Axiomatic Semantics of Loops

1 VC For Let

First, we unwind `let`, where `t` is a fresh variable:

$$\text{let x = e in c} \quad \equiv \quad \text{t := x; x := e; c; x := t}$$

Thus we compute the VC:

$$\text{VC(let x = e in C}, B) \quad \equiv \quad \text{VC(t := x; x := e; c; x:= t}, B)$$

So the result is:

$$\begin{array}{ll} \text{VC(t := x; x := e; c; x := t}, B) & = \\ \text{VC(t := x, VC(x := e; c; x := t}, B)) & = \\ \text{[x/t]VC(x := e; c; x := t}, B) & = \\ \text{[x/t]VC(x := e, VC(c; x := t}, B)) & = \\ \text{[x/t] [e/x]VC(c; x := t}, B) & = \\ \text{[x/t] [e/x]VC(c, VC(x := t}, B)) & = \\ \text{[x/t] [e/x]VC(c,[t/x]}B) & \end{array}$$

An alternative formulation is to apply substitution to the command:

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

2 Unsound Let

The basic problem with the buggy let rule is that it does not restore the old value.

1. $c =$ let x = 1 in skip
2. $B =$ x = 1
3. $\sigma(x) = 0$
4. $\sigma \models VC(c, b)$, since $VC(c, b)$ is $1 = 1$
5. $\langle c, \sigma \rangle \Downarrow \sigma'$, where $\sigma'(x) = 0$, because the original value is restored after a let
6. $\sigma' \not\models B$ because $\sigma' \not\models$ x=1 because $\sigma'(x) = 0$.

3 Hoare Rule

First, we unwind `do-while`:

$$\text{do c while b} \quad \equiv \quad \text{c ; while b do c}$$

We will obtain our final Hoare rule by substituting in the appropriate rules:

$$\frac{\vdash \{A\}\text{c}\{B\} \quad \vdash \{B\}\text{while b do c}\{C\}}{\vdash \{A\}\text{do c while b}\{C\}}$$

That answer is officially good enough. You can view it as using the "alternate" `while` rule given on the *Alternate Hoare Rules* slide (near page 16) of the *Introduction To Axiomatic Semantics* lecture. We could also rephrase it using the more common `while` rule:

$$\frac{\vdash \{A\}\text{c}\{B\} \quad \vdash \{B\}\text{while b do c}\{B \wedge \neg b\}}{\vdash \{A\}\text{do c while b}\{B \wedge \neg b\}}$$

Which is then equivalent to:

$$\frac{\vdash \{A\}\text{c}\{B\} \quad \vdash \{B \wedge b\}\text{c}\{B\}}{\vdash \{A\}\text{do c while b}\{B \wedge \neg b\}}$$

4 Backwards VC

First, we unwind `do-while`:

$$\text{do}_{Inv1}\ \text{c while b} \quad \equiv \quad \text{assert}(Inv1);\ \text{c ;}\ \text{while}_{Inv2}\ \text{b do c}$$

The assertion comes from the problem description that $Inv1$ is true before and after c is executed. We use $Inv2$ to refer to the loop invariant of the while loop; it is typically the same as $Inv1$, but may potentially be stronger (i.e., it may incorporate information from the first execution of c). Note that $Inv2$ must imply $Inv1$ since $Inv1$ must also be true on every iteration of the while loop. So the result is:

$$\begin{array}{ll}
\text{VC}(\text{assert}(Inv1 \wedge Inv2 \Rightarrow Inv1);\ \text{c;}\ \text{while}_{Inv2}\ \text{b do c}, B) & = \\
\text{VC}(\text{assert}(Inv1 \wedge Inv2 \Rightarrow Inv1), \text{VC}(\text{c;}\ \text{while}_{Inv2}\ \text{b do c}, B)) & = \\
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \text{VC}(\text{c;}\ \text{while}_{Inv2}\ \text{b do c}, B) & = \\
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \text{VC}(\text{c}, \text{VC}(\text{while}_{Inv2}\ \text{b do c}, B)) & = \\
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \text{VC}(\text{c}, Inv2 \wedge (\forall x_1 \ldots x_n.\ Inv2 \implies (b \implies VC(c, Inv2)) \wedge \neg b \implies B)) &
\end{array}$$

where $x_1 \ldots x_n$ are the variables modified in c.

4.1 Common Mistake

The fact that the VC encodes the first execution of the command c is critical:

$$\begin{array}{ll}
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \underline{\text{VC}(\text{c}, \text{VC}(\text{while}_{Inv2}\ \text{b do c}, B))} & = \\
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \underline{\text{VC}(\text{c}, Inv2 \wedge (\forall x_1 \ldots x_n.\ Inv2 \implies (b \implies VC(c, Inv2)) \wedge \neg b \implies B))} &
\end{array}$$

One common mistake was to use something like the while rule from class:

$$Inv \wedge (\forall x_1 \ldots x_n.\ Inv \implies (b \implies VC(c, Inv)) \wedge \neg b \implies B))$$

Consider the program "do x := 1 while false". The program is basically an assignment statement dressed up as a loop. We should be able to compute the VC of it with respect to the post-condition x=1. We expect that VC to be equivalent to "true". Unfortunately, with the mistaken rule, we get:

$$\begin{array}{ll}
Inv \wedge (\forall x.\ Inv \implies (false \implies VC(x := 1, Inv)) \wedge true \implies x = 1)) & = \\
Inv \wedge (\forall x.\ Inv \implies true \implies x = 1)) & = \\
Inv \wedge (\forall x.\ Inv \implies x = 1)) &
\end{array}$$

There is no value of Inv for which this works. If we take Inv to be $x = 1$ we satisfy the right conjunct but cannot sastify the left. If we take Inv to be true, we satisfy the left but cannot satisfy the right.

If we do not use the mistaken rule but instead use the correct rule above, we get the following VC:

$$\begin{array}{ll}
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \underline{\text{VC}(\text{x:=1}, Inv2 \wedge (\forall x.\ Inv \implies (false \implies VC(c, Inv)) \wedge true \implies x = 1))} & = \\
Inv1 \wedge Inv2 \Rightarrow Inv1 \wedge \underline{\text{VC}(\text{x:=1}, Inv2 \wedge (\forall x.\ Inv2 \implies x = 1))} & =
\end{array}$$

We can take $Inv1$ to be true and $Inv2$, the inner loop invariant, to be $x = 1$:

$$\begin{array}{rl}
true \wedge x = 1 \Rightarrow true \wedge \underline{\text{VC}(\text{x:=1}, x = 1 \wedge (\forall x.\ x = 1 \implies x = 1))} & = \\
\underline{\text{VC}(\text{x:=1}, x = 1 \wedge (\forall x.\ x = 1 \implies x = 1))} & = \\
\underline{\text{VC}(\text{x:=1}, x = 1 \wedge (\forall x.\ true))} & = \\
\underline{\text{VC}(\text{x:=1}, x = 1)} & = \\
\underline{[1/\text{x}]x = 1} & = \\
1 = 1 & = \\
true &
\end{array}$$

5 The Man They Call Jayne

The Jayne rule is incomplete because it does not allow you to assume $\neg b$ after the loop terminates.

1. jayne
2. A = true

3. $B = \text{x} = 1$
4. $\sigma(x) = 0$
5. $\sigma'(x) = 1$
6. $c = \text{while x} \neq 1 \text{ do x} := 1$
7. $\langle c, \sigma \rangle \Downarrow \sigma'$ — not shown, but easily verified
8. $\sigma \models A$ because $\sigma \models \text{true}$
9. $\sigma' \models B$ because $\sigma \models \text{x} = 1$ because $\sigma'(x) = 1$.
10. it is not possible to prove {A} while x ≠ 1 do x = 1 {B} using the jayne rule. Assume, for the purpose of contradiction, that it is. Then we must have a derivation D that does so:

$$D :: \quad \vdash \{\text{true}\} \text{ while x} \neq 1 \text{ do x} := 1 \{\text{x=1}\}$$

By inversion, the last rule used in D must have been jayne or the rule of consequence. We proceed by cases.

(a) The last rule used in D was jayne. We reach an immediate contradiction, since jayne requires the pre- and post-conditions to be identical, but true is not identical to x=1.

(b) The last rule used in D was the rule of consequence. If so, by inversion again the derivation D must look like this:

$$\frac{\vdash \text{true} \Rightarrow P \quad D_2 :: \quad \vdash \{P\}\text{while x} \neq 1 \text{ do x} := 1\{P\} \quad \vdash P \Rightarrow \{x = 1\}}{\vdash \{\text{true}\} \text{ while x} \neq 1 \text{ do x} := 1 \{\text{x=1}\}}$$

where D_2 is an instance of the jayne rule. However, there is no P such that $\vdash \text{true} \Rightarrow P \Rightarrow x = 1$, so we reach a contradiction.

Since both cases led to a contradiction, we cannot have any such derivation D, so we cannot prove {A} while x ≠ 1 do x := 1 {B} using the jayne rule. You can also show the jayne rule to incomplete using the "counting to six" loop from class.

The River rule is incomplete because it does not allow you to assume b inside the loop body. We'll use the "count to six" loop as a counterexample.

1. river
2. $A = x \leq 6$
3. $B = x = 6$
4. $\sigma(x) = 0$
5. $\sigma'(x) = 6$
6. $c = \text{while x} < 6 \text{ do x} := \text{x} + 1$
7. $\langle c, \sigma \rangle \Downarrow \sigma'$ — not shown, but easily verified (and discussed in class)
8. $\sigma \models A$ because $\sigma \models x \leq 6$ because $\sigma(x) = 0 \leq 6$
9. $\sigma' \models B$ because $\sigma' \models x = 6$ because $\sigma'(x) = 6$
10. it is not possible to prove {A} while x < 6 do x := x+1 {B} using the river rule. Assume, for the purposes of contradiction, that is it. Then we must have a derivation D that does so:

$$D :: \quad \vdash \{x \leq 6\} \text{ while x} < 6 \text{ do x} := \text{x+1} \{\text{x=6}\}$$

By inversion, the last rule used in D must have been river or the rule of consequence. We proceed by cases.

(a) The last rule used in D was river. We reach an immediate contradiction, since river requires the post condition be textually equal to the precondition $\wedge \neg x < 6$.

(b) The last rule used in D was the rule of consequence. If so, by inversion again the derivation D must look like this:

$$\frac{D_1 :: \ \vdash x \leq 6 \Rightarrow P \quad D_2 :: \ \dfrac{D_4 :: \ \vdash \{P\}\text{x := x+1}\{P\}}{\vdash \{P\}\text{while x < 6 do x := x+1}\{P \wedge \neg x < 6\}} \quad D_3 :: \ \vdash P \wedge \neg x < 6 \Rightarrow x = 6}{\vdash \{x \leq 6\} \text{ while x < 6 do x := x+1 } \{\text{x=6}\}}$$

where D_2 is an instance of the river rule. From D_3 and D_1, we know that P must be $x \leq 6$. When then reach a contradiction, because $D_4 :: \ \vdash \{x \leq 6\}x := x + 1\{x \leq 6\}$ cannot exist by soundness (consider $x = 6$ going in to the assignment).

Since both cases led to a contradiction, we cannot have any such derivation D, so we cannot prove $\{x \leq 6\}$ while x < 6 do x := x+1 {x=6} using the river rule.

5.1 Common Mistakes

A common mistake was to choose the mal rule. The mal rule is not incomplete; it is the while rule from class composed with the rule of consequence. We will show its derivation in gory detail below. Recall these rules from class:

$$\frac{\vdash \{A \wedge b\}\text{c}\{A\}}{\vdash \{A\}\text{while b do c}\{A \wedge \neg b\}} \ \textit{while} \qquad \frac{\vdash P' \Rightarrow P \quad \vdash \{P\}\text{c}\{Q\} \quad \vdash Q \Rightarrow Q'}{\vdash \{P'\}\text{c}\{Q'\}} \ \textit{consequence}$$

We will start with consequence:

$$\frac{\vdash P' \Rightarrow P \quad \vdash \{P\}\text{c}\{Q\} \quad \vdash Q \Rightarrow Q'}{\vdash \{P'\}\text{c}\{Q'\}} \ \textit{step}1$$

And choose $P' = (b \Rightarrow X \wedge \neg b \Rightarrow Y)$:

$$\frac{\vdash (b \Rightarrow X \wedge \neg b \Rightarrow Y) \Rightarrow P \quad \vdash \{P\}\text{c}\{Q\} \quad \vdash Q \Rightarrow Q'}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{c}\{Q'\}} \ \textit{step}2$$

We choose c to be "while b do c":

$$\frac{\vdash (b \Rightarrow X \wedge \neg b \Rightarrow Y) \Rightarrow P \quad \vdash \{P\}\text{while b do c}\{Q\} \quad \vdash Q \Rightarrow Q'}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{while b do c}\{Q'\}} \ \textit{step}3$$

We choose $P = A$ and $Q = A \wedge \neg b$:

$$\frac{\vdash (b \Rightarrow X \wedge \neg b \Rightarrow Y) \Rightarrow A \quad \vdash \{A\}\text{while b do c}\{A \wedge \neg b\} \quad \vdash A \wedge \neg b \Rightarrow Q'}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{while b do c}\{Q'\}} \ \textit{step}4$$

Inlining the while rule from above, we obtain:

$$\frac{\vdash (b \Rightarrow X \wedge \neg b \Rightarrow Y) \Rightarrow A \quad \vdash \{A \wedge b\}\text{c}\{A\} \quad \vdash A \wedge \neg b \Rightarrow Q'}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{while b do c}\{Q'\}} \ \textit{step}5$$

Choose $A = (b \Rightarrow X \wedge \neg b \Rightarrow Y)$, and eliminate now-trivial left hypothesis, while simplifying the middle hypothesis:

$$\frac{\vdash \{X\}\text{c}\{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\} \quad \vdash (b \Rightarrow X \wedge \neg b \Rightarrow Y) \wedge \neg b \Rightarrow Q'}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{while b do c}\{Q'\}} \ \textit{step}6$$

Choose $Q' = Y$ and simplify away the now-trivial right hypothesis:

$$\frac{\vdash \{X\}\text{c}\{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}}{\vdash \{(b \Rightarrow X \wedge \neg b \Rightarrow Y)\}\text{while b do c}\{Y\}} \ \textit{step}7$$

Note that we've arrived at the mal rule just by substituting values in to consequence and while rules. So the mal rule has to be equivalent to those rules. Since they are not incomplete, the mal rule is not incomplete.