

Weakest Precondition Calculus II

COMP2600 — Formal Methods for Software Engineering

Ranald Clouston

Australian National University

Semester 2, 2011

Guaranteeing Termination

If a loop is never entered, then the postcondition must already be true and the boolean control expression false. We'll call that precondition P_0 .

$$P_0 \equiv \neg b \wedge Q$$

Now suppose the loop executes exactly once. In that case:

- b must be true initially;
- after the first time through the loop, P_0 *must become true* (so that the loop terminates next time through):

$$P_1 \equiv b \wedge wp(S, P_0)$$

Guaranteeing Termination ctd

Similarly,

$$P_2 \equiv b \wedge wp(S, P_1)$$

$$P_3 \equiv b \wedge wp(S, P_2)$$

...

This is an ***inductive definition***:

$$P_0 \equiv \neg b \wedge Q$$

$$P_{k+1} \equiv b \wedge wp(S, P_k)$$

If any of the P_k is true in the initial state, then we are guaranteed that the loop will terminate and establish the postcondition Q .

Weakest Preconditions for While Loops (Rule 4/4)

$$wp(\mathbf{while} \ b \ \mathbf{do} \ S, Q) \equiv \exists k. ((k \geq 0) \wedge P_k)$$

where P_k is defined inductively:

$$P_0 \equiv \neg b \wedge Q$$

$$P_{k+1} \equiv b \wedge wp(S, P_k)$$

Interpretation:

P_k is the weakest precondition that ensures that the body S executes exactly k times and terminates in a state in which postcondition Q holds.

Using the While Rule in Practice

The rule asks us to write P_k down as a single formula, but how do we do that when P_0, P_1, P_2, \dots is an infinitely long chain of formulas?

Trick One: The following (informal) equivalence may be useful:

$$\exists k. ((k \geq 0) \wedge P_k) \equiv P_0 \vee P_1 \vee P_2 \vee \dots$$

e.g. if

$$P_0 \equiv (n = 0)$$

$$P_1 \equiv (n = 1)$$

$$P_2 \equiv (n = 2) \text{ etc...}$$

then our precondition is

$$(n = 0) \vee (n = 1) \vee (n = 2) \vee \dots \equiv (n \geq 0)$$

Using the While Rule in Practice ctd

Trick Two: express P_k as a single formula **parameterised by k** .

e.g. if

$$P_0 \equiv (n = 0)$$

$$P_1 \equiv (n = 1)$$

$$P_2 \equiv (n = 2) \text{ etc...}$$

then

$$P_k \equiv (n = k)$$

and so our precondition is

$$\exists k. ((k \geq 0) \wedge (n = k)) \equiv (n \geq 0)$$

Strictly speaking, the correctness of our P_k should be ***proved by induction***.

Example 1

Suppose we want to find:

$$wp(\text{while } n > 0 \text{ do } n := n - 1, n = 0)$$

Step 1 — finding P_k :

$$P_0 \equiv \neg (n > 0) \wedge (n = 0) \equiv (n = 0)$$

$$P_1 \equiv (n > 0) \wedge wp(n := n - 1, n = 0) \equiv (n = 1)$$

$$P_2 \equiv (n > 0) \wedge wp(n := n - 1, n = 1) \equiv (n = 2)$$

...

$$P_k \equiv (n = k)$$

This is only an informed hypothesis — We really need to prove by induction on k that P_k is defined as above. (Exercise.)

Example 1 ctd

Step 2 — finding the weakest precondition:

$$\begin{aligned}\exists k. ((k \geq 0) \wedge P_k) &\equiv \exists k. ((k \geq 0) \wedge (n = k)) \\ &(\equiv (n = 0) \vee (n = 1) \vee (n = 2) \vee \dots) \\ &\equiv (n \geq 0)\end{aligned}$$

Thus,

$$wp(\text{while } n > 0 \text{ do } n := n - 1, n = 0) \equiv (n \geq 0)$$

as we might have expected.

Example 2 (Total Correctness)

If the condition in example 1 was $(n \neq 0)$, the algorithm will not terminate for initial values of n less than 0.

We want to find

$$wp(\text{while } n \neq 0 \text{ do } n:=n-1, n = 0)$$

Step 1 – finding P_k :

$$P_0 \equiv \neg (n \neq 0) \wedge (n = 0) \equiv (n = 0)$$

$$P_1 \equiv (n \neq 0) \wedge wp(n:=n-1, n = 0) \equiv (n = 1)$$

$$P_2 \equiv (n \neq 0) \wedge wp(n:=n-1, n = 1) \equiv (n = 2)$$

...

$$P_k \equiv (n = k)$$

Example 2 ctd

Step 2 — finding the weakest precondition:

$$\begin{aligned}\exists k. ((k \geq 0) \wedge P_k) &\equiv \exists k. ((k \geq 0) \wedge (n = k)) \\ &\equiv (n \geq 0)\end{aligned}$$

Thus,

$$wp(\text{while } n \neq 0 \text{ do } n:=n-1, n = 0) \equiv (n \geq 0)$$

This is not really any different from Example 1, of course.

Example 1, Revisited ...

Suppose we want to calculate

$$wp(\text{while } (n > 0) \text{ do } n := n - 1, n = -100)$$

Intuitively, if $n \leq 0$, the loop terminates immediately with the value of n unchanged, so we expect the weakest precondition above to be $n = -100$.

Step 1 – finding P_k :

$$P_0 \equiv \neg (n > 0) \wedge (n = -100) \equiv (n = -100)$$

$$P_1 \equiv (n > 0) \wedge wp(n := n - 1, n = -100) \equiv (n > 0) \wedge (n = -99) \equiv \mathbf{False}$$

$$P_2 \equiv (n > 0) \wedge wp(n := n - 1, \mathbf{False}) \equiv (n > 0) \wedge \mathbf{False} \equiv \mathbf{False}$$

...

Example 1, Revisited, ctd ...

Therefore:

$$P_0 \equiv (n = -100)$$

$$P_k \equiv \mathbf{False}, \quad \text{for all } k > 0$$

Step 2 — finding the weakest precondition:

$$\begin{aligned} \exists k. ((k \geq 0) \wedge P_k) &\equiv P_0 \vee P_1 \vee P_2 \vee \dots \\ &\equiv (n = -100) \vee \mathbf{False} \vee \mathbf{False} \vee \mathbf{False} \vee \dots \\ &\equiv (n = -100) \end{aligned}$$

which is what we expected.

Example 2, Revisited

What happens if we provide the same postcondition to example 2?

$$wp(\text{while } (n \neq 0) \text{ do } n:=n-1, n = -100)$$

Intuitively, there is *no* initial state which will lead to this postcondition. If $n < 0$ in the initial state, the loop will not terminate.

Step 1 – finding P_k :

$$P_0 \equiv \neg (n \neq 0) \wedge (n = -100) \equiv \mathbf{False}$$

$$P_1 \equiv (n \neq 0) \wedge wp(n:=n-1, \mathbf{False}) \equiv (n \neq 0) \wedge \mathbf{False} \equiv \mathbf{False}$$

$$P_2 \equiv (n \neq 0) \wedge wp(n:=n-1, \mathbf{False}) \equiv (n \neq 0) \wedge \mathbf{False} \equiv \mathbf{False}$$

...

Example 2, Revisited, ctd ...

Therefore:

$$P_k \equiv \mathbf{False}, \quad \text{for all } k \geq 0$$

Step 2 — finding the weakest precondition:

$$\begin{aligned} \exists k. ((k \geq 0) \wedge P_k) &\equiv P_0 \vee P_1 \vee P_2 \vee \dots \\ &\equiv \mathbf{False} \vee \mathbf{False} \vee \mathbf{False} \vee \dots \\ &\equiv \mathbf{False} \end{aligned}$$

So there is *no* initial state which leads to a final state with $n = -100$.

Example 1, Again ...

Suppose we wanted to determine the conditions under which the program *terminates*. A postcondition of **True** says nothing about the final state. So, if we calculate

$$wp(\text{while } (n > 0) \text{ do } n := n - 1, \mathbf{True})$$

then the precondition will guarantee termination but nothing else.

Step 1 – finding P_k :

$$P_0 \equiv \neg (n > 0) \wedge \mathbf{True} \equiv (n \leq 0)$$

$$P_1 \equiv (n > 0) \wedge wp(n := n - 1, n \leq 0) \equiv (n > 0) \wedge (n - 1 \leq 0) \equiv (n = 1)$$

$$P_2 \equiv (n > 0) \wedge wp(n := n - 1, n = 1) \equiv (n = 2)$$

...

Example 1 Again, ctd

Step 2 — finding the weakest precondition:

$$\begin{aligned}\exists k. ((k \geq 0) \wedge P_k) &\equiv (n \leq 0) \vee (n = 1) \vee (n = 2) \vee \dots \\ &\equiv \mathbf{True}\end{aligned}$$

So the program always terminates.

Example 2, Termination ...

$wp(\text{while } n \neq 0 \text{ do } n:=n-1, \mathbf{True})$

$$P_0 \equiv \neg(n \neq 0) \wedge \mathbf{True} \equiv (n = 0)$$

$$P_1 \equiv (n \neq 0) \wedge wp(\mathbf{n:=n-1}, n = 0) \equiv (n \neq 0) \wedge (n - 1 = 0) \equiv (n = 1)$$

$$P_2 \equiv (n \neq 0) \wedge wp(\mathbf{n:=n-1}, n = 1) \equiv (n = 2)$$

...

So

$$\exists k. ((k \geq 0) \wedge P_k) \equiv (n = 0) \vee (n = 1) \vee (n = 2) \vee \dots \equiv (n \geq 0)$$

Therefore the program terminates provided n is non-negative.