetic, while $y_0 \geq 0$ is a premise.

(2) $y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg(y_1 \leq 0) \ \Rightarrow\ (x_1 + 1) + (y_1 - 1) = y_0 \wedge y_1 - 1 \geq 0$
Valid, since $(x_1 + 1) + (y_1 - 1) = y_0$ simplifies to $x_1 + y_1 = y_0$, which is a premise, while $y_1 - 1 \geq 0$ is entailed by the premise $\neg(y_1 \leq 0)$.

(3) $y_0 \geq 0 \wedge x_1 + y_1 = y_0 \wedge y_1 \geq 0 \wedge \neg\neg(y_1 \leq 0) \ \Rightarrow\ x_1 = y_0$
Valid, since $y_1 \geq 0 \wedge \neg\neg(y_1 \leq 0)$ entails $y_1 = 0$, which together with $x_1 + y_1 = y_0$ entails $x_1 = y_0$.

So, we can conclude that the program is *correct* with respect to its specification.

2 Level C

For grade D you need to have passed level E and obtained 5 (out of 12) points from this section. For passing level C you need 8 points from this section.

1. Consider the *denotational semantics* of **While**. In class, we suggested the following definition for the **repeat** S **until** b statement: [6p]

$$\mathcal{S}_{ds}[\![\textbf{repeat } S \textbf{ until } b]\!] \stackrel{\mathsf{def}}{=} \mathsf{FIX}\ H_{S,b}$$

where:

$$H_{S,b}\ (g) \stackrel{\mathsf{def}}{=} \textbf{cond}(\mathcal{B}[\![b]\!], id, g) \circ \mathcal{S}_{ds}[\![S]\!]$$

and where:

$$\mathsf{FIX}\ H_{S,b} = \bigcup_{i \geq 0} H^i_{S,b}(\emptyset)$$

Use denotational semantics to prove that the statement:

$$\textbf{repeat } S \textbf{ until } b$$

is *equivalent* to the unfolding:

$$S;\ \textbf{if } b \textbf{ then skip else } (\textbf{repeat } S \textbf{ until } b)$$

Solution:

Our proof relies on the (unproved) assumption that $\mathsf{FIX}\ H_{S,b}$ is a fixed point of $H_{S,b}$:

$$\begin{array}{lll}
 & \mathcal{S}_{ds}[\![S;\ \textbf{if } b \textbf{ then skip else } (\textbf{repeat } S \textbf{ until } b)]\!] & \\
= & \mathcal{S}_{ds}[\![\textbf{if } b \textbf{ then skip else } (\textbf{repeat } S \textbf{ until } b)]\!] \circ \mathcal{S}_{ds}[\![S]\!] & \{\text{Def. } \mathcal{S}_{ds}\} \\
= & \textbf{cond}(\mathcal{B}[\![b]\!],\ \mathcal{S}_{ds}[\![\textbf{skip}]\!],\ \mathcal{S}_{ds}[\![\textbf{repeat } S \textbf{ until } b]\!]) \circ \mathcal{S}_{ds}[\![S]\!] & \{\text{Def. } \mathcal{S}_{ds}\} \\
= & \textbf{cond}(\mathcal{B}[\![b]\!],\ id,\ \mathsf{FIX}\ H_{S,b}) \circ \mathcal{S}_{ds}[\![S]\!] & \{\text{Def. } \mathcal{S}_{ds}\} \\
= & H_{S,b}\ (\mathsf{FIX}\ H_{S,b}) & \{\text{Def. } H_{S,b}\} \\
= & \mathsf{FIX}\ H_{S,b} & \{\text{Fixed-point property}\} \\
= & \mathcal{S}_{ds}[\![\textbf{repeat } S \textbf{ until } b]\!] & \{\text{Def. } \mathcal{S}_{ds}\}
\end{array}$$

2. For a Hoare triple $\{P\}\,S\,\{Q\}$, the pair (P, Q) is often called the *contract* of the program S. In fact, for many applications it is meaningful to separate a contract $C = (P, Q)$ from its possible implementations. We can then say that a particular implementation S *meets* its contract C, denoted $S \models_{par} C$, if and only if the corresponding Hoare triple is semantically valid, that is $\models_{par} \{P\}\,S\,\{Q\}$. [6p]

   Now, your task is to provide a *denotational semantics for contracts*. That is, define the denotation $\mathcal{S}_{ds}[\![C]\!]$ of a contract $C = (P, Q)$ as a single mathematical object (of what type?), in the spirit of how we defined the denotation of statements. Your definition should allow the relation $S \models_{par} C$ to be equivalently expressed as a simple relationship between the denotations of S and C. Show this relationship and motivate your reasoning.

   Solution:

   Recall that $(s, s') \in \mathcal{S}_{ds}[\![S]\!]$ whenever execution of statement S from state s terminates in state s'. Also recall that $\models_{par} \{P\}\,S\,\{Q\}$ iff for any state s such that $s \models P$, if execution of S from s terminates in s', then $s' \models Q$. This suggests that we can define the denotation $\mathcal{S}_{ds}[\![C]\!]$ of contracts $C = (P, Q)$ as a *binary relation on states*! More specifically, using the notiation $\|P\| = \{s \in \textbf{State} \mid s \models P\}$ for the set of all states satisfying assertion P, if we define:

$$\mathcal{S}_{ds}[\![C]\!] \stackrel{\mathsf{def}}{=} (\|P\| \times \|Q\|) \cup (\overline{\|P\|} \times \textbf{State})$$

   we obtain the following simple set-theoretic relationship:

$$S \models_{par} C \ \text{ iff } \ \mathcal{S}_{ds}[\![S]\!] \subseteq \mathcal{S}_{ds}[\![C]\!]$$

3 Level A

For grade B you need to have passed level C and obtained 3 (out of 8) points from this section. For grade A you need 6 points from this section.

Consider again the *denotational semantics* of **While** and its extension with **repeat** S **until** b as proposed in Problem C1 above.

1. Explain what the i-th approximant of $H_{S,b}$ captures. [2p]

   Solution:

   We have $H^i_{S,b}(\emptyset)(s) = s'$ iff exection of **repeat** S **until** b from state s terminates in state s' by executing the loop body S at most i times.

2. Compute iteratively the denotation of the statement **repeat** $x := x - 2$ **until** $x \leq 0$. That is, start by simplifying $H_{x:=x-2,\ x\leq 0}\ (g)$ as much as possible, by evaluating all occurrences of semantic functions. Then, compute the first few fixed-point approximants of $H_{x:=x-2,\ x\leq 0}$, and guess the i-th approximant. Finally, present the fixed point, which is also the denotation of the statement. Simplify all denotations as much as possible. [6p]

   Solution:

   Simplifying $H_{x:=x-2,\ x\leq 0}\ (g)$ results in (where for readability we omit the subscript to H):

$$H(g)(s) = \begin{cases} s[x \mapsto s(x) - 2] & \text{if } s(x) \leq 2 \\ g(s[x \mapsto s(x) - 2]) & \text{if } s(x) > 2 \end{cases}$$

   The first approximant, after simplification, becomes:

$$H^1(\emptyset)(s) = H(\emptyset)(s) = \begin{cases} s[x \mapsto s(x) - 2] & \text{if } s(x) \leq 2 \\ \textbf{undef} & \text{if } s(x) > 2 \end{cases}$$

   The second approximant, after simplification, becomes:

$$H^2(\emptyset)(s) = H(H^1(\emptyset))(s) = \begin{cases} s[x \mapsto s(x) - 2] & \text{if } s(x) \leq 2 \\ s[x \mapsto s(x) - 4] & \text{if } s(x) > 2 \text{ and } s(x) \leq 4 \\ \textbf{undef} & \text{if } s(x) > 4 \end{cases}$$

   The i-th approximant can be guessed as:

$$H^i(\emptyset)(s) = \begin{cases} s[x \mapsto s(x) - 2j] & \text{if } j > 0 \text{ is the smalles positive number} \\ & \text{such that } j \leq i \text{ and } s(x) - 2j \leq 0 \\ \textbf{undef} & \text{if no such } j \text{ exists, i.e., if } s(x) - 2i > 0 \end{cases}$$

   which can also be presented as:

$$H^i(\emptyset)(s) = \begin{cases} s[x \mapsto s(x) - 2] & \text{if } s(x) \leq 0 \\ s[x \mapsto -1] & \text{if } s(x) = 2k - 1 \text{ for some } 0 < k \leq i \\ s[x \mapsto 0] & \text{if } s(x) = 2k \text{ for some } 0 < k \leq i \\ \textbf{undef} & \text{if } s(x) > 2i \end{cases}$$

   Since $\mathcal{S}_{ds}[\![\textbf{repeat } S \textbf{ until } b]\!] = \mathsf{FIX}\ H_{S,b} = \bigcup_{i\geq 0} H^i_{x:=x-2,\ x\leq 0}(\emptyset)$, we finally obtain:

$$\mathcal{S}_{ds}[\![\textbf{repeat } S \textbf{ until } b]\!] = \begin{cases} s[x \mapsto s(x) - 2] & \text{if } s(x) \leq 0 \\ s[x \mapsto -1] & \text{if } s(x) = 2k - 1 \text{ for some } k > 0 \\ s[x \mapsto 0] & \text{if } s(x) = 2k \text{ for some } k > 0 \end{cases}$$

   Notice that the denotation is a total function, signifying that program execution terminates from *any* state.