

Strongest Postcondition Calculus

Andrei Arusoaie

Hoare Logic - quick recap

- ▶ C. A. R. Hoare, 1969:
<https://dl.acm.org/doi/pdf/10.1145/363235.363259>

Hoare Logic - quick recap

- ▶ C. A. R. Hoare, 1969:
<https://dl.acm.org/doi/pdf/10.1145/363235.363259>
- ▶ Hoare's work was inspired R.W.Floyd's work on *Assigning Meanings to Programs*

Hoare Logic - quick recap

- ▶ C. A. R. Hoare, 1969:
<https://dl.acm.org/doi/pdf/10.1145/363235.363259>
- ▶ Hoare's work was inspired R.W.Floyd's work on *Assigning Meanings to Programs*
- ▶ Robert Willoughby Floyd, 1967:
<https://people.eecs.berkeley.edu/~necula/Papers/FloydMeaning.pdf>

Hoare Logic - quick recap

- ▶ C. A. R. Hoare, 1969:
<https://dl.acm.org/doi/pdf/10.1145/363235.363259>
- ▶ Hoare's work was inspired R.W.Floyd's work on *Assigning Meanings to Programs*
- ▶ Robert Willoughby Floyd, 1967:
<https://people.eecs.berkeley.edu/~necula/Papers/FloydMeaning.pdf>
- ▶ However, there are some differences in their approaches

Summary of Hoare Logic Proof Rules

Assignment

$$\frac{}{\vdash \{Q[e/x]\} x := e \{Q\}}$$

Precondition Strengthening

$$\frac{\vdash \{P'\} S \{Q\} \quad P \rightarrow P'}{\vdash \{P\} S \{Q\}}$$

Postcondition Weakening

$$\frac{\vdash \{P\} S \{Q'\} \quad Q' \rightarrow Q}{\vdash \{P\} S \{Q\}}$$

Composition

$$\frac{\vdash \{P\} S_1 \{Q\} \quad \vdash \{Q\} S_2 \{R\}}{\vdash \{P\} S_1; S_2 \{R\}}$$

If

$$\frac{\vdash \{P \wedge C\} S_1 \{Q\} \quad \vdash \{P \wedge \neg C\} S_2 \{Q\}}{\vdash \{P\} \text{if } C \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

While

$$\frac{\vdash \{I \wedge C\} S \{I\}}{\vdash \{I\} \text{while } C \text{ do } S \text{ end } \{I \wedge \neg C\}}$$

Floyd's approach

Assignment

$$\frac{}{\vdash \{P\} x := e \{ \exists v. (x = e[v/x]) \wedge P[v/x] \}}$$

Precondition Strengthening

$$\frac{\vdash \{P'\} S \{Q\} \quad P \rightarrow P'}{\vdash \{P\} S \{Q\}}$$

Postcondition Weakening

$$\frac{\vdash \{P\} S \{Q'\} \quad Q' \rightarrow Q}{\vdash \{P\} S \{Q\}}$$

Composition

$$\frac{\vdash \{P\} S_1 \{Q\} \quad \vdash \{Q\} S_2 \{R\}}{\vdash \{P\} S_1; S_2 \{R\}}$$

If

$$\frac{\vdash \{P \wedge C\} S_1 \{Q\} \quad \vdash \{P \wedge \neg C\} S_2 \{Q\}}{\vdash \{P\} \text{if } C \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

While

$$\frac{\vdash \{I \wedge C\} S \{I\}}{\vdash \{I\} \text{while } C \text{ do } S \text{ end } \{I \wedge \neg C\}}$$

Assignment has existential quantifiers in Floyd's approach

$$\frac{}{\vdash \{P\} \ x := e \ \{ \exists v. (x = e[v/x]) \wedge P[v/x] \}}$$

The existentially quantified value v corresponds to the old value of x . Therefore, after the assignment, we replace in e the value v for x , and we also keep the information that P holds for v .

Assignment has existential quantifiers in Floyd's approach

$$\frac{}{\vdash \{P\} \ x := e \ \{ \exists v. (x = e[v/x]) \wedge P[v/x] \}}$$

The existentially quantified value v corresponds to the old value of x . Therefore, after the assignment, we replace in e the value v for x , and we also keep the information that P holds for v .

Example:

$$\vdash \{s = 0\} \ s := s + i \ \{ \exists v. (s = (s+i)[v/s]) \wedge (s = 0)[v/s] \}$$

Assignment has existential quantifiers in Floyd's approach

$$\frac{}{\vdash \{P\} x := e \{ \exists v. (x = e[v/x]) \wedge P[v/x] \}}$$

The existentially quantified value v corresponds to the old value of x . Therefore, after the assignment, we replace in e the value v for x , and we also keep the information that P holds for v .

Example:

$$\vdash \{s = 0\} s := s + i \{ \exists v. (s = (s+i)[v/s]) \wedge (s = 0)[v/s] \}$$

After applying substitutions, we get:

$$\vdash \{s = 0\} s := s + i \{ \exists v. (s = (v+i)) \wedge (v = 0) \}$$

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \quad s \quad := \quad s + i \quad \{\exists v. (s = (v + i)) \wedge (v = 0)\}$$

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \ s \ := \ s + i \ \{ \exists v. (s = (v + i)) \wedge (v = 0) \}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \ s \ := \ s + i \ \{s = 0 + i\}$$

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \ s := s + i \ \{ \exists v. (s = (v + i)) \wedge (v = 0) \}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \ s := s + i \ \{s = 0 + i\}$$

$\{s = 0 + i\}$ is the *strongest postcondition* after the assignment.

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \ s \ := \ s + i \ \{ \exists v. (s = (v+i)) \wedge (v = 0) \}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \ s \ := \ s + i \ \{s = 0 + i\}$$

$\{s = 0 + i\}$ is the *strongest postcondition* after the assignment.

Using Hoare's rule, with postcondition $\{s = 0 + i\}$ we obtain the same precondition:

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \quad s \quad := \quad s + i \quad \{\exists v. (s = (v+i)) \wedge (v = 0)\}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \quad s \quad := \quad s + i \quad \{s = 0 + i\}$$

$\{s = 0 + i\}$ is the *strongest postcondition* after the assignment.

Using Hoare's rule, with postcondition $\{s = 0 + i\}$ we obtain the same precondition:

$$\vdash \{s = (0 + i)[s + i/s]\} \quad s \quad := \quad s + i \quad \{s = 0 + i\}$$

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \ s := s + i \ \{ \exists v. (s = (v + i)) \wedge (v = 0) \}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \ s := s + i \ \{s = 0 + i\}$$

$\{s = 0 + i\}$ is the *strongest postcondition* after the assignment.

Using Hoare's rule, with postcondition $\{s = 0 + i\}$ we obtain the same precondition:

$$\vdash \{s = (0 + i)[s + i/s]\} \ s := s + i \ \{s = 0 + i\}$$

$$\vdash \{s + i = 0 + i\} \ s := s + i \ \{s = 0 + i\}$$

Example – continued

We obtain an existentially quantified postcondition:

$$\vdash \{s = 0\} \text{ s } := \text{ s + i } \{ \exists v. (s = (v + i)) \wedge (v = 0) \}$$

Note: in this example, this can be further simplified, by substituting 0 for v , and eliminate the quantifier (this is not always possible!):

$$\vdash \{s = 0\} \text{ s } := \text{ s + i } \{s = 0 + i\}$$

$\{s = 0 + i\}$ is the *strongest postcondition* after the assignment.

Using Hoare's rule, with postcondition $\{s = 0 + i\}$ we obtain the same precondition:

$$\vdash \{s = (0 + i)[s + i/s]\} \text{ s } := \text{ s + i } \{s = 0 + i\}$$

$$\vdash \{s + i = 0 + i\} \text{ s } := \text{ s + i } \{s = 0 + i\}$$

$$\vdash \{s = 0\} \text{ s } := \text{ s + i } \{s = 0 + i\}$$

From Floyd's assignment rule to strongest postconditions

- ▶ It is not yet clear who introduced the concept of strongest postconditions, but Floyd discusses a related notion of “strongest verifiable consequents” in his 1967 paper.

From Floyd's assignment rule to strongest postconditions

- ▶ It is not yet clear who introduced the concept of strongest postconditions, but Floyd discusses a related notion of “strongest verifiable consequents” in his 1967 paper.
- ▶ Idea: compute a postcondition predicate for a statement S by transforming the precondition P .
- ▶ The result is denoted $sp(S, P)$ – the *strongest* postcondition which holds after executing S in a state satisfying P .

From Floyd's assignment rule to strongest postconditions

- ▶ It is not yet clear who introduced the concept of strongest postconditions, but Floyd discusses a related notion of “strongest verifiable consequents” in his 1967 paper.
- ▶ Idea: compute a postcondition predicate for a statement S by transforming the precondition P .
- ▶ The result is denoted $sp(S, P)$ – the *strongest* postcondition which holds after executing S in a state satisfying P .
- ▶ $sp(S, P)$ is strongest in the sense that any other postcondition Q for a valid triple $\{P\} S \{Q\}$ is implied by $sp(S, P)$.

sp for Assignment

The strongest postcondition calculus for assignment is:

$$\text{sp}(x := e, P) = \exists v. (x = e[v/x]) \wedge P[v/x]$$

Property: $\{P\}x := e\{Q\}$ holds iff $\text{sp}(x := e, P) \rightarrow Q$.

sp for Assignment

The strongest postcondition calculus for assignment is:

$$\text{sp}(x := e, P) = \exists v. (x = e[v/x]) \wedge P[v/x]$$

Property: $\{P\}x := e\{Q\}$ holds iff $\text{sp}(x := e, P) \rightarrow Q$.

Recall the weakest precondition definition for assignment:

$$\text{wp}(x := e, Q) = Q[e/x]$$

Property: $\{P\}x := e\{Q\}$ holds iff $P \rightarrow \text{wp}(x := e, Q)$.

Example

Is $\{s = 0\} s := s + i \{s \geq i\}$ valid?

Example

Is $\{s = 0\} s := s + i \{s \geq i\}$ valid?

First, we compute $sp(s := s + i, s = 0)$:

Example

Is $\{s = 0\} s := s + i \{s \geq i\}$ valid?

First, we compute $sp(s := s + i, s = 0)$:

$$\begin{aligned} sp(s := s + i, s = 0) &= \exists v. (s = (s + i)[v/s]) \wedge (s = 0)[v/s] \\ &= \exists v. (s = v + i) \wedge (v = 0) \\ &\equiv \exists v. s = 0 + i \equiv s = 0 + i \equiv s = i. \end{aligned}$$

Example

Is $\{s = 0\} s := s + i \{s \geq i\}$ valid?

First, we compute $sp(s := s + i, s = 0)$:

$$\begin{aligned} sp(s := s + i, s = 0) &= \exists v. (s = (s + i)[v/s]) \wedge (s = 0)[v/s] \\ &= \exists v. (s = v + i) \wedge (v = 0) \\ &\equiv \exists v. s = 0 + i \equiv s = 0 + i \equiv s = i. \end{aligned}$$

Finally, we check whether $sp(s := s + i, s = 0) \rightarrow s \geq i$, that is:

$$s = i \rightarrow s \geq i.$$

Example

Is $\{s = 0\} s := s + i \{s \geq i\}$ valid?

First, we compute $sp(s := s + i, s = 0)$:

$$\begin{aligned} sp(s := s + i, s = 0) &= \exists v. (s = (s + i)[v/s]) \wedge (s = 0)[v/s] \\ &= \exists v. (s = v + i) \wedge (v = 0) \\ &\equiv \exists v. s = 0 + i \equiv s = 0 + i \equiv s = i. \end{aligned}$$

Finally, we check whether $sp(s := s + i, s = 0) \rightarrow s \geq i$, that is:

$$s = i \rightarrow s \geq i.$$

Since this is true, we have $\{s = 0\} s := s + i \{s \geq i\}$ valid.

sp for Sequences

The definition of sp for sequences is:

$$sp(S_1; S_2, P) = sp(S_2, sp(S_1, P))$$

Note: $sp(S_1, P)$ serves as precondition for S_2 .

sp for Sequences

The definition of sp for sequences is:

$$sp(S_1; S_2, P) = sp(S_2, sp(S_1, P))$$

Note: $sp(S_1, P)$ serves as precondition for S_2 .

In contrast, the weakest precondition definition for sequences is:

$$wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$$

Note: $wp(S_2, Q)$ serves as postcondition for S_1 .

Example

We want to compute: $\text{sp}(i := i + 1; s := s + i, i = 0 \wedge s = 0)$.

Example

We want to compute: $\text{sp}(i := i + 1; s := s + i, i = 0 \wedge s = 0)$.

Step 1:

$$\begin{aligned}\text{sp}(i := i + 1, i = 0 \wedge s = 0) &= \exists v. (i = (i + 1)[v/i]) \wedge (i = 0 \wedge s = 0)[v/i] \\ &= \exists v. (i = v + 1) \wedge (v = 0 \wedge s = 0)\end{aligned}$$

Example

We want to compute: $\text{sp}(i := i + 1; s := s + i, i = 0 \wedge s = 0)$.

Step 1:

$$\begin{aligned}\text{sp}(i := i + 1, i = 0 \wedge s = 0) &= \exists v. (i = (i + 1)[v/i]) \wedge (i = 0 \wedge s = 0)[v/i] \\ &= \exists v. (i = v + 1) \wedge (v = 0 \wedge s = 0)\end{aligned}$$

Step 2:

$$\begin{aligned}\text{sp}(s := s + i, \exists v. (i = v + 1) \wedge (v = 0 \wedge s = 0)) &= \\ = \exists v'. (s := (s + i)[v'/s]) \wedge (\exists v. (i = v + 1) \wedge (v = 0 \wedge s = 0))[v'/s] \\ = \exists v'. (s := v' + i) \wedge (\exists v. (i = v + 1) \wedge (v = 0 \wedge v' = 0)) \\ \equiv \exists v. \exists v'. (s := v' + i) \wedge (i = v + 1) \wedge (v = 0 \wedge v' = 0) \\ \equiv \exists v. \exists v'. (s := 0 + i) \wedge (i = 0 + 1) \wedge (v = 0 \wedge v' = 0) \\ \equiv s = i \wedge i = 1 \equiv s = 1 \wedge i = 1.\end{aligned}$$

Skip

The definition of sp for `skip` is:

$$sp(\text{skip}, P) = P$$

Skip

The definition of sp for `skip` is:

$$sp(\text{skip}, P) = P$$

Also recall that for wp we have

$$wp(\text{skip}, Q) = Q$$

Conditional Statement

The definition of sp for the `if-then-else` is:

$$sp(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = sp(S_1, P \wedge C) \vee sp(S_2, P \wedge \neg C)$$

Conditional Statement

The definition of sp for the `if-then-else` is:

$$sp(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = sp(S_1, P \wedge C) \vee sp(S_2, P \wedge \neg C)$$

Recall that wlp for `if-then-else` is computed as:

$$wlp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = (C \rightarrow wlp(S_1, Q)) \wedge (\neg C \rightarrow wlp(S_2, Q))$$

Example

We compute $sp(\text{if } x < 0 \text{ then } m := -x \text{ else } m := x, \text{true})$, using

$$sp(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = sp(S_1, P \wedge C) \vee sp(S_2, P \wedge \neg C)$$

Example

We compute $sp(\text{if } x < 0 \text{ then } m := -x \text{ else } m := x, \text{true})$, using

$$sp(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = sp(S_1, P \wedge C) \vee sp(S_2, P \wedge \neg C)$$

► First, we compute for $S_1 : m := -x$ with $C : x < 0$:

$$\begin{aligned} sp(m := -x, \text{true} \wedge (x < 0)) &= \\ &= \exists v. m = (-x)[v/m] \wedge (\text{true} \wedge (x < 0))[v/m] \\ &= \exists v. m = -x \wedge (\text{true} \wedge (x < 0)) \equiv m = -x \wedge x < 0. \end{aligned}$$

Example

We compute $sp(\text{if } x < 0 \text{ then } m := -x \text{ else } m := x, \text{true})$, using

$$sp(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = sp(S_1, P \wedge C) \vee sp(S_2, P \wedge \neg C)$$

- First, we compute for $S_1 : m := -x$ with $C : x < 0$:

$$\begin{aligned} sp(m := -x, \text{true} \wedge (x < 0)) &= \\ &= \exists v. m = (-x)[v/m] \wedge (\text{true} \wedge (x < 0))[v/m] \\ &= \exists v. m = -x \wedge (\text{true} \wedge (x < 0)) \equiv m = -x \wedge x < 0. \end{aligned}$$

- Then, we compute for $S_2 : m := x$ with $\neg C : x \geq 0$:

$$\begin{aligned} sp(m := x, \text{true} \wedge \neg(x < 0)) &= \\ &= \exists v. m = (x)[v/m] \wedge (\text{true} \wedge \neg(x < 0))[v/m] \\ &= \exists v. m = x \wedge (\text{true} \wedge \neg(x < 0)) \equiv m = x \wedge x \geq 0. \end{aligned}$$

Example – continued

So:

- ▶ $sp(m := -x, \text{true} \wedge (x < 0)) = m = -x \wedge x < 0$
- ▶ $sp(m := x, \text{true} \wedge \neg(x < 0)) = m = x \wedge x \geq 0$

Example – continued

So:

- ▶ $sp(m := -x, \text{true} \wedge (x < 0)) = m = -x \wedge x < 0$
- ▶ $sp(m := x, \text{true} \wedge \neg(x < 0)) = m = x \wedge x \geq 0$

$$\begin{aligned} sp(\text{if } x < 0 \text{ then } m := -x \text{ else } m := x, \text{true}) &= \\ &= (m = -x \wedge x < 0) \vee (m = x \wedge x \geq 0) \end{aligned}$$

Loops

The definition of sp for the `while` loop is:

$$sp(\text{while } C \text{ do } S, P) = sp(\text{while } C \text{ do } S, sp(S, P \wedge C)) \vee (P \wedge \neg C)$$

Note: strongest postcondition calculus only makes sense for partial correctness, since it captures the changes made to the program state after executing a statement.

Loops

The definition of sp for the `while` loop is:

$$sp(\text{while } C \text{ do } S, P) = sp(\text{while } C \text{ do } S, sp(S, P \wedge C)) \vee (P \wedge \neg C)$$

Note: strongest postcondition calculus only makes sense for partial correctness, since it captures the changes made to the program state after executing a statement.

The definition of wlp for loops uses the invariants:

$$wlp(\text{while } C \text{ inv: } I \text{ do } S, Q) = I \wedge \forall x_1, \dots, x_k. \left(((C \wedge I) \rightarrow wp(S, I)) \wedge ((\neg C \wedge I) \rightarrow Q) \right) [x_i/w_i],$$

where w_1, \dots, w_k are variables modified in s , and x_1, \dots, x_k are fresh variables.

Example

We want to compute $\text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 0 \wedge n = 2)$.

$$\begin{aligned}\text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, n = 2) &= \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 0 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 0 \wedge n = 2 \wedge i > n)\end{aligned}$$

Example

We want to compute $\text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 0 \wedge n = 2)$.

$$\begin{aligned} \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, n = 2) &= \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 0 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 1 \wedge n = 2) \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 1 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

Example

We want to compute $\text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 0 \wedge n = 2)$.

$$\begin{aligned} \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, n = 2) &= \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 0 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 1 \wedge n = 2) \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 1 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 2 \wedge n = 2 \wedge (i \leq n)) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \\ &= \dots \end{aligned}$$

Example – continued

$$\begin{aligned} \dots &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ &\vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

Example – continued

$$\dots = \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ \vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n)$$

$$= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n))) \\ \vee (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n)$$

Example – continued

$$\dots = \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ \vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n)$$

$$= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n))) \\ \vee (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n)$$

$$\dots (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n)$$

Example – continued

$$\begin{aligned} \dots &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ &\vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} \dots &(i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

Note 1: each conjunction corresponds to a particular program path.

Example – continued

$$\begin{aligned} \dots &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ &\vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} \dots &(i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

Note 1: each conjunction corresponds to a particular program path.

Note 2: the last 3 conjunctions are false suggesting impossible executions.

Example – continued

$$\begin{aligned} \dots &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n)) \\ &\vee (i = 2 \wedge n = 2 \wedge i > n) \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} &= \text{sp}(\text{while } (i \leq n) \text{ do } i := i + 1, \text{sp}(i := i + 1, i = 3 \wedge n = 2 \wedge (i \leq n))) \\ &\quad \vee (i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

$$\begin{aligned} \dots &(i = 3 \wedge n = 2 \wedge i > n) \vee (i = 2 \wedge n = 2 \wedge i > n) \\ &\quad \vee (i = 1 \wedge n = 2 \wedge i > n) \vee (i = 0 \wedge n = 2 \wedge i > n) \end{aligned}$$

Note 1: each conjunction corresponds to a particular program path.

Note 2: the last 3 conjunctions are false suggesting impossible executions.

Note 3: at the end of the execution, the program state satisfies

$$i = 3 \wedge n = 2 \wedge i > n.$$

Summary of Strongest Postcondition Calculus

- *Assignment:*

$$\text{sp}(x := e, P) = \exists v. (x = e[v/x]) \wedge P[v/x]$$

- *Sequential Composition:*

$$\text{sp}(S_1; S_2, P) = \text{sp}(S_2, \text{sp}(S_1, P))$$

- *Skip:*

$$\text{sp}(\text{skip}, P) = P$$

- *Conditional:*

$$\text{sp}(\text{if } C \text{ then } S_1 \text{ else } S_2, P) = \text{sp}(S_1, P \wedge C) \vee \text{sp}(S_2, P \wedge \neg C)$$

- *While Loop:*

$$\text{sp}(\text{while } C \text{ do } S, Q) = \text{sp}(\text{while } C \text{ do } S, \text{sp}(S, P \wedge C)) \vee (P \wedge \neg C)$$

Properties

Theorem

A Hoare triple $\{P\} S \{Q\}$ is valid if and only if $\models \text{sp}(S, P) \rightarrow Q$.

Theorem

A Hoare triple $\{P\} S \{Q\}$ is valid if and only if $\models P \rightarrow \text{wp}(S, Q)$.

Remarks:

- ▶ $\models \text{sp}(S, P) \rightarrow Q$ iff $\{P\} S \{Q\}$ is valid iff $\models P \rightarrow \text{wp}(S, Q)$.
- ▶ wp is more appropriate for automated verification, as it supports loop invariants and can manage termination through variants.
- ▶ Strongest postcondition is often aligned with symbolic execution, and it is less suitable for verification tools.

Strongest Postcondition and Symbolic Execution

- ▶ Formally, $\text{sp}(S, P)$ is a condition satisfied by the program state resulted after S is executed.

Strongest Postcondition and Symbolic Execution

- ▶ Formally, $\text{sp}(S, P)$ is a condition satisfied by the program state resulted after S is executed.
- ▶ Symbolic execution corresponds to *symbolically* executing the program, that is, we run the program symbolic inputs (i.e., symbolic states) - described by predicates - instead of concrete inputs (i.e., concrete states).

Strongest Postcondition and Symbolic Execution

- ▶ Formally, $\text{sp}(S, P)$ is a condition satisfied by the program state resulted after S is executed.
- ▶ Symbolic execution corresponds to *symbolically* executing the program, that is, we run the program symbolic inputs (i.e., symbolic states) - described by predicates - instead of concrete inputs (i.e., concrete states).
- ▶ When computing strongest postconditions, we aim to modify the precondition s.t. it captures the changes made to the program states by the executed program.

Symbolic execution by example

```
read n;  
s = 0;  
while n > 0 do  
    s = s + n;  
    n = n - 1;  
end while;
```

Symbolic execution by example

Symbolic execution with symbolic input x

```
read n;  
s = 0;  
while  $n > 0$  do  
     $s = s + n$ ;  
     $n = n - 1$ ;  
end while;
```

Symbolic execution by example

Symbolic execution with symbolic input x

```
read n;  
s = 0;  
while  $n > 0$  do  
     $s = s + n$ ;  
     $n = n - 1$ ;  
end while;
```

► $x \leq 0$
state: $s = 0$

Symbolic execution by example

Symbolic execution with symbolic input x

```
read n;  
s = 0;  
while  $n > 0$  do  
     $s = s + n$ ;  
     $n = n - 1$ ;  
end while;
```

- ▶ $x \leq 0$
state: $s = 0$
- ▶ $x > 0 \wedge (x - 1) \leq 0$
state: $s = x$

Symbolic execution by example

Symbolic execution with symbolic input x

```
read n;  
s = 0;  
while  $n > 0$  do  
     $s = s + n$ ;  
     $n = n - 1$ ;  
end while;
```

- ▶ $x \leq 0$
state: $s = 0$
- ▶ $x > 0 \wedge (x - 1) \leq 0$
state: $s = x$
- ▶ $x > 0 \wedge (x - 1) > 0 \wedge (x - 2) \leq 0$
state: $s = x + (x - 1)$

Symbolic execution by example

Symbolic execution with symbolic input x

```
read n;  
s = 0;  
while  $n > 0$  do  
     $s = s + n$ ;  
     $n = n - 1$ ;  
end while;
```

- ▶ $x \leq 0$
state: $s = 0$
- ▶ $x > 0 \wedge (x - 1) \leq 0$
state: $s = x$
- ▶ $x > 0 \wedge (x - 1) > 0 \wedge (x - 2) \leq 0$
state: $s = x + (x - 1)$
- ▶ ...

Key concepts

- Symbolic values x, \dots

Key concepts

- Symbolic values
- Path condition

x, \dots

$x \leq 0, x > 0 \wedge x - 1 \leq 0, \dots$

Key concepts

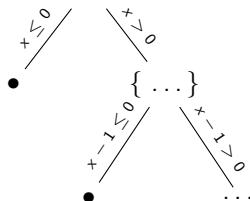
- Symbolic values
- Path condition

x, \dots

$x \leq 0, x > 0 \wedge x - 1 \leq 0, \dots$

```
read n;  
    |  $x \in \mathbb{Z}$   
s = 0;  
    |  $x \in \mathbb{Z}$   
while (n > 0) { ... }
```

- Symbolic execution tree



Strongest Postcondition and Symbolic Execution

- ▶ Symbolic execution is a technique that allows one to execute programs with symbolic values rather than concrete ones.

Strongest Postcondition and Symbolic Execution

- ▶ Symbolic execution is a technique that allows one to execute programs with symbolic values rather than concrete ones.
- ▶ It generates symbolic expressions representing the program's state and path conditions.

Strongest Postcondition and Symbolic Execution

- ▶ Symbolic execution is a technique that allows one to execute programs with symbolic values rather than concrete ones.
- ▶ It generates symbolic expressions representing the program's state and path conditions.
- ▶ The path conditions generated during symbolic execution represent the conditions under which each path is feasible.

Strongest Postcondition and Symbolic Execution

- ▶ Symbolic execution is a technique that allows one to execute programs with symbolic values rather than concrete ones.
- ▶ It generates symbolic expressions representing the program's state and path conditions.
- ▶ The path conditions generated during symbolic execution represent the conditions under which each path is feasible.
- ▶ The final symbolic state, combined with the path conditions, forms the strongest postcondition for the given program and initial condition.

Desired Properties and Difficulties

Symbolic execution returns path conditions instead of values.

Desired Properties and Difficulties

Symbolic execution returns path conditions instead of values.

Desired Properties:

- ▶ Coverage (soundness): each concrete execution is covered by a symbolic execution.
- ▶ Precision (completeness): each symbolic execution captures at least one concrete execution.

Desired Properties and Difficulties

Symbolic execution returns path conditions instead of values.

Desired Properties:

- ▶ Coverage (soundness): each concrete execution is covered by a symbolic execution.
- ▶ Precision (completeness): each symbolic execution captures at least one concrete execution.

Difficulties:

- ▶ the *path explosion* problem: the number of paths in a program grows exponentially.

Desired Properties and Difficulties

Symbolic execution returns path conditions instead of values.

Desired Properties:

- ▶ Coverage (soundness): each concrete execution is covered by a symbolic execution.
- ▶ Precision (completeness): each symbolic execution captures at least one concrete execution.

Difficulties:

- ▶ the *path explosion* problem: the number of paths in a program grows exponentially.
- ▶ requires various mechanisms to tackle the big number of states (e.g., eliminate some of them using automated reasoning, employ abstract interpretation, etc.).

Desired Properties and Difficulties

Symbolic execution returns path conditions instead of values.

Desired Properties:

- ▶ Coverage (soundness): each concrete execution is covered by a symbolic execution.
- ▶ Precision (completeness): each symbolic execution captures at least one concrete execution.

Difficulties:

- ▶ the *path explosion* problem: the number of paths in a program grows exponentially.
- ▶ requires various mechanisms to tackle the big number of states (e.g., eliminate some of them using automated reasoning, employ abstract interpretation, etc.).
- ▶ dealing with loops and recursion is a known problem.
- ▶ scalability and performance remain an issue in practice.

Tools based on Symbolic Execution

- ▶ **KLEE**: <https://klee.github.io/>
- ▶ **S2E (Selective Symbolic Execution)**: <https://s2e.systems/>
- ▶ **Angr**: <https://angr.io/>
- ▶ **SymCC**: <https://github.com/eurecom-s3/symcc>
- ▶ **Manticore**: <https://github.com/trailofbits/manticore>
- ▶ **Driller**: <https://github.com/shellphish/driller>