Axiomatic Semantics of Loops

1 VC For Let

First, we unwind \texttt{let}, where \( t \) is a fresh variable:

\[
\text{let } x = e \text{ in } c \equiv t := x; x := e; c; x := t
\]

Thus we compute the VC:

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

So the result is:

\[
\begin{align*}
\text{VC}(t := x; x := e; c; x := t, B) &= \\
[\frac{x}{t}]\text{VC}(x := e; c; x := t, B) &= \\
[\frac{e}{x}]\text{VC}(c; x := t, B) &= \\
[\frac{e}{x}]\text{VC}(c, t/x, B)
\end{align*}
\]

An alternative formulation is to apply substitution to the command:

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

2 Unsound Let

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

1. \( c = \text{let } x = 1 \text{ in } \text{skip} \)
2. \( B = x = 1 \)
3. \( \sigma(x) = 0 \)
4. \( \sigma \models VC(c, b) \), since \( VC(c, b) \) is \( 1 = 1 \)
5. \( \langle c, \sigma \rangle \not\Downarrow \sigma' \), 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 \texttt{do-while}:

\[
\text{do } c \text{ while } b \equiv c ; \text{while } b \text{ do } c
\]

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

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

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

\[
\frac{\vdash \{ A \} c \{ B \} \quad \vdash \{ B \} \text{while } b \text{ do } c \{ B \land \neg b \}}{\vdash \{ A \} \text{do } c \text{ while } b \{ B \land \neg b \}}
\]

Which is then equivalent to:

\[
\frac{\vdash \{ A \} c \{ B \} \quad \vdash \{ B \land b \} c \{ B \}}{\vdash \{ A \} \text{do } c \text{ while } b \{ B \land \neg b \}}
\]

1
4 Backwards VC

First, we unwind do-while:

\[
do_{\text{Inv}} c \text{ while } b \equiv c \text{ ; while}_{\text{Inv}} b \text{ do } c
\]

Now we can compute the VC:

\[
\begin{align*}
\text{VC}(c; \text{while}_{\text{Inv}} b \text{ do } c, B) & = \\
\text{VC}(c, \text{VC(while}_{\text{Inv}} b \text{ do } c, B)) & = \\
\text{VC}(c, \text{Inv} \land (\forall x_1 \ldots x_n. \text{Inv} \Rightarrow ((b \Rightarrow \text{VC}(c, \text{Inv})) \land \neg b \Rightarrow B))) & = \\
\text{VC}(c, \text{Inv} \land (\forall x_1 \ldots x_n. \text{Inv} \Rightarrow ((b \Rightarrow \text{VC}(c, \text{Inv})) \land \neg b \Rightarrow B)) & =
\end{align*}
\]

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

For the alternate formulation (4B), we again start by unwinding:

\[
do_{\text{Inv}1, \text{Inv}2} c \text{ while } b \equiv \text{assert}(\text{Inv}1); c ; \text{while}_{\text{Inv}2} b \text{ do } c
\]

The assertion comes from the problem description that \(\text{Inv}1\) is true before and after \(c\) is executed. We use \(\text{Inv}2\) to refer to the loop invariant of the while loop; it is typically the same as \(\text{Inv}1\), but may potentially be stronger (i.e., it may incorporate information from the first execution of \(c\)). Note that \(\text{Inv}2\) must imply \(\text{Inv}1\) since \(\text{Inv}1\) must also be true on every iteration of the while loop. So the result is:

\[
\begin{align*}
\text{VC}(&\text{assert}(\text{Inv}1 \land \text{Inv}2 \Rightarrow \text{Inv}1); c ; \text{while}_{\text{Inv}2} b \text{ do } c, B) = \\
\text{VC}(&\text{assert}(\text{Inv}1 \land \text{Inv}2 \Rightarrow \text{Inv}1), \text{VC}(c; \text{while}_{\text{Inv}2} b \text{ do } c, B)) = \\
\text{Inv}1 \land \text{Inv}2 &\Rightarrow \text{Inv}1 \land \text{VC}(c; \text{while}_{\text{Inv}2} b \text{ do } c, B) \\
\text{Inv}1 \land \text{Inv}2 &\Rightarrow \text{Inv}1 \land \text{VC}(c, \text{Inv}2 \land (\forall x_1 \ldots x_n. \text{Inv}2 \Rightarrow (b \Rightarrow \text{VC}(c, \text{Inv}2)) \land \neg b \Rightarrow B))
\end{align*}
\]

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

\[
\text{Inv} \land (\forall x_1 \ldots x_n. \text{Inv} \Rightarrow (b \Rightarrow \text{VC}(c, \text{Inv}) \land \neg b \Rightarrow B))
\]

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:

\[
\begin{align*}
\text{Inv} \land (\forall x. \text{Inv} \Rightarrow (false \Rightarrow \text{VC}(x := 1, \text{Inv}) \land true \Rightarrow x = 1)) & = \\
\text{Inv} \land (\forall x. \text{Inv} \Rightarrow true \Rightarrow x = 1) & = \\
\text{Inv} \land (\forall x. \text{Inv} \Rightarrow x = 1) & =
\end{align*}
\]

There is no value of \(\text{Inv}\) for which this works. If we take \(\text{Inv}\) to be \(x = 1\) we satisfy the right conjunct but cannot satisfy the left. If we take \(\text{Inv}\) 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:

\[
\begin{align*}
\text{VC}(x := 1, x = 1) \land (\forall x. \text{Inv} \Rightarrow (false \Rightarrow \text{VC}(x := 1, \text{Inv}) \land true \Rightarrow x = 1)) & = \\
\text{VC}(x := 1, x = 1) \land (\forall x. \text{Inv} \Rightarrow (true \Rightarrow x = 1)) & =
\end{align*}
\]

We can take \(\text{Inv}\) to be \(x = 1\):

\[
\begin{align*}
\text{VC}(x := 1, x = 1) \land (\forall x. x = 1 \Rightarrow (true \Rightarrow x = 1)) & = \\
\text{VC}(x := 1, x = 1) \land (\forall x. x = 1 \Rightarrow x = 1) & = \\
\text{VC}(x := 1, x = 1) \land (\forall x. true) & = \\
\text{VC}(x := 1, x = 1) & = \\
\frac{1/x}{x = 1} & = \\
1 & = true
\end{align*}
\]