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/jushbmjmi1l3yb).

2 Operational Semantics of Expressions and Commands

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.

\[
\begin{align*}
(n \text{ is a number}) &\quad \frac{}{(n, \sigma) \downarrow n} \\
(b \text{ is a boolean}) &\quad \frac{}{(b, \sigma) \downarrow b} \\
(\sigma(x) = v) &\quad \frac{}{(x, \sigma) \downarrow v} \\
(e_1, \sigma) \downarrow v_1 &\quad (e_2, \sigma) \downarrow v_2 & (v_1 \oplus v_2 = v) &\quad \frac{}{(e_1 \oplus e_2, \sigma) \downarrow v} \\
(e, \sigma) \downarrow \text{true} &\quad (e_1, \sigma) \downarrow v & (e, \sigma) \downarrow \text{false} &\quad (e_2, \sigma) \downarrow v \\
(x := e, \sigma) \rightarrow \text{(skip, } \sigma[x \mapsto v]) &\quad (c_1, \sigma) \rightarrow (c_1', \sigma') \\
(c_1 ; c_2, \sigma) \rightarrow (c_1', c_2', \sigma') &\quad (\text{skip}; c_2, \sigma) \rightarrow (c_2, \sigma)
\end{align*}
\]

where $\oplus$ is an arithmetic or boolean operator

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 \((x = y) ||\ true, \{x = 5, y = 4\}\) \(\Downarrow\) true. In other words, show that \((x = y) ||\ true\) evaluates to true in the state where \(x\) is 5 and \(y\) is 4. (Note that \(=\) is the binary comparison operator, not the assignment command.)

Let \(\sigma_0\) be \(\{x = 5, y = 4\}\).

\[
\begin{array}{c}
\frac{(\sigma_0(x) = 5)}{(x, \sigma_0) \Downarrow 5} & \frac{(\sigma_0(y) = 4)}{(y, \sigma_0) \Downarrow 4} & ((5 = 4) = \text{false}) \\
\hline
(x = y, \sigma_0) \Downarrow \text{false} & (\text{true}, \sigma_0) \Downarrow \text{true} & (\text{false || true = true}) \\
\hline
((x = y) ||\ true, \sigma_0) \Downarrow \text{true}
\end{array}
\]
2. (3 points) Construct a proof tree showing that

\[
(c := (x = y) || true; x := if c then 1 else 2, \{x = 5, y = 4\}) \rightarrow \\
(skip; x := if c then 1 else 2, \{x = 5, y = 4, c = true\})
\]

When you reach a piece of the proof tree that matches the answer to the previous problem, you can write “P1” to stand in for it.

Again, let \(\sigma_0\) be \(\{x = 5, y = 4\}\).

\[
\begin{array}{c}
P1 \\
((x = y) || true, \sigma_0) \Downarrow true \\
(c := (x = y) || true, \sigma_0) \rightarrow (skip, \{x = 5, y = 4, c = true\})
\end{array}
\]

\[
(c := (x = y) || true; x := if c then 1 else 2, \sigma_0) \rightarrow \\
(skip; x := if c then 1 else 2, \{x = 5, y = 4, c = true\})
\]
3. (7 points) Construct a proof tree for the next step that

\[(x := \text{if } c \text{ then } 1 \text{ else } 2, \{x = 5, y = 4, c = \text{true}\})\]

takes.

Let \(\sigma_1\) be \(\{x = 5, y = 4, c = \text{true}\}\).

\[
\begin{array}{c}
\frac{(\sigma_1(c) = \text{true})}{(c, \sigma_1) \downarrow \text{true}} \quad \frac{(1, \sigma_1) \downarrow 1}{(\text{if } c \text{ then } 1 \text{ else } 2, \sigma_1) \downarrow 1} \\
\hline
\frac{(x := \text{if } c \text{ then } 1 \text{ else } 2, \sigma_1) \rightarrow (\text{skip}, \{x = 1, y = 4, c = \text{true}\})}{(x := \text{if } c \text{ then } 1 \text{ else } 2, \sigma_1) \downarrow 1}
\end{array}
\]
4. (9 points) Suppose we extended the language with a command “\( x := e \text{ anytime in } c \)” that executes \( c \), but can pause at any time to set the value of \( x \) to \( e \). More precisely:

- \( x := e \text{ anytime in } c \) can always execute a step of \( c \).
- If \( e \) can evaluate to a value in the current state, then \( x := e \text{ anytime in } c \) can assign the value of \( e \) to \( x \), and then continue to execute the rest of \( c \).
- If \( c \) has finished executing, \( x := e \text{ anytime in } c \) can also finish executing.

Give small-step semantic rules for \( x := e \text{ anytime in } c \). You may use the hybrid style (big-step for expressions, small-step for commands). Remember that a command becomes “skip” when it is finished executing. As a test case, if you’ve written your rules correctly,\( x := y \text{ anytime in } (y := 3; z := x) \) should step to \((\text{skip}, \{x = 3, y = 3, z = 3\})\) in three small steps.

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

Here is one possible solution, in the hybrid style:

\[
\frac{(c, \sigma) \rightarrow (c', \sigma')}{(x := e \text{ anytime in } c, \sigma) \rightarrow (x := e \text{ anytime in } c', \sigma')}
\]

\[
\frac{(e, \sigma) \Downarrow v}{(x := e \text{ anytime in } c, \sigma) \rightarrow (c, \sigma[x \mapsto v])}
\]

\[
(x := e \text{ anytime in } \text{skip}, \sigma) \rightarrow (\text{skip}, \sigma)
\]