Dijkstra's weakest precondition calculus (WP)

A formal system for reasoning about program correctness, extending Hoare logic

Weakest Precondition

- Hoare logic provides rules to verify {P} S {Q} for given P and Q.
- Weakest precondition *calculus* computes the minimal P required for a given S and Q, i.e., starts from {?} S {Q}
- For a program S and postcondition Q, wp(S, Q) is the weakest condition such that executing S from any state satisfying wp(S, Q) **terminates** in a state satisfying Q.
- Example (already seen):
  For $S \equiv x := y+1$ and $Q \equiv x > 5$, then $wp(S, Q) \equiv y+1 > 5$ (i.e., $y > 4$).
- wp(S, ·) is a *predicate transformer*: it maps postconditions to preconditions. Dijkstra's approach treats programs as functions operating on predicates rather than states.
- Think of WP as "working backward" from Q through S to derive the minimal requirements on the initial state. Often we know only the postcondition and we want to deduce the precondition.

Why WP calculus matters?

- Unlike Hoare logic, it provides a calculational method for correctness (not just verification).
- Adopted by theorem assistants for proving program correctness (e.g., Why3).
- Foundation for modern program analysis (e.g., abstract interpretation, static analysis).
- Key to Dijkstra’s "correct-by-construction" philosophy.
- If you know Hoare logic,

Wp Rules

- **Assignment**: wp(x := e, Q) = Q[x ↦ e] (substitute x with e in Q).
- **Sequence**: wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q)).
- **Conditional**: wp(if B then S_1 else S_2, Q) = (B ⇒ wp(S_1, Q)) ∧ (¬B ⇒ wp(S_2, Q)).
- **Loops**: For while B do S, the weakest precondition is the loop invariant (hardest part)+loop variant. Dijkstra uses a predicate transformer approach, defining it as the ***least fixed point*** satisfying:

  wp(while B do S, Q) = (¬B ⇒ Q) ∧ (B ⇒ wp(S, wp(while B do S, Q))).

  Formally, wp(while B do S, Q) is the strongest solution of the equation:

  Z = ((¬B ⇒ Q) v (B ⇒ wp(S, Z))

Example: {?} S {Q}

$S \equiv$ if $(x > 0)$ then $y := x$; else $y := -x$;
$Q \equiv y \geq 0$

wp(S, Q) =
$= (x > 0 \Rightarrow wp(y := x, y \geq 0)) \;\wedge\; (x \leq 0 \Rightarrow wp(y := -x, y \geq 0))$

$= (x > 0 \Rightarrow x \geq 0) \;\wedge\; (x \leq 0 \Rightarrow -x \geq 0)$

= true $\wedge$ true (since both implications hold for all x)

= true.

- The postcondition $y \geq 0$ is always satisfied

Intuition for loops

- In Hoare calculus, for while B do S, we need an invariant I such that:
- $I \wedge \neg B \Rightarrow Q$ (loop exit implies postcondition),
- $I \wedge B \Rightarrow wp(S, I)$ (each iteration preserves the invariant).
- Dijkstra’s approach generalizes this via fixed-point theory, but practical use often relies on manually proposing invariants.

Example: a simple loop

- wp(while $x > 0$ do $x := x - 1$, $x \leq 0$) =
  ($!x > 0 \Rightarrow x \leq 0$) $\wedge$ ($x > 0 \Rightarrow$ wp($x := x - 1$, wp(while $x > 0$ do $x := x - 1$, $x \leq 0$))) =
  ($x > 0 \Rightarrow$ wp($x := x - 1$, wp(while $x > 0$ do $x := x - 1$, $x \leq 0$)))
- Least fixed-point computation (assume x is an Integer):
  Guess of the weakest precondition at 0th iteration: wp_0 = **true** (weakest possible condition).
- Substitute wp_0 into the equation
  wp_1 = ($x > 0 \Rightarrow$ wp($x := x - 1$, true)) = ($x > 0 \Rightarrow$ true) = true
- Substitute wp_1 = true again:
  wp_2 = ($x > 0 \Rightarrow$ wp($x := x - 1$, true)) = true.
- Still no change. It's clear that iterating the loop nothing will change: we found the least fixed point.
  So the precondition of this loop is just **true** (which is also an invariant)
- In more complex cases we need to find a good invariant

A more complex example, with Hoare

```
i := 0; s:=0;
while (i < n)
  s := s + i;
  i := i + 1
```

{$s = 0 + 1 + 2 + ... + (n-1)= n(n-1)/2$}

Hoare stle: Possible invariant: $I \equiv s = 0 + 1 + ... + (i-1) \land i \le n$
At loop exit, satisfies the post ($i \le n \land !(i<n) \equiv i=n$).

Easy to see that is invariant during the loop: assume $I \land i < n$ holdsand check it at the end:
$s' = s + i = 0 + 1 + ... + (i-1) + i$,
$i' = i + 1$.
hence at the end of the iteration $s' = s + i = 0 + 1 + ... + i' -1$, and $i < n \Rightarrow i' \le n$.

We can now find the wp of the program:
wp(i:=0 , I): $0 \le n \land s = 0$
wp(s:=0, $0 \le n \land s = 0$) = $\mathbf{0 \le n}$

To prove also termination, just define the variant $v = n-i$ (the variant is a well-founded relation). The proof is very easy

Find the wp

s:=0; i := 0;
while (i < n)
  s := s + i;
  i := i + 1
{Q}   with Q ≡ s = 0 + 1 + 2 + ... + (n-1)

Invariant: I ≡ i ≤ n ∧ s = 0 + 1 + ... + (i-1)
Let's try wp(while (i < n) ..., Q) = I (we proved I =>Q and I is an invariant)

Then,
wp(i:=0 , I):   0 ≤ n ∧ s = 0
wp(s:=0, 0 ≤ n ∧ s = 0) = **0 ≤ n**

The wp (n ≥ 0) ensures the postcondition holds.

With fixed point?

$\mathbf{wp(while\ i < n\ do\ S, Q) = (i \geq n \Rightarrow Q) \wedge (i < n \Rightarrow wp(S, wp(while\ i < n\ do\ S, Q)))}$

Start with the weakest possible precondition (top of the lattice): wp_0 = true and replace it the equation

$\mathbf{wp_1 = (i \geq n \Rightarrow Q) \wedge (i < n \Rightarrow wp(S, true)).}$

but wp(S, true) = wp(s := s + i; i := i + 1, true) = true, hence:

$\mathbf{wp_1 = (i \geq n \Rightarrow Q) \wedge (i < n \Rightarrow true) = (i \geq n \Rightarrow Q) \wedge true = (i \geq n \Rightarrow Q) .}$

$\mathbf{wp_2 = (i \geq n \Rightarrow Q) \wedge (i < n \Rightarrow wp(S, (i \geq n \Rightarrow Q)).}$

but $wp(S, (i \geq n \Rightarrow Q))$ is $wp(S, (i \geq n \Rightarrow s = n(n-1)/2))$

$= wp(s := s + i, wp(i := i + 1, (i \geq n \Rightarrow s = n(n-1)/2)))$

$= wp(s := s + i, (i + 1 \geq n \Rightarrow s = n(n-1)/2))$

$= (i + 1 \geq n \Rightarrow s + i = n(n-1)/2).$

Thus $\mathbf{wp_2 = (i \geq n \Rightarrow s = n(n-1)/2) \wedge (i < n \Rightarrow (i + 1 \geq n \Rightarrow s + i = n(n-1)/2)).}$

But the second conjunct is just $i = n - 1 \Rightarrow s + i = n(n-1)/2$), i.e., $i = n - 1 \Rightarrow s=1+2+\text{--}+n-1-(n-1)= 1+2+..+n-2$.

It is clear that at each iteration the expression of s decreases of n-2, then of n-3, etc., until n-i, with $i \leq n$.

Thus the fixed-point converges to $wp = (i \leq n) \wedge (s = 0 + 1 + ... + (i-1))$ (which was the same I we found before, hence I is also the least fixed point), and by retropropagating it we get again the wp $n \geq 0$ of the program.

Termination is guaranteed since reaching a fixed-point implies having a variant function.