1 Instructions

This assignment is to be completed by hand (or in LaTeX if you know how to use it). Submit your answers as a PDF file via Gradescope. If you don’t have easy access to a scanner, you can use the one in SEO 1120, the main CS office – the staff will be happy to help you. As always, please don’t hesitate to ask for help on Piazza (https://piazza.com/class/jkh8q52qrh06v).

2 Operational Semantics of IMP

Here are the operational semantics rules for a simple imperative programming language, using the “hybrid style” of big steps for expressions and small steps for commands.

\[
\begin{align*}
(n \text{ is a number}) & \quad \frac{(n, \sigma) \Downarrow n}{(n, \sigma) \Downarrow n} \\
(b \text{ is a boolean}) & \quad \frac{(b, \sigma) \Downarrow b}{(b, \sigma) \Downarrow b} \\
(\sigma(x) = v) & \quad \frac{(\sigma(x) = v)}{(x, \sigma) \Downarrow v} \\
(e_1, \sigma) \Downarrow v_1 & \quad (e_2, \sigma) \Downarrow v_2 \\
(e_1 + e_2, \sigma) \Downarrow v & \quad (v_1 + v_2 = v) \\
(e_1 \oplus e_2, \sigma) \Downarrow v & \quad \text{where } \oplus \text{ is an arithmetic or boolean operator} \\
(e, \sigma) \Downarrow \text{true} & \quad (e_1, \sigma) \Downarrow v \\
\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma \Downarrow v & \quad (e, \sigma) \Downarrow false \\
(e_1, \sigma) \Downarrow v & \quad (e_2, \sigma) \Downarrow v \\
\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma \Downarrow v & \quad (e, \sigma) \Downarrow false \\
(x := e, \sigma) \rightarrow \text{skip}(\sigma[x \mapsto v]) & \quad (c_1, \sigma) \rightarrow (c'_1, \sigma') \\
(c_1; c_2, \sigma) \rightarrow (c'_1; c_2, \sigma') & \quad (c_1; c_2, \sigma) \rightarrow (c'_1; c_2, \sigma') \\
\text{skip}; c_2, \sigma \rightarrow (c_2, \sigma) & \quad (c_1; c_2, \sigma) \rightarrow (c'_1; c_2, \sigma')
\end{align*}
\]

3 Problems

There are four problems in all. Each problem is on a separate page. Use as much space as you need for each problem. You can add extra pages if you need to.
1. (6 points) Using the rules above, construct a proof tree showing that \((x + (2 * y), \{x = 2, y = 3\}) \Downarrow 8\).
   In other words, show that \(x + (2 * y)\) evaluates to 8 in the state where \(x = 2\) and \(y = 3\).

   Let \(\sigma_0\) be \(\{x = 2, y = 3\}\).

   \[
   \begin{array}{c}
   (\sigma_0(x) = 2) \quad (\sigma_0(y) = 3) \\
   \hline
   (x, \sigma_0) \Downarrow 2 \quad (2, \sigma_0) \Downarrow 2 \quad (y, \sigma_0) \Downarrow 3 \\
   \hline
   (2 \ast y, \sigma_0) \Downarrow 6 \quad (2 \ast y, \sigma_0) \Downarrow 6 \\
   \hline
   (x + (2 \ast y), \sigma_0) \Downarrow 8 \quad (2 \ast 3 = 6) \quad (2 + 6 = 8)
   \end{array}
   \]
2. (3 points) Construct a proof tree showing that

\[(z := x + (2 \ast y); x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \{x = 2, y = 3\}) \rightarrow (\text{skip}; x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \{x = 2, y = 3, z = 8\})\]

You can write “P1” to stand for the proof tree from the previous problem.

Again, let \(\sigma_0\) be \(\{x = 2, y = 3\}\).

\[
\frac{\begin{array}{c}
\text{P1} \\
(x + (2 \ast y), \sigma_0) \Downarrow 8 \\
\hline
(z := x + (2 \ast y), \sigma_0) \rightarrow (\text{skip}, \{x = 2, y = 3, z = 8\}) \\
\hline
\end{array}}{(z := x + (2 \ast y); x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \sigma_0) \rightarrow (\text{skip}; x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \{x = 2, y = 3, z = 8\})}
\]
3. (7 points) Construct a proof tree for the next step that

\[(x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \{x = 2, y = 3, z = 8\})\]

takes.

Let \(\sigma_1\) be \(\{x = 2, y = 3, z = 8\}\).

\[
\begin{array}{c}
\frac{(\sigma_1(z) = 8)}{(z, \sigma_1) \Downarrow 8} & \frac{(7, \sigma_1) \Downarrow 7}{(8 = 7) = \text{false}} & (4, \sigma_1) \Downarrow 4 \\
\frac{(z = 7, \sigma_1) \Downarrow \text{false}}{(\text{if } z = 7 \text{ then } 3 \text{ else } 4, \sigma_1) \Downarrow 4} \\
\frac{(x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \sigma_1) \rightarrow (\text{skip}, \{x = 4, y = 3, z = 8\})}{(x := \text{if } z = 7 \text{ then } 3 \text{ else } 4, \sigma_1)}
\end{array}
\]
4. (9 points) Suppose we extended the language with a command “c1 andthen c2 if e” that behaves as follows:

• First, it executes c1 normally.
• If e is true in the resulting state, it then executes c2.
• Otherwise, it ignores c2 and is finished executing.

In other words, to execute c1 andthen c2 if e, first execute c1 normally, and then execute c2 only if e is true.

Give small-step semantic rules for c1 andthen c2 if e. Remember that a command becomes “skip” when it is finished executing. As a test case, if you’ve written your rules correctly, x := 3 andthen y := 4 if x = 3 should step to (skip, {x = 3, y = 4}) in three small steps. For full credit, do not translate it into an if-then-else command.

Hint: it is probably easiest to define the command using three separate rules.

There are a number of possible solutions to this, some of which take more or fewer steps than others. As long as they produce the right result, all of them are correct. Here is one possible solution, in the hybrid style:

\[
\begin{align*}
(c_1, \sigma) \to (c'_1, \sigma') \\
(c_1 \text{ andthen } c_2 \text{ if } e, \sigma) \to (c'_1 \text{ andthen } c_2 \text{ if } e, \sigma') \\
(e, \sigma) \Downarrow \text{true} \\
\text{skip andthen } c_2 \text{ if } e, \sigma) \to (c_2, \sigma) \\
(e, \sigma) \Downarrow \text{false} \\
\text{skip andthen } c_2 \text{ if } e, \sigma) \to (\text{skip}, \sigma)
\end{align*}
\]