DD2457 Program Semantics and Analysis

Examination Problems
with partial solutions
14 December 2009

Dilian Gurov
KTH CSC
tel: 08-790 8198

Give solutions in English or Swedish, each problem beginning on a new sheet. Write your name on all sheets. The maximal number of points is given for each problem. Up to two bonus points per section will be taken into account. The course book, the handouts, own notes taken in class, as well as reference material are admissible at the exam.

1 Level E

For passing level E you need 6 points from this section.

Consider the extension of **While** with non–deterministic choice S_1 **or** S_2 discussed in class and in the book.

1. In the structural operational semantics of this extended language, compute the *configuration graph* of the program $\boxed{5\text{p}}$

   $$\textbf{while } (0 \leq x) \wedge (x \leq 1) \textbf{ do } (x := x - 1 \textbf{ or } x := x + 1)$$

   from a state s such that $s(x) = 0$. Draw the graph as informatively as possible. Every transition of the graph should be justified by a derivation (but you can point out and omit derivations that are almost identical to an existing one).

   Solution: Routine.

2. Consider the following two types of termination properties for a (possibly non–deterministic) statement S from a state s: $\boxed{3\text{p}}$

   - *possible termination*, meaning that there is a terminating execution, and
   - *necessary termination*, meaning that all executions terminate.

   Formalise the two termination properties in both natural semantics and structural operational semantics. If you consider that it is not possible to formalise some case, give a justification why.

   Solution: In natural semantics we have:

   - *possible termination:* $\exists s' \in \textbf{State}.\ \langle S, s\rangle \to s'$
   - *necessary termination:* not expressible, since one cannot capture (absense of) on–going behaviour by relating initial to final configurations;

   while in structural operational semantics:

   - *possible termination:* $\exists s' \in \textbf{State}.\ \langle S, s\rangle \Rightarrow^+ s'$
   - *necessary termination:* more cumbersome to express, but still formalisable as "there is no infinite execution starting at $\langle S, s\rangle$":
     $\neg\, \exists S_0, S_1, \ldots \in \textbf{Stm},\ s_0, s_1, \ldots \in \textbf{State}.\ S_0 = S \wedge s_0 = s \wedge \forall i \geq 0.\ \langle S_i, s_i\rangle \Rightarrow \langle S_{i+1}, s_{i+1}\rangle$

2 Level C

For grade D you need to have passed level E and obtained 4 points from this section. For passing level C you need 7 points from this section.

1. Recall the extension of **While** with division a_1/a_2 and exception–handling **try** S_1 **catch** S_2 considered in the first laboratory assignment. To adapt the semantics of **While**, we added the special error value $\perp$ to the set of integer values, letting $Z_\perp \stackrel{\text{def}}{=} Z \cup \{\perp\}$, and re-defined the evaluation function $\mathcal{A} : \mathbf{AExp} \to (\mathbf{State} \to Z_\perp)$ to capture division by zero as the source of producing an exception, and propagation of the error value by all arithmetic operations. Similarly, we added $\perp$ to the set of truth values, letting $\mathbf{T}_\perp \stackrel{\text{def}}{=} \mathbf{T} \cup \{\perp\}$, and re-defined the evaluation function $\mathcal{B} : \mathbf{BExp} \to (\mathbf{State} \to \mathbf{T}_\perp)$ so that the error value propagates. Finally, to distinguish between normal and exceptional termination, we introduced the set of extended states $\mathbf{EState} \stackrel{\text{def}}{=} \mathbf{State} \times \{\top, \perp\}$, where an extended state $(s, \top)$ is normal and $(s, \perp)$ is exceptional. By abuse of notation, we decided to let s denote the normal state $(s, \top)$, and $\hat{s}$ denote the exceptional state $(s, \perp)$.

   (a) Adapt the direct style *denotational semantics* of statements to the extended language (assuming that mappings $\mathcal{A}$ and $\mathcal{B}$ are already adapted suitably, for instance as you have done in the laboratory assignment). Show only the changed or added defining clauses. 3p
   Hint: statement denotation is now of type $\mathcal{S}_{ds} : \mathbf{Stm} \to (\mathbf{EState} \hookrightarrow \mathbf{EState})$. You can have separate defining clauses for normal and exceptional states.

   Solution: For exceptional states we just need one (non–inductive) clause:

   $$\mathcal{S}_{ds}[\![S]\!]\,(\hat{s}) \stackrel{\text{def}}{=} \hat{s}$$

   For normal states, the rule for assignment becomes:

   $$\mathcal{S}_{ds}[\![x := a]\!]\,(s) \stackrel{\text{def}}{=} \begin{cases} s[x \mapsto \mathcal{A}[\![a]\!]\,(s)] & \text{if } \mathcal{A}[\![a]\!]\,(s) \in Z \\ \hat{s} & \text{if } \mathcal{A}[\![a]\!]\,(s) = \perp \end{cases}$$

   The remaining rules stay as they are, but the auxilliary function cond is redefined (still only for normal states) as follows:

   $$\mathsf{cond}(p, g_1, g_2)(s) \stackrel{\text{def}}{=} \begin{cases} g_1(s) & \text{if } p(s) = \mathbf{tt} \\ g_2(s) & \text{if } p(s) = \mathbf{ff} \\ \hat{s} & \text{if } p(s) = \perp \end{cases}$$

   However, the functional $F_{b,s}$ then needs an explicit clause for exceptional states:

   $$F_{b,s}(g)(\hat{s}) \stackrel{\text{def}}{=} \hat{s}$$

   Finally, we have to add a clause for exception handling:

   $$\mathcal{S}_{ds}[\![\mathbf{try}\ S_1\ \mathbf{catch}\ S_2]\!]\,(s) \stackrel{\text{def}}{=} \begin{cases} s' & \text{if } \mathcal{S}_{ds}[\![S_1]\!]\,(s) = s' \\ \mathcal{S}_{ds}[\![S_2]\!]\,(s') & \text{if } \mathcal{S}_{ds}[\![S_1]\!]\,(s) = \hat{s}' \\ \underline{\text{undef}} & \text{if } \mathcal{S}_{ds}[\![S_1]\!]\,(s) = \underline{\text{undef}} \end{cases}$$

   or more elegantly by using function composition, with the help of a suitably defined functional applied to the denotation of S_2.

(b) Use your denotational semantics to compute the denotation of the program: $\boxed{2p}$

$$x := 7; \textbf{try } x := x - 7; x := 7/x; x := x + 7 \textbf{ catch } x := x - 7$$

applied to an arbitrary normal initial state s.

Solution:

$$
\begin{array}{ll}
& \mathcal{S}_{ds}[\![x := 7; \textbf{try } x := x - 7; x := 7/x; x := x + 7 \textbf{ catch } x := x - 7]\!]\,(s) \\
= & \mathcal{S}_{ds}[\![\textbf{try } x := x - 7; x := 7/x; x := x + 7 \textbf{ catch } x := x - 7]\!]\,(\mathcal{S}_{ds}[\![x := 7]\!]\,(s)) \\
= & \mathcal{S}_{ds}[\![\textbf{try } x := x - 7; x := 7/x; x := x + 7 \textbf{ catch } x := x - 7]\!]\,(s[x \mapsto 7]) \\
& \text{Now, } \mathcal{S}_{ds}[\![x := x - 7; x := 7/x; x := x + 7]\!]\,(s[x \mapsto 7]) \\
& = \mathcal{S}_{ds}[\![x := x + 7]\!]\,(\mathcal{S}_{ds}[\![x := 7/x]\!]\,(\mathcal{S}_{ds}[\![x := x - 7]\!]\,(s[x \mapsto 7]))) \\
& = \mathcal{S}_{ds}[\![x := x + 7]\!]\,(\mathcal{S}_{ds}[\![x := 7/x]\!]\,(s[x \mapsto 0])) \\
& = \mathcal{S}_{ds}[\![x := x + 7]\!]\,(\hat{s}[x \mapsto 0]) \\
& = \hat{s}[x \mapsto 0] \\
& \text{and therefore} \\
= & \mathcal{S}_{ds}[\![x := x - 7]\!]\,(s[x \mapsto 0]) \\
= & s[x \mapsto -7]
\end{array}
$$

2. Consider again the extension of **While** with non–deterministic choice S_1 **or** S_2.

   (a) For a post–condition Q, express the weakest liberal pre–condition $wlp(S_1 \textbf{ or } S_2, Q)$ compositionally, that is in terms of the weakest liberal pre–conditions for S_1 and S_2. Justify your answer! Note: we are taking the intensional view to Hoare logic, so pre– and post–conditions are assertions. $\boxed{1p}$

   Solution: We have:

   $$wlp(S_1 \textbf{ or } S_2, Q) = wlp(S_1, Q) \wedge wlp(S_2, Q)$$

   because execution of S_1 **or** S_2 results in execution of either S_1 or S_2, but the post–condition Q must hold (upon termination) in *both* cases.

   (b) Guided by your answer, extend the verification condition generator discussed in class by adding a defining clause for $vcg\,[\![S_1 \textbf{ or } S_2]\!]\,(P, Q)$. $\boxed{1p}$

   Solution: We have:

   $$
   \begin{array}{ll}
   \multicolumn{2}{l}{vcg\,[\![S_1 \textbf{ or } S_2]\!]\,(P,Q) \stackrel{\text{def}}{=}} \\
   \textbf{let} & (P_1, Q_1) = vcg\,[\![S_1]\!]\,(P,Q) \\
   & (P_2, Q_2) = vcg\,[\![S_2]\!]\,(P,Q) \\
   \textbf{in} & (P_1 \wedge P_2, Q_1 \wedge Q_2)
   \end{array}
   $$

   (c) Verify the Hoare triple $\boxed{3p}$

   $$\{true\}\, \textbf{while } (0 \leq x) \wedge (x \leq 1) \textbf{ do } (x := x - 1 \textbf{ or } x := x + 1)\, \{x < 0 \vee x > 1\}$$

   by extracting a verification condition with your verification condition generator and justifying the verification condition.
   Note: you need first to annotate the while loop with a suitable loop invariant.

   Solution: The loop invariant has to be implied by the pre–condition *true* and hence we take the formula *true* as a loop invariant.

Then, we compute:

$$
\begin{array}{rl}
& vcg\,[\![\{true\}\,\mathbf{while}\ (0 \leq x) \wedge (x \leq 1)\ \mathbf{do}\ (x := x-1\ \mathbf{or}\ x := x+1)]\!]\,(x < 0 \vee x > 1, true) \\
= & \mathbf{let}\ (P, Q) = vcg\,[\![x := x-1\ \mathbf{or}\ x := x+1]\!]\,(true, true) \\
& \mathbf{in}\ (true, true \wedge Q \wedge (true \wedge (0 \leq x) \wedge (x \leq 1) \Rightarrow P)) \wedge (true \wedge \neg((0 \leq x) \wedge (x \leq 1)) \Rightarrow x < 0 \vee x > 1))) \\
= & \mathbf{let}\ (P, Q) = (true \wedge true, true \wedge true) \\
& \mathbf{in}\ (true, true \wedge Q \wedge (true \wedge (0 \leq x) \wedge (x \leq 1) \Rightarrow P)) \wedge (true \wedge \neg((0 \leq x) \wedge (x \leq 1)) \Rightarrow x < 0 \vee x > 1))) \\
= & (true, true \wedge true \wedge (true \wedge (0 \leq x) \wedge (x \leq 1) \Rightarrow true)) \wedge (true \wedge \neg((0 \leq x) \wedge (x \leq 1)) \Rightarrow x < 0 \vee x > 1)))
\end{array}
$$

and finally we obtain the verification condition:

$$
\begin{array}{rl}
& VCG(\{true\}\,\mathbf{while}\ (0 \leq x) \wedge (x \leq 1)\ \mathbf{do}\ (x := x-1\ \mathbf{or}\ x := x+1)\,\{x < 0 \vee x > 1\}) \\
= & (true \Rightarrow true) \wedge true \wedge true \wedge (true \wedge (0 \leq x) \wedge (x \leq 1) \Rightarrow true)) \wedge (true \wedge \neg((0 \leq x) \wedge (x \leq 1)) \Rightarrow x < 0 \vee x > 1))
\end{array}
$$

of which only the last conjunct is not trivial, but is still easy to justify, since the post–condition is logically equivalent to the negation of the loop guard.

---

3 Level A

For grade B you need to have passed level C and obtained 5 points from this section. For grade A you need 8 points from this section.

1. Show that statement **while** b **do** (**if** b **then** S_1 **else** S_2) is semantically equivalent to statement **while** b **do** S_1. Base your proof on a semantic style of your choice. 3p

   Solution: Proofs based on operational semantics require induction, since the while–rules unfold the loop. It is therefore conceptually simpler to give a proof in *denotational semantics*. We have the following equality on functionals:

$$
\begin{array}{rl}
& F_{b,\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2}(g)(s) \\
= & \begin{cases} g(\mathcal{S}_{ds}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\,(s)) & \text{if } \mathcal{B}[\![b]\!]\,(s) = \mathbf{tt} \\ s & \text{if } \mathcal{B}[\![b]\!]\,(s) = \mathbf{ff} \end{cases} \\
= & \begin{cases} g(\mathcal{S}_{ds}[\![S_1]\!]\,(s)) & \text{if } \mathcal{B}[\![b]\!]\,(s) = \mathbf{tt} \\ s & \text{if } \mathcal{B}[\![b]\!]\,(s) = \mathbf{ff} \end{cases} \\
= & F_{b,\ S_1}(g)(s)
\end{array}
$$

   and therefore:

$$
\begin{array}{rl}
& \mathcal{S}_{ds}[\![\mathbf{while}\ b\ \mathbf{do}\ (\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)]\!] \\
= & \mathsf{FIX}\ F_{b,\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2} \\
= & \mathsf{FIX}\ F_{b,\ S_1} \\
= & \mathcal{S}_{ds}[\![\mathbf{while}\ b\ \mathbf{do}\ S_1]\!]
\end{array}
$$

2. In the denotational semantics you developed above for **While** extended with division and exception handling, compute the denotational semantics of the statement

$$\textbf{while } 0 \leq y \textbf{ do } x := x/y; y := y - 1$$

That is:

(a) determine the functional F for this loop, simplifying as much as possible; $\boxed{3\text{p}}$

Solution:

$$
\begin{array}{rcl}
F(g)(\hat{s}) & = & \hat{s} \\
F(g)(s) & = & \begin{cases} g(\mathcal{S}_{ds}[\![x := x/y; y := y-1]\!](s)) & \text{if } \mathcal{B}[\![0 \leq y]\!](s) = \mathbf{tt} \\ s & \text{if } \mathcal{B}[\![0 \leq y]\!](s) = \mathbf{ff} \\ \hat{s} & \text{if } \mathcal{B}[\![0 \leq y]\!](s) = \bot \end{cases} \\
& = & \begin{cases} g(\mathcal{S}_{ds}[\![y := y-1]\!](\mathcal{S}_{ds}[\![x := x/y]\!](s))) & \text{if } s(y) \geq 0 \\ s & \text{if } s(y) < 0 \end{cases} \\
& = & \begin{cases} g(s[x \mapsto s(x)/s(y)][y \mapsto s(y)-1]) & \text{if } s(y) > 0 \\ g(\hat{s}) & \text{if } s(y) = 0 \\ s & \text{if } s(y) < 0 \end{cases}
\end{array}
$$

(b) compute the first two approximants in the iterative fixed–point construction and explain intuitively their meaning; $\boxed{3\text{p}}$

Solution: As a first approximant we obtain:

$$
\begin{array}{rcl}
F(\emptyset)(\hat{s}) & = & \hat{s} \\
F(\emptyset)(s) & = & \begin{cases} \underline{\text{undef}} & \text{if } s(y) \geq 0 \\ s & \text{if } s(y) < 0 \end{cases}
\end{array}
$$

indicating that the statement terminates from a state s without executing the loop body exactly when $s(y) < 0$, and then it terminates in the same state.
As a second approximant we obtain:

$$
\begin{array}{rcl}
F^2(\emptyset)(\hat{s}) & = & F(F(\emptyset))(\hat{s}) = \hat{s} \\
F^2(\emptyset)(s) & = & F(F(\emptyset))(s) \\
& = & \begin{cases} F(\emptyset)(s[x \mapsto s(x)/s(y)][y \mapsto s(y)-1]) & \text{if } s(y) > 0 \\ F(\emptyset)(\hat{s}) & \text{if } s(y) = 0 \\ s & \text{if } s(y) < 0 \end{cases} \\
& = & \begin{cases} \underline{\text{undef}} & \text{if } s(y) > 0 \wedge s(y) - 1 \geq 0 \\ F(\emptyset)(s[x \mapsto s(x)/s(y)][y \mapsto s(y)-1]) & \text{if } s(y) > 0 \wedge s(y) - 1 < 0 \\ \hat{s} & \text{if } s(y) = 0 \\ s & \text{if } s(y) < 0 \end{cases} \\
& = & \begin{cases} \underline{\text{undef}} & \text{if } s(y) > 0 \\ \hat{s} & \text{if } s(y) = 0 \\ s & \text{if } s(y) < 0 \end{cases}
\end{array}
$$

indicating that the statement terminates from a state s with executing the loop body at most once exactly when $s(y) \leq 0$, and then it terminates in the same state s if $s(y) < 0$ and in the respective exceptional state $\hat{s}$ if $s(y) = 0$.

(c) guess the i–th approximant and explain intuitively its meaning; $\boxed{2\text{p}}$

Solution:

$$
\begin{array}{rcl}
F^i(\emptyset)(\hat{s}) & = & \hat{s} \\
F^i(\emptyset)(s) & = & \left\{ \begin{array}{ll}
\underline{\text{undef}} & \text{if } s(y) \geq i-1 \\
\hat{s}[x \mapsto s(x)/s(y)/s(y)-1/\ldots/2][y \mapsto 0] & \text{if } 2 \leq s(y) < i-1 \\
\hat{s}[y \mapsto 0] & \text{if } 0 \leq s(y) \leq 1 \\
s & \text{if } s(y) < 0
\end{array} \right.
\end{array}
$$

indicating that the statement terminates from a state s with executing the loop body at most $i-1$ times exactly when $s(y) < i-1$, and then it terminates in the same state s if $s(y) < 0$ and otherwise in an exceptional state after a corresponding number of integer divisions to x and y being 0.

(d) present the denotation of the loop as the limit of the construction and explain intuitively its meaning. $\boxed{1\text{p}}$

Solution:

$$
\begin{array}{rcl}
(\mathsf{FIX}\ F)(\hat{s}) & = & \hat{s} \\
(\mathsf{FIX}\ F)(s) & = & (\bigcup_{i\geq 0} F^i(\emptyset))(s) \\
& = & \left\{ \begin{array}{ll}
\hat{s}[x \mapsto s(x)/s(y)/s(y)-1/\ldots/2][y \mapsto 0] & \text{if } 2 \leq s(y) \\
\hat{s}[y \mapsto 0] & \text{if } 0 \leq s(y) \leq 1 \\
s & \text{if } s(y) < 0
\end{array} \right.
\end{array}
$$

indicating that the statement, when executed from a state s, always terminates, and then it terminates in the same state s if $s(y) < 0$ and otherwise in an exceptional state after a corresponding number of integer divisions to x and y being 0.