

Weakest Precondition Calculus

COMP2600 — Formal Methods for Software Engineering

Ranald Clouston

Australian National University
Semester 2, 2011

Edsger W. Dijkstra (1930 – 2002)

The originator of this week's material (in 1976), and a good source of quotes:

Program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence.

The question of whether Machines Can Think... [is] about as relevant as the question of whether Submarines Can Swim.

He also had some pretty uncompromising views on how introductory computer science should be taught:

In order to drive home the message that this introductory programming course is primarily a course in formal mathematics, we see to it that the programming language in question has not been implemented on campus so that students are protected from the temptation to test their programs.

Introduction

Dijkstra's **Weakest Precondition Calculus** is a variant on Hoare Logic.

- also known as *Predicate Transformer Semantics*.

Hoare Logic presents **logic** problems:

- Given a precondition P , code fragment S and postcondition Q , is $\{P\}S\{Q\}$ true?

WP is about evaluating a **function**:

- Given a code fragment S and postcondition Q , find the **unique** P which is the weakest precondition for S and Q .

The wp Function

If S is a code fragment and Q is a predicate on states, then the **weakest precondition** for S with respect to Q is a predicate that is true for *precisely* those initial states from which:

- S **must terminate**, and
- executing S must produce **a state satisfying Q** .

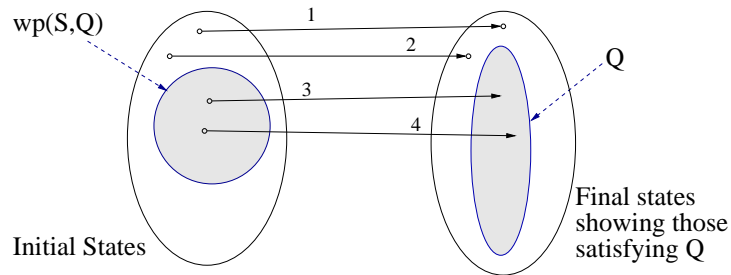
That is, the weakest precondition P is a **function** of S and Q :

$$P = wp(S, Q)$$

wp is sometimes called a **predicate transformer**, but it's just a **higher-order function**:

$$wp :: \text{Code} \times (\text{State} \rightarrow \text{Bool}) \rightarrow (\text{State} \rightarrow \text{Bool})$$

A Diagrammatic View of Weakest Preconditions



- Arcs 1 and 2 map initial states to states not satisfying Q .
- Arcs 3 and 4 map initial states to states that do satisfy Q .

Intuition Example 1

Consider code $x:=x+1$ and postcondition $(x > 0)$

One valid precondition is $(x > 0)$, so in Hoare Logic the following is true:

$$\{x > 0\} x:=x+1 \{x > 0\}$$

Another valid precondition is $(x > -1)$, so:

$$\{x > -1\} x:=x+1 \{x > 0\}$$

$(x > -1)$ is *weaker* than $(x > 0)$ (...because $(x > 0) \rightarrow (x > -1)$)

In fact, $(x > -1)$ is the *weakest* precondition:

$$wp(x:=x+1, x > 0) = (x > -1)$$

Relationship with Hoare logic

Hoare Logic is *relational*:

- For each Q , there are *many* P such that $\{P\}S\{Q\}$.
- For each P , there are *many* Q such that $\{P\}S\{Q\}$.

WP is *functional*:

- For each Q there is *exactly one* predicate $wp(S, Q)$.

Hoare Logic is about *partial correctness*:

- We *don't* care about termination.

WP is about *total correctness*:

- We *do* care about termination.

WP respects Hoare Logic: $\{wp(S, Q)\}S\{Q\}$ is true.

Intuition Example 2

Consider code $a:=a+1; b:=b-1$ and postcondition $a \times b = 0$

A very strong precondition is $(a = -1) \wedge (b = 1)$:

$$\{(a = -1) \wedge (b = 1)\} a:=a+1; b:=b-1 \{a \times b = 0\}$$

A weaker precondition is $a = -1$.

The *weakest* precondition is $(a = -1) \vee (b = 1)$

$$wp(a:=a+1; b:=b-1, a \times b = 0) = (a = -1) \vee (b = 1)$$

Intuition Example 3

The assignment axiom of Hoare Logic gives us (for any postcondition Q):

$$\{Q(y + 3 * z - 5)\} x := y + 3 * z - 5 \{Q(x)\}$$

Intuitive justification:

- Let v be the value arrived at by computing $y + 3 * z - 5$.
- If $Q(y + 3 * z - 5)$ is true initially, then so is $Q(v)$.
- Since the variable x has value v after the assignment (and nothing else changes in the state), it must be that $Q(x)$ holds after that assignment.

Intuition Example 3 ctd.

In fact $Q(y + 3 * z - 5)$ is the **weakest precondition**.

Intuitive justification:

- Let v be the value arrived at by computing $y + 3 * z - 5$.
- If $Q(x)$ is true after the assignment, so is $Q(v)$.
- If $Q(v)$ is true after the assignment, then it must also be true *before* the assignment, because x does not appear in $Q(v)$ and nothing else has changed.
- Thus, $Q(y + 3 * z - 5)$ was true initially.
- So (combined with previous slide), $Q(x)$ holds after the assignment **if and only if** $Q(y + 3 * z - 5)$ held before it.

Weakest Precondition of Assignment (Rule 1/4)

We previously argued that the Assignment axiom of Hoare Logic is designed to give the “best” — i.e. the **weakest** precondition:

$$\{Q(e)\} x := e \{Q(x)\}$$

Therefore we should expect that the rule for Assignment in the weakest precondition calculus corresponds closely:

$$wp(x := e, Q(x)) \equiv Q(e)$$

Assignment *wp* Examples

$$\begin{aligned} wp(x := y + 3, (x > 3)) &\equiv y + 3 > 3 && \text{(substitute } y + 3 \text{ for } x) \\ &\equiv y > 0 && \text{(simplify)} \end{aligned}$$

$$\begin{aligned} wp(n := n + 1, (n > 5)) &\equiv n + 1 > 5 && \text{(substitute } n + 1 \text{ for } n) \\ &\equiv n > 4 && \text{(simplify)} \end{aligned}$$

Weakest Preconditions for Sequences (Rule 2/4)

As in Hoare Logic, we expect the rule for sequencing to **compose** the effect of the consecutive statements. The rule is:

$$wp(S_1; S_2, Q) \equiv wp(S_1, wp(S_2, Q))$$

Example:

$$\begin{aligned} & wp(\mathbf{x:=x+2; y:=y-2}, (x + y = 0)) \\ \equiv & wp(\mathbf{x:=x+2}, wp(\mathbf{y:=y-2}, (x + y = 0))) \\ \equiv & wp(\mathbf{x:=x+2}, (x + (y - 2) = 0)) \\ \equiv & ((x + 2) + y - 2 = 0) \\ \equiv & (x + y = 0) \end{aligned}$$

Weakest Preconditions for Conditionals (Rule 3a/4)

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

Proof:

By cases on condition b ,

- b is true:
 wp for the conditional is the weakest precondition for S_1 guaranteeing postcondition Q – that is $wp(S_1, Q)$.
The right hand side reduces to the same thing.
- b is false:
Similarly, both left hand and right hand sides reduce to $wp(S_2, Q)$

Conditional Example:

$$\begin{aligned} & wp(\mathbf{if } x > 2 \mathbf{ then } y:=1 \mathbf{ else } y:=-1, (y > 0)) \\ \equiv & ((x > 2) \rightarrow wp(\mathbf{y:=1}, (y > 0))) \wedge (\neg(x > 2) \rightarrow wp(\mathbf{y:=-1}, (y > 0))) \\ \equiv & ((x > 2) \rightarrow (1 > 0)) \wedge (\neg(x > 2) \rightarrow (-1 > 0)) \\ \equiv & ((x > 2) \rightarrow \mathit{True}) \wedge ((x \leq 2) \rightarrow \mathit{False}) \\ \equiv & x > 2 \end{aligned}$$

(If you are unhappy with the last step, draw a truth table.)

Alternative Rule for Conditionals (Rule 3b/4)

It is often easier to deal with disjunctions and conjunctions than implications, so the following rule for conditionals is usually more convenient.

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

Conditional Example Again:

$$\begin{aligned}
 & wp(\text{if } x > 2 \text{ then } y := 1 \text{ else } y := -1, (y > 0)) \\
 \equiv & ((x > 2) \wedge wp(y := 1, (y > 0))) \vee (\neg(x > 2) \wedge wp(y := -1, (y > 0))) \\
 \equiv & ((x > 2) \wedge (1 > 0)) \vee (\neg(x > 2) \wedge (-1 > 0)) \\
 \equiv & ((x > 2) \wedge \text{True}) \vee (\neg(x > 2) \wedge \text{False}) \\
 \equiv & x > 2
 \end{aligned}$$

(Again the final step can be confirmed via truth table.)

Why The Rules are Equivalent

All that has changed is the form of the proposition. Rather than

$$(b \rightarrow p) \wedge (\neg b \rightarrow q)$$

we have

$$(b \wedge p) \vee (\neg b \wedge q) :$$

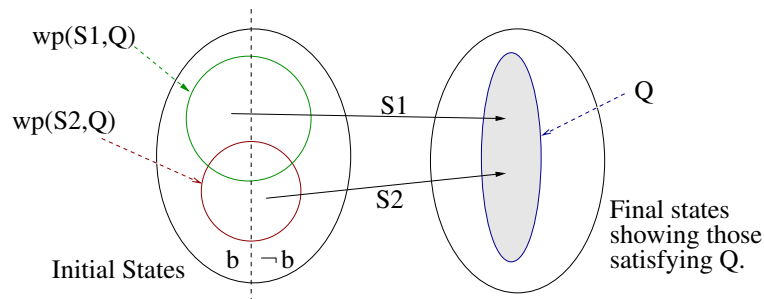
b	p	q	$(b \rightarrow p)$	\wedge	$(\neg b \rightarrow q)$	$(b \wedge p)$	\vee	$(\neg b \wedge q)$
T	T	T	T	T	T	T	T	F
T	T	F	T	T	T	T	T	F
T	F	T	F	F	T	F	F	F
T	F	F	F	F	T	F	F	F
F	T	T	T	T	T	F	T	T
F	T	F	T	F	F	F	F	F
F	F	T	T	T	T	F	T	T
F	F	F	T	F	F	F	F	F

A Diagrammatic View of Conditionals

The validity of

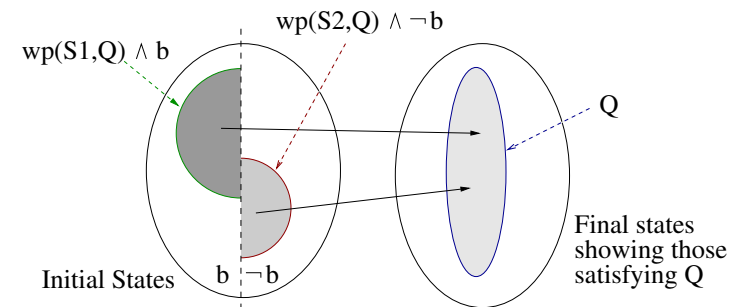
$$wp(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) \equiv (b \Rightarrow (wp(S_1, Q))) \wedge (\neg b \Rightarrow wp(S_2, Q))$$

can be seen from this diagram.



Another View

More suggestive of $(b \wedge wp(S_1, Q)) \vee (\neg b \wedge wp(S_2, Q))$?



Loops

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

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...

The Halting Problem

In week 7 we learned that determining if a program terminates or not is in general an *undecidable* problem.

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

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