Homework 5: Axiomatic semantics and Hoare-style verification

15-819O: Program Analysis
Claire Le Goues
clegoues@cs.cmu.edu

Due: Tuesday, March 29, 2016 (11:59 pm) *80 points total, 12 points extra credit*

Assignment Objectives:

- Demonstrate understanding of verification condition generation and use in verifying programs.
- Write new Hoare Rules/axiomatic semantics for a particular language.
- Reason about soundness/completeness of a system for axiomatic semantics.

To submit, turn in a PDF file electronically containing your responses via Blackboard, under Homework 5. Note that a .docx is not a .pdf. Name your file andrewid-hw5.pdf. Include your name at the top of the PDF. Proper submission is worth two points.

Question 1, VCGen, *(20 points).* Consider the following rules for VCGen. We saw the first two in class, the third is a new proposed rule for `let`:

$$\begin{array}{lcl} VC(S_1; S_2, B) & = & VC(S_1, VC(S_2, B)) \\ VC(x := e, B) & = & [e/x]B \\ VC(\texttt{let } x = e \texttt{ in } S, B) & = & [e/x]VC(S, B) \end{array}$$

The rule for `let` is incorrect. Explain why (English prose is fine; see next question for extra credit for a formal answer), and then give a correct rule for `let`.

Extra Credit Question, Let rule soundness, *(12 points).* Given $\{A\}\ S\ \{B\}$, we desire that $A \Rightarrow VC(c, B) \Rightarrow WP(c, B)$. We say that our VC rules are *sound* if $\vDash \{VC(S, B)\}\ S\ \{B\}$. Demonstrate the unsoundness of the buggy `let` rule above by giving/showing the following six things:

1. a statement S and
2. a post-condition B and
3. a state E, all such that
4. $E \vDash VC(S, B)$ and
5. $\langle S, E\rangle \Downarrow E'$ but
6. $E \nvDash B$

Question 2, Do-while, *(15 points).* Write a sound/complete Hoare rule for `do` S `while` b. The statement has the standard semantics (i.e., S is executed at least once, before b is tested). You do not need to formally prove soundness/completeness, but make sure the rule makes sense.

Question 3, Loop proof obligations, *(18 points).* Consider the following program:

```
{N > 0 }
i := 0;
{ i <= N }
while (i < N) do
    { i <= N }
     i := i + 1
    { i <= N }
{i=N}
```

Assuming the loop invariant $i \leq N$, write the proof obligations for the `while` loop. Don't solve/prove them, just write them out. The form of your answer should be three mathematical implications, one for each of the following proof obligations:

- Invariant is initially true:
- Invariant is preserved by the loop body:
- Invariant and exit condition imply postcondition:

Question 4, Soundness/Completeness, *(25 points).* Consider the following three Hoare rules:

$$\frac{\vdash \{X\}\ S\ \{b \Rightarrow X \wedge \neg b \Rightarrow Y\}}{\vdash \{b \Rightarrow X \wedge \neg b \Rightarrow Y\}\ \texttt{while}\ b\ \texttt{do}\ S\ \{Y\}}\ \text{rule1} \qquad \frac{\vdash \{X \wedge b\}\ S\ \{X\}}{\vdash \{X\}\ \texttt{while}\ b\ \texttt{do}\ S\ \{X\}}\ \text{rule2}$$

$$\frac{\vdash \{X\}\ S\ \{X\}}{\{X\}\ \texttt{while}\ b\ \texttt{d}\ S\ \{X \wedge \neg b\}}\ \text{rule3}$$

Recall that a system of axiomatic semantics is *sound* if everything we can prove is also true: if $\vdash \{A\}\ S\ \{B\}$ then $\vDash \{A\}\ S\ \{B\}$. A system of axiomatic semantics is *complete* if we can prove all true things: if $\vDash \{A\}\ S\ \{B\}$ then $\vdash \{A\}\ S\ \{B\}$. All three of the rules above are sound, but only one is complete.

Part (a): Identify the two incomplete rules.

Part (b): Choose one of the identified incomplete rules; indicate which one you have chosen. Then give/show example A, B, E, S, and E' such that $\langle S, E\rangle \Downarrow E'$ and both $E \vDash A$ and and $E' \vDash B$, but it is not possible given the rule to prove $\vdash \{A\}\ S\ \{B\}$.

(Educational) note: An incomplete system cannot prove all possible properties or handle all possible programs. Incompleteness in an axiomatic semantics or type system is typically not as dire as unsoundness. Many research results that claim to work for the C language, for example, are actually incomplete because they fail to address, say, setjmp/longjmp or bitfields. (Many of them are also unsound because they do not correctly model various language features like unsafe casts, pointer arithmetic, or integer overflow.