Summary of Lecture I

Rules for calculating the weakest precondition:

Assignment Rule: $wp(\texttt{x:=e},\ Q(x)) \equiv Q(e)$

Sequence Rule: $wp(S_1;S_2,\ Q) \equiv wp(S_1,\ wp(S_2,\ Q))$

Conditional Rule:

$$wp(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, Q) \equiv (b \Rightarrow wp(S_1, Q)) \wedge (\neg b \Rightarrow wp(S_2, Q))$$

Equivalent Conditional Rule:

$$wp(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, Q) \equiv (b \wedge wp(S_1, Q)) \vee (\neg b \wedge wp(S_2, Q))$$

Conditionals Without Else Rule:

$$wp(\textbf{if } b \textbf{ then } S, Q) \equiv (b \Rightarrow wp(S_1, Q)) \wedge (\neg b \Rightarrow Q) \equiv (b \wedge wp(S_1, Q)) \vee (\neg b \wedge Q)$$

Loops

(The thing to do now is hang on tightly . . .)

Suppose we have a while loop and some postcondition Q.

The precondition P we seek is the weakest that:

- establishes Q
- *guarantees termination*

We can take hints for the first requirement from the corresponding rule for Hoare Logic. That is, think in terms of *loop invariants*.

But termination is a bigger problem...

An Undecidable Problem

You already know from the lectures on Turing machines that some problems are ***undecidable***.

This doesn't mean just that we haven't yet found a suitable algorithm;

- It means that we can prove with maths that there *cannot be such an algorithm!*

Determining if a program terminates or not on a given input is just such an undecidable problem.

So there's no algorithm to compute $wp(\textbf{while } b \textbf{ do } S, Q)$ in all cases.

But that doesn't mean there are no techniques to tackle this problem that at least work some of the time!

Guaranteeing Termination: $\{P\}$ `while` b `do` S $\{Q\}$

The precondition P we seek is the weakest that establishes Q and *guarantees termination*. Our rules for $wp(S,Q)$ give us the first part, but termination is a bigger problem ... so let us look at how a loop can terminate ...

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

$$P_0 \equiv \neg b \wedge Q \qquad\qquad i.e.\ \{\neg b \wedge Q\} \text{ do nothing } \{Q\}$$

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):

$$P_1 \equiv b \wedge wp(S, P_0) \qquad\qquad i.e.\ \{b \wedge wp(S,P_0)\}\ \mathtt{S}\ \{P_0\}$$

Guaranteeing Termination ctd: $\{P\}$ `while` b `do` S $\{Q\}$

$$P_0 \equiv \neg b \wedge Q \qquad i.e.\ \{\neg b \wedge Q\} \text{ do nothing } \{Q\}$$

$$P_1 \equiv b \wedge wp(S, P_0) \qquad i.e.\ \{b \wedge wp(S, P_0)\}\ \mathtt{S}\ \{P_0\}$$

Similarly,

$$P_2 \equiv b \wedge wp(S, P_1) \qquad i.e.\ \{b \wedge wp(S, P_1)\}\ \mathtt{S}\ \{P_1\}$$

$$P_3 \equiv b \wedge wp(S, P_2) \qquad i.e.\ \{b \wedge wp(S, P_2)\}\ \mathtt{S}\ \{P_2\}$$

$$\ldots$$

Read P_k as the ***weakest precondition*** under which the loop terminates with postcondition Q after *exactly* k iterations.

But each of these P_k looks quite similar to the next, so we can capture this sequence with an ***inductive definition***.

An Inductive Definition

$$P_0 \equiv \neg b \wedge Q \qquad \{\neg b \wedge Q\} \text{ loop does nothing } \{\neg b \wedge Q\}$$

$$P_1 \equiv b \wedge wp(S, P_0) \qquad \{b \wedge wp(S, P_0)\} \text{ S } \{P_0\}$$

$$P_2 \equiv b \wedge wp(S, P_1) \qquad \{b \wedge wp(S, P_1)\} \text{ S } \{P_1\}$$

$$\ldots$$

leads to the 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.

i.e. $\{P_0 \vee P_1 \vee \cdots\}$ `while` b `do` S $\{Q\}$ is true

Weakest Preconditions for While Loops (Rule 4/4)

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

where P_k is defined inductively:

$$\begin{aligned} P_0 &\equiv \neg b \land Q \\ P_{k+1} &\equiv b \land wp(S, P_k) \end{aligned}$$

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.

If our loop is to terminate with postcondition Q, some P_k must hold before we enter the loop.
i.e. $\{P_0 \lor P_1 \lor \cdots\}$ `while` b `do` S $\{Q\}$ is true

The problem with P_k

Applying the wp function to a while loop and postcondition will produce an assertion of the form

$$\exists k. \, (\, k \geq 0 \, \wedge \, P_k \,)$$

But P_k is defined only via an inductive definition 'on the side'.

Indeed, P_k may be *different* for each k, so our wp function has dropped an *infinitely long* assertion on us!

Such an assertion is unsuitable for further manipulations, e.g. if before the loop there are some assignments we want to apply the assignment rule to.

The problem with P_k ctd.

We can simplify matters by expressing P_k as a *single, finite* formula that is **parameterised by k**.

e.g. if

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

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

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

then

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

This looks like a likely choice, but the correctness of our P_k must be ***proved by induction***.

Example 1

Suppose we want to find:

$wp(\texttt{while n>0 do n:=n-1}, n=0)$ $\qquad$ $i.e.\ wp(\texttt{while b do S}, Q)$

We can start by generating some of our P_k sequence:

$$P_0 \equiv \neg(n>0) \wedge (n=0) \equiv (n=0) \qquad i.e.\ \neg b \wedge Q$$

$$P_1 \equiv (n>0) \wedge wp(\texttt{n:=n-1}, n=0) \equiv (n=1) \qquad i.e.\ b \wedge wp(S, P_0)$$

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

... so it looks pretty likely that

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

But we need induction to be sure - `http://spikedmath.com/449.html` [a].

[a] See `http://mathworld.wolfram.com/CircleDivisionbyChords.html` if you're curious.

Example 1 – Using Induction to prove $P_k \;\equiv\; (n = k)$

$wp($`while n>0 do n:=n-1`$,\, n=0)$ $\qquad$ *i.e.* $wp($`while b do S`$,\, Q)$

We've already done our ***base case***:

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

Now for our ***induction step***:

- we'll assume that $P_i \;\equiv\; (n = i)$ for some $i \geq 0$
- and investigate P_{i+1}: $\qquad$ recall that $P_{i+1} \;\equiv\; b \wedge wp(S, P_i)$

$$\begin{aligned}
P_{i+1} \;&\equiv\; n > 0 \wedge wp(\texttt{n:=n-1}, n = i) \\
&\equiv\; (n > 0) \wedge (n - 1 = i) \\
&\equiv\; (n > 0) \wedge (n = i + 1) \\
&\equiv n = i + 1 \qquad\qquad ((n = i + 1) \wedge (i \geq 0)) \Rightarrow (n > 0)
\end{aligned}$$

By the principle of induction: $\forall k \geq 0.(P_k \equiv (n = k))$

Example 1 ctd

Induction proof under our belt, we now have

$$wp(\texttt{while n>0 do n:=n-1},\ n=0)\ \equiv\ \exists k.\ (\ k \geq 0\ \wedge\ n=k\ )$$

This is finite, which is certainly an improvement, but we can simplify it further.

Useful trick: Use the general fact that

$$\exists k.\ (\ (k \geq 0) \wedge P_k\ )\ \ \equiv\ \ P_0 \vee P_1 \vee P_2 \vee P_3 \vee \cdots$$

So in this example we have

$$(n=0) \vee (n=1) \vee (n=2) \vee (n=3) \vee \cdots$$

We can compress this infinite disjunction into a finite final result:

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

Example 2 (Total Correctness)

We want to find

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

Step 1 – finding P_k:

$$P_0 \equiv \neg(n \neq 0) \wedge (n = 0) \equiv (n = 0) \qquad i.e.\ \neg b \wedge Q$$

$$P_1 \equiv (n \neq 0) \wedge wp(\texttt{n:=n-1}, n = 0) \equiv (n = 1) \qquad i.e.\ b \wedge wp(S, P_0)$$

$$\ldots$$

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

(Induction omitted)

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,

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

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

But look more closely ... what is the trap in this while-loop?

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,

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

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

But look ... we have automatically found the fact that the while-loop will not terminate for initial values of `n` less than 0.

The Postcondition ‘True’

Suppose we wanted to calculate

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

$True$ may seem a ludicrous postcondition to prove something about.

After all, $True$ is an assertion so weak it holds of *any* memory state!

Indeed, $\{P\}S\{True\}$ is a true statement of Hoare Logic for any precondition P and code fragment S whatsoever.

But remember the WP calculus cares about ***total correctness***, so $wp(S, True)$ is the weakest precondition on which S ***terminates***.

(To be precise, on which S terminates into a state satisfying $True$, but this addition is vacuous.)

Example 1, Revisited . . .

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

Step 1 – finding P_k:

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

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

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

$$\ldots$$

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

(Induction omitted)

Example 1 Revisited ctd.

Step 2 — finding the weakest precondition:

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

So the program

```
while (n>0) do n:=n-1
```

always terminates.

Example 2, Termination ...

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

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

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

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

$$\ldots$$

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

(Induction omitted)

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

Therefore the program terminates provided n is non-negative.

Example 1, Again . . .

Suppose we want to calculate

$$wp(\texttt{while (n>0) do n:=n-1},\ n = -5)$$

Intuitively, if $n \leq 0$, the loop terminates immediately with the value of n unchanged, so we expect the weakest precondition above to be $n = -5$.

Step 1 – finding P_k:

$$\begin{aligned}
P_0 &\equiv \neg(n > 0) \wedge (n = -5) \;\equiv\; (n = -5) \\
P_1 &\equiv (n > 0) \wedge wp(\texttt{n:=n-1},\ n = -5) \;\equiv\; (n > 0) \wedge (n = -4) \;\equiv\; \textbf{False} \\
P_2 &\equiv (n > 0) \wedge wp(\texttt{n:=n-1},\ \textbf{False}) \;\equiv\; (n > 0) \wedge \textbf{False} \;\equiv\; \textbf{False} \\
&\ldots
\end{aligned}$$

Will it be $False$ all the way down?

When $P_k \equiv False$

Here's another ***useful trick***:

Suppose $P_k \equiv False$ for some k. Then

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

On what inputs will S terminate with an output satisfying $False$?

No memory state satisfies $False$, so $wp(S, False) \equiv False$ always, and

$$P_{k+1} \equiv b \wedge False \equiv False$$

Intuition: **if a loop cannot terminate after k steps then it cannot terminate after *any* larger number of steps**.