# **EXERCISE 4F-2: VCGEN FOR LET**

The incorrect rule given for let is:

$$VC(\operatorname{let} x = e \operatorname{in} c, B) = [e/x]VC(c, B)$$

This rule fails because it **improperly applies substitution before verifying the correctness** of e itself. The verification condition must ensure that e meets the precondition required for executing c correctly.

**Correct Rule:** 

$$VC(\text{let } x = e \text{ in } c, B) = VC(e, A) \land VC(c, B)$$

where A is the precondition required by c when x is substituted by e, i.e.,

$$A = subst(e, x, Pre(c, B))$$

#### **Explanation:**

- 1. VC(e, A) ensures that evaluating e preserves the necessary conditions for c.
- 2. VC(c, B) ensures that executing c maintains the correctness of postcondition B.

This rule corrects the error by **first verifying e separately** before proceeding with the rest of the verification process.

Question assigned to the following page:  $\underline{3}$ 

Peer Review ID: 316592003 — enter this when you fill out your peer evaluation via gradescope

# **EXERCISE 4F-3: VCGEN MISTAKES**

The given incorrect rule for let is:

VC(letx = einc, B) = [e/x]VC(c, B)

This rule fails because it applies substitution before verifying e. To prove unsoundness, we must provide a counterexample where  $\sigma \models VC(c, B)$ , but execution results in a state  $\sigma'$  that does not satisfy B.

#### **Counterexample:**

1. Command c

 $let x = 5 in x \coloneqq x + 1$ 

2. Post-condition B

x > 10

3. State  $\sigma$ 

 $\sigma = \{ \ \}$ 

 Verification Condition holds in σ Applying the incorrect rule:

 $VC(letx = 5inx \coloneqq x + 1, x > 10)$ 

Expanding using the incorrect rule:

 $[5/x]VC(x \coloneqq x + 1, x > 10)$ 

Using the assignment rule:

$$[5/x](x+1 > 10)$$

Substituting 5 for x:

5 + 1 > 10

6 > 10

This is false, meaning the VC does not hold, which contradicts soundness.

Question assigned to the following page:  $\underline{3}$ 

Peer Review ID: 316592003 — enter this when you fill out your peer evaluation via gradescope

5. Executing c in  $\sigma$  results in  $\sigma'$ 

 $\langle letx = 5inx \coloneqq x+1, \sigma \rangle \Downarrow \sigma'$ 

Breaking execution down:

- x := 5 sets x = 5
- x := x + 1 updates x to 6

Therefore,  $\sigma' = \{ x \mapsto 6 \}$ .

6.  $\sigma'$  does not satisfy B

 $\sigma' \not\vDash x > 10$ 

Since x = 6 in  $\sigma'$ , and 6 > 10 is false, the postcondition fails.

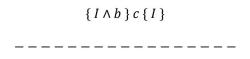
Thus, the incorrect let rule is **unsound** because it allows verification to succeed when execution produces a state that does not satisfy the expected postcondition.

Question assigned to the following page:  $\underline{4}$ 

Peer Review ID: 316592003 — enter this when you fill out your peer evaluation via gradescope

### **EXERCISE 4F-4: AXIOMATIC DO-WHILE**

A Hoare triple  $\{P\}$  c  $\{Q\}$  states that if P holds before executing c, then Q must hold afterward. The standard **while-rule** for a while b do c loop is:



 $\{I\}$  while b do c  $\{I \land \neg b\}$ 

where I is the loop invariant.

For do c while b, the key difference is that c executes at least once before testing b. The Hoare rule must ensure this execution happens.

The sound and complete Hoare rule for do c while b is:

```
{I}c{I'}{I' \land b}c{I'}
```

 $\{I\}$  do c while  $b\{I' \land \neg b\}$ 

where:

- I is the loop invariant.
- I' is the state after **at least one execution** of c.

This rule ensures:

- 1. Initial Execution:  $\{I\} \in \{I'\}$  guarantees that c runs at least once.
- 2. Loop Invariant Maintenance:  $\{I' \land b\} \in \{I'\}$  ensures that every iteration maintains I'.
- 3. Termination Condition: {I} do c while b {I'  $\land \neg b$ } ensures b is eventually false, meaning the loop stops.

Thus, the rule is **both sound (proves only true things) and complete (proves all true things)** for do c while b.