Hoare Logic

Zhaoguo Wang

Adapted From:

ETH Zurich program verification, Lecture 6, 7
(https://www.pm.inf.ethz.ch/education/courses/program-verification.html)
<<Forward with Hoare>>
<<Weakest-Precondition of Unstructured Programs>>
Foundations of Programming Languages(https://cs.nju.edu.cn/xyfeng/teaching/FOPL, Hoare Logic)
<<Background reading on Hoare Logic>>, Mike Gordon

Lab1

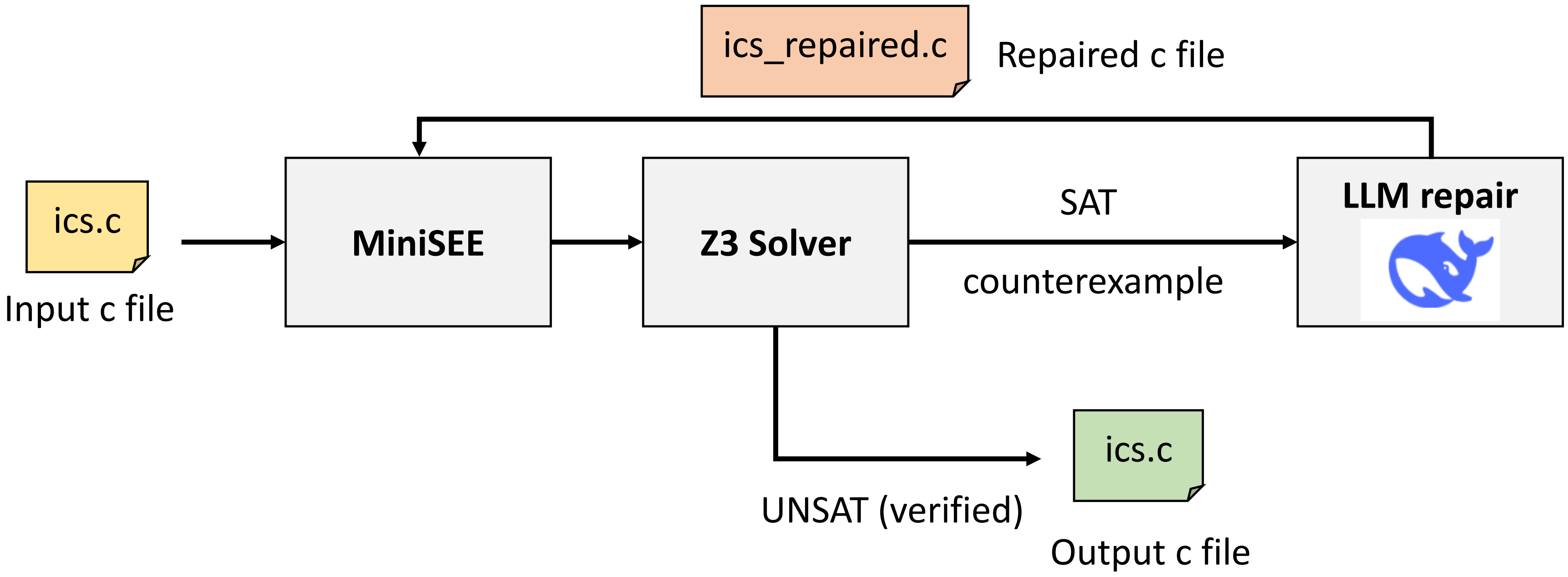
- **Lab1 DDL is 11.16. Hurry up!**
- **If you need to use grace day, please inform TA in advance.**


Lab2 is coming!
BUG
POWDER
Program Repair

Lab2

MiniRepairer: Automatic Repairer for Simple C Programs

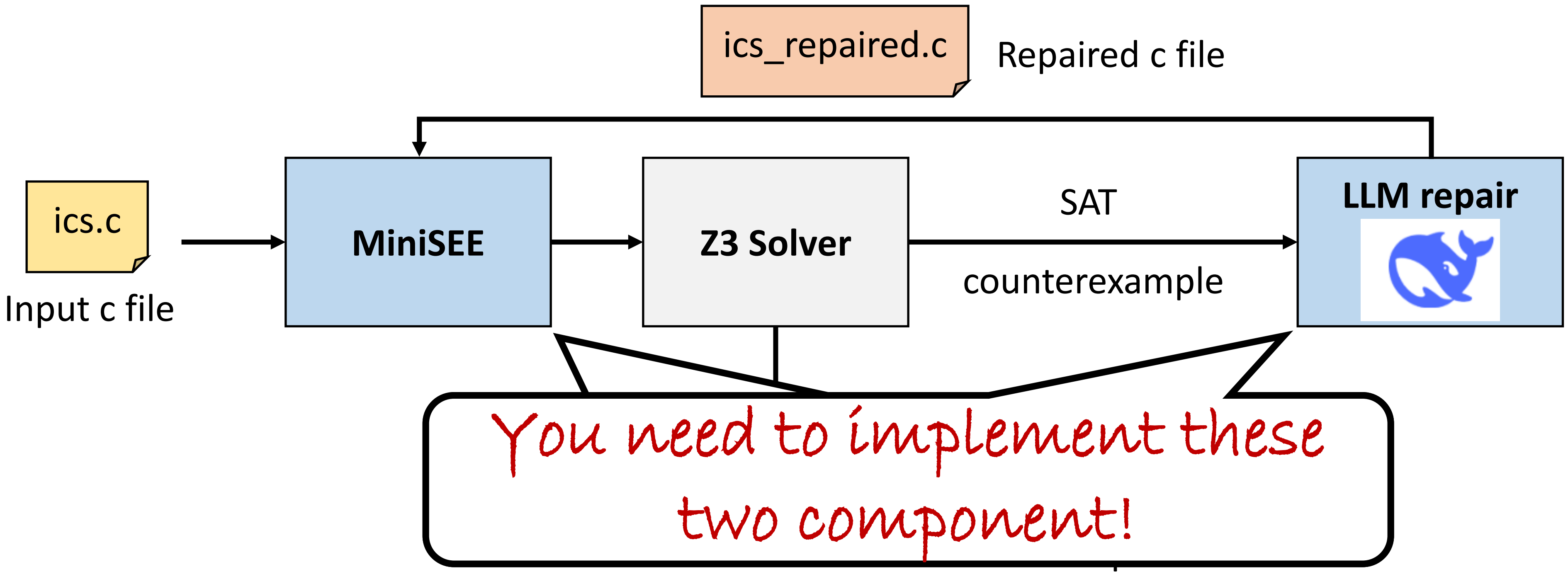
- Verify assertions in the input c files
- Try to repair the code with LLM



Lab2

MiniRepairer: Automatic Repairer for Simple C Programs

- Verify assertions in the input c files
- Try to repair the code with LLM

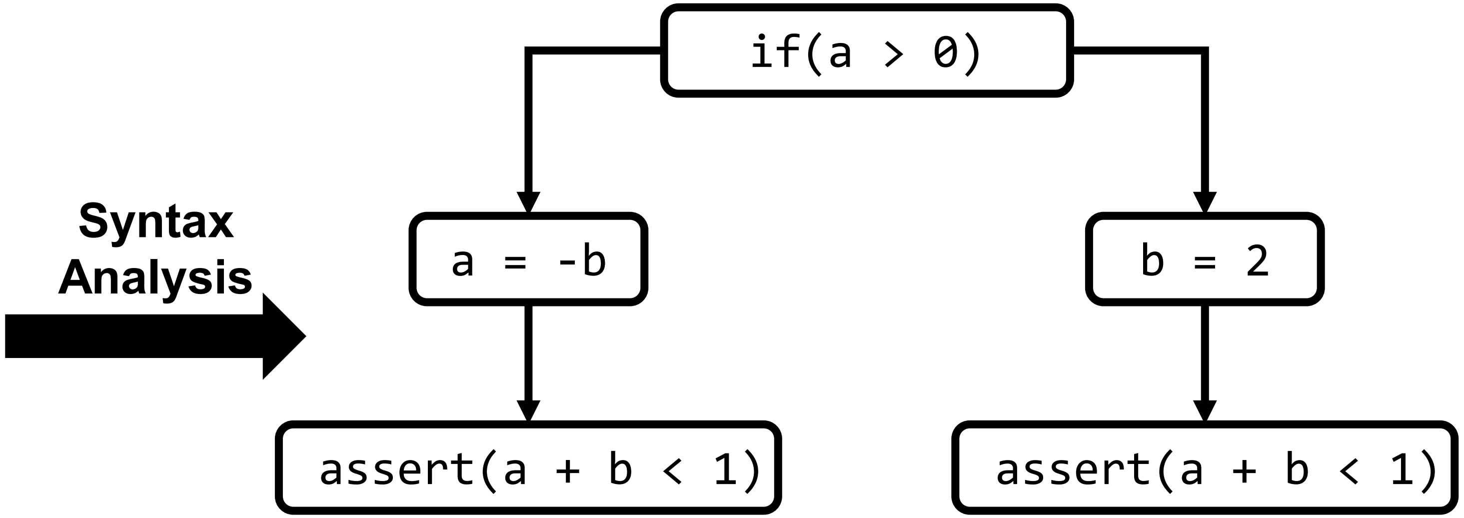


Lab2

MiniSEE: A symbolic execution engine for simple C program

func1.c

```
int func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```



MiniSEE first parses the function into a syntax tree via syntax analysis

- Each node denotes a statement
- Each arrow denotes the execution order between two statements

Then MiniSEE performs symbolic execution based on the tree, invoke Z3 to solve the assertion!

Lab2

MiniSEE: A symbolic execution engine for simple C program

func1.c

```c
int func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

MiniSEE →

```
> minisee("func1.c")

-1 0
```

Counterexample that makes assertion fail

Lab2

You can even use MiniSEE to verify your ICS lab1 solutions

ics.c

```c
// compute !x without using !
int bang(int x){
  int r1, neg_x, r2, res;
  r1 = x >> 31;
  neg_x = ~x + 1;
  r2 = neg_x >> 31;
  res = (r1 | r2);
  assert((x == 0 && res == 1)
      || (x != 0 && res == 0));
  return res;
}
```

MiniSEE →

```
> minisee("ics.c")

0
```

Lab2

You can even use MiniSEE to verify your ICS lab1 solutions

ics.c

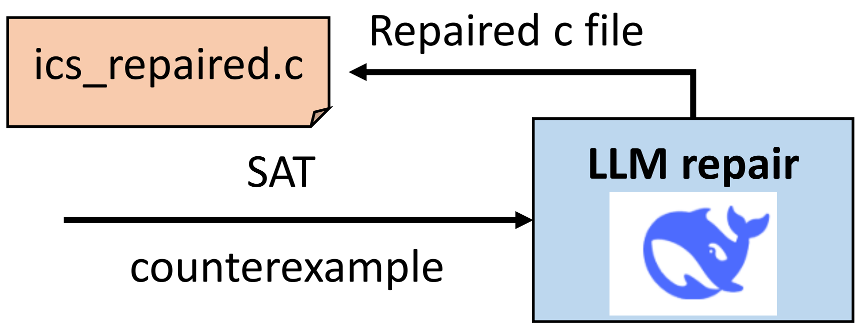
```
// compute !x without using !
int bang(int x){
  int r1, neg_x, r2, res;
  r1 = x >> 31;
  neg_x = ~x + 1;
  r2 = neg_x >> 31;
  res = (r1 | r2) + 1;
  assert((x == 0 && res == 1)
      || (x != 0 && res == 0));
  return res;
}
```

MiniSEE →

```
> minisee("ics.c")

verified
```

Lab2



LLM Repair: Try to fix the bug with LLM (recursively)

`ics.c`

The following C code failed symbolic execution verification.

```
int bang(int x){
  int r1, neg_x, r2, res;
  r1 = x >> 31;
  neg_x = ~x + 1;
  r2 = neg_x >> 31;
  res = (r1 | r2);
  assert((x == 0 && res == 1)
      || (x != 0 && res == 0));
  return res;
}
```

Counterexample (input values that trigger assertion failure):

```
x=0
```

Please fix the code to pass the assertion.

`ics_repaired.c`

```
int bang(int x){
  int r1, neg_x, r2, res;
  r1 = x >> 31;
  neg_x = ~x + 1;
  r2 = neg_x >> 31;
  res = (r1 | r2) + 1;
  assert((x == 0 && res == 1)
      || (x != 0 && res == 0));
  return res;
}
```

Hoare Logic （霍尔逻辑）

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

Outline – This Lecture

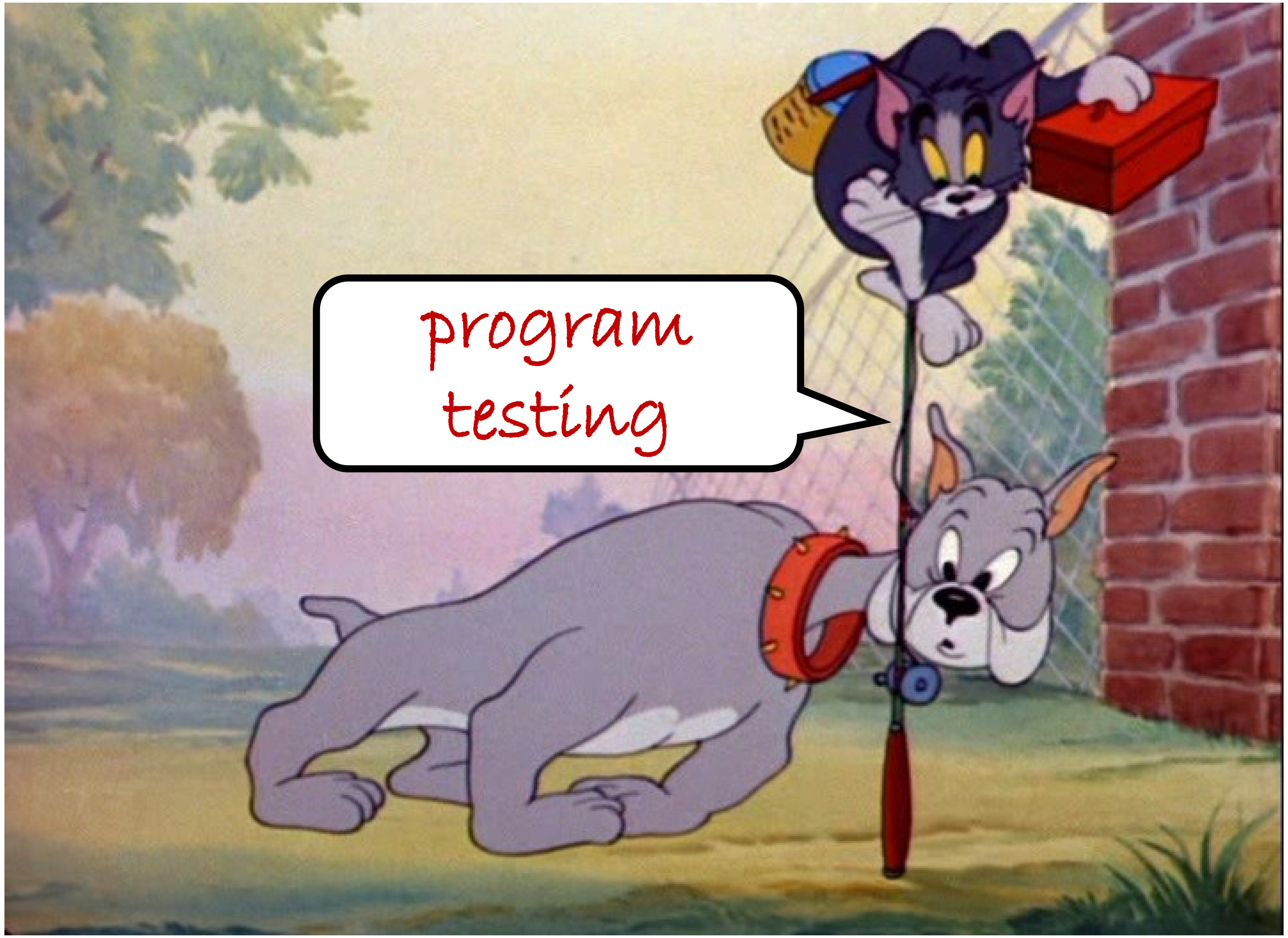
- Axiom System
- Hoare Logic

Outline – This Lecture

- Axiom System
- Hoare Logic

Program Analysis

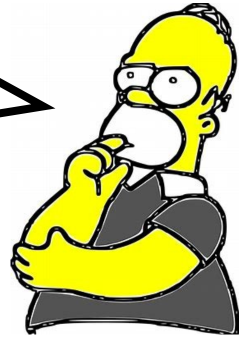
Program Analysis



Program Analysis

Program Analysis

With SMT solvers, how to convert the program to predicate logic formulas?



Program Analysis

```
int select(int a, int b, int c)
{
	if(c >= 0)
		return a;
	else
		return b;
}
```

Prove that the return value is greater than zero if both a and b are greater than zero.

Program Analysis

```
int select(int a, int b, int c)
{
	if(c >= 0)
		return a;
	else
		return b;
}
```

➡

$$(a > 0) \wedge$$
$$(b > 0) \wedge$$
$$((c \geq 0) \rightarrow (result = a)) \wedge$$
$$(\neg(c \geq 0) \rightarrow (result = b))$$
$$\rightarrow (result > 0)$$

Prove it is universally valid.

Program Analysis

$$(a > 0) \wedge$$
$$(b > 0) \wedge$$
$$((c \geq 0) \rightarrow (result = a)) \wedge$$
$$(\neg(c \geq 0) \rightarrow (result = b))$$
$$\rightarrow (result > 0)$$

Prove it is universally valid.

➡

$$(a > 0) \wedge$$
$$(b > 0) \wedge$$
$$((c \geq 0) \rightarrow (result = a)) \wedge$$
$$(\neg(c \geq 0) \rightarrow (result = b)) \wedge$$
$$\neg(result > 0)$$

Prove it is unsatisfiable.

Loops

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    while(i > 0)
    {
        a = a + 1;
        i = i - 1;
    }
    assert(a == 3);
}
```

Loops

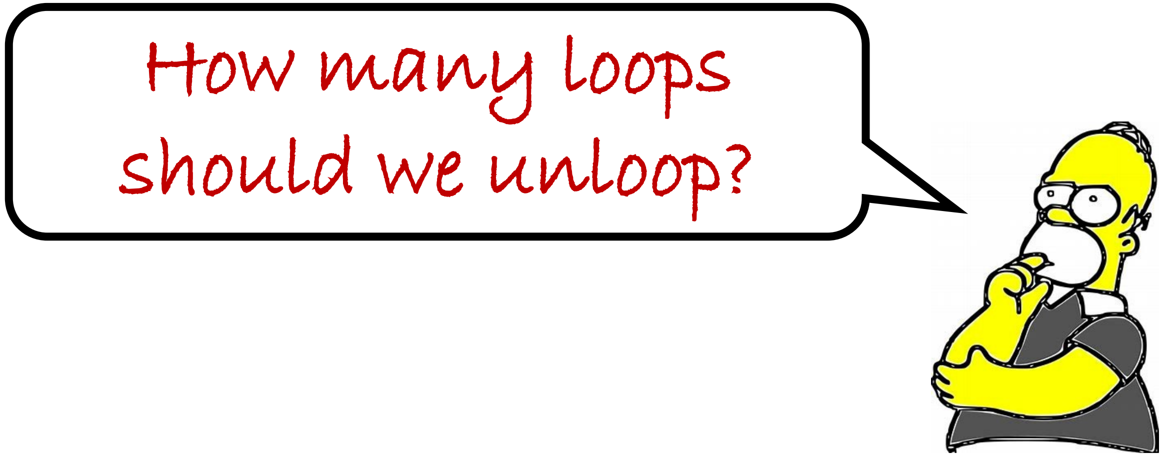
unroll

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
}
```

Straight-Line Code

Loops

```
void f(unsigned a)
{
    unsigned i = 0;
    while(i < a)
    {
        i = i + 1;
    }
    assert(i == a);
}
```

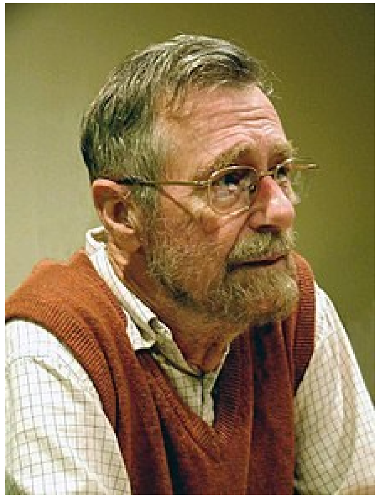


Loops

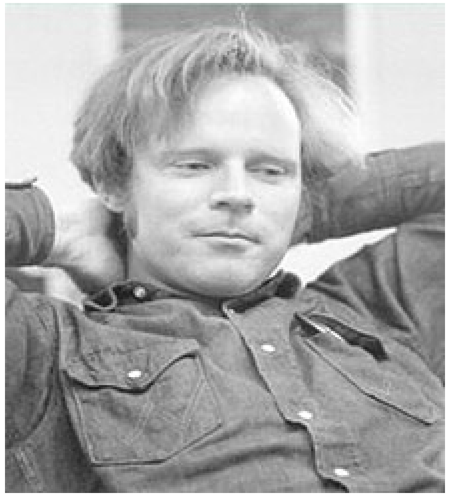
```
void f(unsigned a)
{
    unsigned i = 0;
    while(i < a)
    {
        i = i + 1;
    }
    assert(i == a);
}
```

Hoare Logic Can Do It!

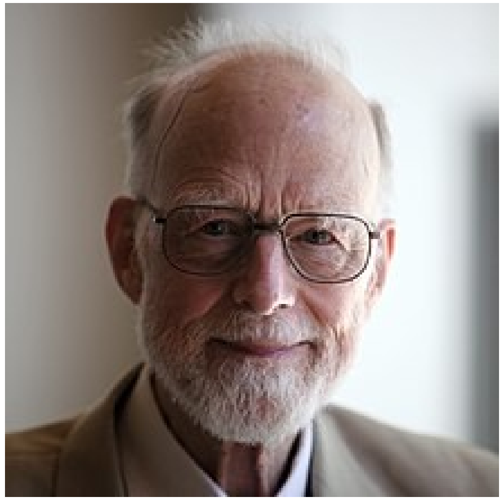
Assign logical meaning to programs



Edsger Wybe Dijkstra
Turing Award(1972)



Robert W. Floyd
Turing Award(1978)



Charles Antony Richard Hoare
Turing Award(1980)

C. A. R. Hoare:
An Axiomatic Basis for Computer Programming.
Commun. ACM 12(10): 576-580, 1969.

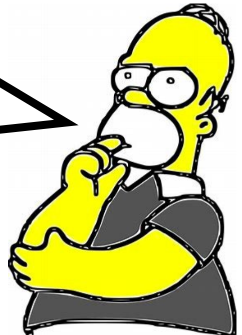
Hoare logic is an axiom system for program verification.

Axiom System(公理系统)

DEFINITION

An axiom system is a logical system which possesses a group of axioms(公理) from which theorems(定理) can be derived.

We can firstly use propositional logic to explain the components of an axiom system.



Axiom System(公理系统)

DEFINITION

Initial symbols(初始符号) include all the symbols in the axiom system.

propositional variables: $A, B, C, \ldots$

complete connectives: $\neg, \vee$

$(,)$

$\vdash$ — It is before a formula, denoting the formula is a tautology.

Axiom System(公理系统)

DEFINITION

Formation rules(形成规则) define the legal symbol strings in the axiom system.

INDUCTIVE DEFINITION of WFF

1). Every single proposition (symbol) is in WFF.

2). If *A* and *B* are WFF, so are (¬A) and (A v B).

3). No expression is WFF unless forced by 1) or 2).

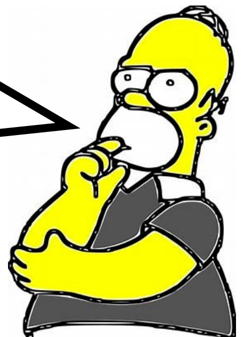
Axiom System(公理系统)

$$(A \to B) = (\neg A \vee B)$$

$$(A \wedge B) = \neg(\neg A \vee \neg B)$$

$$(A \leftrightarrow B) = ((A \to B) \wedge (B \to A))$$

Sometimes we can define some new symbol strings for abbreviation.



Axiom System(公理系统)

DEFINITION

Axioms(公理) include some tautologies from which theorems can be derived. We don't need to prove them in the axiom system.

They should be selected carefully. Otherwise, some theorems cannot be derived.

$$Axiom1 \quad \vdash ((P \vee P) \rightarrow P)$$
$$Axiom2 \quad \vdash (P \rightarrow (P \vee Q))$$
$$Axiom3 \quad \vdash ((P \vee Q) \rightarrow (Q \vee P))$$
$$Axiom4 \quad \vdash ((Q \rightarrow R) \rightarrow ((P \vee Q) \rightarrow (P \vee R)))$$

Axiom System(公理系统)

DEFINITION

Inference rules(变形规则/推导规则) defines how to infer new theorems from axioms and known theorems.

substitution rule(代入规则): $\vdash P \vee \neg P \quad \frac{P}{(R \vee S)} \Longrightarrow \quad \vdash (R \vee S) \vee \neg(R \vee S)$

分离规则: If $\vdash A$, $\vdash A \rightarrow B$, then $\vdash B$

置换规则: Replace part of A by equivalent(等值) expressions to get B: $\Longrightarrow$ if $\vdash A$, then $\vdash B$

Axiom System(公理系统)

DEFINITION

Building theorems(建立定理) includes all the tautologies and their proofs.

$$\vdash (Q \to R) \to ((P \to Q) \to (P \to R))$$

$Proof.$

Axiom System(公理系统)

DEFINITION

Building theorems(建立定理) includes all the tautologies and their proofs.

$$\vdash (Q \to R) \to ((P \to Q) \to (P \to R))$$

Proof.

$(1) \vdash (Q \to R) \to (P \vee Q \to P \vee R)$ $\quad Axiom4$

Axiom System(公理系统)

DEFINITION

Building theorems(建立定理) includes all the tautologies and their proofs.

$$\vdash (Q \to R) \to ((P \to Q) \to (P \to R))$$

Proof.

$(1) \vdash (Q \to R) \to (P \vee Q \to P \vee R)$ $\quad Axiom4$

$(2) \vdash (Q \to R) \to (\neg P \vee Q \to \neg P \vee R)$ $\quad \frac{P}{\neg P}$ on (1)

Axiom System(公理系统)

DEFINITION

Building theorems(建立定理) includes all the tautologies and their proofs.

$$\vdash (Q \to R) \to ((P \to Q) \to (P \to R))$$

Theorem 1

Proof.

$(1) \vdash (Q \to R) \to (P \vee Q \to P \vee R)$ $\qquad$ $Axiom4$

$(2) \vdash (Q \to R) \to (\neg P \vee Q \to \neg P \vee R)$ $\qquad$ $\frac{P}{\neg P}$ $on\ (1)$

$(3) \vdash (Q \to R) \to ((P \to Q) \to (P \to R))$ $\qquad$ $definition\ of\ \to on(2)$

Axiom System(公理系统)

$\vdash P \to P$ — Theorem 2

Proof.

(1) $\vdash P \to P \vee Q$	Axiom 2
(2) $\vdash P \to P \vee P$	Replace Q with P in (1)
(3) $\vdash P \vee P \to P$	Axiom 1
(4) $\vdash (Q \to R) \to \big((P \to Q) \to (P \to R)\big)$	Theorem 1
(5) $\vdash ((P \vee P) \to R) \to ((P \to (P \vee P)) \to (P \to R))$	Replace Q with P ∨ P in (4)
(6) $\vdash ((P \vee P) \to P) \to ((P \to (P \vee P)) \to (P \to P))$	Replace R with P in (5)
(7) $\vdash (P \to (P \vee P)) \to (P \to P)$	Hypothetical reasoning on (3) (6)
(8) $\vdash P \to P$	Hypothetical reasoning on (2) (7)

Axiom System(公理系统)

$\vdash P \to P$ Theorem 2

Proof.

(1) $\vdash P \to P \vee Q$	Axiom 2
(2) $\vdash P \to P \vee P$	Replace Q with P in (1)
(3) $\vdash P \vee P \to P$	Axiom 1
(4) $\vdash (Q \to R) \to \big((P \to Q) \to (P \to R)\big)$	Theorem 1
(5) $\vdash ((P \vee P) \to R) \to ((P \to (P \vee P)) \to (P \to R))$	Replace Q with $P \vee P$ in (4)
(6) $\vdash ((P \vee P) \to P) \to ((P \to (P \vee P)) \to (P \to P))$	Replace R with P in (5)
(7) $\vdash (P \to (P \vee P)) \to (P \to P)$	Hypothetical reasoning on (3) (6)
(8) $\vdash P \to P$	Hypothetical reasoning on (2) (7)

$\vdash \neg P \vee P$ Theorem 3

Proof.

(1) $\vdash P \to P$	Theorem 2
(2) $\vdash \neg P \vee P$	Definition of $\to$ on (1)

Completeness and Soundness

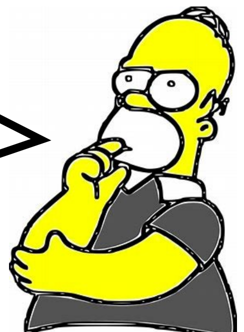
DEFINITION

Completeness: if P is a tautology, $\vdash P$ can be proved in the system.

DEFINITION

Soundness: if $\vdash P$ can be proved in the system, then P must be a tautology.

Soundness is the converse of completeness.
Propositional logic is complete and sound.



Completeness Proof

Completeness: if A is a tautology, $\vdash A$ can be proved in the system

PROOF

1. All proposition A can be converted to CNF:

$$\mathrm{A} = A_1 \land A_2 \land \cdots \land A_n$$

Completeness Proof

Completeness: if A is a tautology, $\vdash A$ can be proved in the system

PROOF

1. All proposition A can be converted to CNF:

$$\mathrm{A} = A_1 \land A_2 \land \cdots \land A_n$$

2. If A is tautology, then each disjunctive clause contains a complementary pair.

$$A_i = \neg\pi \lor \pi \lor \cdots$$

Completeness Proof

Completeness: if A is a tautology, $\vdash A$ can be proved in the system

PROOF

1. All proposition A can be converted to CNF:

$$\mathrm{A} = A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

2. If A is tautology, then each disjunctive clause contains a complementary pair.

$$A_i = \neg\pi \vee \pi \vee \cdots$$

3. Each $\vdash A_i$ can be proved in system:

$$\vdash \neg\pi \vee \pi, \quad \vdash P \rightarrow (P \vee Q) \Longrightarrow \vdash \neg\pi \vee \pi \vee Q$$

$\vdash A_i$

We have proved it as theorem 3

Axiom 2

Completeness Proof

Completeness: if A is a tautology, $\vdash A$ can be proved in the system

PROOF

1. All proposition A can be converted to CNF:
$$A = A_1 \land A_2 \land \cdots \land A_n$$
2. If A is tautology, then each disjunctive clause contains a complementary pair.
$$A_i = \neg\pi \lor \pi \lor \cdots$$
3. Each $\vdash A_i$ can be proved in system:
$$\vdash \neg\pi \lor \pi, \quad \vdash P \to (P \lor Q) \Longrightarrow \vdash \neg\pi \lor \pi \lor Q$$
$\vdash A_i$
4. Since each clause is proven, $\vdash A$ can be proved in system. ■

Soundness Proof

Soundness: if $\vdash P$ can be proved in the system, then P must be a tautology.

PROOF

1. All axioms are tautology.

Soundness Proof

Soundness: if $\vdash P$ can be proved in the system, then P must be a tautology.

PROOF

1. All axioms are tautology.
2. If all theorems used is tautology, the new theorem created by inference rules is also tautology.

Soundness Proof

Induction!

Soundness: if $\vdash P$ can be proved in the system, then P must be a tautology.

PROOF

1. All axioms are tautology.
2. If all theorems used is tautology, the new theorem created by inference rules is also tautology.

If $\vdash$ A , $\vdash$ A $\rightarrow B$, then $\vdash$ B

If A and $A \rightarrow B$ hold, then B must hold

Other inference rules can also be proved

Prove it by listing all possible situations (truth table)

Soundness Proof

Soundness: if $\vdash P$ can be proved in the system, then P must be a tautology.

PROOF

1. All axioms are tautology.
2. If all theorems used is tautology, the new theorem created by inference rules is also tautology.

   If $\vdash \mathrm{A}$, $\vdash \mathrm{A} \rightarrow B$, then $\vdash \mathrm{B}$

   If A and $A \rightarrow B$ hold, then B must hold
3. So all the $\vdash P$ deduced in the system must be tautology. ■

Outline – This Lecture

- Axiom System
- Hoare Logic

Hoare logic is designed specifically for program verification. It answers what is program correctness and how to formally prove the correctness.

Problem1:

How to define program correctness?

Hoare Triples(霍尔三元组)

DEFINITION

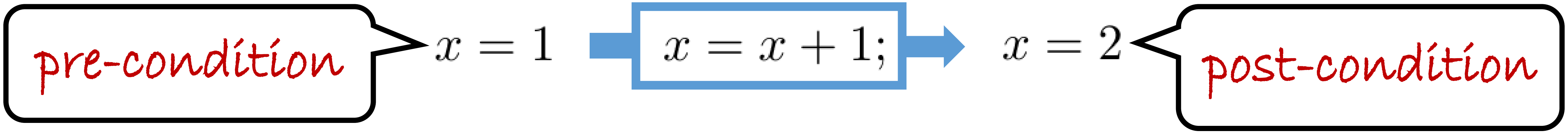
Program state includes the value of every variable used in the program.

DEFINITION

Pre-condition describes the program state before executing the program.

DEFINITION

Post-condition describes the program state after executing the program.



Hoare Triples

DEFINITION

Program state includes the value of every variable used in the program.

DEFINITION

Pre-condition describes the program state before executing the program.

DEFINITION

Post-condition describes the program state after executing the program.

$$x = 3 \wedge y = 5$$

$$x > 3 \wedge x \neq 0$$

Pre-/post-condition can be predicate logic formulas.

Hoare Triples

DEFINITION

A program C is partially correct iff:

whenever C is executed in a state initially satisfying pre-condition and if the execution of C terminates, then the state in which C's execution terminates satisfies post-condition.

$$\{P\}\ C\ \{Q\}$$

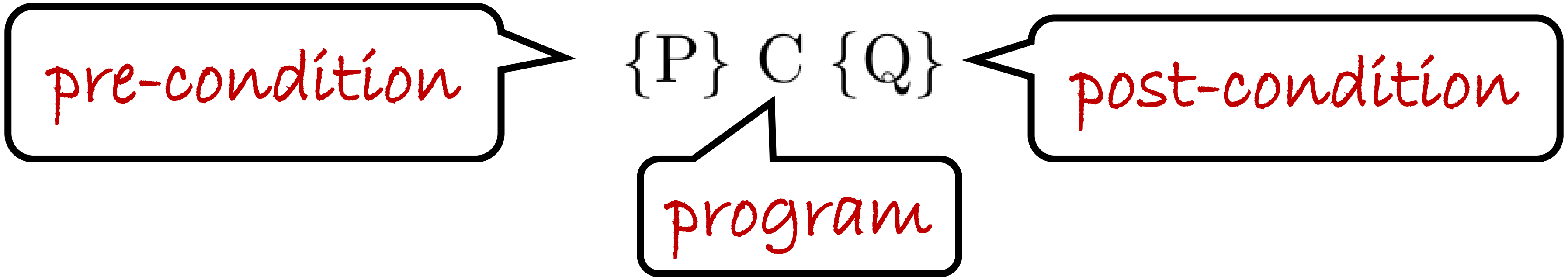
We use Hoare triples to define program correctness.

Hoare Triples

DEFINITION

A program C is partially correct iff:

whenever C is executed in a state initially satisfying pre-condition and if the execution of C terminates, then the state in which C's execution terminates satisfies post-condition.



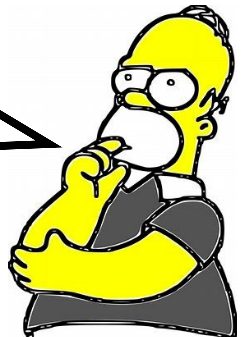
Hoare Triples

DEFINITION

A program C is partially correct iff:

whenever C is executed in a state initially satisfying pre-condition and if the execution of C terminates, then the state in which C's execution terminates satisfies post-condition.

We don't consider whether the given program can terminate (partially correct).



Halting Problem

DEFINITION

Halting problem means if we can have an algorithm that will tell that the arbitrary given program will halt or not.

Basically halting means terminating.

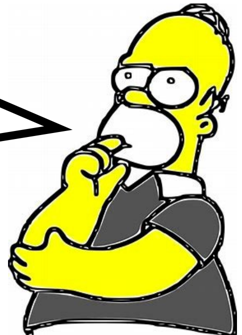
Alan Turing proved in 1936 that a general algorithm to solve the halting problem cannot exist.

Hoare Triples

The Hoare triple is the correctness criterion of program C, which is called specification.

$$\{P\}\ C\ \{Q\}$$

P and Q are predicate logic propositions. What about C?



A Small Imperative Language

- Program variables
  - a, b, c, … (can be assigned to)
- Numbers 1, 2, 3,…
- Expressions
  - Expression evaluation is assumed to be side-effect-free for all expressions
  - Arithmetic expressions: e1+e2, e1-e2, …
  - Boolean expressions: e1 > e2, e1 == e2, e1 < e2, …
  - We typically write b1, b2,… for boolean-typed expressions and e1, e2, … for arithmetic expressions

A Small Imperative Language

- Statements
  - skip (does nothing when executed)
  - a := e (assignment: changes value of a)
  - s1; s2 (sequential composition: execute s1 followed by s2)
  - if(b1) {s1} else {s2} (execute s1 if b is true; s2 otherwise)
  - While(b1) {s} (repeatedly execute s while b is true)
- We don’t consider
  - heap, pointer, break, continue, function, bit-operator, object-oriented, struct, union, array, float numbers…

Hoare Logic

- Initial symbols
  - symbols in predicate logic : ∀, ∃, →, ¬, ...
  - symbols in the simple imperative language: if, while, :=, …
  - vdash : ⊢ (we omit it in following slides)

- Formation rules
  - {P} C {Q}
  - P and Q should be predicate logic formulas, which describe program state
  - C should be a legal program (no syntax error)

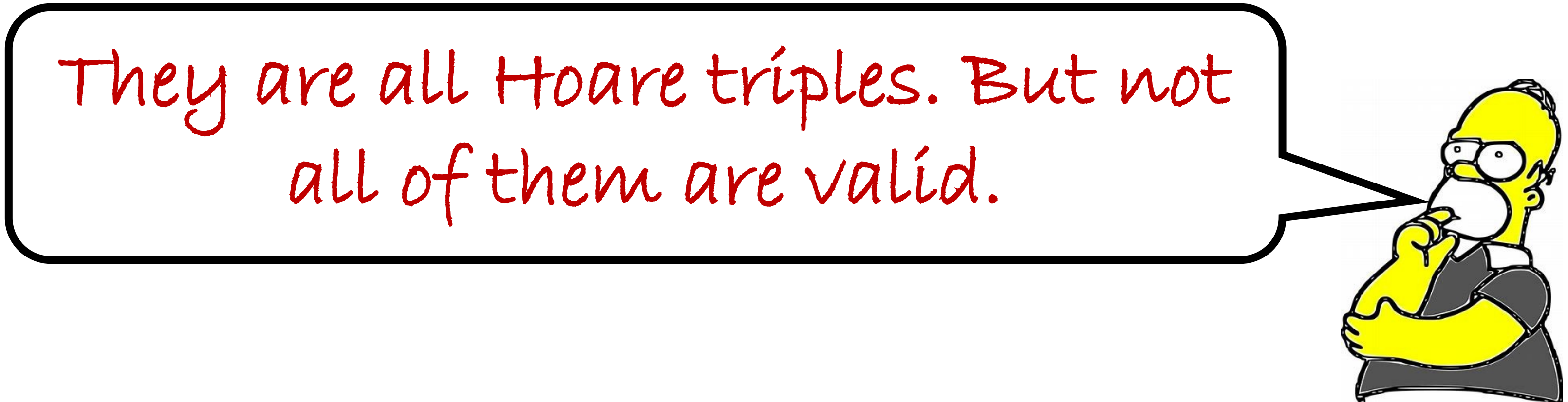
Hoare Logic

$$\{a = 0\}a := 3\{a = 3\}$$

$$\{a = 1\}a := 2; a := a + 1\{a = 3\}$$

$$\{a = 1\}a := 2\{a = 10\}$$

$$\{a = 1 \wedge b = 5\}if(a == 1)a := b\{a = 5 \wedge b = 5\}$$



Hoare Logic

$$\{a = 0\}a := 3\{a = 3\}$$ ✓

$$\{a = 1\}a := 2; a := a + 1\{a = 3\}$$ ✓

$$\{a = 1\}a := 2\{a = 10\}$$ ✗

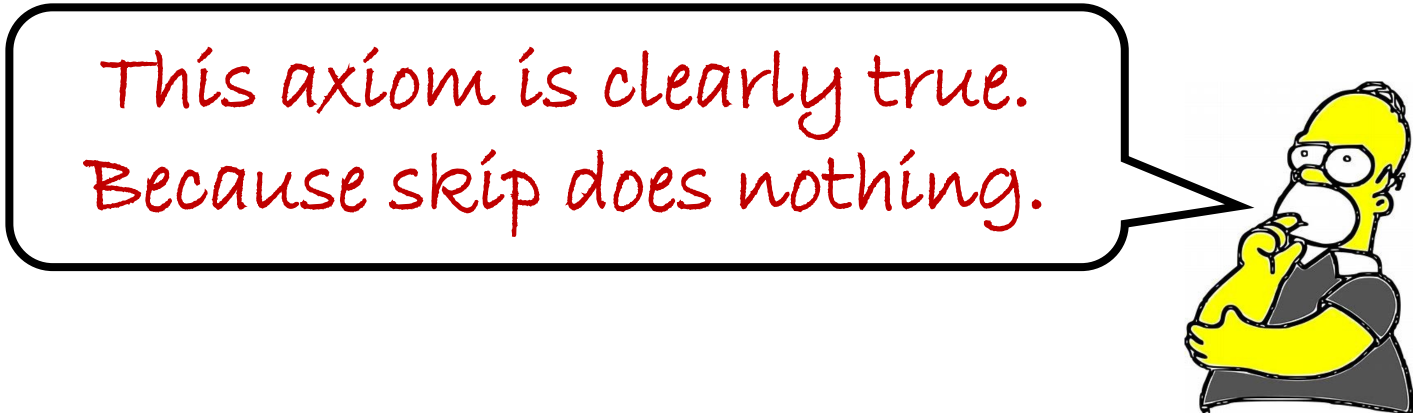
$$\{a = 1 \wedge b = 5\}if(a == 1)a := b\{a = 5 \wedge b = 5\}$$ ✓

We need to formally prove whether a Hoare triple is right. In other words, we need to prove the program satisfies the specification.

Problem 2:
How to prove program correctness?

Axioms

$$\{P\} skip \{P\}$$



Axioms

$$\{?\}a := e\{?\}$$

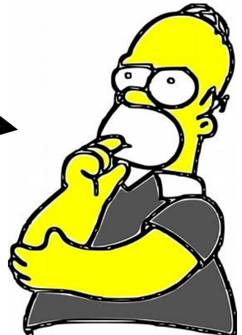
Axioms

Use a to replace e in P

$$\{P\}a := e\{P[a/e]\}$$

$$\{a = 0\}a := 1\{a = 0\}$$ ☒

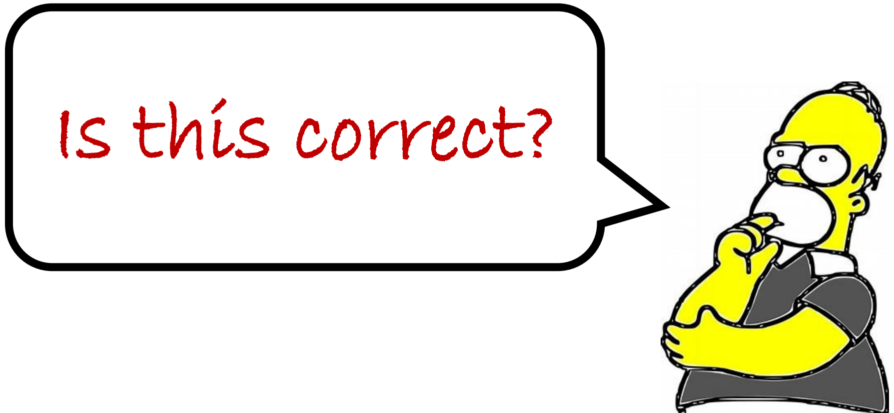
Is this correct?



Axioms

$$\{P\}a := e\{P \wedge a = e\}$$

$$\{a = 5\}a := a + 1\{a = 5 \wedge a = a + 1\}$$



Axioms

v is a fresh variable.

$$\{P\}a := e\{(\exists v)(a = e[v/a] \wedge P[v/a])\}$$

$$\{a = 1\}a := a + 1\{(\exists v)(a = (a+1)[v/a] \wedge (a = 1)[v/a])\}$$

$$\Downarrow$$

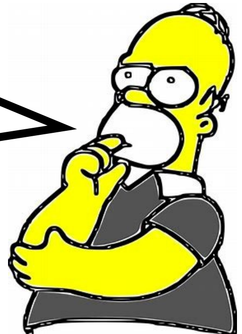
$$\{a = 1\}a := a + 1\{(\exists v)(a = v + 1 \wedge v = 1)\}$$

simplification

$$\Downarrow$$

$$\{a = 1\}a := a + 1\{a = 2\}$$

This is correct and a forward assignment axiom.



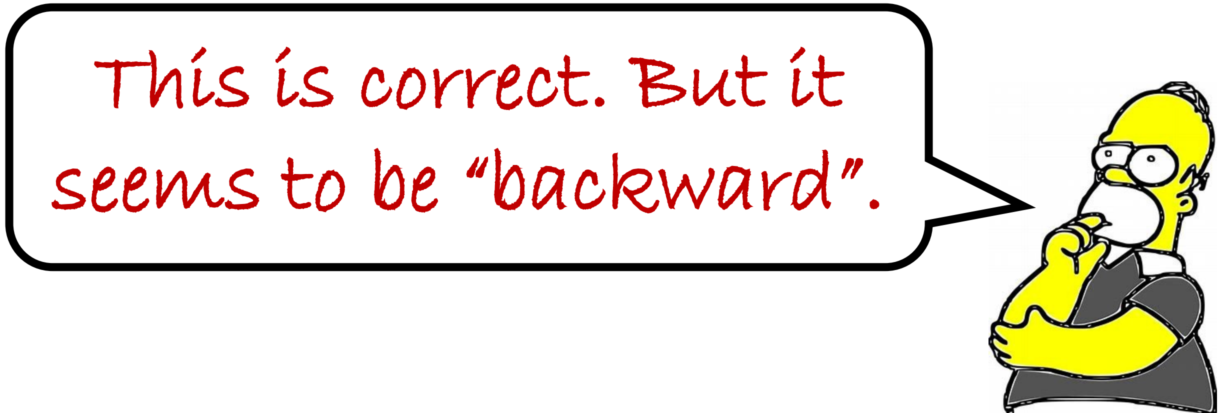
Axioms

$$\{P[e/a]\}a := e\{P\}$$

$$\{b + 1 = 42\}a := b + 1\{a = 42\}$$

$$\{42 = 42\}a := 42\{a = 42\}$$

$$\{a - b > 3\}a := a - b\{a > 3\}$$



Axioms

The abbreviation is AS-FW.

$$\{P\}a := e\{(\exists v)(a = e[v/a] \wedge P[v/a])\}$$

- Forward
- Quantifier
- Intuitive

It is more widely used because it's quantifier-free.

The abbreviation is AS.

$$\{P[e/a]\}a := e\{P\}$$

- Backward
- Quantifier-free
- Not intuitive

Axioms

$$\{a = 1\}a := a + 1\{(\exists v)(a = (a+1)[v/a] \wedge (a = 1)[v/a])\}$$

$$\Downarrow$$

$$\{a = 1\}a := a + 1\{(\exists v)(a = v + 1 \wedge v = 1)\}$$

$$\Downarrow$$

$$\{a = 1\}a := a + 1\{a = 2\}$$

The simplification is correct.

But Hoare logic is an axiom system. We need some rules to do this simplification.

Inference Rules

strong $P \to Q$ weak

$$\frac{\{P\}C\{Q\} \quad Q \to R}{\{P\}C\{R\}}$$

The line means inference.

weakening consequent(WC)

$$\frac{\begin{array}{c}\{a = 1\}a := a + 1\{(\exists v)(a = (a+1)[v/a] \wedge (a = 1)[v/a])\} \\ (\exists v)(a = (a+1)[v/a] \wedge (a = 1)[v/a]) \to a = 2\end{array}}{\{a = 1\}a := a + 1\{a = 2\}}$$

Inference Rules

strong $P \to Q$ weak

$$\frac{P \to Q \quad \{Q\}C\{R\}}{\{P\}C\{R\}}$$

We can also strengthen precedent(SP).

$\{a = n\}a := a + 1\{a = n + 1\}$

Proof.

Inference Rules

strong $P \rightarrow Q$ weak

$$\frac{P \rightarrow Q \quad \{Q\}C\{R\}}{\{P\}C\{R\}}$$

We can also strengthen precedent(SP).

$\{a = n\}a := a + 1\{a = n + 1\}$

$Proof.$

$(1) a = n \rightarrow a + 1 = n + 1$ $\quad$ *predicate logic*

Inference Rules

strong $P \to Q$ weak

$$\frac{P \to Q \quad \{Q\}C\{R\}}{\{P\}C\{R\}}$$

We can also strengthen precedent(SP).

$\{a = n\}a := a + 1\{a = n + 1\}$

Proof.

$(1) a = n \to a + 1 = n + 1$ *predicate logic*

$(2) \{a + 1 = n + 1\}a := a + 1\{a = n + 1\}$ *AS*

Inference Rules

strong $P \to Q$ weak

$$\frac{P \to Q \quad \{Q\}C\{R\}}{\{P\}C\{R\}}$$

We can also strengthen precedent(SP).

$\{a = n\}a := a + 1\{a = n + 1\}$

Proof.

$(1) a = n \to a + 1 = n + 1$ *predicate logic*

$(2) \{a + 1 = n + 1\}a := a + 1\{a = n + 1\}$ *AS*

$(3) \{a = n\}a := a + 1\{a = n + 1\}$ *SP* (1)(2)

Inference Rules

Program structure

1. Sequence : s1; s2

2. Branch : if(b1) { s1 } then { s2 }

3. Loop structure : while(b1) { s }

We will introduce inference rules for these structures.

This is what we want the most.

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1; C2\{Q\}}$$

The sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

$Proof.$

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1;C2\{Q\}}$$

The sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

$Proof.$

$(1)\{a - b \geq 4\}a := a - b\{a \geq 4\}$ AS

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1;C2\{Q\}}$$

The sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

$Proof.$

$(1)\{a - b \geq 4\}a := a - b\{a \geq 4\}$ AS

$(2)\{2 * b - b \geq 4\}a := 2 * b\{a - b \geq 4\}$ AS

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1;C2\{Q\}}$$

The sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

Proof.

$(1)\{a - b \geq 4\}a := a - b\{a \geq 4\}$ AS

$(2)\{2 * b - b \geq 4\}a := 2 * b\{a - b \geq 4\}$ AS

$(3) b > 3 \rightarrow 2 * b - b \geq 4$ *predicate logic*

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1;C2\{Q\}}$$

The sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

Proof.

$(1)\{a - b \geq 4\}a := a - b\{a \geq 4\}$	AS
$(2)\{2 * b - b \geq 4\}a := 2 * b\{a - b \geq 4\}$	AS
$(3)b > 3 \rightarrow 2 * b - b \geq 4$	*predicate logic*
$(4)\{b > 3\}a := 2 * b\{a - b \geq 4\}$	$SP(2)(3)$

Inference Rules

$$\frac{\{P\}C1\{R\} \quad \{R\}C2\{Q\}}{\{P\}C1;C2\{Q\}}$$

the sequential composition rule(SC)

$\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$

Proof.

$(1)\{a - b \geq 4\}a := a - b\{a \geq 4\}$	AS
$(2)\{2 * b - b \geq 4\}a := 2 * b\{a - b \geq 4\}$	AS
$(3) b > 3 \to 2 * b - b \geq 4$	*predicate logic*
$(4)\{b > 3\}a := 2 * b\{a - b \geq 4\}$	$SP(2)(3)$
$(5)\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$	$SC(1)(4)$

It's clearly proved backwards.

Inference Rules

Convert b to a proposition.

Convert b to a proposition.

$$\frac{\{P \wedge istrue(b1)\}C1\{Q\} \quad \{P \wedge isfalse(b1)\}C2\{Q\}}{\{P\}if(b1)\ then\ C1\ else\ C2\ \{Q\}}$$

The conditional rule (CD)

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a,b)\}$$

$Proof.$

$$(1)\{b = max(a,b)\}c := b\{c = max(a,b)\} \qquad AS$$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

$$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\} \qquad AS$$

$$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\} \qquad AS$$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

$$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\} \qquad AS$$

$$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\} \qquad AS$$

$$(3)T \wedge a < b \rightarrow b = max(a, b) \qquad predicate\ logic$$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\}$ $\quad AS$

$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\}$ $\quad AS$

$(3)T \wedge a < b \rightarrow b = max(a, b)$ $\quad predicate\ logic$

$(4)T \wedge \neg(a < b) \rightarrow a = max(a, b)$ $\quad predicate\ logic$

Inference Rules

$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$

$Proof.$

$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\}$ AS

$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\}$ AS

$(3)T \wedge a < b \to b = max(a, b)$ $predicate\ logic$

$(4)T \wedge \neg(a < b) \to a = max(a, b)$ $predicate\ logic$

$(5)\{T \wedge a < b\}c := b\{c = max(a, b)\}$ $SP(1)(3)$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\}$ $\qquad AS$

$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\}$ $\qquad AS$

$(3)T \wedge a < b \rightarrow b = max(a, b)$ $\qquad predicate\ logic$

$(4)T \wedge \neg(a < b) \rightarrow a = max(a, b)$ $\qquad predicate\ logic$

$(5)\{T \wedge a < b\}c := b\{c = max(a, b)\}$ $\qquad SP(1)(3)$

$(6)\{T \wedge \neg(a < b)\}c := a\{c = max(a, b)\}$ $\qquad SP(2)(4)$

Inference Rules

$$\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$$

$Proof.$

$(1)\{b = max(a, b)\}c := b\{c = max(a, b)\}$ $\qquad AS$

$(2)\{a = max(a, b)\}c := a\{c = max(a, b)\}$ $\qquad AS$

$(3)T \wedge a < b \rightarrow b = max(a, b)$ $\qquad predicate\ logic$

$(4)T \wedge \neg(a < b) \rightarrow a = max(a, b)$ $\qquad predicate\ logic$

$(5)\{T \wedge a < b\}c := b\{c = max(a, b)\}$ $\qquad SP(1)(3)$

$(6)\{T \wedge \neg(a < b)\}c := a\{c = max(a, b)\}$ $\qquad SP(2)(4)$

$(7)\{T\}\ if(a < b)\ then\ c := b\ else\ c := a\{c = max(a, b)\}$ $\qquad CD(5)(6)$

Inference Rules

The while rule (WHP)

loop invariant

$$\frac{\{I \wedge istrue(b1)\} C \{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

It says that

- if executing C once preserves the truth of `I`, then executing C any number of times also preserves the truth of `I`
- after a while loop has terminated, the test must be false

Inference Rules

The while rule (WHP)

loop invariant

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

It can be proved by induction on number of loop times

- If the loop execute for 0 time, `I` of course holds.
- If `I` holds for n times loops, it holds for n+1 times because of the condition above the line.

Inference Rules

$$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$$

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

Proof.

Inference Rules

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$$

$Proof.$

$(1)\{a + 1 \leq 10\}a := a + 1\{a \leq 10\}$ $\qquad AS$

loop invariant

Inference Rules

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$$

$Proof.$

$(1)\{a + 1 \leq 10\}a := a + 1\{a \leq 10\}$ $\quad AS$

$(2)a \leq 10 \wedge a \neq 10 \rightarrow a + 1 \leq 10$ $\quad predicate\ logic$

Inference Rules

$$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$$

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$Proof.$

$(1)\{a + 1 \leq 10\}a := a + 1\{a \leq 10\}$ $\quad AS$

$(2)a \leq 10 \wedge a \neq 10 \rightarrow a + 1 \leq 10$ $\quad predicate\ logic$

$(3)\{a \leq 10 \wedge a \neq 10\}a := a + 1\{a \leq 10\}$ $\quad SP(1)(2)$

loop invariant
always holds.

Inference Rules

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$

$Proof.$

$(1)\{a + 1 \leq 10\}a := a + 1\{a \leq 10\}$ $\quad AS$

$(2)a \leq 10 \wedge a \neq 10 \rightarrow a + 1 \leq 10$ $\quad predicate\ logic$

$(3)\{a \leq 10 \wedge a \neq 10\}a := a + 1\{a \leq 10\}$ $\quad SP(1)(2)$

$(4)\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a \leq 10 \wedge a = 10\}$ $\quad WHP(3)$

Inference Rules

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$$

$Proof.$

$(1)\{a + 1 \leq 10\}a := a + 1\{a \leq 10\}$	AS
$(2)a \leq 10 \wedge a \neq 10 \rightarrow a + 1 \leq 10$	$predicate\ logic$
$(3)\{a \leq 10 \wedge a \neq 10\}a := a + 1\{a \leq 10\}$	$SP(1)(2)$
$(4)\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a \leq 10 \wedge a = 10\}$	$WHP(3)$
$(5)a \leq 10 \wedge a = 10 \rightarrow a = 10$	$predicate\ logic$
$(6)\{a \leq 10\}\ while(a! = 10)\ a := a + 1\ \{a = 10\}$	$WC(4)(5)$

Exercise

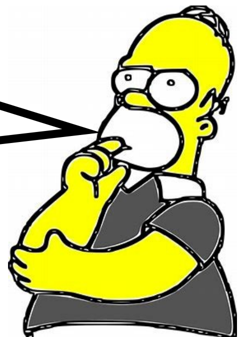
$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

```
void f(unsigned a)
{
    unsigned i = 0;
    {i = 0 ∧ a ≥ 0}
    while(i < a)
    {
        i = i + 1;
    }
    {i = a}
}
```

pre-condition: $\{i = 0 \wedge a \geq 0\}$

post-condition: $\{i = a\}$

What is the loop invariant?



Exercise

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

```
void f(unsigned a)
{
    unsigned i = 0;
```

$\{i = 0 \wedge a \geq 0\}$ ← pre-condition

```
    while(i < a)
    {
        i = i + 1;
    }
```

$\{i = a\}$ ← post-condition

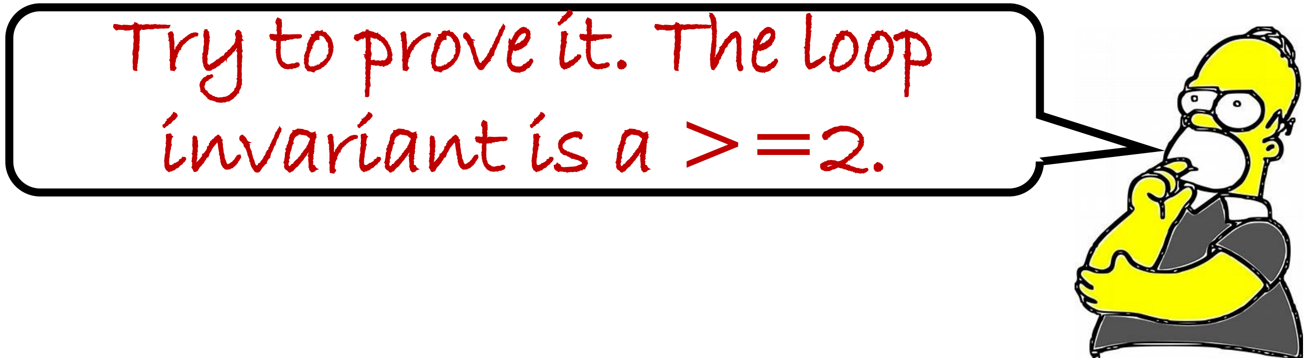
```
}
```

$$I : i \geq 0 \wedge i \leq a \wedge a \geq 0$$

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

Example

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$



Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\} \qquad AS$$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$ $\quad AS$

$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$ $\quad predicate \quad logic$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\} \qquad AS$$

$$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2) \qquad predicate \quad logic$$

$$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\} \qquad SP(1)(2)$$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\} \qquad AS$$

$$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2) \qquad predicate \quad logic$$

$$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\} \qquad SP(1)(2)$$

$$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\} \qquad WHP(3)$$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$ $\quad AS$

$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$ $\quad predicate \quad logic$

$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$ $\quad SP(1)(2)$

$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$ $\quad WHP(3)$

$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$ $\quad predicate \quad logic$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$	AS
$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$	$predicate \quad logic$
$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$	$SP(1)(2)$
$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$	$WHP(3)$
$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$	$predicate \quad logic$
$(6)\{a \geq 2\}while(a > 2)a := a - 1; \{a > 1\}$	$WC(4)(5)$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$ $\qquad AS$

$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$ $\qquad predicate \quad logic$

$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$ $\qquad SP(1)(2)$

$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$ $\qquad WHP(3)$

$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$ $\qquad predicate \quad logic$

$(6)\{a \geq 2\}while(a > 2)a := a - 1; \{a > 1\}$ $\qquad WC(4)(5)$

$(7)\{b \geq 2\}a := b\{a \geq 2\}$ $\qquad AS$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$	AS
$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$	$predicate \quad logic$
$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$	$SP(1)(2)$
$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$	$WHP(3)$
$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$	$predicate \quad logic$
$(6)\{a \geq 2\}while(a > 2)a := a - 1; \{a > 1\}$	$WC(4)(5)$
$(7)\{b \geq 2\}a := b\{a \geq 2\}$	AS
$(8)b \geq 20 \rightarrow b \geq 2$	$predicate \quad logic$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$	AS
$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$	$predicate \quad logic$
$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$	$SP(1)(2)$
$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$	$WHP(3)$
$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$	$predicate \quad logic$
$(6)\{a \geq 2\}while(a > 2)a := a - 1; \{a > 1\}$	$WC(4)(5)$
$(7)\{b \geq 2\}a := b\{a \geq 2\}$	AS
$(8)b \geq 20 \rightarrow b \geq 2$	$predicate \quad logic$
$(9)\{b \geq 20\}a := b\{a \geq 2\}$	$SP(7)(8)$

Example

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$$

$(1)\{a - 1 \geq 2\}a := a - 1; \{a \geq 2\}$	AS
$(2)(a \geq 2 \wedge a > 2) \rightarrow (a - 1 \geq 2)$	$predicate \quad logic$
$(3)\{a \geq 2 \wedge a > 2\}a := a - 1; \{a \geq 2\}$	$SP(1)(2)$
$(4)\{a \geq 2\}while(a > 2)a := a - 1; \{a \geq 2 \wedge a \leq 2\}$	$WHP(3)$
$(5)(a \geq 2 \wedge a \leq 2) \rightarrow (a > 1)$	$predicate \quad logic$
$(6)\{a \geq 2\}while(a > 2)a := a - 1; \{a > 1\}$	$WC(4)(5)$
$(7)\{b \geq 2\}a := b\{a \geq 2\}$	AS
$(8)b \geq 20 \rightarrow b \geq 2$	$predicate \quad logic$
$(9)\{b \geq 20\}a := b\{a \geq 2\}$	$SP(7)(8)$
$(10)\{b \geq 20\}a := b; while(a > 2)a := a - 1; \{a > 1\}$	$SC(6)(9)$

Partial Correctness

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

The loop invariant is "true"

$\{\textbf{true}\}\ \textbf{while}\ x \neq 10\ \textbf{do skip}\ \{x = 10\}$

Proof:

1. $\{\textbf{true} \wedge x \neq 10\}\ \textbf{skip}\ \{\textbf{true} \wedge x \neq 10\}$ — SK
2. $\textbf{true} \wedge x \neq 10 \Rightarrow \textbf{true}$
3. $\{\textbf{true} \wedge x \neq 10\}\ \textbf{skip}\ \{\textbf{true}\}$ — WC, 1, 2
4. $\{\textbf{true}\}\ \textbf{while}\ x \neq 10\ \textbf{do skip}\ \{\textbf{true} \wedge \neg(x \neq 10)\}$ — WHP, 3
5. $\textbf{true} \wedge \neg(x \neq 10) \Rightarrow x = 10$
6. $\{\textbf{true}\}\ \textbf{while}\ x \neq 10\ \textbf{do skip}\ \{x = 10\}$ — WC, 4, 5

Partial Correctness

$$\frac{\{I \wedge istrue(b1)\} C \{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

The loop invariant is "true"

{**true**} **while** $x \neq 10$ **do skip** $\{x = 10\}$

Proof:

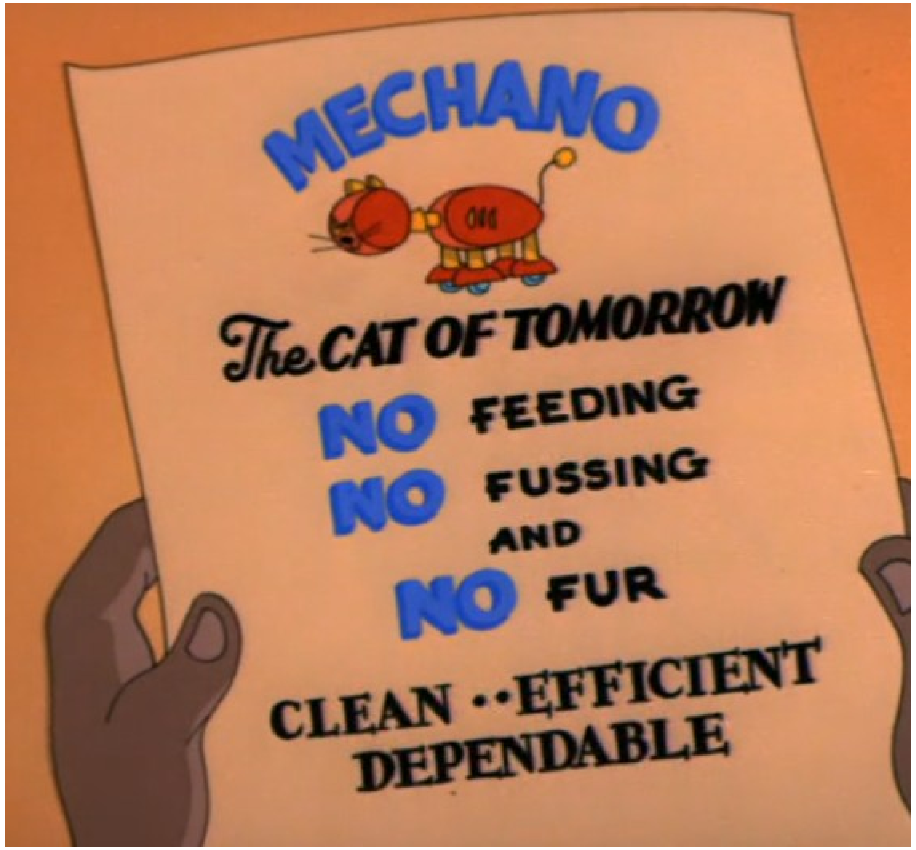
1. {**true** $\wedge\ x \neq 10$} **skip** {**true** $\wedge\ x \neq 10$} — SK
2. **true** $\wedge\ x \neq 10 \Rightarrow$ **true**
3. {**true** $\wedge\ x \neq 10$} **skip** {**true**} — WC, 1, 2
4. {**true**} **while** $x \neq 10$ **do skip** {**true** $\wedge \neg(x \neq 10)$} — WHP, 3

The proof does not ensure that the program can always terminate.

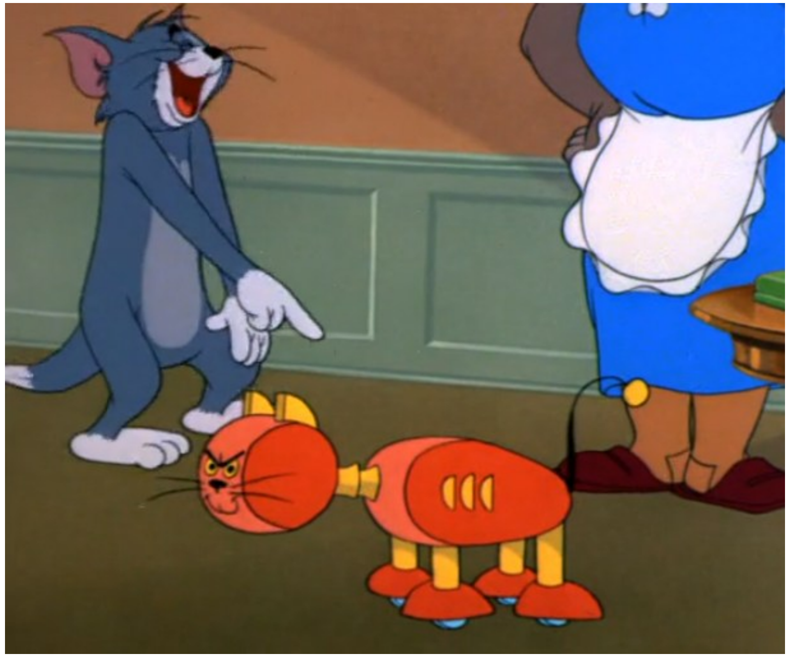
6. {**true**} **while** $x \neq 10$ **do skip** $\{x = 10\}$

However,

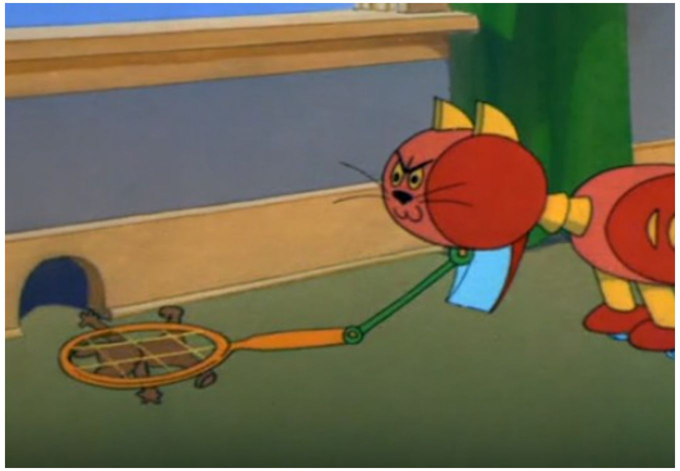
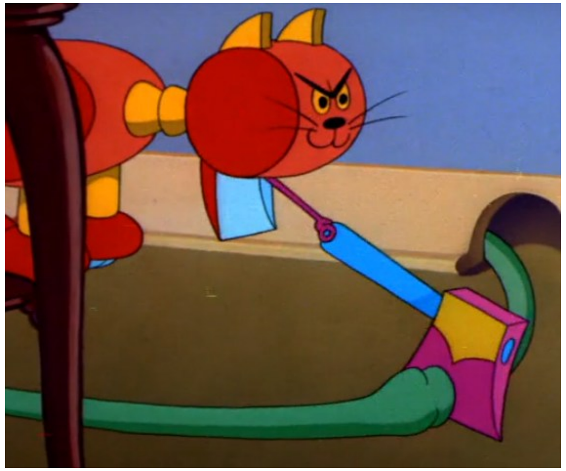
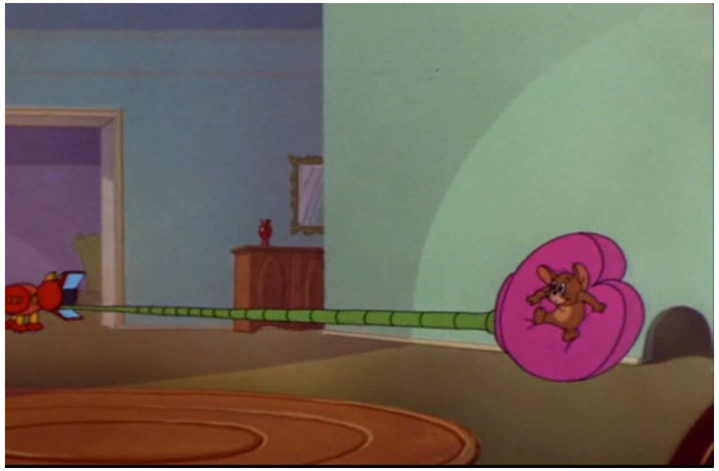
Manual Proof is usually tedious



We want an automated program verifier.



But previous methods (symbolic execution) cannot handle loops

What if we keep all invariants/conditions in the proof?

Automatic Proof, Again???

Predicate Transformer

stronger $P \to Q$ weaker

DEFINITION

Given a program C and postcondition Q, P is the weakest precondition means for all P', {P'} C {Q} iff P' → P. We use wp(C, Q) to denote P.

$$\{a = 2\} a := a + 1 \{a = 3\}$$

weakest precondition

$$\{a = 2 \wedge b = 4\} a := a + 1 \{a = 3\} \qquad a = 2 \wedge b = 4 \to a = 2$$

$$\{a = 2 \wedge a > 1\} a := a + 1 \{a = 3\} \qquad a = 2 \wedge a > 1 \to a = 2$$

Predicate Transformer

stronger $P \rightarrow Q$ weaker

DEFINITION

Given a program C and precondition P, Q is the strongest postcondition means for all Q', {P} C {Q'} iff Q → Q'. We use sp(C, P) to denote Q.

$\{a = 1\} a := 3 \{a = 3\}$ strongest postcondition

$\{a = 1\} a := 3 \{T\}$ $\qquad a = 3 \rightarrow T$

$\{a = 1\} a := 3 \{(\exists v)(a = v)\}$ $\qquad a = 3 \rightarrow (\exists v)(a = v)$

Predicate Transformer

We can consider the problem from the following perspective:

- Given a precondition and program, can we find the strongest postcondition?

- Or given a postcondition and program, can we find the weakest precondition?

- This is called predicate transformer semantics: how programs transform assertions

$$\{P\}C\{Q\}$$

$$P \rightarrow wp(C, Q)$$

$$sp(C, P) \rightarrow Q$$

If we can find wp or sp, we can convert hoare triples to predicate logic formula.

Predicate Transformer

We can consider the problem from the following perspective:

- Given a precondition and program, can we find the strongest postcondition?
- Or given a postcondition and program, can we find the weakest precondition?
- This is called predicate transformer semantics: how programs transform assertions

$$\{P\}C\{Q\}$$

$$P \rightarrow wp(C, Q)$$

We just need to prove it is true.

$$sp(C, P) \rightarrow Q$$

We just need to prove it is true.

Predicate Transformer

We can consider the problem from the following perspective:

- Given a precondition and program, can we find the strongest postcondition?

- Or given a postcondition and program, can we find the weakest precondition?

- This is called predicate transformer semantics: how programs transform assertions

$$\{P\}C\{Q\}$$

$$P \wedge \neg wp(C, Q)$$

We just need to prove it is unsatisfiable.

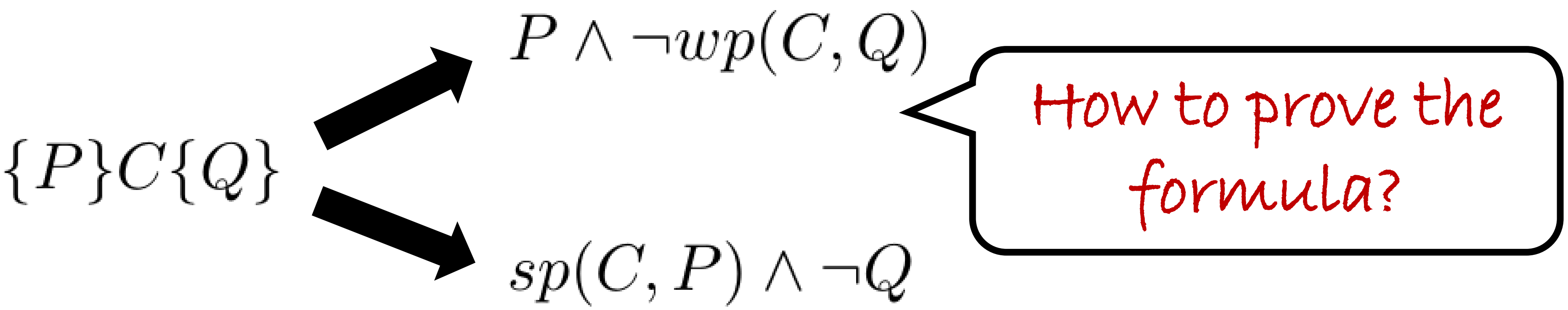
$$sp(C, P) \wedge \neg Q$$

We just need to prove it is unsatisfiable.

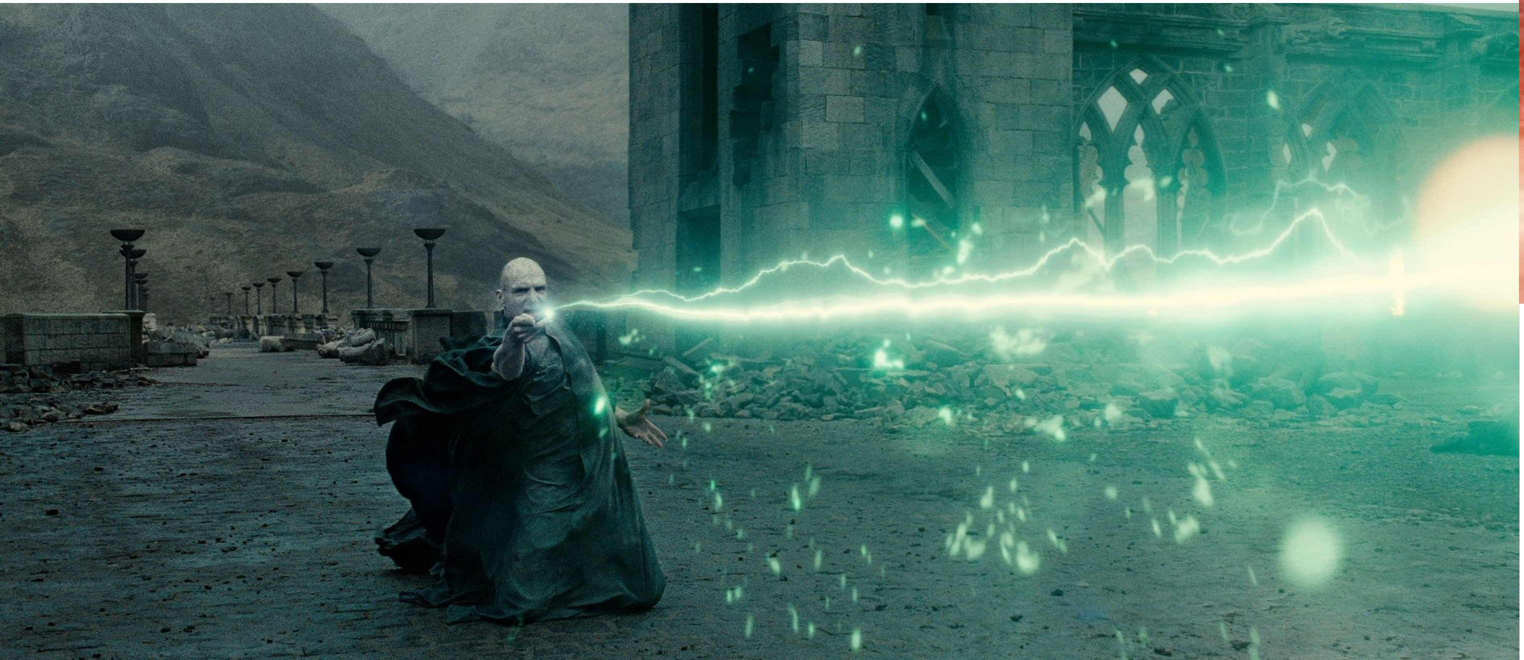
Basic Idea

To prove `{P}C{Q}`

- Find the `wp(C, Q)`, then prove `P` implies `wp(C, Q)`.
- Find the `sp(C, P)`, then prove `sp(C, P)` implies `Q`.



Working with SMT solvers is like magic



Question: How to automatically find the weakest precondition (strongest postcondition)?

By rules (Theorems)…

A Quick Recap

AS-FW gives the strongest postcondition.

$$\{P\}a := e\{(\exists v)(a = e[v/a] \wedge P[v/a])\}$$

- Forward
- Quantifier
- Intuitive

It is more widely used because it's simpler.

AS gives the weakest precondition.

$$\{P[e/a]\}a := e\{P\}$$

- Backward
- Quantifier-free
- Not intuitive

A Quick Recap

AS-FW gives the strongest postcondition.

$$\{P\}a := e\{(\exists v)(a = e[v/a] \land P[v/a])\}$$

AS gives the weakest postcondition.

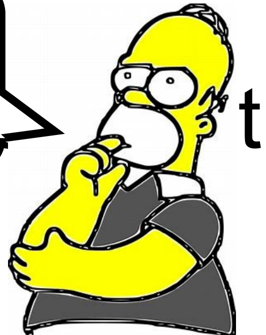
$$\{P[e/a]\}a := e\{P\}$$

- Forward
- Quantifier
- Intuitive

It is more widely used because it's simpler.

- Backward
- Quantifier-free
- tive

We will focus on wp.



Weakest Precondition

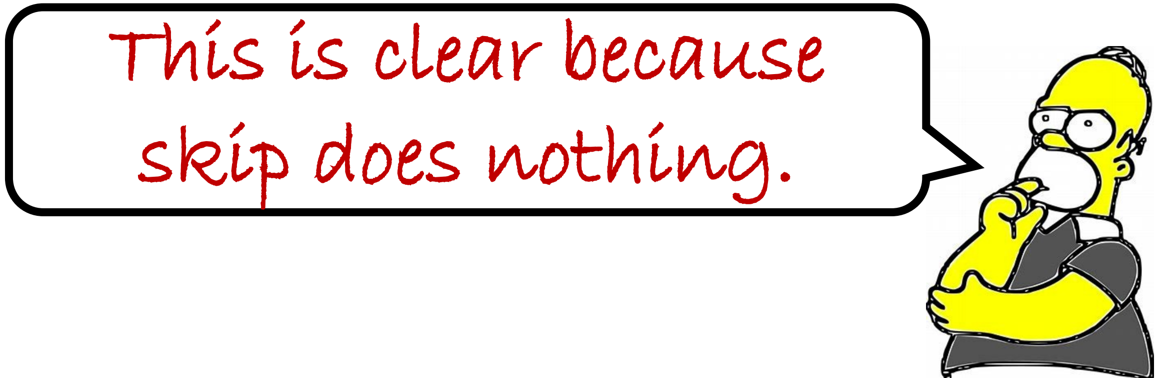
We use wlp(C, Q) instead of wp(C, Q) in following slides:

- C is the program, Q is the intended postcondition
- Name stands for weakest "liberal" precondition: it ignores termination
- For all C and Q, it is guaranteed that { wlp(C, Q) } C {Q} is true
- For all P, C and Q, {P} C {Q} iff $P \rightarrow wlp(C, Q)$

Wlp must be computable. We will introduce the details.

Weakest Precondition

$$wlp(skip, Q) = Q$$



Weakest Precondition

This is similar to AS in Hoare logic.

$$\{P[e/a]\}a := e\{P\}$$

$$wlp(a := e, P) = P[e/a]$$

$$wlp(a := b + 1, a = 42) = b + 1 = 42$$

$$wlp(a := a - b, a > 3) =$$?

Weakest Precondition

This is similar to AS in Hoare logic.

$$\{P[e/a]\}a := e\{P\}$$

$$wlp(a := e, P) = P[e/a]$$

$$wlp(a := b + 1, a = 42) = b + 1 = 42$$

$$wlp(a := a - b, a > 3) = a - b > 3$$

Weakest Precondition

$$wlp(C1; C2, Q) = wlp(C1, wlp(C2, Q))$$

$$wlp(a := a + 1; b := a, a = 3 \wedge b > 0)$$

Weakest Precondition

$$wlp(C1; C2, Q) = wlp(C1, wlp(C2, Q))$$

$$wlp(a := a + 1; b := a, a = 3 \wedge b > 0)$$

$$= wlp(a := a + 1, wlp(b := a, a = 3 \wedge b > 0))$$

$$= wlp(a := a + 1, a = 3 \wedge a > 0)$$

$$= \boxed{a + 1 = 3 \wedge a + 1 > 0}$$

Weakest Precondition

$$wlp(if(b1)\ then\ \{C1\}\ else\ \{C2\}, Q)$$

$$=(istrue(b1) \rightarrow wlp(C1, Q)) \wedge (isfalse(b1) \rightarrow wlp(C2, Q))$$

$$wlp(if(a == 0)\ then\ \{b := 1\}\ else\ \{b := 0\}, b = 1 \vee b = 0)$$

Weakest Precondition

$$wlp(if(b1)\ then\ \{C1\}\ else\ \{C2\}, Q)$$
$$=(istrue(b1) \rightarrow wlp(C1, Q)) \land (isfalse(b1) \rightarrow wlp(C2, Q))$$

$$wlp(if(a == 0)\ then\ \{b := 1\}\ else\ \{b := 0\}, b = 1 \lor b = 0)$$
$$=(a = 0 \rightarrow wlp(b := 1, b = 1 \lor b = 0)) \land (a \neq 0 \rightarrow wlp(b := 0, b = 1 \lor b = 0))$$
$$=(a = 0 \rightarrow 1 = 1 \lor 1 = 0) \land (a \neq 0 \rightarrow 0 = 1 \lor 0 = 0)$$

Exercise

$$a := b + 2;\ \ if(a > 2)\ then\ b := c + 1;\ \ else\ b := c - 1;$$

Assume the post-condition is $b > 5 \wedge a < 10$

Exercise

$$a := b + 2;\ if(a > 2)\ then\ b := c + 1;\ else\ b := c - 1;$$

$$wlp(\quad a := b + 2;\ if(a > 2)\ then\ b := c + 1;\ else\ b := c - 1;,\quad b > 5 \land a < 10)$$

$$= wlp(a := b + 2, ((a > 2) \rightarrow wlp(b := c + 1, b > 5 \land a < 10)) \land (\neg(a > 2) \rightarrow wlp(b := c - 1, b > 5 \land a < 10)))$$

$$= wlp(a := b + 2, ((a > 2) \rightarrow (c + 1 > 5 \land a < 10)) \land (\neg(a > 2) \rightarrow (c - 1 > 5 \land a < 10)))$$

$$= ((b + 2 > 2) \rightarrow (c + 1 > 5 \land b + 2 < 10)) \land (\neg(b + 2 > 2) \rightarrow (c - 1 > 5 \land b + 2 < 10))$$

Weakest Precondition

$$wlp(while(b1)\{C\}, Q) = ?$$

:-) 呵呵

Weakest Precondition

$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

$$\{P\}while(b1)C\{Q\} \quad \begin{cases} P \rightarrow I \\ (I \wedge isfalse(b1)) \rightarrow Q \\ \{I \wedge istrue(b1)\}C\{I\} \end{cases}$$

The loop invariant needs to be written manually

This can be further proved via the weakest precondition

Lab3 : Dafny

```
method func(a : int, b : int) returns (c : int)
  ensures c == 0
{
  var i := a/b;
  return i;
}
```

post-condition

		Description	Line
	1	possible division by zero	4

https://github.com/dafny-lang/dafny

Dafny

post-condition

```
method func(n: nat) returns (result: nat)
  ensures n == result
{
    var i: int := 0;
    while i < n
        invariant i <= n
    {
        i := i + 1;
    }
    return i;
}
```

loop invariant

'►' shortcut: Alt+B

```
Dafny 2.3.0.10506

Dafny program verifier finished with 1 verified, 0 errors
Program compiled successfully
```

Dafny

pre-condition

```
method func(a: array<int>, n: int)
  requires n > 0
  ensures forall k :: 0 <= k < n ==> a[k] == 0
{
  var i : int := 0;
  while i < n
    invariant 0 <= i
    invariant i <= n
    invariant forall k :: 0 <= k < i ==> a[k] == 0
  {
    a[i] := 0;
    i := i + 1;
  }
  return;
}
```

		Description	Line	Column
	1	/!\ No terms found to trigger on.	9	14
⊗	2	index out of range	3	38
⊗	3	index out of range	9	42
⊗	4	assignment may update an array element not in the enclosing context's modifies clause	11	5
⊗	5	index out of range	11	5
⊗	6	A postcondition might not hold on this return path.	14	2
	7	This is the postcondition that might not hold.	3	10

Dafny

pre-condition

```
method func(a: array<int>, n: int)
  requires n > 0 && n <= a.Length
  ensures forall k :: 0 <= k < n ==> a[k] == 0
{
  var i : int := 0;
  while i < n
    invariant 0 <= i
    invariant i <= n && n <= a.Length
    invariant forall k :: 0 <= k < i ==> a[k] == 0
  {
    a[i] := 0;
    i := i + 1;
  }
  return;
}
```

```
Dafny 2.3.0.10506

Dafny program verifier finished with 1 verified, 0 errors
Program compiled successfully
```

A Quick Recap

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

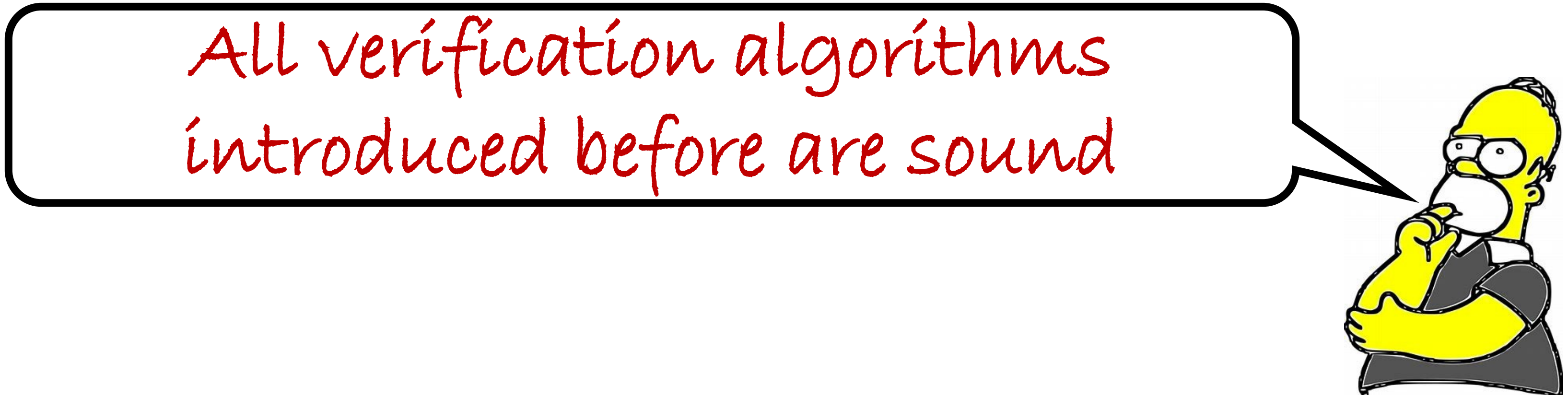
Step 2. Ask the computer to check the invariants

Soundness and Completeness for Verifiers

DEFINITION

A verification algorithm is **sound** means that when a program passes the verification, it is a “correct” program.

Note that we can define the correctness in multiple different ways.



Soundness and Completeness for Verifiers

DEFINITION

A verification algorithm is **complete** means that every correct program can pass the verification.

Previous algorithms are not complete because of two main reasons:

- They may not automatically prove programs with unbounded loops.
- The satisfiability problem of FOL formulas is undecidable.

Thanks & Questions