AUTOMATIC VERIFICATION

The process of annotating a program that is specified with a precondition and a postcondition to a complete proof, and extracting the resulting proof obligations, can be completely automated, provided all loops are annotated with (candidate) loop invariants.

The conjunction of all proof obligations is usually called verification condition.

The tricky part in extracting the verification condition compositionally (by induction on the structure of the program) is that it has to perform two tasks:
- compute a precondition
- accumulate all proof obligations

Here is one possible presentation of a verification condition generator (VCG). It takes a program annotated with precondition, postcondition and loop invariants, and applies recursively a helper function that takes a command, a postcondition and a verification condition and returns a precondition and a new verification condition (essentially the original one with all new proof obligations in conjunction).

A VC GENERATOR

We define a mapping $vcg : Cmd \to A \times A \to A \times A$

$$vcg[\![x = E]\!](\phi, \psi) \overset{def}{=} (\phi[E/x], \psi)$$

$$vcg[\![C_1; C_2]\!](\phi, \psi) \overset{def}{=} vcg[\![C_1]\!](vcg[\![C_2]\!](\phi, \psi))$$

$$\begin{aligned} & vcg[\![\text{if } B \{C_1\} \text{ else } \{C_2\}]\!](\phi, \psi) \overset{def}{=} \\ & \quad \underline{\text{let}}\ (\phi_1, \psi_1) = vcg[\![C_1]\!](\phi, \psi), \\ & \qquad (\phi_2, \psi_2) = vcg[\![C_2]\!](\phi, \psi) \\ & \quad \underline{\text{in}}\ ((B \to \phi_1) \wedge (\neg B \to \phi_2),\ \psi_1 \wedge \psi_2) \end{aligned}$$

$$\begin{aligned} & vcg[\![\langle\!| \eta |\!\rangle \text{ while } B \{C\}]\!](\phi, \psi) \overset{def}{=} \\ & \quad \underline{\text{let}}\ (\phi', \psi') = vcg[\![C]\!](\eta, \text{true}) \\ & \quad \underline{\text{in}}\ (\eta, \psi \wedge \psi' \wedge (\eta \wedge B \to \phi') \wedge (\eta \wedge \neg B \to \phi)) \end{aligned}$$

Finally, we define:

$$\begin{aligned} & VCG(\langle\!| \phi |\!\rangle\ C\ \langle\!| \psi |\!\rangle) \overset{def}{=} \\ & \quad \underline{\text{let}}\ (\phi', \psi') = vcg[\![C]\!](\psi, \text{true}) \\ & \quad \underline{\text{in}}\ \psi' \wedge (\phi \to \phi') \end{aligned}$$

EXAMPLE

We verify $(\!|x<0|\!)$ while $(x!=0)\{x=x-1;\}$ $(\!|\text{false}|\!)$.
Notice that we chose the precondition as loop invariant!

First, we compute:

$$
\begin{aligned}
& \text{vcg}[\![(\!|x<0|\!) \text{ while} (x!=0)\{x=x-1\}]\!](\text{false}, \text{true}) \\
= \; & \underline{\text{let}}\ (\phi', \psi') = \text{vcg}[\![x=x-1]\!](x<0, \text{true}) \\
& \quad \underline{\text{in}}\ (x<0, \text{true} \wedge \psi' \wedge (x<0 \wedge x\neq 0 \to \phi') \wedge (x<0 \wedge \neg(x\neq 0) \to \text{false})) \\
= \; & \underline{\text{let}}\ (\phi', \psi') = (x-1<0, \text{true}) \\
& \quad \underline{\text{in}}\ \dots \\
= \; & (x<0, \text{true} \wedge \text{true} \wedge (x<0 \wedge x\neq 0 \to x-1<0) \wedge (x<0 \wedge \neg(x\neq 0) \to \text{false}))
\end{aligned}
$$

Finally, we obtain:

$$
\begin{aligned}
& \text{VCG}((\!|x<0|\!) \text{ while } (x!=0)\{x=x-1;\} (\!|\text{false}|\!)) \\
= \; & \text{true} \wedge \text{true} \wedge (x<0 \wedge x\neq 0 \to x-1<0) \wedge (x<0 \wedge \neg(x\neq 0) \to \text{false}) \wedge (x<0 \to x<0)
\end{aligned}
$$

The resulting verification condition is easily proved.

WEAKEST LIBERAL PRECONDITIONS

The weakest liberal precondition w.r.t. a command C and a postcondition Ψ is the weakest (w.r.t. logical implication) formula Θ such that $\models_{par} (\!|\Theta|\!)\, C\, (\!|\Psi|\!)$. We denote such a formula $wlp(C, \Psi)$.

Then $\models_{par} (\!|\phi|\!)\, C\, (\!|\Psi|\!)$ if and only if $\models \phi \to wlp(C, \Psi)$.

If we could compute $wlp(C, \Psi)$ we would have an algorithmic method to reduce program correctness to validity of arithmetic formulas.

One can prove that:

$$wlp(x=E, \Psi) \Leftrightarrow \Psi[E/x]$$
$$wlp(C_1; C_2, \Psi) \Leftrightarrow wlp(C_1, wlp(C_2, \Psi))$$
$$wlp(\text{if } B\{C_1\} \text{else}\{C_2\}, \Psi) \Leftrightarrow (B \to wlp(C_1, \Psi)) \wedge (\neg B \to wlp(C_2, \Psi))$$

Notice that the equivalences respect the principle of structural induction: we can therefore use them to compute inductively the weakest liberal precondition for loop-free programs. However, there is no possibility to define $wlp(\text{while } B\{C\}, \Psi)$ in our assertion language.

EXERCISE Use the above ideas to verify Abs:

$$(\!|x = x_0|\!)\ \text{if}\,(x>0)\{y=x;\}\,\text{else}\{y=-x;\}\ (\!|y = |x_0||\!)$$

Note: We could handle loops in a unsound way by unfolding them a given number of times, thus replacing while with if.