软件分析与测试
符号执行技术

严俊

中国科学院大学
University of Chinese Academy of Sciences
ISCAS 中国科学院软件研究所
Institute of Software, Chinese Academy of Sciences

本次课内容

- 符号执行的技术路线及技术难点
  - 静态符号执行技术
  - 采用静态符号执行技术的测试数据生成技术
  - 动态（混合）符号执行
- 采用符号执行的静态分析技术

测试用例生成

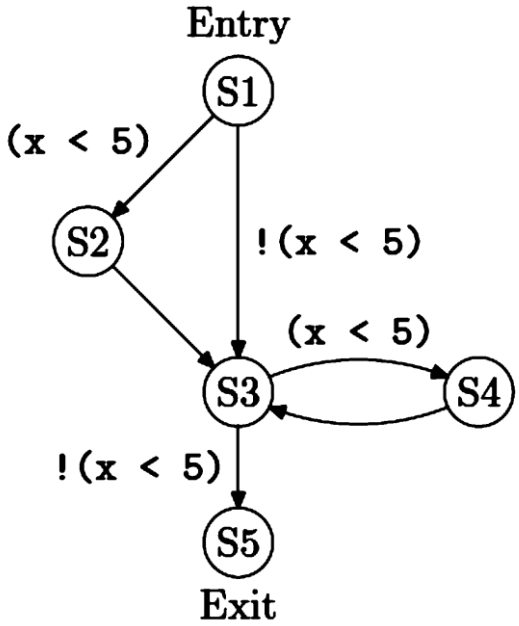
- Test data/case generation: How can we generate a test suite to achieve 100% statement/branch/… coverage? 测试生成

- Does high coverage indicate better fault detection capability? 发现错误的能力

测试路径

- **CFG上的路径**
  - 完整路径：***A path that starts at an initial node and ends at a final node***
  - 例：**S1-S2-S3-S4-S3-S5**
- **不可行路径**
  - **S1-S3-S4-S3-S5**

```
void f(int x）{
/*S1*/ if (x < 5）
/*S2*/ y = 2;
/*S3*/ while (x < 5）
/*S4*/ x++;
/*S5*/ return;
}
```



测试路径

- 是控制流图中的一个抽象概念
- 每条路径对应程序的一个等价类
- 在路径的基础上定义了覆盖率
- 从路径可以分析出测试数据

基于路径的测试生成过程

```
TS = EmptySet;
  do {
      生成一条对覆盖率有贡献的测试路径p;
      if (可从p产生测试数据t) {
            将t加入测试集TS;
            根据p统计覆盖率;
      }
  } while (覆盖率不达标);
  return TS;
```

简单的程序

```
int  i, j;
bool good;
if ((i > 2) && (j > 3))
{
     j = j-1;
     if ( i+2j < 5 )
        good = FALSE;
     else  good = TRUE;
}
```

- **Path 1: (infeasible)**

```
((i > 2) && (j > 3))
j = j-1;
(i+2j < 5)
good = FALSE;
```

- **Path 2: (feasible, executable)**

```
((i > 2) && (j > 3))
j = j-1;
! (i+2j < 5)
good = TRUE;
```

从底层看程序

- 带条件执行
  - If (cond) then action(s)
  - While (cond) action(s)
- 条件、前后断言

  if (expr) …
- 动作（赋值语句）

  $var_i := f(vars)$

  …

while语言

- Stmt ::= Stmt; Stmt
  | while(Expr) Stmt
  | if (Expr) Stmt else Stmt
  | Var = Expr
  | skip

Hoare Logic

- 霍尔三元组 {P}C{Q}
  - P前条件, C语句, Q后条件
  - 表示三者之间的推导关系
  - 又称公理语义

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

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

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

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

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

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

条件和动作交替的路径（冒泡排序）

```
 i = n-1;
@ i > 0;
 indx = 0;
 j = 0;
@ j < i;
@ a[j+1] < a[j];
 temp = a[j];
 a[j] = a[j+1];
 a[j+1] = temp;
 indx = j;    j = j+1;
@ j >= i;
 i = indx;
@ i <= 0;
```

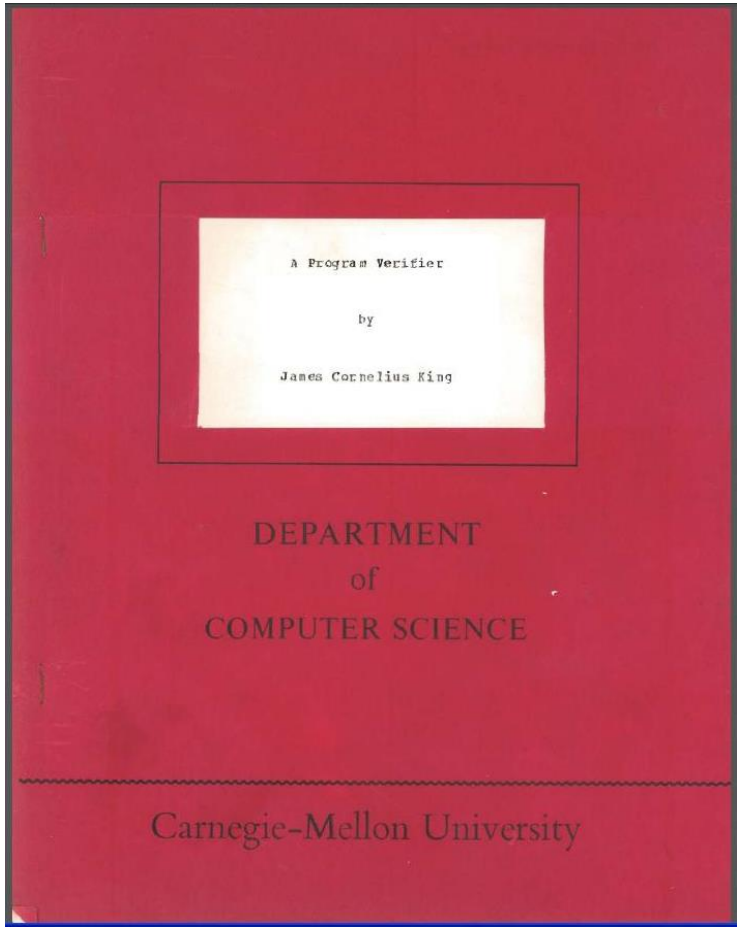
符号执行（Symbolic Execution）

用符号（而不是具体的数值）作为输入来“执行”程序。

Example. After the assignment $x = a + b$, the variable x gets the symbolic value a_0+b_0, rather than some concrete value such as 3+5=8.

[Boyer et al. 1975] [King 1976] [Clarke 1976]

J. C. King 1969



A Program Verifier

by

James Cornelius King

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

September 1969

Submitted to the Carnegie-Mellon University
In partial fulfillment of the requirements
for the degree of Doctor of Philosophy

A path in bubble-sort

```
 i = n-1;
@ i > 0;
 indx = 0;
 j = 0;
@ j < i;
@ a[j+1] < a[j];
 temp = a[j];
 a[j] = a[j+1];
 a[j+1] = temp;
 indx = j;    j = j+1;
@ j >= i;
 i = indx;
@ i <= 0;
```

- **Path condition ?**

Under what condition does the program execute along this path ?

A path in bubble-sort

```
 i = n-1;
@ i > 0;
 indx = 0;
 j = 0;
@ j < i;
@ a[j+1] < a[j];
 temp = a[j];
 a[j] = a[j+1];
 a[j+1] = temp;
 indx = j;    j = j+1;
@ j >= i;
 i = indx;
@ i <= 0;
```

- **Symbolic execution**

variables' values;

constraints.

```
 i: n0 - 1;
n0 - 1 > 0;
 indx: 0;
 …
```

前向符号执行

语句	执行后
`i = n-1;`	$i=n_0-1$

前向符号执行

语句	执行后
`i = n-1;`	$i=n_0-1$
`@ i > 0;`	$i=n_0-1$ $n_0-1>0$

前向符号执行

语句	执行后
`i = n-1;`	$i=n_0-1$
`@ i > 0;`	$i=n_0-1$ $n_0-1>0$
`indx = 0;`	$i=n_0-1$, `indx=0` $n_0-1>0$
`j = 0;`	$i=n_0-1$, `indx=0, j=0` $n_0-1>0$

前向符号执行

语句	执行后
…	…
`indx = 0;`	$i=n_0-1$, indx=0 $n_0-1>0$
`j = 0;`	$i=n_0-1$, indx=0, j=0 $n_0-1>0$
`@ j < i;`	$i=n_0-1$, indx=0, j=0 $n_0-1>0$, ~~$0<n_0-1$~~
`@ a[j+1] < a[j];`	$i=n_0-1$, indx=0, j=0, a[0]=a_0, a[1]=a_1 $n_0-1>0$, $a_1<a_0$

前向符号执行

语句	执行后
…	…
`@ j < i;`	$i=n_0-1$, indx=0, j=0 $n_0-1>0$,
`@ a[j+1] < a[j];`	$i=n_0-1$, indx=0, j=0, a[0]=a_0, a[1]=a_1 $n_0-1>0$, $a_1<a_0$
`temp = a[j];`	$i=n_0-1$, indx=0, j=0, a[0]=a_0, a[1]=a_1, temp=a_0 $n_0-1>0$, $a_1<a_0$
`a[j] = a[j+1];`	$i=n_0-1$, indx=0, j=0, a[0]=a_1, a[1]=a_1, temp=a_0 $n_0-1>0$, $a_1<a_0$
`a[j+1] = temp;`	$i=n_0-1$, indx=0, j=0, a[0]=a_1, a[1]=a_0, temp=a_0 $n_0-1>0$, $a_1<a_0$
`indx = j; j = j+1;`	$i=n_0-1$, indx=0, j=1, a[0]=a_1, a[1]=a_0, temp=a_0 $n_0-1>0$, $a_1<a_0$,

前向符号执行

语句	执行后
…	…
temp = a[j];	$i=n_0-1$, indx=0, j=0, $a[0]=a_0$, $a[1]=a_1$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$
a[j] = a[j+1];	$i=n_0-1$, indx=0, j=0, $a[0]=a_1$, $a[1]=a_1$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$
a[j+1] = temp;	$i=n_0-1$, indx=0, j=0, $a[0]=a_1$, $a[1]=a_0$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$
indx = j; j = j+1;	$i=n_0-1$, indx=0, j=1, $a[0]=a_1$, $a[1]=a_0$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$
@ j >= i;	$i=n_0-1$, indx=0, j=1, $a[0]=a_1$, $a[1]=a_0$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$, $1>=n_0-1$

前向符号执行

语句	执行后
…	…
@ j >= i;	$i=n_0-1$, indx=0, j=1, $a[0]=a_1$, $a[1]=a_0$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$, $1>=n_0-1$
i = indx;	i=0, indx=0, j=1, $a[0]=a_1$, $a[1]=a_0$, $temp=a_0$ $n_0-1>0$, $a_1<a_0$, $1>=n_0-1$

前向符号执行

语句	执行后
…	…
i = indx;	i=0, indx=0, j=1, a[0]=a_1, a[1]=a_0, temp=a_0 n_0-1>0, a_1<a_0, 1>=n_0-1
@ i <= 0;	i=0, indx=0, j=1, a[0]=a_1, a[1]=a_0, temp=a_0 n_0-1>0, a_1<a_0, 1>=n_0-1, ~~0<=0~~

A path in bubble-sort

```
 i = n-1;
@ i > 0;
 indx = 0;
 j = 0;
@ j < i;
@ a[j+1] < a[j];
 temp = a[j];
 a[j] = a[j+1];
 a[j+1] = temp;
 indx = j;    j = j+1;
@ j >= i;
 i = indx;
@ i <= 0;
```

- **Path condition:**

  `n-1 > 0`

  `a[1] < a[0]`

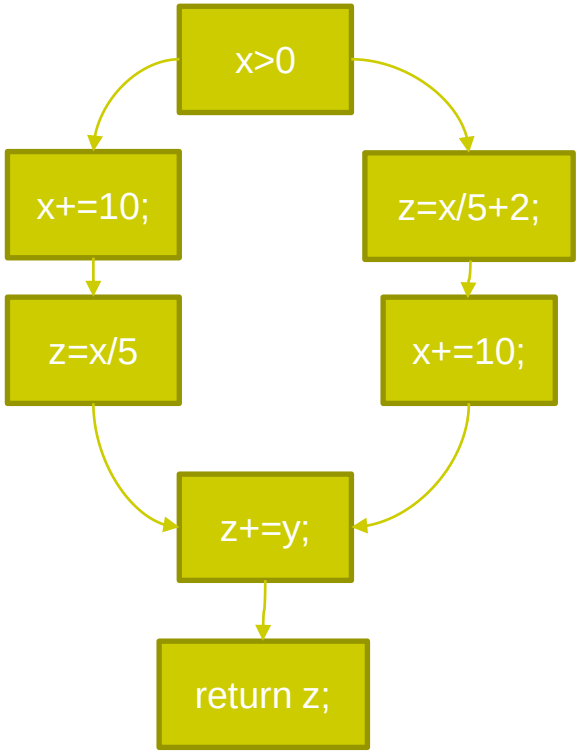
  `n-1 <= 1`

具体输入：n = 2

a: { 3, 2 } 或 { 2, 1} …

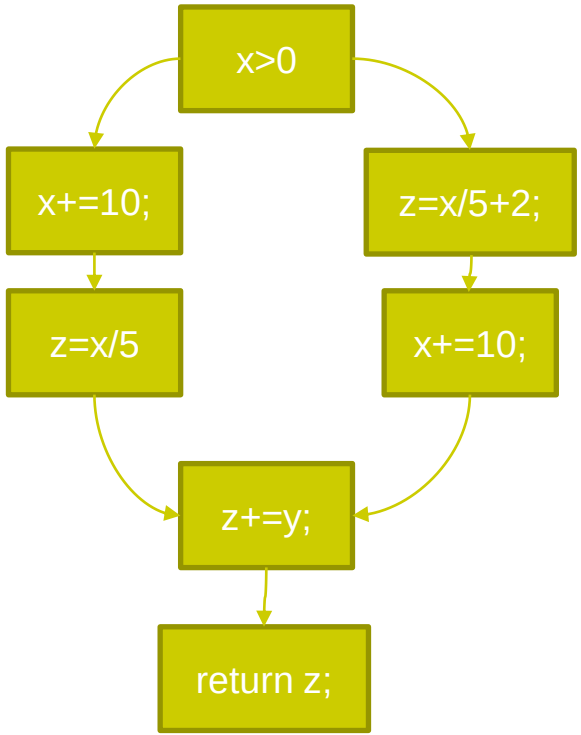
符号执行 – another example

```
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 – another example

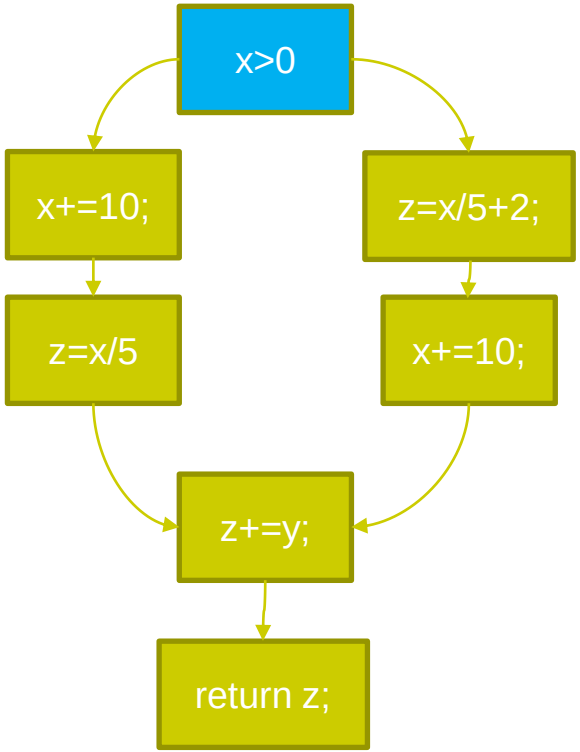
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a
y=b
z=?

符号执行 – another example

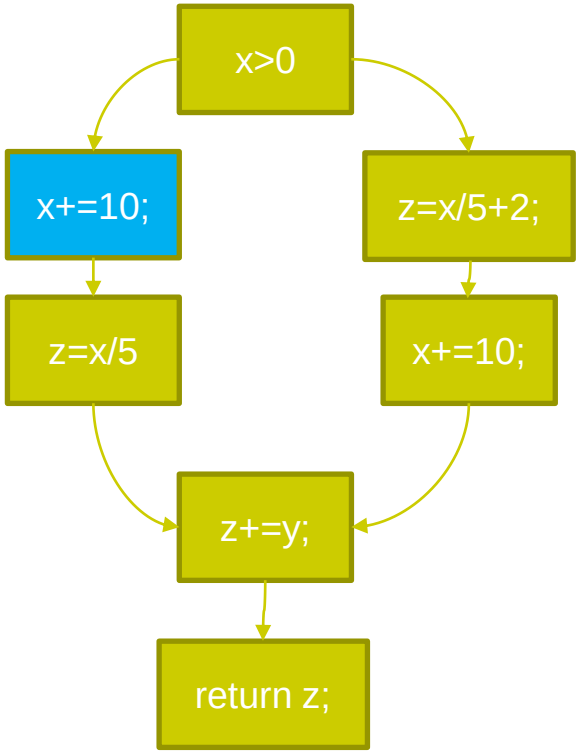
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



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

符号执行 – another example

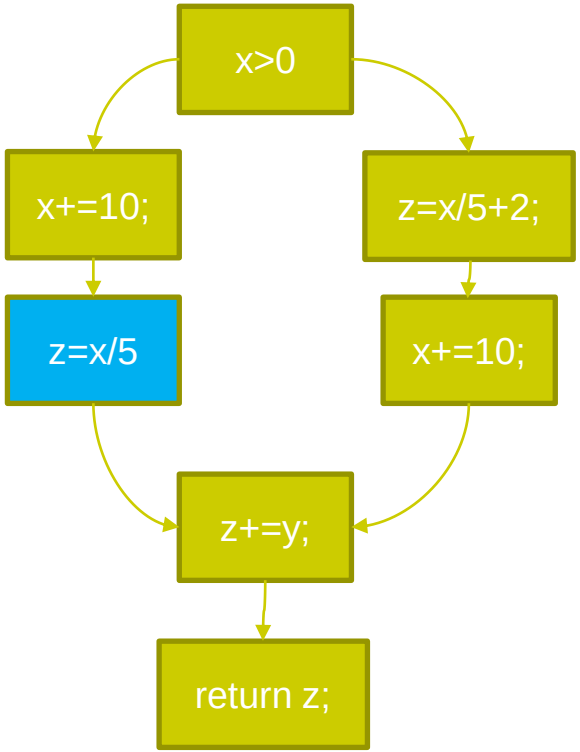
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



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

符号执行 – another example

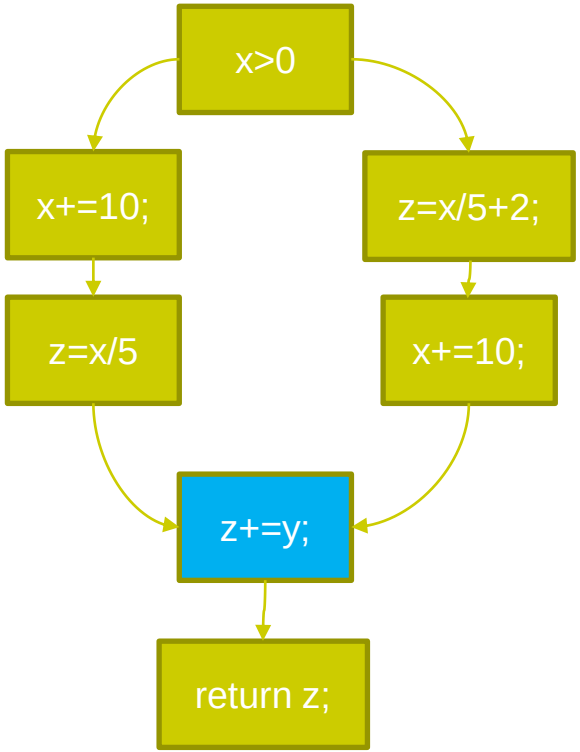
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



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

符号执行 – another example

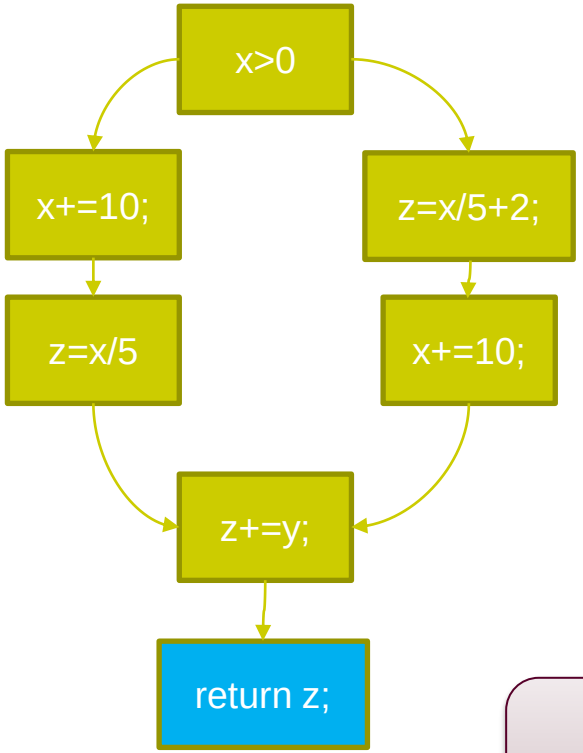
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



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

符号执行 – another example

```
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

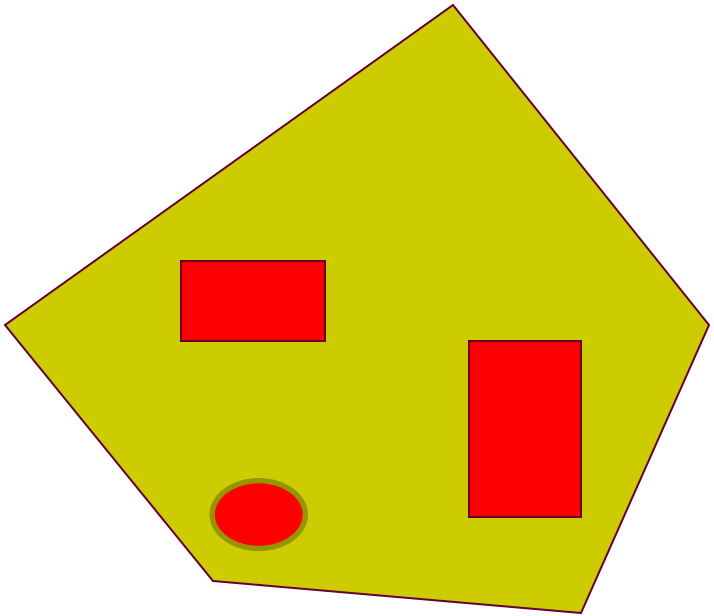
另一种计算方法—Backward Substitution

语句	替换后
i = n-1;	1>=n-1, a[1]<a[0], 0<n-1
@ i > 0;	1>=i, a[1]<a[0], 0<i, ~~i>0~~
indx = 0;	1>=i, a[1]<a[0], 0<i
j = 0;	~~0<=0~~, 1>=i, a[1]<a[0], 0<i
@ j < i;	j<=0, j+1>=i, a[j+1]<a[j], j<i
@ a[j+1] < a[j];	j<=0, j+1>=i, a[j+1]<a[j]
temp = a[j];	j<=0, j+1>=i
a[j] = a[j+1];	j<=0, j+1>=i
a[j+1] = temp;	j<=0, j+1>=i
indx = j;	j<=0, j+1>=i
j = j+1;	indx<=0, j+1>=i
@ j >= i;	indx<=0, j>=i
i = indx;	indx<=0
@ i <= 0;	i <= 0

ISCAS

具体执行 vs 符号执行

- 检查输入空间中的点 vs区域



Satisfiability Modulo Theories (SMT)

- solvers – CVC3/CVC4, Yices, Z3, Boolector …
- x3, x2, x1, x0: INT;

```
CHECKSAT (x0 >= 0 AND x0 <= 9 AND x1 >= 0
 AND x1 <= 9 AND x2 >= 0 AND x2 <= 9 AND x3
 >= 0 AND x3 <= 9 AND (x0 > 0 OR x1 > 0 OR x2 >
 0 OR x3 > 0) AND 1000*x0 + 100*x3 + 10*x2 + x1
 = 2000*x3 + 200*x2 + 20*x1 + 2*x0);
```

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli:
Solving SAT and SAT Modulo Theories: From an abstract Davis--Putnam--Logemann--Loveland procedure to DPLL(T). J. ACM 53(6): 937-977 (2006)

基于符号执行与约束求解的路径分析

Constraint Solving + Symbolic Execution

- verify – or find bugs in – certain programs
- check the error messages produced by other static analyzers, to eliminate some false alarms
- automate an important part of unit testing, i.e., generating test cases (input data) for the program
- generate test cases for black-box testing or model-based testing, if a proper specification (like EFSM) is provided.

Jian Zhang: **Constraint Solving and Symbolic Execution.** VSTTE 2005: 539-544

1. C程序单元测试数据生成

基于路径的测试生成

- 每条可行测试路径对应着一个等价类/测试用例

- 一般性思路
  - 对C代码进行词法分析和语法分析
    生成AST，并在其之上构造CFG
  - 生成程序的路径
  - 对这些路径进行符号执行并求解路径条件
  - 根据路径覆盖要求选择一组可行的路径

难点

- 在有非线性运算，如指针、数组、结构和变量是符号值情况下对程序操作语义的模拟；
- 循环不变式无法自动得到，无法穷举状态空间；
- 对函数调用的效果的模拟。

数组：符号化的下标表达式

- "For example, suppose the current constraint is `a[i]+a[j] > 3`, and the assignment is `a[k]=c`. “
- 新的约束条件？

```
    ((k == i) && (k == j) && (2c > 3))
|| ((k == i) && (k != j) && (c+a[j] > 3))
|| ((k != i) && (k == j) && (a[i]+c > 3))
|| ((k != i) && (k != j) && (a[i]+a[j] > 3))
```

一个关于数组的例子

```
int i, j;
int a[4];
{
 @(i >= 0); @(j >= 0);
  a[i] = j;
 @(i + j < 2);
 @(a[1] > 4);
 @(a[0] > 8);
}
```

数组下标变量

```
int i, j;
int a[4];
{
 @(i >= 0); @(j >= 0);
  a[i] = j;
 @(i + j < 2);
 @(a[1] > 4);
 @(a[0] > 8);
}
```

- case 1:

```
i==0:

  a[0] = j;
@(j < 2);
@(a[1] > 4);
@(a[0] > 8);
  → @(j > 8);
```

数组下标变量

```
int i, j;
int a[4];
{
 @(i >= 0); @(j >= 0);
  a[i] = j;
 @(i + j < 2);
 @(a[1] > 4);
 @(a[0] > 8);
}
```

- case 2:

```
i==1:

 a[1] = j;
@(1+j < 2);
  → @(j < 1);
@(a[1] > 4);
  → @(j > 4);
@(a[0] > 8);
```

数组下标变量

- **case 3:**

```
i==2:

 a[2] = j;
@(2 + j < 2);
  → @(j < 0);
@(a[1] > 4);
@(a[0] > 8);
```

- **case 4:**

```
i==3:

 a[3] = j;
@(3 + j < 2);
  → @(j < -1);
@(a[1] > 4);
@(a[0] > 8);
```

更为复杂的例子

```
1  char b64[256] = {−1, −1, −1, ... 62, −1, −1, −1, 63, 52, 53, 54, 55,
       56, 57, 58, 59, 60, 61, −1, −1, −1, −1, ..., −1 };
2
3  unsigned isBase64(unsigned char k) {
4    if (b64[k] >= 0)
5      return 1;
6    else
7      return 0;
8  }
```

Figure 1: A simplified excerpt from the *base64* decoding routine in Coreutils. Array b64 has positive values at offsets 43, 47–57, 65–90, and 97–122.

$$(b64_0 = -1) \wedge (b64_1 = -1) \wedge \cdots \wedge (b64_{255} = -1) \wedge$$
$$(b64_k \geq 0) \wedge$$
$$(k = 0 \rightarrow b64_k = b64_0) \wedge \ldots \wedge (k = 255 \rightarrow b64_k = b64_{255}))$$

D M Perry, *et al.*: Accelerating array constraints in symbolic execution. ISSTA 2017: 68-78

Pointer and Structure Variables

- Essential constructs in many prog. langs.
- May cause various kinds of errors.
- Not/rarely considered in previous works on symbolic execution (or automated test data generation in general).

- [Zhang, QSIC 2004]

指针

A.J. Offutt, Z. Jin and J. Pan, The dynamic domain reduction procedure for test data generation, Software – Practice and Experience, 29(2): 167–193, 1999.

- Their tool "has only been applied to numeric software", because it "does not handle pointers and the expression handling is limited to expressions that use numeric operators".

一个简单的例子

```
void f(int x)
{
   int y;
   y = x + 3;
}
```

简单符号执行之后

$y = x_0 + 3;$

稍微复杂的例子

```
void f(char *s)
{
  char *p;
  p = s;
  while (*p != 'a')
    p++;
}
```

一条路径翻译结果：

p = 1;

@ mem[p] != 'a';

p++;

@ mem[p] == 'a';

1. mem是引入的用来模拟内存的辅助数组。
2. 语句“p = s”被翻译成“p = 1”，这里1是数组s的模拟起始地址
3. 计算出一组解为s[1] = 'b'， s[2] = 'a'。

更为复杂的例子

```
void f(void)
{
  int a;
  int *p;
  int **q;
  a = 1000;
  p = &a;
  q = &p;
  **q = 1000;
  p--;
}
```

采用一个大数组模拟指针

mem[1] = 1000;

mem[2] = 1;

mem[3] = 2;

mem[mem[mem[3]]] = 1000;

mem[2] = mem[2] - 1;

符号指针与具体指针

- 一个具体指针指向一个已知大小的内存区域。从而它的模拟地址就是已知的，在生成的路径中，它被替换成它的模拟地址。
- 一个符号指针代表一个未知的地址，这个地址在程序中随着程序的执行会改变，也会被路径条件约束。它以一个未知量的形式出现在路径中

```
void mem_free(char *buf, char
*p)
{
  ...
  position = (p - buf) / 16;
  ...
}
```

翻译成

position = (p - 1) / 16;

buf是一个具体指针，指向一个内存区域的起始地址。应该给buf分配一块内存。

根据代码可以推断出p指向的不是一块新的内存区域，而是buf中的某个位置。p的值应该根据路径约束求解出来。

Search strategy [Zhang-Xu-Wang 2004]

- It should be noted that a naive depth-first search procedure such as the algorithm in Fig. 2 may not terminate even if the target program terminates on any input data. Consider the gcd program in Example 1.
- Suppose the initial values are: m = 2, n = 2k (k > 1).
- Then the program will take the loop t3-t4 for (k-1) times. In other words, if the test data generation algorithm keeps trying this loop, it will always be able to find suitable values for the variables.

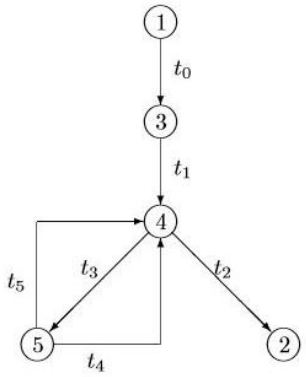


Figure 1. EFSM for the GCD Program

transition	predicate	action
t_0		`scanf("%d",&m);` `scanf("%d",&n);`
t_1	`(m>1)&&(n>1)`	`x = m; y = n;`
t_2	`(x==y)`	`gcd = x;`
t_3	`(x!=y)`	
t_4	`(x<=y)`	`y = y-x;`
t_5	`(x>y)`	`x = x-y;`

Table 1. Definitions of the transitions in Fig. 1

无穷长的可行路径？

```
@((m>1) && (n>1));
  x = m;  y = n;
@(x != y);

@(x <= y);
  y = y-x;

@(x <= y);
  y = y-x;
```

- **Path condition:**

```
m > 1
n >  1
m != n
m <= n
m <= n-m
...
```

函数调用

跨过程分析 – 难！　if (f(x1) < g(x2)) …

- Inlining
- Modeling

(1) 原始语句 c = getchar();

替换为 c = INPUT; @ 0 <= c < 256;

(2) random() { return int; }

(3) strcat() { return arg1; }

更多需要考虑的问题

- 函数指针
- 函数副作用
- 使用递归数据结构的程序，不能生成测试数据，比如链表、树等。
- 递归函数调用

- 解决方案
  - 更为精确的函数行为建模，每个对象对应一个独立的区域
  - 更为复杂的约束处理策略
  - ……

1.1面向分支/语句覆盖的路径生成策略

基于路径的测试生成过程

```
TS = EmptySet;
  do {
    生成一条对覆盖率有贡献的测试路径p;
    if (可从p产生测试数据t) {
        将t加入测试集TS;
        根据p统计覆盖率;
    }
  } while (覆盖率不达标);
return TS;
```

- 深度优先搜索
  - 给循环限定次数的方式来避免陷入一个长的循环
- 宽度优先搜索
  - 限制生成路径的最大长度
- 最优测试路径选择方法
  - 0-1整数规划
  - $|TS_{SC}| \le |TS_{BC}| \le |TS_{BP}| \le 10$

最优测试选择

- **给定一组测试用例 *P*，选择其（最小）子集*Ps*，达到高覆盖度。**
- **(branch coverage)**

H. S. Wang, S. R. Hsu, and J. C. Lin. A generalized optimal path-selection model for structural program testing. Journal of Systems and Software, 10: 55-63, 1989.

路径的覆盖度矩阵

Branch

Path	Br_1	Br_2	…	Br_j	…
P_1	b_{11}	b_{12}	…	b_{1j}	…
P_2	b_{21}	b_{22}	…	b_{2j}	…
…					
P_i	b_{i1}	b_{i2}	…	b_{ij}	…
…					

b_{ij}: coverage frequency for path P_i over branch Br_j.

建模为优化问题

- Bool X_i = ITE(P_i is selected, 1, 0)
- 优化问题

  $min. \sum_i X_i$

  $s.t. \ \sum_i X_i b_{ij} \geq 1$ for all j

  （对每条边 Br_j）
- 整数规划/伪布尔优化

Branch Path Matrix

Figure 4b. NP matrix.

5.

Path No.	branch No. / branch	1 a	2 b	3 c	4 d	5 e	6 f	7 g	8 h	9 i	10 j	11 k	12 l	13 m	14 n	15 o	16 p	17 q	18 r	19 s	20 t	P_BN
1>	ac	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
2>	bdedfghjkmprti	0	1	0	2	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	14
3>	bdedfghjkmqsti	0	1	0	2	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	14
4>	bdedfghjlnkmprti	0	1	0	2	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	16
5>	bdedfghjlnkmqsti	0	1	0	2	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	16
6>	bdedfghjlnloprti	0	1	0	2	1	1	1	1	1	1	0	2	0	1	1	1	0	1	0	1	16
7>	bdedfghjlnloqsti	0	1	0	2	1	1	1	1	1	1	0	2	0	1	1	0	1	0	1	1	16
8>	bdedfghjloprti	0	1	0	2	1	1	1	1	1	1	0	1	0	0	1	1	0	1	0	1	14
9>	bdedfghjloqsti	0	1	0	2	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	1	14
10>	bdedfgi	0	1	0	2	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	7
11>	bdfghjkmprti	0	1	0	1	0	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	12
12>	bdfghjkmqsti	0	1	0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	12
13>	bdfghjlnkmprti	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	14
14>	bdfghjlnkmqsti	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	14
15>	bdfghjlnloprti	0	1	0	1	0	1	1	1	1	1	0	2	0	1	1	1	0	1	0	1	14
16>	bdfghjlnloqsti	0	1	0	1	0	1	1	1	1	1	0	2	0	1	1	0	1	0	1	1	14
17>	bdfghjloprti	0	1	0	1	0	1	1	1	1	1	0	1	0	0	1	1	0	1	0	1	12
18>	bdfghjloqsti	0	1	0	1	0	1	1	1	1	1	0	1	0	0	1	0	1	0	1	1	12
19>	bdfgi	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	5

BRANCH PATH MATRIX

SCAS

```
   *** STATEMENT-TESTING test path recommendation ***

 testpath set  1 , the min. path number=3
       path no.  1    12F
       path no.  4    134345678A89BCE6F
       path no.  9    134345678ABDE6F


   *** BRANCH-TESTING test path recommendation ***

 testpath set  1 , the min. path number=3
       path no.  1    ac
       path no.  4    bdedfghjlnkmprti
       path no.  7    bdedfghjlnloqsti
```

Test Data Generation Examples: GNU coreutils

coreutils

Standard GNU file utilities (chmod, cp, dd, dir, ls, . . .), text utilities (sort, tr, head, wc, . . .), and shell utilities (whoami, who, . . .).

- **remove_suffix() in basename.c**
- **cat() in cat.c**
- **cut_bytes() in cut.c**
- **parse_line() in dircolors.c**
- **set_prefix() in fmt.c**
- **attach() in ls.c**
- **bsd_split_3() and hex_digit() in md5sum.c**

GNU coreutils 中 的remove_suffix()

```
void remove_suffix(char *name,
const char *suffix)
{
  ...
  np = name + strlen(name);
  sp = suffix + strlen(suffix);
  while (np > name && sp > suffix)
    if (*--np != *--sp) return;
  if (np > name) *np = '\0';
  ...
}
```

- 代码有非常多的指针运算和解引用。指针引起的别名关系也存在。
- 生成了5组测试数据，尽可能覆盖了所有的分支。

GNU coreutils 中 的strtol()

```
int strtol(const STRING_TYPE *nptr,
  STRING_TYPE **endptr,
  int base,
  int group LOCALE_PARAM_PROTO)
{
  ...
  save = s = nptr;
  while (ISSPACE (*s))
    ++s;
  if (*s == L_('-')) {
    negative = 1;
    ++s;
  }
  ...
}
```

- 路径最大长度为20时，生成了10组测试数据完成对CFG的覆盖。

InsertionSort()

```
void sort(int a[], int n)
{
  int current, j, lindex, temp;
  for (current = 0; current < n-1;
  current++) {
    lindex = current;
    for(j = current + 1; j < n; j++)
  {
        if (a[j] < a[lindex]) {
            lindex = j;
        }
    }
    if (lindex != current) {
        temp = a[current];
        a[current] = a[lowestindex];
        a[lowestindex] = temp;
    }
  }
  return;
}
```

- 路径长度限制为30
- 生成9个测试用例
  - {},
  - {0,0}, {1,0},
  - {1,0,1}, {1,1,0}, {0,1,0}, {2,1,0}, {1,0,0}, {1,2,0}

Example. GNU make: dir.c

```
char *dosify(char filename[20]) {
…
for (i = 0; *filename != '\0' && i < 8 &&…; i++) {
  *df = *filename;  df++;  filename++;
}
if (*filename != '\0') {
  *df = *filename;  df++;  filename++;
for (i = 0; *filename != '\0' && i < 3 &&…; i++) {
  *df = *filename;  df++;  filename++;
} }
```

dir.c 测试数据

- 自动产生420 条路径，其中 50 条可行
- 从中选择3条路径达到分支覆盖
- 一个具体的测试数据

  Filename[0] = 47
  Filename[1] = 47
  Filename[2] = 47
  Filename[3] = 46
  Filename[4] = 0

getop()

```
int getop(char *s, int lim)
{
  c = getchar();
  while (c == ' ' || c == '\t' || c == '\n')
  c = getchar();
  if (c != '.' && (c < '0' || c > '9'))
    return(c);
  ...
  for(i = 1; c >= '0' && c <= '9'; i++) {...}
  if (c == '.') {
    if (i < lim) s[i] = c;
    c = getchar();
    for(i++; c >= '0' && c <= '9'; i++) {...}
  }
  if (i < lim) {
    ...
  } else {
    while (c != '\n' && c != -1)
    ...
  }
  return;
}
```

- 这个程序有复杂的控制结构
- 限制路径长度为20。生成了178组测试数据
- 路径选择工具，仅用4条路径就覆盖了函数

符号执行：课堂作业

- 对于以下求最大公约数的代码（$m \geq 0, n \geq 0$），回答问题：

```
1   uint32_t gcd(
2     uint32_t m, uint32_t n
3   ) {
4     uint32_t x, y;
5     x = m;
6     y = n;
7     while (x != y) {
8       if (x > y)
9         x = x - y;
10      else
11        y = y - x;
12    }
13    return x;
14  }
```

- 1. 画出控制流图，并求该程序的圈复杂度
- 2. 使用符号执行的方法找到尽可能少的、达到分支和语句覆盖的路径，写出行号表示的路径，以及用符号值表示的路径约束
- 3. 针对以上路径采用符号执行技术分别给出一组可行输入数据（测试用例）
- 4. 仔细观察程序结构和 m、n 的取值范围，分析是否存在缺陷？

  结合符号执行的特点简要阐述。

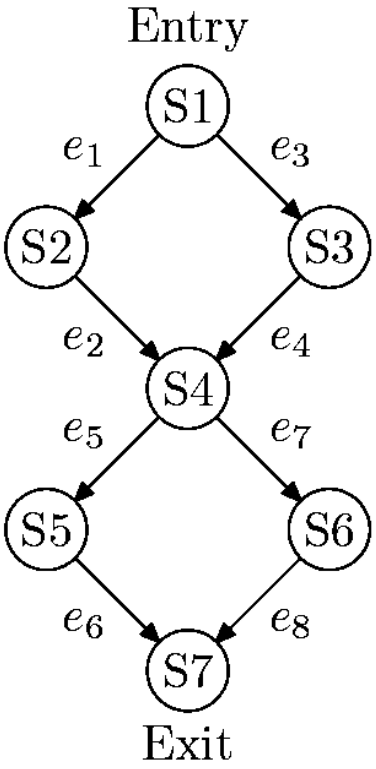
1.2 基本路径选择策略

路径的线性组合

- 定义路径向量 $A = \langle a_1, a_2, \ldots, a_K \rangle$，
  其中 a_i 表示路径中第 i 条边出现的次数
- 路径 B 是路径 $A_1, A_2, \ldots, A_n$ 的线性组合，如果
  $B = \lambda_1 A_1 + \lambda_2 A_2 + \cdots + \lambda_n A_n$
- 可以类似地定义线性相关、线性无关等概念

路径的线性组合（例）

- p1 = $e_1e_2e_5e_6$
  = ⟨1, 1, 0, 0, 1, 1, 0, 0⟩
- p2 = $e_1e_2e_7e_8$
  = ⟨1, 1, 0, 0, 0, 0, 1, 1⟩
- p3 = $e_3e_4e_5e_6$
  = ⟨0, 0, 1, 1, 1, 1, 0, 0⟩
- p4 = $e_3e_4e_7e_8$
  = ⟨0, 0, 1, 1, 0, 0, 1, 1⟩
  = −p1 + p2 + p3



基本路径测试

- 对于一个测试集合来说，如果流图中的任意一条完整路径都是测试路径集合的线性组合，那么这个测试集合满足基本路径覆盖
- 最优的基本路径集合大小 = V(G)
- 语句覆盖 $\subseteq$ 分支覆盖 $\subseteq$ 基本路径覆盖
- 重要的测试准则

对基本路径测试方法的批评

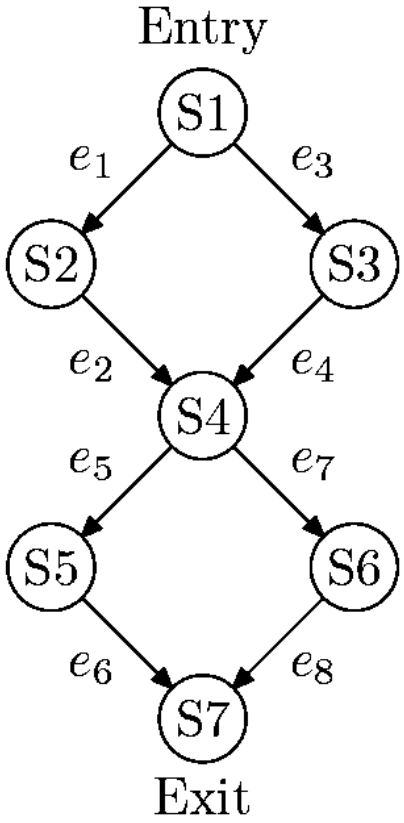
- M. Evangelist, An Analysis of Control Flow Complexity, COMPSAC 1984.
  - Basis path testing "*relies on deriving test data for a set of specific paths, which is an undecidable problem*".
- ……
- 大部分人仅仅从图的角度看待这种测试方法

Joseph Poole 的方法 （NIST, 1995）

```
FindBasis( node )
    if this node is a sink then print out this path as a solution
    else if this node has not been visited before
            mark the node as visited
            label a default edge
            FindBasis( destination of default edge)
            for all other outgoing edges,
                    FindBasis( destination of edge)
    else
            FindBasis( destination of default edge).
```

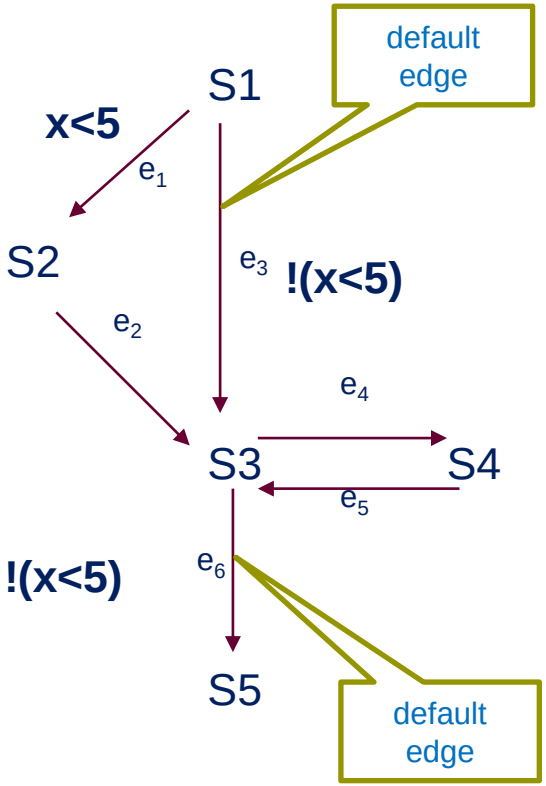
Poole方法示例

- 标记 e_1 e_5 为default edge
- p1 = $e_1e_2e_5e_6$
  = ⟨1, 1, 0, 0, 1, 1, 0, 0⟩
- p2 = $e_1e_2e_7e_8$
  = ⟨1, 1, 0, 0, 0, 0, 1, 1⟩
- p3 = $e_3e_4e_5e_6$
  = ⟨0, 0, 1, 1, 1, 1, 0, 0⟩
- V(G) = 3



Poole 方法是完美的吗？

```
          int x, y;
/*S1*/    if (x < 5)
/*S2*/      y = 2;
/*S3*/    while (x < 5)
/*S4*/      x++;
/*S5*/    return;
```



如果标记 e3 e6 为default edge

P1: e_3e_6

P2: $e_1e_2e_6$ // infeasible

P3: $e_3e_4e_5e_6$ // infeasible

其他的方法

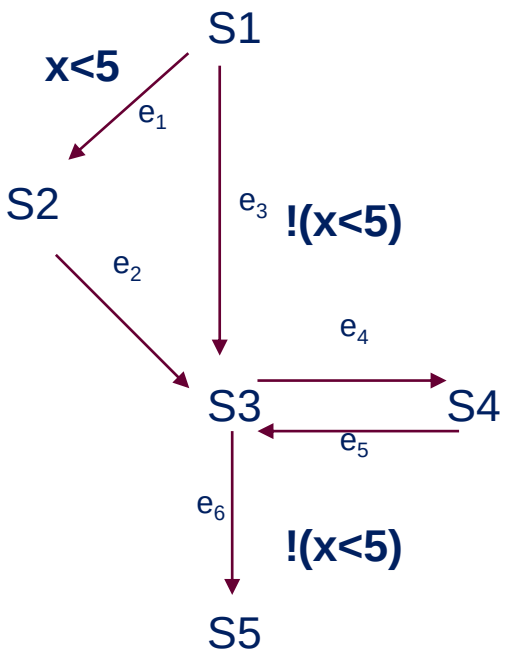
- **Baseline (McCabe, 1989)**
  - **Select a path with highest number of decision nodes**
  - **Retrace each decision in baseline path**
  - **Flip each decision to create a new path**
- **Dynamic (A.H. Watson, 1996)**

改进 [Yan-Zhang IPL 2008]

- 测试集合中的路径具有如下的性质：
  - 路径应该是可行的
  - 路径之间应该是线性无关的
- 依次枚举路径，满足如下条件则加入测试集
  1. 与测试集中的路径线性无关
  2. 可行
- 测试集合大小为V(G)时算法终止

示例

```
        int x, y;
/*S1*/  if (x < 5)
/*S2*/    y = 2;
/*S3*/  while (x < 5)
/*S4*/    x++;
/*S5*/  return;
```



Path	Sequence	Length	Test Data
1	$e_3\,e_6$	2	x = 5
2	$e_1\,e_2\,e_6$	3	infeasible
3	$e_3\,e_4\,e_5\,e_6$	4	infeasible
4	$e_1\,e_2\,e_4\,e_5\,e_6$	5	x = 4
5	$e_3\,e_4\,e_5\,e_4\,e_5\,e_6$	6	infeasible
6	$e_1\,e_2\,e_4\,e_5\,e_4\,e_5\,e_6$	7	x = 3

实验结果

- **coreutils** （<10s）

Function	v(G)	Max. Len.
getop()	11	16
strol()	7	7
InsertionSort()	5	24
dosify()	8	25
bsd_split_3()	6	41
attach()	5	6
remove_suffix()	4	30
quote_for_env()	4	6
isint()	9	39

小结

- 对程序行为的模拟是符号执行的关键
- 测试用例生成需要考虑的因素比较多
  - 覆盖准则
  - 路径可行性
  - 小测试集
  - 算法效率
- 针对性的测试生成是有挑战的工作

2 动态符号执行

静态符号执行遇到的困难/面对的挑战

- path explosion: the number of execution paths is huge. (Search strategy/heuristics)
- complicated symbolic expressions
- various P.L. features (arrays, pointers, floating-point operations, etc.)
- calls to external functions
- Constraint solving – can be difficult.
- ……

具体/混合符号执行

- 约束求解可能失败
  - 复杂的约束条件：高阶非线性，浮点数
- Concolic testing （concrete + symbolic）
  - 同时进行：具体执行、符号执行
    The algorithm follows the path that the concrete execution takes.
  - 约束无法求解时，用具体值代替符号值

Papers on concolic testing

- **Koushik Sen, Darko Marinov, Gul Agha. CUTE: a concolic unit testing engine for C. ESEC/SIGSOFT FSE 2005: 263-272.**
  - **The approach used builds on previous work combining symbolic and concrete execution, and more specifically, using such a combination to generate test inputs to explore all feasible execution paths.**
- **Koushik Sen, Gul Agha. CUTE and jCUTE: Concolic Unit Testing and Explicit Path Model-Checking Tools. CAV 2006: 419-423.**

[Koushik Sen, ASE 2007]

- **Concolic testing first generates random values for primitive inputs and the NULL value for pointer inputs. Then the algorithm does the following in a loop:**
  - **it executes the code concolically with the generated input. At the end of the execution a symbolic constraint in the path constraint is negated and solved using constraint solvers to generate a new test input that directs the program along a different execution path.**
- **The loop is repeated with the newly generated test input.**

- for some symbolic constraints, our constraint solver may not be powerful enough to compute concrete values that satisfy the constraints.
- To address this difficulty, such symbolic constraints are simplified by *replacing some of the symbolic values with concrete values.*

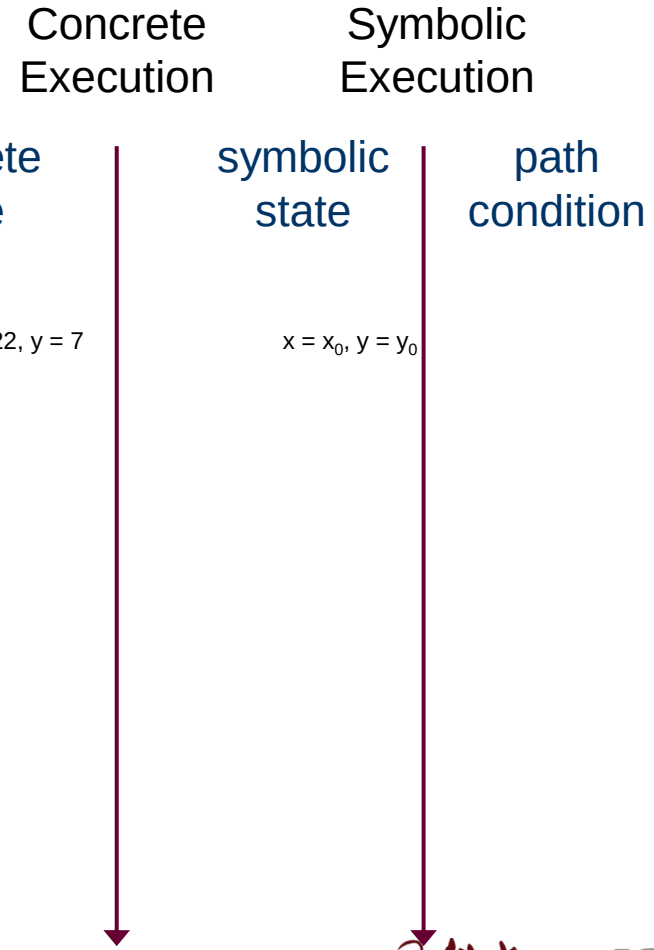
A simple example (Godefroid et al.)

```
int foo(int x) {
  // x is an input
  int y = x + 3;
  if (y == 13) abort();
  // error
  return 0;
}
```

- $x = 0$.
- The *else* branch is taken.
- Path constraint: $x+3 \neq 13$.
- This constraint is negated and solved, and we get $x = 10$.
- A new input is generated, which causes the program to follow the *then* branch.

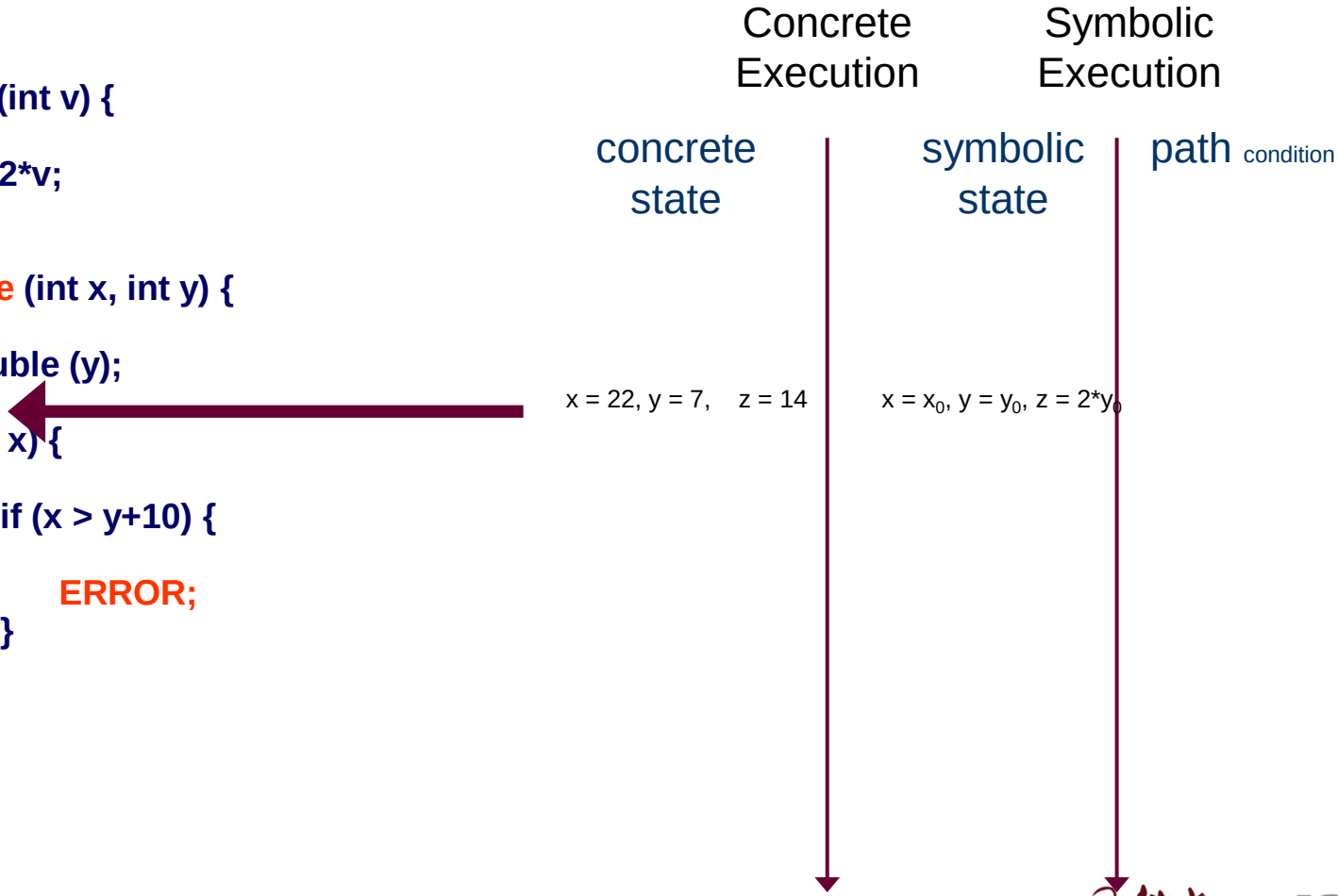
Concolic Testing Approach

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

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



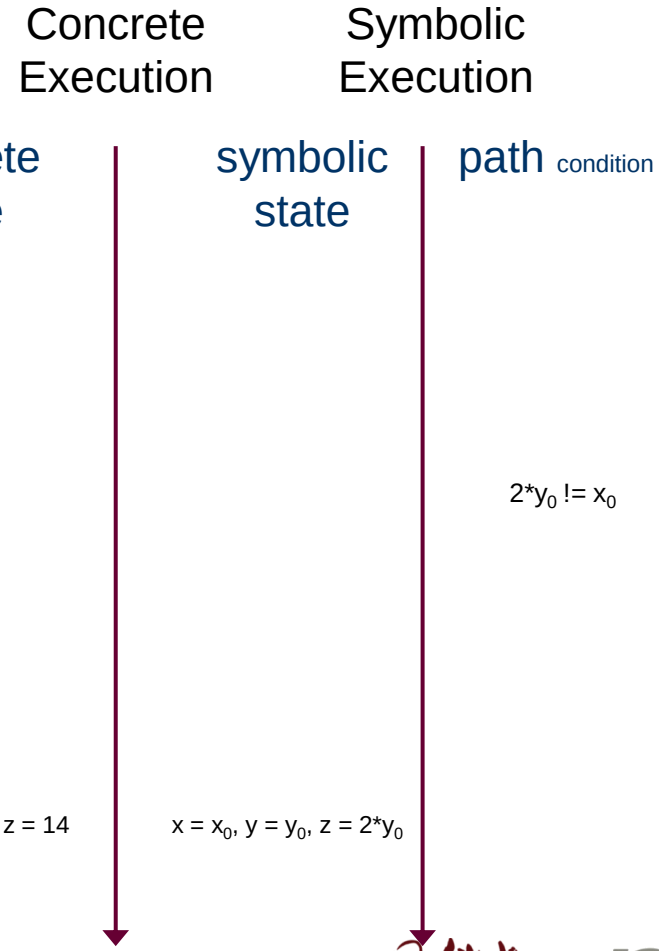
Concolic Testing Approach

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

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



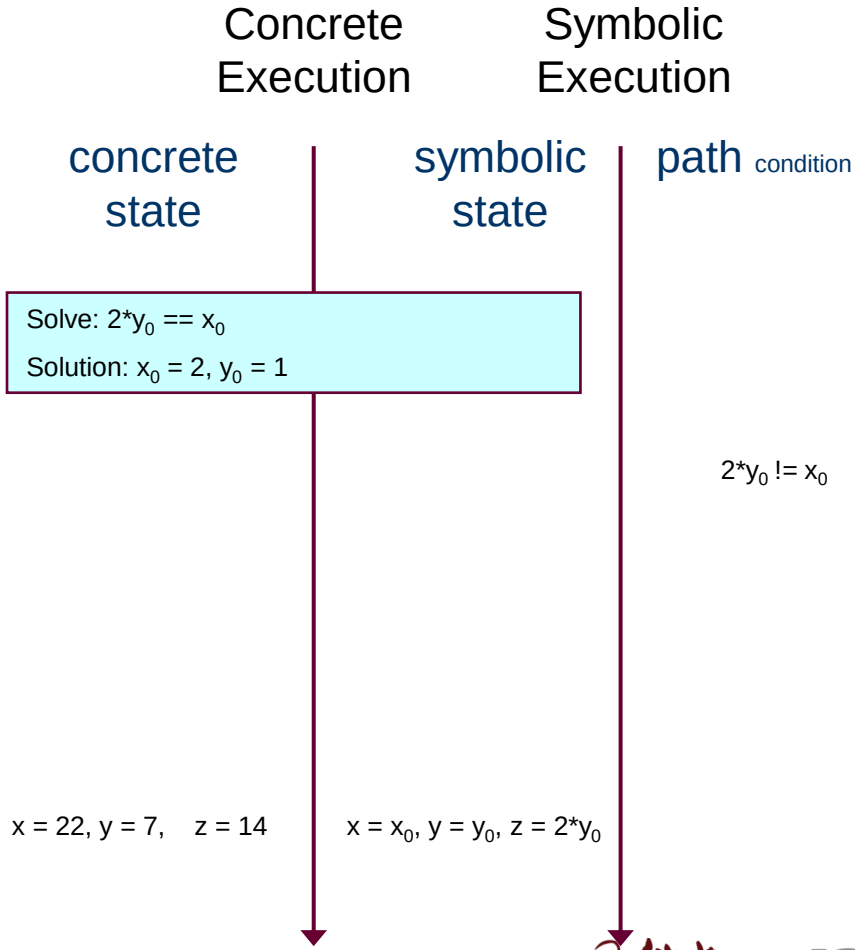
Concolic Testing Approach

```
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;
        }
    }
}
```



Concolic Testing Approach

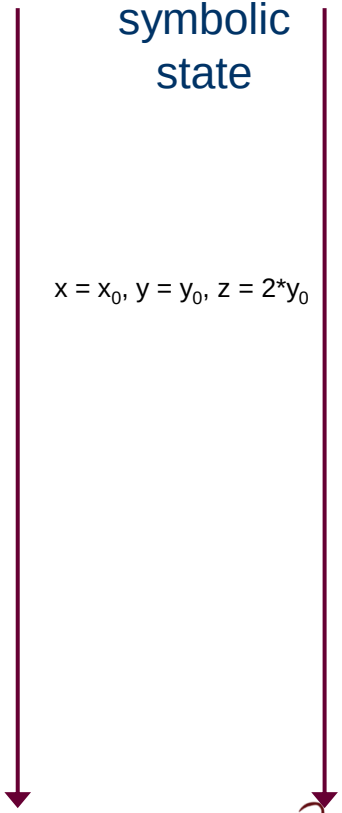
```
int double (int v) {
    return 2*v;
}

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

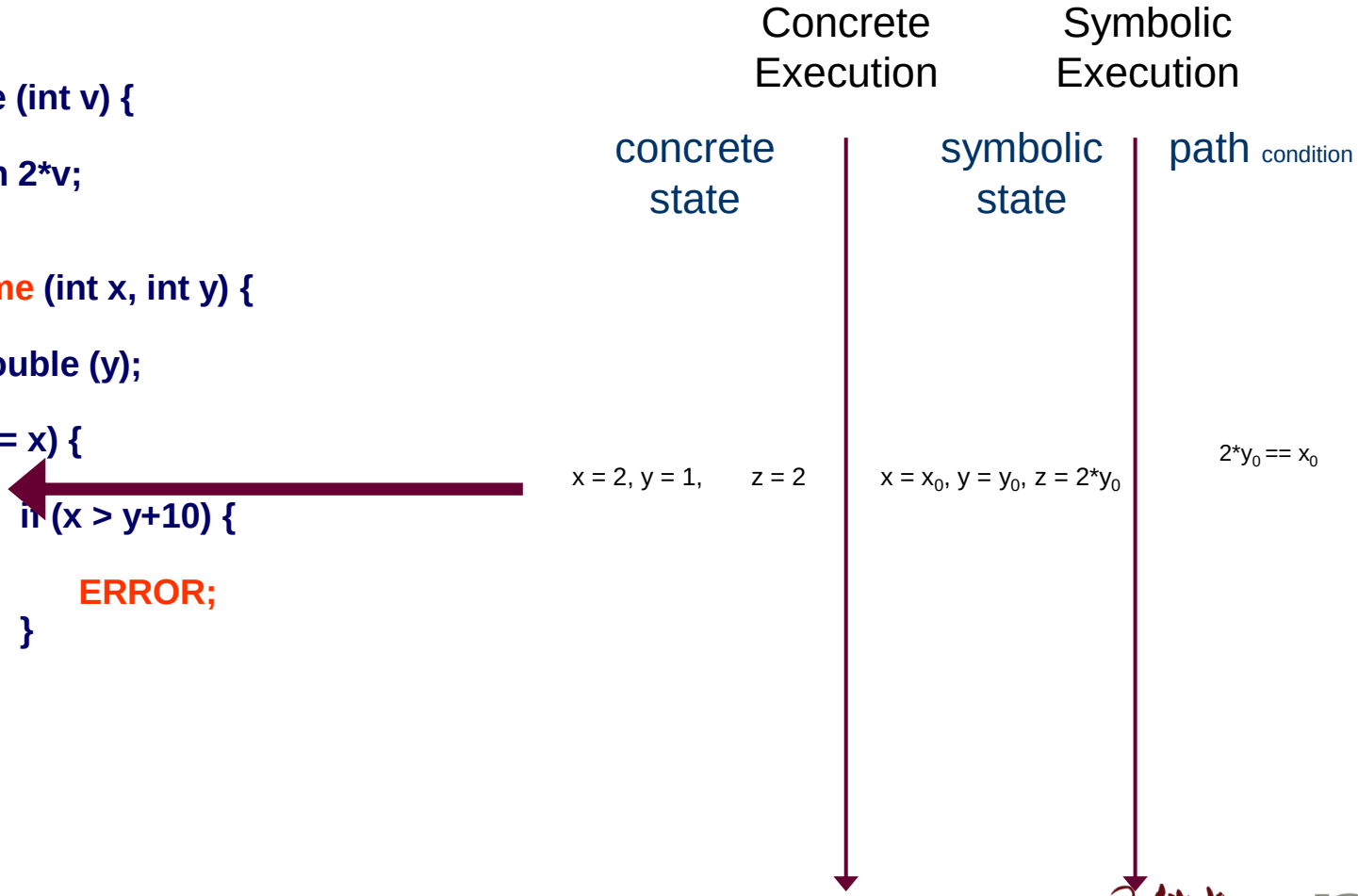
```
int double (int v) {
    return 2*v;
}

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	$x = x_0, y = y_0, z = 2*y_0$	



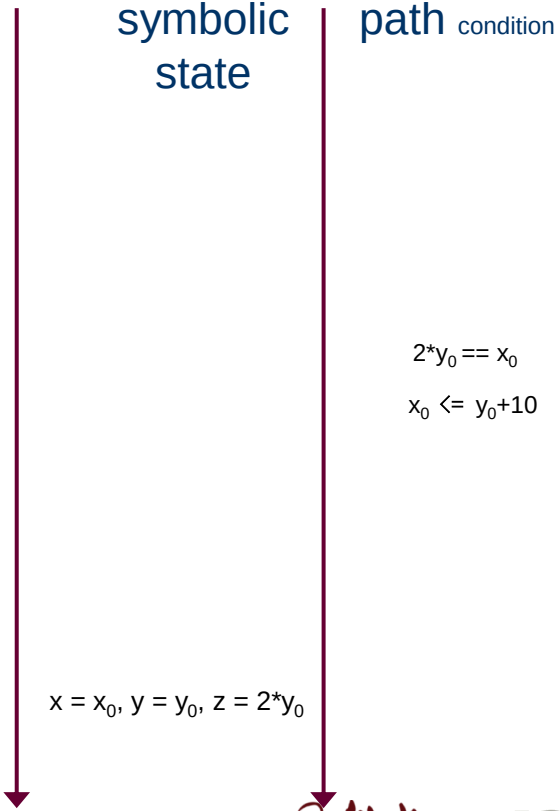
Concolic Testing Approach

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

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



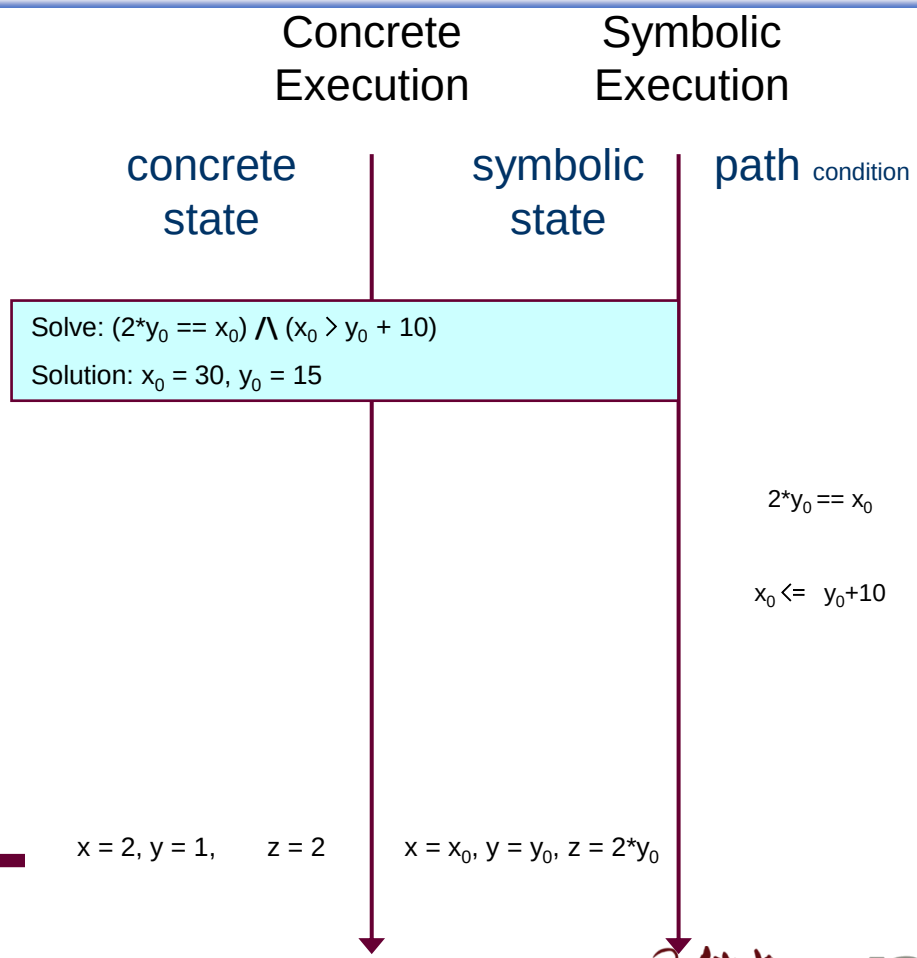
Concolic Testing Approach

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

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



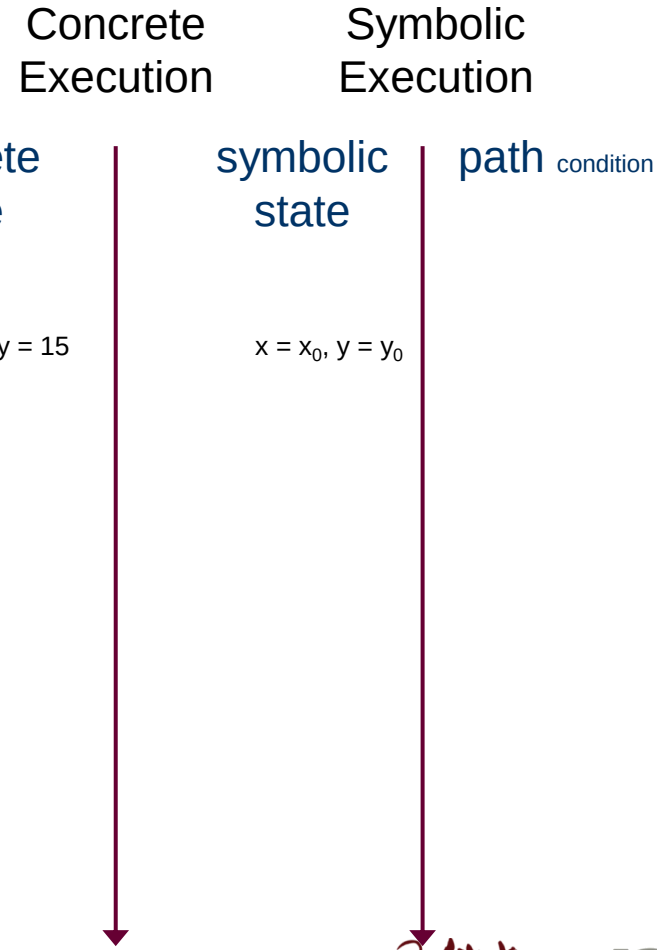
Concolic Testing Approach

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

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



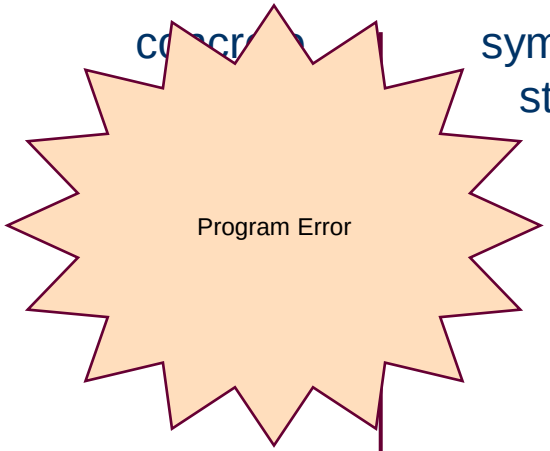
Concolic Testing Approach

```
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

$2*y_0 == x_0$

$x_0 > y_0+10$

x = 30, y = 15

$x = x_0, y = y_0$

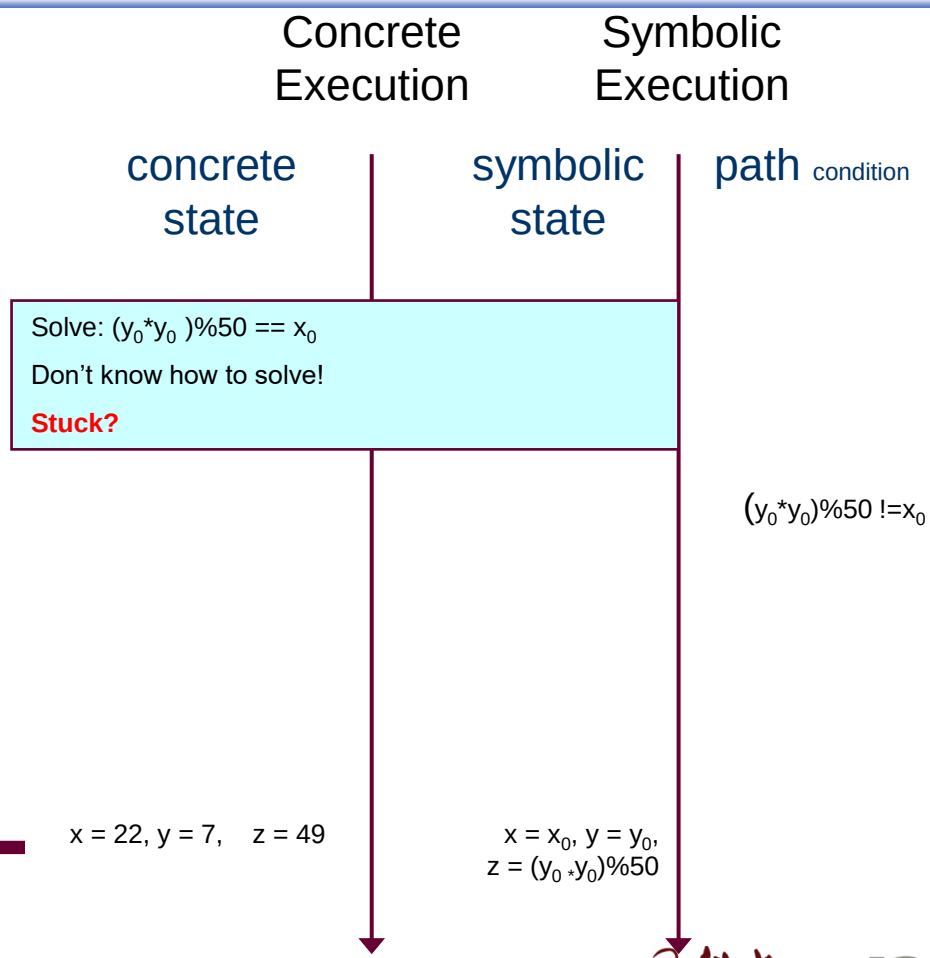
Novelty: Simultaneous Concrete and Symbolic Execution

```
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
x = 22, y = 7	$x = x_0, y = y_0$	

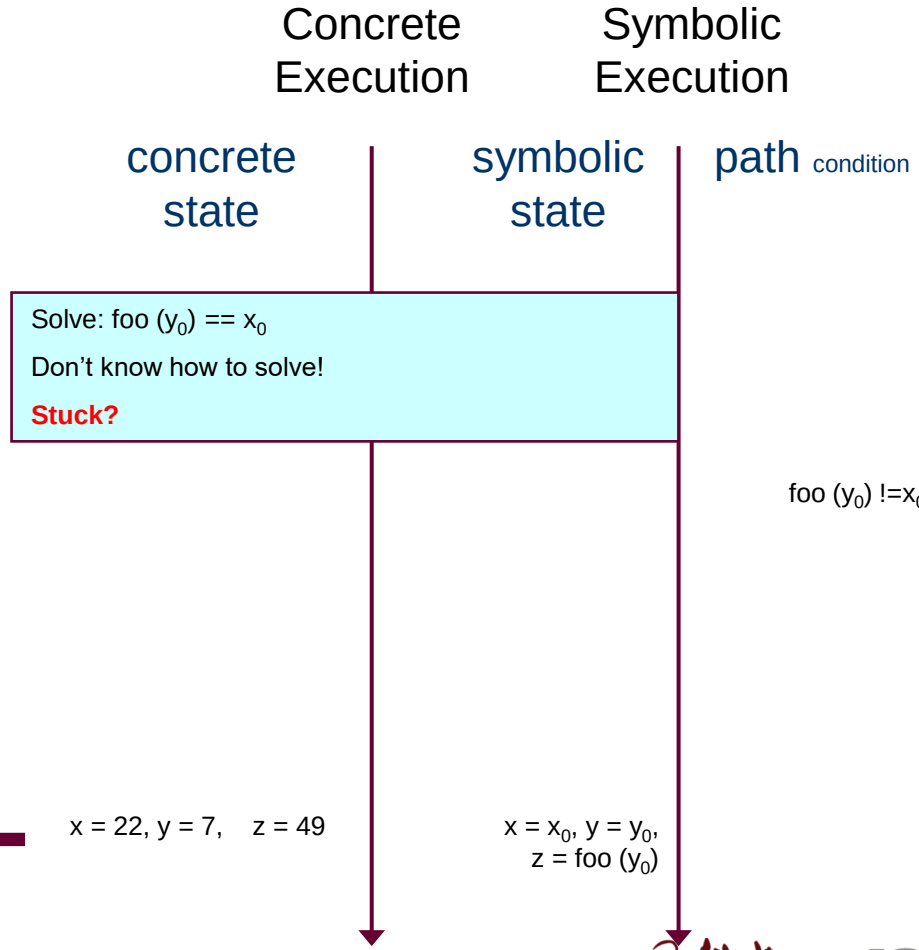
Novelty : Simultaneous Concrete and Symbolic Execution

```
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;
        }
    }
}
```



Novelty : Simultaneous Concrete and Symbolic Execution

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



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)

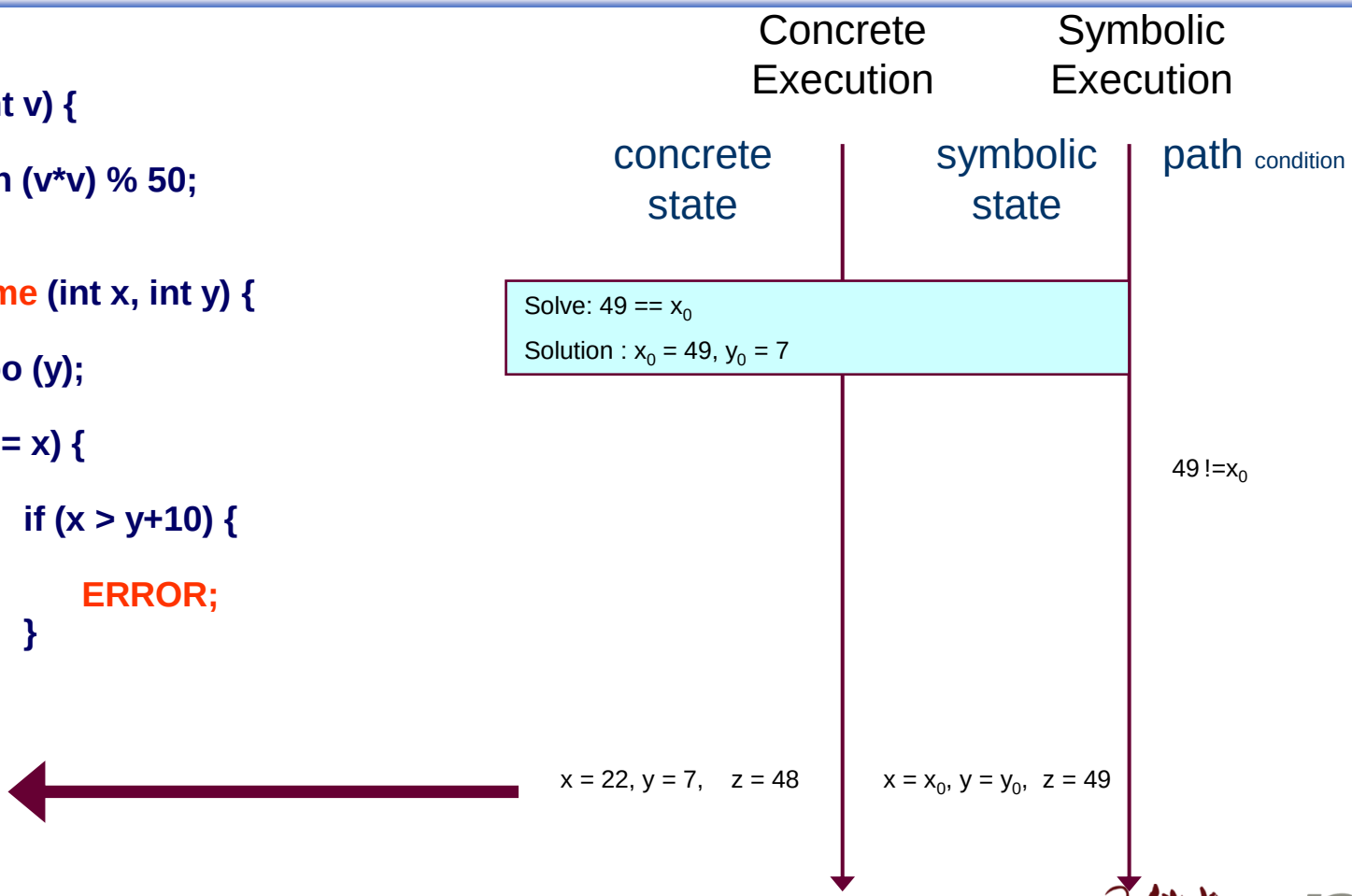
$(y_0 * y_0) \% 50 != x_0$

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

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

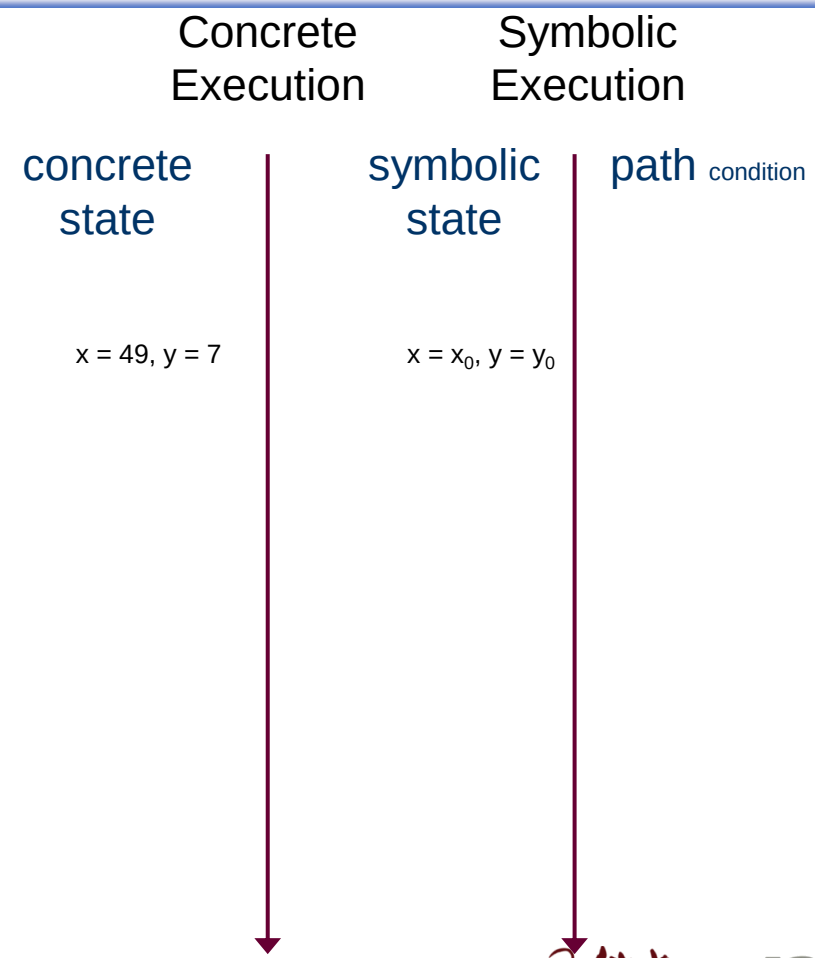
Novelty : Simultaneous Concrete and Symbolic Execution

```
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;
        }
    }
}
```



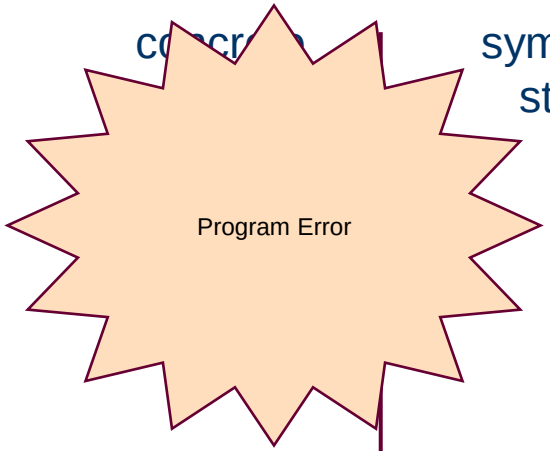
Novelty : Simultaneous Concrete and Symbolic Execution

```
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;
        }
    }
}
```



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;
        }
    }
}
```



SAGE (Scalable Automated Guided Execution)

- by Patrice Godefroid et al. (Microsoft)

- an alternative to blackbox fuzzing, called whitebox fuzzing. It builds upon recent advances in systematic dynamic test generation and extends its scope from unit testing to whole-program security testing.

  - P. Godefroid, M.Y. Levin, D. Molnar, Automated whitebox fuzz testing. NDSS 2008.

SAGE

- Starting with a well-formed input, white-box fuzzing consists of symbolically executing the program under test dynamically, gathering constraints on inputs from conditional branches encountered along the execution.
- The collected constraints are then systematically *negated* and solved with a constraint solver, whose solutions are mapped to new inputs that exercise different program execution paths.
- This process is repeated using novel search techniques that attempt to sweep through all (in practice, many) feasible execution paths of the program while checking many properties simultaneously using a runtime checker.

SAGE

- **SAGE implements a novel directed-search algorithm, dubbed *generational search*, that maximizes the number of new input tests generated from each symbolic execution.**
- **Given a path constraint, *all* the constraints in that path are systematically negated one by one, placed in a conjunction with the prefix of the path constraint leading to it.**

Techniques for handling constraints

- symbolic-expression caching
- local constraint caching
- unrelated constraint elimination
- constraint subsumption
- flip count limit
- …

3. 基于路径的静态分析

CGC

- **CYBER GRAND CHALLENGE，是由美国国防部高级研究计划局(DARPA)发起的“网络安全挑战赛”。**
  - 参赛团队打造“自动攻击系统”，在无人干预条件下相互竞争，寻找漏洞、利用漏洞攻击敌方，部署补丁抵御敌方的攻击。
  - 基于人力查找漏洞已经跟不上漏洞出现的速度和频率了。DARPA推出的这个CGC挑战赛就是为了解决此问题。
  - **CGC 2016 – 多支队伍采用符号执行技术。**

访问越界错误的自动查找

[Zitser et al. 2004] 使用各种分析、验证工具查找开源软件中14个越界错误

- static analysis tools (ARCHER, BOON, PolySpace C Verifier, Splint, and UNO)
- Sendmail, BIND and WU-FTPD
  - crackaddr.c; mime2.c

Misha Zitser, Richard Lippmann, Tim Leek: Testing static analysis tools using exploitable buffer overflows from open source code. SIGSOFT FSE 2004: 97-106

Abstract of [Zitser et al. 2004]

- Each code example included a "BAD" case with and a "OK" case without buffer overflows. Buffer overflows varied and included stack, heap, bss and data buffers; access above and below buffer bounds; access using pointers, indices, and functions; and scope differences between buffer creation and use. Detection rates for the "BAD" examples were low except for PolySpace and Splint which had average detection rates of 87% and 57%, respectively. However, average false alarm rates were high and roughly 50% for these two tools.

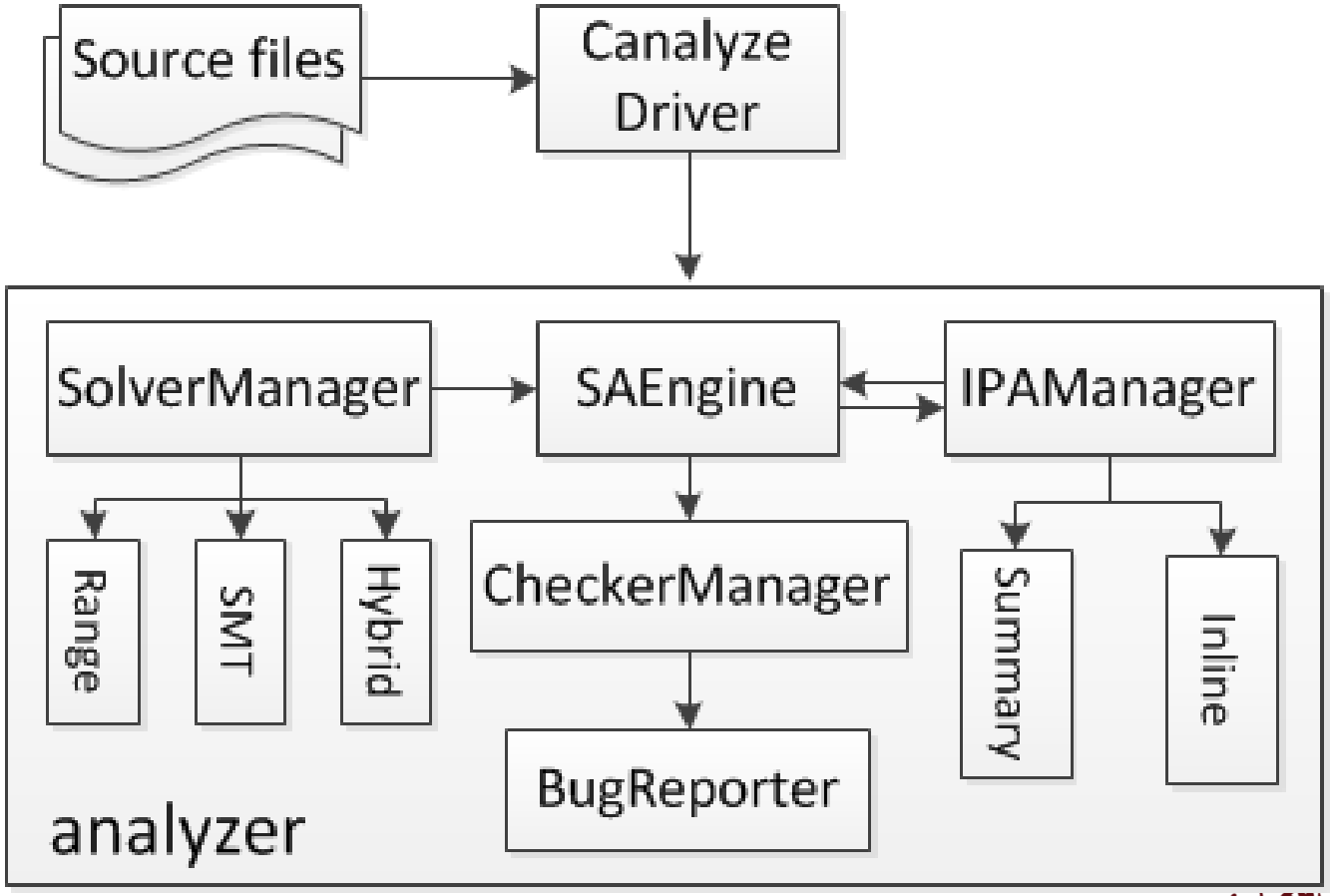
Canalyze: 采用路径分析的通用静态检测工具

- **Goal**
  - Automatically detect bugs in C program
  - As precise as possible (low false positive rate with high detection ability)
  - Rich diagnostic information in bug reports
- **Approach**
  - Symbolic analysis
  - Constraint-based analysis
  - Path-Sensitive analysis

Tool Overview

- **Bug Checkers**
  - **Memory related**
    - Use of undefined variables
    - NULL pointer deference
    - Double free
    - Memory leak
    - Buffer overflow
    - …
  - **Integer related**
    - Division by zero
    - Shift error
    - …

工具内部结构


Source files
Canalyze Driver
SolverManager
SAEngine
IPAManager
Range
SMT
Hybrid
CheckerManager
Summary
Inline
BugReporter
analyzer

错误报告

Canalyze Analysis Results

General Information

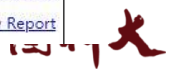
Source files	12
Total lines of code	5500
Start time	2013-05-05 20:03:29.715793
End time	2013-05-05 20:03:29.715808

Defect Summary

Defect Type	Quantity	Display
The operand has undefined value.	1	☑
Function argument is an uninitialized value.	2	☑
The pointer is NULL.	6	☑
NULL pointer passed as an argument to a nonnull parameter.	10	☑
Memory allocated in heap space is not freed.	2	☑
No call of chdir("/") immediately after chroot	28	☑

Reports

Bug ID	Defect Type	File	Path Length	
D100-49	The operand has undefined value.	options.c	63	View Report
D106-8	Function argument is an uninitialized value.	commands.c	9	View Report
D106-7	Function argument is an uninitialized value.	commands.c	232	View Report
D110-5	The pointer is NULL.	options.c	38	View Report
D110-2	The pointer is NULL.	options.c	46	View Report
D110-20	The pointer is NULL.	list.c	3	View Report
D110-6	The pointer is NULL.	login.c	27	View Report
D110-3	The pointer is NULL.	options.c	38	View Report
D110-4	The pointer is NULL.	options.c	69	View Report
D112-16	NULL pointer passed as an argument to a nonnull parameter.	commands.c	9	View Report
D112-19	NULL pointer passed as an argument to a nonnull parameter.	commands.c	20	View Report
D112-10	NULL pointer passed as an argument to a nonnull parameter.	commands.c	9	View Report
D112-9	NULL pointer passed as an argument to a nonnull parameter.	commands.c	40	View Report



ISCAS

Bugs in open source software

Bug Reports

- A complete list of bug reports of libosip2-4.0.0 can be seen here.
- A complete list of bug reports of bftpd can be seen here.
- A complete list of bug reports of httpd-2.4.4 can be seen here.
- A complete list of bug reports of lighttpd-1.4.32 can be seen here.
- A complete list of bug reports of OpenSSH-4.6p1 can be seen here.
- A complete list of bug reports of OpenSSH-5.9p1 can be seen here.
- A complete list of bug reports of pure-ftpd-1.0.36 can be seen here.
- A complete list of bug reports of wget-1.10.2 can be seen here.
- A complete list of bug reports of wget-1.13 can be seen here.

Bugs Found in Open Source Software

Software	KLoC	Undef. value	NULL ptr	Mem. leak	Use after free
libosip2-4.0.0	28.9		√	√	√
libosip2-3.6.0	29.0			√	
lighttpd-1.4.32	46.3	√		√	√
Openssh-5.9p1	89.8			√	
wget-1.13	91.8	√		√	
sqlite-3.7.11	139.2			√	
Coreutils-8.15	202.3		√		
Coreutils-8.17	211.9				√
sed-4.2	30.4	√			
glibc-2.15	1020.5		√		

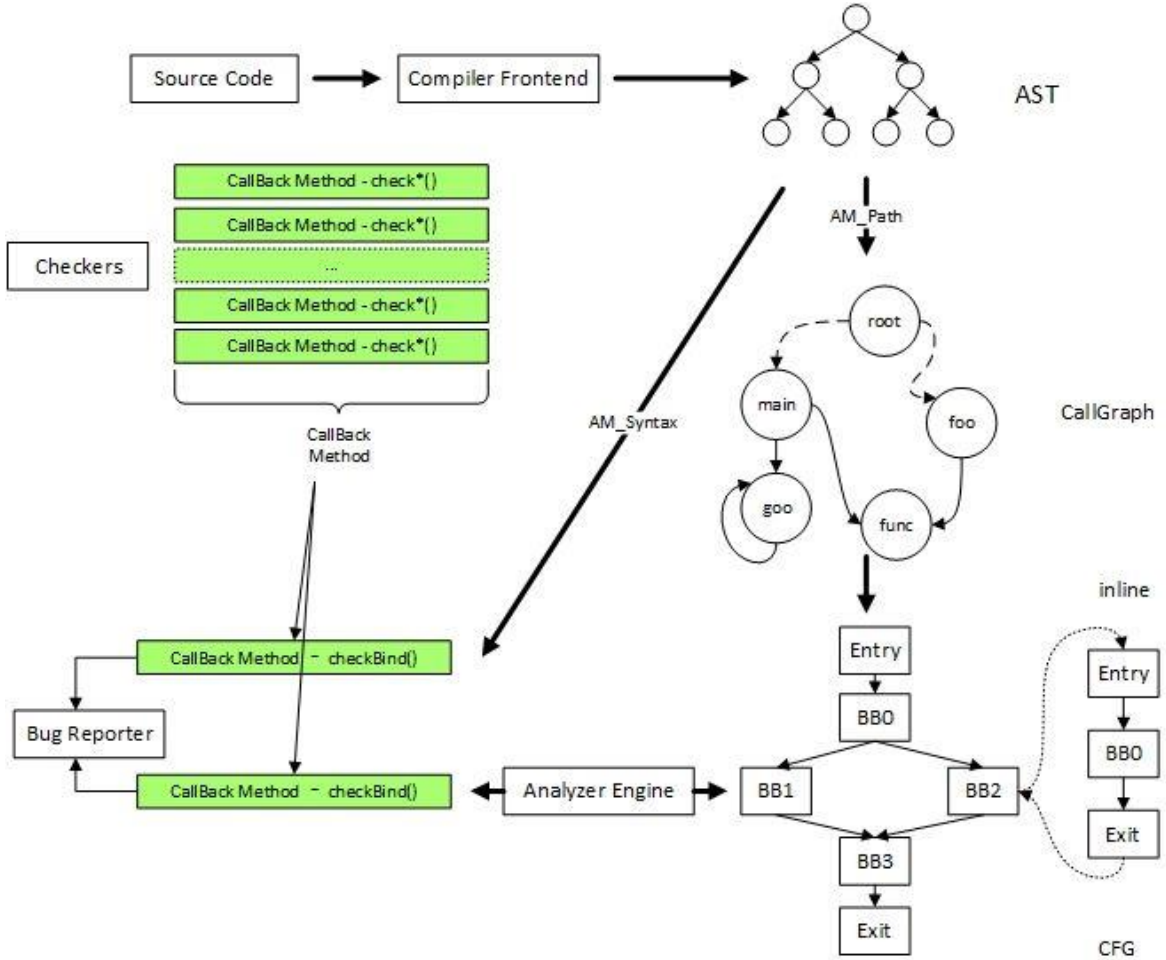
CSA（Clang Static Analyzer）

The Clang Static Analyzer is a source code analysis tool that finds bugs in C, C++, and Objective-C programs.

The analyzer is 100% open source and is part of the Clang project. Like the rest of Clang, the analyzer is implemented as a C++ library that can be used by other tools and applications.*

https://clang-analyzer.llvm.org/

CSA的结构


Source Code
Compiler Frontend
AST
CallBack Method - check*()
CallBack Method - check*()
...
CallBack Method - check*()
CallBack Method - check*()
Checkers
CallBack Method
AM_Path
AM_Syntax
root
main
foo
goo
func
CallGraph
inline
CallBack Method - checkBind()
Bug Reporter
CallBack Method - checkBind()
Analyzer Engine
Entry
BB0
BB1
BB2
BB3
Exit
Entry
BB0
Exit
CFG

CSA的报告

Bug Summary

File: /home/luoji/testcase/testcase/000095442/CWE369_Divide_by_Zero__int_zero_divide_01.c
Warning: line 30, column 22
Division by zero

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  /* TEMPLATE GENERATED TESTCASE FILE
2  Filename: CWE369_Divide_by_Zero__int_zero_divide_01.c
3  Label Definition File: CWE369_Divide_by_Zero__int.label.xml
4  Template File: sources-sinks-01.tmpl.c
5  */
6  /*
7   * @description
8   * CWE: 369 Divide by Zero
9   * BadSource: zero Fixed value of zero
10  * GoodSource: Non-zero
11  * Sinks: divide
12  *    GoodSink: Check for zero before dividing
13  *    BadSink : Divide a constant by data
14  * Flow Variant: 01 Baseline
15  *
16  * */
17
18 #include "std_testcase.h"
19
20 #ifndef OMITBAD
21
22 void CWE369_Divide_by_Zero__int_zero_divide_01_bad()
23 {
24     int data;
25     /* Initialize data */
26     data = -1;
27     /* POTENTIAL FLAW: Set data to zero */
28     data = 0;
       (1) The value 0 is assigned to 'data' →
29     /* POTENTIAL FLAW: Possibly divide by zero */
30     printIntLine(100 / data);
                    (2) ← Division by zero
31 }
32
33 #endif /* OMITBAD */
```

- 报告中有文件名、位置和错误类型。
- 报告中有涉及文件的源代码。
- 报告中有这个错误发生的路径。

Juliet测试集.
https://samate.nist.gov/SRD/view_testcase.php?tID=95442

小结

- 路径可行性及其判定 – 测试及分析的基础
- 符号执行 + 约束求解
  - 一条程序路径 → 路径条件 (PC)
  - 整个程序（流图） → 符号执行树
- 符号执行面临的困难
  - 表达式爆炸，路径爆炸，约束求解难，数据类型复杂，…

符号执行—参考文献

- Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. SELECT–a formal system for testing and debugging programs by symbolic execution. ICRS, pages 234– 245, 1975.
- James C. King. Symbolic execution and program testing. CACM, 19(7):385–394, 1976.
- Lori Clarke. A system to generate test data and symbolically execute programs, IEEE Transactions on Software Engineering, 2(3): 215–222, 1976.
- William E. Howden. Symbolic testing and the DISSECT symbolic evaluation system. IEEE Transactions on Software Engineering, 3(4): 266–278, 1977.

- KLEE. http://klee.github.io/