Exercise 5F-2. VCGen Do-While [8 points].

I chose option 1: Give the (backward) verification condition formula for the command doInv c while b with respect to a post-condition P. The invariant Inv is true before each evaluation of the predicate b. Your answer may not be defined in terms of VC(while...).

do-while loop:

do {c;} while (b);

with:

- Invariant: Inv(which holds before each evaluation of b)
- Postcondition: P (which should hold after the loop terminates).

Thus, to derive the backward verification condition formula ensuring its correctness,

1. **Verify the Postcondition after termination:**

   The loop exits when b becomes false.

   Since the invariant Inv is always true before evaluating b, we require:

$$(\neg b \wedge Inv) \Rightarrow P$$

2. **Ensuring invariant preservation in backward reasoning:**

   Since the loop executes at least once, we must ensure that if Inv is held before evaluating b, executing c maintains it.

   Thus, the execution of c must re establish Inv whenever b was true:

$$(Inv \wedge b) \Rightarrow \mathrm{wp}(c, Inv)$$

where wp(c, Inv) represents the weakest precondition ensuring that Inv holds after executing c.

Thus, the final backward Verification condition formula is as follows:

$$\Big((\neg b \wedge Inv) \Rightarrow P\Big) \wedge \Big((Inv \wedge b) \Rightarrow \mathrm{wp}(c, Inv)\Big)$$

Excercise 5F-3. VCGen Mistakes [20 points].

Here, to demonstrate the unsoundness of the buggy let rule, we construct an example as follows:

From the given set of rules, Lannister and Stark seems incomplete for the following reasons:

1. **Targaryen Rule:**

   $$\frac{\vdash \{X\}\ c\ \{X\}}{\vdash \{X\}\ \text{while}\ b\ \text{do}\ c\ \{X \wedge \neg b\}}$$

   This rule assumes X remains unchanged after execution, which is too restrictive. It does not allow proving properties about values before execution begins.

2. **Stark Rule:**

   $$\frac{\vdash \{X \wedge b\}\ c\ \{X\}}{\vdash \{X\}\ \text{while}\ b\ \text{do}\ c\ \{X\}}$$

   This rule is incomplete because it does not allow proving properties about termination or guarantees on final states when b becomes false.

Thus, let us first consider **Stark:**

Rule Name: Stark

A: X (precondition)

B: X (postcondition)

σ (initial state): A state satisfying X

σ′ (final state): A state after execution where X may not necessarily hold

command c: A command that modifies the state such that X does not hold after some executions

(c,σ)⇓σ′: c executes from σ to σ′

Aσ⊨A: σ satisfies X

$\sigma' \not\models B$: σ′ does not satisfy X

It is not possible to prove ⊢{A} c {B} because the rule does not ensure that X remains true after the loop executes.

Thus, let us secondly consider **Targaryen:**

Rule Name: Targaryen

A: X (the precondition)

B: $X \land \neg bX$ (the postcondition)

σ (initial state): A state satisfying X

σ' (final state): A state where b is false, but X may not hold anymore

Command c: A command that alters the state in a way that X does not necessarily remain true

$(c,\sigma) \Downarrow \sigma'$: c executes from σ to σ'

$\sigma \vDash A \sigma \vDash A$: σ satisfies X

$\sigma' \not\vDash B$: σ' does not satisfy $X \land \neg b$

It is not possible to prove $\vdash \{A\}\ c\ \{B\}$ because the rule assumes X remains invariant, which is too restrictive.