EXERCISE 5F-2: VCGEN FOR LET

We choose the first option:

`doInv c while b` with postcondition `P` and invariant `Inv` true before each evaluation of `b`.

We need to define the backward verification condition `VC(doInv c while b, P)` without referring to `VC(while...)`.

The goal is to ensure:

1. The first execution of `c` establishes the invariant.
2. The invariant is preserved across loop iterations when `b` holds.
3. The postcondition `P` holds when the loop exits (`¬b`).

Hence the formula is:

$$
\begin{aligned}
&VC(doInv\ c\ while\ b, P)\ = \\
&\quad VC(c, Inv) \\
&\wedge\ (Inv \wedge b \Rightarrow VC(c, Inv)) \\
&\wedge\ (Inv \wedge \neg b \Rightarrow P)
\end{aligned}
$$

Therefore:

- `VC(c, Inv)`: the first run of `c` must establish the invariant `Inv`
- `Inv ∧ b ⇒ VC(c, Inv)`: if the loop continues, executing `c` must reestablish `Inv`
- `Inv ∧ ¬b ⇒ P`: when the loop exits, `P` must follow from `Inv` and `¬b`

This captures the backward VC for `doInv c while b`, ensuring total correctness without relying on `VC(while...)`.

Peer Review ID: 320087237 — enter this when you fill out your peer evaluation via gradescope

Question assigned to the following page: 3

EXERCISE 5F-3: VCGEN MISTAKES

We choose the following two incomplete rules to analyze:

Rule 1: `stark`

Rule Definition:

$$\frac{\{X \wedge b\}\, c\, \{X\}}{\{X\}\ \mathit{while}\ b\ \mathit{do}\ c\ \{X\}}$$

Counterexample:

1. **Rule name**: stark
2. **A (Precondition)**: `true`
3. **B (Postcondition)**: `x > 10`
4. **σ (initial state)**: `x = 5`
5. **σ′ (final state)**: `x = 11`
6. **c**: `while x ≤ 10 do x := x + 1`
7. **Execution**: `x` increases from 5 to 11, so ⟨`c, σ`⟩ ⇓ `σ′`
8. **σ ⊨ A**: `true` holds
9. **σ′ ⊨ B**: `x = 11 ⇒ x > 10` ✓
10. **But cannot derive** `{A} c {B}` using `stark`:

Try `X = true`. Then:

$$\{X \wedge b\}\, c\, \{X\}\ \mathit{becomes}\ \{x \leq 10\}\, x := x + 1\, \{\mathit{true}\}\ —\ \mathit{valid}$$

But rule only lets us derive `{true} while b do c {true}`. This does not imply `x > 10`, which is our target postcondition. No stronger `X` is preserved by `x := x + 1`. So the rule cannot derive the true postcondition. Incomplete.

Peer Review ID: 320087237 — enter this when you fill out your peer evaluation via gradescope

Question assigned to the following page: 3

Rule 2: `targaryen`

Rule Definition:

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

Counterexample:

1. **Rule name**: targaryen
2. **A (Precondition)**: `true`
3. **B (Postcondition)**: `x > 10`
4. **σ (initial state)**: `x = 5`
5. **σ′ (final state)**: `x = 11`
6. **c**: `while x ≤ 10 do x := x + 1`
7. **Execution**: loop runs until `x = 11`, so `⟨c, σ⟩ ⇓ σ′`
8. **σ ⊨ A**: `true` holds
9. **σ′ ⊨ B**: `x = 11 ⇒ x > 10` ✓
10. **But cannot derive** `{A} c {B}` using `targaryen`:

Try `X = true`. Then:

$$\{X\}\ x := x + 1\ \{X\}\ \mathit{becomes}\ \{\mathit{true}\}\ x := x + 1\ \{\mathit{true}\}\ —\ \mathit{valid}$$

So we get: `{true} while b do c {true ∧ ¬(x ≤ 10)} ⇒ {x > 10}`.
This works here, but only because the postcondition happens to match `X ∧ ¬b`.

Now modify the goal to `B = x = 11`. Still:

- `{true} x := x + 1 {true}` holds
- But `{true} while x ≤ 10 do x := x + 1 {x = 11}` cannot be proven

No `X` such that `{X} x := x + 1 {X}` holds and `X ∧ ¬b ⇒ x = 11`. So rule is incomplete: true postcondition, not provable.

Peer Review ID: 320087237 — enter this when you fill out your peer evaluation via gradescope

Question assigned to the following page: 3

Conclusion: both `stark` and `targaryen` fail to derive `{A} c {B}` in cases where the execution does produce `B`, confirming incompleteness by example.