1 4F-1 Bookkeeping

- **0 pts** Correct

4F-2. VCGen for Let

To fix the issues with the provided `let` VC rule, we will need to make sure that changes to the variable defined by evaluating the `let` command do not persist in substitutions.

To address this, we select a fresh variable x', and define VC(let $x = e$ in c, B) = $[x'/x]$ VC($x' := e$; c, B). Essentially, we α-convert the body of the `let` command, such that the x within the command is distinct from any x outside of the command. In this way, we ensure that bindings made within the `let` do not incorrectly affect the outer state.

4F-3. VCGen Mistakes

The buggy `let` rule does not capture the scoping property of `let`. To demonstrate its incompleteness, we will define the following:

- c: let $x = 3$ in *skip*
- B: $x = 3$
- σ such that $\sigma(x) = 2$

By the buggy VC rule for `let`, we get that VC(c, B) = $[3/x]$ VC(*skip*, $x = 3$). By the VC rule for `skip`, we get that VC(*skip*, $x = 3$) is $x = 3$. Finally, $[3/x]$ $x = 3$ is just $3 = 3$, which is true. Thus, VC(let $x = 3$ in *skip*, $x = 3$) is true.

However, we know that `let` commands introduce new bindings for their inner variables. That is to say, assigning to the x defined by `let` does not modify any variable x already defined in σ. So, if we start with $\sigma(x) = 2$, and $\langle c, \sigma\rangle \Downarrow \sigma'$, in our resulting state $\sigma'(x) = 2$ as well. This violates our postcondition B, as $x \neq 3$, and thus VC(c, B) $\not\Longrightarrow$ B.

Therefore, the VC rule for `let` is unsound.

4F-4. Axiomatic Do-While

A do-while is just a while loop that does at least one iteration. In other words, `do` c `while` b is the same as c ; `while` b `do` c. So, we can define a Hoare rule for the construct as a sequence between these two subcommands:

$$\frac{\vdash \{A\}\ c\ \{B\} \qquad \vdash \{B\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{C\}}{\vdash \{A\}\ \texttt{do}\ c\ \texttt{while}\ b\ \{C\}}$$

2 4F-2 VCGen for Let

- **0 pts** Correct

4F-2. VCGen for Let

To fix the issues with the provided `let` VC rule, we will need to make sure that changes to the variable defined by evaluating the `let` command do not persist in substitutions.

To address this, we select a fresh variable x', and define VC(let $x = e$ in c, B) = $[x'/x]$ VC($x' := e$; c, B). Essentially, we α-convert the body of the `let` command, such that the x within the command is distinct from any x outside of the command. In this way, we ensure that bindings made within the `let` do not incorrectly affect the outer state.

4F-3. VCGen Mistakes

The buggy `let` rule does not capture the scoping property of `let`. To demonstrate its incompleteness, we will define the following:

- c: let $x = 3$ in *skip*
- B: $x = 3$
- σ such that $\sigma(x) = 2$

By the buggy VC rule for `let`, we get that VC(c, B) = $[3/x]$ VC(*skip*, $x = 3$). By the VC rule for `skip`, we get that VC(*skip*, $x = 3$) is $x = 3$. Finally, $[3/x]$ $x = 3$ is just $3 = 3$, which is true. Thus, VC(let $x = 3$ in *skip*, $x = 3$) is true.

However, we know that `let` commands introduce new bindings for their inner variables. That is to say, assigning to the x defined by `let` does not modify any variable x already defined in σ. So, if we start with $\sigma(x) = 2$, and $\langle c, \sigma\rangle \Downarrow \sigma'$, in our resulting state $\sigma'(x) = 2$ as well. This violates our postcondition B, as $x \neq 3$, and thus VC(c, B) $\not\Longrightarrow$ B.

Therefore, the VC rule for `let` is unsound.

4F-4. Axiomatic Do-While

A do-while is just a while loop that does at least one iteration. In other words, `do` c `while` b is the same as c ; `while` b `do` c. So, we can define a Hoare rule for the construct as a sequence between these two subcommands:

$$\frac{\vdash \{A\}\ c\ \{B\} \qquad \vdash \{B\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{C\}}{\vdash \{A\}\ \texttt{do}\ c\ \texttt{while}\ b\ \{C\}}$$

3 4F-3 VCGen Mistakes

- **0 pts** Correct

4F-2. VCGen for Let

To fix the issues with the provided `let` VC rule, we will need to make sure that changes to the variable defined by evaluating the `let` command do not persist in substitutions.

To address this, we select a fresh variable x', and define VC(let $x = e$ in c, B) = $[x'/x]$ VC($x' := e$; c, B). Essentially, we α-convert the body of the `let` command, such that the x within the command is distinct from any x outside of the command. In this way, we ensure that bindings made within the `let` do not incorrectly affect the outer state.

4F-3. VCGen Mistakes

The buggy `let` rule does not capture the scoping property of `let`. To demonstrate its incompleteness, we will define the following:

- c: let $x = 3$ in *skip*
- B: $x = 3$
- σ such that $\sigma(x) = 2$

By the buggy VC rule for `let`, we get that VC(c, B) = $[3/x]$ VC(*skip*, $x = 3$). By the VC rule for `skip`, we get that VC(*skip*, $x = 3$) is $x = 3$. Finally, $[3/x]$ $x = 3$ is just $3 = 3$, which is true. Thus, VC(let $x = 3$ in *skip*, $x = 3$) is true.

However, we know that `let` commands introduce new bindings for their inner variables. That is to say, assigning to the x defined by `let` does not modify any variable x already defined in σ. So, if we start with $\sigma(x) = 2$, and $\langle c, \sigma\rangle \Downarrow \sigma'$, in our resulting state $\sigma'(x) = 2$ as well. This violates our postcondition B, as $x \neq 3$, and thus VC(c, B) $\not\Longrightarrow$ B.

Therefore, the VC rule for `let` is unsound.

4F-4. Axiomatic Do-While

A do-while is just a while loop that does at least one iteration. In other words, `do` c `while` b is the same as c ; `while` b `do` c. So, we can define a Hoare rule for the construct as a sequence between these two subcommands:

$$\frac{\vdash \{A\}\ c\ \{B\} \qquad \vdash \{B\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{C\}}{\vdash \{A\}\ \texttt{do}\ c\ \texttt{while}\ b\ \{C\}}$$

4 4F-4 Axiomatic Do-While

- **0 pts** Correct