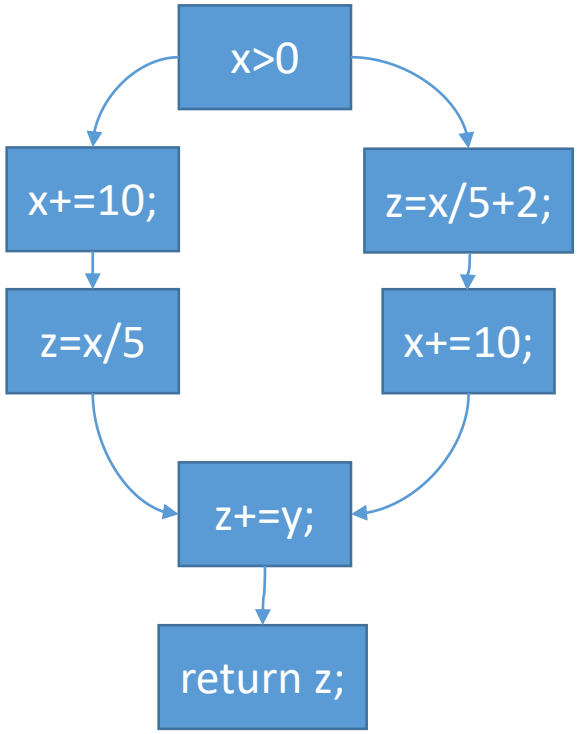
软件分析

符号执行

熊英飞
北京大学
2018

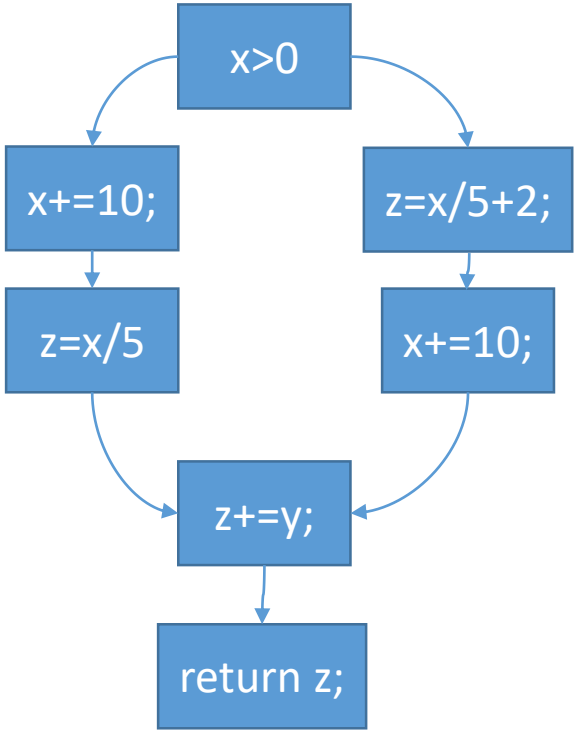
符号执行

```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



符号执行

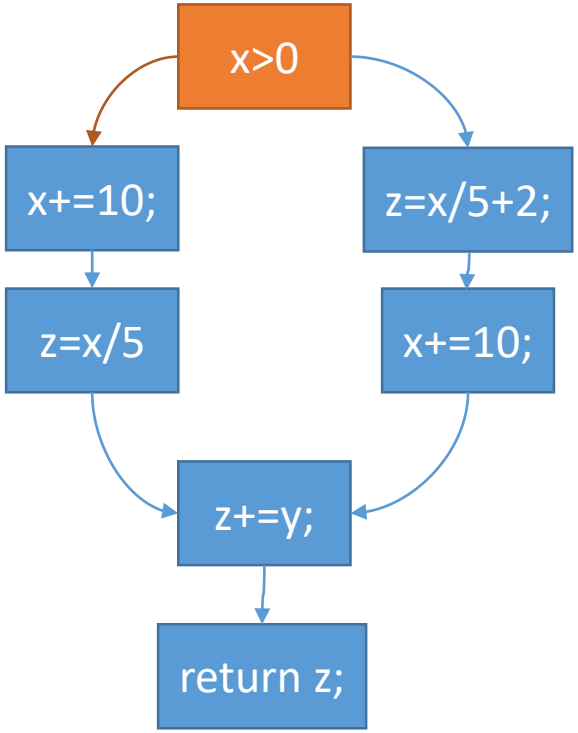
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a
y=b
z=?

符号执行

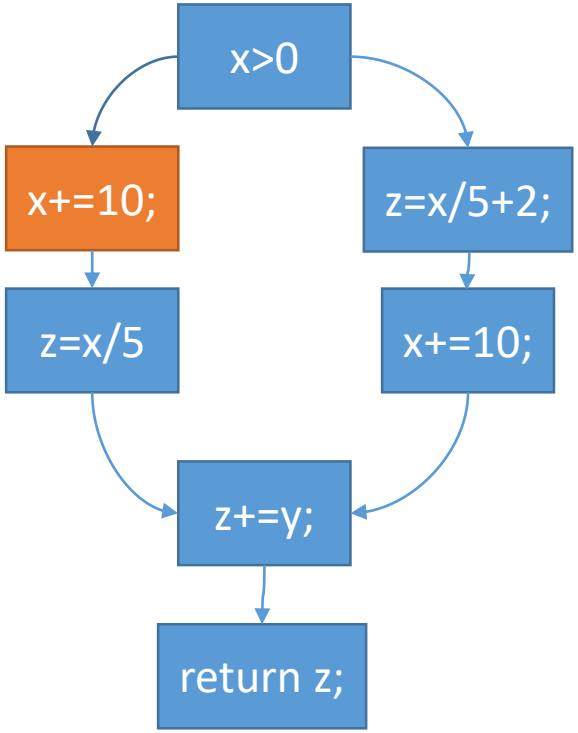
- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



x=a
y=b
z=?
a>0

符号执行

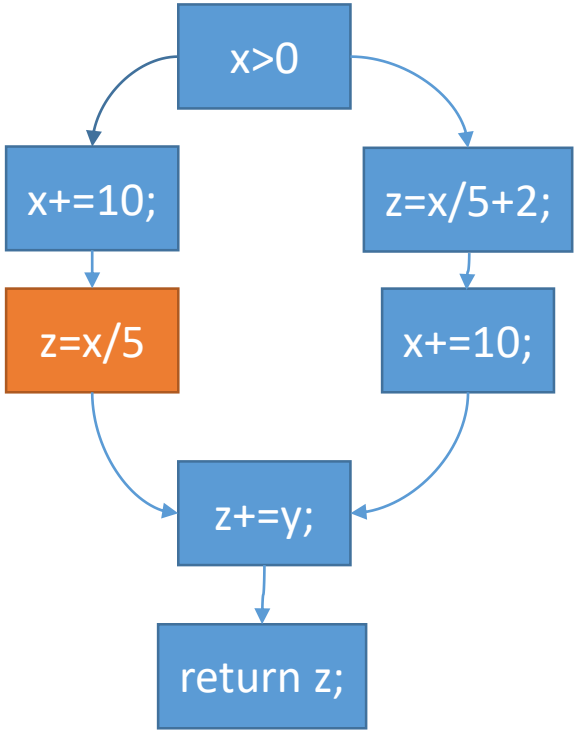
- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



x=a+10
y=b
z=?
a>0

符号执行

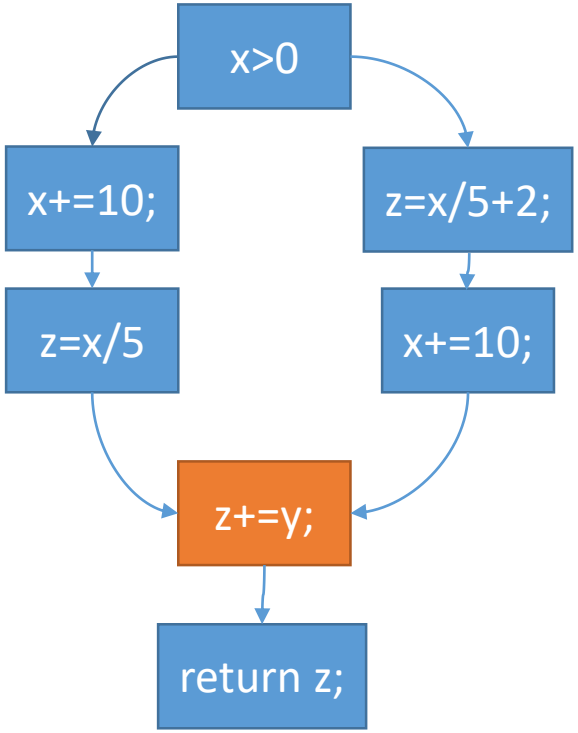
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a+10
y=b
z=(a+10)/5
a>0

符号执行

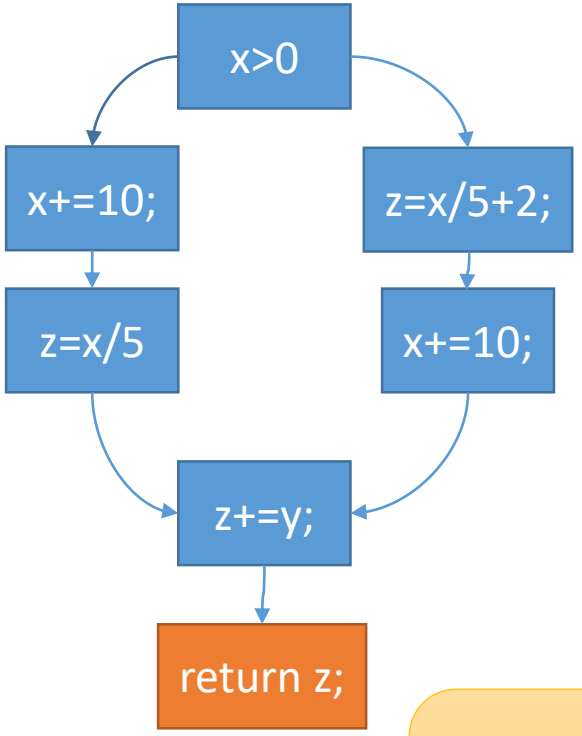
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a+10
y=b
z=(a+10)/5+b
a>0

符号执行

```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a+10
y=b
z=(a+10)/5+b
a>0

返回值为(a+10)/5+b
且a>0

符号执行

- 程序的规约通常表示为前条件和后条件
  - 前条件：a>0, b>0
  - 后条件：return > 0
- 形成命题：
  - a>0 /\ b>0 => (a+10)/5+b > 0
  - 命题成立=逆命题不可满足
  - 用SMT Solver可求解
- 规约被违反=任意路径对应的命题不成立
- 规范被满足=所有路径对应的命题都成立
  - 通常做不到
  - 对于循环，遍历有限次

符号执行举例

- 用符号执行发现缓冲区溢出
  - ……
  - a[i] = 4;
  - 判断后条件0<=i&&i<a.length是否总是成立
- 用符号执行发现除0错误
  - ……
  - x = 3 / i;
  - 判断后条件i!=0是否总是成立
- 用符号执行发现路径可行性
  - 判断给定路径上的路径约束是否可满足

基于霍尔逻辑的符号执行

霍尔逻辑

- 霍尔三元组
  - {前条件}语句{后条件}
- 霍尔逻辑表示三者之间的推导关系
- 又称为公理语义

While语言

Statement ::=

| skip

| while (Expr) Statement

| if (Expr) Statement else Statement

| Statement; Statement

| Var = Expr

霍尔逻辑规则

$$\textsc{Skip}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}} \qquad \textsc{Assign}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

$$\textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}} \qquad \textsc{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

$$\textsc{Consequence}\ \frac{\vDash (P \Rightarrow P') \qquad \{P'\}\ c\ \{Q'\} \qquad \vDash (Q' \Rightarrow Q)}{\{P\}\ c\ \{Q\}}$$

$$\textsc{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

用霍尔逻辑证明举例

- if (x > 0) x += 10; else x = 20;
  - 该程序执行结束后，x是否一定大于0?
- 根据Assign，可得
  - {x+10>0} x+=10 {x > 0}
  - {True} x=20 {x >0}
- 因为x>0 => x+10 > 0且¬x>0 => True，根据Consequence，可得
  - {x>0} x+=10 {x > 0}
  - {¬x>0} x+=10 {x > 0}
- 根据If，可得
  - {True} if (x > 0) x += 10; else x = 20;{x>0}

用霍尔逻辑证明练习

- while (x < 10) x += 1;
  - 该程序执行结束后，x是否一定大于0?
- 根据Assign，可得
  - {True} x+=1 {True}
- 根据Consequence，可得
  - {x<10/\True} x+=10 {True}
- 根据While，可得
  - {True} while (x < 10) x += 1; {x>=10}
- 根据Consequence，可得
  - {True} while (x < 10) x += 1; {x>0}

谓词转换计算

- 最弱前条件计算：给定后条件和语句，求能形成霍尔三元组的最弱前条件
- 最强后条件计算：给定前条件和语句，求能形成霍尔三元组的最强后条件

- 基于谓词转换的符号执行
  - 给定输入需要满足的条件P，代码c，输出需要满足的条件Q
  - 前向符号执行：基于P和c计算最强后条件Q’，验证Q’->Q是否恒成立
  - 后向符号执行：基于Q和c计算最弱后条件P’，验证P->P’是否恒成立

最弱前条件计算

- $wp(skip, Q) = Q$

$$\textsc{Skip}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

- $wp(x := a, Q) = Q[a/x]$

$$\textsc{Assign}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

- $wp(c_1; c_2, Q) =$
  $wp\big(c_1, wp(c_2, Q)\big)$

$$\textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

- $wp(if\ b\ then\ c_1\ else\ c_2, Q) =$
  $\big(b \to wp(c_1, Q)\big)$
  $\wedge \big(\neg b \to wp(c_2, Q)\big)$

$$\textsc{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

最弱前条件：举例

- wp(if (x > 0) x += 10; else x = 20, x>0)
  - =(x>0->wp(x+=10, x>0)) /\ (x<=0 -> wp(x=20, x>0))
  - =(x>0->x+10>0) /\ (x<=0 -> 20>0)
  - =True

最弱前条件计算：循环

- $wp(while\ b\ do\ c, Q) = \exists i \in Nat. L_i(Q)$
  - where
    - $L_0(Q) = false$
    - $L_{i+1}(Q) = (\neg b \Rightarrow Q) \wedge \Big(b \Rightarrow wp\big(c, L_i(Q)\big)\Big)$
- 解释： i表示将循环展开几次
  - $skip;$ i=1
  - $if\,(b)\ then\ c\ else\ skip;$ i=2
  - $if\,(b)\ then\ (c; if\,(b) then\ c\ else\ skip)\ else\ skip$ i=3
  - …

$$\textsc{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

最弱前条件计算：循环

- 因为存在量词求解困难，有时使用全程量词的更强版本
- $wp'(while\ b\ do\ c, Q) = \forall i \in Nat. L_i(Q)$
  - where
    - $L_0(Q) = false$
    - $L_{i+1}(Q) = (\neg b \Rightarrow Q) \wedge \Big(b \Rightarrow wp'\big(c, L_i(Q)\big)\Big)$

最强后条件计算

- $sp(P, skip) = P$
- $sp(P, x := a) = \exists n.\ x = a[n/x] \wedge P[n/x]$
- $sp(P, c_1; c_2) = sp(sp(P, c_1), c_2)$
- $sp(P, if\ b\ then\ c_1\ else\ c_2) = sp(b \wedge P, c_1) \vee sp(\neg b \wedge P, c_2)$
- $sp(P, while\ b\ do\ c) = \neg b \wedge \exists i.\ L_i(P)$
  - where
    - $L_0(P) = P$
    - $L_{i+1}(P) = sp(b \wedge L_i(P), c)$

因为约束更复杂，实际使用较少

符号执行和谓词转换

- 在没有循环的情况下，最弱前条件和符号执行等价
- 例：If (y > 0) x+=1; else x+=2; assert(x<3)
- 符号执行
  - 令x=a, y=b, 计算得到
    - $(b > 0 \wedge a + 1 < 3) \vee (\neg b > 0 \wedge a + 2 < 3)$
- 最弱前条件
  - $wp(x \mathrel{+}= 1, x < 3) = x + 1 < 3$
  - $wp(x \mathrel{+}= 2, x < 3) = x + 2 < 3$
  - $wp(\text{原程序}, x < 3) = (y > 0 \rightarrow x + 1 < 3) \wedge (\neg y > 0 \rightarrow$

动态符号执行

约束求解失败的情况

- 形成了复杂条件
  - $x^5 + 3x^3 == y$
  - p->next->value == x
- 调用了系统调用
  - If (file.read()==x)

- 动态符号执行
  - 混合程序的真实执行和符号执行
  - 在约束求解无法进行的时候，用真实值代替符号值
    - 如果真实值x=10，则$x^5 + 3x^3 == y$变为103000==y，可满足

动态符号执行

- 动态符号执行主要用于生成测试输入
- 代表性工作：
  - Concolic Testing, Koushik Sen
    - 主要工具：CUTE
  - Execution-Generated Testing, Cristian Cadar
    - 主要工具：KLEE

Concolic Testing Approach

```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution

Symbolic Execution

concrete state	symbolic state	path condition
x = 22, y = 7	$x = x_0, y = y_0$	

Concolic Testing Approach


```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);
    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution

Symbolic Execution

	concrete state	symbolic state	path condition
z = double (y);	x = 22, y = 7, z = 14	$x = x_0, y = y_0, z = 2*y_0$	

Concolic Testing Approach


```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

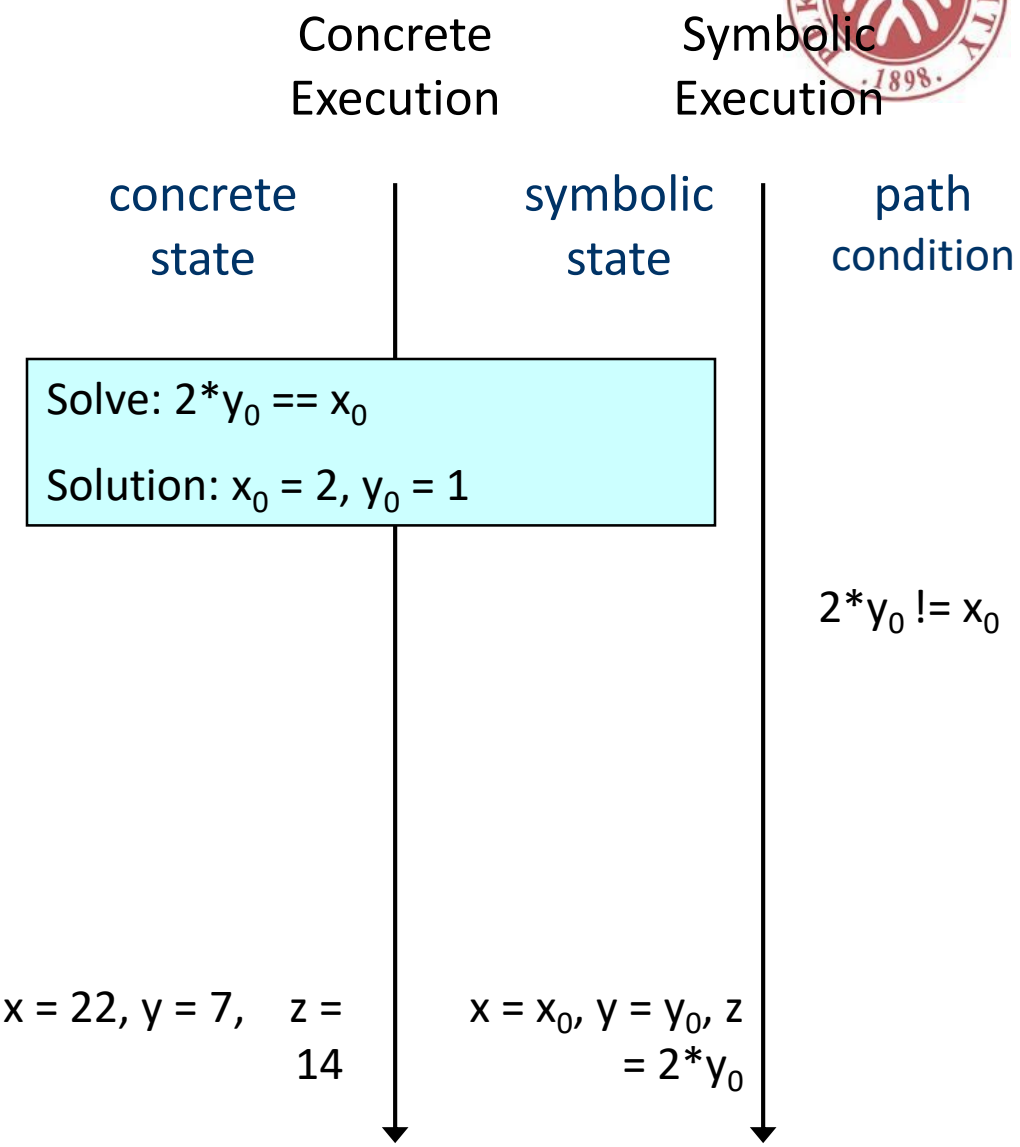
Concrete Execution

Symbolic Execution

	concrete state	symbolic state	path condition
			$2*y_0 != x_0$
	x = 22, y = 7, z = 14	$x = x_0, y = y_0, z = 2*y_0$	

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach


```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 2, y = 1	$x = x_0, y = y_0$	

Concolic Testing Approach

```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);
    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 2, y = 1, z = 2	x = x_0, y = y_0, z = 2*y_0	

Concolic Testing Approach


```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {
        if (x > y+10) {

            ERROR;
        }
    }

}
```

Concrete Execution

Symbolic Execution

concrete state	symbolic state	path condition
x = 2, y = 1, z = 2	x = x_0, y = y_0, z = $2*y_0$	$2*y_0 == x_0$

Concolic Testing Approach


```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

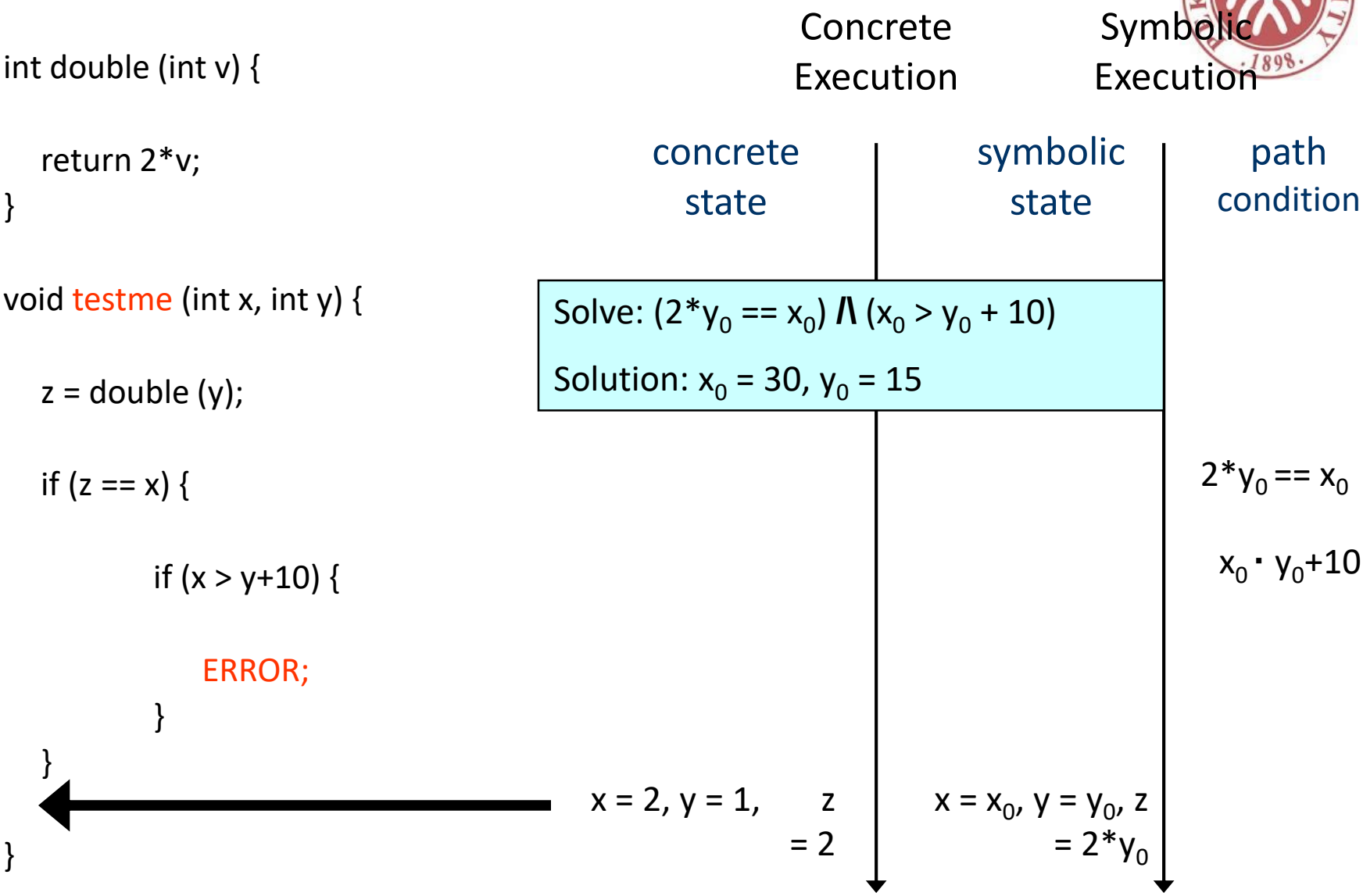
Concrete Execution

Symbolic Execution

concrete state	symbolic state	path condition
		$2*y_0 == x_0$
		$x_0 \cdot y_0+10$
x = 2, y = 1, z = 2	$x = x_0, y = y_0, z = 2*y_0$	

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach

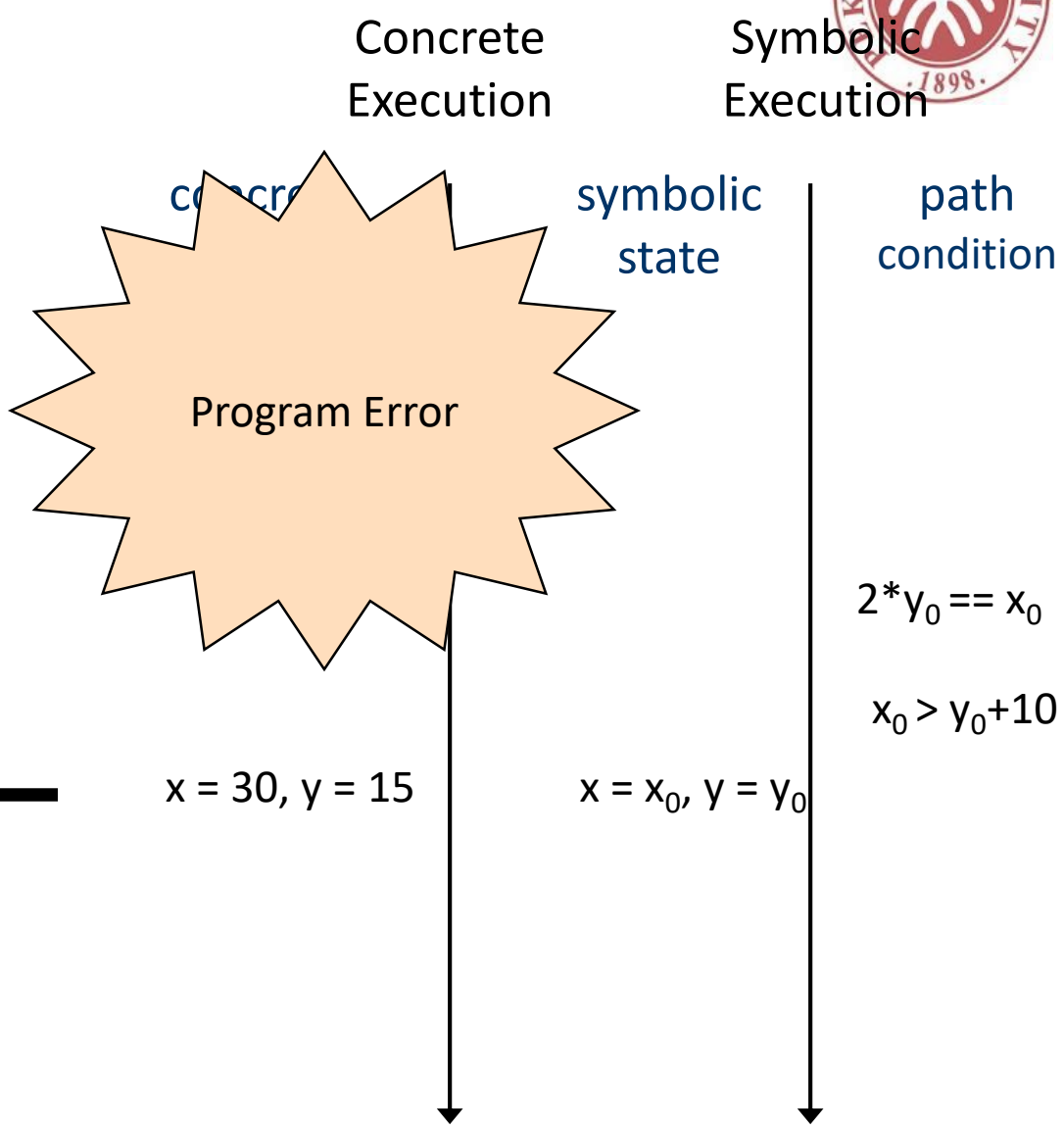

```
int double (int v) {

    return 2*v;
}

void testme (int x, int y) {

    z = double (y);

    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution

Symbolic Execution

concrete state	symbolic state	path condition
x = 30, y = 15	$x = x_0, y = y_0$	

Concolic Testing Approach

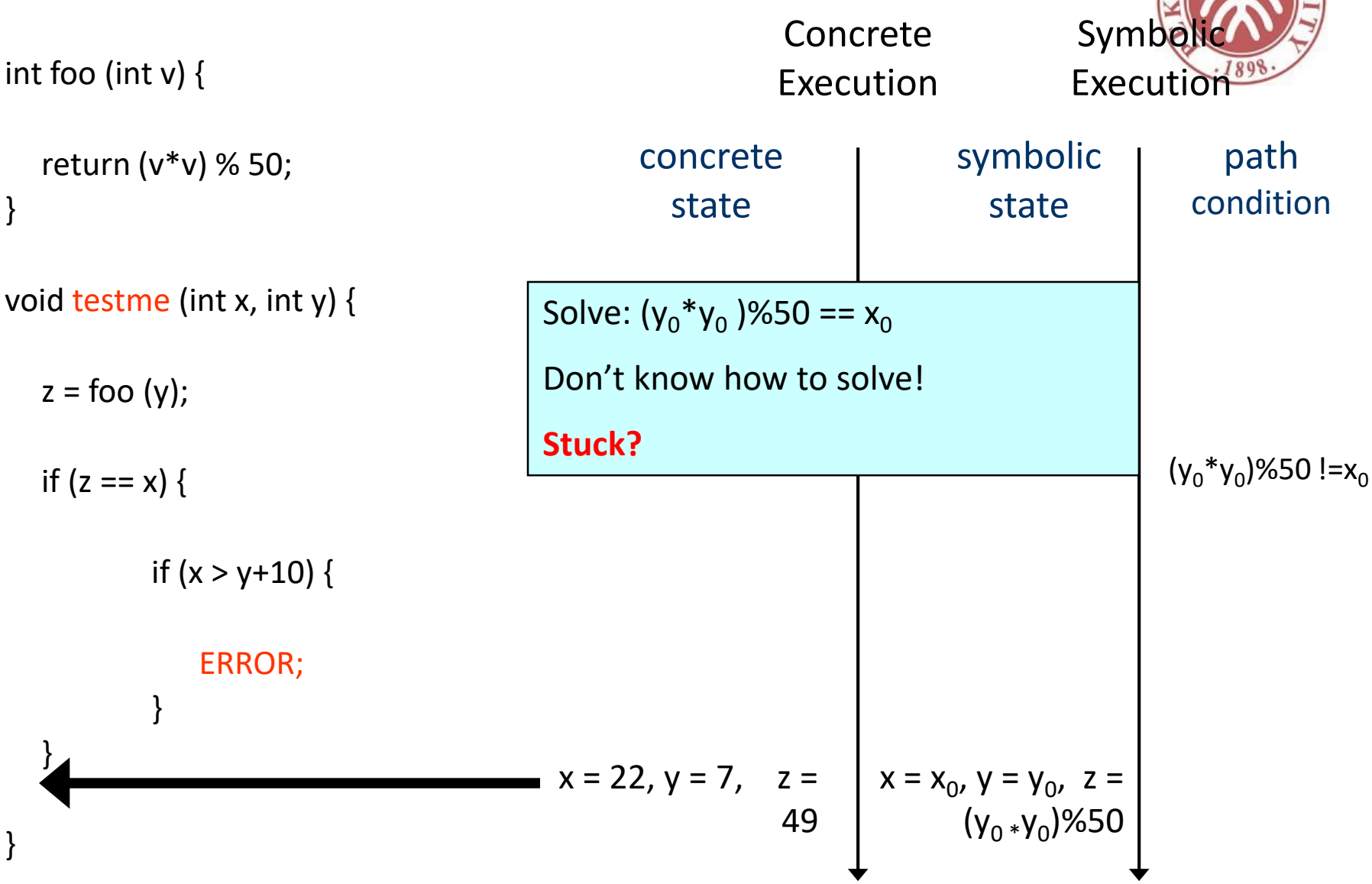


Novelty : Simultaneous Concrete and Symbolic Execution

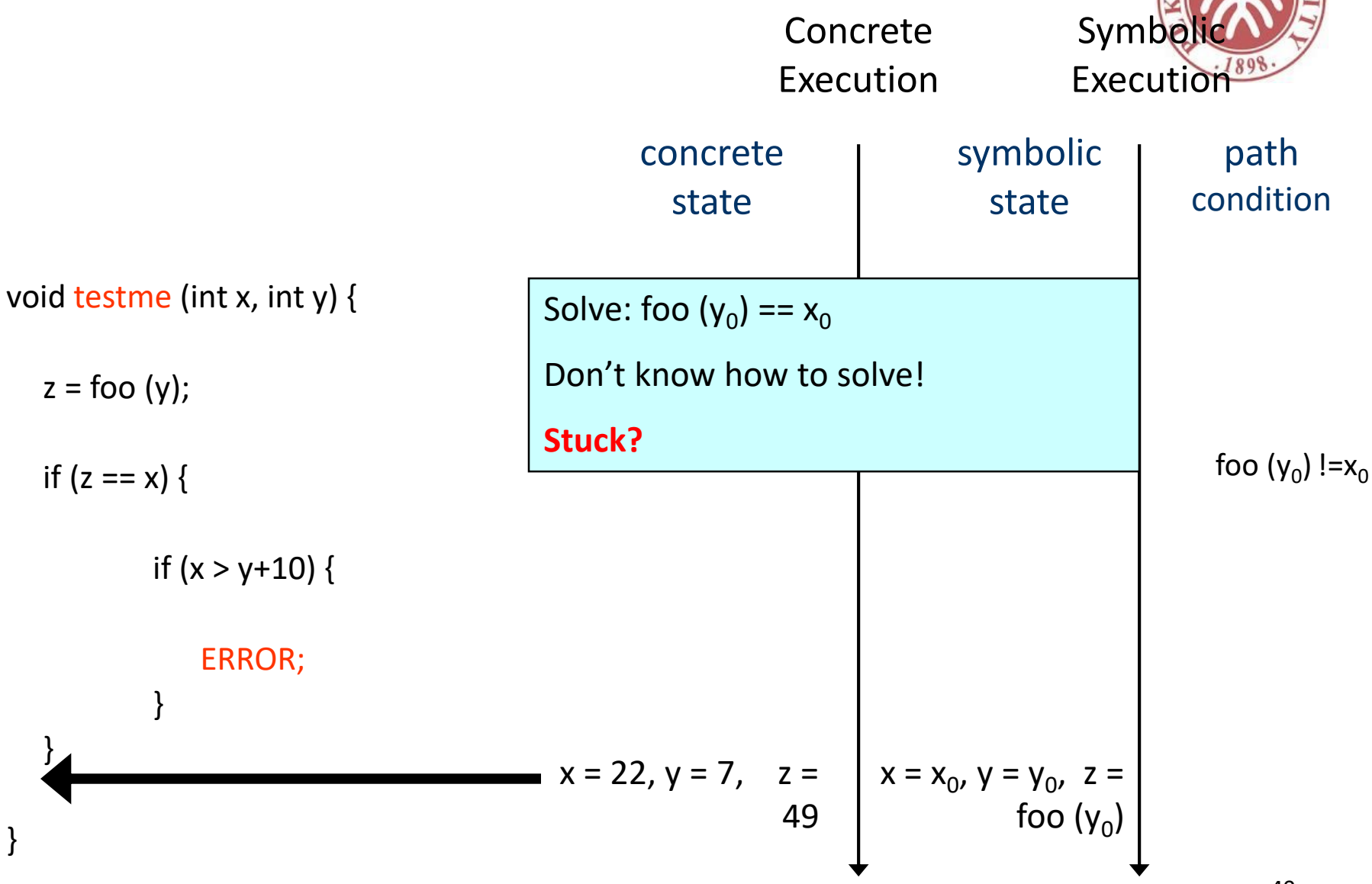

```
int foo (int v) {
    return (v*v) % 50;
}

void testme (int x, int y) {
    z = foo (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 22, y = 7	$x = x_0, y = y_0$	

Novelty : Simultaneous Concrete and Symbolic Execution



Novelty : Simultaneous Concrete and Symbolic Execution



Novelty : Simultaneous Concrete and Symbolic Execution

```
int foo (int v) {

    return (v*v) % 50;
}

void testme (int x, int y) {

    z = foo (y);

    if (z == x) {

            if (x > y+10) {

                ERROR;

            }
    }
}
```

Concrete Execution

Symbolic Execution

concrete state

symbolic state

path condition

Solve: $(y_0 * y_0) \%50 == x_0$

Don't know how to solve!

Not Stuck!

Use concrete state

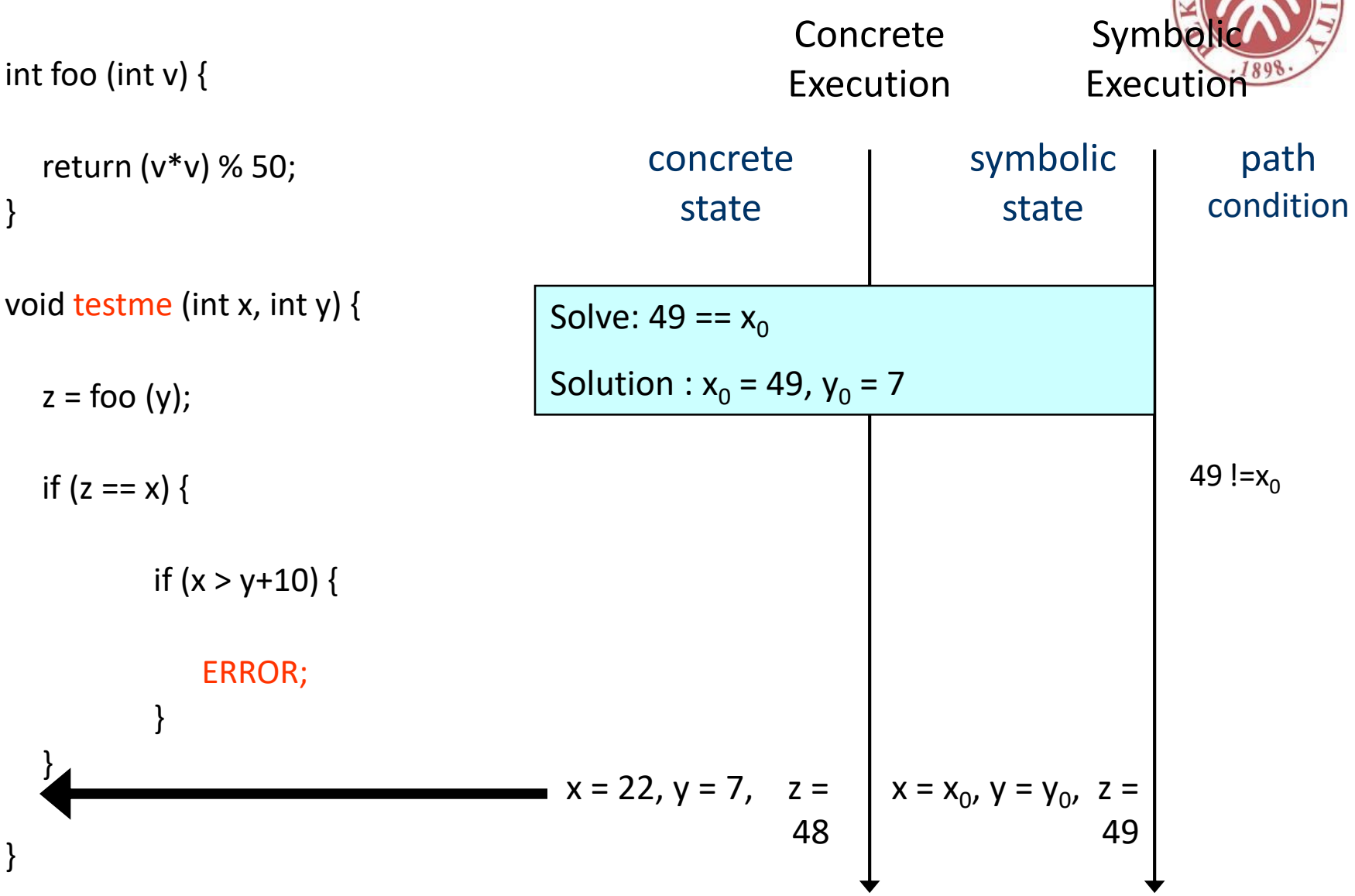
Replace y_0 by 7 (sound)

$(y_0 * y_0) \%50 != x_0$

x = 22, y = 7, z = 49

x = x_0, y = y_0, z = $(y_0 * y_0) \%50$

Novelty : Simultaneous Concrete and Symbolic Execution

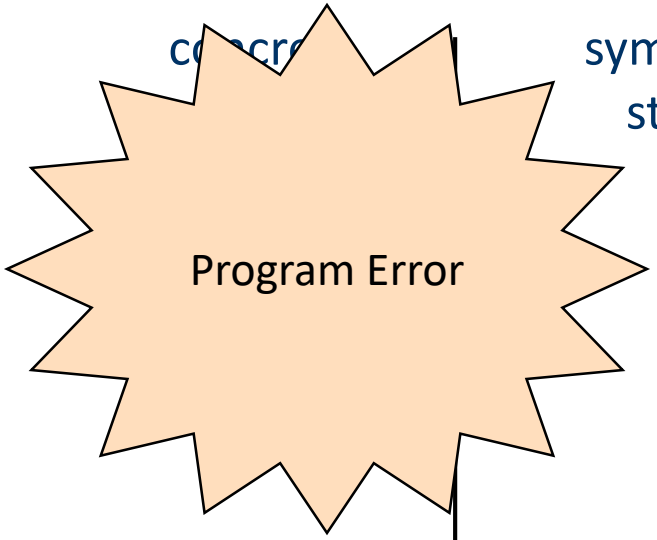


Novelty : Simultaneous Concrete and Symbolic Execution


```
int foo (int v) {

    return (v*v) % 50;
}

void testme (int x, int y) {

    z = foo (y);

    if (z == x) {

            if (x > y+10) {

                ERROR;
            }
    }

}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
x = 49, y = 7	$x = x_0, y = y_0$	

Novelty : Simultaneous Concrete and Symbolic Execution



```
int foo (int v) {
    return (v*v) % 50;
}

void testme (int x, int y) {
    z = foo (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution

Symbolic Execution

concrete state

symbolic state

path condition

Program Error

$2*y_0 == x_0$

$x_0 > y_0+10$

x = 49, y = 7, z = 49

$x = x_0, y = y_0, z = 49$

常见符号执行工具

- C语言：KLEE
- Java语言：SymbolicPathFinder

参考资料

- 《程序设计语言的形式语义》Glynn Winskel著
- Symbolic execution for software testing: three decades later. C Cadar, K Sen. Communication of ACM. 2013