HW3 – Proof Trees and Operational Semantics
CS 476, Fall 2019
Due Sep. 23

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.

\[
\begin{align*}
(n &\text{ is a number}) & (n, \sigma) \Downarrow n \\
(b &\text{ is a boolean}) & (b, \sigma) \Downarrow b \\
(\sigma(x) = v) & (x, \sigma) \Downarrow v \\
(e_1, \sigma) \Downarrow v_1 & (e_2, \sigma) \Downarrow v_2 & (v_1 \oplus v_2 = v) & (e_1 \oplus e_2, \sigma) \Downarrow v \\
(e, \sigma) \Downarrow true & (e_1, \sigma) \Downarrow v & (e, \sigma) \Downarrow false & (e_2, \sigma) \Downarrow v \\
(\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma) \Downarrow v &
\end{align*}
\]

where \( \oplus \) is an arithmetic or boolean operator

\[
\begin{align*}
(x := e, \sigma) \rightarrow (\text{skip, } \sigma[x \mapsto v]) \\
(c_1, \sigma) \rightarrow (c'_1, \sigma') \\
(c_1; c_2, \sigma) \rightarrow (c'_1; c_2, \sigma') \\
(\text{skip}; c_2, \sigma) \rightarrow (c_2, \sigma)
\end{align*}
\]

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 \(((x = y) \mid\mid \text{true}, \{x = 5, y = 4\}) \Downarrow \text{true}\). In other words, show that \((x = y) \mid\mid \text{true}\) 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.)
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.
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.
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, without performing the assignment.

Give small-step semantic rules for \(x := e \text{ anytime in } c\). 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 (skip, \(\{x = 3, y = 3, z = 3\}\)) in three small steps.

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