

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

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



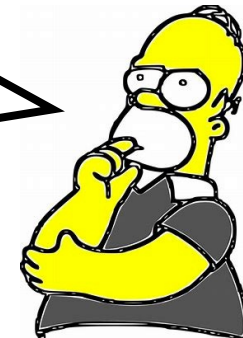
Program Analysis



SMT-based
program analysis

Program Analysis

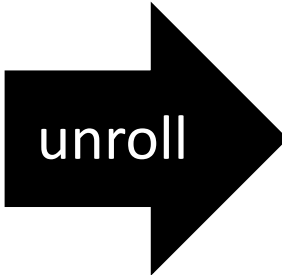
With SMT solvers, how to
convert the program to
predicate logic formulas?



Loops

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

Loops



unroll

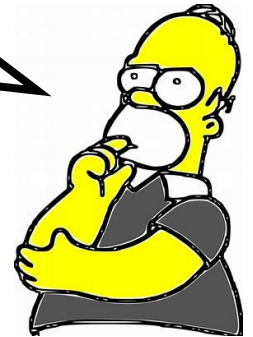
```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
}
```

Straight-Line Code

Loops

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

*How many loops
should we unloop?*



Loops

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

It seems that current
technique doesn't
work. We need more
powerful tools.



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.

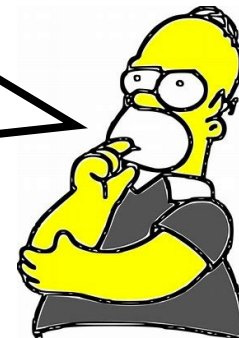
Hoare logic is an axiom system
for program verification.

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

$A, B, C, \dots$

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 $(\neg A)$ and $(A \vee B)$.
- 3). No expression is WFF unless forced by 1) or 2).

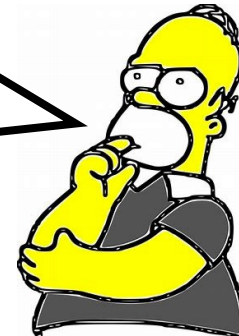
Axiom System(公理系统)

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

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

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

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.

$$\text{Axiom1} \vdash ((P \vee P) \rightarrow P)$$

$$\text{Axiom2} \vdash (P \rightarrow (P \vee Q))$$

$$\text{Axiom3} \vdash ((P \vee Q) \rightarrow (Q \vee P))$$

$$\text{Axiom4} \vdash ((Q \rightarrow R) \rightarrow ((P \vee Q) \rightarrow (P \vee R)))$$

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)

$$\vdash P \vee \neg P \quad \text{---} \quad \frac{P}{(R \vee S)} \quad \text{---} \quad \vdash (R \vee S) \vee \neg(R \vee S)$$

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: $\text{---} \rightarrow$ 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.

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

Proof.

Axiom System(公理系统)

DEFINITION

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

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

Proof.

$$(1) \vdash (Q \rightarrow R) \rightarrow (P \vee Q \rightarrow P \vee R)$$

Axiom4

Axiom System(公理系统)

DEFINITION

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

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

Proof.

$$(1) \vdash (Q \rightarrow R) \rightarrow (P \vee Q \rightarrow P \vee R)$$

Axiom4

$$(2) \vdash (Q \rightarrow R) \rightarrow (\neg P \vee Q \rightarrow \neg P \vee R)$$

$\frac{P}{\neg P}$ on (1)

Axiom System(公理系统)

DEFINITION

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

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

Theorem 1

Proof.

$$(1) \vdash (Q \rightarrow R) \rightarrow (P \vee Q \rightarrow P \vee R)$$

Axiom4

$$(2) \vdash (Q \rightarrow R) \rightarrow (\neg P \vee Q \rightarrow \neg P \vee R)$$

$\frac{P}{\neg P}$ on (1)

$$(3) \vdash (Q \rightarrow R) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$$

definition of $\rightarrow$ on(2)

Axiom System(公理系统)

$\vdash P \rightarrow P$

Theorem 2

Proof.

(1) $\vdash P \rightarrow P \vee Q$

(2) $\vdash P \rightarrow P \vee P$

(3) $\vdash P \vee P \rightarrow P$

(4) $\vdash (Q \rightarrow R) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$

(5) $\vdash ((P \vee P) \rightarrow R) \rightarrow ((P \rightarrow (P \vee P)) \rightarrow (P \rightarrow R))$

(6) $\vdash ((P \vee P) \rightarrow P) \rightarrow ((P \rightarrow (P \vee P)) \rightarrow (P \rightarrow P))$

(7) $\vdash (P \rightarrow (P \vee P)) \rightarrow (P \rightarrow P)$

(8) $\vdash P \rightarrow P$

Axiom 2

Replace Q with P in (1)

Axiom 1

Theorem 1

Replace Q with $P \vee P$ in (4)

Replace R with P in (5)

Hypothetical reasoning on (3) (6)

Hypothetical reasoning on (2) (7)

Axiom System(公理系统)

$\vdash P \rightarrow P$

Theorem 2

Proof.

(1) $\vdash P \rightarrow P \vee Q$

(2) $\vdash P \rightarrow P \vee P$

(3) $\vdash P \vee P \rightarrow P$

(4) $\vdash (Q \rightarrow R) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$

(5) $\vdash ((P \vee P) \rightarrow R) \rightarrow ((P \rightarrow (P \vee P)) \rightarrow (P \rightarrow R))$

(6) $\vdash ((P \vee P) \rightarrow P) \rightarrow ((P \rightarrow (P \vee P)) \rightarrow (P \rightarrow P))$

(7) $\vdash (P \rightarrow (P \vee P)) \rightarrow (P \rightarrow P)$

(8) $\vdash P \rightarrow P$

$\vdash \neg P \vee P$

Theorem 3

Proof.

(1) $\vdash P \rightarrow P$

(2) $\vdash \neg P \vee P$

Axiom 2

Replace Q with P in (1)

Axiom 1

Theorem 1

Replace Q with $P \vee P$ in (4)

Replace R with P in (5)

Hypothetical reasoning on (3) (6)

Hypothetical reasoning on (2) (7)

Theorem 2

Definition of $\rightarrow$ on (1)

Completeness and Soundness

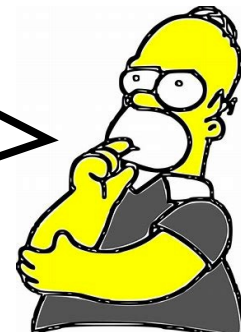
DEFINITION

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:

$$A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

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:

$$A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

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

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

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:

$$A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

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:

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

$\vdash A_i$

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:

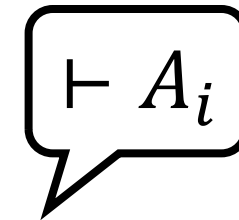
$$A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

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:

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



4. Since each clause is proven, $\vdash A$ can be proved in system. ■

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.

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.

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 $A \rightarrow B$ is tautology, then B must be tautology

Other inference rules
can also be proved

Prove it by listing all
possible situations

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 $A \rightarrow B$ is tautology, then B must be tautology
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.

pre-condition

$x = 1$

$x = x + 1;$

$x = 2$

post-condition

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**.

pre-condition

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

post-condition

program

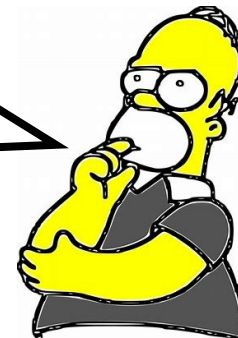
Hoare Triples

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.

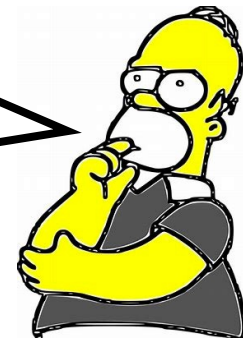


Hoare Triples

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

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

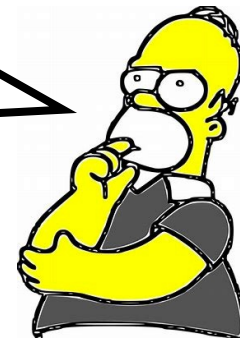
P and Q are predicate logic propositions. What about C ?



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: $e_1 + e_2$, $e_1 - e_2$, ...
 - Boolean expressions: $e_1 > e_2$, $e_1 == e_2$, $e_1 < e_2$, ...
 - We typically write b_1 , b_2 , ... for boolean-typed expressions and e_1 , e_2 , ... for arithmetic expressions

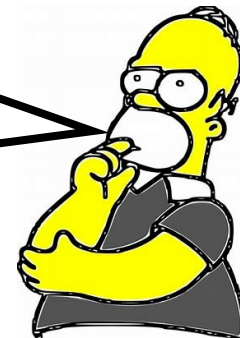
For simplicity, we just consider a simple imperative language.



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 $s2$)
 - $\text{if}(b1) \{s1\} \text{ else } \{s2\}$ (execute $s1$ if b is true; $s2$ otherwise)
 - $\text{While}(b1) \{s\}$ (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...

For simplicity, we just consider a simple imperative language.



Hoare Logic

- Initial symbols
 - symbols in predicate logic : $\forall, \exists, \rightarrow, \neg, \dots$
 - 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

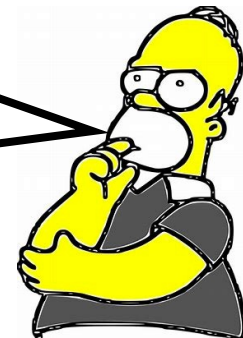
$$\{a = 0\} a := 3 \{a = 3\}$$

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

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

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

They are all Hoare triples. But not all of them are valid.



Hoare Logic

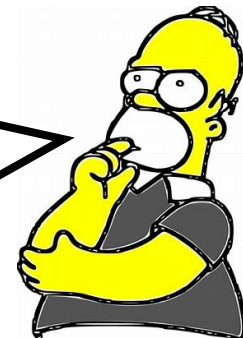
$\{a = 0\} a := 3 \{a = 3\}$ ☒

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

$\{a = 1\} a := 2 \{a = 10\}$ ☐

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

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



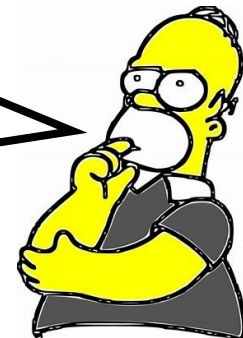
Problem 2:

How to prove program correctness?

Axioms

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

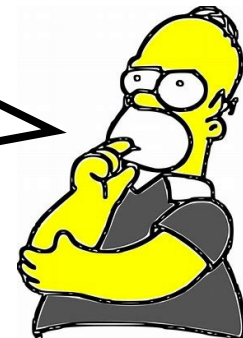
This axiom is clearly true.
Because skip does nothing.



Axioms

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

The assignment axiom
brings troubles.

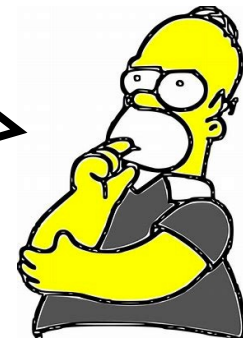


Axioms

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

$$\{a = 0\}a := 1\{a = 0\} \quad \boxed{\times}$$

Is this correct?

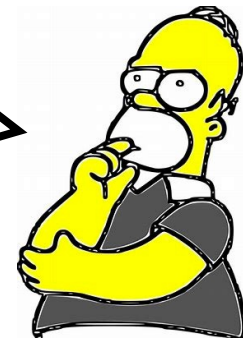


Axioms

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

$$\{a = 5\}a := a + 1\{a = 5 \wedge a = a + 1\} \quad \boxed{\times}$$

Is this correct?



Axioms

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

v is a fresh variable.

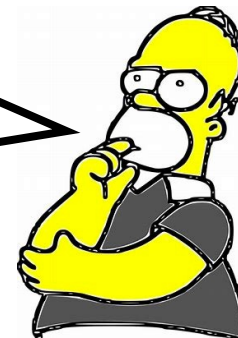
$$\{a = 1\}a := a + 1\{(\exists v)(a = (a + 1)[v/a] \wedge (a = 1)[v/a])\}$$

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

simplification

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

This is correct and a forward assignment axiom.



Axioms

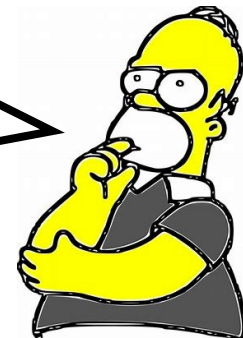
$$\{P[e/a]\}a := e\{P\}$$

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

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

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

This is correct. But it seems to be "backward".



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

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



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



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

The simplification
is correct.

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

Inference Rules

$$\boxed{\text{strong}} \quad P \rightarrow Q \quad \boxed{\text{weak}}$$

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

The line means inference.

weakening consequent (WC)

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

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.

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) $a = n \rightarrow a + 1 = n + 1$

predicate logic

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) $a = n \rightarrow a + 1 = n + 1$

predicate logic

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

AS

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.

$$(1) a = n \rightarrow a + 1 = n + 1$$

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 : $\text{if}(b1) \{ s1 \} \text{ then } \{ s2 \}$
3. Loop structure : $\text{while}(b1) \{ s \}$

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

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

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

Proof.

Inference Rules

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

Proof.

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

AS

Inference Rules

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

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*

Inference Rules

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

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

Inference Rules

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

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

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

Inference Rules

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

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

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

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

Inference Rules

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

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

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

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

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

Inference Rules

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

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

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

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

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

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

Inference Rules

Loop invariant

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

The while
rule (WHP)

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

Loop invariant

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

The while
rule (WHP)

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

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

Proof.

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

Inference Rules

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

Proof.

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

loop invariant

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

AS

Inference Rules

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

Proof.

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

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

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

AS

predicate logic

Inference Rules

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

Proof.

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

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

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

loop invariant
always holds.

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

AS

predicate logic

SP(1)(2)

Inference Rules

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

Proof.

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

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

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

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

Inference Rules

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

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

Proof.

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

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

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

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

$$(5) a \leq 10 \wedge a = 10 \rightarrow a = 10 \quad predicate\ logic$$

$$(6) \{a \leq 10\} while(a \neq 10) a := a + 1 \{a = 10\} \quad WC(4)(5)$$

Inference Rules

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

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

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) $\{a \leq 10\} \text{ while}(a \neq 10) \ a := a + 1 \ \{a \leq 10 \wedge a = 10\}$

WHP(3)

(5) $a \leq 10 \wedge a = 10 \rightarrow a = 10$

predicate logic

(6) $\{a \leq 10\} \text{ while}(a \neq 10) \ a := a + 1 \ \{a = 10\}$

WC(4)(5)

How to find the
loop invariant?



How to find loop invariant ?

There is no automatic algorithm to generate loop invariant

You must find it by yourself. The invariant should say that:

- what has been done so far together with what remains to be done
- holds at each iteration of the loop
- and gives the desired result when the loop terminates

Some researchers try to generate invariant via machine learning

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

post-condition

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

loop invariant

Exercise

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

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

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

- (1) $\{(i + 1 \geq 0) \wedge (i + 1 \leq a) \wedge (a \geq 0)\} i := i + 1 \{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0)\}$
- (2) $((i \geq 0) \wedge (i \leq a) \wedge (a \geq 0) \wedge (i < a)) \rightarrow ((i + 1 \geq 0) \wedge (i + 1 \leq a) \wedge (a \geq 0))$
- (3) $\{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0) \wedge (i < a)\} i := i + 1 \{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0)\}$
- (4) $\{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0)\} while(i < a) i := i + 1; \{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0) \wedge \neg(i < a)\}$
- (5) $((i \geq 0) \wedge (i \leq a) \wedge (a \geq 0) \wedge \neg(i < a)) \rightarrow i = a$
- (6) $\{(i \geq 0) \wedge (i \leq a) \wedge (a \geq 0) \wedge (i < a)\} while(i < a) i := i + 1; \{i = a\}$

AS

Predicate Logic

SP(1)(2)

WHP(3)

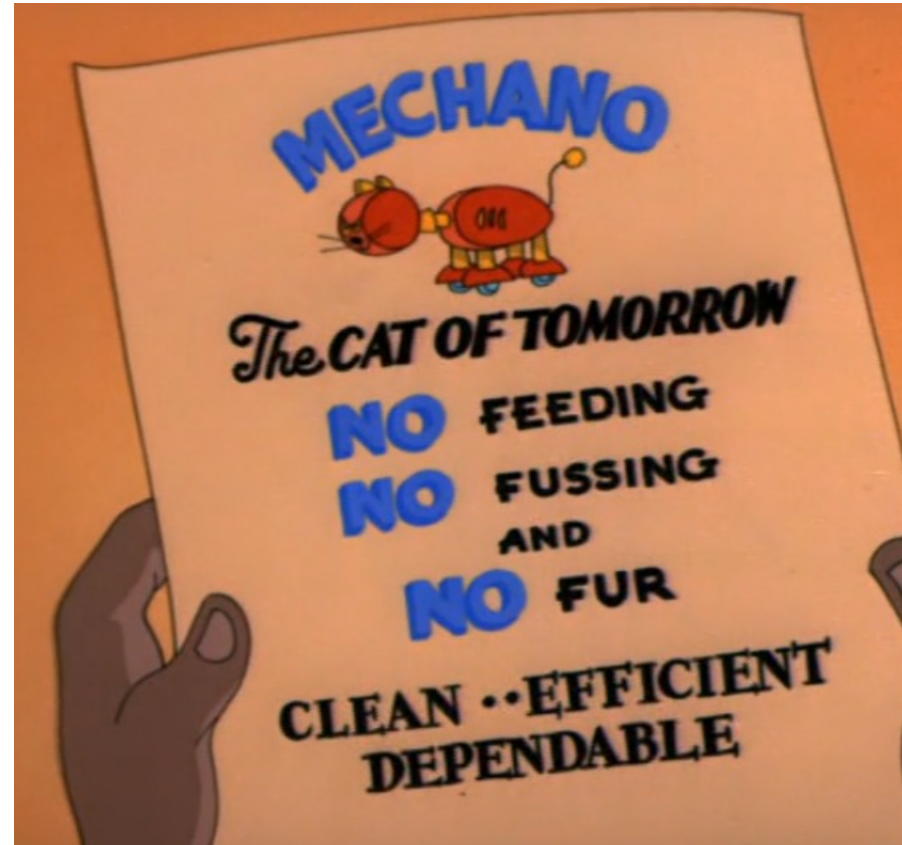
Predicate Logic

WC(4)(5)

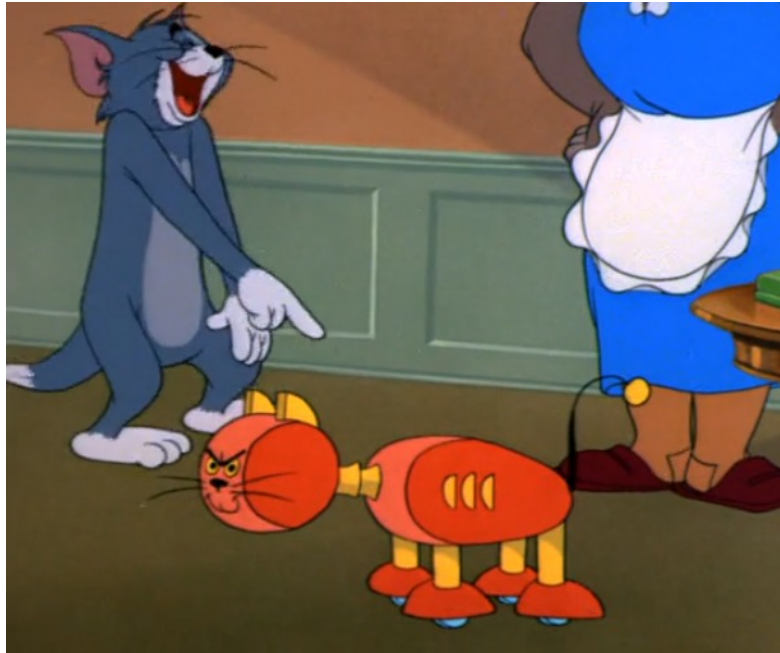
However,



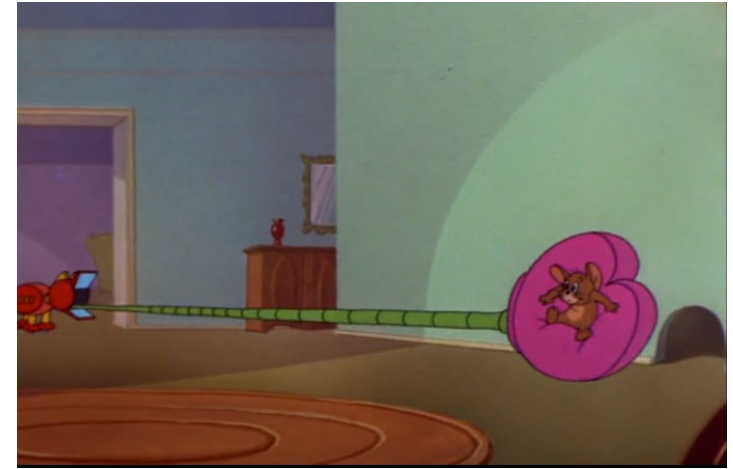
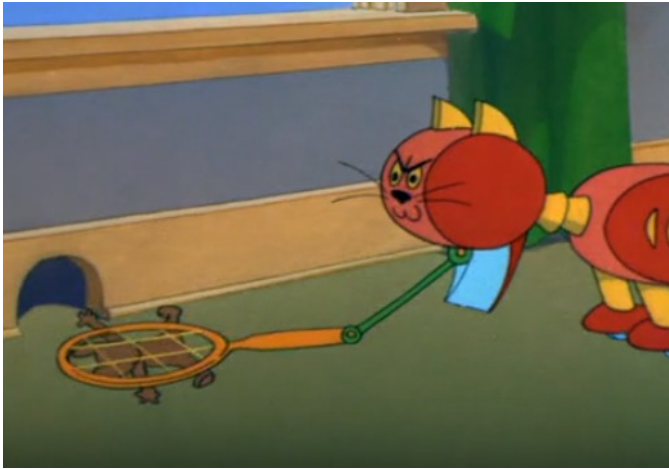
Manual Proof is
usually tedious



We want an automated program verifier.



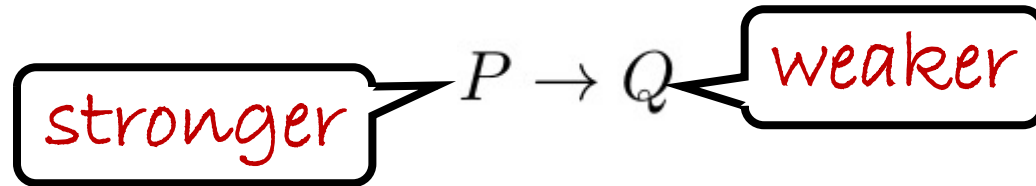
But previous methods (symbolic execution) cannot handle loops



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

Automatic Proof, Again???

Predicate Transformer



DEFINITION

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

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

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

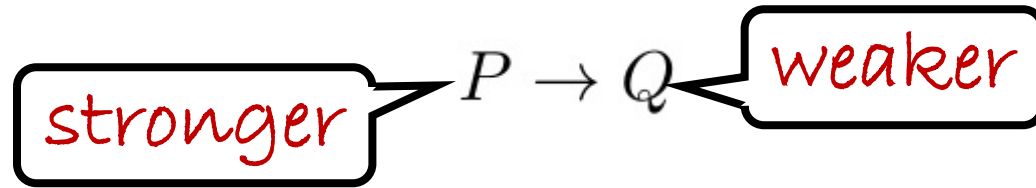
$$\{a = 2 \wedge a > 1\} a := a + 1 \{a = 3\}$$

weakest
precondition

$$a = 2 \wedge b = 4 \rightarrow a = 2$$

$$a = 2 \wedge a > 1 \rightarrow a = 2$$

Predicate Transformer



DEFINITION

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

A diagram showing the relationship between two predicates, P and Q , connected by an arrow pointing from P to Q . A callout box labeled "strongest postcondition" points to the arrow. The label is in red, handwritten-style text.

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

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

$a = 3 \rightarrow T$

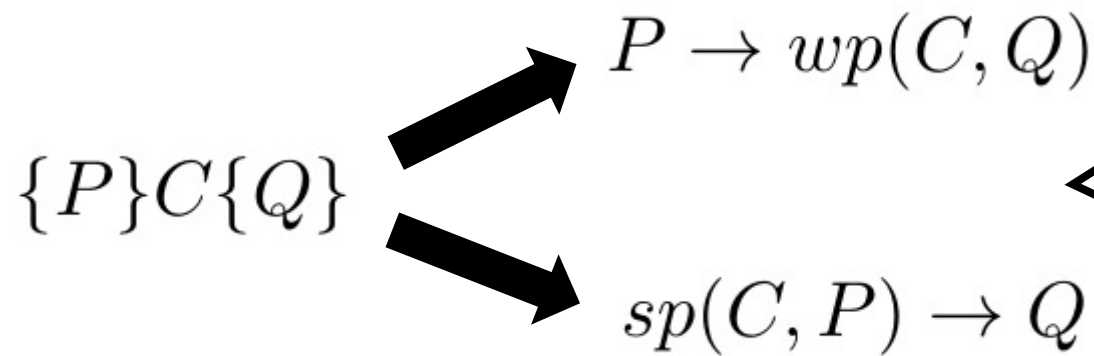
$\{a = 1\} a := 3 \{(\exists v)(a = v)\}$

$a = 3 \rightarrow (\exists v)(a = v)$

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

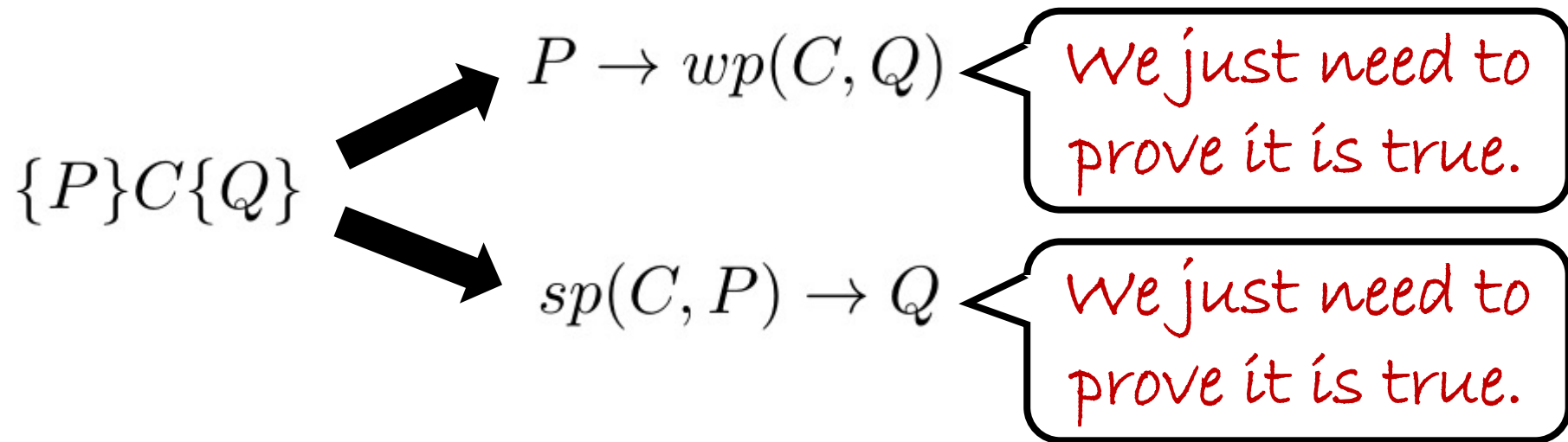


If we can find wp or sp,
we can convert hoare
triples to predicate
logic formula.

Predicate Transformer

We can consider the problem from the following perspective:

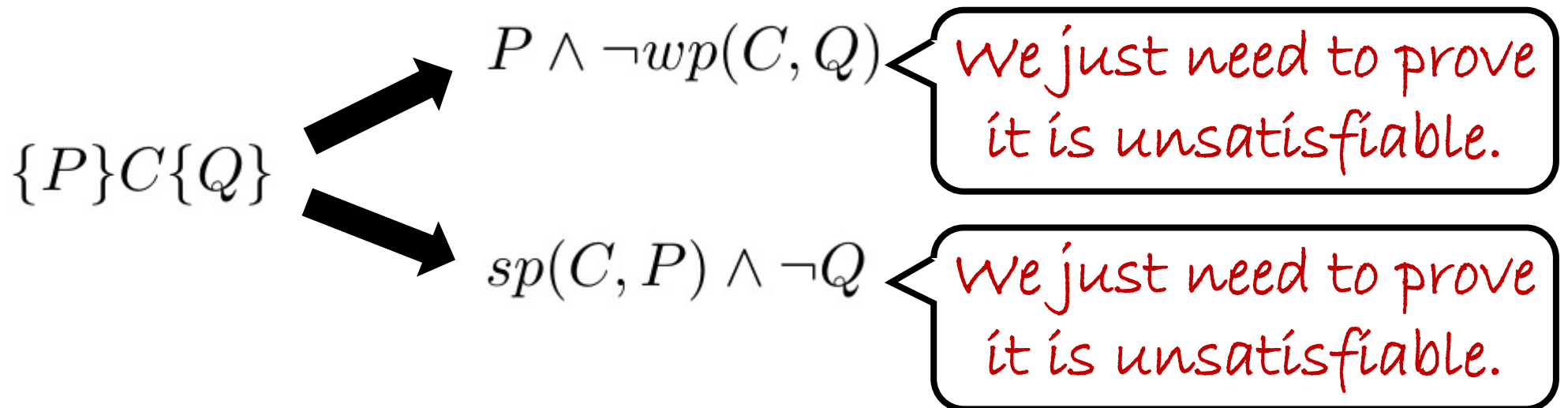
- Given a precondition and program, can we find the strongest postcondition?
- Or given a postcondition and program, can we find the weakest precondition?
- This is called **predicate transformer** semantics: how programs transform assertions



Predicate Transformer

We can consider the problem from the following perspective:

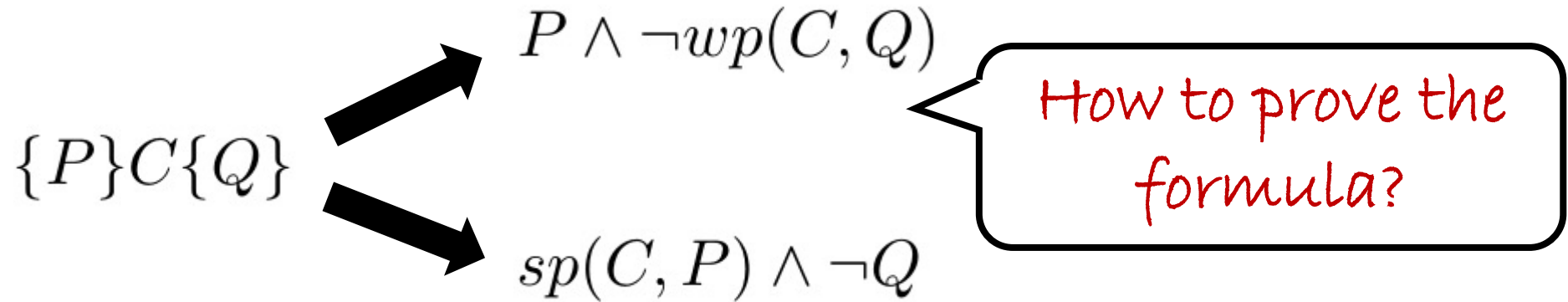
- Given a precondition and program, can we find the strongest postcondition?
- Or given a postcondition and program, can we find the weakest precondition?
- This is called **predicate transformer** semantics: how programs transform assertions



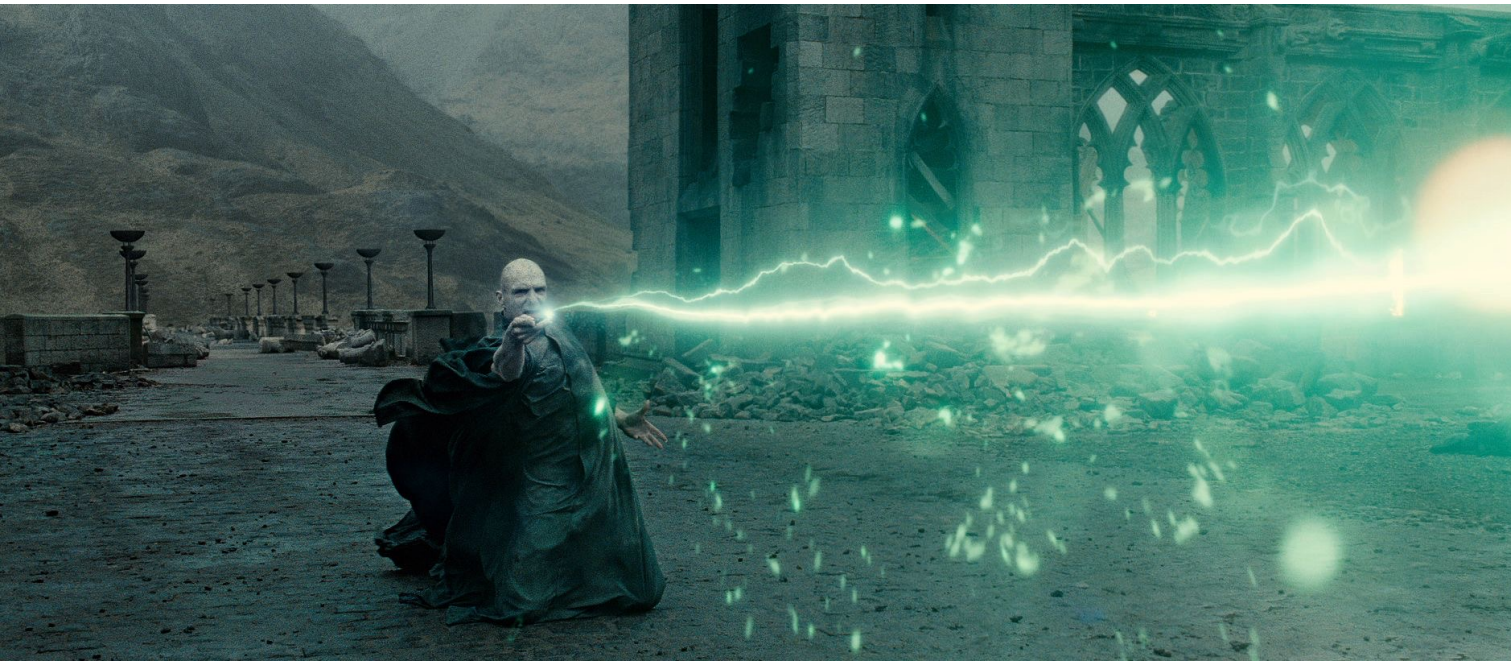
Basic Idea

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

AS gives the weakest precondition.

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

It is more widely used because it's simpler.

- Backward
- Quantifier-free
- Not intuitive

A Quick Recap

AS-FW gives the strongest postcondition.

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

AS gives the weakest precondition.

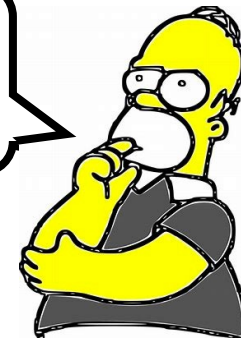
$$\{P[e/a]\}a := e \{P\}$$

- Forward
- Quantifier
- Intuitive

It is more widely used because it's simpler.

- Backward
- Quantifier-free
- Intuitive

We will focus on wp.

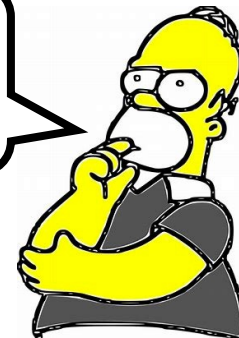


Weakest Precondition

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 $P \rightarrow wlp(C, Q)$

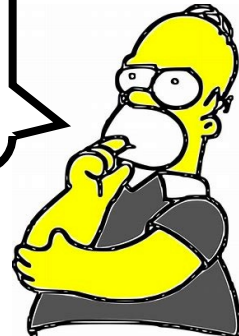
*wlp must be computable. We
will introduce the details.*



Weakest Precondition

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

This is clear because
skip does nothing.



Weakest Precondition

This is similar to AS in Hoare logic.

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

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

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

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

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

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

$$\begin{aligned} & wlp(a := a + 1; b := a, a = 3 \wedge b > 0) \\ &= wlp(a := a + 1, wlp(b := a, a = 3 \wedge b > 0)) \\ &= wlp(a := a + 1, a = 3 \wedge a > 0) \\ &= a + 1 = 3 \wedge a + 1 > 0 \end{aligned}$$

Weakest Precondition

$$\begin{aligned} & wlp(if(b1) \text{ then } \{C1\} \text{ else } \{C2\}, Q) \\ & = (istrue(b1) \rightarrow wlp(C1, Q)) \wedge (isfalse(b1) \rightarrow wlp(C2, Q)) \end{aligned}$$

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

Weakest Precondition

$$\begin{aligned} & wlp(if(b1) \text{ then } \{C1\} \text{ else } \{C2\}, Q) \\ &= (istrue(b1) \rightarrow wlp(C1, Q)) \wedge (isfalse(b1) \rightarrow wlp(C2, Q)) \end{aligned}$$

$$\begin{aligned} & wlp(if(a == 0) \text{ then } \{b := 1\} \text{ else } \{b := 0\}, b = 1 \vee b = 0) \\ &= (a = 0 \rightarrow wlp(b := 1, b = 1 \vee b = 0)) \wedge (a \neq 0 \rightarrow wlp(b := 0, b = 1 \vee b = 0)) \\ &= (a = 0 \rightarrow 1 = 1 \vee 1 = 0) \wedge (a \neq 0 \rightarrow 0 = 1 \vee 0 = 0) \end{aligned}$$

Weakest Precondition

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

:-) 呵呵

Weakest Precondition

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

$\{P\}while(b1)C\{Q\}$

$$\left\{ \begin{array}{l} P \rightarrow I \\ (I \wedge isfalse(b1)) \rightarrow Q \\ \{I \wedge istrue(b1)\}C\{I\} \end{array} \right.$$

Invariant need be
written manually

Dafny

post-condition

```
method m(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

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

Dafny 2.3.0.10506

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

Dafny

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

		Description	Line
	1	possible division by zero	4

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[i] == 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

Thanks & Questions