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

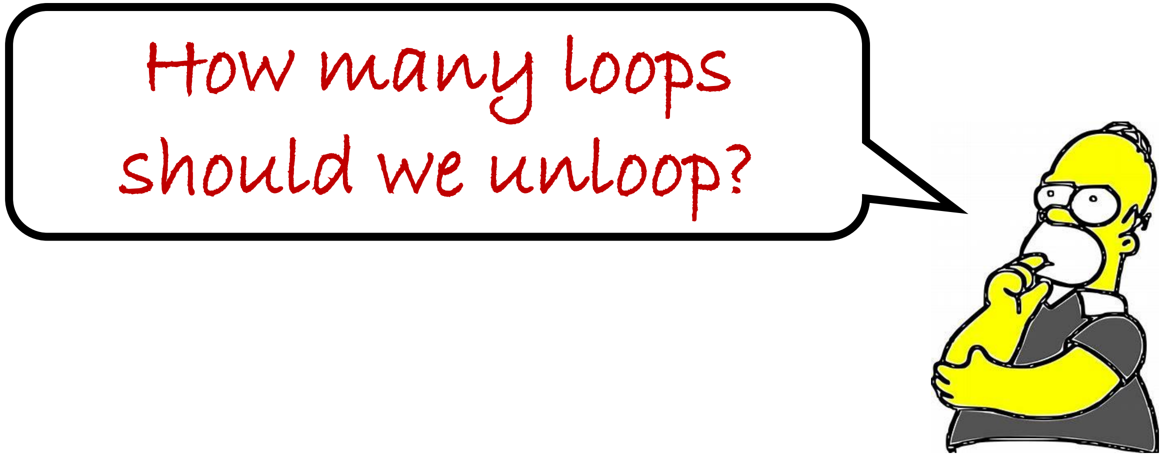
unroll

```
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);
}
```



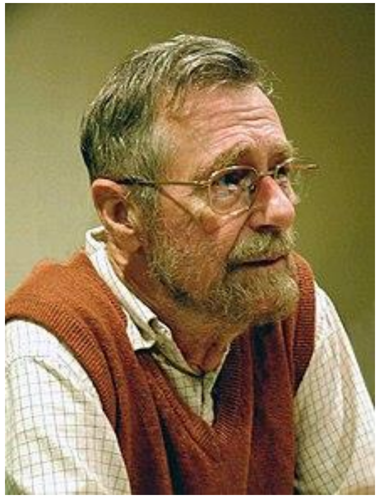
Loops

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

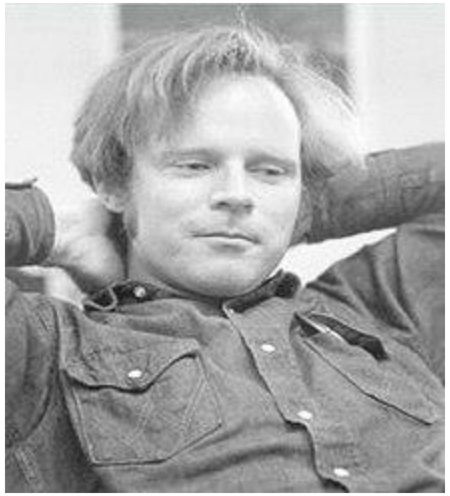
It seems that current technique doesn't work. We need more powerful tools.

Hoare Logic Can Do It!

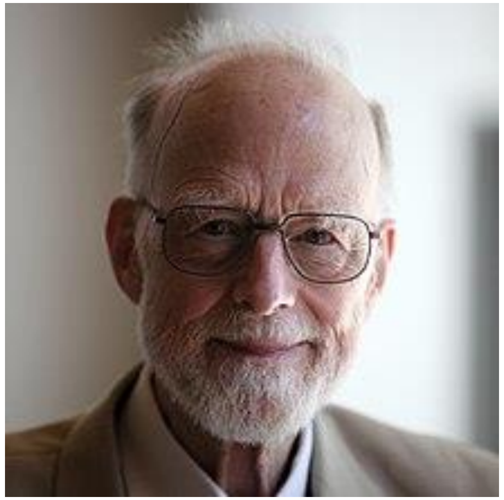
Assign logical meaning to programs



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.

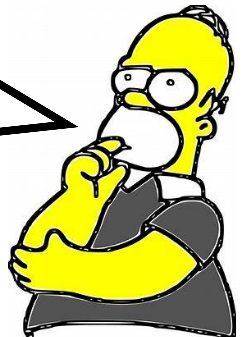
Hoare logic is an axiom system for program verification.

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

$A, B, C, \ldots$

complete connectives

$\neg, \vee$

$(,)$

It is before a formula, denoting the formula is a tautology.

$\vdash$

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).

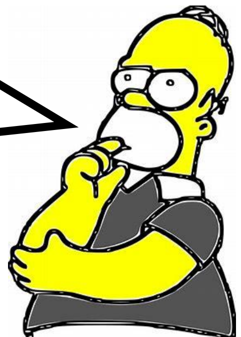
Axiom System(公理系统)

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

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

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

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.

$$
\begin{aligned}
&Axiom1 \quad \vdash ((P \vee P) \to P) \\
&Axiom2 \quad \vdash (P \to (P \vee Q)) \\
&Axiom3 \quad \vdash ((P \vee Q) \to (Q \vee P)) \\
&Axiom4 \quad \vdash ((Q \to R) \to ((P \vee Q) \to (P \vee R)))
\end{aligned}
$$

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) $\vdash P \vee \neg P \xrightarrow{\frac{P}{(R \vee S)}} \vdash (R \vee S) \vee \neg(R \vee S)$

分离规则 if $\vdash A, \vdash A \rightarrow B$, 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.

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

$Proof.$

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

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.

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

$Proof.$

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

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

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

Soundness and Completeness

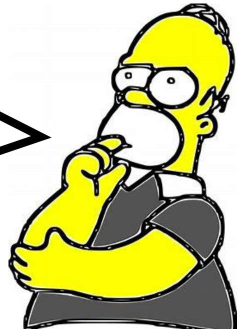
DEFINITION

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

DEFINITION

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

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



Outline – This Lecture

- Axiom System
- Hoare Logic

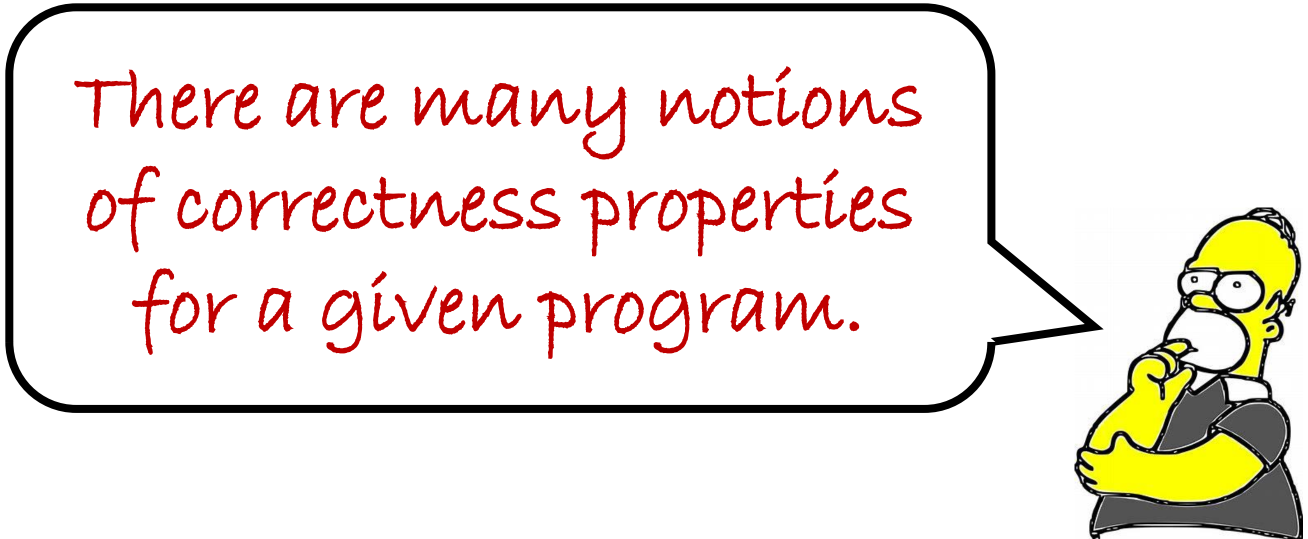
Hoare logic is designed specifically for program verification. It answers what is program correctness and how to formally prove the correctness.

Problem1:

How to define program correctness?

Hoare Logic

- Is the program guaranteed to reach a certain program point (e.g. terminate)?
- When the program reaches this point, are certain values guaranteed?
- Could the program encounter runtime errors / raise certain exceptions?
- Will the program leak memory / secret data, etc.?



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 $x = 1$ $\boxed{x = x + 1;}$ $\Rightarrow$ $x = 2$ 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.

$$x = 3 \land y = 5$$

$$x > 3 \land x \neq 0$$

Pre-/post-condition are assertions, i.e., conditions on the program variables.

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

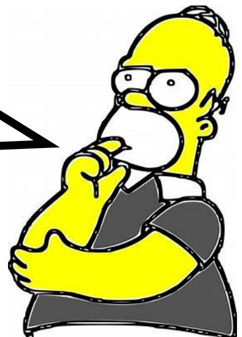
Hoare Triples

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.

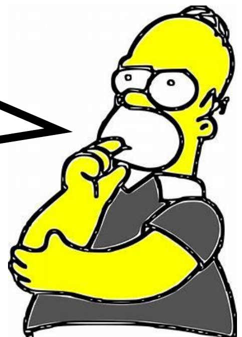
Alan Turing proved in 1936 that a general algorithm to solve the halting problem cannot exist.

Hoare Triples

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

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

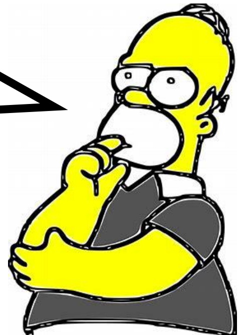
P and Q are predicate logic propositions. What about C?



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: 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

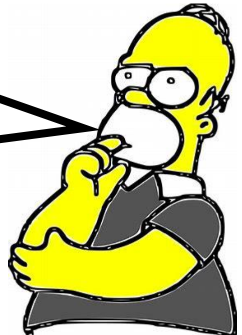
For simplicity, we just consider a simple imperative language.



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)
  - 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…

For simplicity, we just consider a simple imperative language.



Hoare Logic

- Initial symbols
  - symbols in predicate logic : $\forall$, $\exists$, $\rightarrow$, $\neg$, ...
  - symbols in the simple imperative language: if, while, :=, …
  - vdash : $\vdash$ (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

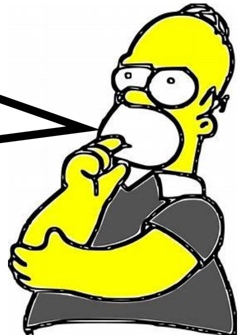
$$\{a = 0\}a := 3\{a = 3\}$$

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

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

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

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



Hoare Logic

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

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

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

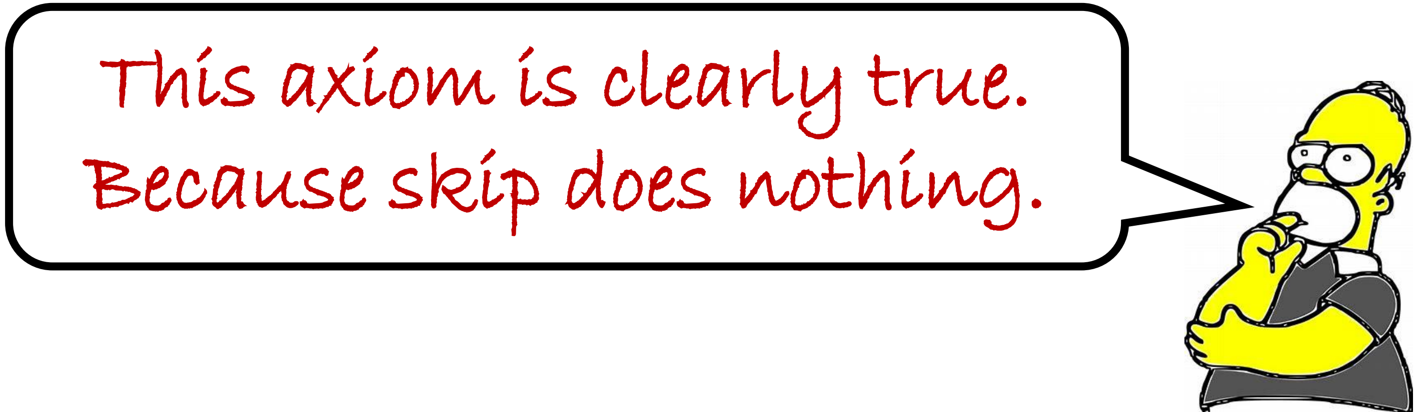
$$\{a = 1 \wedge b = 5\}if(a == 1)a := b\{a = 5 \wedge b = 5\}$$

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 and Inference Rules

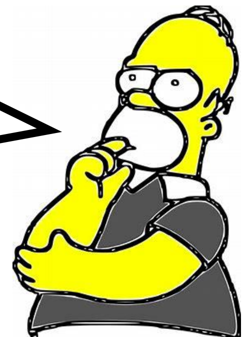
$$\{P\} skip \{P\}$$



Axioms and Inference Rules

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

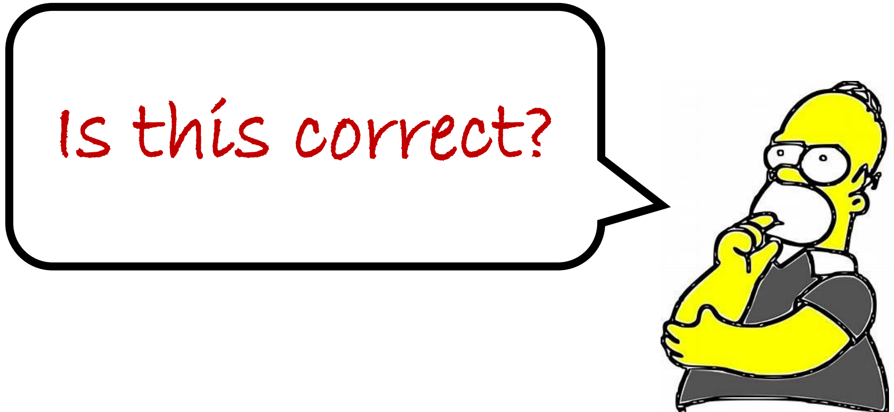
The assignment axiom brings troubles.



Axioms and Inference Rules

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

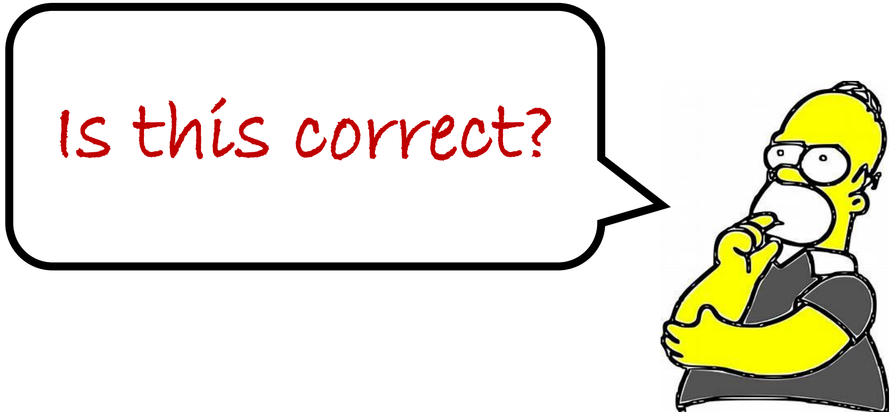
$$\{a = 0\}a := 1\{a = 0\}$$



Axioms and Inference Rules

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

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



Axioms and Inference Rules

v is a fresh variable.

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

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

$$\Downarrow$$

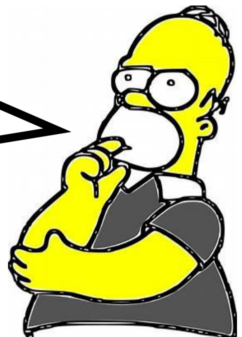
$$\{a = 1\}a := a + 1\{(\exists v)(a = \boxed{v + 1} \wedge \boxed{v = 1})\}$$

simplification

$$\Downarrow$$

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

This is correct and a forward assignment axiom.



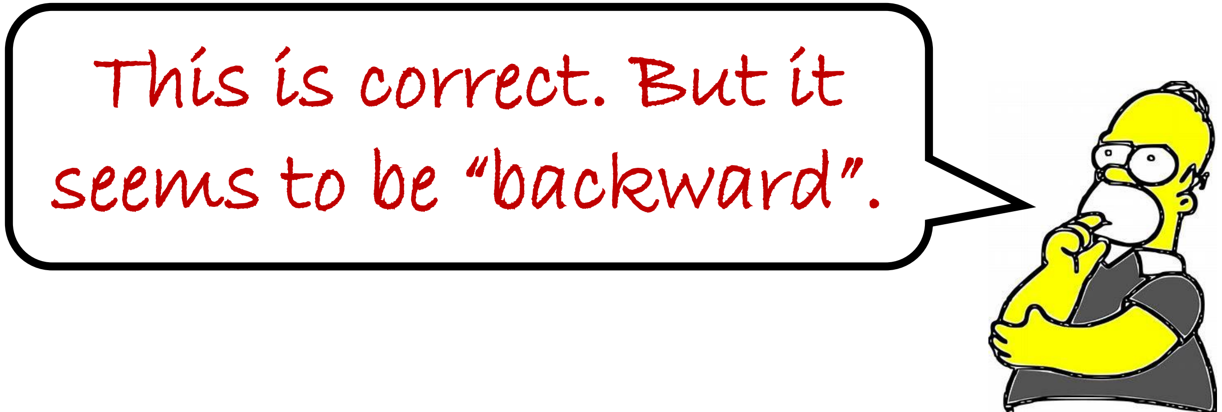
Axioms and Inference Rules

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

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

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

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



Axioms and Inference Rules

The abbreviation is AS-FW.

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

- Forward
- Quantifier
- Intuitive

The abbreviation is AS.

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

- Backward
- Quantifier-free
- Not intuitive

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

Axioms and Inference Rules

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

$$\Downarrow$$

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

$$\Downarrow$$

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

The simplification is correct.

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

Axioms and Inference Rules

strong $P \rightarrow Q$ weak

The line means inference.

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

weakening consequent(WC)

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

Axioms and Inference Rules

strong $P \rightarrow Q$ weak

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

We can also strengthen precedent(SP).

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

$Proof.$

Axioms and Inference Rules

strong $P \to Q$ weak

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

We can also strengthen precedent(SP).

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

$Proof.$

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

Axioms and Inference Rules

strong $P \to Q$ weak

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

We can also strengthen precedent(SP).

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

$Proof.$

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

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

Axioms and 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

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

Axioms and 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.

Axioms and Inference Rules

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

The sequential composition rule(SC)

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

$Proof.$

Axioms and Inference Rules

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

The sequential composition rule(SC)

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

$Proof.$

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

Axioms and Inference Rules

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

The sequential composition rule(SC)

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

$Proof.$

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

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

Axioms and 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.

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

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

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

Axioms and Inference Rules

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

The sequential composition rule(SC)

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

$Proof.$

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

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

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

$(4)\{b > 3\}a := 2 * b\{a - b \geq 4\}$ $\quad SP(2)(3)$

Axioms and 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.

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

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

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

$(4)\{b > 3\}a := 2 * b\{a - b \geq 4\}$ $\quad SP(2)(3)$

$(5)\{b > 3\}a := 2 * b; a := a - b\{a \geq 4\}$ $\quad SC(1)(4)$

It's clearly proved backwards.

Axioms and 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)

Axioms and Inference Rules

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

$Proof.$

Axioms and Inference Rules

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

$Proof.$

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

Axioms and Inference Rules

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

$Proof.$

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

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

Axioms and Inference Rules

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

$Proof.$

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

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

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

Axioms and Inference Rules

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

$Proof.$

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

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

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

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

Axioms and Inference Rules

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

$Proof.$

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

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

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

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

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

Axioms and Inference Rules

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

$Proof.$

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

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

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

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

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

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

Axioms and Inference Rules

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

$Proof.$

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

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

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

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

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

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

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

Axioms and 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

Axioms and 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.

Axioms and Inference Rules

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

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

$Proof.$

Axioms and Inference Rules

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

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

$Proof.$

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

loop invariant

Axioms and Inference Rules

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

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

$Proof.$

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

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

Axioms and Inference Rules

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

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

$Proof.$

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

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

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

loop invariant always holds.

Axioms and Inference Rules

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

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

$Proof.$

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

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

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

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

Axioms and 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

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

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

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

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

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

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

Prove Partial Correctness

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

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

Proof:

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

Loop invariant is true

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

```
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

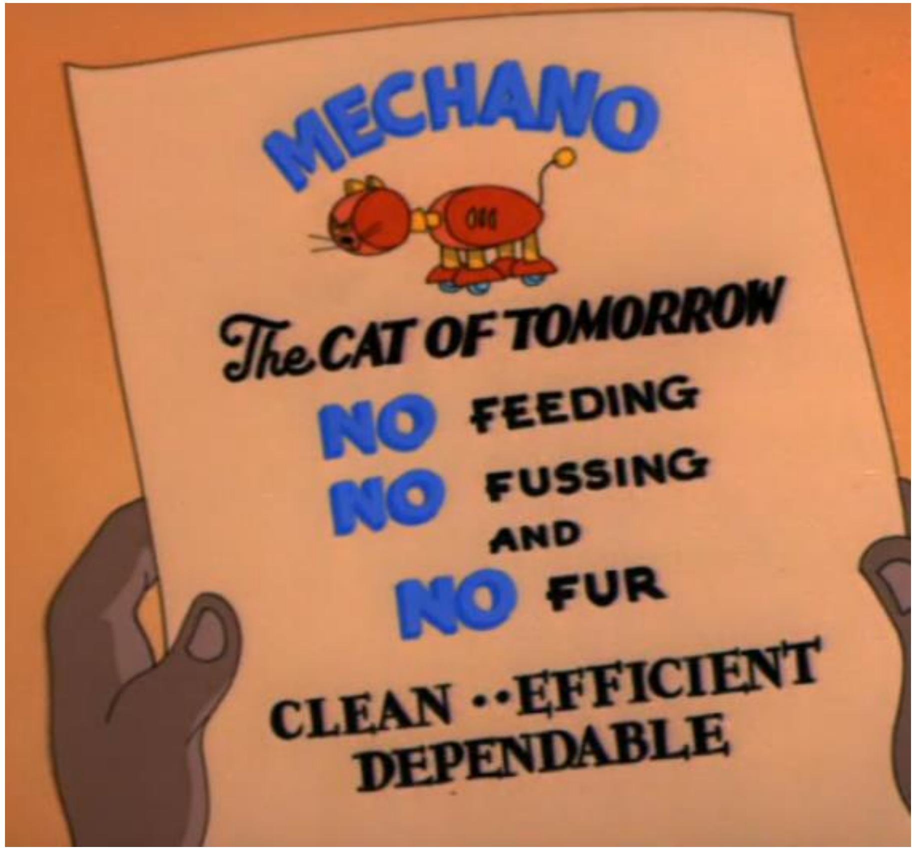
```
}
```

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

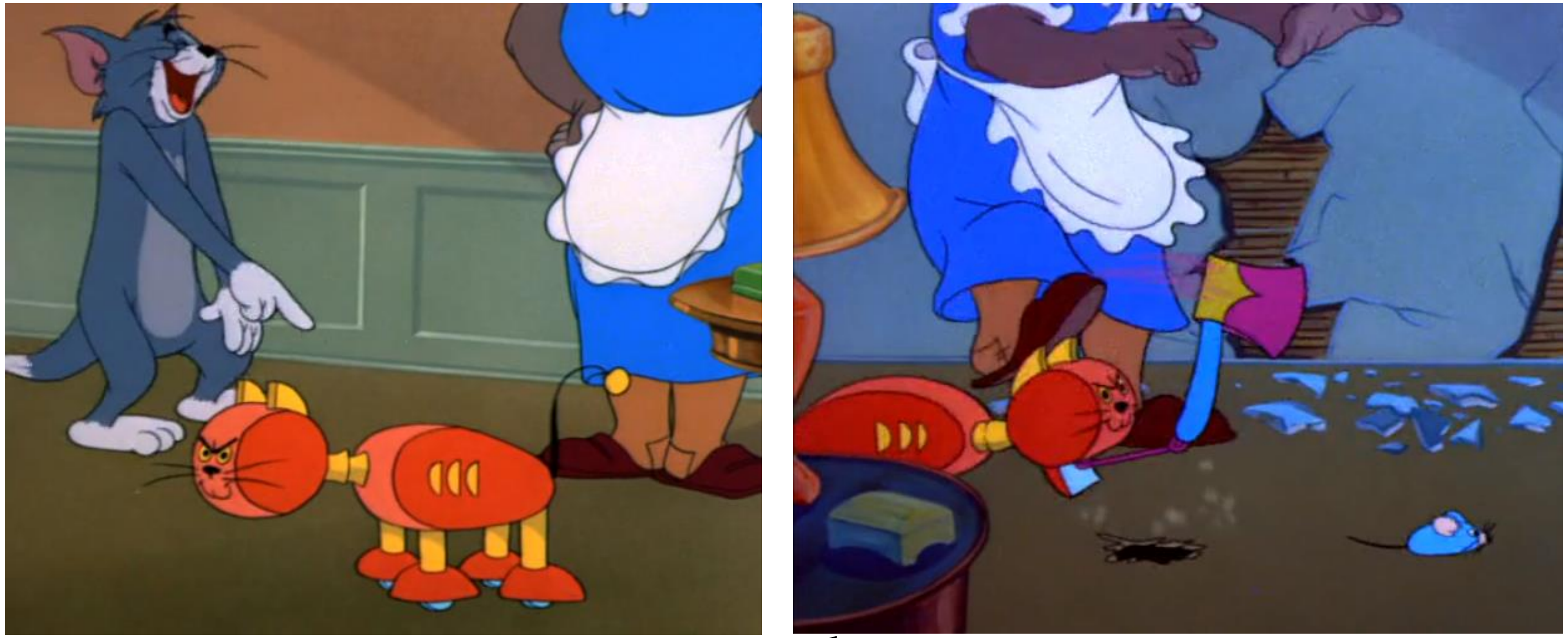
loop invariant

However,

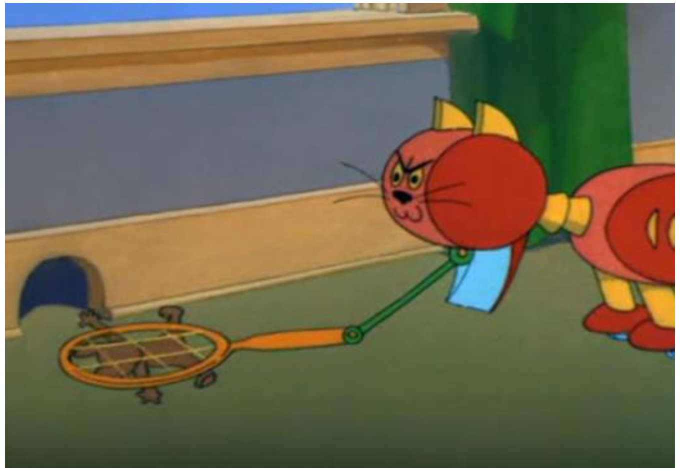
Manual Proof is usually tedious

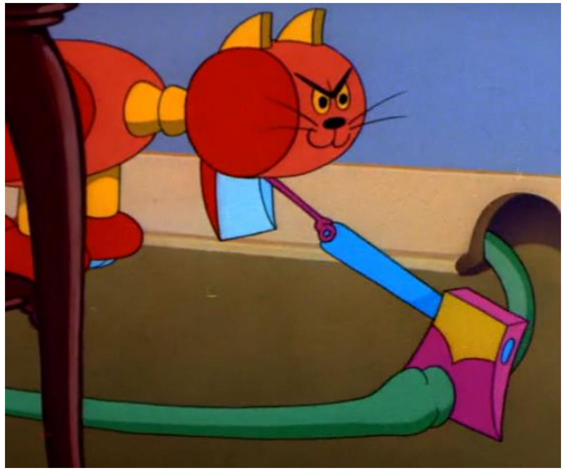


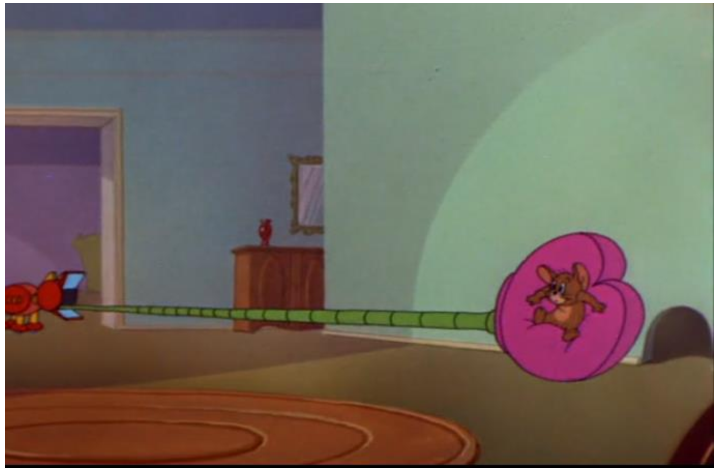
We want an automated program verifier.



But previous methods (symbolic execution) cannot handle loops







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

Automatic Proof, Again???

Predicate Transformer

stronger $P \rightarrow Q$ 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.

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

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

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

weakest precondition

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

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

Predicate Transformer

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

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

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

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

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

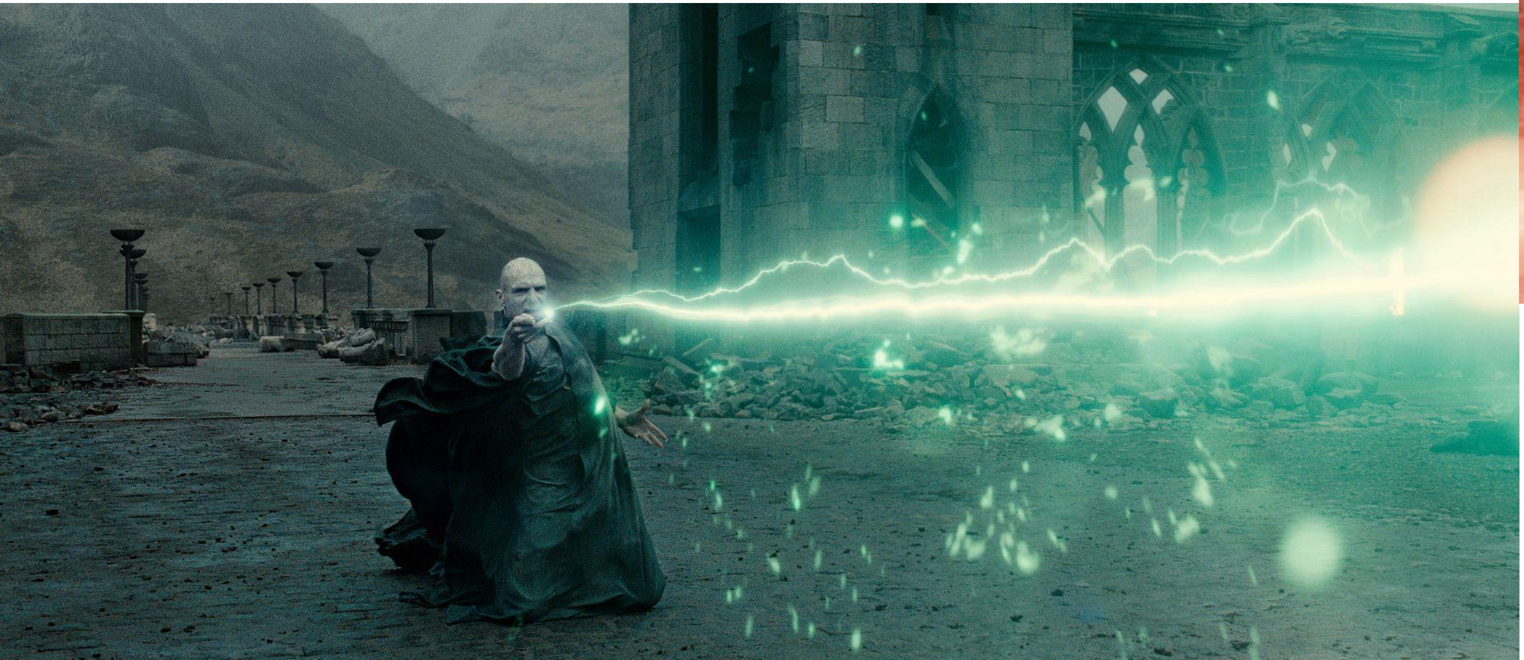
To prove `{P}C{Q}`

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

$$\begin{tikzcd} & P \land \neg wp(C,Q) \\ \{P\}C\{Q\} \arrow[ur] \arrow[dr] & \\ & sp(C,P) \land \neg Q \end{tikzcd}$$

How to prove the formula?

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 postcondition.

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

- Backward
- Quantifier-free
- Not intuitive

It is more widely used because it's simpler.

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 postcondition.

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

- Backward
- Quantifier-free

We will focus on wp.

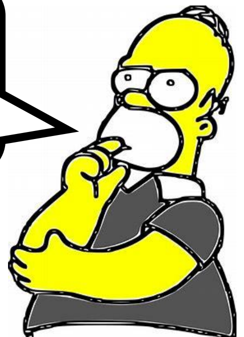
tive

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 $P \rightarrow wlp(C, Q)$

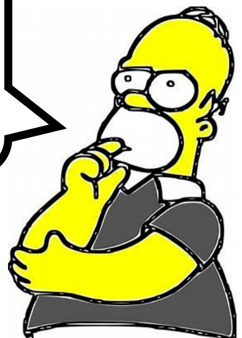
Wlp must be computable. We will introduce the details.



Weakest Precondition

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

This is clear because skip does nothing.



Weakest Precondition

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

This is similar to AS in Hoare logic.

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

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

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

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

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

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

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

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

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

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

$$= (a = 0 \rightarrow wlp(b := 1, b = 1 \vee b = 0)) \wedge (a \neq 0 \rightarrow wlp(b := 0, b = 1 \vee b = 0))$$

$$= (a = 0 \rightarrow 1 = 1 \vee 1 = 0) \wedge (a \neq 0 \rightarrow 0 = 1 \vee 0 = 0)$$

Weakest Precondition

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

:-) 呵呵

Weakest Precondition

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

Invariant need be written manually

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

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

```
Dafny 2.3.0.10506

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

https://rise4fun.com/Dafny/tutorial

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