Weakest Precondition Calculus

– lecture notes –

October 17, 2025

Contents

1 Introduction **3**

2 Review of Hoare Logic **3**
2.1 Hoare Triples . . . 3
2.2 Partial vs. Total Correctness . . . 3
2.3 Proof Rules for Hoare Logic . . . 3
2.4 Soundness . . . 4

3 Motivation for Weakest Preconditions **4**
3.1 The Challenge of Automation . . . 4
3.2 IMP Language with Loop Invariants . . . 4

4 Verification Conditions **5**
4.1 The Verification Condition Approach . . . 5
4.2 Forward vs. Backward Methods . . . 5

5 Weakest Liberal Precondition (Partial Correctness) **5**
5.1 Definition and Intuition . . . 5
5.2 Soundness . . . 5
5.3 Computing WLP for Different Constructs . . . 6
5.3.1 Assignment . . . 6
5.3.2 Skip . . . 6
5.3.3 Sequential Composition . . . 6
5.3.4 Conditional Statement . . . 6
5.3.5 While Loop . . . 7

6 Soundness and Completeness of WLP **7**
6.1 Soundness . . . 7
6.2 WLP Property . . . 8

7 Total Correctness and Weakest Precondition **8**
7.1 The Challenge of Termination . . . 8
7.2 Loop Variants . . . 8
7.3 Modified Loop Rule for Total Correctness . . . 8
7.4 IMP with Variants . . . 8

8 Weakest (Strict) Precondition **8**
8.1 Definition . . . 8
8.2 Computing WP . . . 9
8.3 Extended Example . . . 9

9 Soundness Results for WP **10**

10 Summary **10**
10.1 Key Distinctions 10
10.2 Advantages of the WP Calculus 10
10.3 Limitations 10

11 Further Reading **11**

The content of this document is not claimed to be original or to be published as original research. It is meant to serve as a learning resource for students.

1 Introduction

These notes provide an introduction to weakest precondition calculus, a fundamental technique in program verification. We begin with a brief reminder of Hoare logic before introducing the concept of weakest preconditions and showing how they enable automated verification of program correctness.

2 Review of Hoare Logic

2.1 Hoare Triples

Hoare logic, introduced by C. A. R. Hoare in 1969, provides a formal system for reasoning about program correctness. The fundamental construct is the *Hoare triple*:

$$\{P\}\, S\, \{Q\}$$

where:

- P is the *precondition* (a logical formula describing the state before execution)
- S is a program statement
- Q is the *postcondition* (a logical formula describing the state after execution)

2.2 Partial vs. Total Correctness

Hoare logic can express two kinds of correctness:

Definition 2.1 (Partial Correctness)**.** *A triple $\{P\}S\{Q\}$ is valid for partial correctness, written $\models \{P\}S\{Q\}$, if: whenever S is executed in a state satisfying P and the execution terminates, then the resulting state satisfies Q.*

Definition 2.2 (Total Correctness)**.** *A triple $[P]S[Q]$ is valid for total correctness, written $\models [P]S[Q]$, if: whenever S is executed in a state satisfying P, then the execution terminates and the resulting state satisfies Q.*

The major difference is that total correctness requires *termination*.

2.3 Proof Rules for Hoare Logic

The following are the standard proof rules for Hoare logic:

Assignment:

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

Precondition Strengthening:

$$\frac{\vdash \{P'\}\, S\, \{Q\} \quad P \to P'}{\vdash \{P\}\, S\, \{Q\}}$$

Postcondition Weakening:

$$\frac{\vdash \{P\}\, S\, \{Q'\} \quad Q' \to Q}{\vdash \{P\}\, S\, \{Q\}}$$

Sequential Composition:

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

Conditional:

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

While Loop:

$$\frac{\vdash \{I \wedge C\}\, S\, \{I\}}{\vdash \{I\}\, \texttt{while}\ C\ \texttt{do}\ S\, \{I \wedge \neg C\}}$$

2.4 Soundness

Theorem 2.3 (Soundness of Hoare Logic)**.** *If* $\vdash \{P\}S\{Q\}$*, then* $\models \{P\}S\{Q\}$*.*

However, Hoare logic is not complete: there exist valid triples $\models \{P\}S\{Q\}$ that cannot be derived using the proof rules.

3 Motivation for Weakest Preconditions

3.1 The Challenge of Automation

Manual construction of Hoare logic proofs is tedious and error-prone. Several aspects can be automated:

- **Assignment preconditions:** Yes, can be computed automatically
- **Loop invariants:** No, generally require human insight
- **Other proof steps:** Yes, if invariants are provided

The weakest precondition calculus provides a systematic approach to automation by working backwards from postconditions.

3.2 IMP Language with Loop Invariants

We work with an extended version of the IMP language that includes invariant annotations:

Arithmetic Expressions:

$$\text{AExp} ::= \text{Var} \mid \text{Int} \mid \text{AExp} + \text{AExp} \mid \text{AExp}/\text{AExp}$$

Boolean Expressions:

$$\text{BExp} ::= \texttt{true} \mid \texttt{false} \mid \text{AExp} < \text{AExp} \mid \texttt{not}\ \text{BExp} \mid \text{BExp}\ \texttt{and}\ \text{BExp}$$

Statements:

$$
\begin{aligned}
\text{Stmt} ::= \ & \text{Var} := \text{AExp} \\
& \mid \texttt{if}\ \text{BExp}\ \texttt{then}\ \text{Stmt}\ \texttt{else}\ \text{Stmt} \\
& \mid \texttt{while}\ \text{BExp}\ \texttt{inv:}\ \ \varphi\ \texttt{do}\ \text{Stmt}\ \texttt{end} \\
& \mid \text{Stmt}; \text{Stmt} \\
& \mid \texttt{skip}
\end{aligned}
$$

The addition (w.r.t to the initial syntax of IMP) is the `inv:` annotation in while loops.

4 Verification Conditions

4.1 The Verification Condition Approach

Automated verification proceeds in two steps:

1. **Generate verification conditions (VCs):** These are logical formulas whose validity implies program correctness
2. **Prove the VCs:** Use automated theorem provers (SMT solvers, etc.)

4.2 Forward vs. Backward Methods

There are two main approaches to generating verification conditions:

Forward (Strongest Postcondition): Start from the precondition and propagate forward through the program to derive conditions that imply the postcondition.

Backward (Weakest Precondition): Start from the postcondition and work backward through the program to compute the weakest precondition.

These notes focus on the backward method.

5 Weakest Liberal Precondition (Partial Correctness)

5.1 Definition and Intuition

Definition 5.1 (Weakest Liberal Precondition)**.** *The* weakest liberal precondition $wlp(S, Q)$ *for a statement S and postcondition Q is a formula such that:*

1. *If $wlp(S, Q)$ holds before executing S and S terminates, then Q holds after execution*
2. *$wlp(S, Q)$ is the* weakest *such formula (i.e., any other valid precondition implies it)*

The term "liberal" indicates that we do not require termination—this corresponds to partial correctness.

5.2 Soundness

Theorem 5.2. *The sequent $\vdash \{P\}S\{Q\}$ is valid if and only if $\models P \to wlp(S, Q)$.*

This theorem reduces the problem of proving a Hoare triple to proving a logical implication.

5.3 Computing WLP for Different Constructs

5.3.1 Assignment

For an assignment statement $x := e$:

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

This means we substitute e for x in the postcondition Q.

Example 5.3. *Let $Q \equiv i \leq n \land 2 \cdot s = i \cdot (i+1)$. Then:*

$$wlp(s := s + i, Q) = i \leq n \land 2 \cdot (s + i) = i \cdot (i + 1)$$

Important clarification: *The formula $2 \cdot (s+i) = i \cdot (i+1)$ must hold* before *the assignment. After executing $s := s+i$, the variable s has the new value $s+i$, so the postcondition $2 \cdot s = i \cdot (i+1)$ is satisfied with this new value.*

Example 5.4. *Continuing the previous example, let $Q' \equiv i \leq n \land 2 \cdot (s + i) = i \cdot (i + 1)$. Then:*

$$\begin{aligned} wlp(i := i + 1, Q') &= (i + 1) \leq n \land 2 \cdot (s + (i + 1)) \\ &= (i + 1) \cdot ((i + 1) + 1) \end{aligned}$$

5.3.2 Skip

For the skip statement:

$$\text{wlp}(\texttt{skip}, Q) = Q$$

Since skip does nothing, the precondition equals the postcondition.

5.3.3 Sequential Composition

For sequential composition $S_1; S_2$:

$$\text{wlp}(S_1; S_2, Q) = \text{wlp}(S_1, \text{wlp}(S_2, Q))$$

We work backwards: first compute the weakest precondition for S_2, then use that as the postcondition for S_1.

Example 5.5. *Compute $wlp(i := i + 1; s := s + i, Q)$ where $Q \equiv i \leq n \land 2 \cdot s = i \cdot (i + 1)$.*

Step 1: *Calculate $wlp(s := s + i, Q)$:*

$$Q' = wlp(s := s + i, Q) = i \leq n \land 2 \cdot (s + i) = i \cdot (i + 1)$$

Step 2: *Calculate $wlp(i := i + 1, Q')$:*

$$\begin{aligned} wlp(i := i + 1, Q') &= (i + 1) \leq n \land 2 \cdot (s + (i + 1)) \\ &= (i + 1) \cdot ((i + 1) + 1) \end{aligned}$$

5.3.4 Conditional Statement

For a conditional $\texttt{if } C \texttt{ then } S_1 \texttt{ else } S_2$:

$$\text{wlp}(\texttt{if } C \texttt{ then } S_1 \texttt{ else } S_2, Q) = (C \to \text{wlp}(S_1, Q)) \land (\neg C \to \text{wlp}(S_2, Q))$$

Example 5.6. *Compute* $wlp(\texttt{if}\ x < 0\ \texttt{then}\ m := -x\ \texttt{else}\ m := x, m \geq 0)$.

Then branch:

$$wlp(m := -x, m \geq 0) = -x \geq 0$$

Else branch:

$$wlp(m := x, m \geq 0) = x \geq 0$$

Combined:

$$\begin{aligned}
&wlp(\texttt{if}\ x < 0\ \texttt{then}\ m := -x\ \texttt{else}\ m := x, m \geq 0) \\
&= (x < 0 \to -x \geq 0) \wedge (x \geq 0 \to x \geq 0) \\
&\equiv \texttt{true}
\end{aligned}$$

This means the postcondition $m \geq 0$ *holds for any initial state.*

5.3.5 While Loop

For a while loop with invariant I:

$$\begin{aligned}
&\text{wlp}(\texttt{while}\ C\ \texttt{inv:}\ \ I\ \texttt{do}\ S, Q) = \\
&\quad I \wedge \forall x_1, \ldots, x_k. \Big((C \wedge I \to \text{wlp}(S, I)) \wedge (\neg C \wedge I \to Q) \Big)[x_i / w_i]
\end{aligned}$$

where $w_1, \ldots, w_k$ are the variables modified in S, and $x_1, \ldots, x_k$ are fresh variables.

Example 5.7. *Compute* $wlp(\texttt{while}\ i < n\ \texttt{inv:}\ \ i \leq n\ \texttt{do}\ i := i + 1, i = n)$.

$$\begin{aligned}
&wlp(\texttt{while}\ i < n\ \texttt{inv:}\ \ i \leq n\ \texttt{do}\ i := i + 1, i = n) \\
&= i \leq n \wedge \forall x. \Big((x < n \wedge x \leq n \to wlp(x := x + 1, x \leq n)) \\
&\qquad\qquad \wedge (\neg(x < n) \wedge x \leq n \to x = n) \Big) \\
&= i \leq n \wedge \forall x. \Big((x < n \wedge x \leq n \to x + 1 \leq n) \\
&\qquad\qquad \wedge (x \geq n \wedge x \leq n \to x = n) \Big) \\
&\equiv i \leq n \wedge \forall x. \Big((x < n \to x + 1 \leq n) \wedge (x = n \to x = n) \Big) \\
&\equiv i \leq n
\end{aligned}$$

6 Soundness and Completeness of WLP

6.1 Soundness

Theorem 6.1 (Soundness of WLP)**.** *For all statements* S *and postconditions* Q*, we have* $\vdash \{wlp(S, Q)\} S \{Q\}$.

Proof Sketch. By structural induction on S. For loops, an additional induction on the length of execution is required. A key insight is that the universally quantified formula in the wlp of loops does not depend on the values of the modified variables, so it remains valid across consecutive program states. □

6.2 WLP Property

Theorem 6.2 (WLP Property)**.** *For any triple $\{P\}S\{Q\}$ derivable using Hoare logic proof rules (with the modified loop rule using invariants), we have $P \to wlp(S, Q)$.*

Consequence: To prove $\{P\}S\{Q\}$, it suffices to prove $P \to \text{wlp}(S, Q)$, which can be done using automated theorem provers.

7 Total Correctness and Weakest Precondition

7.1 The Challenge of Termination

For total correctness, we must prove both:

1. The postcondition holds after execution (correctness)
2. The program terminates (termination)

The difficulty lies in proving termination, especially for loops.

7.2 Loop Variants

To prove termination of loops, we use *variants*—expressions that decrease with each iteration according to a well-founded relation.

Well-Founded Relation: A relation $\prec$ is well-founded if there are no infinite descending chains $\xi_1 \succ \xi_2 \succ \xi_3 \succ \cdots$.

Example: On integers, define:

$$x \prec y \equiv x < y \land 0 \leq y$$

This is well-founded because we cannot have an infinite sequence decreasing below 0.

7.3 Modified Loop Rule for Total Correctness

$$\frac{\vdash \{I \land C \land v = \xi\} S \{I \land v \prec \xi\} \quad \text{wf}(\prec)}{\vdash \{I\}\ \texttt{while}\ C\ \texttt{inv:}\ \ I\ \texttt{variant:}\ \ v\ \texttt{do}\ S\{I \land \neg C\}}$$

Here, v is the variant expression and ξ is a fresh logical variable representing the value of v before executing S.

7.4 IMP with Variants

We extend the language to include variant annotations:

$$\text{Stmt} ::= \cdots \mid \texttt{while}\ \text{BExp}\ \texttt{inv:}\ \ \varphi\ \texttt{variant:}\ \ \psi\ \texttt{do}\ \text{Stmt}\ \texttt{end}$$

8 Weakest (Strict) Precondition

8.1 Definition

The weakest precondition (without "liberal") $\text{wp}(S, Q)$ ensures both correctness and termination.

8.2 Computing WP

Most rules are identical to wlp:

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

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

Composition: $\text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q))$

Conditional:

$$\text{wp}(\texttt{if } C \texttt{ then } S_1 \texttt{ else } S_2, Q) = (C \to \text{wp}(S_1, Q)) \wedge (\neg C \to \text{wp}(S_2, Q))$$

While Loop:

$$\begin{aligned}\text{wp}(\texttt{while } C \texttt{ inv: } I \texttt{ variant: } v \texttt{ do } S, Q) = \\ I \wedge \forall x_1, \ldots, x_k, \xi. \Big((C \wedge I \wedge \xi = v \to \text{wp}(S, I \wedge v \prec \xi)) \\ \wedge (\neg C \wedge I \to Q) \Big)[x_i/w_i]\end{aligned}$$

The key difference from wlp is in the loop rule: we must verify that the variant decreases ($v \prec \xi$) after each iteration.

8.3 Extended Example

Example 8.1. *Consider the program:*

```
while i < n
    inv: i <= n
    variant: n - i
do
    i := i + 1
end
```

Compute $wp(\texttt{incToN}, i = n)$.
Step 1: *Apply the wp rule for loops:*

$$\begin{aligned}wp(\texttt{incToN}, i = n) = i \leq n \wedge \forall x, \xi. \Big((x < n \wedge x \leq n \wedge \xi = n - x \\ \to wp(x := x + 1, x \leq n \wedge (n - x) \prec \xi)) \\ \wedge (x \geq n \wedge x \leq n \to x = n) \Big)\end{aligned}$$

Step 2: *Compute* $wp(i := i + 1, i \leq n \wedge (n - i) \prec \xi)$.
Recall that $x \prec y \equiv x < y \wedge 0 \leq y$. *So:*

$$wp(i := i + 1, i \leq n \wedge (n - i) \prec \xi) = i + 1 \leq n \wedge (n - (i + 1)) < \xi \wedge 0 \leq \xi$$

Step 3: *Substitute back:*

$$\begin{aligned}wp(\texttt{incToN}, i = n) = i \leq n \wedge \forall x, \xi. \Big((x < n \wedge x \leq n \wedge \xi = n - x \\ \to (x + 1 \leq n \wedge n - (x + 1) < \xi \wedge 0 \leq \xi)) \\ \wedge (x = n \to x = n) \Big)\end{aligned}$$

Step 4: *Simplify. The second conjunct is trivially true. For the first:*

- $x < n$ *implies* $x + 1 \leq n$
- $\xi = n - x$ *implies* $n - (x + 1) < \xi \equiv n - x - 1 < n - x \equiv -1 < 0$
- $x < n$ *means* $n - x > 0$, *so* $\xi = n - x > 0$ *implies* $0 \leq \xi$

Therefore:

$$wp(\texttt{incToN}, i = n) = i \leq n$$

9 Soundness Results for WP

Theorem 9.1 (Soundness of WP)**.** *For all statements S and postconditions Q, we have $\vdash \{wp(S, Q)\}S\{Q\}$ (for total correctness).*

Theorem 9.2 (WP Property)**.** *For any triple $\{P\}S\{Q\}$ derivable using the total correctness proof rules, we have $P \to wp(S, Q)$.*

A practical consequence of this property is that to prove total correctness of $\{P\}S\{Q\}$, we need to prove the formula $\models P \to \text{wp}(S, Q)$.

10 Summary

10.1 Differences between wlp and wp

Weakest Liberal Precondition (wlp):

- Ensures *partial correctness*
- If $\text{wlp}(S, Q)$ holds before S and S terminates, then Q holds after
- Does not guarantee termination

Weakest Precondition (wp):

- Ensures *total correctness*
- If $\text{wp}(S, Q)$ holds before S, then S terminates and Q holds after
- Requires variants to prove loop termination

10.2 Advantages of the WP Calculus

1. **Automation:** Reduces Hoare logic proofs to logical implications
2. **Systematic:** Provides mechanical rules for computing preconditions
3. **Tool-friendly:** Generated formulas can be checked by SMT solvers
4. **Modular:** Each construct has a clear, compositional semantics

10.3 Limitations

1. Loop invariants and variants must still be provided by the user
2. Generated formulas can be complex and may require powerful theorem provers
3. Completeness is lost when invariants are not strong enough

11 Further Reading

- C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," Communications of the ACM, 1969
- Edsger W. Dijkstra, "A Discipline of Programming," Prentice Hall, 1976
- Krzysztof R. Apt and Ernst-Rüdiger Olderog, "Verification of Sequential and Concurrent Programs," Springer, 1997