Hoare Logic

Zhaoguo Wang

Adapted From:

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

Lab1

- **Lab1 DDL is 10.31. Hurry up!**
- **If you need to use grace day, please inform TA in advance.**

Hoare Logic（霍尔逻辑）

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

Outline – This Lecture

- Axiom System
- Hoare Logic

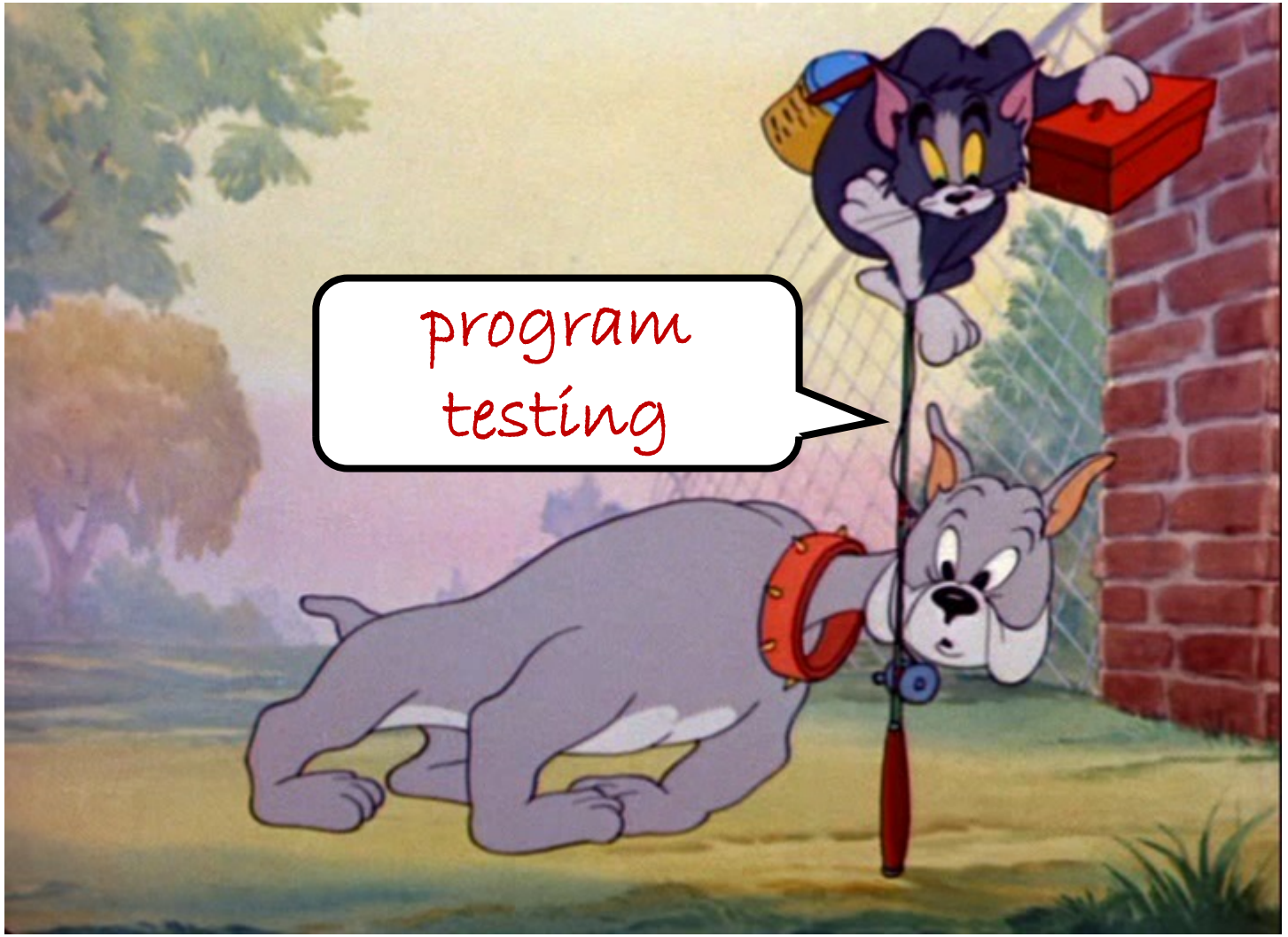
Outline – This Lecture

- Axiom System
- Hoare Logic

Program Analysis



Program Analysis



Program Analysis



Program Analysis

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



Program Analysis

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

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



Program Analysis

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



$$(a > 0) \wedge$$

$$(b > 0) \wedge$$

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

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

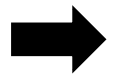
$$\rightarrow (result > 0)$$

Prove it is universally valid.

Program Analysis

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

Prove it is universally valid.



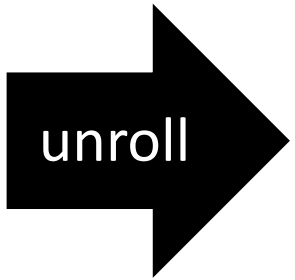
$$(a > 0) \wedge$$
$$(b > 0) \wedge$$
$$((c \geq 0) \rightarrow (result = a)) \wedge$$
$$(\neg(c \geq 0) \rightarrow (result = b)) \wedge$$
$$\neg(result > 0)$$

Prove it is unsatisfiable.

Loops

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

Loops

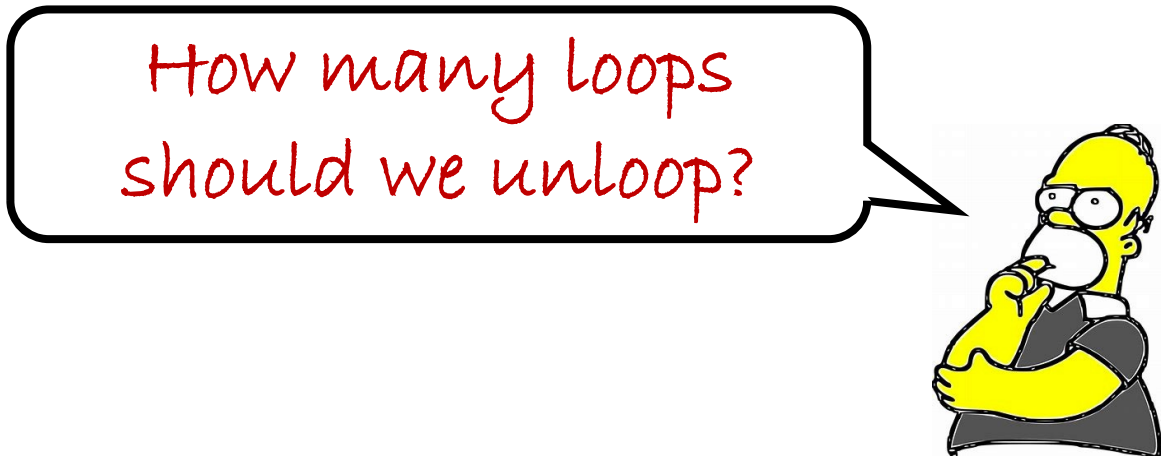



```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
}
```

Straight-Line Code

Loops

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



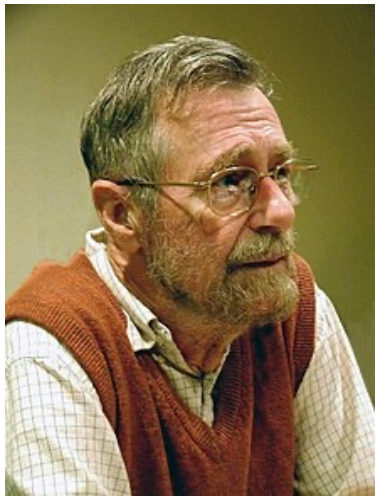
Loops

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

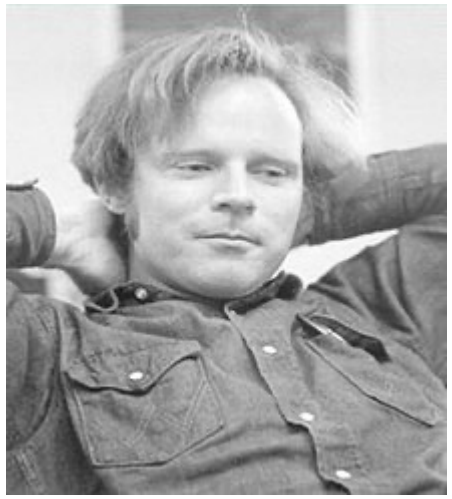


Hoare Logic Can Do It!

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.

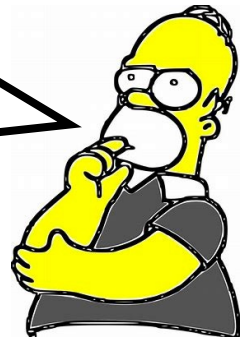
Hoare logic is an axiom system for program verification.

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.

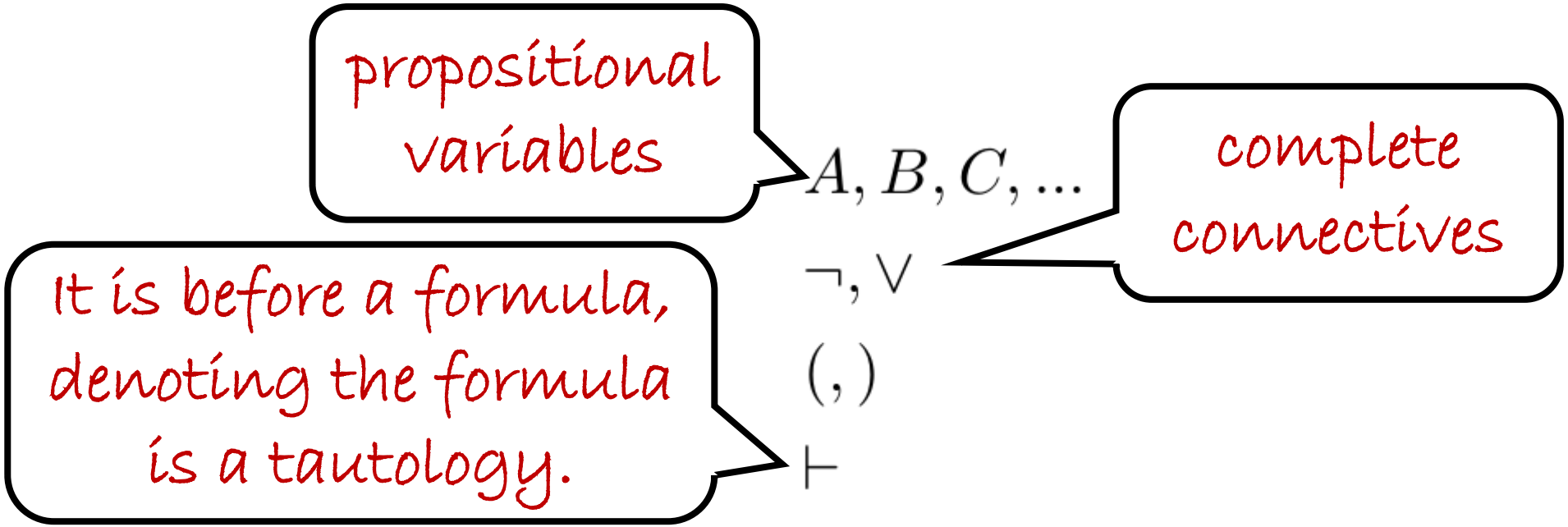
We can firstly use propositional logic to explain the components of an axiom system.



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.

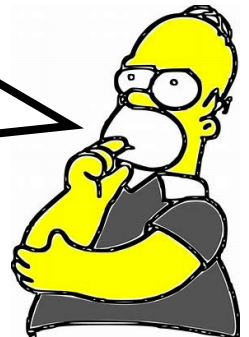
2). If *A* and *B* are WFF, so are (¬A) and (A ∨ B).

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)

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

分离规则

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

置换规则

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

Axiom System(公理系统)

DEFINITION

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

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

Proof.

Axiom System(公理系统)

DEFINITION

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

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

Proof.

(1) $\vdash (Q \to R) \to (P \vee Q \to P \vee R)$ *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.

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

Theorem 1

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

Axiom System(公理系统)

$\vdash P \to P$ Theorem 2

Proof.

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

Axiom System(公理系统)

$\vdash P \to P$ Theorem 2

Proof.

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

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

Proof.

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

Completeness and Soundness

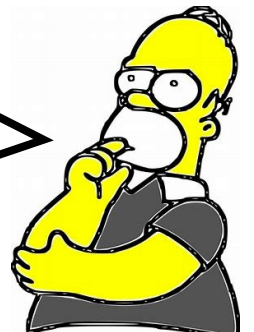
DEFINITION

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:

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

   $\vdash A_i$

   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:
   $\vdash \neg\pi \vee \pi$, $\vdash P \rightarrow (P \vee Q)$ ➡ $\vdash \neg\pi \vee \pi \vee Q$

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

Soundness Proof



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

PROOF

1. All axioms are tautology.

Soundness Proof



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

PROOF

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

Soundness Proof

Induction!



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

PROOF

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

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

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

Other inference rules can also be proved

Prove it by listing all possible situations (truth table)

Soundness Proof



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

PROOF

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

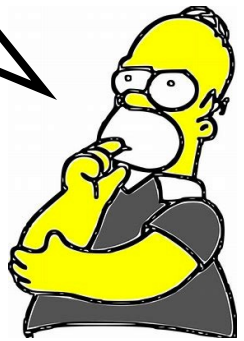
   If $\vdash A$, $\vdash A \to B$, then $\vdash B$

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

Outline – This Lecture

- Axiom System
- Hoare Logic

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



Problem1:
How to define program correctness?

Hoare Triples(霍尔三元组)

DEFINITION

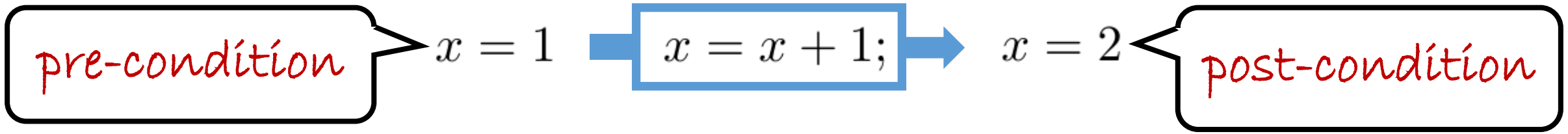
Program state includes the value of every variable used in the program.

DEFINITION

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

DEFINITION

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



Hoare Triples

DEFINITION

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

DEFINITION

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

DEFINITION

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

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

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

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

Hoare Triples

DEFINITION

A program C is partially correct iff:

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

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

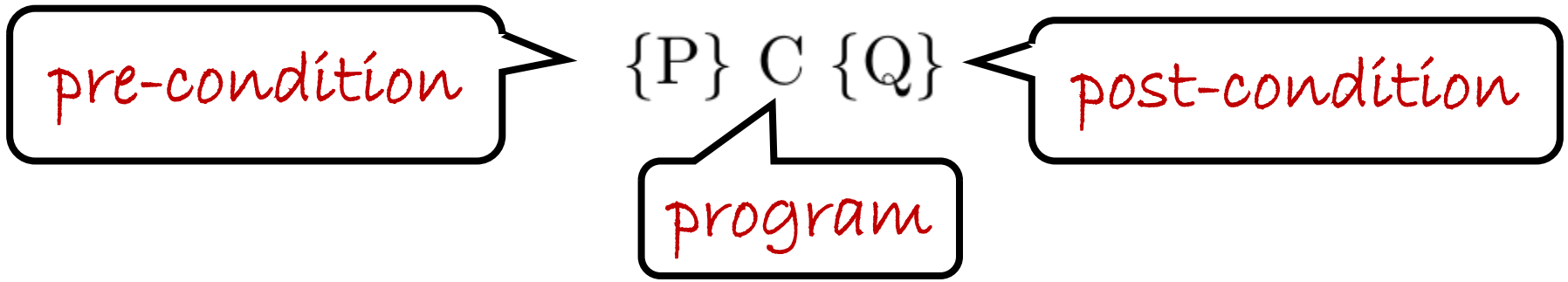
We use Hoare triples to define program correctness.

Hoare Triples

DEFINITION

A program C is partially correct iff:

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



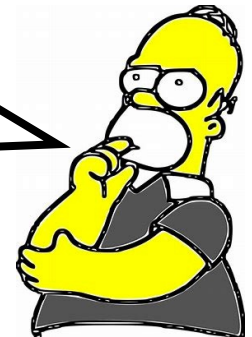
Hoare Triples

DEFINITION

A program C is partially correct iff:

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

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



Halting Problem

DEFINITION

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

Basically halting means terminating.

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

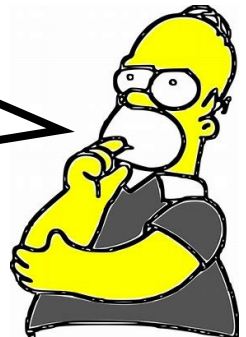


Hoare Triples

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

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

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



A Small Imperative Language

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



A Small Imperative Language

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



Hoare Logic

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

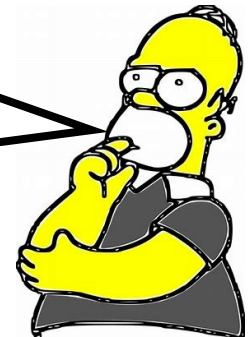
$$\{a = 0\}a := 3\{a = 3\}$$

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

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

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

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



Hoare Logic

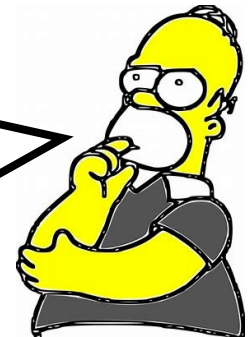
$\{a = 0\}a := 3\{a = 3\}$ ✓

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

$\{a = 1\}a := 2\{a = 10\}$ ✗

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

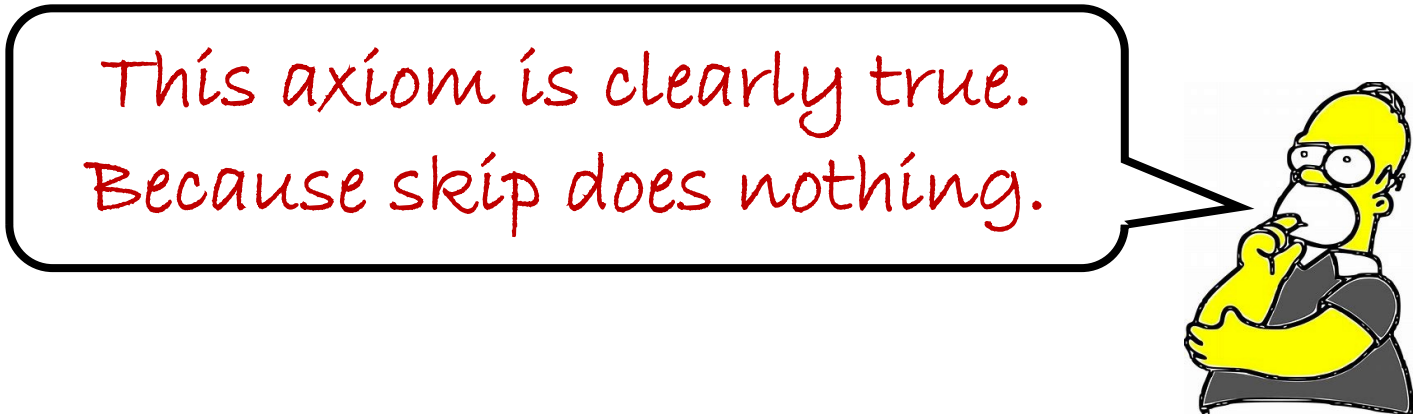
We need to formally prove whether a Hoare triple is right. In other words, we need to prove the program satisfies the specification.



Problem 2:
How to prove program correctness?

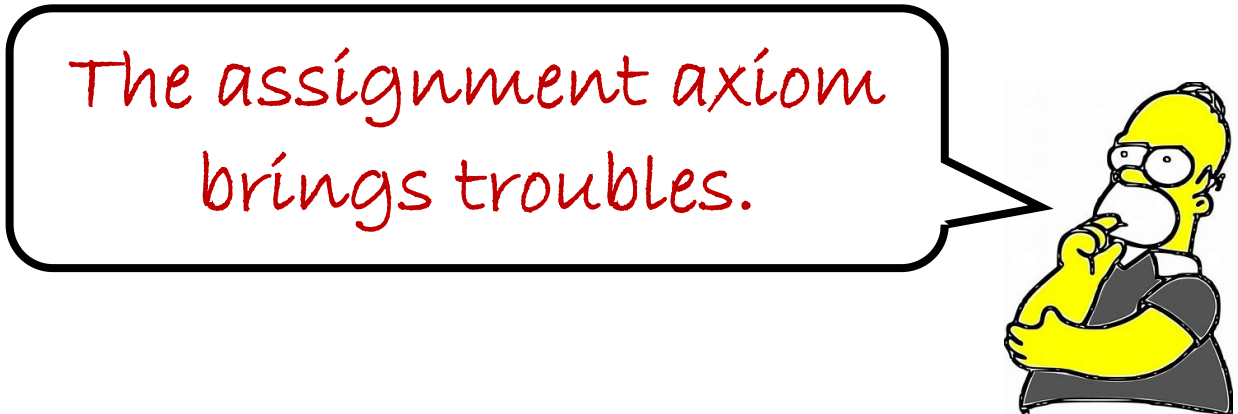
Axioms

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



Axioms

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



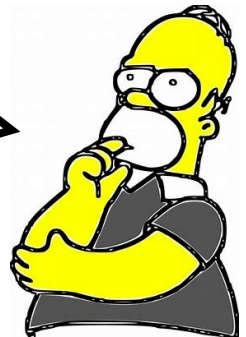
Axioms

Use a to replace e in P

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

$$\{a = 0\}a := 1\{a = 0\}$$ ☒

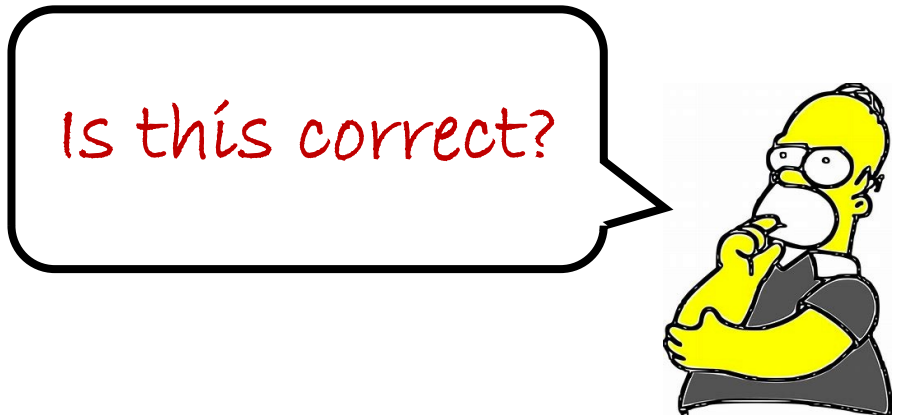
Is this correct?



Axioms

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

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



Axioms

v is a fresh variable.

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

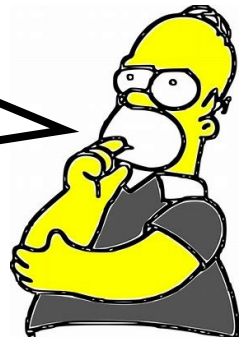
$$\{a = 1\}a := a + 1\{(\exists v)(a = (a + 1)[v/a] \wedge (a = 1)[v/a])\}$$

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

simplification

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

This is correct and a forward assignment axiom.



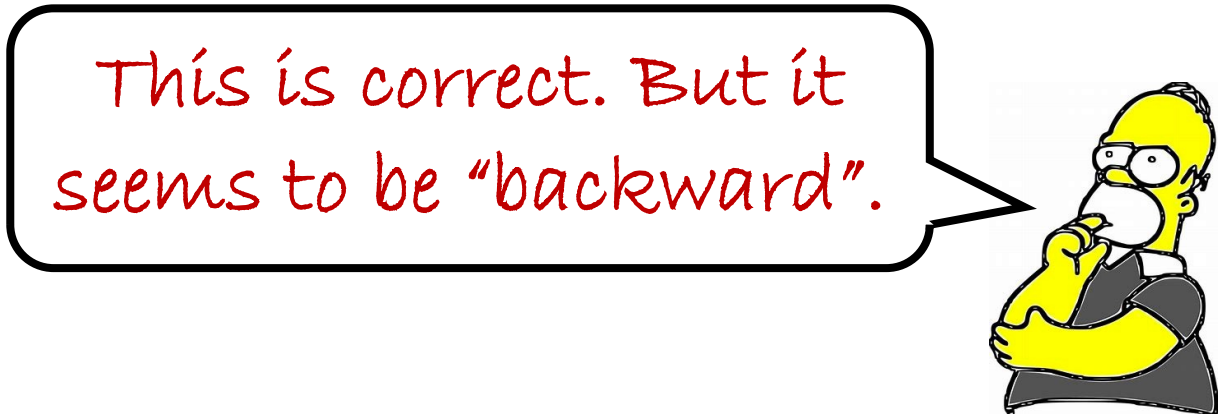
Axioms

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

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

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

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



Axioms

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

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



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



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

The simplification is correct.

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

Inference Rules

strong $P \to Q$ weak

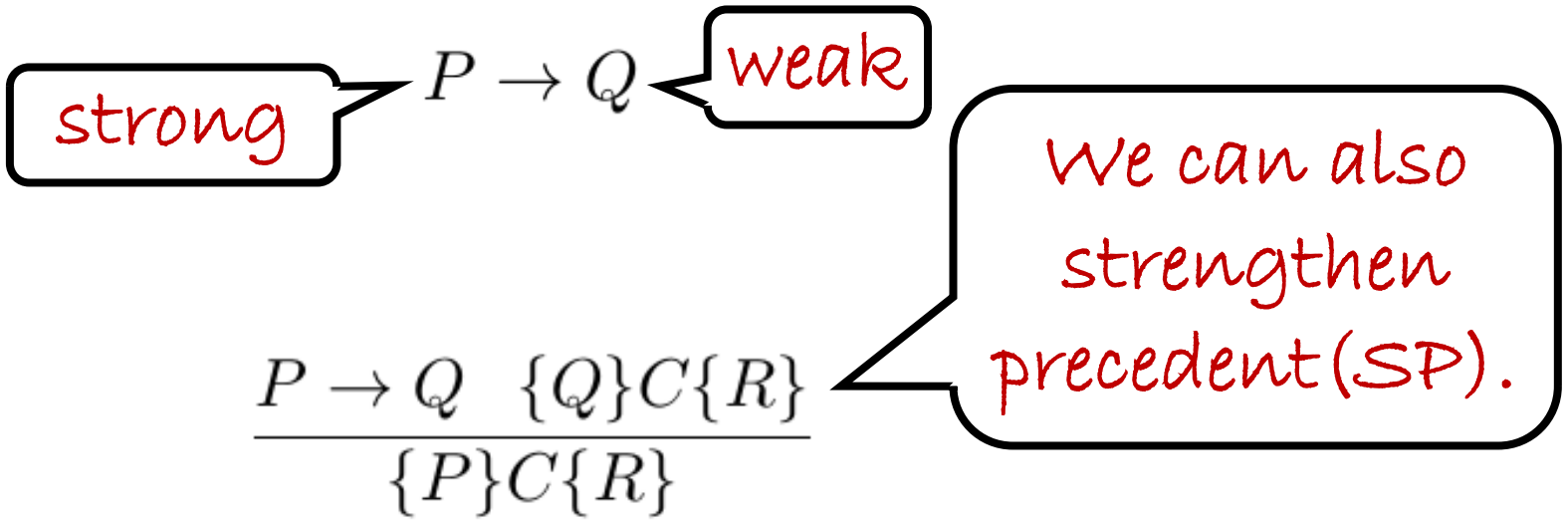
$$\frac{\{P\}C\{Q\} \quad Q \to R}{\{P\}C\{R\}}$$

The line means inference.

weakening consequent (WC)

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

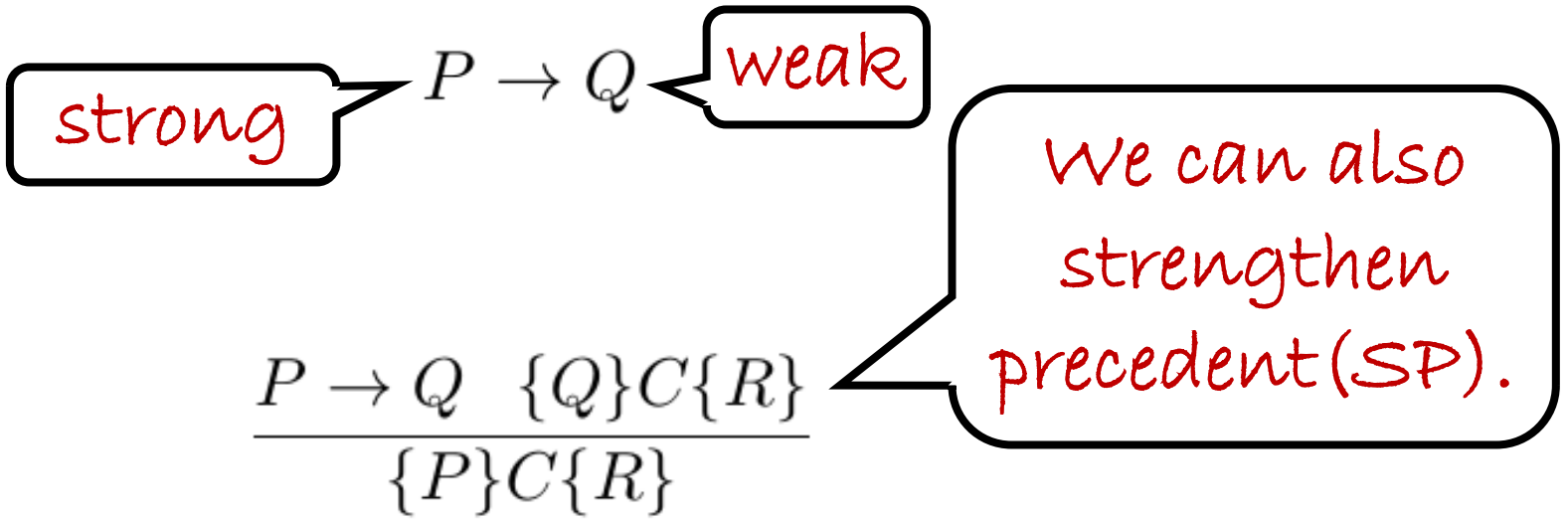
Inference Rules



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

Proof.

Inference Rules

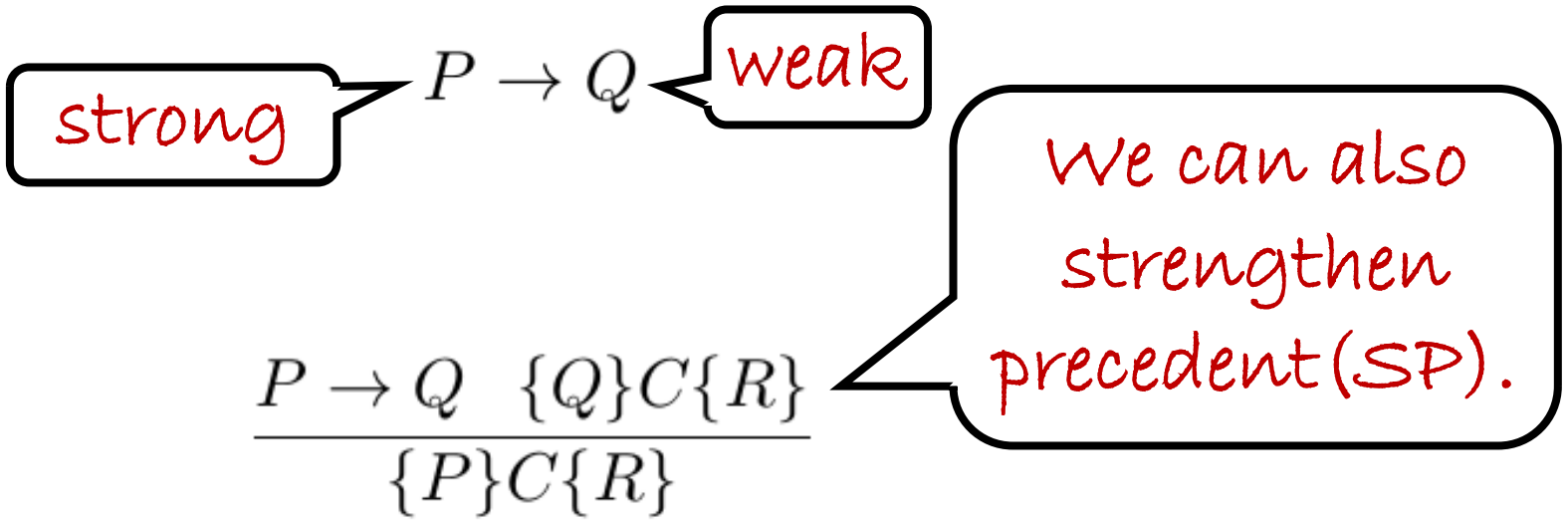


$\{a = n\}a := a + 1\{a = n + 1\}$

Proof.

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

Inference Rules



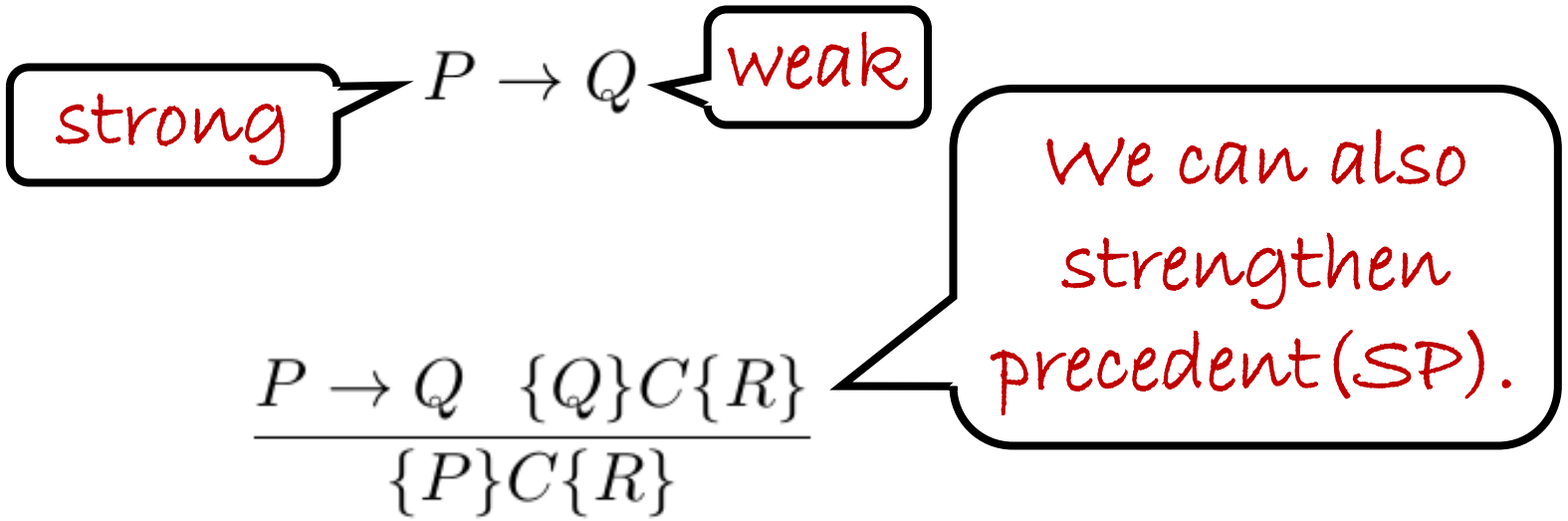
$\{a = n\}a := a + 1\{a = n + 1\}$

Proof.

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

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

Inference Rules



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

Proof.

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

Inference Rules

Program structure

1. Sequence : s1; s2
2. Branch : if(b1) { s1 } then { s2 }
3. Loop structure : while(b1) { s }

We will introduce inference rules for these structures.

This is what we want the most.

Inference Rules

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

The sequential composition rule(SC)

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

Proof.

Inference Rules

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

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

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

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

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

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

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

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

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

$Proof.$

Inference Rules

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

$Proof.$

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

Inference Rules

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

$Proof.$

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

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

Inference Rules

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

$Proof.$

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

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

$Proof.$

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

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

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

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

Inference Rules

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

$Proof.$

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

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

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

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

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

Inference Rules

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

$Proof.$

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

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

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

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

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

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

Inference Rules

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

$Proof.$

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

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

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

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

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

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

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

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

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

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

$Proof.$

Inference Rules

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

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

$Proof.$

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

loop invariant

Inference Rules

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

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

$Proof.$

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

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

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

loop invariant always holds.

Inference Rules

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

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

$Proof.$

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

Inference Rules

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

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

$Proof.$

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

Exercise

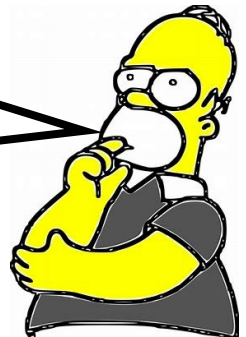
$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

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

pre-condition

post-condition

What is the loop invariant?



Exercise

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

pre-condition

post-condition

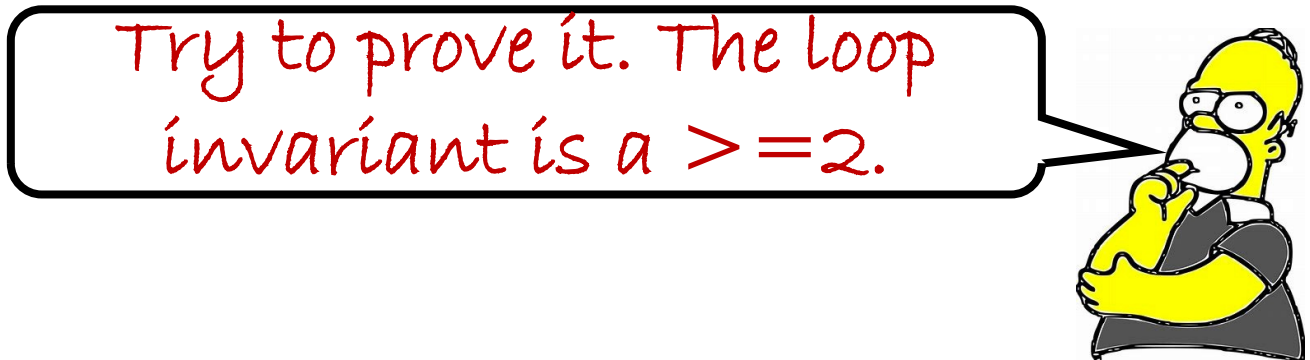
$$\frac{\{I \wedge istrue(b1)\}C\{I\}}{\{I\}\ while(b1)\ C\ \{I \wedge isfalse(b1)\}}$$

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

Example

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

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



Example

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

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

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

Example

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

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

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

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

Example

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

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

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

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

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

Example

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

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

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

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

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

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

Example

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

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

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

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

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Partial Correctness

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

The loop invariant is "true"

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

Proof:

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

Partial Correctness

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

The loop invariant is "true"

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

Proof:

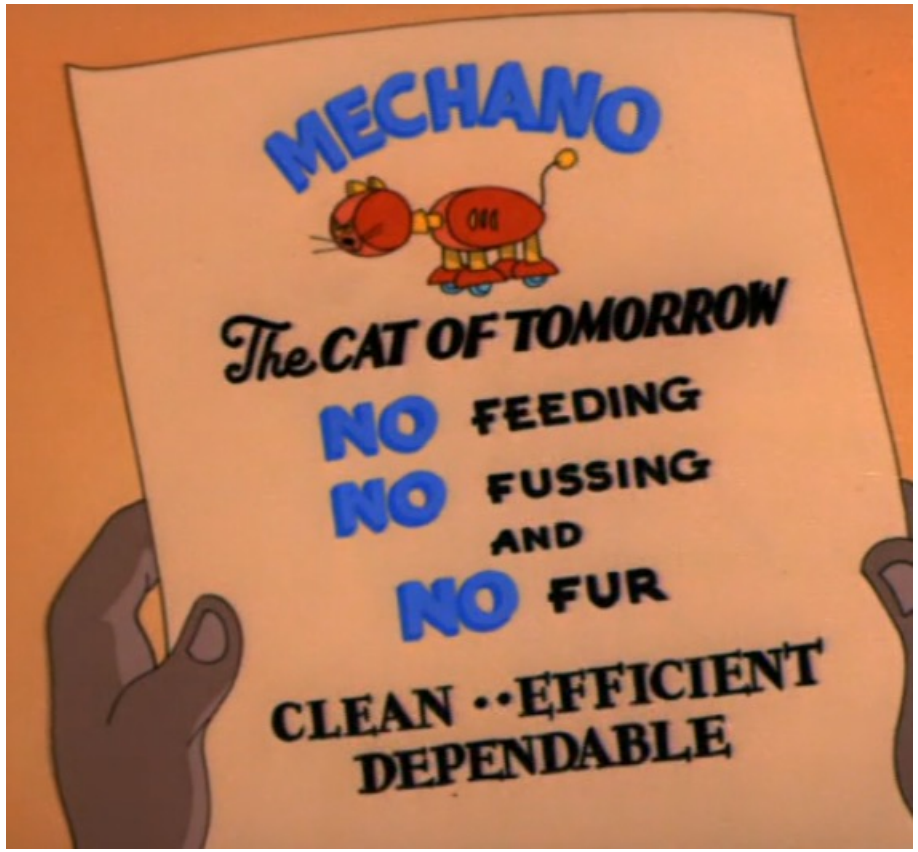
1. **{true** $\wedge\ x \neq 10$**} skip {true** $\wedge\ x \neq 10$**}** SK
2. **true** $\wedge\ x \neq 10 \Rightarrow$ **true**
3. **{true** $\wedge\ x \neq 10$**} skip {true}** WC, 1, 2
4. [illegible] WHP, 3
5. [illegible]
6. **{true} while** $x \neq 10$ **do skip** $\{x = 10\}$

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

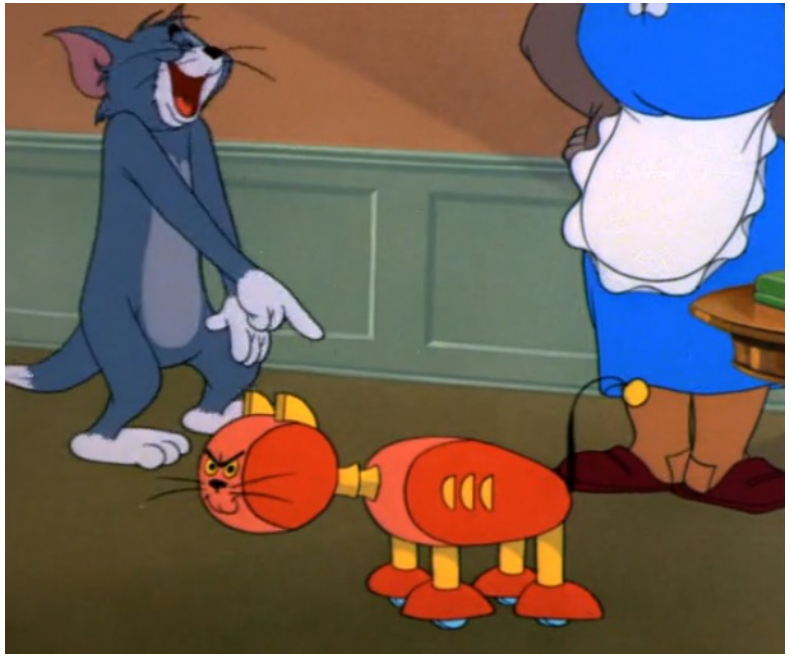


However,



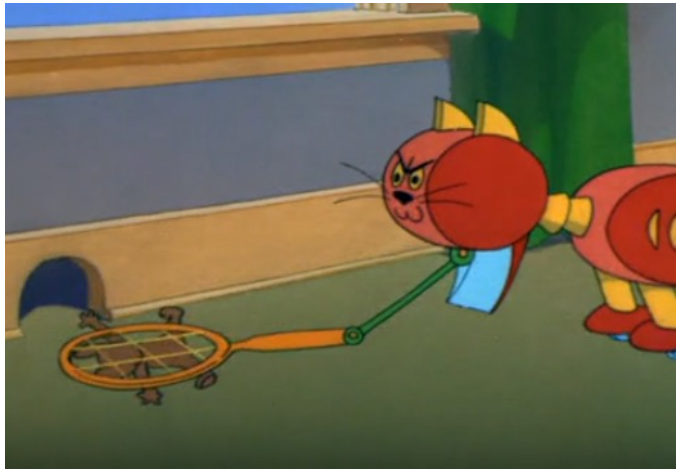
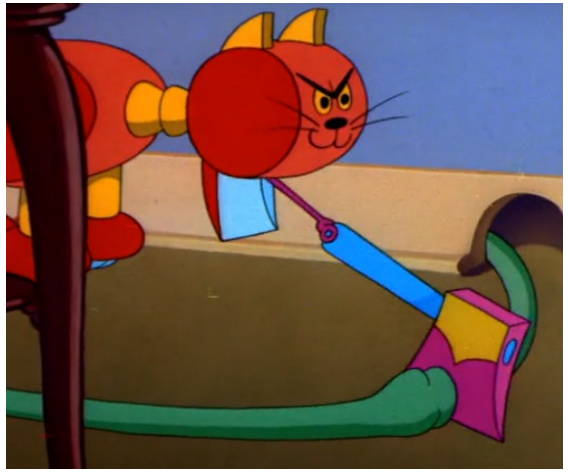
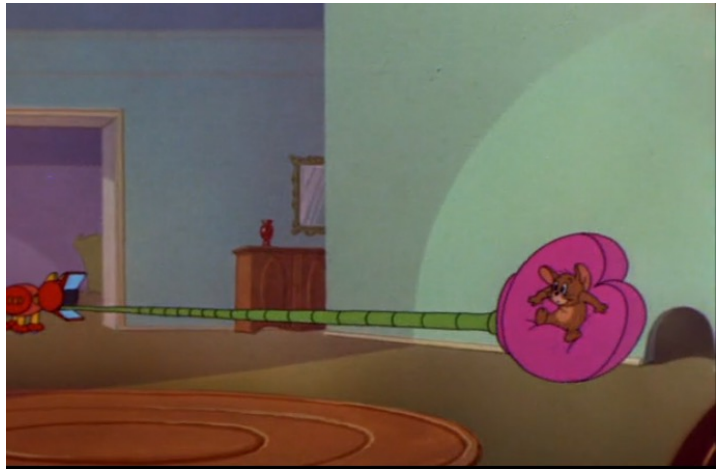


We want an automated program verifier.





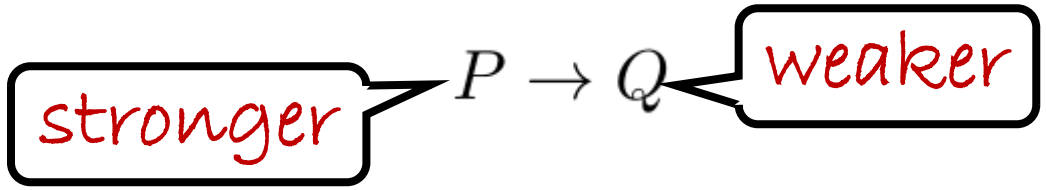
But previous methods (symbolic execution) cannot handle loops

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' → P. We use wp(C, Q) to denote P.

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

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

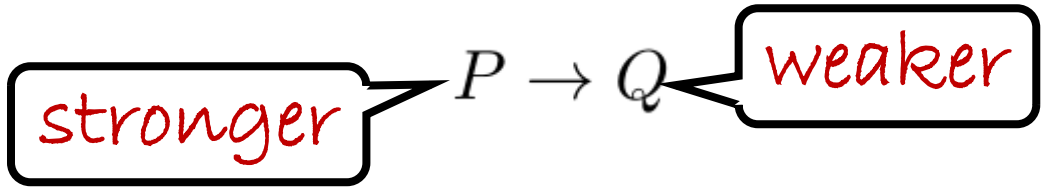
$$\{a = 2 \wedge a > 1\}a := a + 1\{a = 3\}$$

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 → Q'. We use sp(C, P) to denote Q.

strongest postcondition

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

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

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

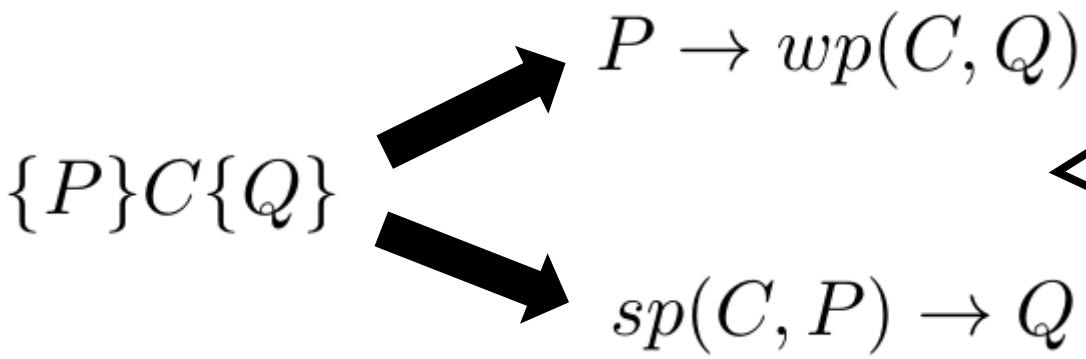
Predicate Transformer

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



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

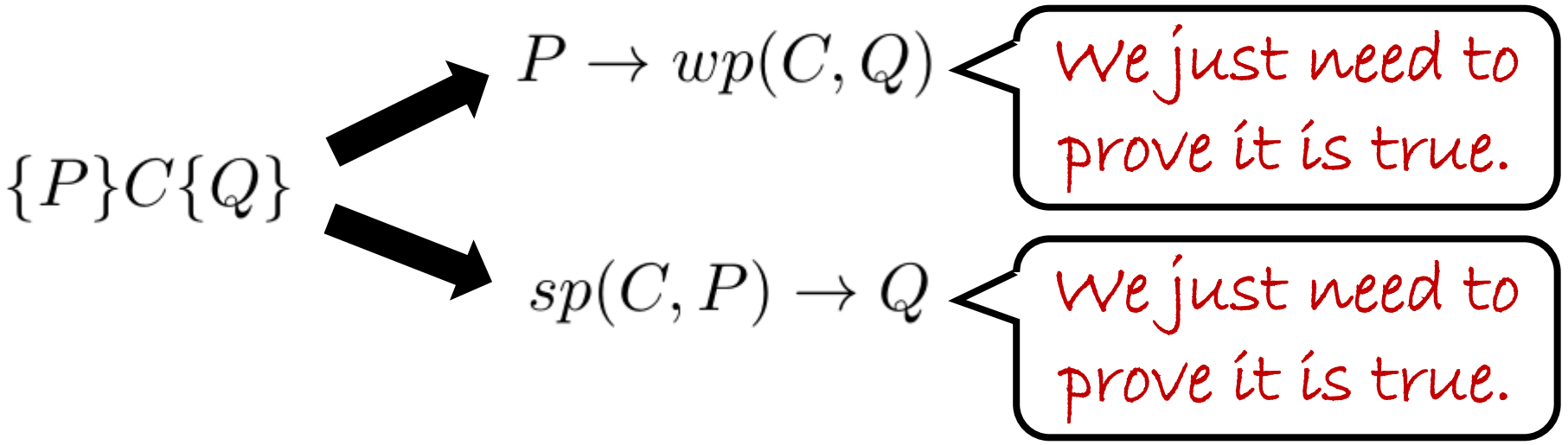
Predicate Transformer

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



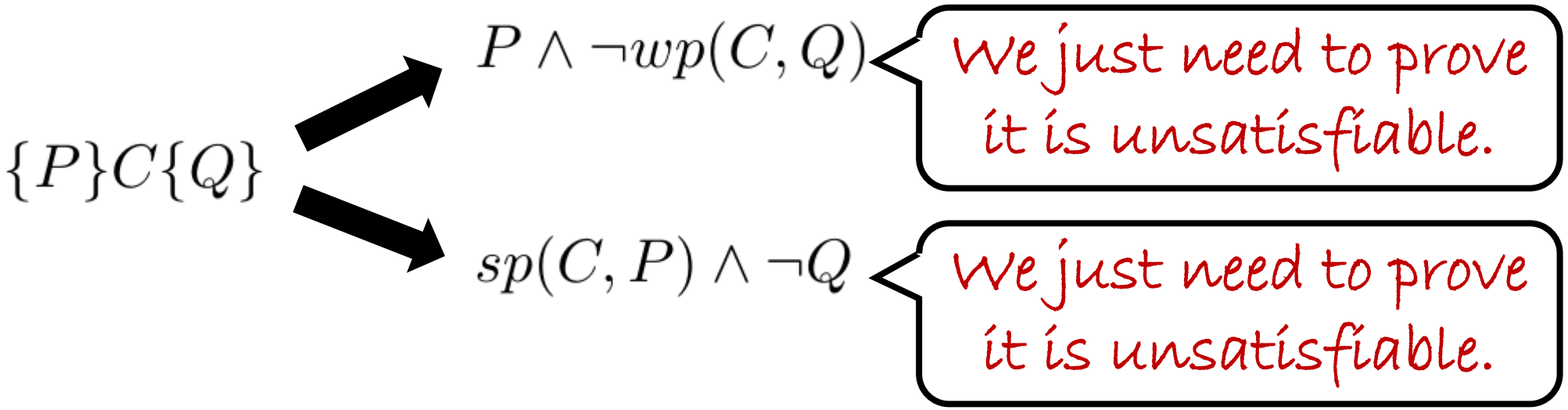
Predicate Transformer

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?

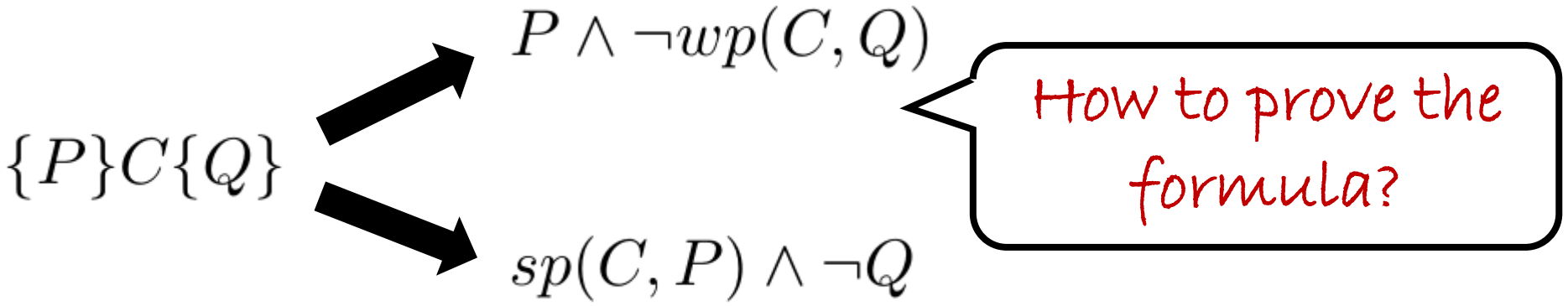
- This is called predicate transformer semantics: how programs transform assertions



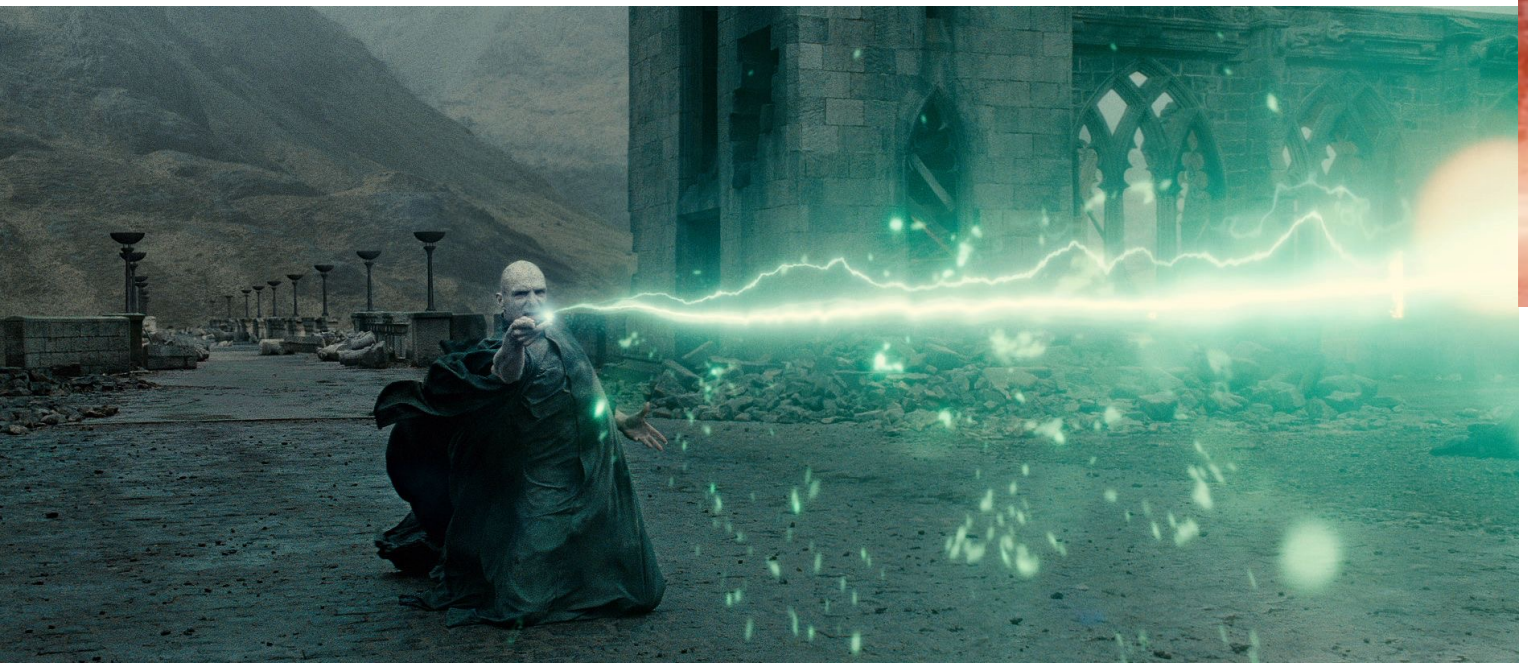
Basic Idea

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.

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

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
- 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 postcondition.

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

- Forward
- Quantifier
- Intuitive

It is more widely used because it's simpler.

- Backward

Quantifier-free

We will focus on wp.

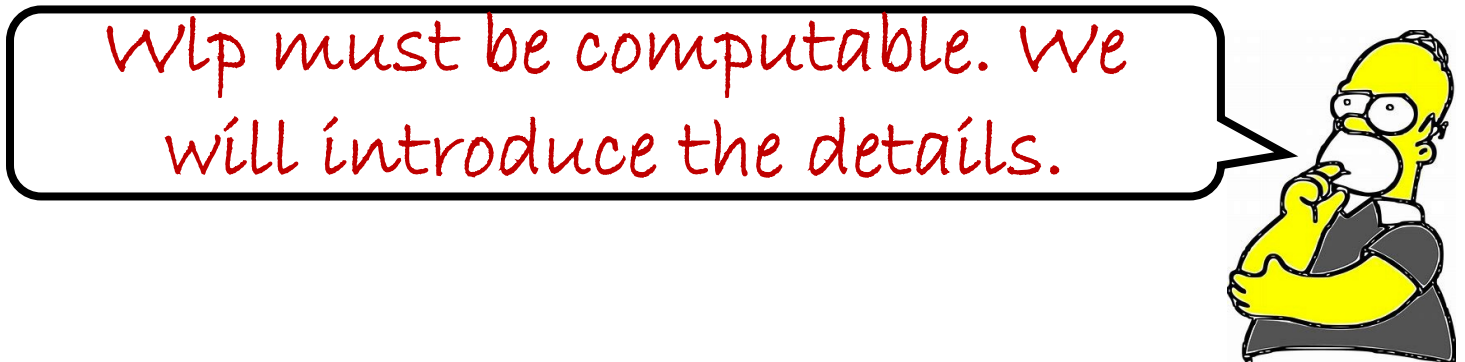


tive

Weakest Precondition

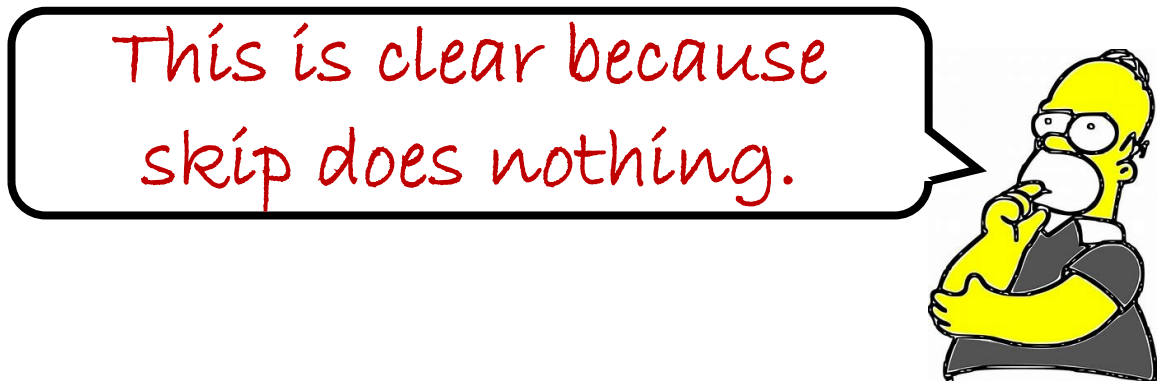
We use wlp(C, Q) instead of wp(C, Q) in following slides:

- 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 $P \rightarrow wlp(C, Q)$



Weakest Precondition

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



Weakest Precondition

This is similar to AS in Hoare logic.

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

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

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

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

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

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

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

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

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

$$= a + 1 = 3 \wedge a + 1 > 0$$

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

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

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

Exercise

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

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



Exercise

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

$$
\begin{aligned}
&wlp(\quad a := b+2;\ \ if(a > 2)\ then\ b := c+1;\ \ else\ b := c-1;,\quad b > 5 \wedge a < 10)\\
=&wlp(a := b+2, ((a > 2) \rightarrow wlp(b := c+1, b > 5 \wedge a < 10)) \wedge (\neg(a > 2) \rightarrow wlp(b := c-1, b > 5 \wedge a < 10)))\\
=&wlp(a := b+2, ((a > 2) \rightarrow (c+1 > 5 \wedge a < 10)) \wedge (\neg(a > 2) \rightarrow (c-1 > 5 \wedge a < 10)))\\
=&((b+2 > 2) \rightarrow (c+1 > 5 \wedge b+2 < 10)) \wedge (\neg(b+2 > 2) \rightarrow (c-1 > 5 \wedge b+2 < 10))
\end{aligned}
$$

Weakest Precondition

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

:-) 呵呵

Weakest Precondition

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

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

The loop invariant needs to be written manually

This can be further proved via the weakest precondition

Lab3 : Dafny

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

post-condition

		Description	Line
⊗	1	possible division by zero	4

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



Dafny

post-condition

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

loop invariant



'►' shortcut: Alt+B

```
Dafny 2.3.0.10506

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

Dafny

pre-condition

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

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

Dafny

pre-condition

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

```
Dafny 2.3.0.10506

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

A Quick Recap

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

PySAT

Z3

Dafny

Soundness and Completeness for Verifiers

DEFINITION

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

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

All verification algorithms introduced before are sound



Soundness and Completeness for Verifiers

DEFINITION

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

Previous algorithms are not complete because of two main reasons:

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

Question: How does an automated verifier implement these?

Basic Design of An Automated Verifier

Input

- Invariants (e.g., Pre-condition) provided by user
- A program to be proved

Workflow

- Convert the program into Control Flow Graph (CFG).
- Insert the invariants into the CFG.
- Translate CFG into Hoare Triples
  - Find the weakest precondition
  - Prove the precondition imply the weakest precondition (Predicate Logic)

Basic Design of An Automated Verifier

Input

- Invariants (e.g., Pre-condition) provided by user
- A program to be proved

Workflow

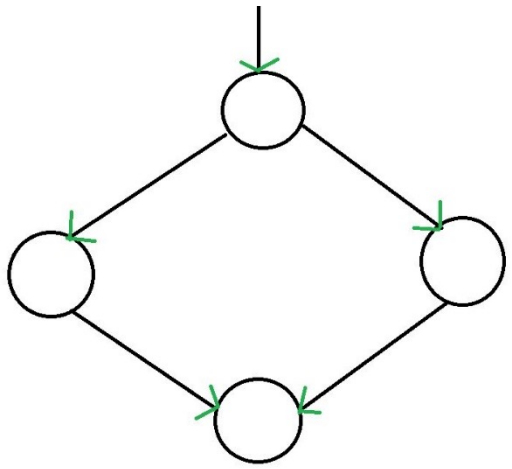
- Convert the program into Control Flow Graph (CFG).
- Insert the invariants into the CFG.
- Translate CFG into Hoare Triples
  - Find the weakest precondition
  - Prove the precondition imply the weakest precondition (Predicate Logic)

Automated Verifier

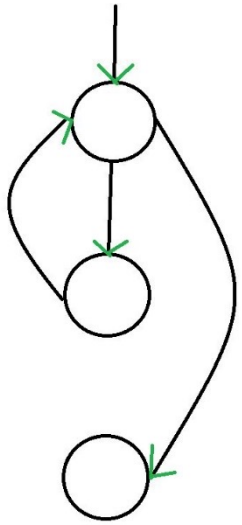
In automated verifiers, programs are represented by the control-flow graph.

DEFINITION

A control-flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

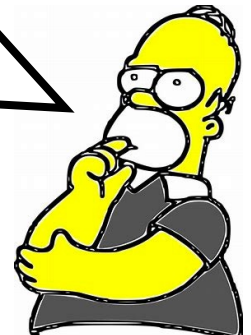


If-then-else



while

We introduce CFG for high level language.

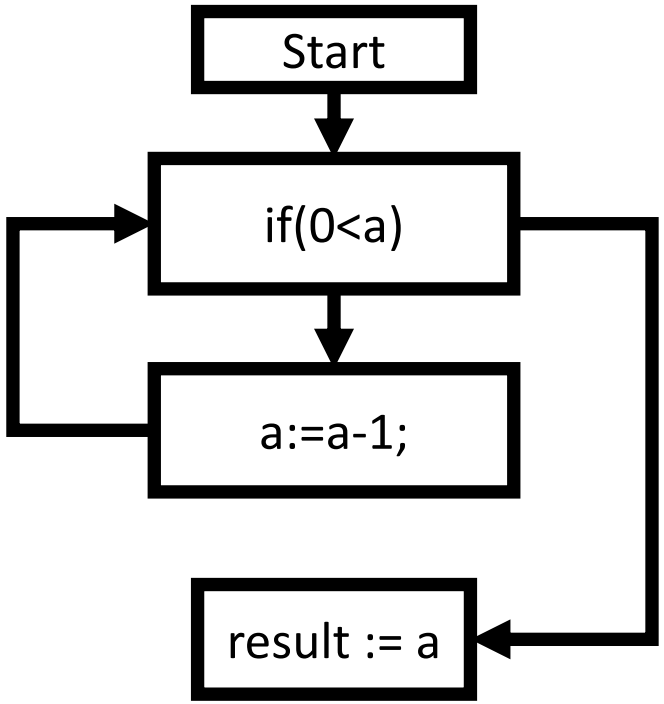


CFG

- Each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets
- Jump targets start a block, and jumps end a block
- Directed edges are used to represent jumps in the control flow

```
int example(int a)
{
	while(0 < a)
	{
		a := a - 1;
	}
	return a;
}
```

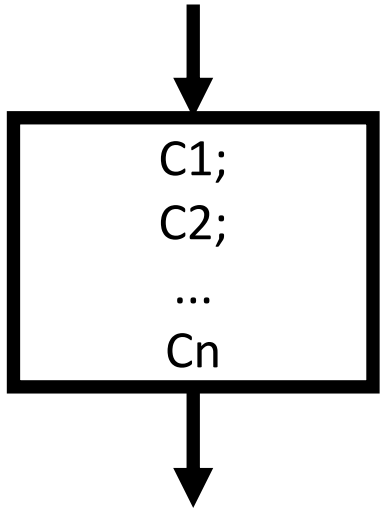




CFG for sequential programs

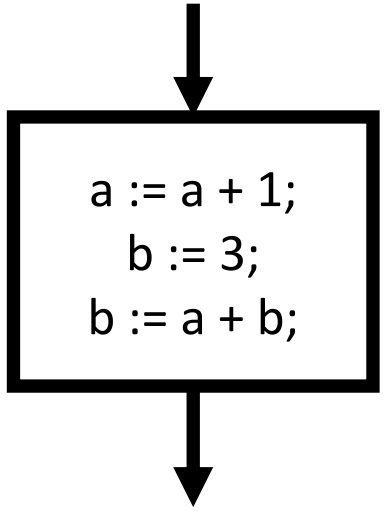
CFG(C1; C2; …; Cn)





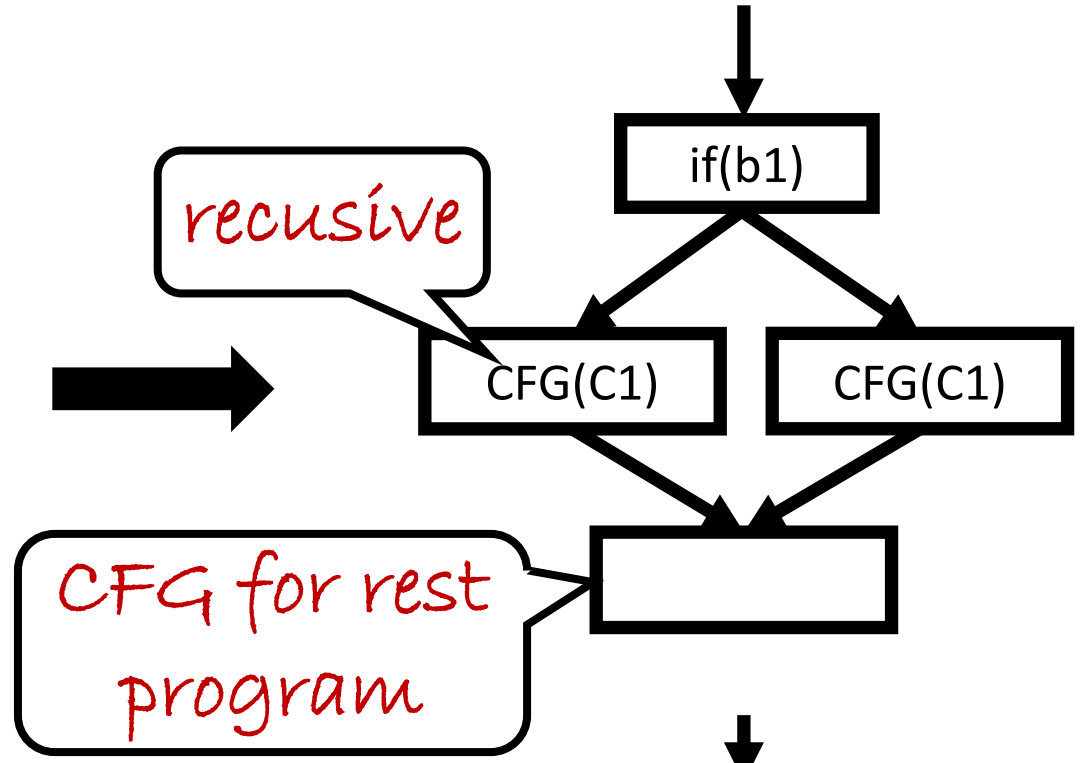

```
……
a := a + 1;
b := 3;
b := a + b;
……
```



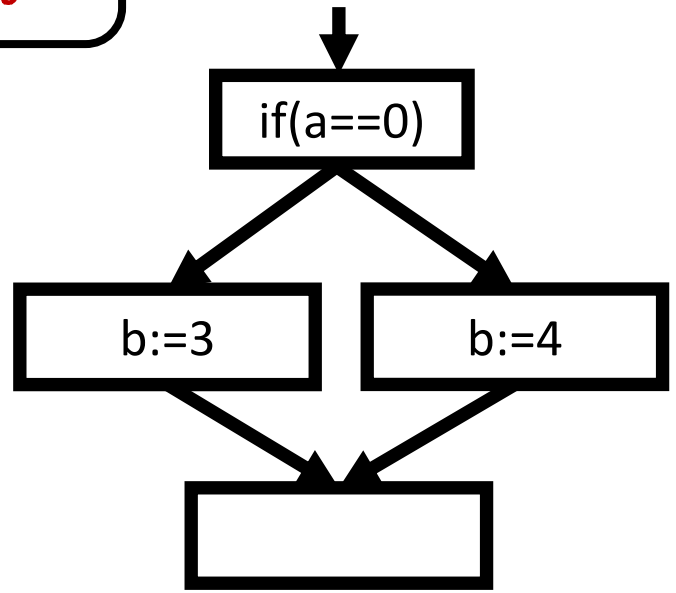


CFG for if-then-else

CFG(if(b1) C1; else C2)

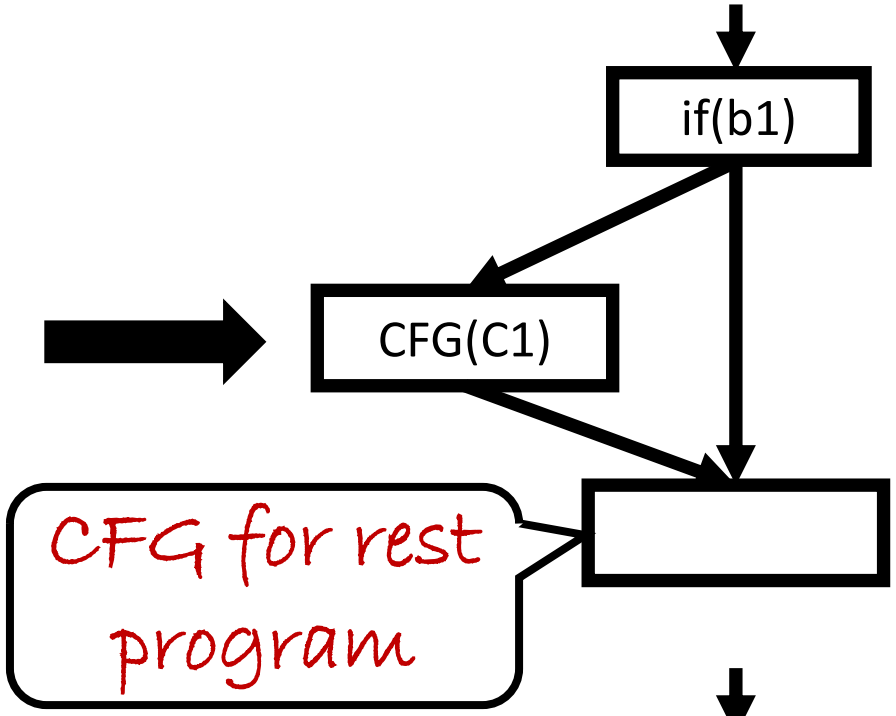



```
……
if(a == 0)
        b := 3;
else
        b := 4;
……
```

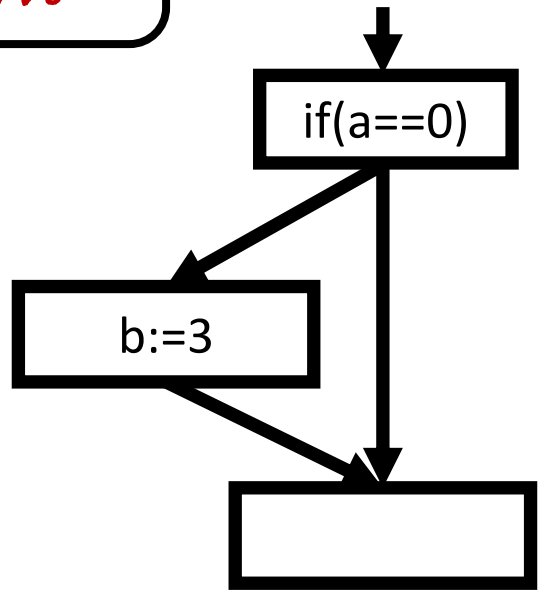


CFG for if-then

CFG(if(b1) C1)

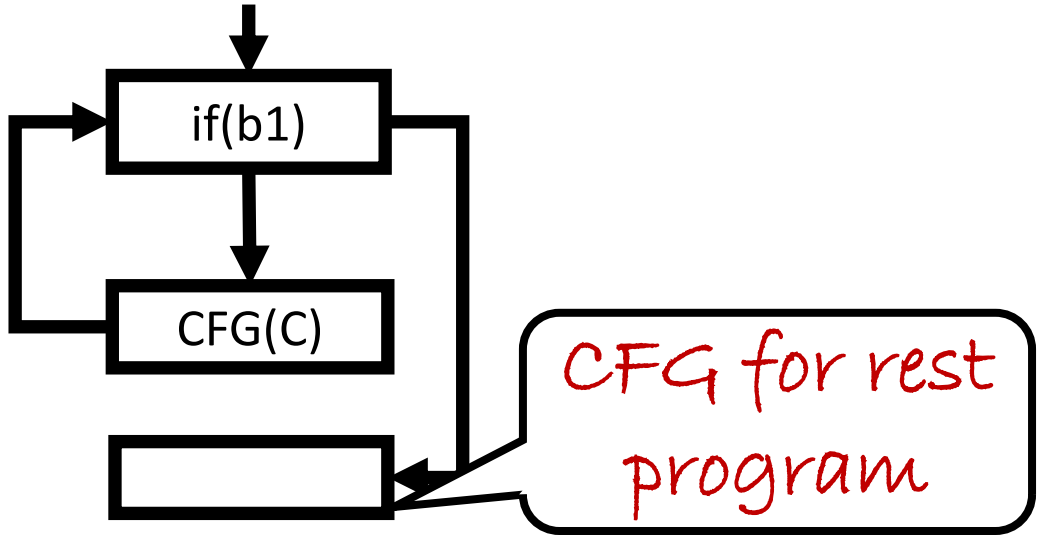



```
……
if(a == 0)
        b := 3;
……
```

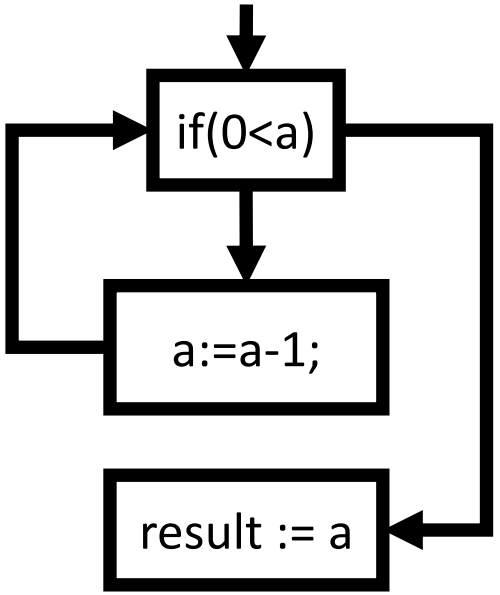


CFG for while

CFG(while(b1) C)

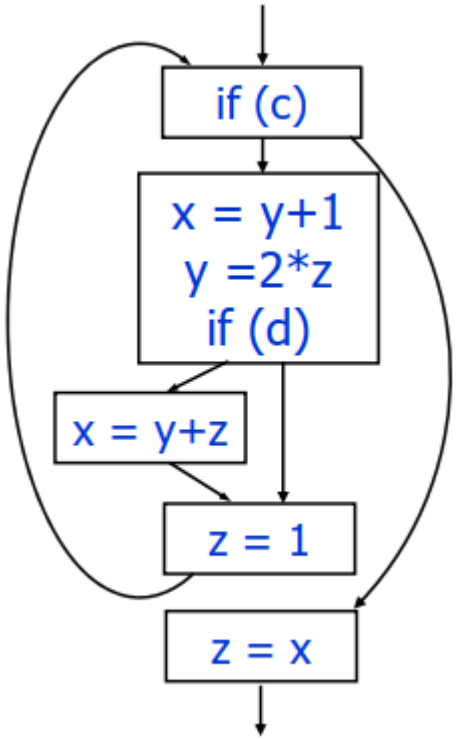



```
……
while(0 < a)
{
        a := a - 1;
}
return a;
```



Exercise

```
while (c) {
        x := y + 1;
        y := 2 * z;
        if (d) x := y+z;
        z := 1;
}
z := x;
```



Construct the CFG recursively.



Basic Design of An Automated Verifier

Input

- Invariants (e.g., Pre-condition) provided by user
- A program to be proved

Workflow

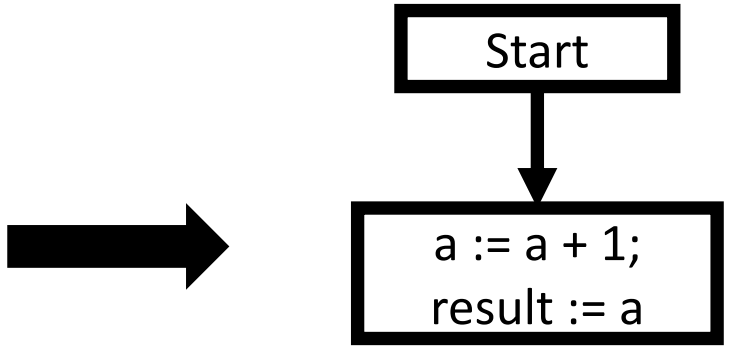
- Convert the program into Control Flow Graph (CFG).
- Insert the invariants into the CFG.
- Translate CFG into Hoare Triples
  - Find the weakest precondition
  - Prove the precondition imply the weakest precondition

Automated Verifier

BASIC IDEA

Automated Verifier transforms control flow graph to logical formulas and uses SMT solvers to solve it.

```
{a=1}
int f(int a)
{
        a := a + 1;
        return a;
}
{result=2}
```



How to denote pre-/post-condition in CFG?



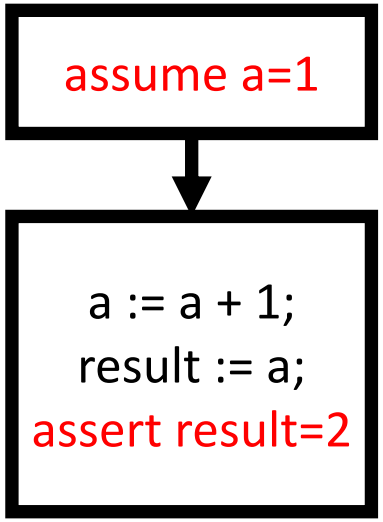
Automated Verifier

To denote pre-/post-conditions, we need add new statements.

- assume A: assume A is true
- assert A: if A is true, then execute the rest program, else go to error state

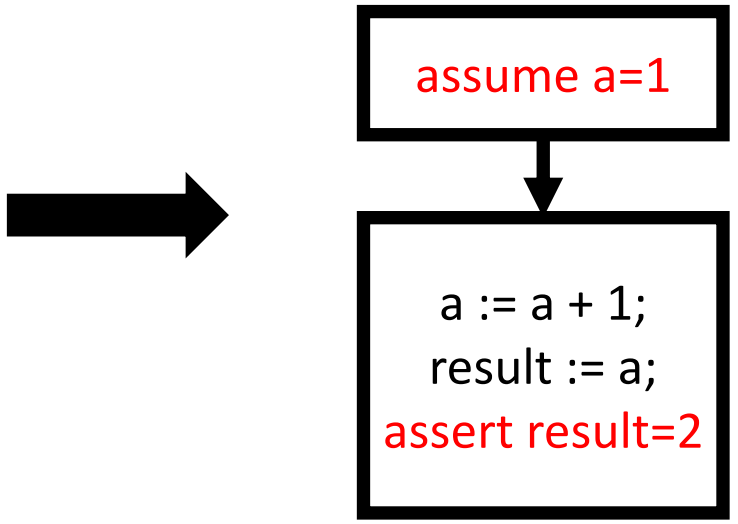
```
{a=1}
int f(int a)
{
        a := a + 1;
        return a;
}
{result=2}
```





Automated Verifier

```
{a=1}
int f(int a)
{
        a := a + 1;
        return a;
}
{result=2}
```



Because there are no loops, we can still use weakest preconditions to reduce the CFG to formulas.

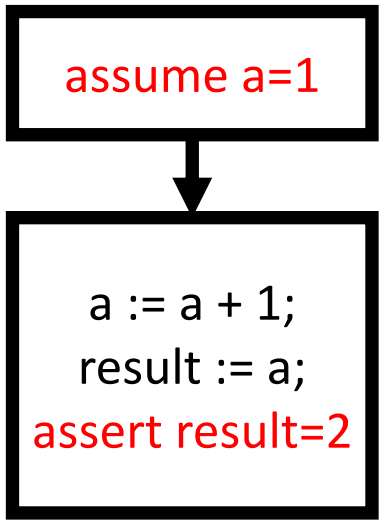
$$wlp(assert\ P, Q) = P \wedge Q$$

$$wlp(assume\ P, Q) = P \rightarrow Q$$

Automated Verifier

```
{a=1}
int f(int a)
{
        a := a + 1;
        return a;
}
{result=2}
```







$$a = 1 \rightarrow$$

$$wlp(a := a + 1; result := a, result = 2)$$



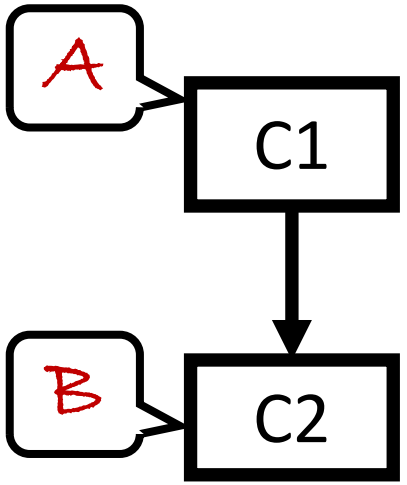
$$a = 1 \rightarrow$$

$$a + 1 = 2$$

How to compute weakest precondition from CFG?



Automated Verifier



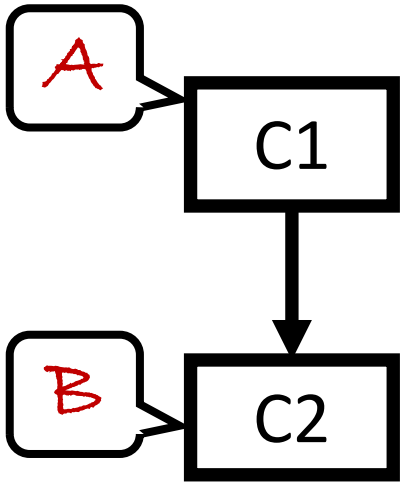
A_{ok} is true if the program is in a state from which all executions beginning from block A are correct.

We define the correctness of CFG:

$$A_{ok} \equiv wlp(C1, B_{ok})$$

Prove a program is correct becomes prove A_{ok} is True.

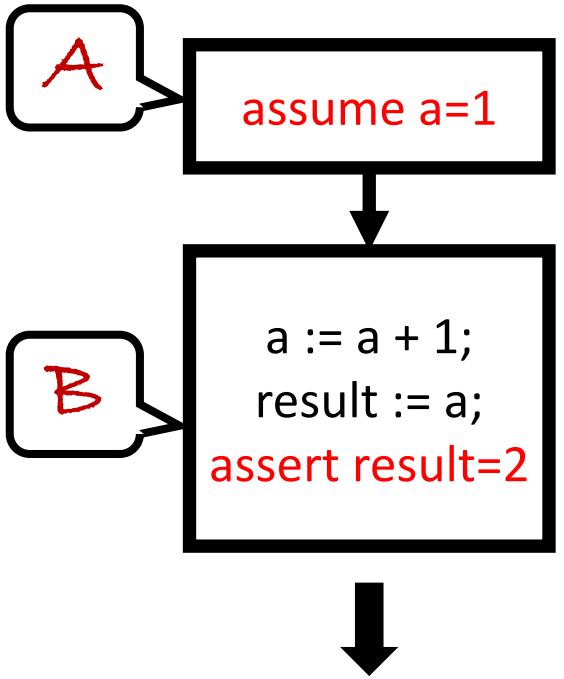
Automated Verifier



We define the correctness

of CFG like this:

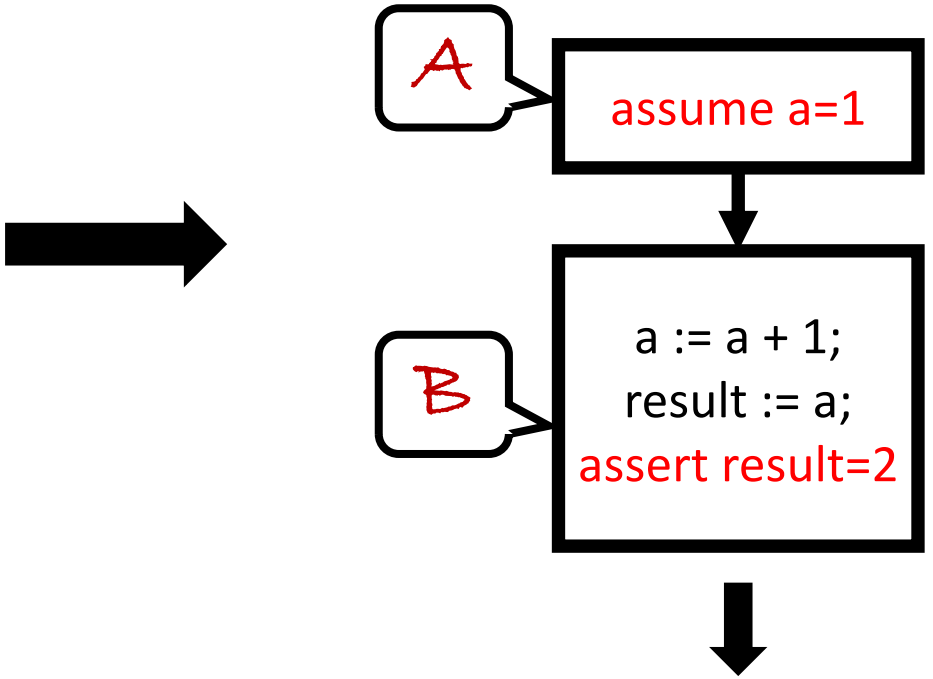
$$A_{ok} \equiv wlp(C1, B_{ok})$$



$$\begin{aligned} B_{ok} &\equiv wlp(a := a+1; result := a; assert\ result = 2, T) \\ &\equiv wlp(a := a+1; result := a, result = 2) \\ &\equiv wlp(a := a+1, a = 2) \\ &\equiv a + 1 = 2 \\ A_{ok} &\equiv wlp(assume\ a = 1, B_{ok}) \\ &\equiv a = 1 \rightarrow a + 1 = 2 \end{aligned}$$

Automated Verifier

```
{a=1}
int f(int a)
{
        a := a + 1;
        return a;
}
{result=2}
```



$$a = 1 \rightarrow$$

$$wlp(a := a + 1; result := a, result = 2)$$

$$a = 1 \rightarrow$$

$$a + 1 = 2$$

=

$$B_{ok} \equiv wlp(a := a + 1; result := a; assert\ result = 2, T)$$

$$\equiv wlp(a := a + 1; result := a, result = 2)$$

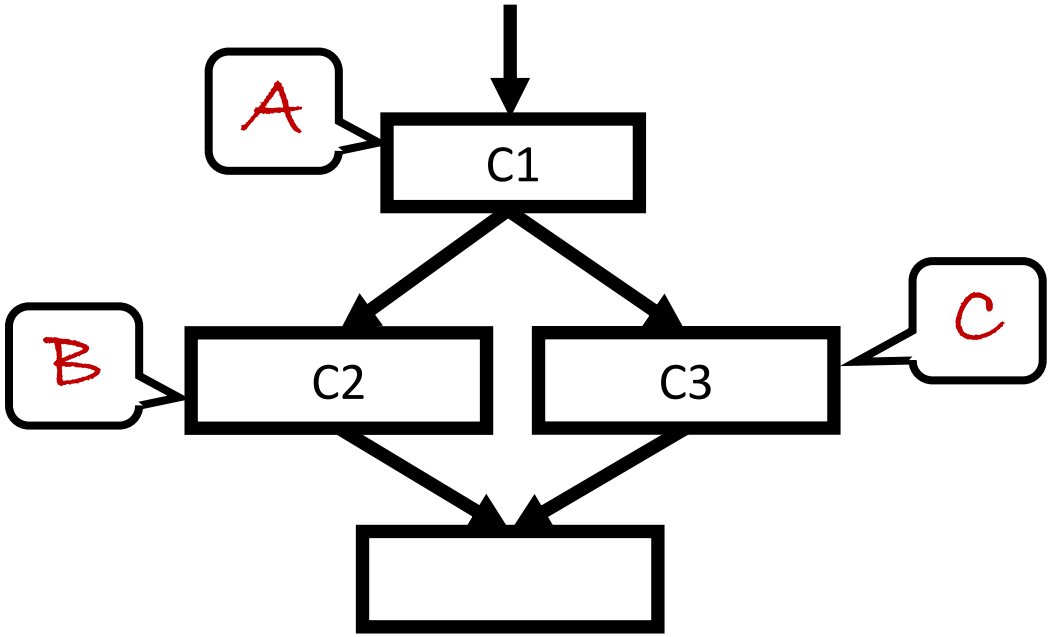
$$\equiv wlp(a := a + 1, a = 2)$$

$$\equiv a + 1 = 2$$

$$A_{ok} \equiv wlp(assume\ a = 1, B_{ok})$$

$$\equiv a = 1 \rightarrow a + 1 = 2$$

Automated Verifier

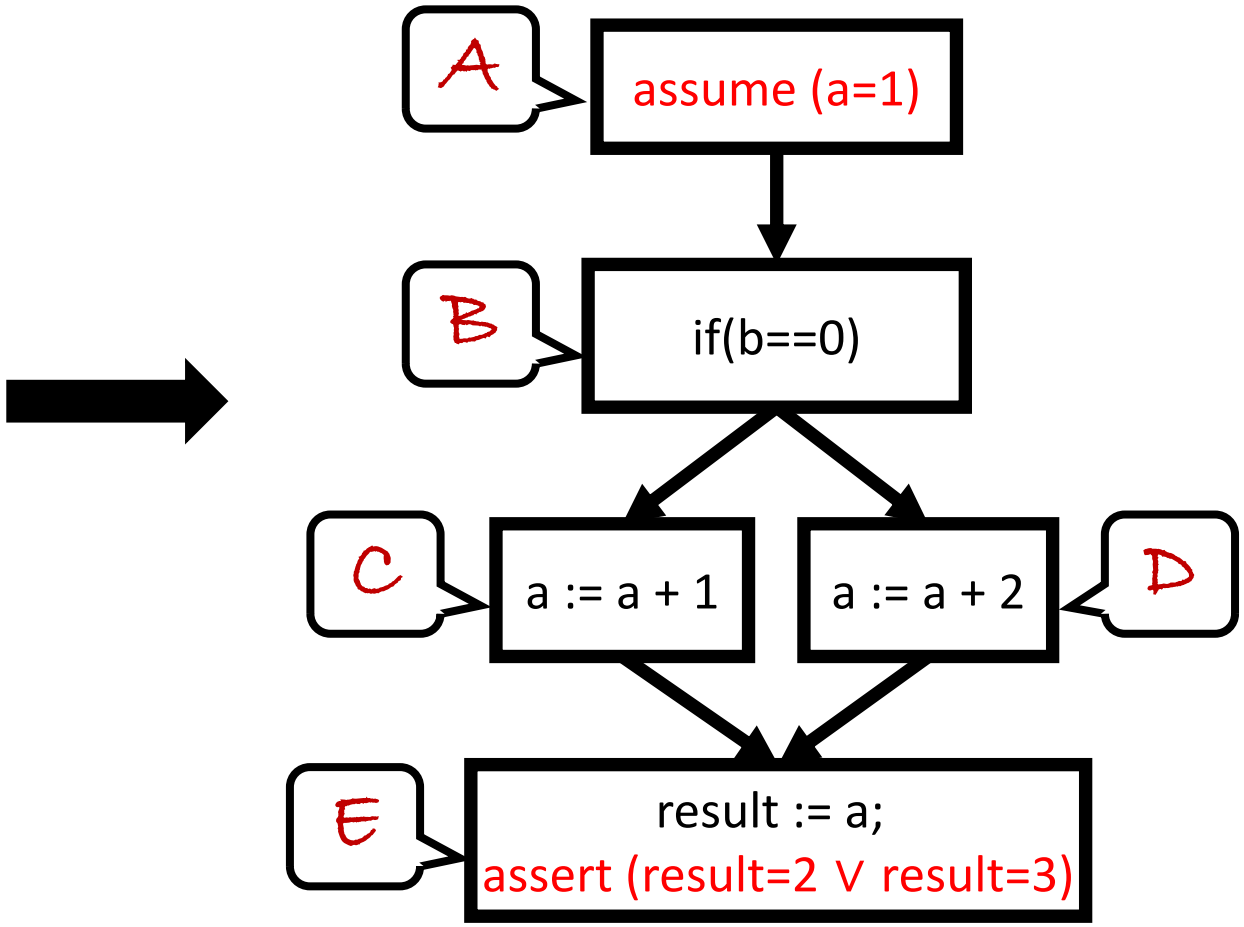


$$A_{ok} \equiv wlp(C1, \bigwedge_{S \text{ is } Succ(A)} S_{ok})$$

Succ(A) denotes the set of successors of A in CFG

Automated Verifier

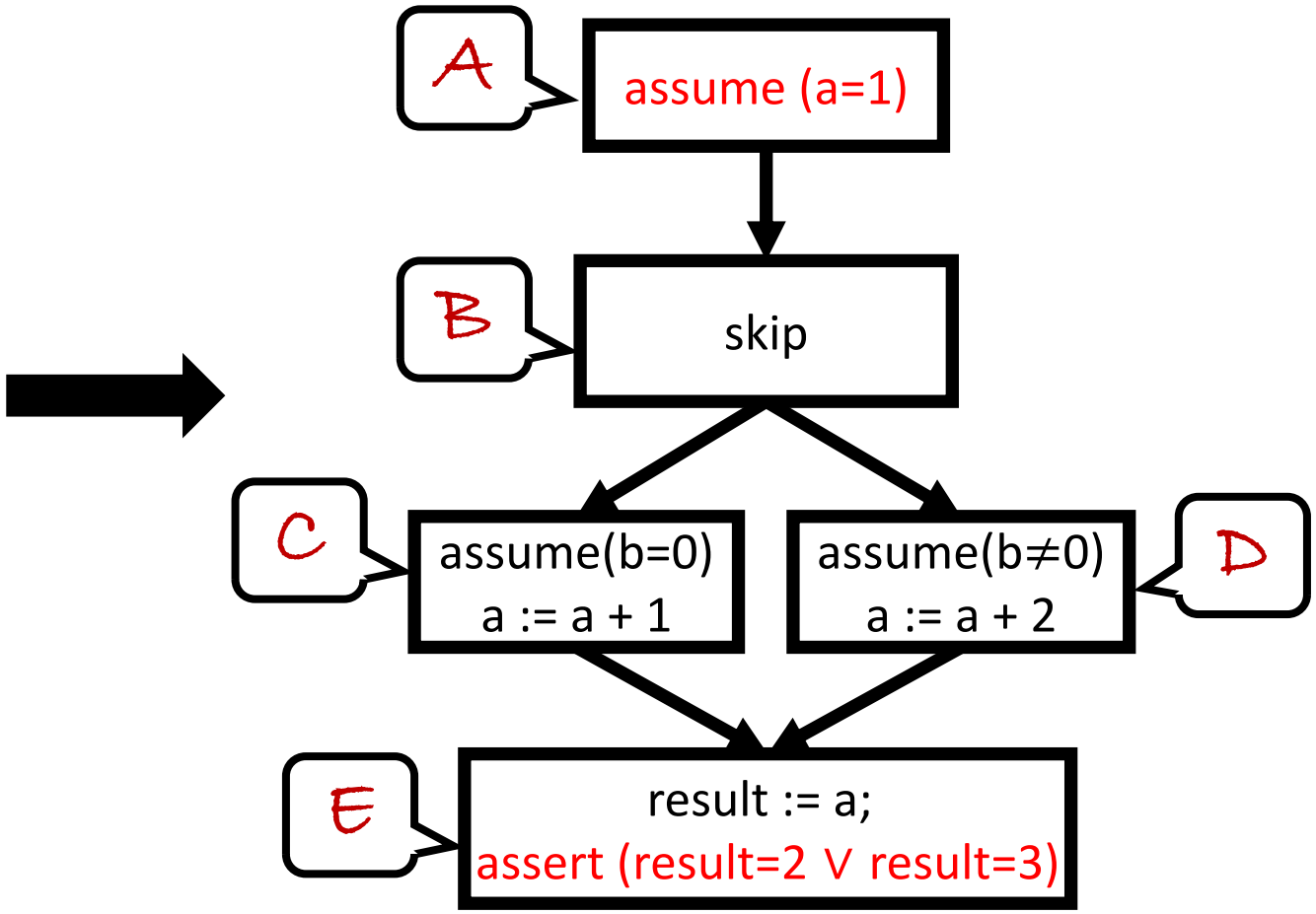
```
{a=1}
int f(int a, int b)
{
        if(b == 0)
                a := a + 1;
        else
                a := a + 2;
        return a;
}
{result=2 ∨ result=3}
```



When addressing branches, we need to add branch condition information to CFG.

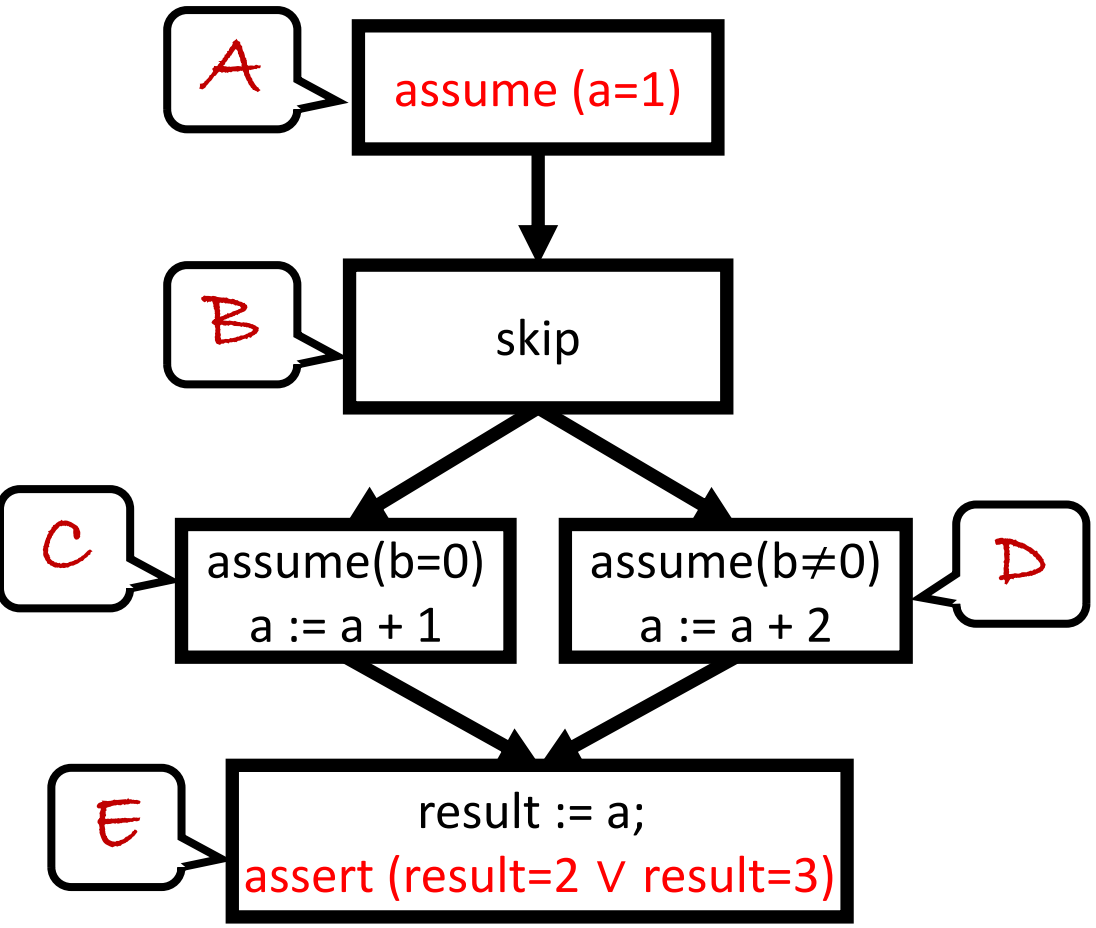
Automated Verifier

```
{a=1}
int f(int a, int b)
{
        if(b == 0)
                a := a + 1;
        else
                a := a + 2;
        return a;
}
{result=2 ∨ result=3}
```



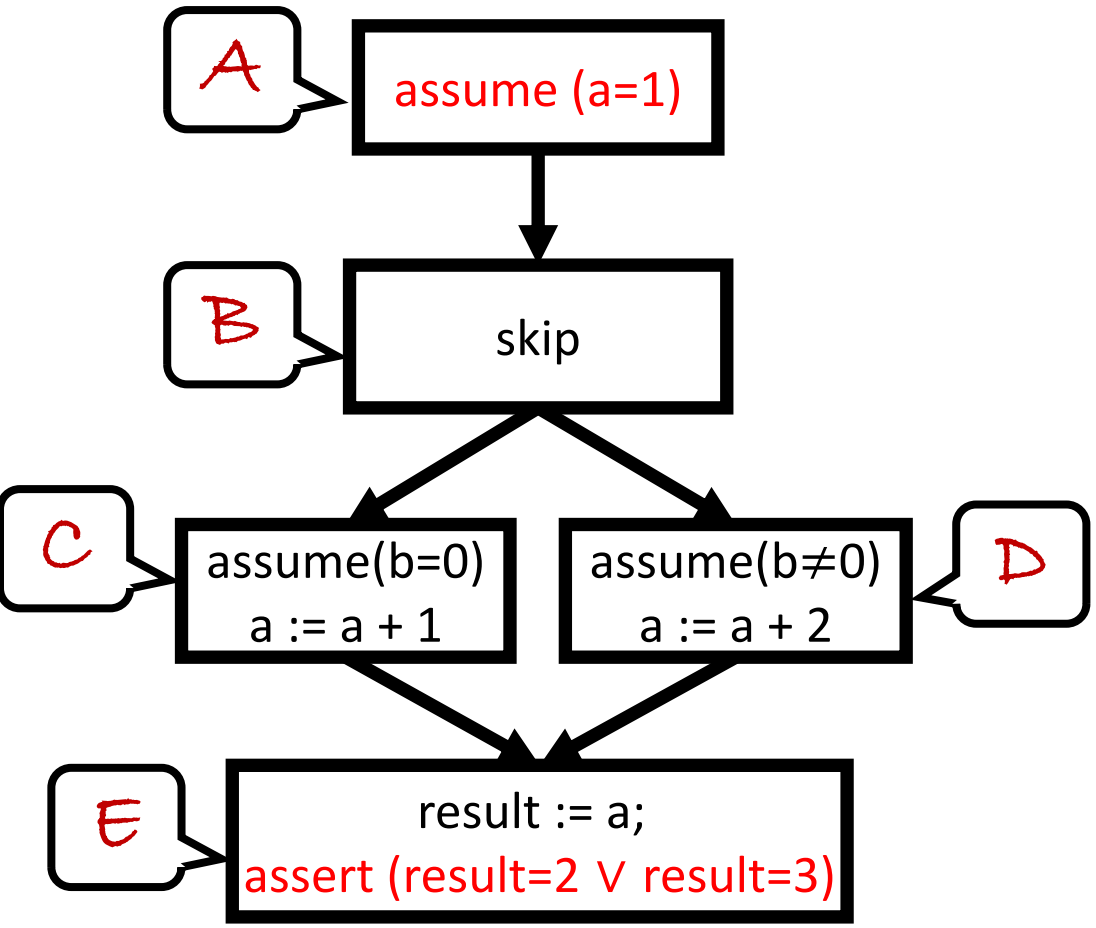
When addressing branches, we need to add branch condition information to CFG.

Automated Verifier



$$E_{ok} \equiv wlp(result := a; assert(result = 2 \vee result = 3), T)$$
$$\equiv wlp(result := a, result = 2 \vee result = 3)$$
$$\equiv (a = 2 \vee a = 3)$$

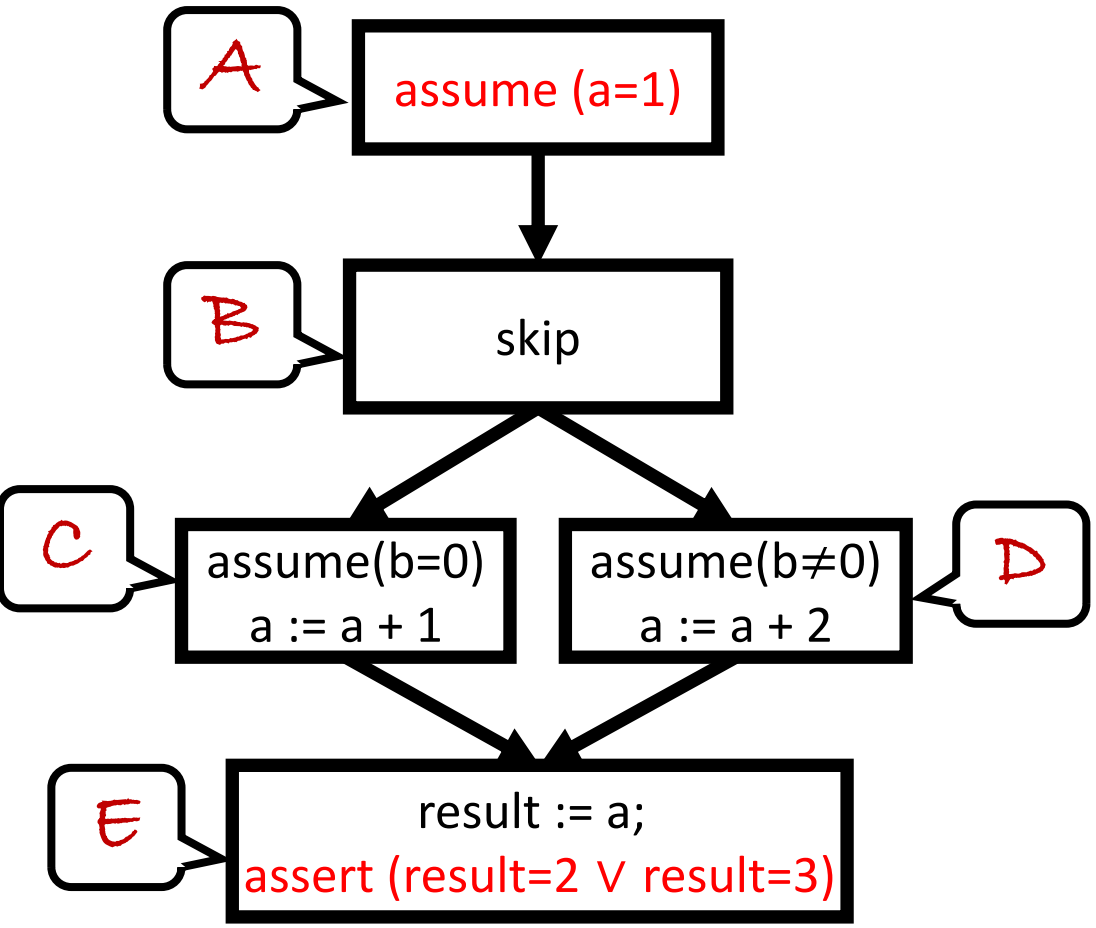
Automated Verifier



$$E_{ok} \equiv (a = 2 \vee a = 3)$$

$$\begin{aligned} D_{ok} &\equiv wlp(assume(b \neq 0); a := a + 2, E_{ok}) \\ &\equiv wlp(assume(b \neq 0), (a + 2 = 2 \vee a + 2 = 3)) \\ &\equiv (b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3)) \end{aligned}$$

Automated Verifier



$$E_{ok} \equiv (a = 2 \vee a = 3)$$

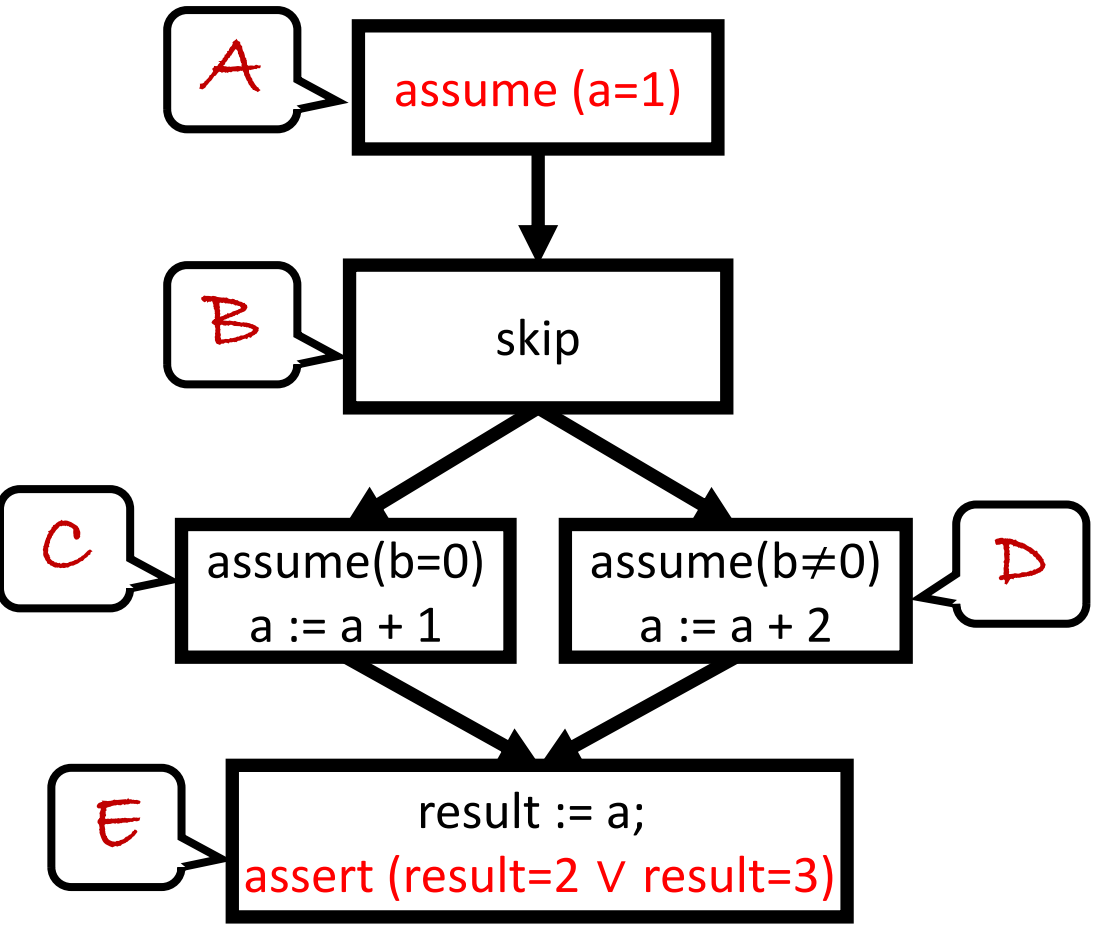
$$D_{ok} \equiv (b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

$$C_{ok} \equiv wlp(assume(b = 0); a := a + 1, E_{ok})$$

$$\equiv wlp(assume(b = 0), (a + 1 = 2 \vee a + 1 = 3))$$

$$\equiv (b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3))$$

Automated Verifier



$$E_{ok} \equiv (a = 2 \vee a = 3)$$

$$D_{ok} \equiv (b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

$$C_{ok} \equiv (b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3))$$

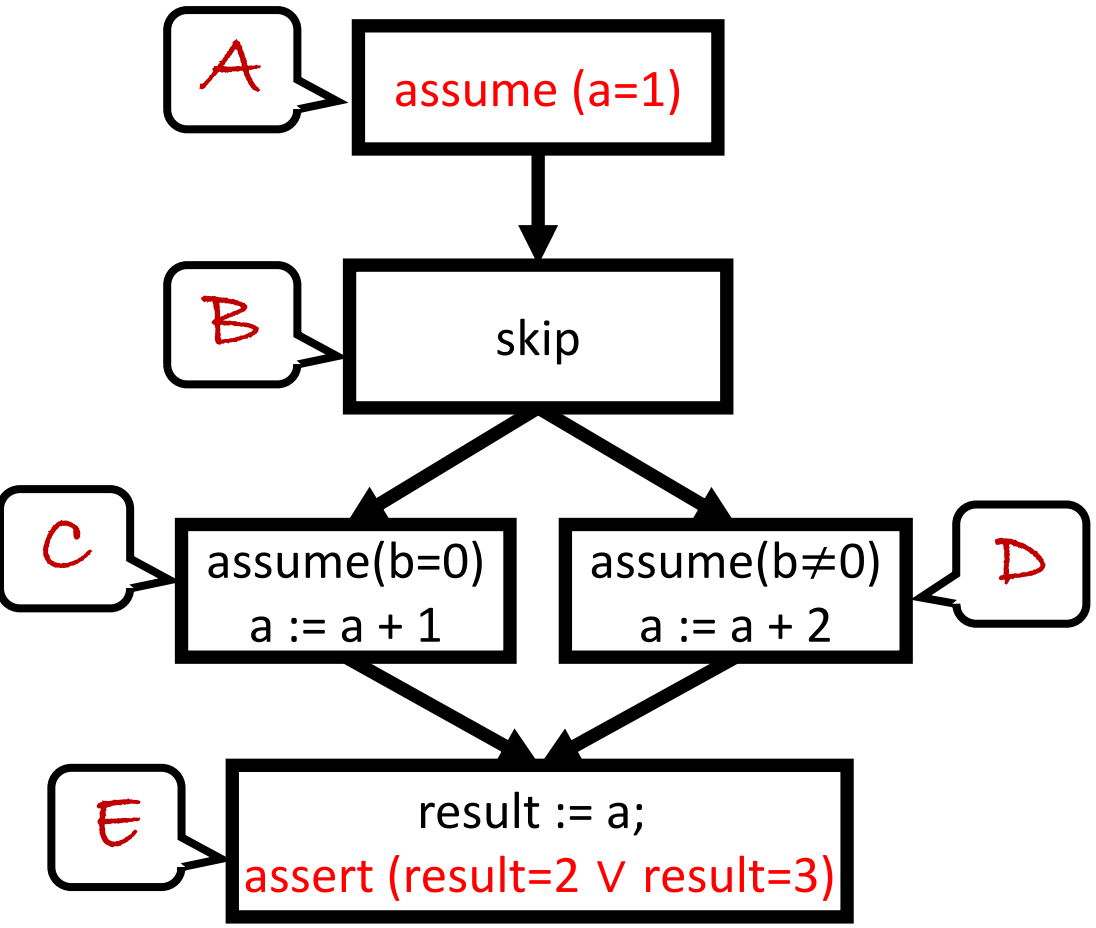
$$B_{ok} \equiv wlp(skip, C_{ok} \wedge D_{ok})$$

$$\equiv C_{ok} \wedge D_{ok}$$

$$\equiv (b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3)) \wedge$$

$$(b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

Automated Verifier



$$E_{ok} \equiv (a = 2 \vee a = 3)$$

$$D_{ok} \equiv (b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

$$C_{ok} \equiv (b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3))$$

$$B_{ok} \equiv (b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3)) \wedge$$
$$(b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

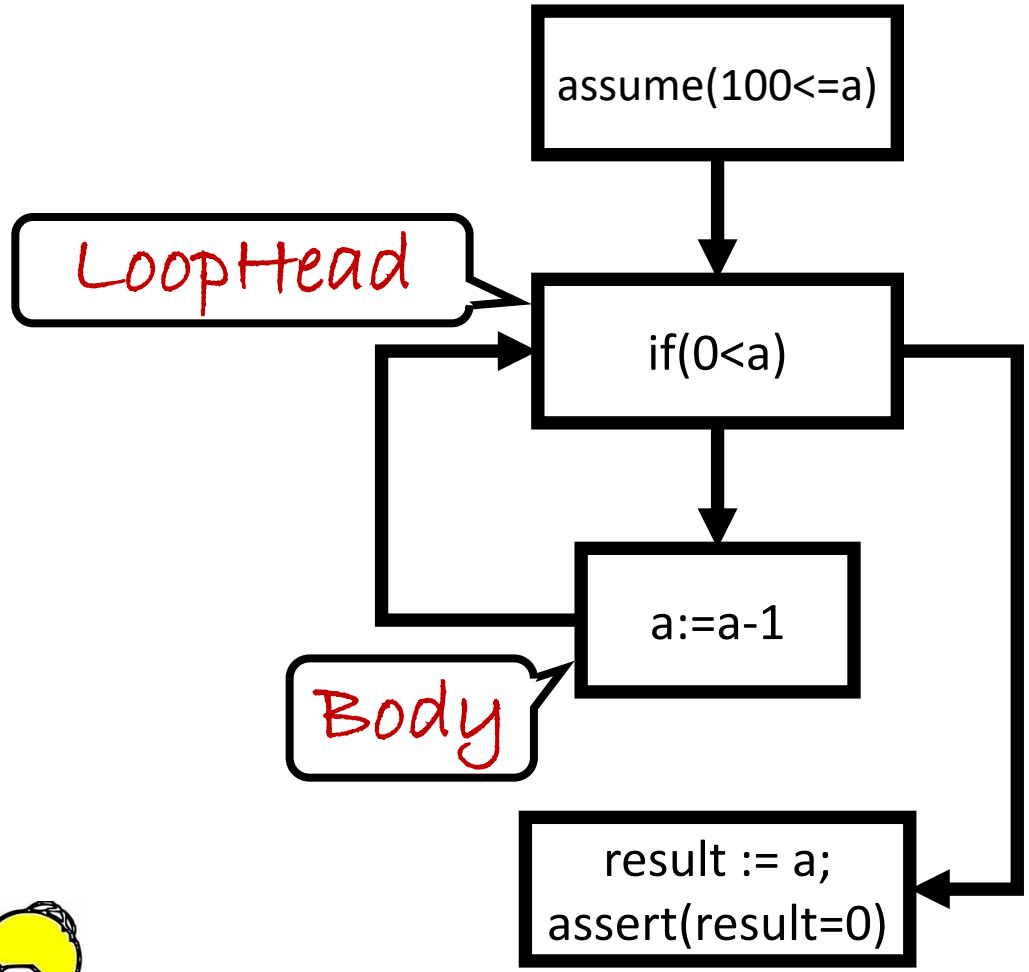
$$A_{ok} \equiv wlp(assume(a = 1), B_{ok})$$
$$\equiv (a = 1) \rightarrow$$
$$(b = 0 \rightarrow (a + 1 = 2 \vee a + 1 = 3)) \wedge$$
$$(b \neq 0 \rightarrow (a + 2 = 2 \vee a + 2 = 3))$$

backward

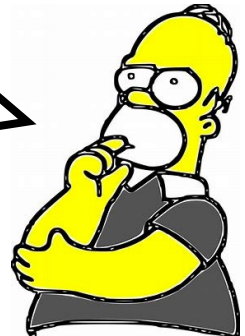
Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





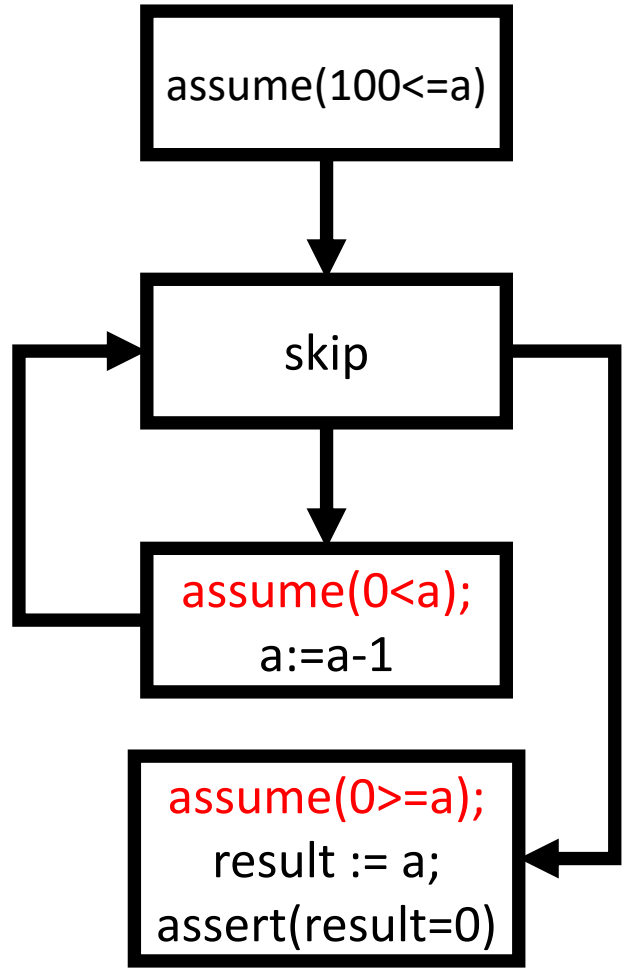
How to address loops?



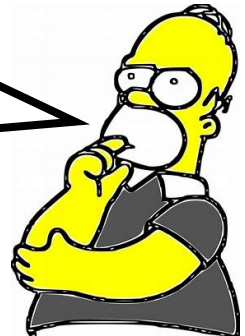
Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





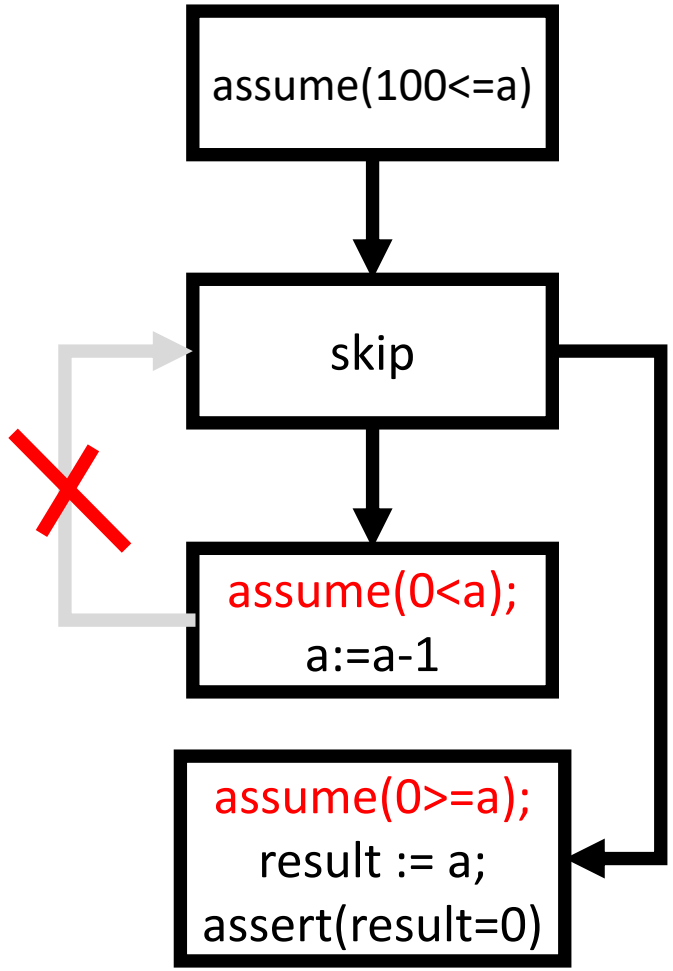
Like that in branch, we need to add loop guard.



Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





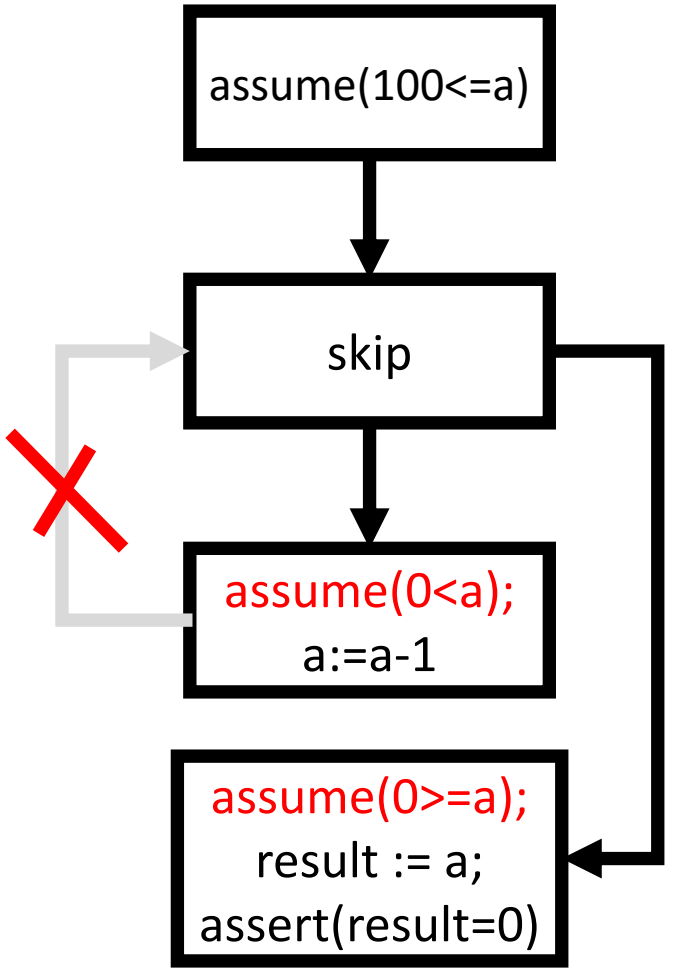
We must eliminate loops.
Otherwise, we cannot use
backward methods like before.



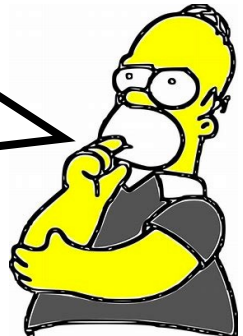
Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





But it's clearly not correct. We change the meaning of programs.



A Quick Recap

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

Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```

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

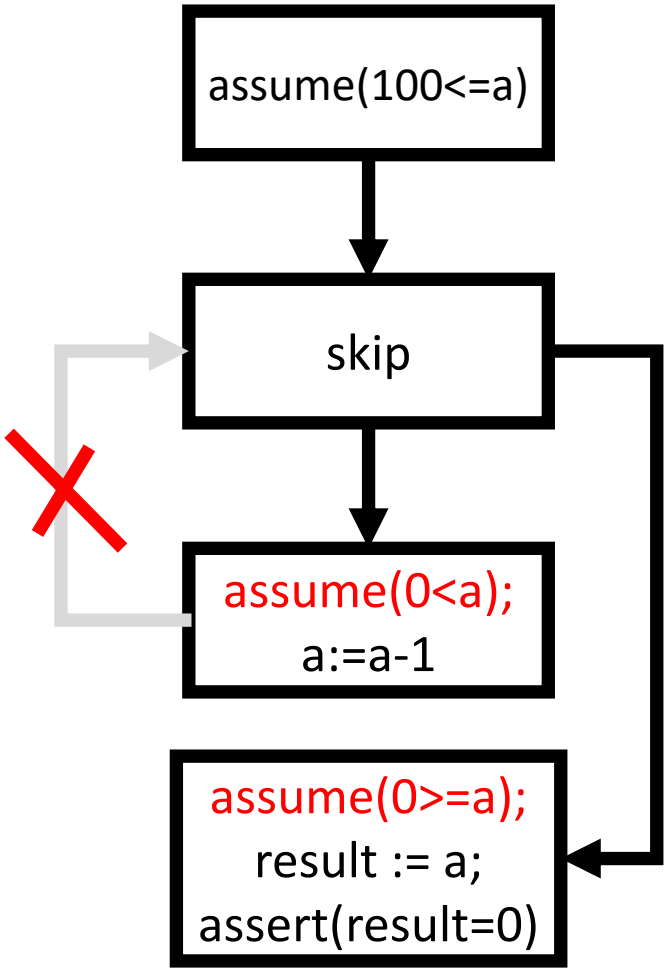
1) Prove invariant
2) Use invariant to generate wlp



Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```

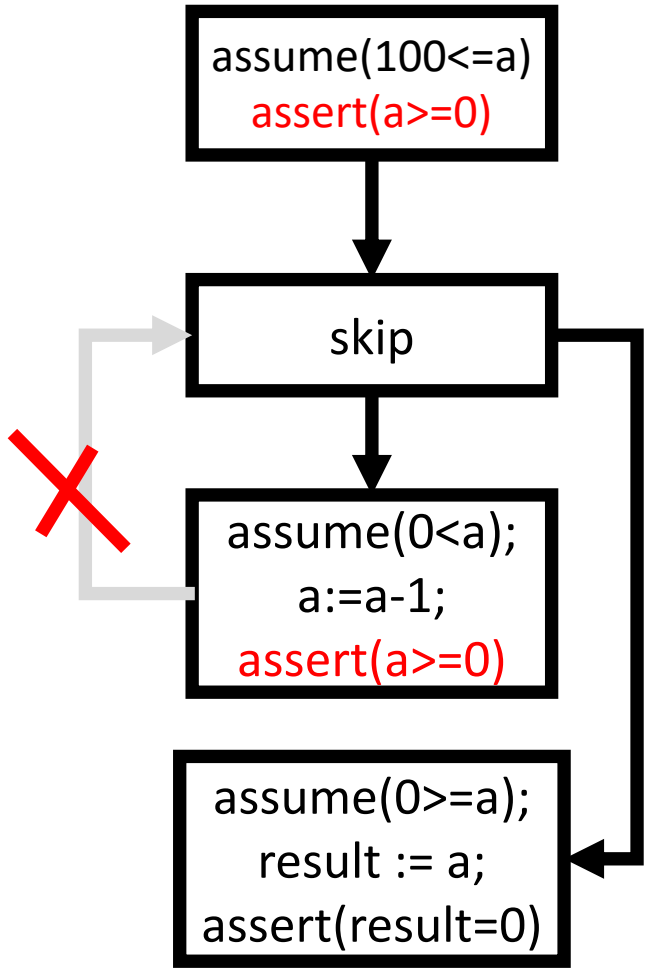




Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





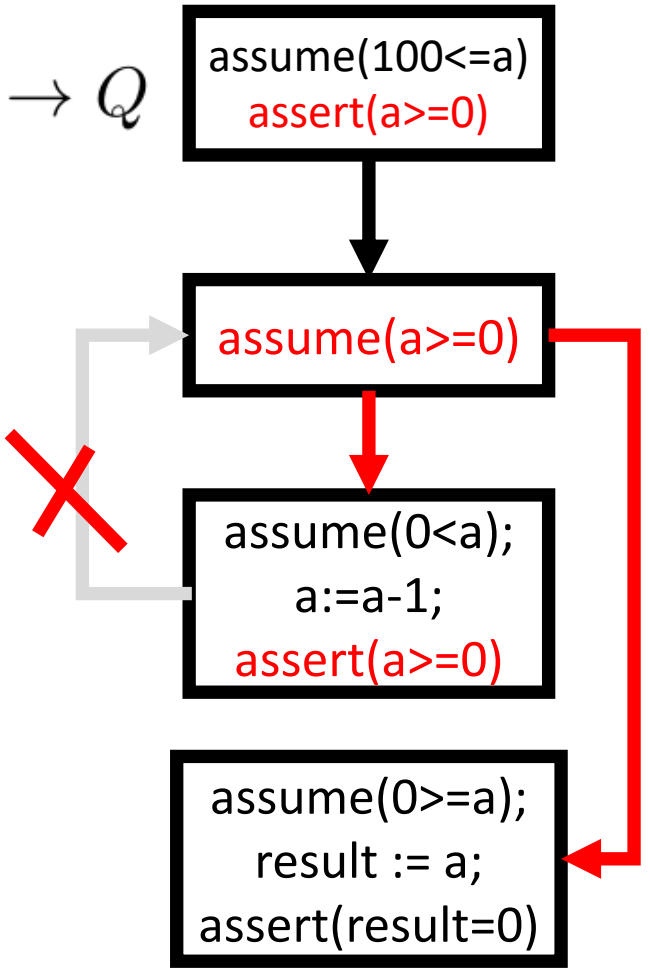
We can use assert to check invariant before and after the loop body.



Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```

$$wlp(assume\ P, Q) = P \rightarrow Q$$

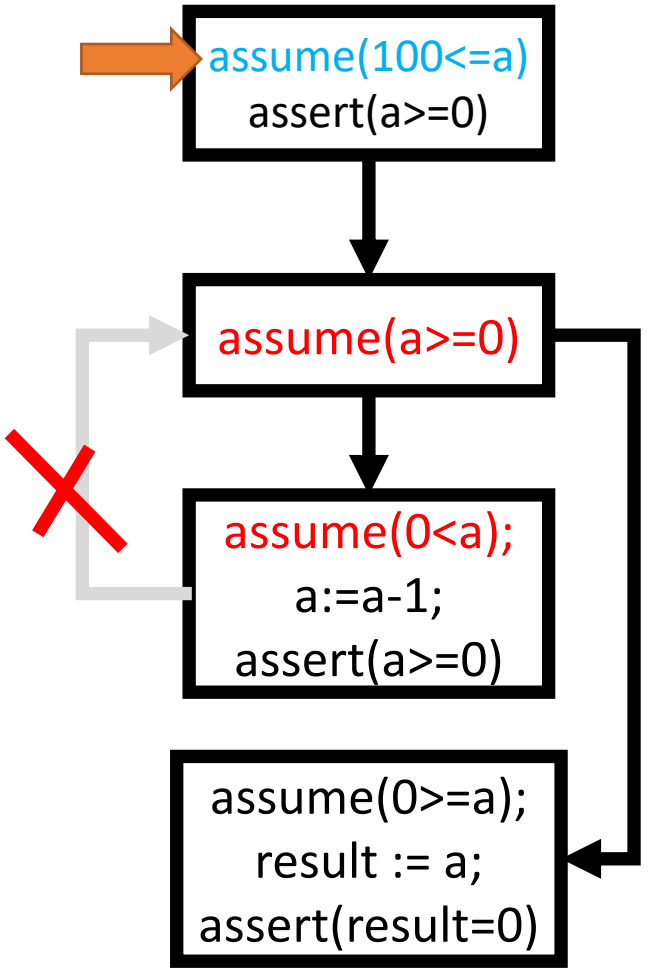


We can use assume to add invariant as premise.

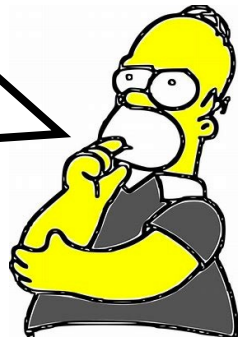


Automated Verifier

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

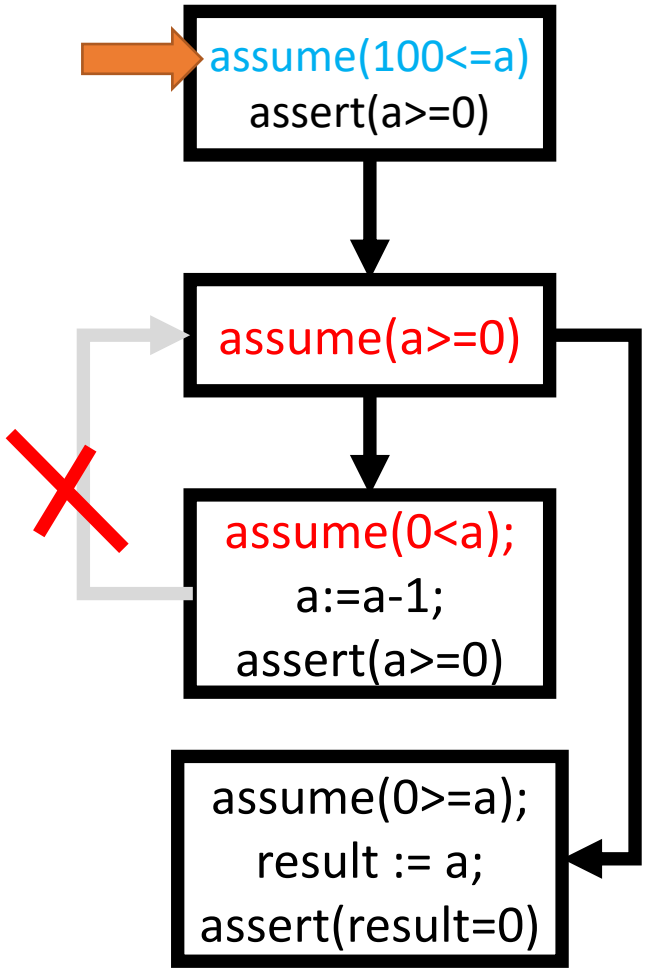


There is a problem. According to WHP, the CFG introduces extra information to prove invariant.

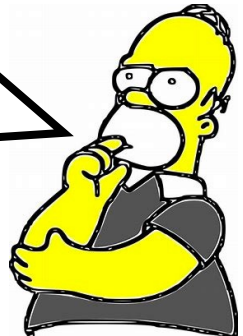


Automated Verifier

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



We cannot directly remove the condition. But we can use havoc.



Automated Verifier

To address loops, we need another new statement.

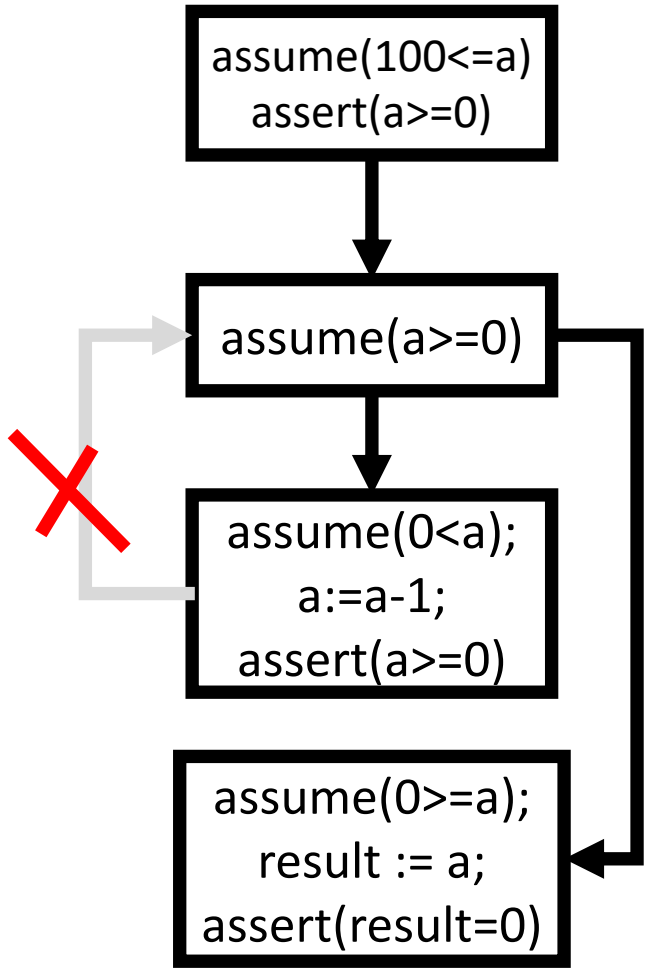
- havoc a: set a to an arbitrary value
- Variables updated in loop body are called loop targets
- We need to havoc loop targets before loop. Otherwise, it is not a strict proof of invariant

$$wlp(havoc\ a, Q) = (\forall v) Q[v/a]$$

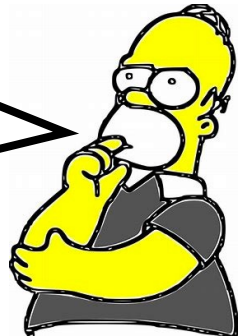
Automated Verifier

```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





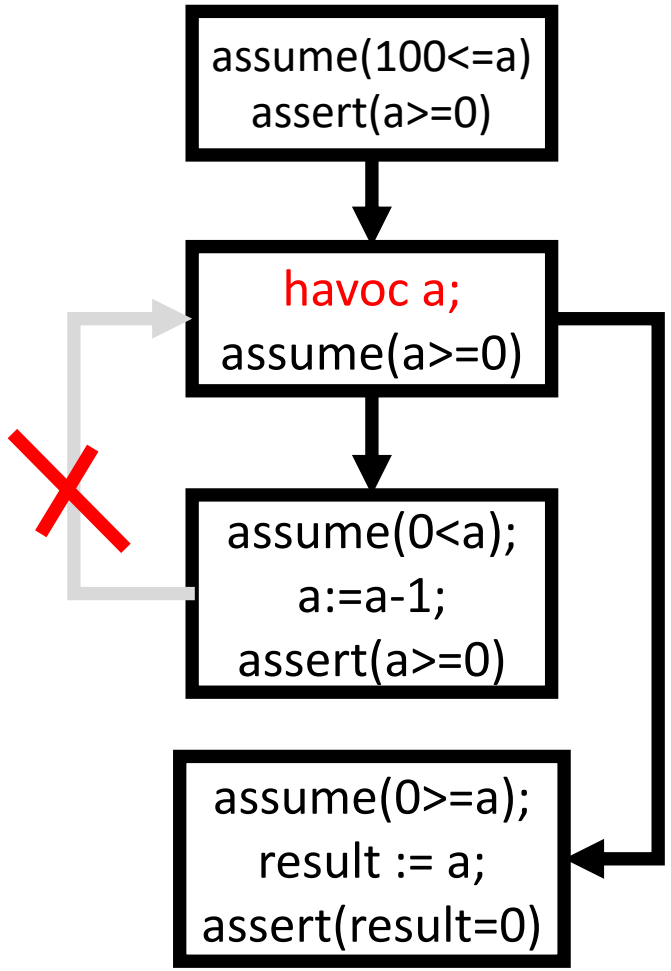
What is the loop target?



Automated Verifier

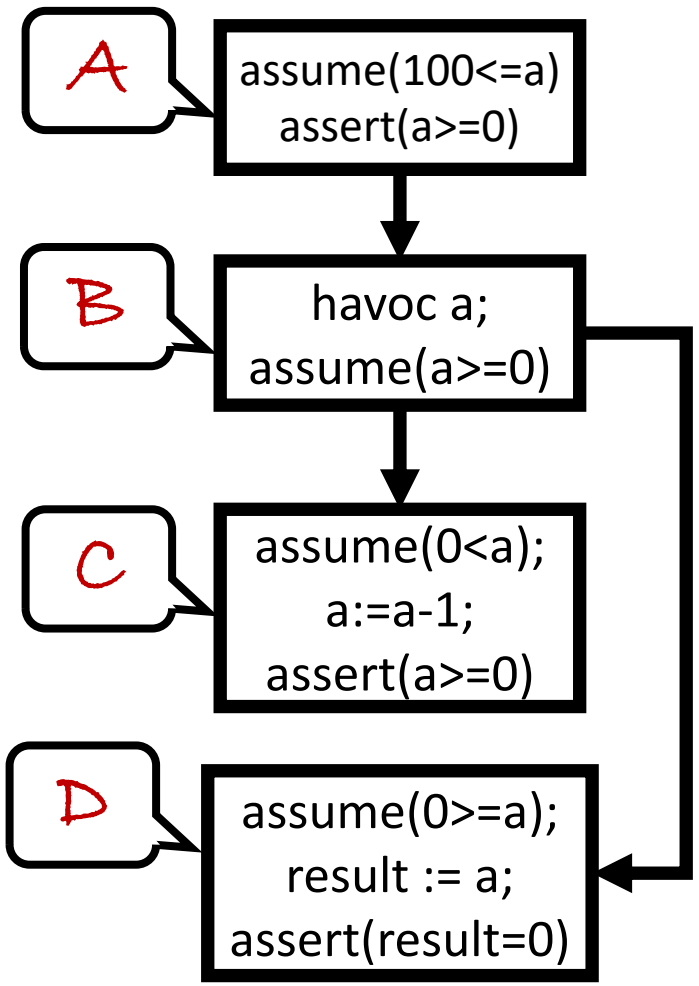
```
{100<=a}
int example(int a)
{
        while(0 < a)
                invariant a>=0
        {
                a := a - 1;
        }
        return a;
}
{result=0}
```





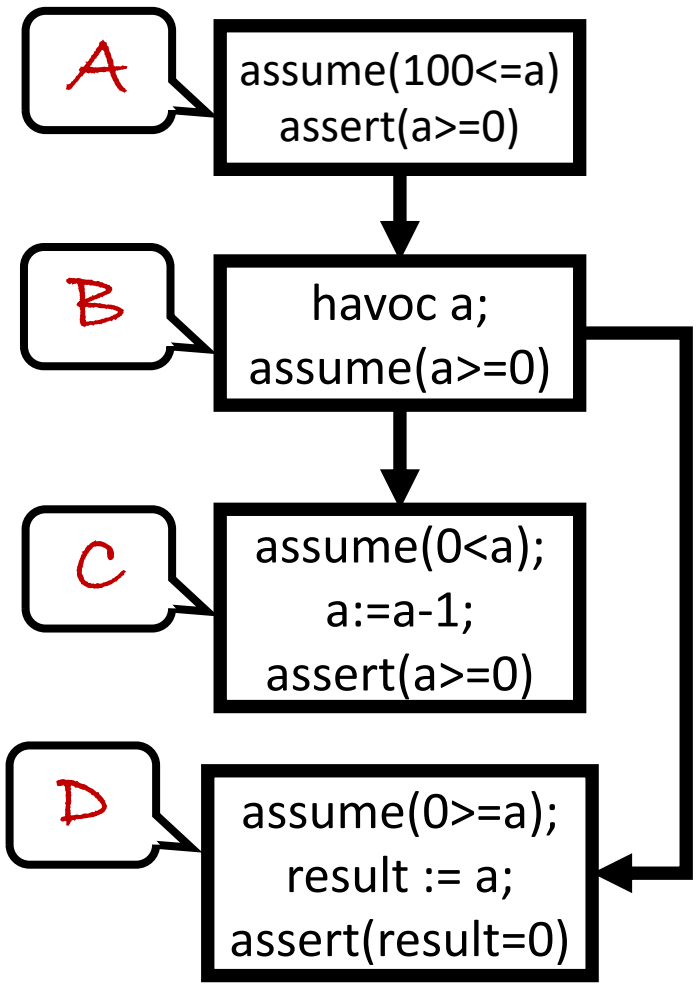


Automated Verifier



$$
\begin{aligned}
D_{ok} &\equiv wlp(assume(0 \geq a); result := a; assert(result = 0), T) \\
&\equiv wlp(assume(0 \geq a); result := a, result = 0) \\
&\equiv wlp(assume(0 \geq a), a = 0) \\
&\equiv (0 \geq a) \rightarrow a = 0
\end{aligned}
$$

Automated Verifier



$$D_{ok} \equiv (0 \geq a) \rightarrow a = 0$$

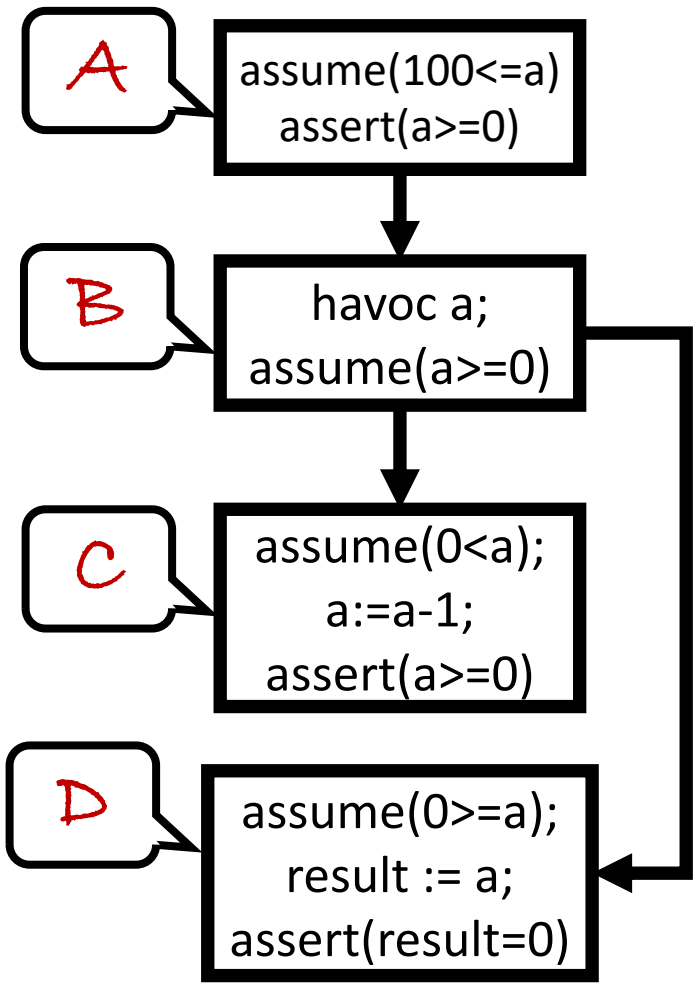
$$C_{ok} \equiv wlp(assume(0 < a); a := a - 1; assert(a \geq 0), T)$$

$$\equiv wlp(assume(0 < a); a := a - 1, a \geq 0)$$

$$\equiv wlp(assume(0 < a), (a - 1) \geq 0)$$

$$\equiv (0 < a) \rightarrow (a - 1) \geq 0$$

Automated Verifier



$$D_{ok} \equiv (0 \geq a) \rightarrow a = 0$$

$$C_{ok} \equiv (0 < a) \rightarrow (a - 1) \geq 0$$

$$B_{ok} \equiv wlp(havoc\ \ a; assume(a \geq 0), C_{ok} \wedge D_{ok})$$

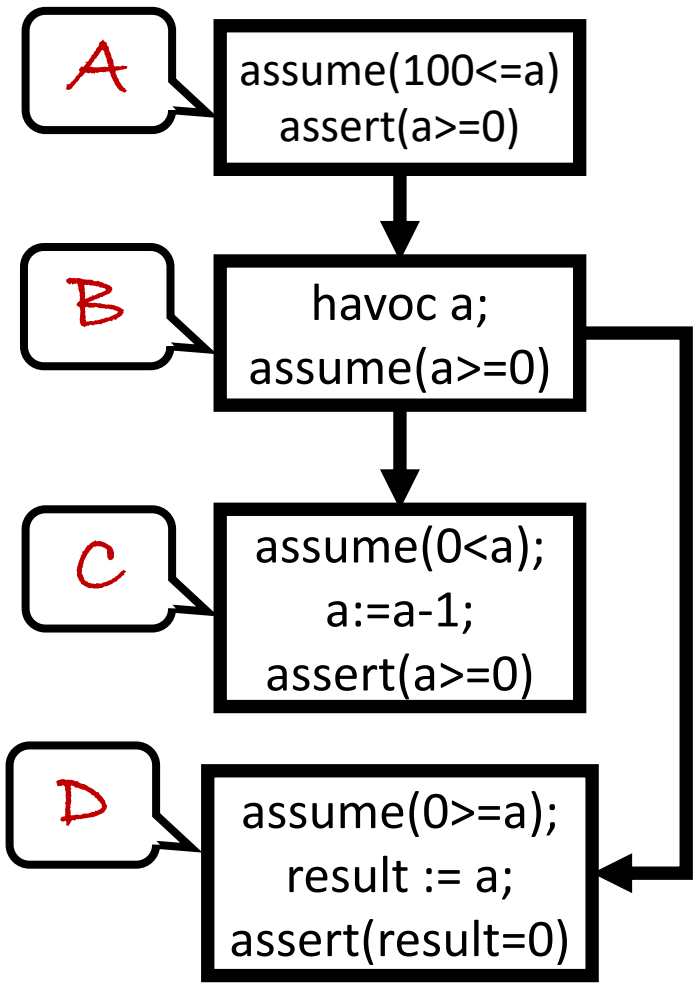
$$\equiv wlp(havoc\ \ a, (a \geq 0) \rightarrow C_{ok} \wedge D_{ok})$$

$$\equiv (\forall v)((a \geq 0) \rightarrow C_{ok} \wedge D_{ok})[v/a]$$

$$\equiv (\forall v)((a \geq 0) \rightarrow ((0 < a) \rightarrow (a - 1) \geq 0) \wedge ((0 \geq a) \rightarrow a = 0))[v/a]$$

$$\equiv (\forall v)((v \geq 0) \rightarrow ((0 < v) \rightarrow (v - 1) \geq 0) \wedge ((0 \geq v) \rightarrow v = 0))$$

Automated Verifier



$$D_{ok} \equiv (0 \geq a) \rightarrow a = 0$$

$$C_{ok} \equiv (0 < a) \rightarrow (a - 1) \geq 0$$

$$B_{ok} \equiv (\forall v)((v \geq 0) \rightarrow ((0 < v) \rightarrow (v - 1) \geq 0) \wedge ((0 \geq v) \rightarrow v = 0))$$

$$A_{ok} \equiv wlp(assume(100 \leq a); assert(a \geq 0), B_{ok})$$

$$\equiv wlp(assume(100 \leq a), (a \geq 0) \wedge B_{ok})$$

$$\equiv (100 \leq a) \rightarrow ((a \geq 0) \wedge B_{ok})$$

$$\equiv (100 \leq a) \rightarrow$$

$$((a \geq 0) \wedge$$

$$((\forall v)((v \geq 0) \rightarrow ((0 < v) \rightarrow (v - 1) \geq 0) \wedge ((0 \geq v) \rightarrow v = 0))))$$

Thanks & Questions