CS6410 Software Verification

#1. Introduction and Overview

Ashish Mishra

with Inputs from Roopsha Samantha's course at Purdue

Instructor

Ashish Mishra

- Asst. Professor at CSE
- Before: Postdoc, Purdue CS, Programming Languages Group.
- Even before: IISc, CSA, PhD
- Research Goal: Help programmers write correct and trustworthy software.
- Areas: Programming Languages, Program Verification, Synthesis.

Logistics

- Lecture:
  - When: Slot R (at least for now)
  - Where: C-LH5 (for now)
- Course Website: Please register on Google Classroom link.
  - TBA
- Office Hours:
  - After the class OR with appointment a day before.

Administrative and Logistics

- Lectures 2* (1.5 hrs) a week (Tuesdays and Fridays).
- Academic honesty during the course is essential -
  - for details
- Class attendance and participation has 5% of the marks.
  - The course only makes sense if it is interactive.

Administrative and Logistics

- Pre-requisites
  - Algorithms.
  - Open to other departments after instructor’s consent.

Major Topics (Theory + Tutorials)

- Preliminaries:
  - SAT and SMT - Propositional Logic: Syntax and Semantics, Resolution-based SAT solving, DPLL, First-Order Logic: Syntax and Semantics, Satisfiability Modulo Theories, First Order Theories
- Program Semantics and Specifications -
  - Operational Semantics, Strongest Post-condition and Weakest Pre-condition, Hoare Logic
- Automated Techniques -
  - Abstract Interpretation, Bounded Model Checking, Predicate Abstraction, CEGAR, Property-Directed Reachability
- Advanced Topics : Concurrent Verification, Program Synthesis

Evaluation

- Assignments 30%
- Final Exam 30%
- Surprise quizzes 10%
- Class participation/attendance: 5%
- Paper Reading 25%

Late Submissions

- Quiz/Exams: No late submission allowed.
- Assignments: 10% deduction for delay of every 24 hours or part.
  - Except for medical reasons,
  - Submission after a week of the designated lab slot will not be evaluated

Resources

- Course Textbook:
  - [BM] The Calculus of Computation: Decision Procedures with Applications to Verification. Aaron R Bradley and Zohar Manna
- This will be supplemented with a variety of sources, including:
  - [NN] Nielson, Flemming, Hanne Riis Nielson, and Chris Hankin. Principles of Program Analysis. 1st ed. 1999.
  - [SAT] Biere, Armin. Handbook of Satisfiability. 2009.
  - [MC] Clarke, Edmund M. et al. Handbook of Model Checking. Ed. Edmund M. Clarke et al. 1st ed. 2018.
  - [CC] Cousot, Patrick. Principles of Abstract Interpretation. 2nd ed. 2021.

Now the good stuff… :)

What is this course about?

Logical foundations & algorithmic techniques to ensure program correctness

Specification	Logics to express expected program behavior
Verification	Methods to automatically check if a program satisfies a specification
Repair	Methods to automatically fix an incorrect program
Synthesis	Methods to automatically generate a correct program

Why should you care?

Programmers make mistakes

Reasoning (About Program Correctness)

Program Verification and Analysis

bugs, bugs everywhere

> A recent study found that 60–70% of vulnerabilities in iOS and macOS are caused by memory unsafety. Microsoft estimates that 70% of all vulnerabilities in their products over the last decade have been caused by memory unsafety. Google estimated that 90% of Android vulnerabilities are memory unsafety. An analysis of 0-days that were discovered being exploited in the wild found that more than 80% of the exploited vulnerabilities were due to memory unsafety.
>
> (Gaynor, 2019)

Why should we care about Correctness?

- When programs have errors, they fail, causing real

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

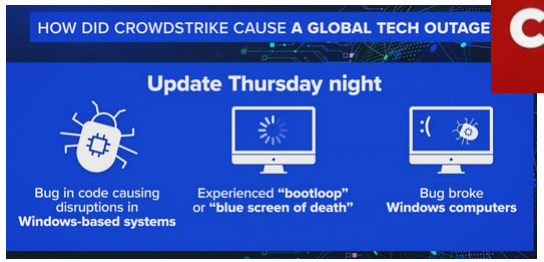
We want our Software to be Correct!

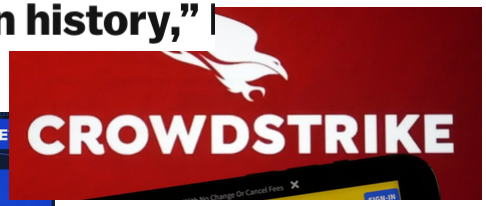
Boeing Fixing New Software Bug on Max; Key Test Flight Nears

- Company says latest flaw shouldn't delay plane's return
- Issue involves aircraft's so-called 'trim' warning system

TECHNOLOGY

The "largest IT outage in history,"





Turing Awards

Dijkstra

Floyd

Hoare

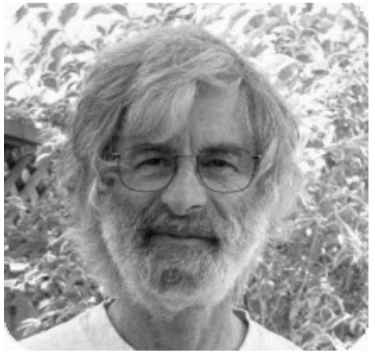
Milner

Pnueli

Clarke

Emerson

Sifakis



Lamport

Success stories

- **Intel** CPU arithmetic and logical operations
- **Microsoft** device drivers
- **Rockwell Collins** AAMP7G microprocessor's partition management
- **Rolls Royce** Trent Series Health Monitoring Units
- **Lockheed Martin** C130J Mission Computers
- **Boeing** "Little Bird" helicopter (seL4 OS-based mission computer)
- **Royal Navy** Ship/Helicopter Operating Limits Unit
- **Airbus 380** primary flight control software
- **Paris Metro** (RATP)
- **NASA** Mars Rover data management subsystem
- **Bombardier ILLBV950L2** railway interlocking system
- **Apple, ARM/SoftBank, Nvidia, IBM, Oracle** RTL
- **AMD** K5 floating point square root microcode
- **Micrium** OS µC/OS-II real-time kernel

SYNOPSYS®

TOYOTA

galois

powered by amazon web services

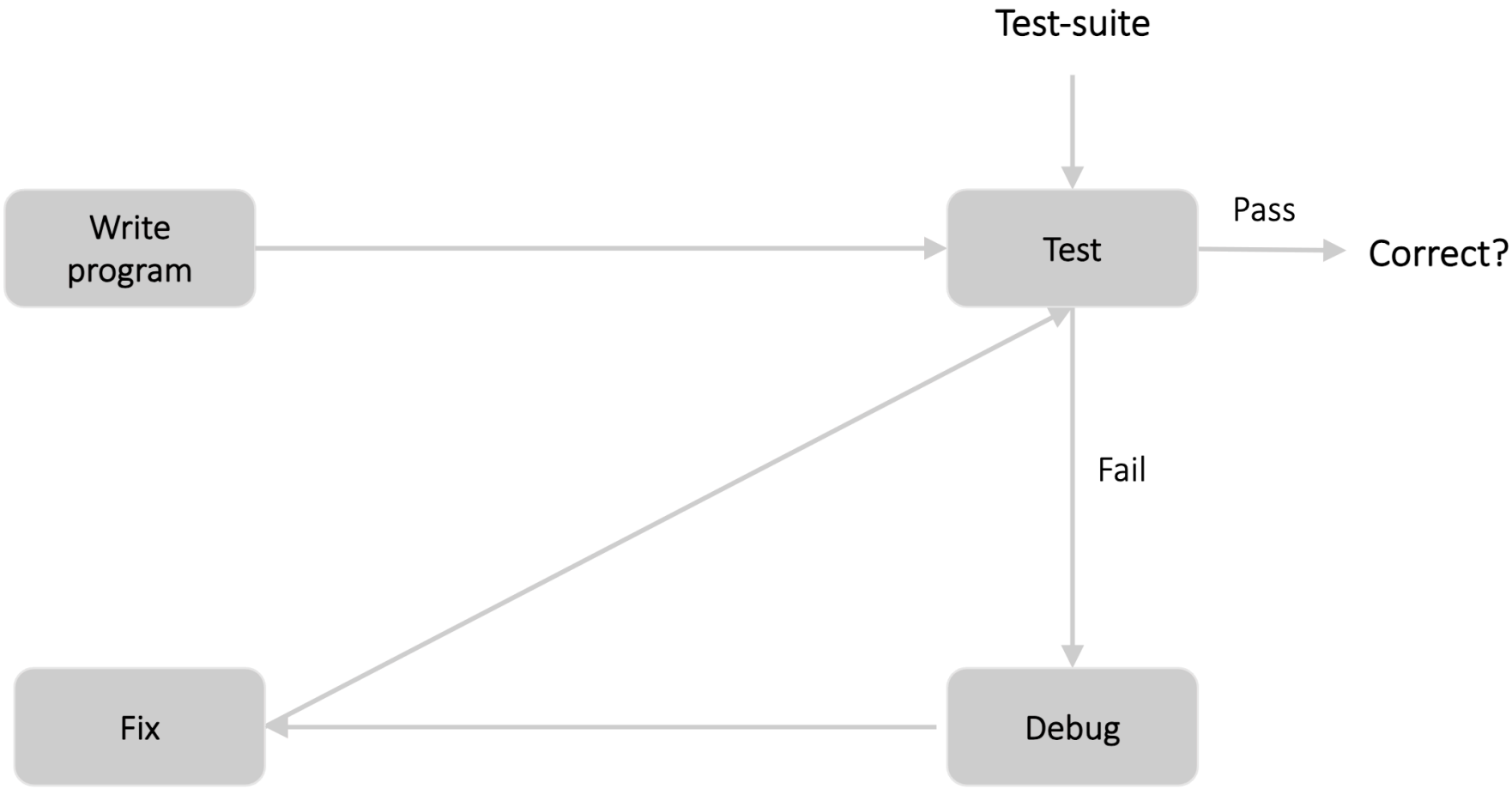
Astrée

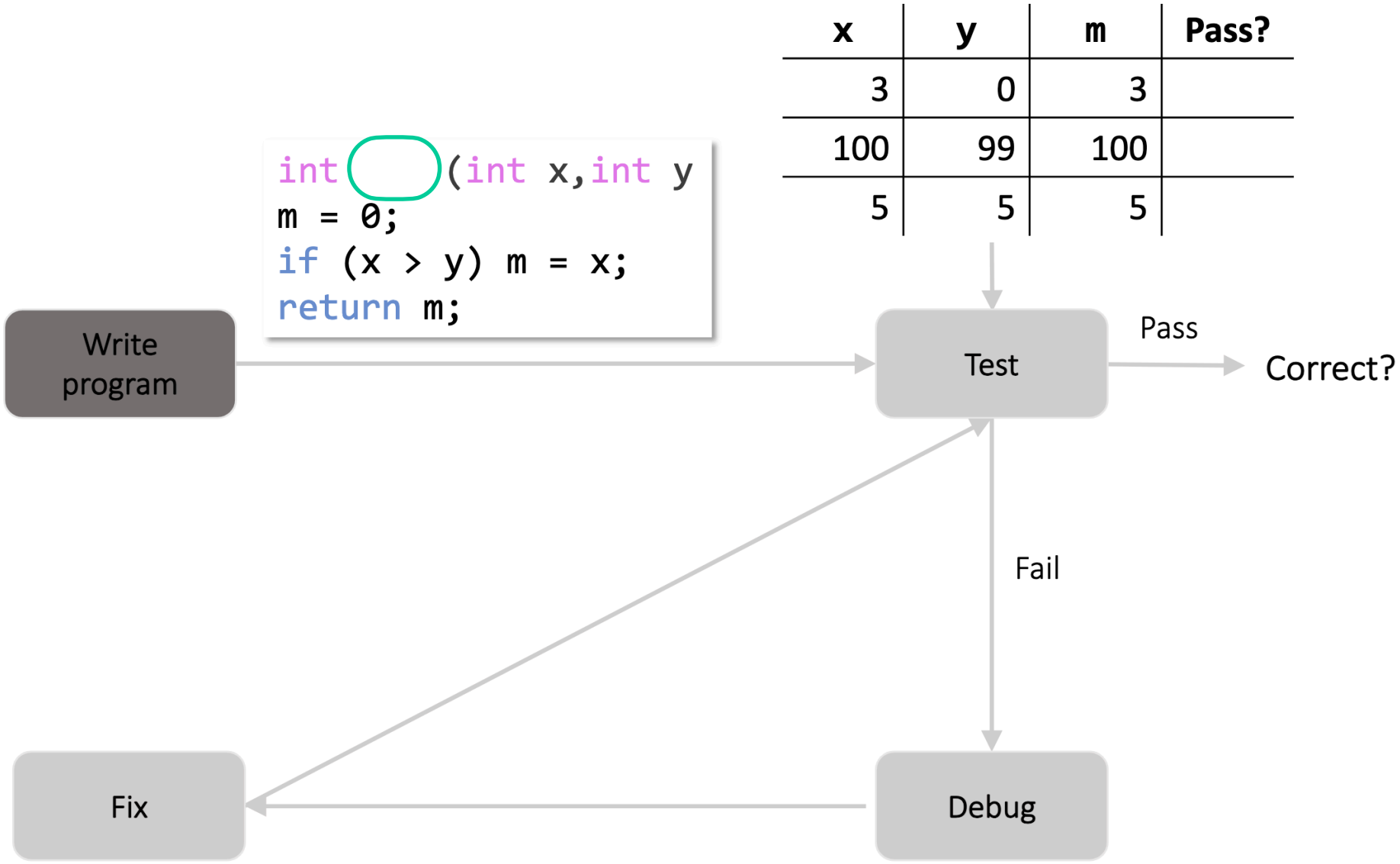
Overview

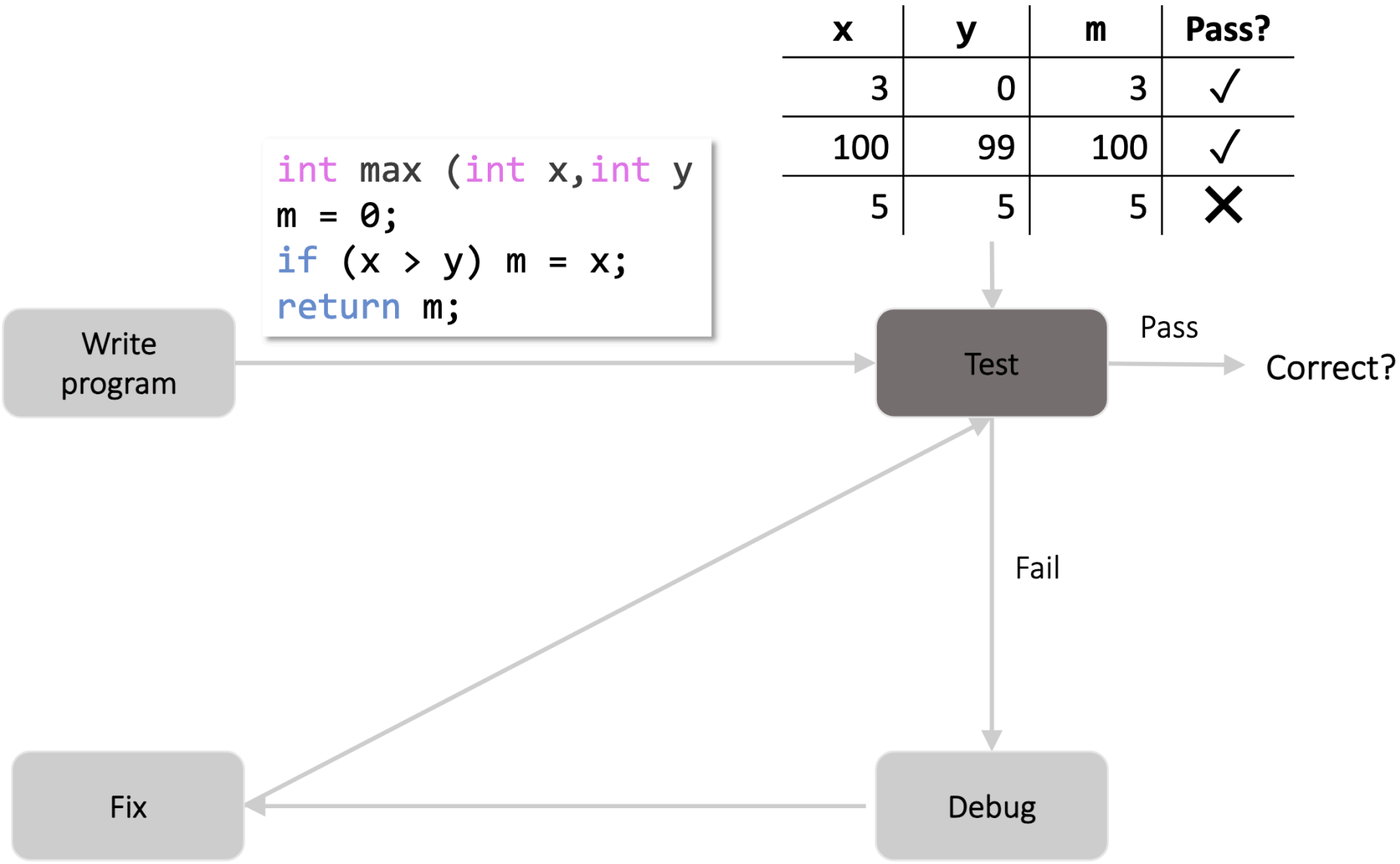
Dijkstra

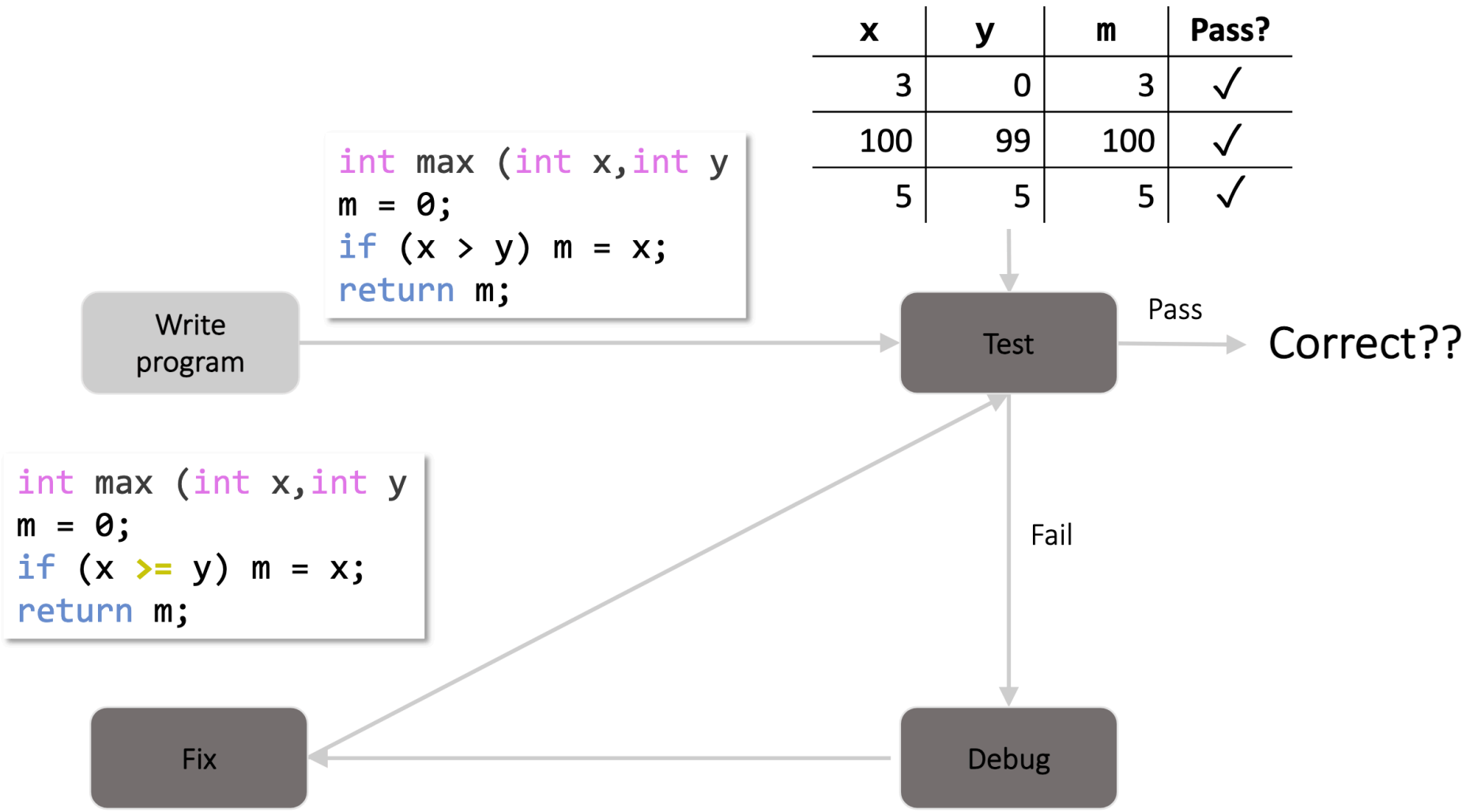
Testing shows the presence,
not the absence of bugs.

Ergo, testing can fail to show the presence of some bugs!


Test-suite
Write program
Test
Pass
Correct?
Fail
Fix
Debug


int (int x,int y
m = 0;
if (x > y) m = x;
return m;
x
y
m
Pass?
3
0
3
100
99
100
5
5
5
Write program
Test
Pass
Correct?
Fail
Fix
Debug


x
y
m
Pass?
3
0
3
✓
100
99
100
✓
5
5
5
✗
int max (int x,int y
m = 0;
if (x > y) m = x;
return m;
Write program
Test
Pass
Correct?
Fail
Fix
Debug


x | y | m | Pass?
3 | 0 | 3 | ✓
100 | 99 | 100 | ✓
5 | 5 | 5 | ✓
int max (int x,int y
m = 0;
if (x > y) m = x;
return m;
Write program
Test
Pass
Correct??
int max (int x,int y
m = 0;
if (x >= y) m = x;
return m;
Fail
Fix
Debug

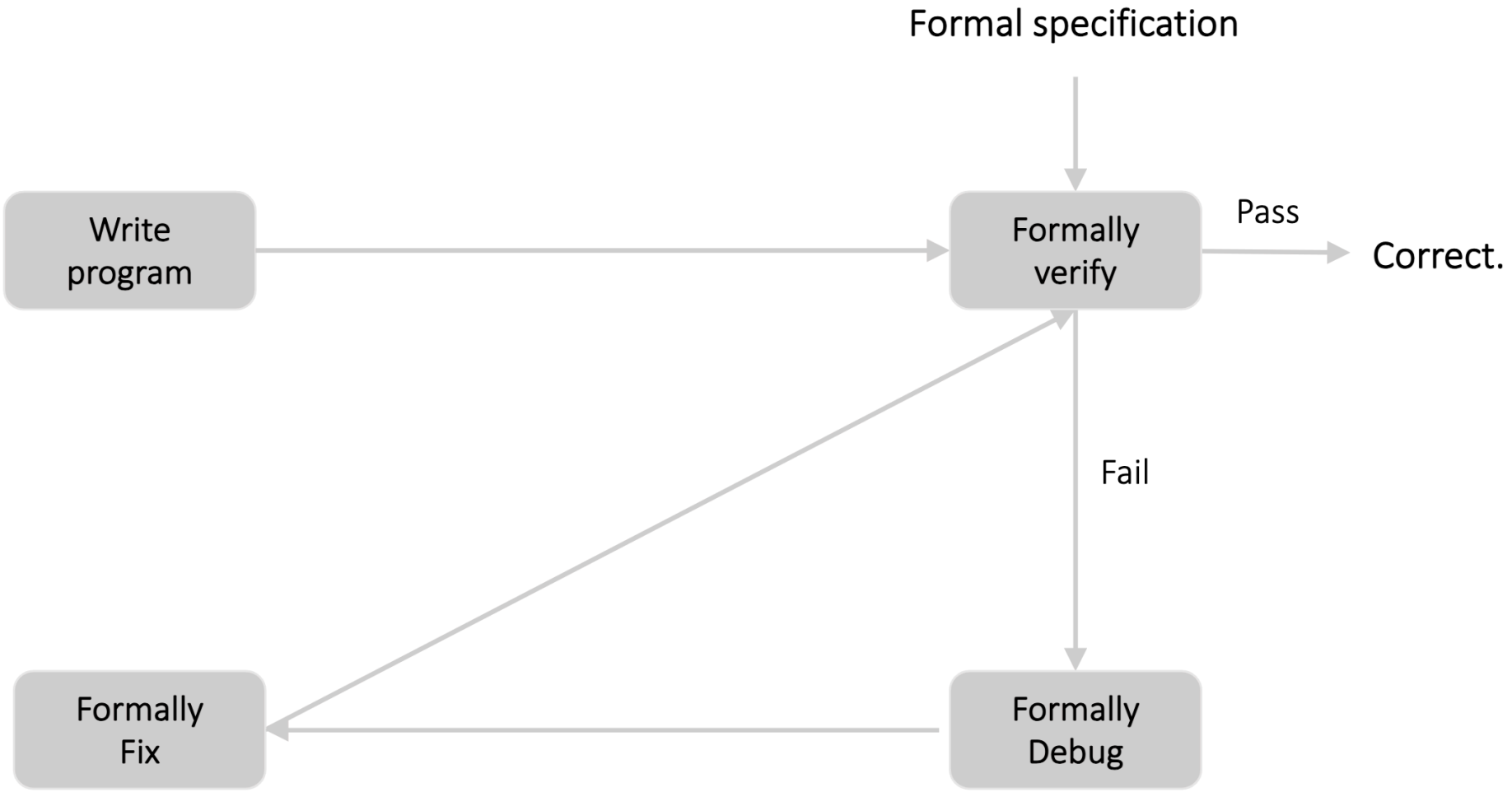
Dijkstra

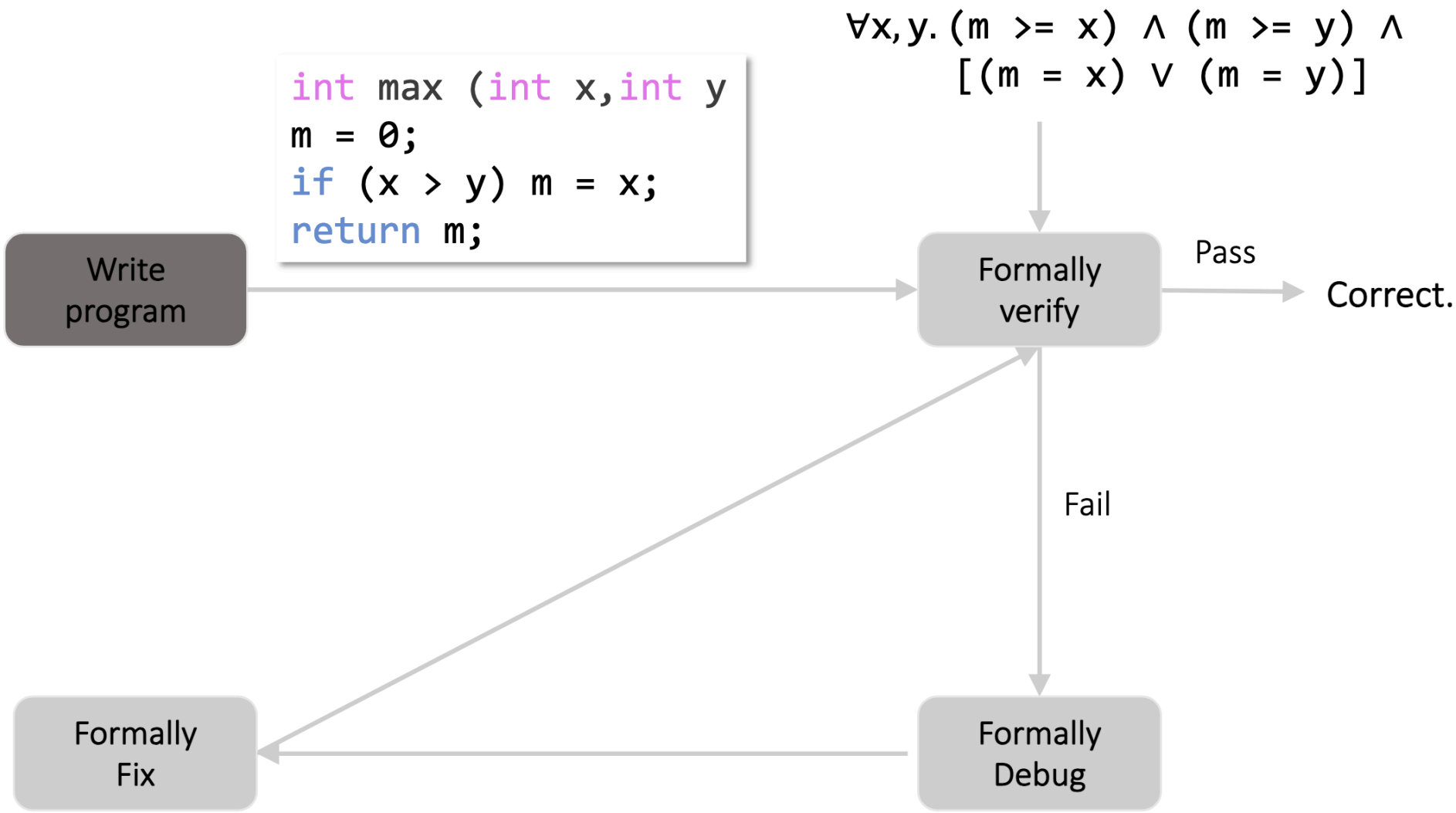
Testing shows the presence,
not the absence of bugs.

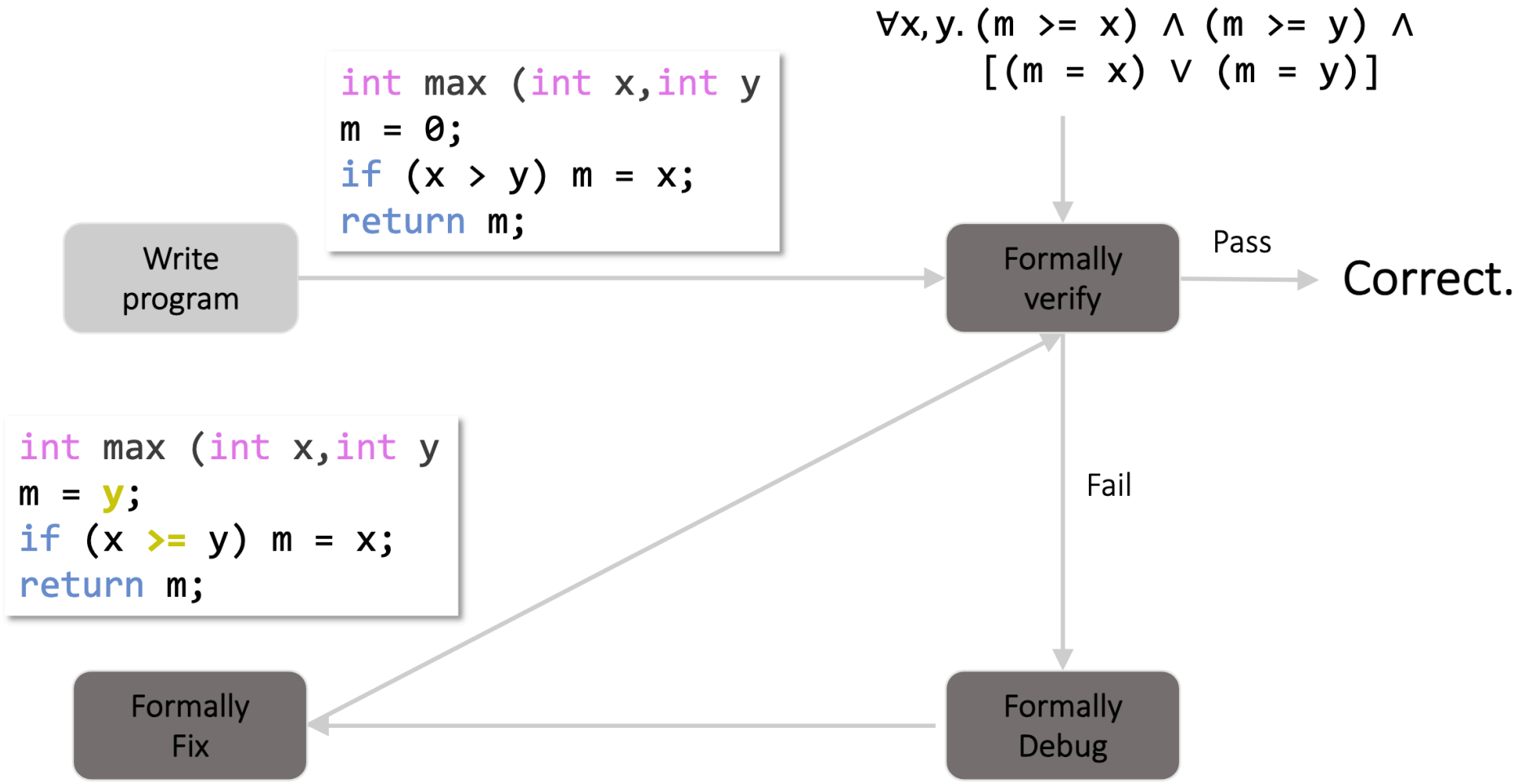
Formal specifications can precisely capture correctness requirements.

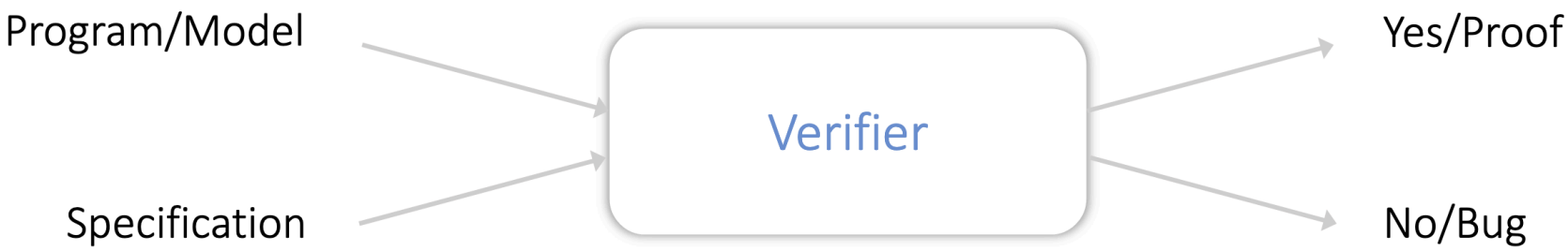
Formal verification can **prove** the absence of bugs!

Formal repair can ensure the absence of bugs for programs with bugs!


Formal specification
Write program
Formally verify
Pass
Correct.
Fail
Formally Fix
Formally Debug


∀x, y. (m >= x) ∧ (m >= y) ∧
[(m = x) ∨ (m = y)]
int max (int x,int y
m = 0;
if (x > y) m = x;
return m;
Write program
Formally verify
Pass
Correct.
Fail
Formally Fix
Formally Debug


∀x, y. (m >= x) ∧ (m >= y) ∧
[(m = x) ∨ (m = y)]
int max (int x,int y
m = 0;
if (x > y) m = x;
return m;
Write program
Formally verify
Pass
Correct.
int max (int x,int y
m = y;
if (x >= y) m = x;
return m;
Fail
Formally Fix
Formally Debug



Type Systems

Deductive Verification

Model Checking

Abstract Interpretation

Type Systems

Deductive Verification

Model Checking

Abstract Interpretation

Static analysis

Interactive theorem provers

Automatic theorem provers

SAT/SMT solvers

Expressiveness
Automation
Scalability
Precision
Applicability

Propositional Logic Normal Forms

Calculus of Computation?

> *It is reasonable to hope that the relationship between* **computation** *and* **mathematical logic** *will be as fruitful in the next century as that between* **analysis** *and* **physics** *in the last. The development of this relationship demands a concern for both applications and mathematical elegance.*

John McCarthy
A Basis for a Mathematical Theory of Computation, 1963

Propositional logic (PL) syntax

Atom	truth symbols	$\top$ ("true") and $\bot$ ("false")
	propositional variables	p, q, r, p_1, q_1
Literal	atom α or its negation $\neg\alpha$	
Formula	literal or application of a logical connective to F, F_1, F_2	
	$\neg F$	"not" (negation)
	$F_1 \vee F_2$	"or" (disjunction)
	$F_1 \wedge F_2$	"and" (conjunction)
	$F_1 \rightarrow F_2$	"implies" (implication)
	$F_1 \leftrightarrow F_2$	"if and only if" (iff)

Example

formula $F : (P \wedge Q) \rightarrow (\top \vee \neg Q)$
atoms: P, Q, $\top$
literals: P, Q, $\top$, $\neg Q$
subformulae: P, Q, $\top$, $\neg Q$, $P \wedge Q$, $\top \vee \neg Q$, F
abbreviation

$$F : P \wedge Q \rightarrow \top \vee \neg Q$$

PL Semantics (Meaning)

Sentence F + Interpretation I = Truth value (true, false)

Interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \cdots\}$$

Evaluation of F under I:

F	$\neg F$
0	1
1	0

where 0 corresponds to value false
1 true

$I \models F$ if F evaluates to true under I
$I \not\models F$ false

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Satisfying and Falsifying Interpretations

Example

$F : P \land Q \rightarrow P \lor \neg Q$
$I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}$

P	Q	$\neg Q$	$P \land Q$	$P \lor \neg Q$	F
1	0	1	0	1	1

$1 = \text{true}$ $0 = \text{false}$

F evaluates to true under I

PL Semantics (Inductive definitions)

Base Case:

$$I \models \top$$

$$I \not\models \bot$$

$$I \models P \quad \text{iff} \quad I[P] = \text{true}$$

$$I \not\models P \quad \text{iff} \quad I[P] = \text{false}$$

Inductive Case:

$$I \models \neg F \quad \text{iff } I \not\models F$$

$$I \models F_1 \wedge F_2 \quad \text{iff } I \models F_1 \text{ and } I \models F_2$$

$$I \models F_1 \vee F_2 \quad \text{iff } I \models F_1 \text{ or } I \models F_2$$

$$I \models F_1 \rightarrow F_2 \quad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$

$$I \models F_1 \leftrightarrow F_2 \quad \text{iff, } I \models F_1 \text{ and } I \models F_2, \text{ or } I \not\models F_1 \text{ and } I \not\models F_2$$

Note:

$$I \not\models F_1 \rightarrow F_2 \quad \text{iff} \quad I \models F_1 \text{ and } I \not\models F_2$$

Example

$$F: \; P \wedge Q \rightarrow P \vee \neg Q$$

$$I: \; \{P \mapsto \text{true}, \; Q \mapsto \text{false}\}$$

1.	I	$\models$	P	since $I[P] = \text{true}$	
2.	I	$\not\models$	Q	since $I[Q] = \text{false}$	
3.	I	$\models$	$\neg Q$	by 2 and $\neg$	
4.	I	$\not\models$	$P \wedge Q$	by 2 and $\wedge$	
5.	I	$\models$	$P \vee \neg Q$	by 1 and $\vee$	
6.	I	$\models$	F	by 4 and $\rightarrow$	Why?

Thus, F is true under I.

Satisfiability and Validity

F is satisfiable iff there exists $I : I \vDash F$

F is valid iff for all $I : I \vDash F$

Duality:
F is valid iff $\neg F$ is unsatisfiable

Procedure for deciding satisfiability *or* validity suffices!

Deciding satisfiability/validity

- Basic techniques
  - Truth table method: search-based
  - Semantic argument method: deductive technique
- SAT solvers
  - Combine search and deduction

Truth table method

1. Enumerate all interpretations
2. Search for satisfying interpretation

Brute-force!
Impractical (2^n interpretations)
Can't be used if domain is not finite, e.g., for first-order logic

$$F: P \wedge Q \rightarrow P \vee \neg Q$$

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
⊥	⊥	⊥	⊤	⊤	⊤
⊥	⊤	⊥	⊥	⊥	⊤
⊤	⊥	⊥	⊤	⊤	⊤
⊤	⊤	⊤	⊥	⊤	⊤

Thus F is valid.

Example

$$F : P \vee Q \rightarrow P \wedge Q$$

P	Q	$P \vee Q$	$P \wedge Q$	F	
0	0	0	0	1	$\leftarrow$ satisfying I
0	1	1	0	0	$\leftarrow$ falsifying I
1	0	1	0	0	
1	1	1	1	1	

Thus F is satisfiable, but invalid.

Method 2: Semantic Argument

Proof by contradiction:

1. Assume F is not valid
2. Apply proof rules
3. Contradiction (i.e, $\bot$) along every branch of proof tree $\Rightarrow F$ is valid
4. Otherwise, F is not valid

A bit of an overhead for PL
Applicable to first-order logic

Proof rules

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \wedge G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \leftarrow \text{and} \qquad \frac{I \not\models F \wedge G}{I \not\models F \mid I \not\models G} \nwarrow \text{or}$$

$$\frac{I \models F \vee G}{I \models F \mid I \models G} \qquad \frac{I \not\models F \vee G}{\begin{array}{l} I \not\models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \rightarrow G}{I \not\models F \mid I \models G} \qquad \frac{I \not\models F \rightarrow G}{\begin{array}{l} I \models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \mid I \not\models F \vee G} \qquad \frac{\begin{array}{l} I \models F \\ I \not\models F \end{array}}{I \models \bot}$$

$$\frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \mid I \models \neg F \wedge G}$$

Example

To Prove $F: P \wedge Q \rightarrow P \vee \neg Q$ is valid.

Let's assume that F is not valid and that I is a falsifying interpretation.

1.	I	$\not\models$	$P \wedge Q \rightarrow P \vee \neg Q$	assumption
2.	I	$\models$	$P \wedge Q$	1 and $\rightarrow$
3.	I	$\not\models$	$P \vee \neg Q$	1 and $\rightarrow$
4.	I	$\models$	P	2 and $\wedge$
5.	I	$\not\models$	P	3 and $\vee$
6.	I	$\models$	$\perp$	4 and 5 are contradictory

Thus F is valid

#2 Overview, Normal Forms and SAT

Overview: Module I : Fundamentals

- Propositional Logic: Theory, Implementation and Applications
- First-Order Logic
- First-Order Theories
- Satisfiability Modulo Theories
- Program Semantics and Specifications
- Induction
- Dafny Verifier (https://dafny.org/)

Overview: Module II : Model Checking

- Transition Systems
- Model Checking
- Model Checking Theory
- Model Checking Applications:
  - CBMC (https://www.cprover.org/cbmc)

Overview: Module II : Abstract Interpretation

- Static and Dynamic Program Analysis
- Abstract Interpretation Theory
- Abstract Interpretation Application
  - Static DataFlow Analysis Frameworks:
    - Soot for Java (http://soot-oss.github.io/soot)
    - NASA IKOS for C/C++ (https://github.com/NASA-SW-VnV/ikos)
  - Dynamic Analysis Framework (https://github.com/analysis-tools-dev)
    - Jalangi : JavaScript
    - KLEE, Valgrind C/C++

Example : Recap PL formula

formula $F : (P \wedge Q) \rightarrow (\top \vee \neg Q)$
atoms: P, Q, $\top$
literals: P, Q, $\top$, $\neg Q$
subformulae: P, Q, $\top$, $\neg Q$, $P \wedge Q$, $\top \vee \neg Q$, F
abbreviation

$$F : P \wedge Q \rightarrow \top \vee \neg Q$$

PL Semantics (Meaning)

Sentence F + Interpretation I = Truth value (true, false)

Interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \cdots\}$$

Evaluation of F under I:

F	$\neg F$
0	1
1	0

where 0 corresponds to value false
1 true

$I \models F$ if F evaluates to true under I
$I \not\models F$ false

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Satisfying and Falsifying Interpretations

PL Semantics (Inductive definitions)

Base Case:

$$I \models \top$$

$$I \not\models \bot$$

$$I \models P \quad \text{iff} \quad I[P] = \text{true}$$

$$I \not\models P \quad \text{iff} \quad I[P] = \text{false}$$

Inductive Case:

$$I \models \neg F \quad \text{iff } I \not\models F$$

$$I \models F_1 \wedge F_2 \quad \text{iff } I \models F_1 \text{ and } I \models F_2$$

$$I \models F_1 \vee F_2 \quad \text{iff } I \models F_1 \text{ or } I \models F_2$$

$$I \models F_1 \rightarrow F_2 \quad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$

$$I \models F_1 \leftrightarrow F_2 \quad \text{iff, } I \models F_1 \text{ and } I \models F_2, \text{ or } I \not\models F_1 \text{ and } I \not\models F_2$$

Note:

$$I \not\models F_1 \rightarrow F_2 \quad \text{iff} \quad I \models F_1 \text{ and } I \not\models F_2$$

Satisfiability and Validity

F is satisfiable iff there exists $I : I \vDash F$

F is valid iff for all $I : I \vDash F$

Duality:
F is valid iff $\neg F$ is unsatisfiable

Procedure for deciding satisfiability *or* validity suffices!

Example 2

To Prove

$F: (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$ is valid.

Example 2

Let's assume that F is not valid.

1.	I	$\not\models$	F	assumption
2.	I	$\models$	$(P \rightarrow Q) \wedge (Q \rightarrow R)$	1 and $\rightarrow$
3.	I	$\not\models$	$P \rightarrow R$	1 and $\rightarrow$
4.	I	$\models$	P	3 and $\rightarrow$
5.	I	$\not\models$	R	3 and $\rightarrow$
6.	I	$\models$	$P \rightarrow Q$	2 and of $\wedge$
7.	I	$\models$	$Q \rightarrow R$	2 and of $\wedge$

Example 2

Two cases from 6

$$\begin{array}{llll} 8a. & I & \not\models & P \quad \text{6 and } \rightarrow \\ 9a. & I & \models & \bot \quad \text{4 and 8a are contradictory} \end{array}$$

and

$$8b. \quad I \quad \models \quad Q \qquad \text{6 and } \rightarrow$$

Two cases from 7

$$\begin{array}{llll} 9ba. & I & \not\models & Q \quad \text{7 and } \rightarrow \\ 10ba. & I & \models & \bot \quad \text{8b and 9ba are contradictory} \end{array}$$

and

$$\begin{array}{llll} 9bb. & I & \models & R \quad \text{7 and } \rightarrow \\ 10bb. & I & \models & \bot \quad \text{5 and 9bb are contradictory} \end{array}$$

Our assumption is incorrect in all cases — F is valid.

Semantic judgements, Equivalence

F_1 and F_2 are equivalent ($F_1 \Leftrightarrow F_2$)

iff for all interpretations I, $I \models F_1 \leftrightarrow F_2$

To prove $F_1 \Leftrightarrow F_2$ show $F_1 \leftrightarrow F_2$ is valid.

F_1 implies F_2 ($F_1 \Rightarrow F_2$)

iff for all interpretations I, $I \models F_1 \rightarrow F_2$

$F_1 \Leftrightarrow F_2$ and $F_1 \Rightarrow F_2$ are not formulae!