SWE3002-42: Introduction to Software Engineering

Lecture 8 – Software Testing (1)

Sooyoung Cha

Department of Computer Science and Engineering

Today's Lecture

1. Why do we need software testing?

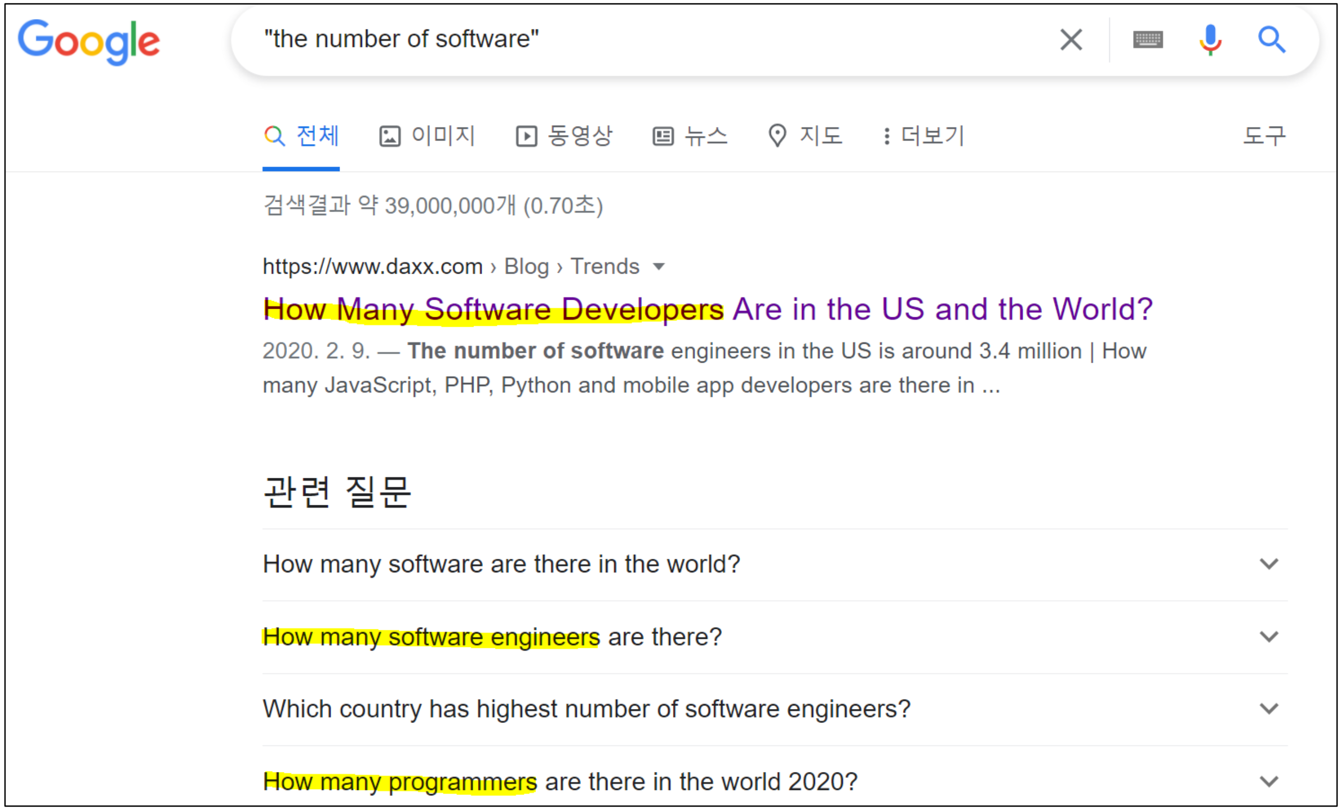
2. What is white-box testing?

Software is Everywhere

- The number of software in the world is ???
  - (1) 10 Million, (2) 100 Million, (3) 1 Billion, (4) 10 Billion

Software is Everywhere

- The number of software in the world is ???
  - (1) 10 Million, (2) 100 Million, (3) 1 Billion, (4) 10 Billion
  - Sorry, I don't know …,



"the number of software"

검색결과 약 39,000,000개 (0.70초)

https://www.daxx.com › Blog › Trends

How Many Software Developers Are in the US and the World?

2020. 2. 9. — **The number of software** engineers in the US is around 3.4 million | How many JavaScript, PHP, Python and mobile app developers are there in ...

관련 질문

How many software are there in the world?

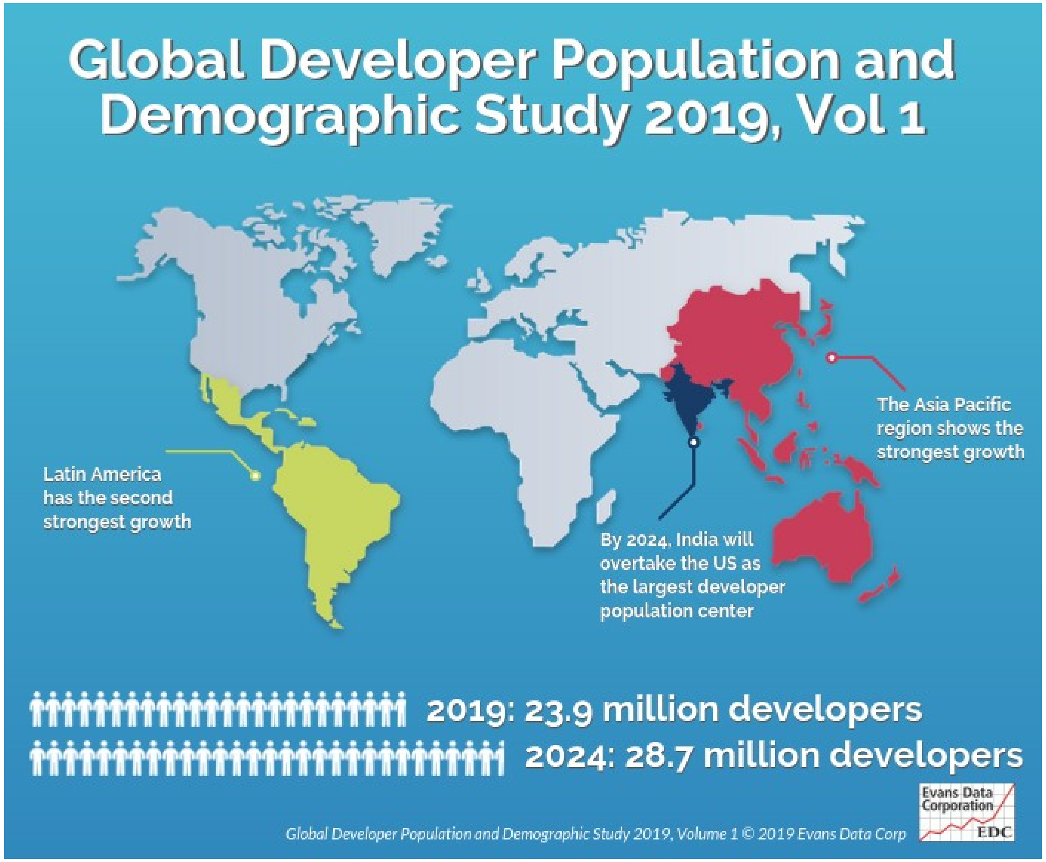
How many software engineers are there?

Which country has highest number of software engineers?

How many programmers are there in the world 2020?

Software is Everywhere

- The number of software developers is increasing.
  - 24 Million (2019) →28 Million (2024) → 40 Million (2030)



Global Developer Population and Demographic Study 2019, Vol 1

Latin America has the second strongest growth

By 2024, India will overtake the US as the largest developer population center

The Asia Pacific region shows the strongest growth

2019: 23.9 million developers

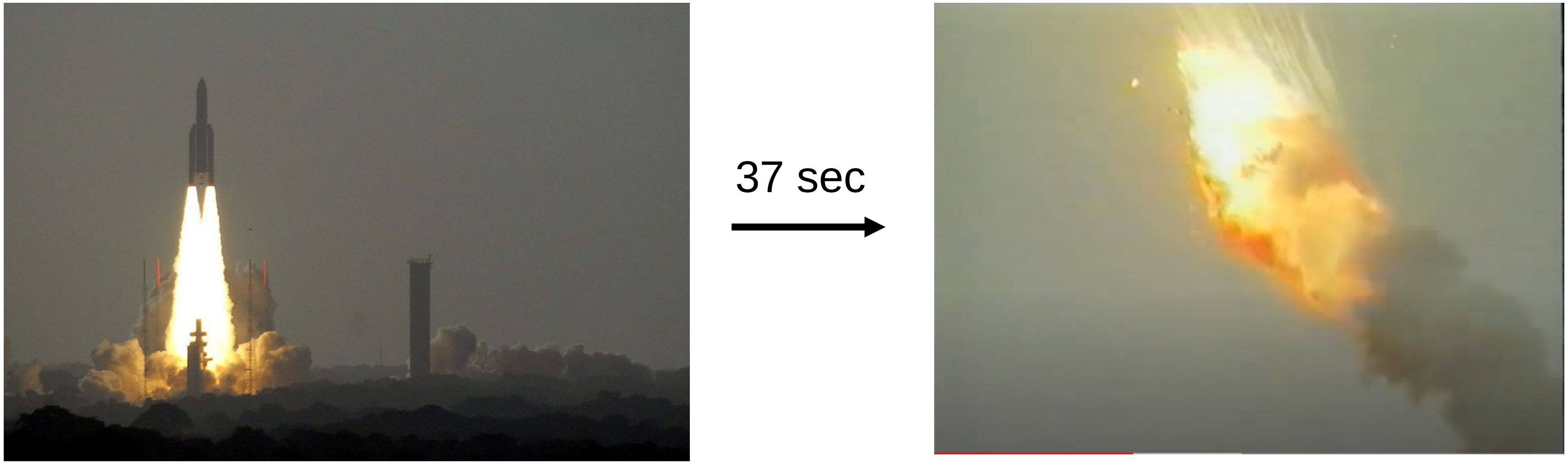
2024: 28.7 million developers

Global Developer Population and Demographic Study 2019, Volume 1 © 2019 Evans Data Corp

https://www.daxx.com/blog/development-trends/number-software-developers-world

Software Bugs are Everywhere

- Ariane 5: The worst software bugs in history
  - Cost: 10 years & 7 billion dollars
  - A data conversion from 64-bit floating point to 16-bit integer value.



[youtube link]

https://www.nytimes.com/1996/12/01/magazine/little-bug-big-bang.html

Software Bugs are Everywhere


연합뉴스 최신기사 정치 북한 경제 산업 사회 전국 세계 문화 라이프 연예 스포츠

#코로나19 #확진자 #김광현 #중국 #가상화폐

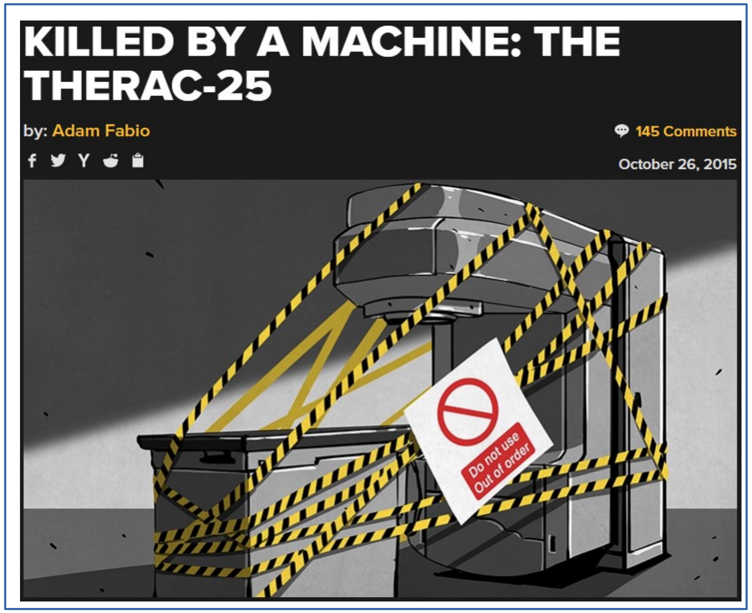
뉴스홈 | 최신기사

나로호 압력측정 소프트웨어 오류가 원인(종합2보)


전자신문

[단독]"도요타 급발진은 SW 결함이 원인" 도요타 조사 보고서 파장

발행일 : 2014.03.20 18:12


KILLED BY A MACHINE: THE THERAC-25

by: Adam Fabio

145 Comments

October 26, 2015


News › Ethereum › Technology

BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits

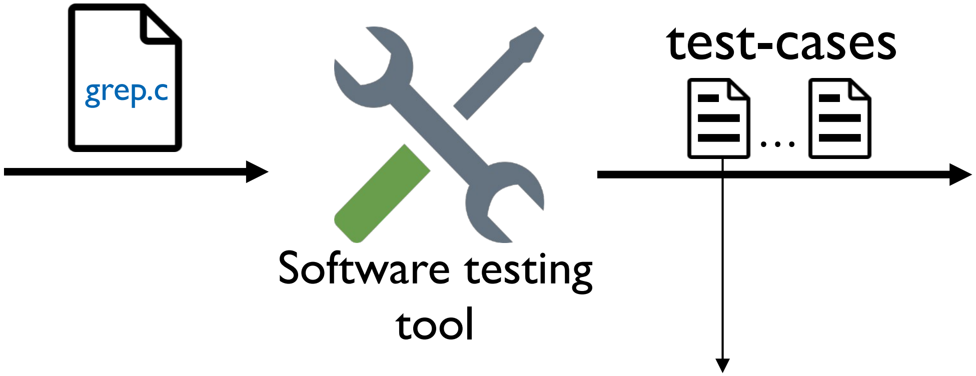
SAM TOWN · APRIL 25, 2018 AT 10:38 PM UTC · 3 MIN READ

Today's Lecture

1. Why do we need software testing?
    - Not to lose our money, time, and life
    - To find software bugs early

Software Testing

- Automatic input generation technique for finding bugs.

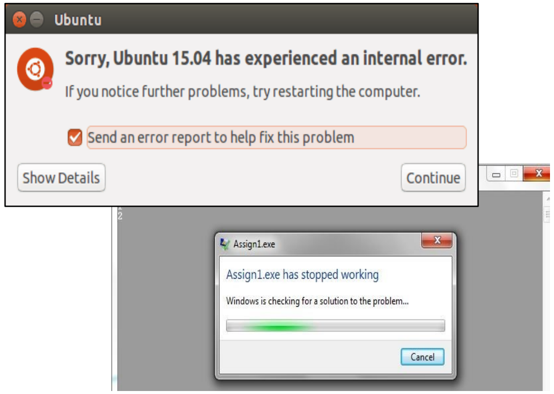


grep.c

Software testing tool

test-cases

…



Ubuntu

Sorry, Ubuntu 15.04 has experienced an internal error.

If you notice further problems, try restarting the computer.

☑ Send an error report to help fix this problem

Show Details

Continue

Assign1.exe

Assign1.exe has stopped working

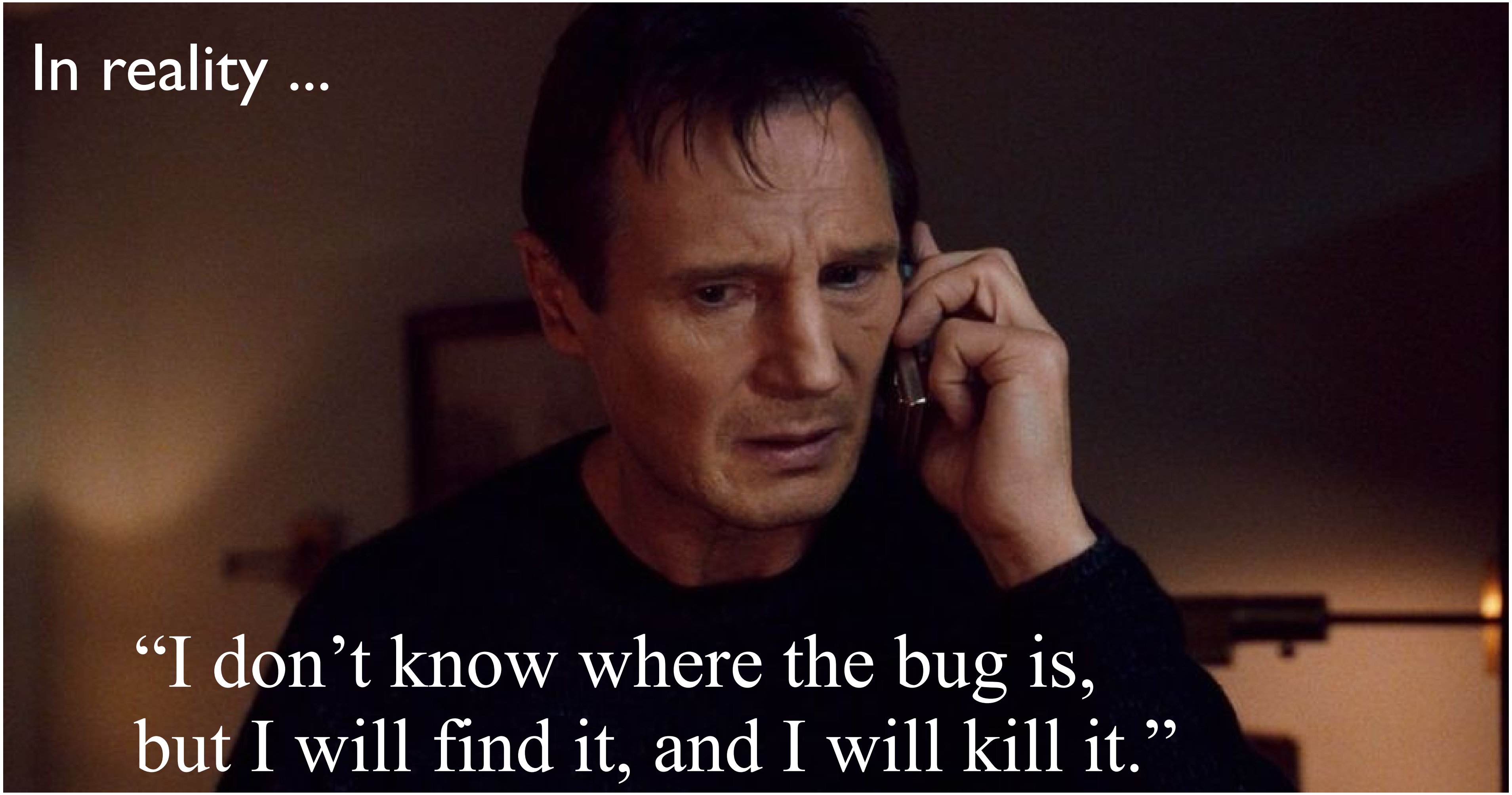
Windows is checking for a solution to the problem...

Cancel

program bugs ↑

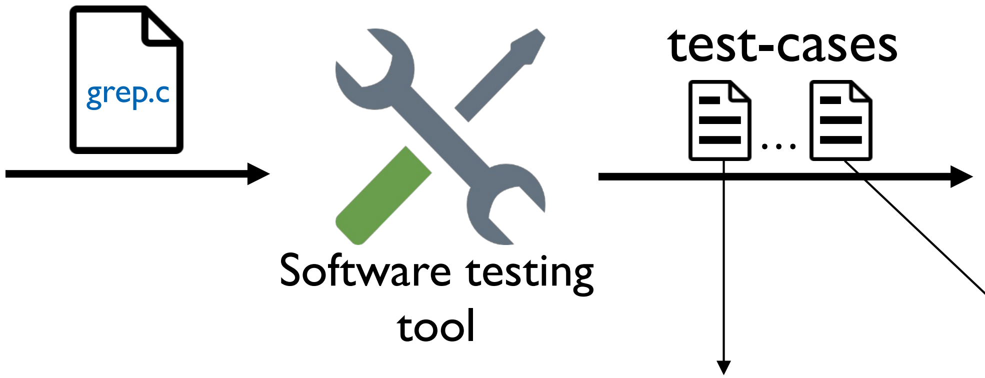
Software Testing

- Automatic input generation technique for finding bugs.



Software Testing

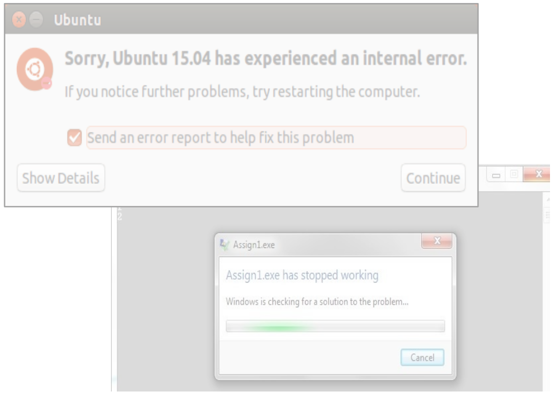
- Automatic input generation technique for increasing code coverage.



grep.c

Software testing tool

test-cases



Ubuntu

Sorry, Ubuntu 15.04 has experienced an internal error.

If you notice further problems, try restarting the computer.

☑ Send an error report to help fix this problem

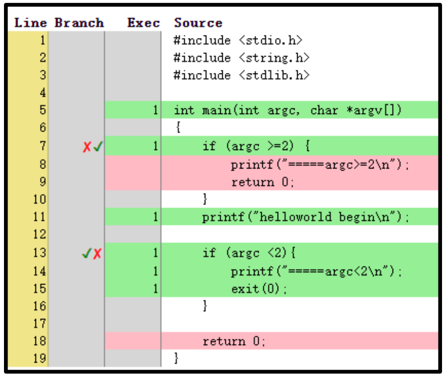
Assign1.exe

Assign1.exe has stopped working

Windows is checking for a solution to the problem...

program bugs ↑

(ultimate goal)



```
Line Branch   Exec  Source
   1                #include <stdio.h>
   2                #include <string.h>
   3                #include <stdlib.h>
   4
   5             1  int main(int argc, char *argv[])
   6                {
   7      ✗✓     1      if (argc >=2) {
   8                        printf("=====argc>=2\n");
   9                        return 0;
  10                    }
  11             1      printf("helloworld begin\n");
  12
  13      ✓✗     1      if (argc <2){
  14             1          printf("=====argc<2\n");
  15             1          exit(0);
  16                    }
  17
  18                    return 0;
  19                }
```

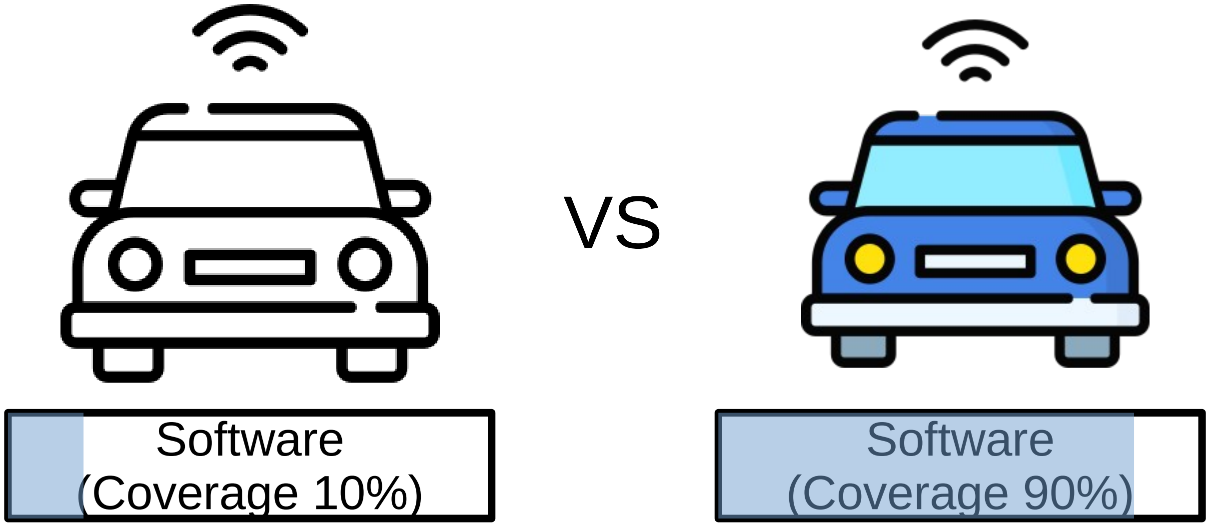
🟩 : executed

🟥 : not executed

code coverage ↑

Software Testing

- Why do we increase code coverage?
  - Hypothesis: "The more code we execute in a program, the more bugs we can find."



Code Coverage

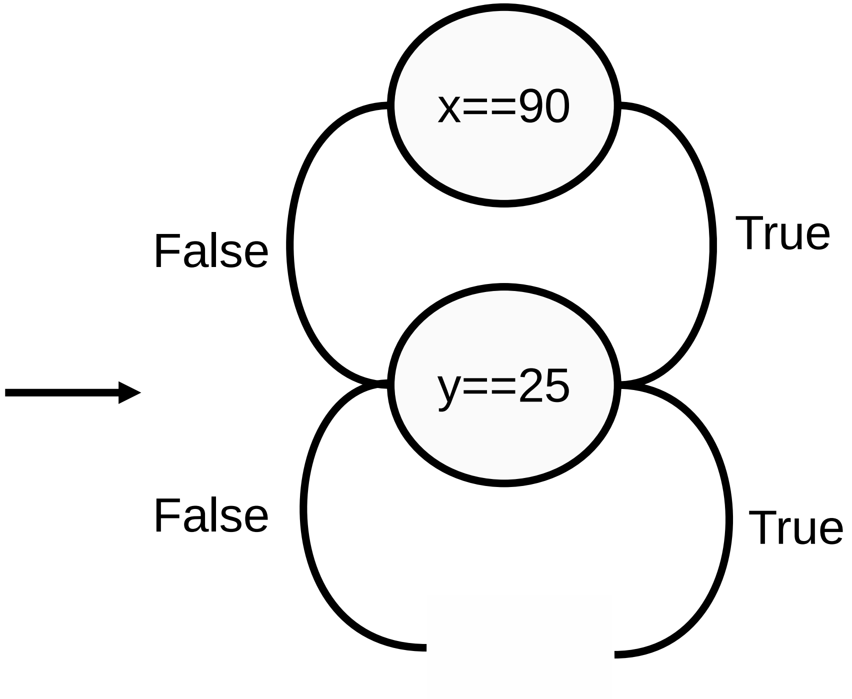
- Diverse coverage metrics: Line, Branch, Path, ...
  - Line coverage: # of executed lines / # of total lines
  - 100% line coverage with test-cases.

```
void main (int x, int y) {
   if (x == 90)
      printf(“X Error”)
   if (y ==25)
      printf(“Y Error”)
}
```

Code Coverage

- Diverse coverage metrics: Line, Branch, Path, ...
  - Branch coverage: # of executed branches / # of total branches
    - # of branches = if (or while) statement X 2
  - 100% branch coverage with test-cases

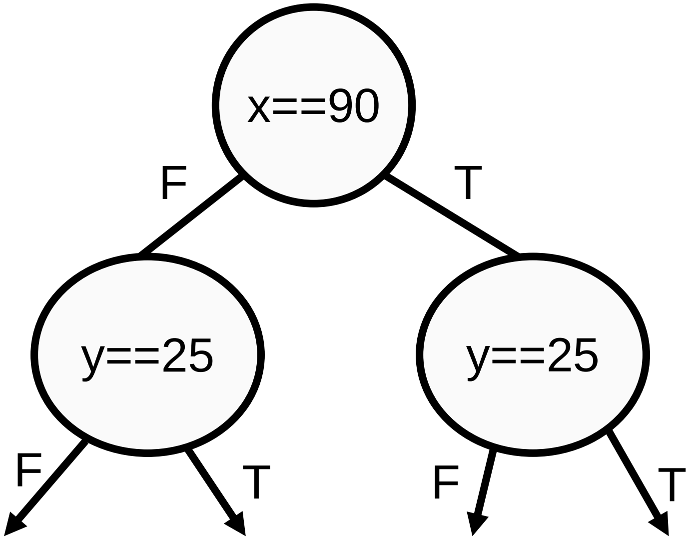
```
void main (int x, int y) {
   if (x == 90)
      printf(“X Error”)
   if (y ==25)
      printf(“Y Error”)
}
```



Code Coverage

- Diverse coverage metrics: Line, Branch, Path, ...
  - Path coverage: # of executed paths / # of total paths
    - # of paths = $2^{\text{\# of if/while statements}}$ (e.g., 4 execution paths = 2^2)
  - 100% path coverage with test-cases

```
void main (int x, int y) {
   if (x == 90)
      printf("X Error")
   if (y ==25)
      printf("Y Error")
}
```

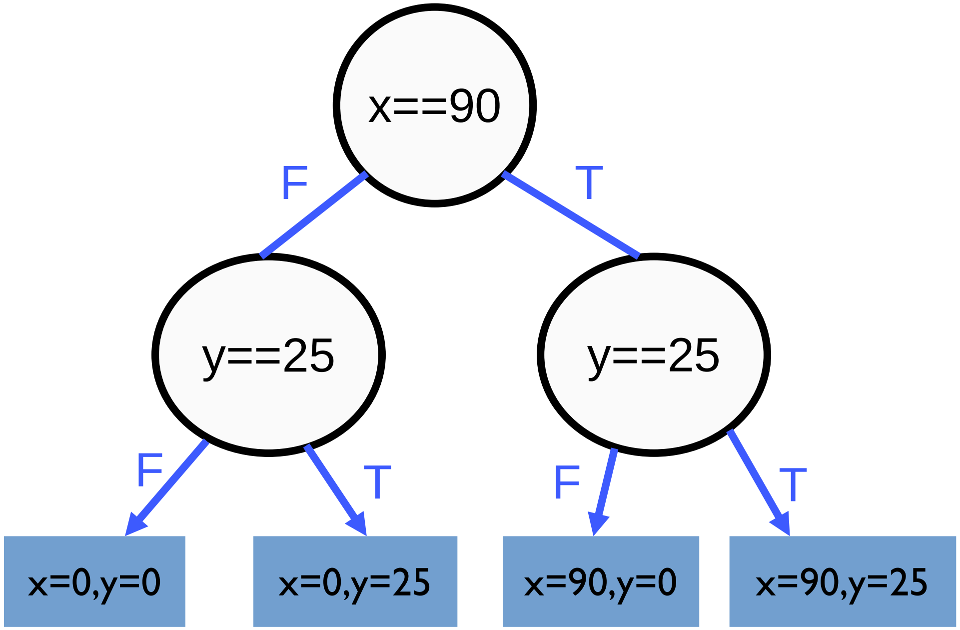


Code Coverage

- Diverse coverage metrics: Line, Branch, Path, ...
  - Path coverage: # of executed paths / # of total paths
    - # of paths = $2^{\#\text{ of branches}}$ (e.g., 4 execution paths = 2^2)
  - 100% path coverage with 4 test-cases
    - Test-cases: (0,0), (0,25), (90,0), (90,25)

```
void main (int x, int y) {
   if (x == 90)
      printf("X Error")
   if (y ==25)
      printf("Y Error")
}
```

Path coverage: 100%



Code Coverage

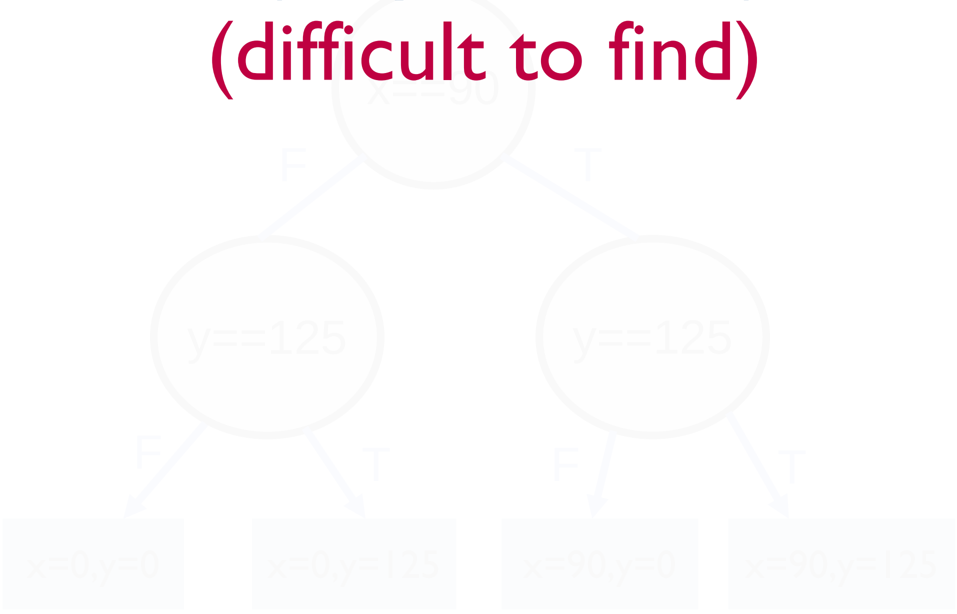
- Diverse coverage metrics: Line, Branch, Path, ...
  - Path coverage: # of executed paths / # of total paths
    - # of paths = $2^{\# \text{ of branches}}$ (e.g., 4 execution paths = 2^2)
  - 100% branch coverage with 4 test cases
    - Test cases: (0,0), (0,125), (90,0), (90,125)

Path Coverage > Branch Coverage

(difficult to reach) (easy to reach)

(easy to find) (difficult to find)

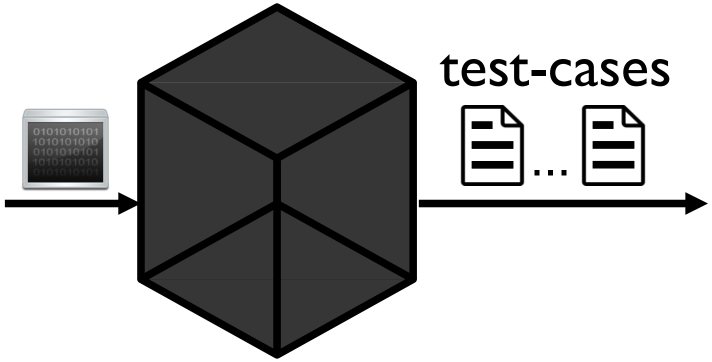
```
void main (int x, int y) {
  if (x == 90)
     printf("X Error")
  if (y ==125)
     printf("Y Error")
}
```



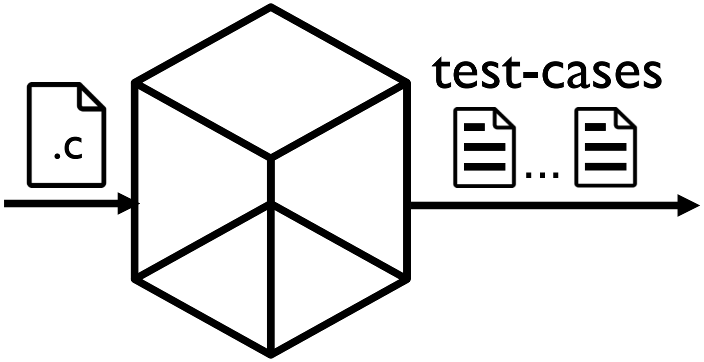
Path coverage: 100%

Software Testing Methods

- Classify into two testing methods



Black-box testing
(fuzzing)
with no source code
(+) cheap
(-) naive



White-box testing
(symbolic execution)
with source code
(-) expensive
(+) systematic

Symbolic Execution

- A promising software testing method
  - Actively used in both academia and industry.
  - Replace program inputs with symbolic variables.

Microsoft SAGE

KLEE

Pex

NASA JPF

Chameleon

HOMI

Symbolic Execution

- A promising software testing method
  - Actively used in both academia and industry

Microsoft
SAGE

JPF

> “SAGE found one-third of all the bugs discovered during the development of Windows 7. ” [1]

Windows 7

---

[1] SAGE: whitebox fuzzing for security testing, Communications of the ACM, 2012

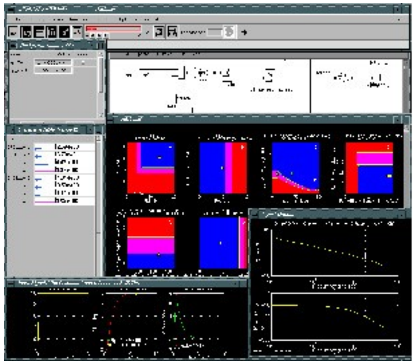
Symbolic Execution

- A promising software testing method
  - Actively used in both academia and industry

Microsoft SAGE

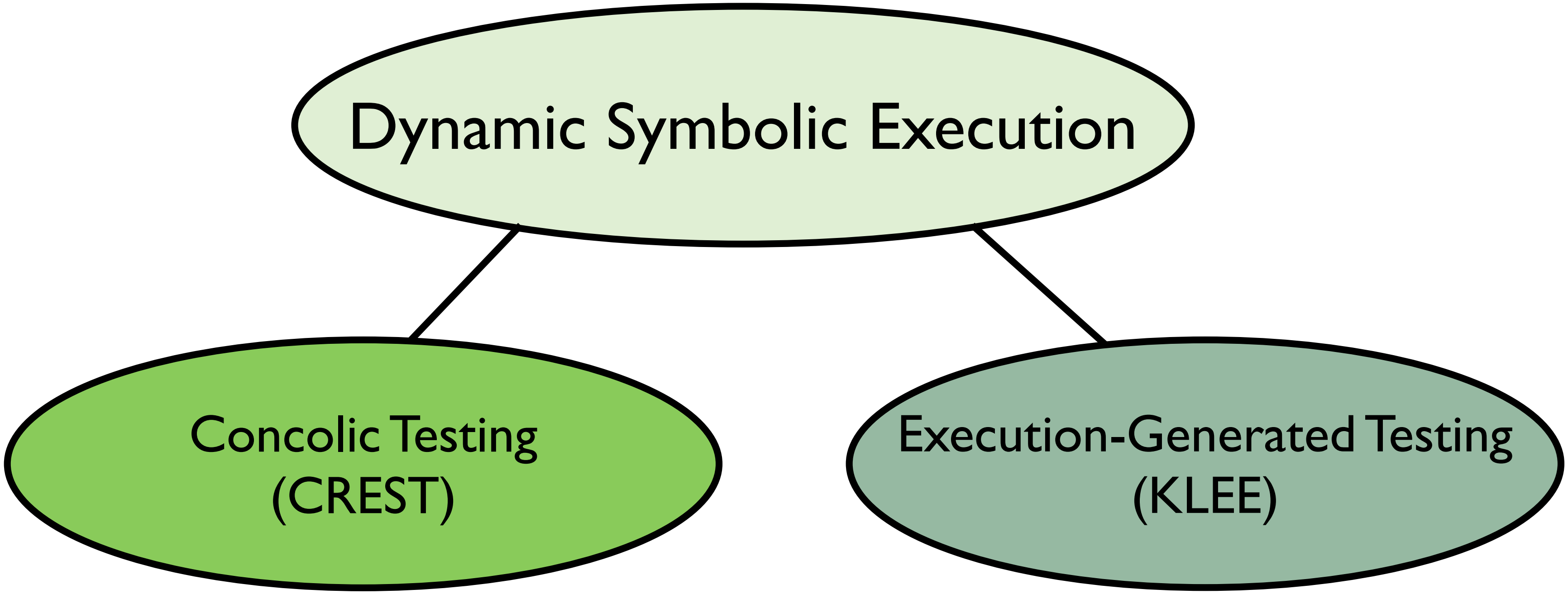
JPF

> "Our analysis discovered a serious bug in the NASA flight software." [2]



[2] Combining Unit-level Symbolic Execution and System-level Concrete Execution for Testing NASA Software, ISSTA, 2008

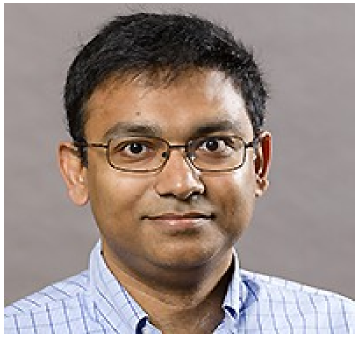
Modern Symbolic Execution



Dynamic Symbolic Execution

Concolic Testing
(CREST)

Execution-Generated Testing
(KLEE)



Koushik sen
(UC Berkeley)

Cristian Cadar
(Imperial College London)

Concolic Testing
(Concrete + Symbolic)

brunch
(breakfast + lunch)

Concolic Testing vs Random Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

- Probability of reaching the error?

  (1≤ x,y ≤100)

  (1) 0.04% (2) 0.4% (3) 4% (4) 40%

Limitation of Random Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

- Probability of reaching the error? (1≤ x,y ≤100)

  0.4 %

  - Random testing requires 250 runs.
  - Concolic testing finds it in 3 runs.

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

x = 22, y = 7

Symbolic State

x = α, y = β

PathCond (PC): true

Initial Random Input
x=22, y=7

1st iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

$x = 22, y = 7,$
$z = 14$

Symbolic State

$x = \alpha, y = \beta, z = 2*\beta$

PC: true

1st iteration

Concolic Testing

```
int twice (int v) {
      return 2 × v;
}

void main (int x, int y) {

      z  = twice (y);

      if ( z == x ) {

            if ( x > y+10 ) {
                  error;
            }
      }
}
```

Concrete State	Symbolic State
x = 22, y = 7, z = 14	x = α, y = β, z = 2*β *PC*: 2*β ≠ α

1st iteration

Concolic Testing

```
int twice (int v) {
      return 2 × v;
}

void main (int x, int y) {

      z  = twice (y);

      if ( z == x ) {

            if ( x > y+10 ) {
                  error;
            }
      }
}
```

Concrete State

Symbolic State

<SMT Solver>

Solve: 2*β = α

Solution: α=2, β=1

x = 22, y = 7,
z = 14

x = α, y = β, z = 2*β
PC: 2*β ≠ α

1st iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State
$x = 2, y = 1$	$x = \alpha, y = \beta$
	PC: true

2nd iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

x = 2, y = 1,
z=2

Symbolic State

x = α, y = β, z = 2*β

PC: true

2nd iteration

Concolic Testing

```
int twice (int v) {
     return 2 × v;
}

void main (int x, int y) {

     z  = twice (y);

     if ( z == x ) {

          if ( x > y+10 ) {
               error;
          }
     }
}
```

Concrete State	Symbolic State
$x = 2, y = 1,$ $z=2$	$x = \alpha, y = \beta, z = 2*\beta$ *PC*: $2*\beta = \alpha$

2nd iteration

Concolic Testing

```
int twice (int v) {
      return 2 × v;
}

void main (int x, int y) {

      z  = twice (y);

      if ( z == x ) {

            if ( x > y+10 ) {
                  error;
            }
      }
}
```

Concrete State

x = 2, y = 1,
z=2

Symbolic State

$x = \alpha, y = \beta, z = 2*\beta$

PC: $(2*\beta = \alpha) \wedge (\alpha \leq \beta+10)$

2nd iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State | Symbolic State

<SMT Solver>

Solve: $(2*\beta = \alpha) \wedge (\alpha > \beta+10)$

Solution: $\alpha=22, \beta=11$

$x = 2, y = 1,$
$z=2$

$x = \alpha, y = \beta, z = 2*\beta$

PC: $(2*\beta = \alpha) \wedge$ **$(\alpha \leq \beta+10)$**

2nd iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

x = 22, y = 11

Symbolic State

$x = \alpha, y = \beta$

PC: true

3rd iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

x = 22, y = 11,
z = 22

Symbolic State

x = α, y = β, z =2*β

PC: true

3rd iteration

Concolic Testing

```
int twice (int v) {
     return 2 × v;
}

void main (int x, int y) {

     z  = twice (y);

     if ( z == x ) {

          if ( x > y+10 ) {
               error;
          }
     }
}
```

Concrete State

x = 22, y = 11,
z = 22

Symbolic State

$x = \alpha, y = \beta, z = 2*\beta$

PC: $2*\beta = \alpha$

3rd iteration

Concolic Testing

```
int twice (int v) {
    return 2 × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

error-triggering input

x = 22, y = 11,
z = 22

Symbolic State

$x = \alpha, y = \beta, z = 2*\beta$

PC: $(2*\beta = \alpha) \wedge (\alpha > \beta+10)$

3rd iteration

What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

x = 22, y = 7

Symbolic State

x = α, y = β

PathCond (PC): true

Initial Random Input
x=22, y=7

1st iteration

What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State
x = 22, y = 7, z = 49	$x = \alpha, y = \beta, z = \beta*\beta$ *PC*: true

1st iteration

What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State
x = 22, y = 7, z = 49	$x = \alpha, y = \beta, z = \beta * \beta$ *PC*: $\beta * \beta \neq \alpha$

1st iteration

What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State

Symbolic State

<SMT Solver>

Solve: $\beta*\beta = \alpha$ //non-linear constraint

Solution: α=?, β=?

$x = 22, y = 7,$
$z = 49$

$x = \alpha, y = \beta, z = \beta*\beta$
PC: $\beta*\beta \neq \alpha$

1st iteration

What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State
x = 22, y = 7, z = 49	x = α, y = β, z = β*β *PC*: β*β ≠ α

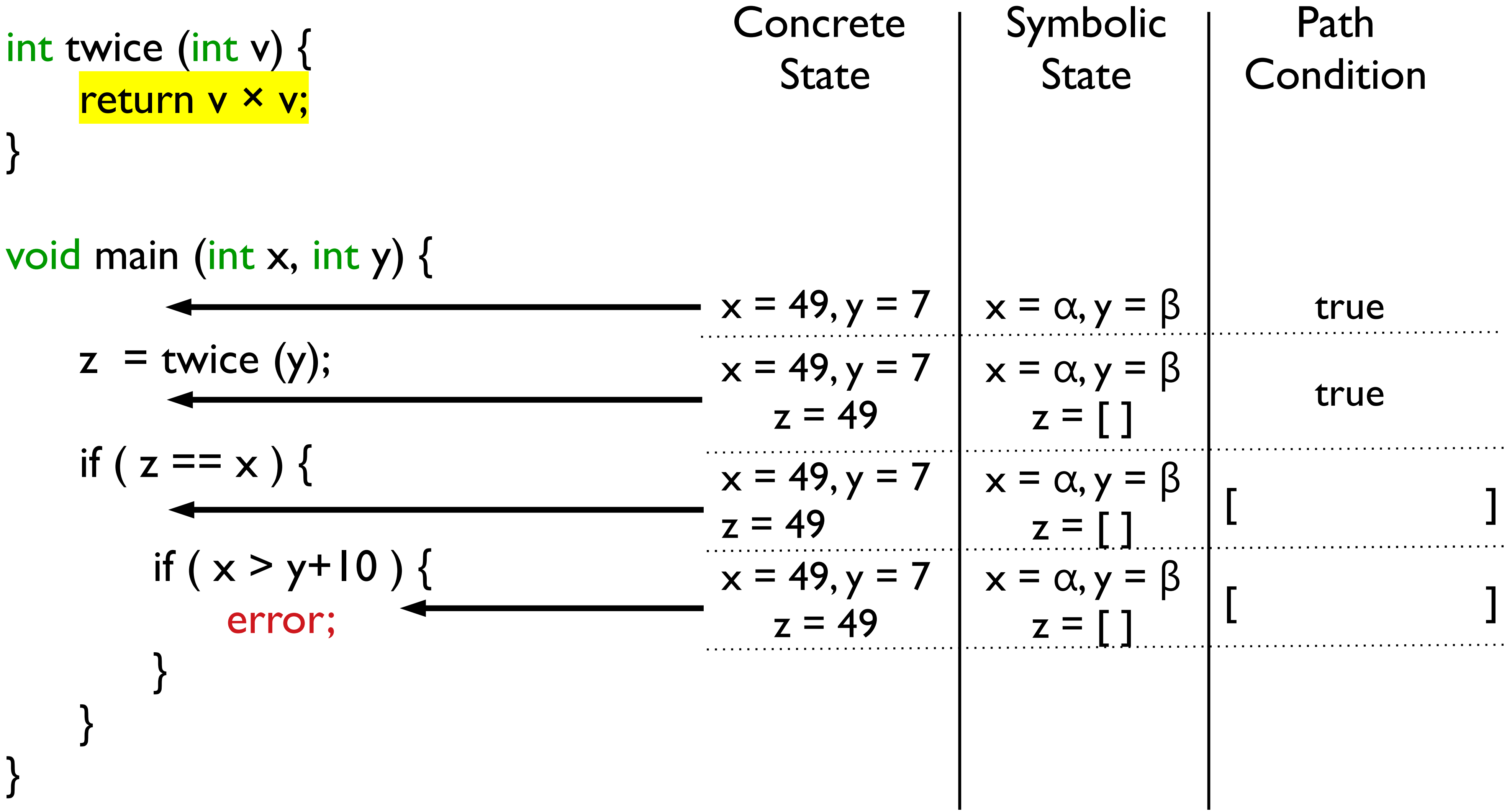
<SMT Solver>

Solve: 7*7 = α **//non-linear constraint**

Solution: α=49, β=7

1st iteration

What If ? (non-linear constraint)



```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = α, y = β	true
x = 49, y = 7 z = 49	x = α, y = β z = [ ]	true
x = 49, y = 7 z = 49	x = α, y = β z = [ ]	[ ]
x = 49, y = 7 z = 49	x = α, y = β z = [ ]	[ ]

2nd iteration

2. What If ? (non-linear constraint)

```
int twice (int v) {
    return v × v;
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x < y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = α, y = β	true
x = 49, y = 7 z = 49	x = α, y = β z = ?	true
x = 49, y = 7 z = 49	x = α, y = β z = ?	[ ]
x = 49, y = 7 z = 49		[ ]

2nd iteration

3. What If ? (external function)

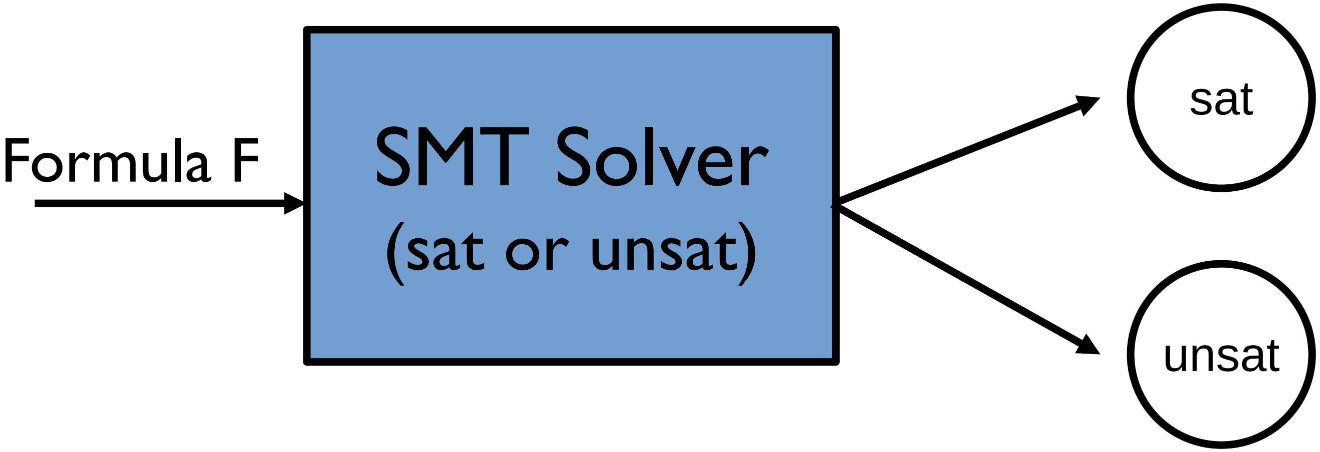
```
int twice (int v) {
    return external(v);
}

void main (int x, int y) {

    z  = twice (y);

    if ( z == x ) {

        if ( x > y+10 ) {
            error;
        }
    }
}
```

Concrete State	Symbolic State	Path Condition
x = 49, y = 7	x = α, y = β	

1st iteration

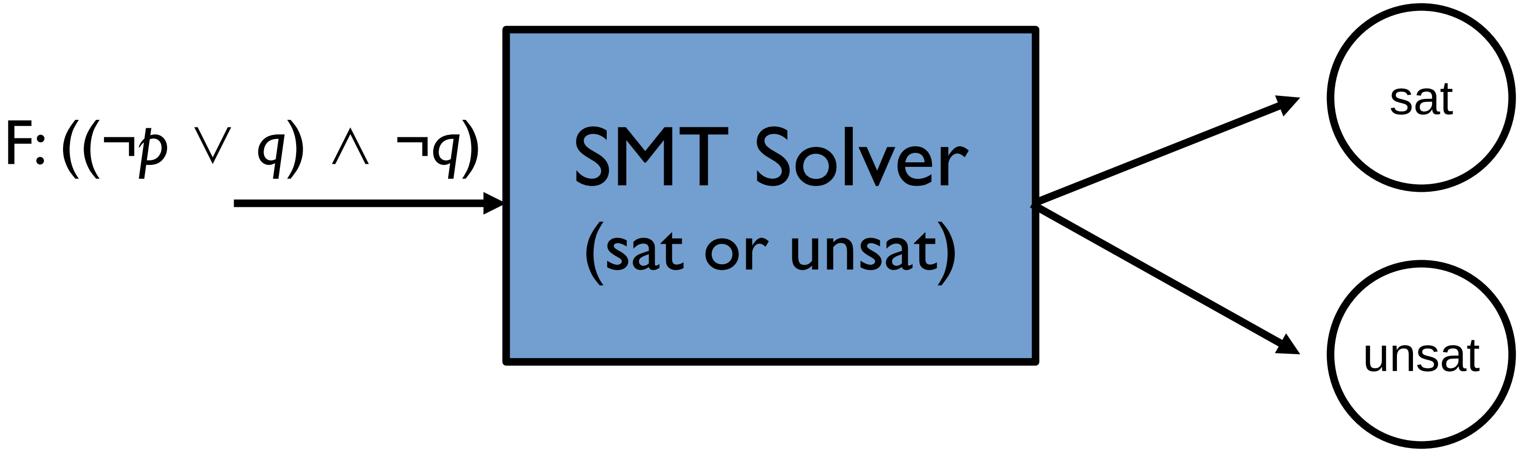
SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Take a given formula F and check whether F is satisfiable or not.
  - ex1) Boolean formula F (*p* and *q* denote boolean variables)
  - F is satisfiable iff there exist *p* and *q* values such that F is true.
  - F is unsatisfiable iff there are none of *p* and *q* values such that F is true.



SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Take a given formula F and check whether F is satisfiable or not.
  - ex1) Boolean formula F (p and q denote boolean variables)
  - F is satisfiable iff there exist p and q values such that F is true.
  - F is unsatisfiable iff there are none of p and q values such that F is true.



F: $((\neg p \vee q) \wedge \neg q)$

SMT Solver

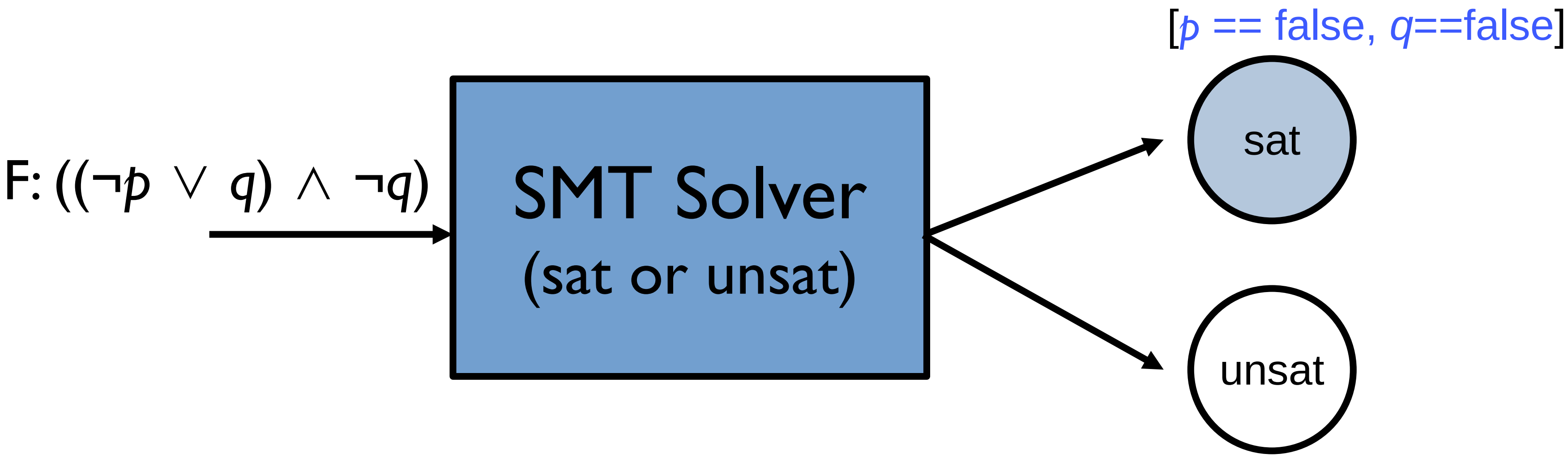
(sat or unsat)

sat

unsat

SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Take a given formula F and check whether F is satisfiable or not.
  - ex1) Boolean formula F (*p* and *q* denote boolean variables)
  - F is satisfiable iff there exist *p* and *q* values such that F is true.
  - F is unsatisfiable iff there are none of *p* and *q* values such that F is true.



F: $((\neg p \lor q) \land \neg q)$

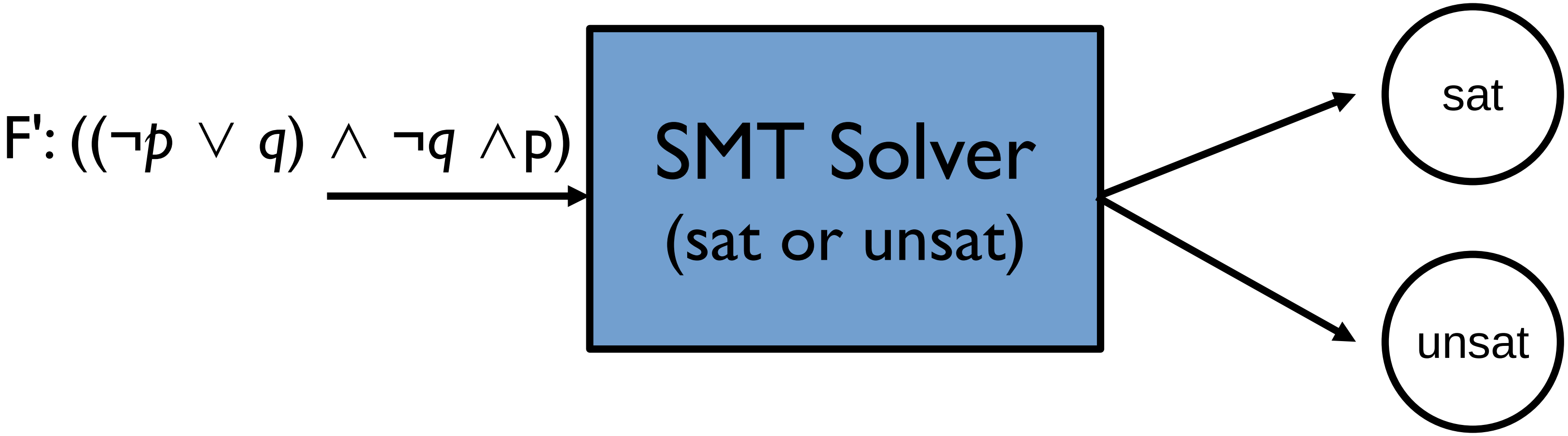
SMT Solver

(sat or unsat)

[*p* == false, *q*==false]

SMT Solver

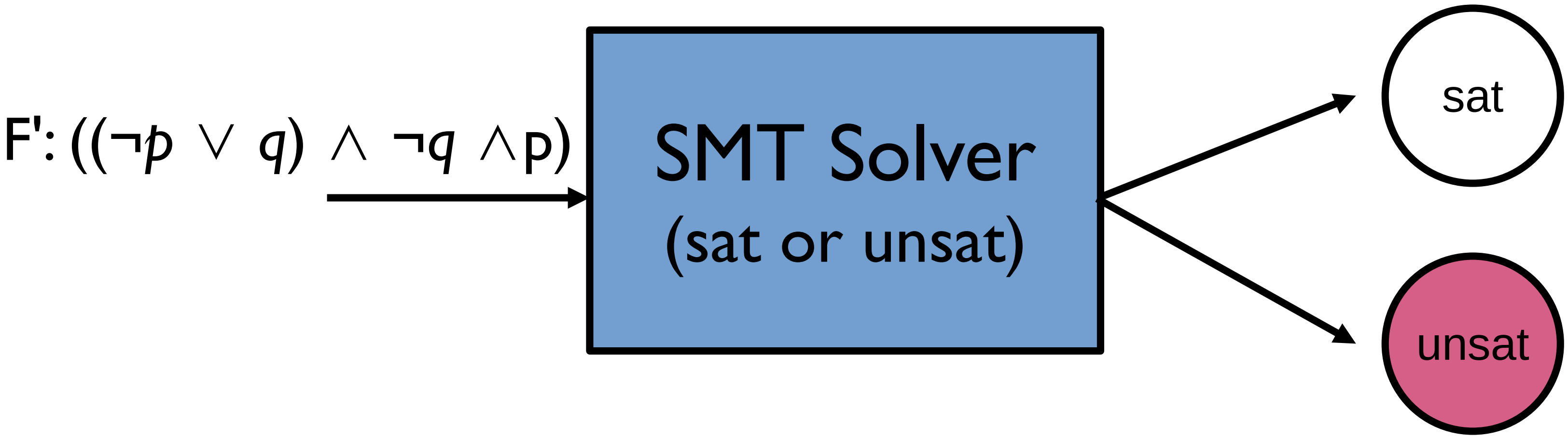
- Satisfiability Modulo Theories (SMT)
  - Take a given formula F and check whether F is satisfiable or not.
  - ex1) Boolean formula F (p and q denote boolean variables)
  - F is satisfiable iff there exist p and q values such that F is true.
  - F is unsatisfiable iff there are none of p and q values such that F is true.



F': $((\neg p \vee q) \wedge \neg q \wedge p)$

SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Take a given formula F and check whether F is satisfiable or not.
  - ex1) Boolean formula F (p and q denote boolean variables)
  - F is satisfiable iff there exist p and q values such that F is true.
  - F is unsatisfiable iff there are none of p and q values such that F is true.



F': $((\neg p \vee q) \wedge \neg q \wedge p)$

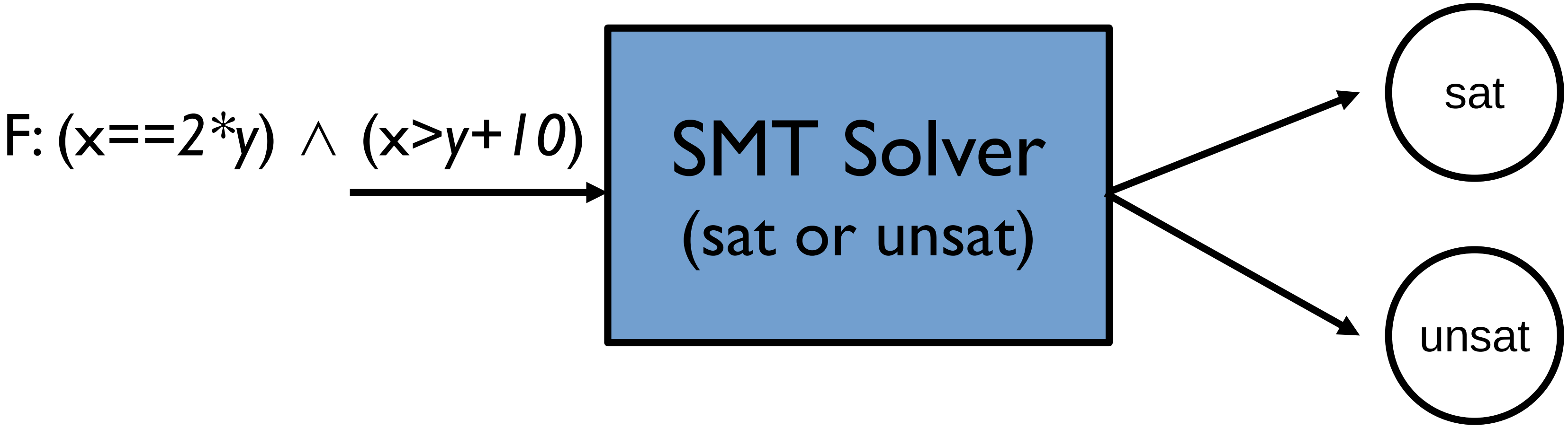
SMT Solver
(sat or unsat)

sat

unsat

SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Try to find an input that satisfies a given formula F.
  - ex2) The formula F (*x* and *y* denote integer variables)
  - F is satisfiable iff there exist *x* and *y* values such that F is true.
  - F is unsatisfiable iff there are none of *x* and *y* values such that F is true.



F: $(x==2*y) \land (x>y+10)$

SMT Solver

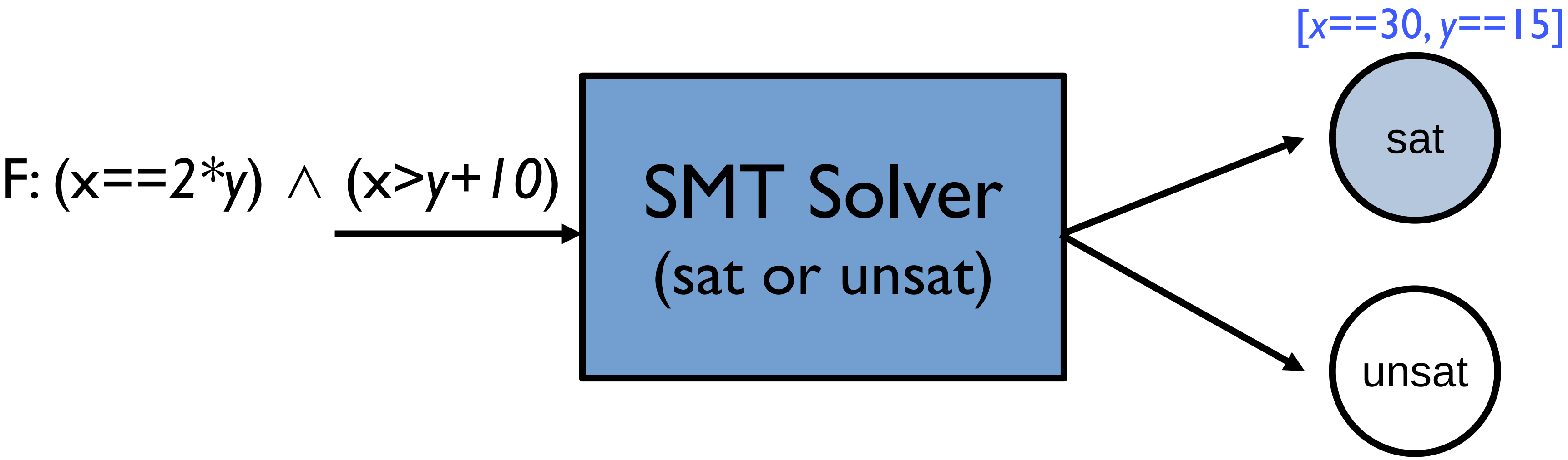
(sat or unsat)

sat

unsat

SMT Solver

- Satisfiability Modulo Theories (SMT)
  - Try to find an input that satisfies a given formula F.
  - ex2) The formula F (*x* and *y* denote integer variables)
  - F is satisfiable iff there exist *x* and *y* values such that F is true.
  - F is unsatisfiable iff there are none of *x* and *y* values such that F is true.



[x==30, y==15]

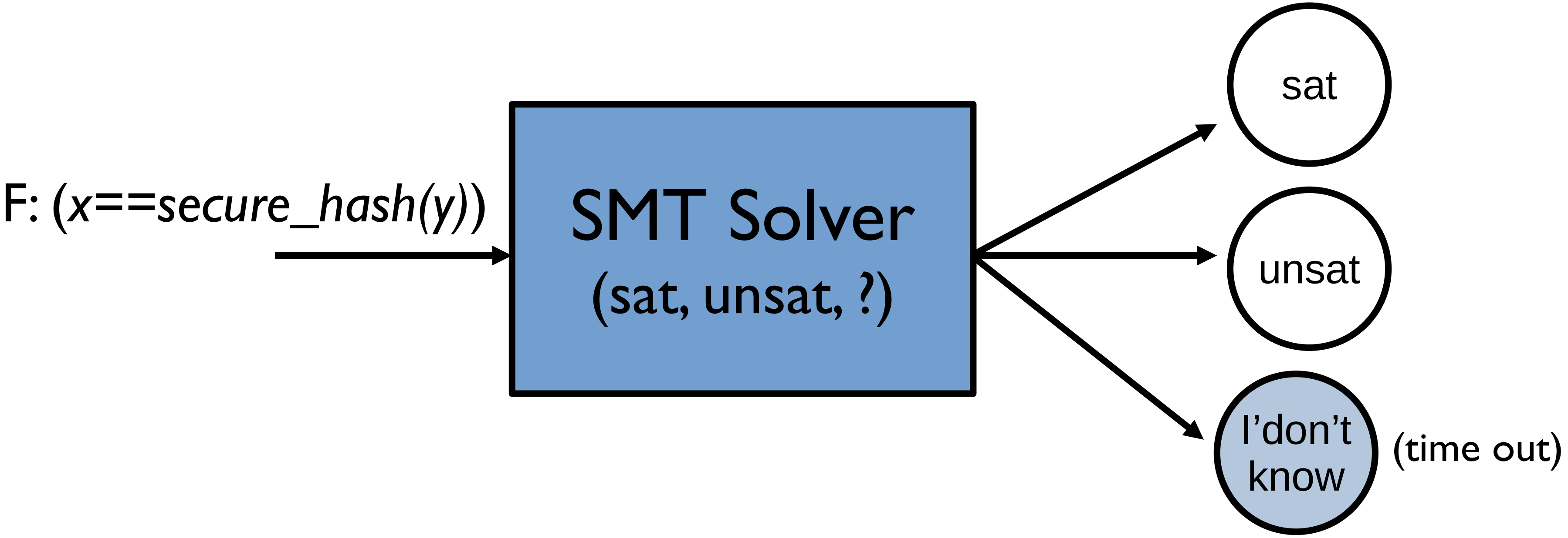
$F: (x==2*y) \wedge (x>y+10)$

SMT Solver

(sat or unsat)

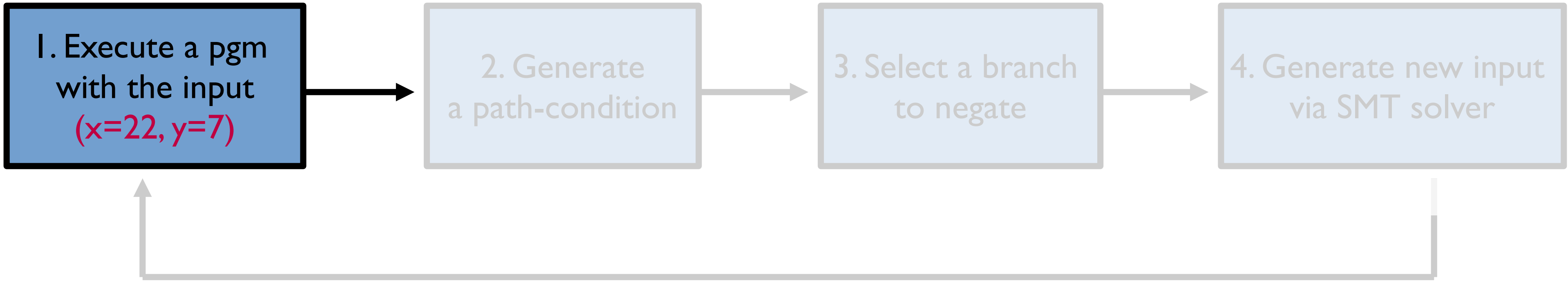
SMT Solver

- SMT solvers in real-world
  - F is satisfiable or unsatisfiable or ???



Concolic Testing

- How does concolic testing work?

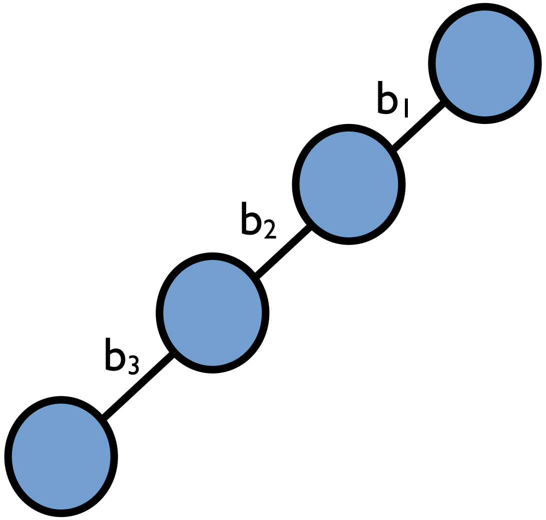


1. Execute a pgm with the input (x=22, y=7)

2. Generate a path-condition

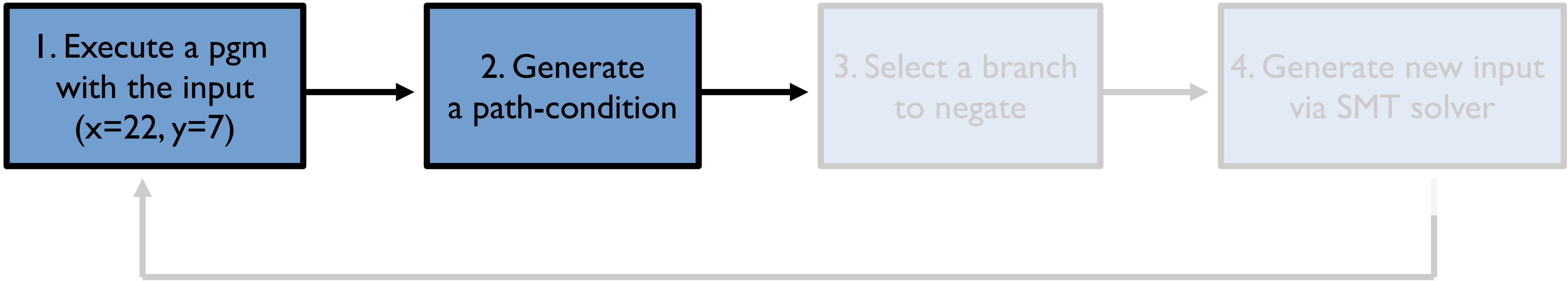
3. Select a branch to negate

4. Generate new input via SMT solver



Concolic Testing

- How does concolic testing work?

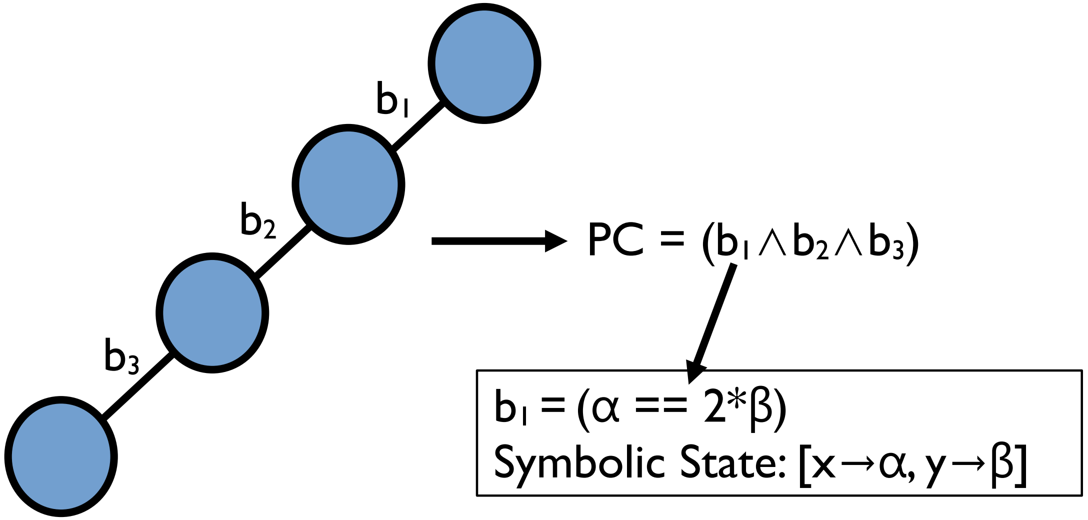


1. Execute a pgm with the input (x=22, y=7)

2. Generate a path-condition

3. Select a branch to negate

4. Generate new input via SMT solver

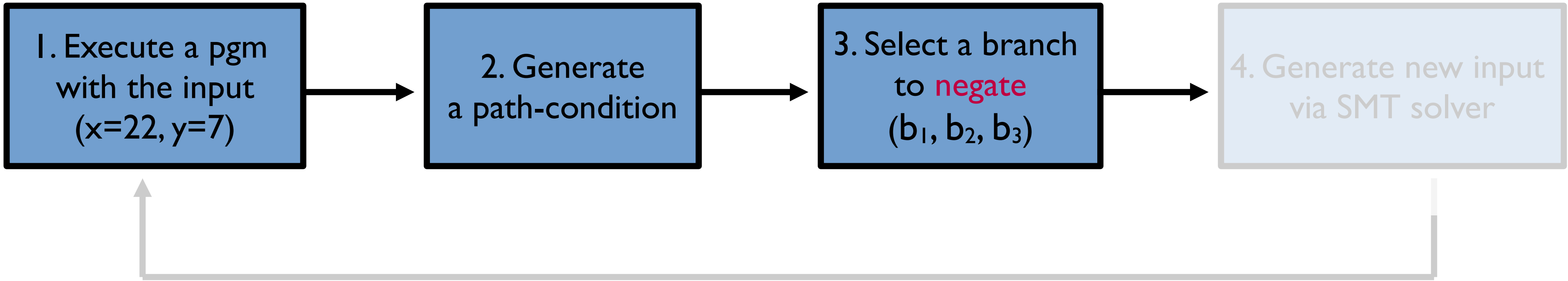


PC = $(b_1 \wedge b_2 \wedge b_3)$

$b_1 = (\alpha == 2*\beta)$
Symbolic State: $[x \rightarrow \alpha, y \rightarrow \beta]$

Concolic Testing

- How does concolic testing work?

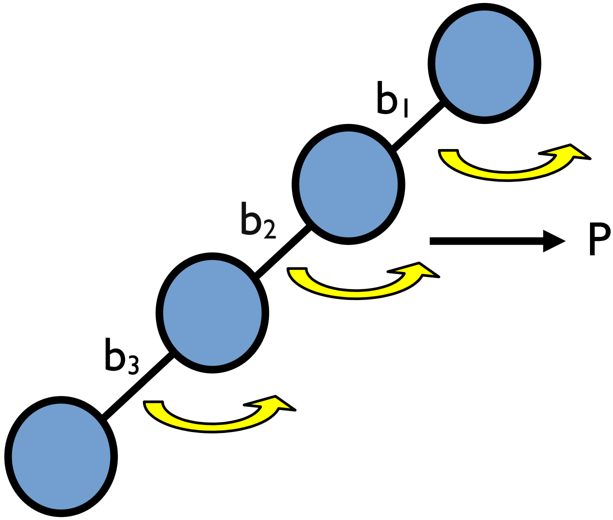


1. Execute a pgm with the input (x=22, y=7)

2. Generate a path-condition

3. Select a branch to negate (b_1, b_2, b_3)

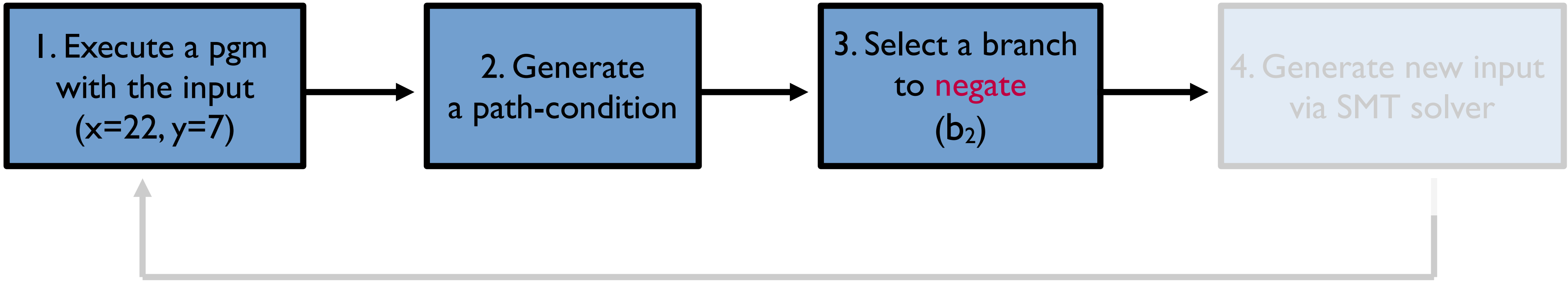
4. Generate new input via SMT solver



PC = $(b_1 \wedge b_2 \wedge b_3)$

Concolic Testing

- How does concolic testing work?

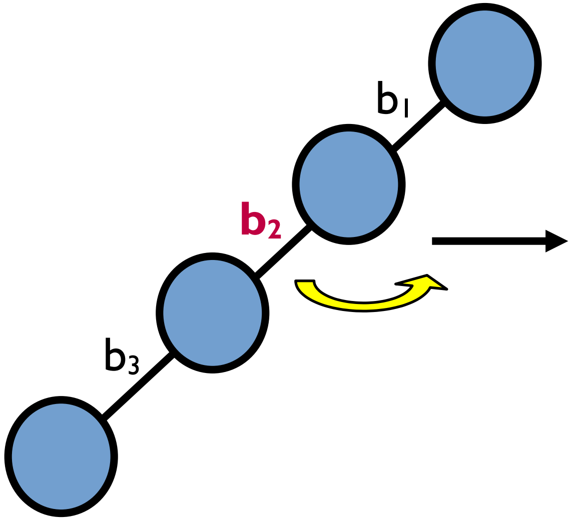


1. Execute a pgm with the input (x=22, y=7)

2. Generate a path-condition

3. Select a branch to **negate** (b_2)

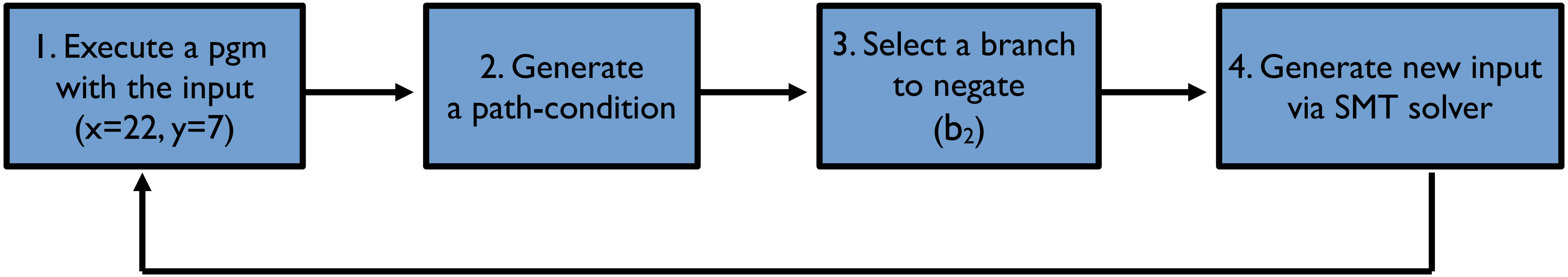
4. Generate new input via SMT solver



PC = $(b_1 \wedge \mathbf{b_2} \wedge b_3)$

Concolic Testing

- How does concolic testing work?

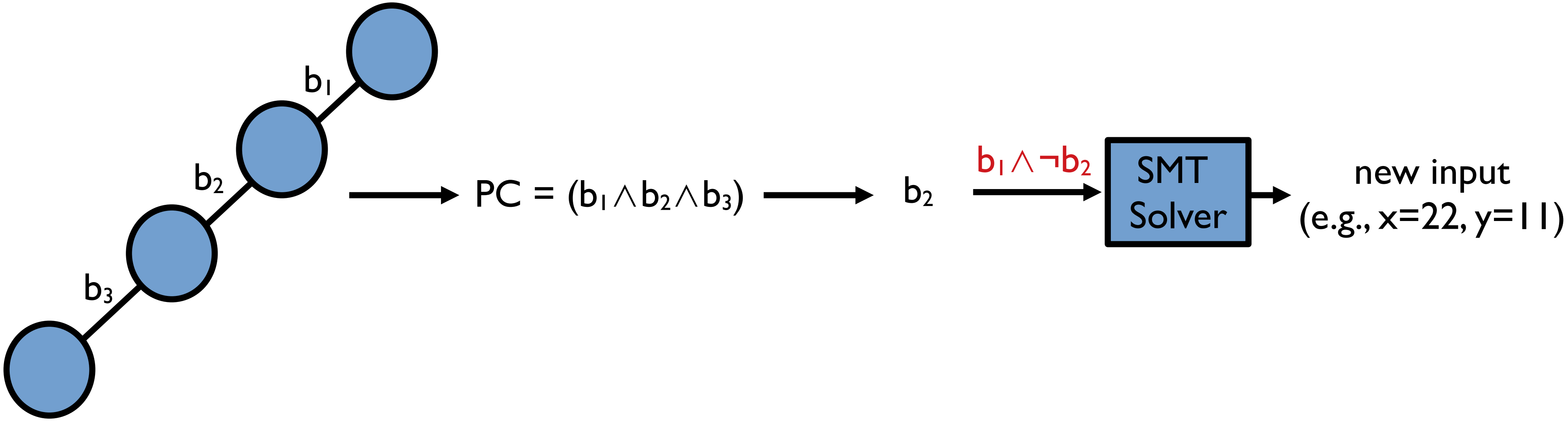


1. Execute a pgm with the input (x=22, y=7)

2. Generate a path-condition

3. Select a branch to negate (b_2)

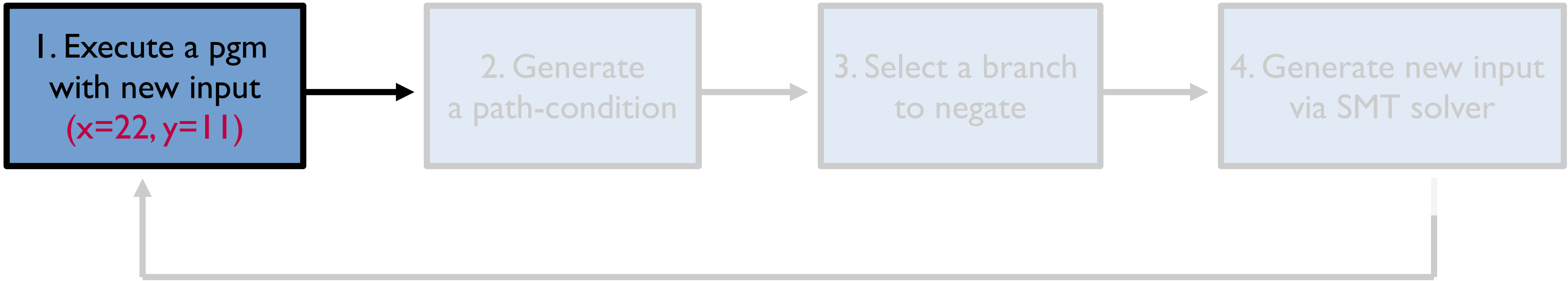
4. Generate new input via SMT solver



$\text{PC} = (b_1 \wedge b_2 \wedge b_3)$ → b_2 $\xrightarrow{b_1 \wedge \neg b_2}$ SMT Solver → new input (e.g., x=22, y=11)

Concolic Testing

- How does concolic testing work?

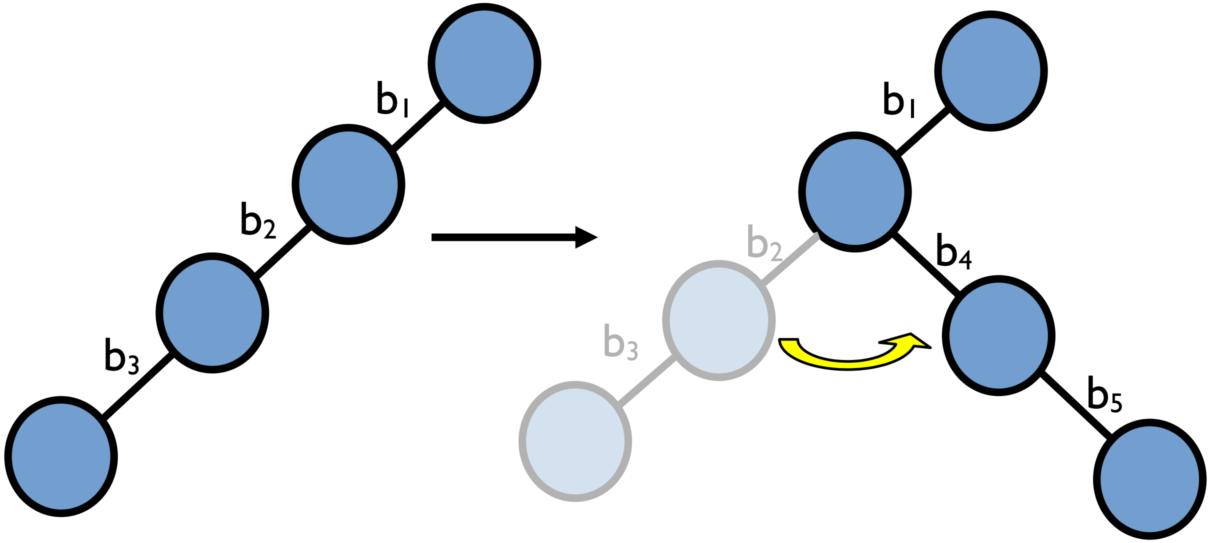


1. Execute a pgm with new input (x=22, y=11)

2. Generate a path-condition

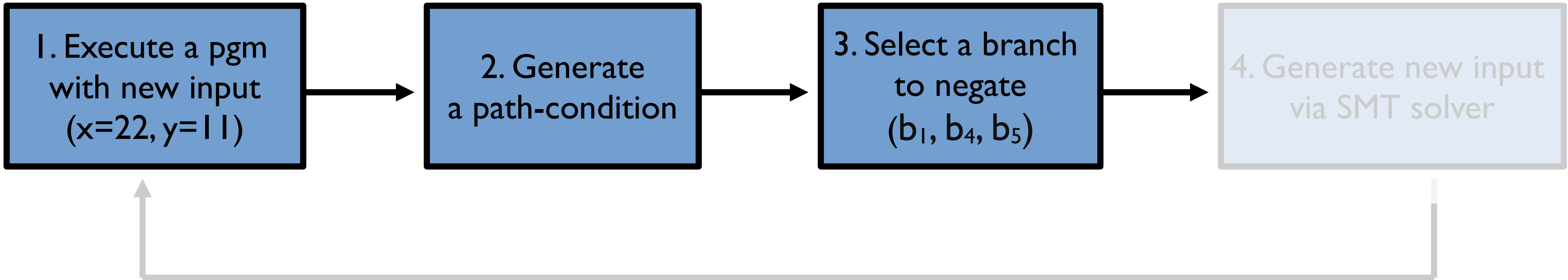
3. Select a branch to negate

4. Generate new input via SMT solver



Concolic Testing

- How does concolic testing work?

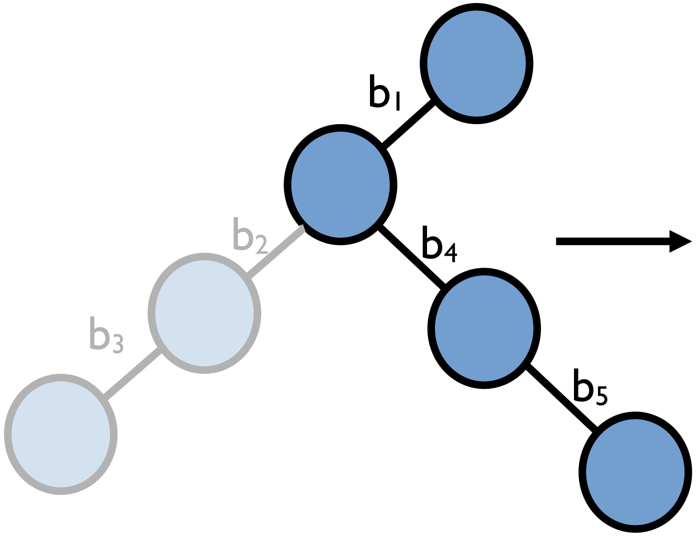


1. Execute a pgm with new input (x=22, y=11)

2. Generate a path-condition

3. Select a branch to negate (b_1, b_4, b_5)

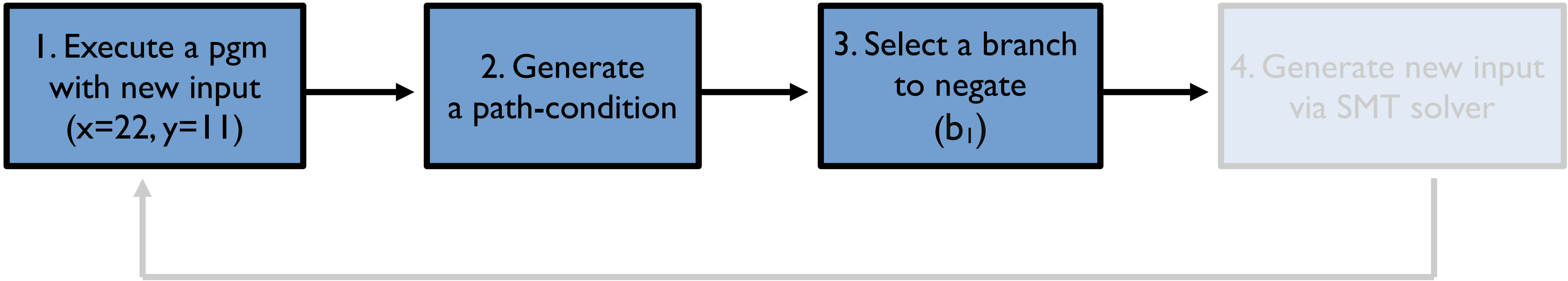
4. Generate new input via SMT solver



PC = ($b_1 \wedge b_4 \wedge b_5$)

Concolic Testing

- How does concolic testing work?

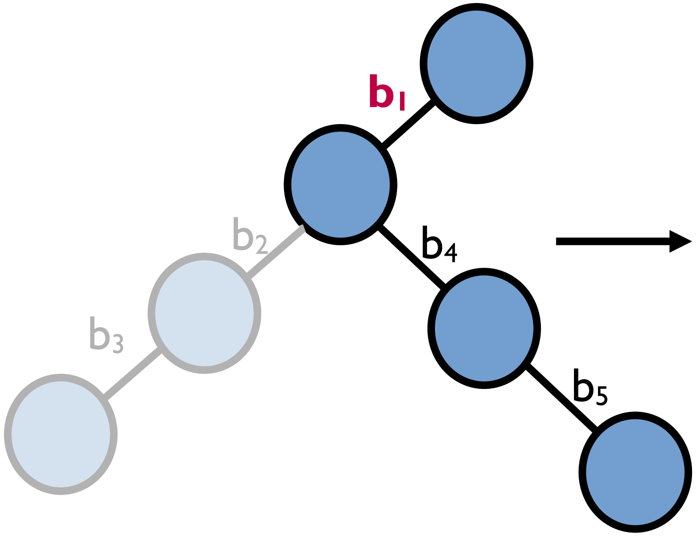


1. Execute a pgm with new input (x=22, y=11)

2. Generate a path-condition

3. Select a branch to negate (b_1)

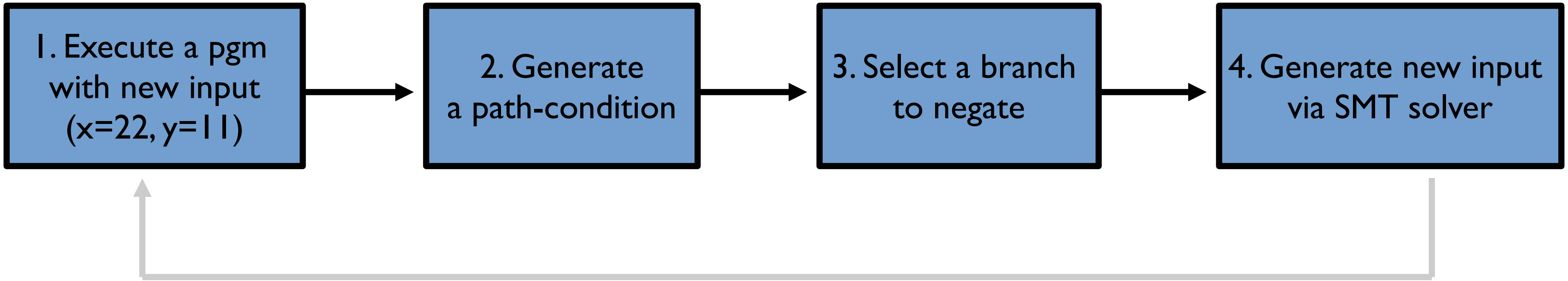
4. Generate new input via SMT solver



PC = ($b_1 \wedge b_4 \wedge b_5$)

Concolic Testing

- How does concolic testing work?

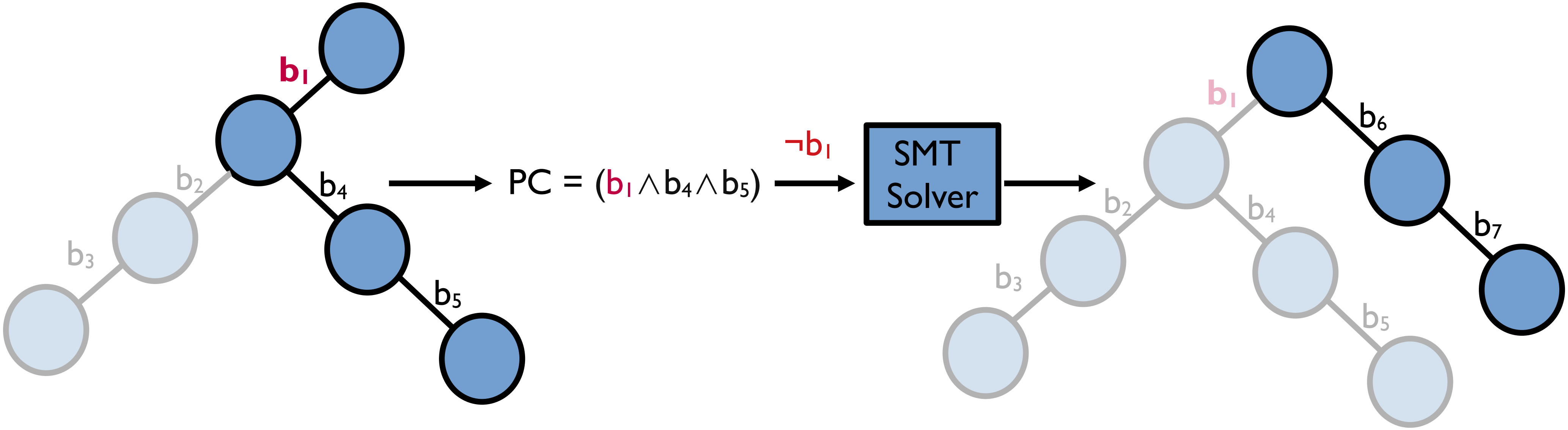


1. Execute a pgm with new input (x=22, y=11)

2. Generate a path-condition

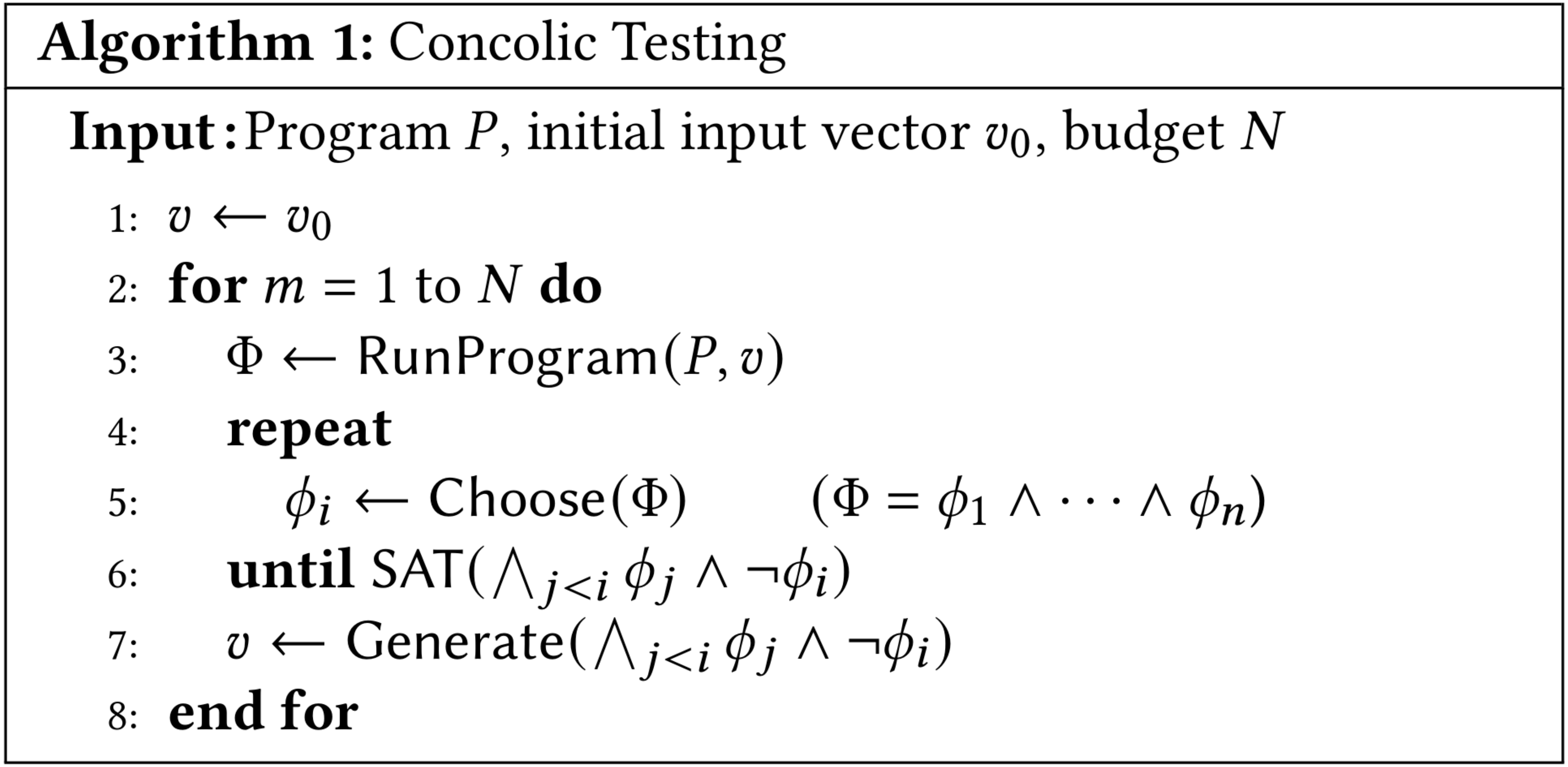
3. Select a branch to negate

4. Generate new input via SMT solver



Concolic Testing

- How does concolic testing work?



Algorithm 1: Concolic Testing

```
Input: Program $P$, initial input vector $v_0$, budget $N$
$v \leftarrow v_0$
for $m = 1$ to $N$ do
    $\Phi \leftarrow \text{RunProgram}(P, v)$
    repeat
        $\phi_i \leftarrow \text{Choose}(\Phi)$        $(\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
    until $\text{SAT}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
    $v \leftarrow \text{Generate}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
end for
```

Summary

- The goal of SW testing is to find the bugs and increase code coverage effectively.

- Concolic testing is a systematic technique that combines concrete and symbolic executions.

Thank You