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 logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Interactive Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

A Quick Recap – Predicate Logic

- DPLL(T)
- EUF
- The Nelson-Oppen Method
- Quantifier SMT

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;
}
```

$$(a > 0) \wedge$$

$$(b > 0) \wedge$$

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

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

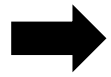
$$\rightarrow (result > 0)$$

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.



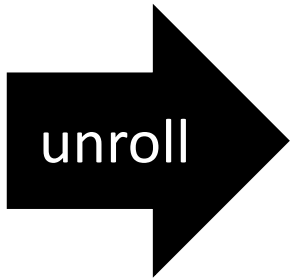
$$(a > 0) \wedge$$
$$(b > 0) \wedge$$
$$((c \geq 0) \rightarrow (result = a)) \wedge$$
$$(\neg(c \geq 0) \rightarrow (result = b)) \wedge$$
$$\neg(result > 0)$$

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

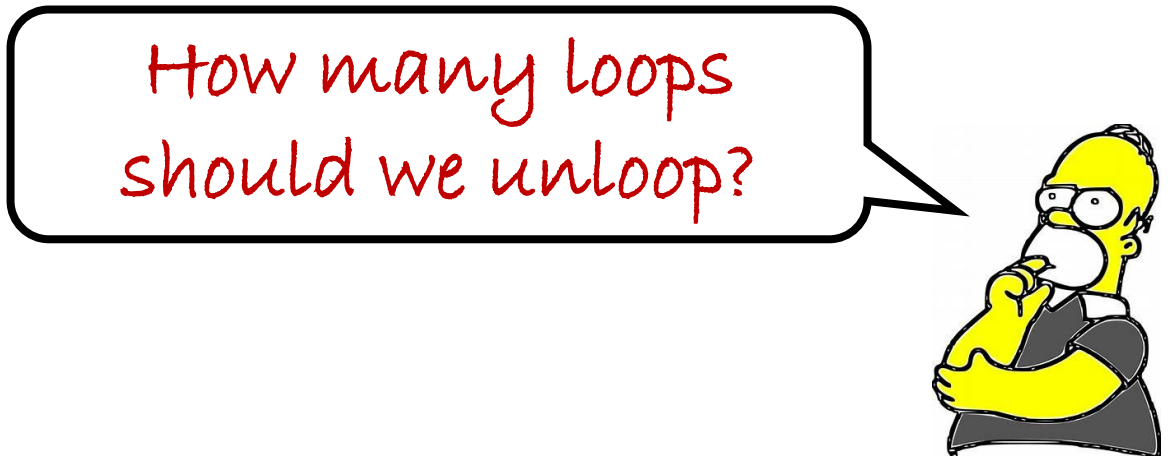



```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
}
```

Straight-Line Code

Loops

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

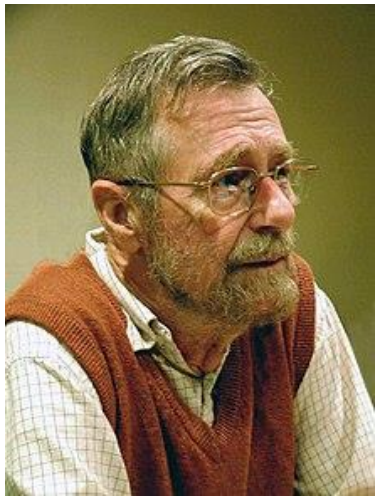


Loops

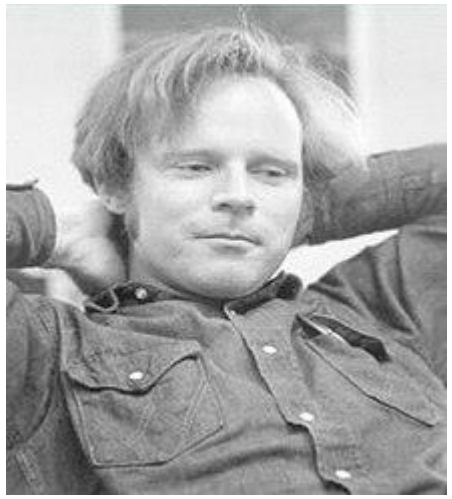
```
void f(unsigned a)
{
    unsigned i = 0;
    while(i < a)
    {
        i = i + 1;
    }
    assert(i == a);
}
```

Hoare Logic Can Do It!

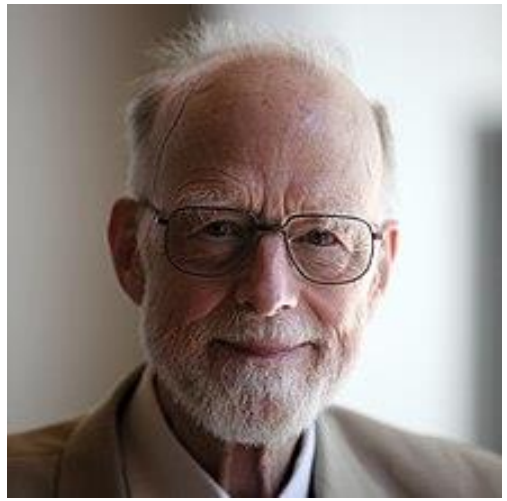
Assign logical meaning to programs



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.

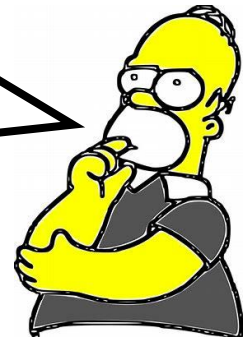
Hoare logic is an axiom system for program verification.

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.

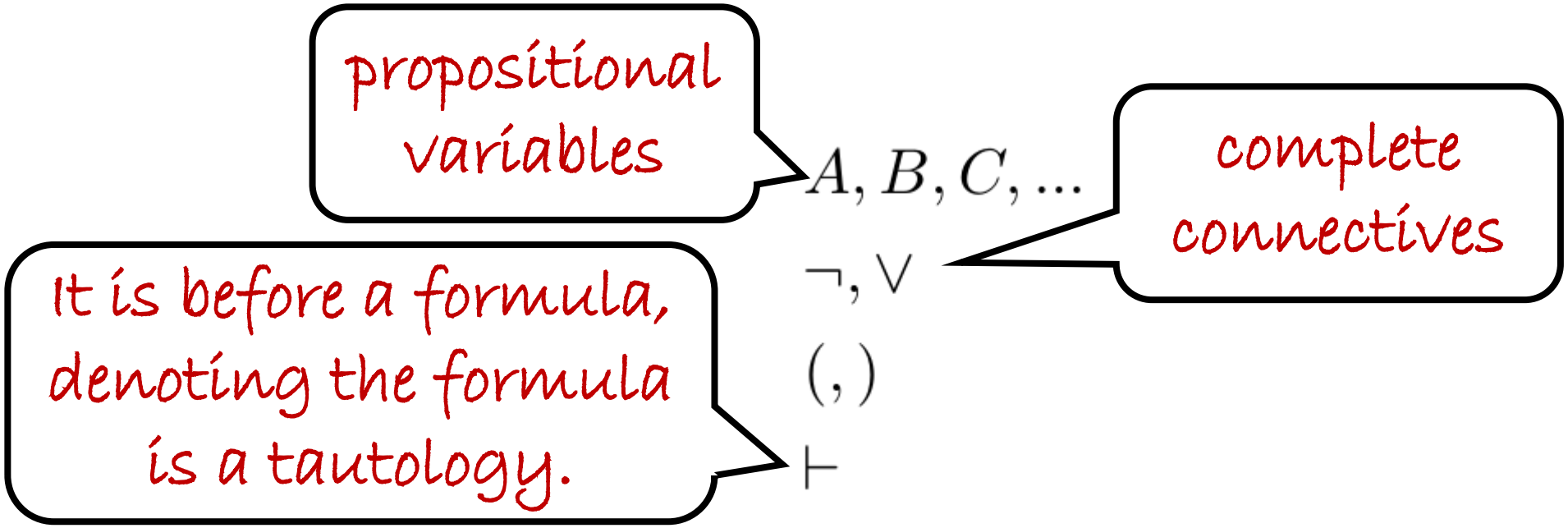
We can firstly use propositional logic to explain the components of an axiom system.



Axiom System(公理系统)

DEFINITION

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



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.

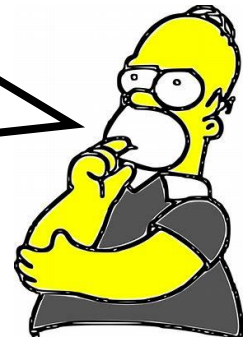
2). If *A* and *B* are WFF, so are (¬A) and (A $\vee$ B).

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.

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

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)

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

分离规则

$$\text{if} \vdash A, \vdash A \to B, \text{ then} \vdash B$$

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) $\vdash (Q \to R) \to (P \vee Q \to P \vee R)$ *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) $\vdash (Q \to R) \to (P \vee Q \to P \vee R)$ $Axiom4$

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

Axiom System(公理系统)

DEFINITION

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

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

Proof.

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

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

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

Soundness and Completeness

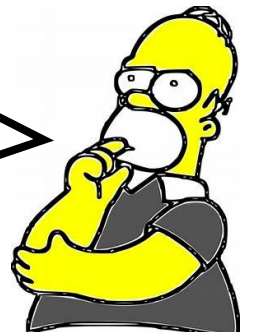
DEFINITION

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

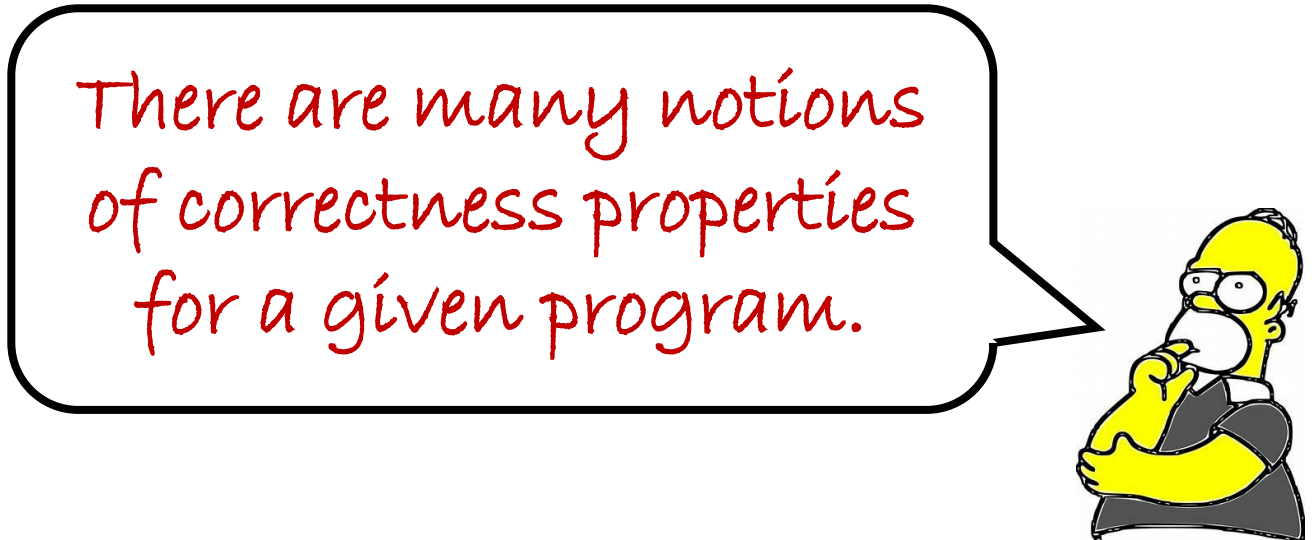
- Axiom System
- Hoare Logic

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

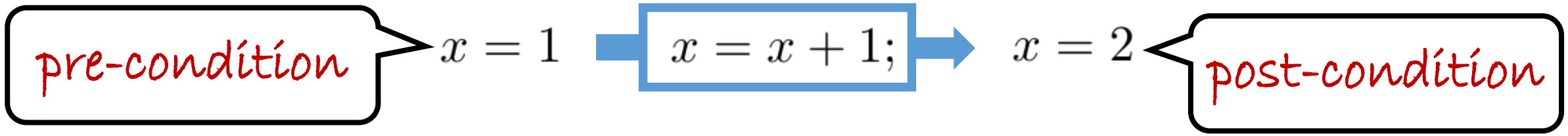
Program state includes the value of every variable used in the program.

DEFINITION

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

DEFINITION

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



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 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.

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

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

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.

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

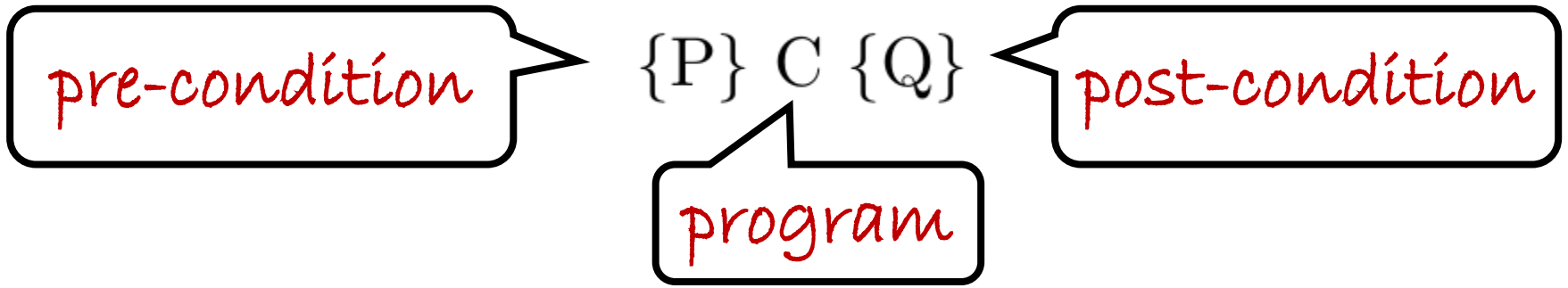
We use Hoare triples to define program correctness.

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.



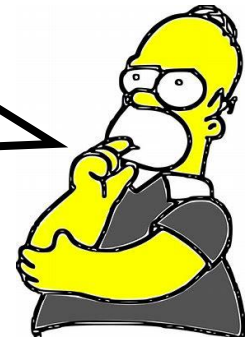
Hoare Triples

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.

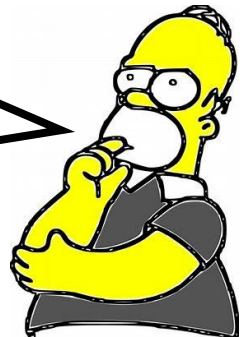
Alan Turing proved in 1936 that a general algorithm to solve the halting problem cannot exist.

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 : $\forall, \exists, \rightarrow, \neg$, ...
  - 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

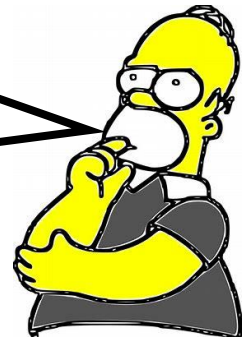
$$\{a = 0\}a := 3\{a = 3\}$$

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

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

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

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.

Thanks & Questions