21. Verification Conditions

Goal: Develop a method that shows

$$\models \{A\}\, c\, \{B\}$$

automatically.

Idea:
- With the previous development, $\models \{A\}\, c\, \{B\}$ is equivalent to proving the assertion

$$A \to pred(c, B)$$

valid.
- Note that in $A \to pred(c, B)$ the program has been eliminated and we reason solely in logic.
The hope is that a theorem prover could carry out the logical reasoning fully automatically.

Problems:
- Finding loop invariants is difficult.
- Proving implications among heavy (quantified) logical formulas
a difficult one could indeed take the invariant with the B-predicate).

Approach: Allow ourselves some human guidance
- Add additional assertions as annotations to our program.
- Annotations contain information concerning loop invariants and intermediate values.
- Besides taking away the burden of finding loop invariants, the information should also ease the logical reasoning for the theorem prover.

Definition (Syntax of annotated programs):
- The syntax of annotated programs is given by the grammar

$$c ::= skip \mid x := a \mid c; x := a \mid c_1; \{D\}\, c_2$$
$$\mid \underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi} \mid \underline{while}\ b\ \underline{do}\ \{D\}\, c\ \underline{od}.$$

Here, D is an assertion.

Moreover, in $c_1 ; \{R\} c_2$ program c_2 is assumed to be different from an assignment.
The point for this side condition is that for assignments the intermediate assertions are easy to compute.

- An annotated partial correctness statement is a triple $\{P\} c \{Q\}$, where c is an annotated program.
It is valid if the corresponding (drop the annotations) unannotated partial correctness statement is.

Example (Factorial function):
Reconsider the program for computing the factorial function for which we determined the invariant

$$I :\Leftrightarrow y \cdot x! = n! \wedge x \geq 0.$$

When written as an annotated program, we make the invariant explicit:

w := while $x > 0$ do
$\{I\}$
$y := xy;$
$x := x - 1;$
od

Approach (in more detail):
Our goal is to associate with an annotated correctness statement $\{P\} c \{Q\}$ a set of assertions, called verification conditions and denoted $vc(\{P\} c \{Q\})$, so that validity of all verification conditions entails validity of the annotated partial correctness statement.

Idea:

Consider the annotated while-loop

$\{A\}$ while b do $\{D\}$ c od $\{B\}$

- As in the example above, D should be chosen to be an invariant.
  This means
  $\{D \wedge b\}\ c\ \{D\}$
  should be a valid annotated partial correctness statement
  (here one sees the recursion for computing vc).
- Moreover, if we can show
  $A \to D$ and $D \wedge \neg b \to B$,
  we get validity of
  $\{A\}$ while b do c od $\{B\}$
  using Hoare's Rules (while) for
  $\{D\}$ while b do c od $\{D \wedge \neg b\}$
  and Rule (Consequence) to conclude.
- So the verification conditions for while-programs should be

$$vc(\{A\}\text{while } b \text{ do } \{D\}\ c \text{ od } \{B\}) := vc(\{D \wedge b\}\ c\ \{D\}) \cup \{A \to D,\ D \wedge \neg b \to B\}.$$

Definition:

The function vc that assigns to an annotated partial correctness statement a set of verification conditions (assertions) is defined by

$$vc(\{A\}\text{skip}\{B\}) := \{A \to B\}$$

$$vc(\{A\}\, x := a\, \{B\}) := \{A \to B[^x/_a]\}$$

$$vc(\{A\}\, c_1; x := a\, \{B\}) := vc(\{A\}\, c_1\, \{B[^x/_a]\})$$

$vc(\{A\}\, c_1; \{D\}\, c_2\, \{B\}) := vc(\{A\}\, c_1\, \{D\}) \cup vc(\{D\}\, c_2\, \{B\})$, where c_2 is not an assignment.

$vc(\{A\}\ \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\ \underline{\text{fi}}\ \{B\}) := vc(\{A \wedge b\}\, c_1\, \{B\}) \cup vc(\{A \wedge \neg b\}\, c_2\, \{B\})$

$vc(\{A\}\ \underline{\text{while}}\ b\ \underline{\text{do}}\ \{D\}\, c\ \underline{\text{od}}\ \{B\}) := vc(\{D \wedge b\}\, c\, \{D\}) \cup \{A \to D,\ D \wedge \neg b \to B\}$.

<u>Theorem (Soundness)</u>:

$\models vc(\{A\}\, c\, \{B\})$ implies $\models \{A\}\, c\, \{B\}$.

Note that the left-hand side is validity of a set of formulas, i.e. the program has been fully eliminated.
The proof is by induction on the structure of annotated programs and makes use of soundness of Hoare's rules.

<u>Example (Factorial function, continued)</u>:

Consider $A := x = n \wedge n \geq 0 \wedge y = 1$

$B := y = n!$

Program w is the one from above.

Then $vc(\{A\}\, w\, \{B\})$

$= vc(\{I \wedge x > 0\}\, y := xy;\ x := x - 1\, \{I\})$

$\cup \{A \to I,\ I \wedge \neg x > 0 \to B\}$

$= vc(\{I \wedge x > 0\}\, y := xy\, \{I[x/x-1]\})$

$\cup \{A \to I,\ I \wedge \neg x > 0 \to B\}$

$= \{\underbrace{I \wedge x > 0 \to (I[x/x-1])[y/xy]}_{V_1},\ \underbrace{A \to I}_{V_2},\ \underbrace{I \wedge \neg x > 0 \to B}_{V_3}\}$.

We have

$\models y \cdot x^{\underline{?}} = n^{\underline{?}} \wedge x \geq 0 \wedge x > 0 \rightarrow xy \cdot (x-1)^{\underline{?}} = n^{\underline{?}} \wedge x - 1 \geq 0$ for V_1,

$\models x = n \wedge n \geq 0 \wedge y = 1 \rightarrow y \cdot x^{\underline{?}} = n^{\underline{?}} \wedge x \geq 0$ for V_2, and

$\models y \cdot x^{\underline{?}} = n^{\underline{?}} \wedge x \geq 0 \wedge \neg x > 0 \rightarrow y = n^{\underline{?}}$ for V_3 (holds as $0^{\underline{?}} = 1$).

Hence, by the theorem □

$\models \{A\} c \{B\}$.

The benefit of verification conditions is that we can discharge the task of checking the assertions (V_1 to V_3 in the example) to a theorem prover (that often implements resolution or tableau) or to a satisfiability modulo theories (SMT) solver.
This means deciding the verification conditions is indeed automated.

Lemma (Verification conditions are sound (see above) but not complete):

$\models \{A\} c \{B\}$ does not imply $\models vc(\{A\} c \{B\})$.

Proof:

Consider

$\models \{true\}$ while false do $\{false\}$ skip od $\{true\}$.

But

$vc(\{true\}$ while false do $\{false\}$ skip od $\{true\})$

$= vc(\{false \wedge false\}$ skip $\{false\})$

$\cup \{true \rightarrow false, false \wedge \neg false \rightarrow true\}$

Since $true \rightarrow false$ is not valid, the set is not valid. □

Remark (On an ugly aspect in the way we defined annotated programs):

The programs built as $(c; x := a_2); x := a_2$ and $c; (x := a_1; x := a_2)$ are typically understood as being the same, namely $c; x := a_1; x := a_2$.

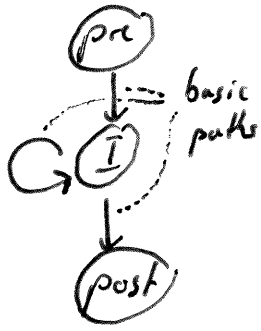
In the syntax of annotated programs given above, however, they support different annotations:

- $(c; x := a_1); x := a_2$ only allows us to annotate c, no annotations before the assignments,
- In the second case, besides the annotations in c we can have
$$c; \{Q\}\, (x := a_1; x := a_2),$$
since a pair of assignments is not an assignment.

Remark (Loop invariants are enough):

One can reduce the annotation effort to loop invariants.

In this setting, a verification condition is computed for every basic path:



A basic path is a sequence of commands that
- starts in the precondition or in a loop invariant
- ends in a loop invariant or in the postcondition
- and does not meet further loop invariants on the way.

In this formulation, conditions are replaced by so-called assume statements.

For details, see Bradley & Manna: The Calculus of Computation, Springer 2007.

An interesting project is to write your own verification condition generator.