Exercise 4F-2. VCGen for Let [6 points]. In class we gave the following rules for the (backward) verification condition generation of assignment and let:

$$\begin{array}{ll} \mathrm{VC}(c_1; c_2, B) & = \mathrm{VC}(c_1, \mathrm{VC}(c_2, B)) \\ \mathrm{VC}(x := e, B) & = [e/x]\ B \\ \mathrm{VC}(\mathsf{let}\ x = e\ \mathsf{in}\ c, B) & = [e/x]\ \mathrm{VC}(c, B) \end{array}$$

That rule for let has a bug. Give a correct rule for let.

The current rule for `let` does not correctly handle scoping. That is, $[e/x]$ will persist even after c has finished its execution.

The correct rule for `let` is as follows:

$$\mathrm{VC}(\mathsf{let}\ x = e\ \mathsf{in}\ c, B) = VC(x_{\text{prior}} := x; x := e; c; x := x_{\text{prior}}, B)$$

Where x_{prior} is a fresh variable. If x is originally undefined at the command $x_{\text{prior}} := x$, then x_{prior} is left undefined and $x := x_{\text{prior}}$ restores x to being undefined.

Question assigned to the following page: 3

Exercise 4F-3. VCGen Mistakes [6 points]. Given $\{A\}c\{B\}$ we desire that $A \implies \text{VC}(c, B) \implies \text{WP}(c, B)$. We say that our VC rules are *sound* if $\models \{\text{VC}(c, B)\}\ c\ \{B\}$. Demonstrate the unsoundness of the buggy let rule by giving the following six things:

1. a command c and
2. a post-condition B and
3. a state σ such that
4. $\sigma \models \text{VC}(c, B)$ and
5. $\langle c, \sigma\rangle \Downarrow \sigma'$ but
6. $\sigma' \not\models B$.

Consider the following...

Let c be (let $x = 4$ in `skip`)
Let B be the post-condition $(x = 4)$
Let σ be the state with $[x := 0]$
We check whether $\sigma \models \text{VC}(c, B)$ holds...

$$\begin{aligned}&\text{VC}(c, B)\\ &= \text{VC}(\text{let } x = 4 \text{ in } \texttt{skip}, x = 4)\\ &= [4/x]\text{VC}(\texttt{skip}, x = 4)\\ &= [4/x](x = 4)\end{aligned}$$

Thus $\sigma \models \text{VC}(c, B)$ indeed holds as $[x := 0] \models [x/4](x = 4)$.

We now show $\langle c, \sigma\rangle \Downarrow \sigma'$, where $\sigma' = [x := 0]$, by using our large-step operational semantics rule for let as defined in Exercise 1F-4...

$$\frac{\langle x := e,\ \sigma\rangle \Downarrow \sigma[x := n] \quad \langle c,\ \sigma[x := n]\rangle \Downarrow \sigma'' \quad \sigma' = \text{recover}(x,\ \sigma'',\ \sigma)}{\langle \text{let } x = e \text{ in } c,\ \sigma\rangle \Downarrow \sigma'}\ \textsf{let}$$

$$\frac{\langle x := 4,\ \sigma\rangle \Downarrow \sigma[x := 4] \quad \langle \texttt{skip},\ \sigma[x := 4]\rangle \Downarrow \sigma'' \quad \sigma' = \text{recover}(x,\ \sigma'',\ \sigma)}{\langle \text{let } x = 4 \text{ in } \texttt{skip},\ \sigma\rangle \Downarrow \sigma'}\ \textsf{let}$$

By definition of "recover", $\sigma' = [x := 0]$

Thus, $\langle \text{let } x = 4 \text{ in } \texttt{skip},\ \sigma\rangle \Downarrow [x := 0]$

$\sigma' \not\models B$ does **not** hold, as $[x := 0] \not\models x = 4$
Thus, the given VC rule for let is *unsound*.

Question assigned to the following page: 4

Exercise 4F-4. Axiomatic Do-While [6 points]. Write a sound and complete Hoare rule for do c while b. This statement has the standard semantics (e.g., c is executed at least once, before b is tested).

We can define the hoare rule for do c while b as follows, using prior definitions we have given:

$$\frac{\{A\}\ c\ \{C\} \qquad \{C\}\ \textsf{while}\ b\ \textsf{do}\ c\ \{B \wedge \neg b\}}{\{A\}\ \textsf{do}\ c\ \textsf{while}\ b\ \{B \wedge \neg b\}}$$

This Hoare rule is both sound and complete as both $\{A\}\ c\ \{C\}$ and $\{C\}\ \textsf{while}\ b\ \textsf{do}\ c\ \{B\}$ have been shown to be sound and complete Hoare rules.