

# 第六讲 软件故障自动定位

---

李宁

2020

1. 程序切片技术
2. 基于程序特征谱的故障定位
3. 故障上下文定位
4. 测试用例约简

## 2.1 程序切片技术

### ■ 程序切片技术:

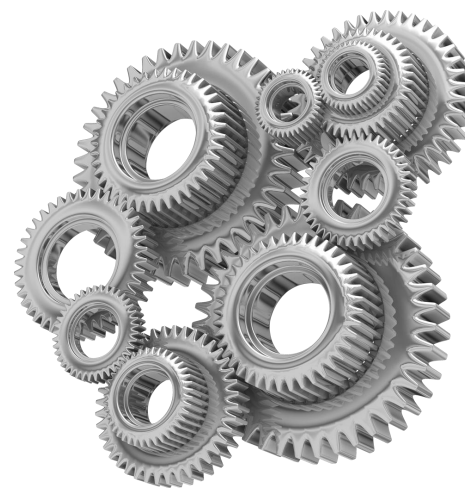
- ✓ 一种用于分解程序的程序分析技术;
- ✓ M. Weiser于1979年在他的博士论文中首次建立;
- ✓ 程序切片的应用
- ✓ 程序切片分类
  - 静态切片与动态切片
  - 前向切片与后向切片
  - 其他类型切片



## 2.1 程序切片技术

### ■ 程序切片的应用领域:

- ✓ 程序调试;
- ✓ 程序理解;
- ✓ 软件测试;
- ✓ 软件维护;
- ✓ 逆向工程
- ✓ ...



B站: 南京大学《软件分析》课程

## 2.1 程序切片技术

### ■ 程序切片的概念:

切片准则:  $S(n, V)$

含义: 在程序P中提取与n位置的变量V相关的语句集合。

```
(1)  read(n);  
(2)  i := 1;  
(3)  sum := 0;  
(4)  product := 1;  
(5)  while i <= n do  
      begin  
(6)    sum := sum + i;  
(7)    product := product * i;  
(8)    i := i + 1  
      end;  
(9)  write(sum);  
(10) write(product)
```

(a)

```
read(n);  
i := 1;  
  
product := 1;  
while i <= n do  
  begin  
    product := product * i;  
    i := i + 1  
  end;  
write(product)
```

(b)

图 2.1 (a) 一个程序示例 (b) 一个程序切片, 切片准则为 (10, product)

## 2.1 程序切片技术

### ■ 程序切片的分类 – 静态切片与动态切片

✓ **静态切片**：在编译时，即程序尚未运行时进行切片。

- 对输入没有任何假设，需分析所有可能的输入！切片大！
- 一般用于程序理解与软件维护

✓ **动态切片**：在某个具体的输入下，进行切片。

· 三元组  $(n, v, x)$ ：

其中  $n, v$  与静态切片含义相同， $x$  是输入序列。

## 2.1 程序切片技术

### ■ 程序切片的分类 – 静态切片与动态切片

```
main ()  
{  
    int x, y, z;  
    scanf ( "%d", &x);  
    if ( x<0 )  
        y = 1;  
    else  
        if ( x == 0 )  
            y = 1;  
        else y = 2;  
    printf ( "y = %d", y);  
    z = 2y;  
}
```

```
main ()  
{  
    int x, y, z;  
    scanf ( "%d", &x);  
    if ( x<0 )  
        y = 1;  
    else  
        if ( x == 0 )  
            y = 1;  
        else y = 2;  
    printf ( "y = %d", y);  
    z = 2y;  
}
```

静态切片（左）和  $x = 1$  时的动态切片（右）

切片准则：

$S(11, y)$

$S(11, y, x=1)$

## 2.1 程序切片技术

### ■ 程序切片的分类 – 后向切片与前向切片

```
main ()  
{  
    int x, y, z;  
    scanf ( "%d", &x);  
    if ( x<0 )  
        y = 1;  
    else  
        if ( x==0 )  
            y = 1;  
        else y = 2;  
    printf ( "y = %d", y );  
    z = 2y;  
}
```

```
main ()  
{  
    int x, y, z;  
    scanf ( "%d", &x);  
    if ( x<0 )  
        y = 1;  
    else  
        if ( x==0 )  
            y = 1;  
        else y = 2;  
    printf ( "y = %d", y );  
    z = 2y;  
}
```

后向切片（左）和前向切片（右）

切片准则：S(11, y)



## 2.1 程序切片技术

### ■ 程序切片的分类 - 层次切片

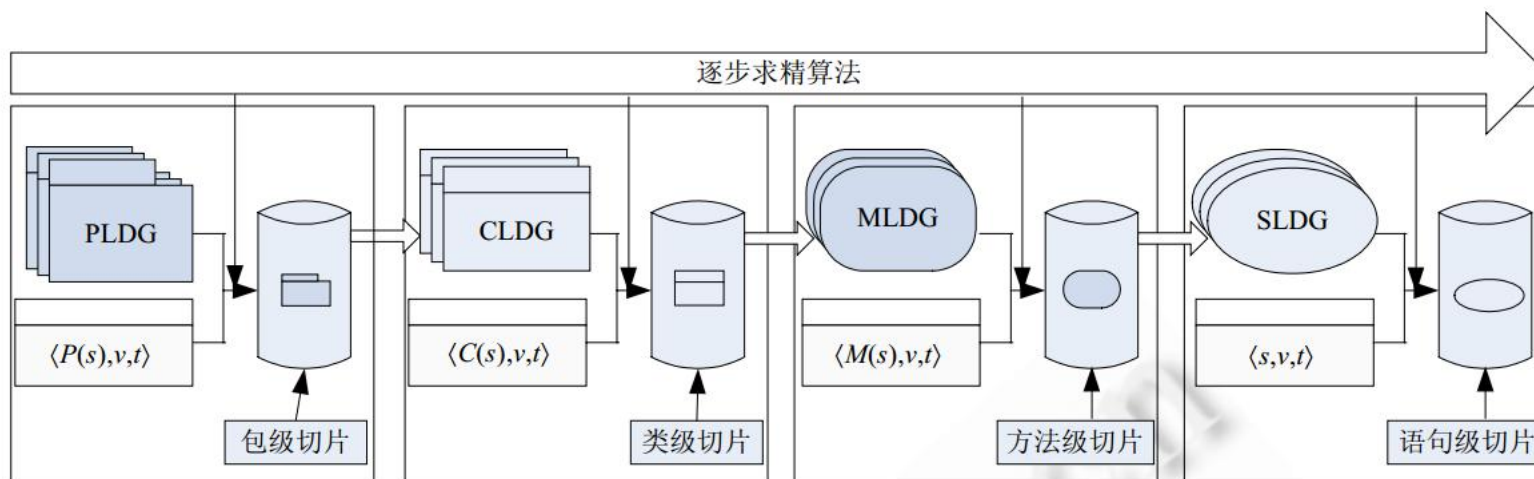


Fig.1 Hierarchical slicing model

图 1 层次切片模型

引自：基于条件执行切片谱的多错误定位，2013，计算机研究与发展

## 2.1 程序切片技术

### ■ 程序切片工具

- ✓ Wisconsin: 面向C语言，前向后向切片都可以
- ✓ Chopshop: 静态切片工具
- ✓ Ghinsu: 静态/动态/前向/后向
- ✓ Unravel: 面向ANSIC的静态切片
- ✓ .....



工具尝试: 尝试使用一种切片工具。

1. 程序切片技术

2. 基于程序特征谱的故障定位

3. 故障上下文定位

4. 测试用例约简



## 2.2 基于程序特征谱的故障定位

### ■ 程序特征谱:

关于**程序实体** (entity) 的**动态运行信息** (run-time profile)。

✓ **程序实体**是程序的基本组成单位，也是故障定位技术中用以描述程序代码错误所在的最小单位。如可执行语句、函数、基本语句块等。

✓ **动态运行信息**指关于程序实体在程序运行过程中的信息，比如覆盖 (hit)，执行序列 (execution sequence)，命中次数 (count) 等。

## 2.2 基于程序特征谱的故障定位

### ■ 基于程序特征谱的故障定位技术 - Tarantula

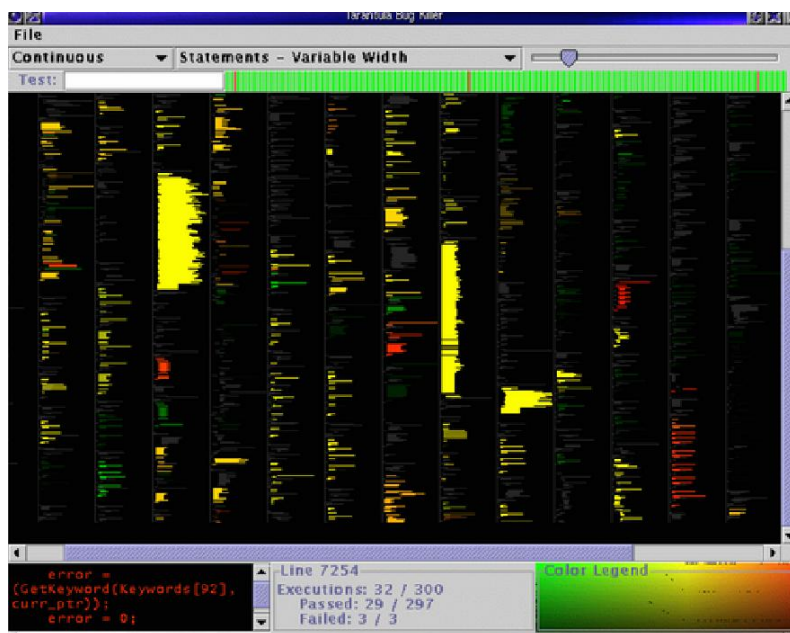
		Test Cases					
		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
mid() { int x,y,z,m;							
1:	read("Enter 3 numbers:",x,y,z);	●	●	●	●	●	●
2:	m = z;	●	●	●	●	●	●
3:	if (y<z)	●	●	●	●	●	●
4:	if (x<y)		●				
5:	m = y;		●				
6:	else if (x<z)	●				●	●
7:	m = y;	●					●
8:	else	●		●	●		
9:	if (x>y)			●			
10:	m = y;			●			
11:	else if (x>z)						
12:	m = x;						
13:	print("Middle number is:",m);	●	●	●	●	●	●
}							
Pass/Fail Status		P	P	P	P	P	F

m=x

- red: only for executed only during **failed** executions
- green: only for executed only during **passed** executions
- yellow: a statement is executed during **both failed and passed** executions

## 2.2 基于程序特征谱的故障定位

### ■ 基于程序特征谱的故障定位技术 - Tarantula



可疑度  
(suspiciousness)

$$hue(s) = \frac{\frac{passed(s)}{totalpassed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

$$\begin{aligned} suspiciousness(e) &= 1 - hue(e) = \\ &= \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}} \end{aligned}$$

● ● ● color range: red -> green

$$color(s) = low\ color\ (red) + \frac{\%passed(s)}{\%passed(s) + \%failed(s)} * color\ range$$

**% failed(s) = failed(s) / all failed**  
**% passed(s) = passed(s) / all passed**

## 2.2 基于程序特征谱的故障定位

### ■ 基于程序特征谱的故障定位技术 - Tarantula

	Test Cases						suspiciousness	rank
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		
mid() { int x,y,z,m;								
1:  read("Enter 3 numbers:",x,y,z);	●	●	●	●	●	●	0.5	7
2:  m = z;	●	●	●	●	●	●	0.5	7
3:  if (y<z)	●	●	●	●	●	●	0.5	7
4:     if (x<y)	●	●			●	●	0.63	3
5:         m = y;		●					0.0	13
6:     else if (x<z)	●				●	●	0.71	2
7:         m = y;   // *** bug ***	●					●	0.83	1
8:  else			●	●			0.0	13
9:     if (x>y)			●	●			0.0	13
10:        m = y;			●				0.0	13
11:     else if (x>z)				●			0.0	13
12:        m = x;							0.0	13
13: print("Middle number is:",m);	●	●	●	●	●	●	0.5	7
}								
	Pass/Fail Status							
	P	P	P	P	P	F		

- 第1句可疑度:  
 $\text{susp} = 1 / (1+1) = 0.5$

- 按照可疑度降序排列

第1, 2, 3, 13句具有相同的可疑度时:

1) ranked-first  
rank: 4

2) ranked-last  
rank: 7

Figure 1: Example of Tarantula technique.



## 2.2 基于程序特征谱的故障定位



### ■ 不同的可疑度计算公式

1) Jaccard:

$$Susp_e = \frac{failed_e}{total\ failed + passed_e}$$

2) Ochiai:

$$Susp_e = \frac{failed_e}{\sqrt{(total\ failed)(passed_e + failed_e)}}$$

3) Naishi1:

$$Susp_e = \begin{cases} -1 & \text{if } failed_e < F \\ P - failed_e & \text{if } failed_e = F \end{cases}$$

4) Naishi2:

$$Susp_e = failed_e - \frac{passed_e}{total\ passed + 1}$$

5) Wong1:

$$Susp_e = failed_e$$

6) Russel & Rao:

$$Susp_e = \frac{failed_e}{total\ failed + total\ passed}$$

Tarantula: 
$$Susp_e = \frac{\frac{failed_e}{total\ failed}}{\frac{passed_e}{total\ passed} + \frac{failed_e}{total\ failed}}$$

哪种最优? 参考paper:  
A Theoretical  
Analysis of the Risk  
Evaluation Formulas  
for Spectrum-Based  
Fault Localization  
(2013,  
ACM Transactions on  
Software Engineering  
and Methodology)



## 2.2 基于程序特征谱的故障定位

### ■ 不同的可疑度计算公式

哪种最优？结论：理论分析和实验表明ER1组和ER5组综合效果比较好  
参考paper: A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-Based Fault Localization (2013, ACM Transactions on Software Engineering and Methodology)


Name		Formula expression
ER1	Naish1 [Naish et al. 2011]	$\begin{cases} -1 & \text{if } a_{ef} < F \\ P - a_{ep} & \text{if } a_{ef} = F \end{cases}$
	Naish2 [Naish et al. 2011]	$a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$
ER5	Wong1 [Wong et al. 2007]	$a_{ef}$
	Russel & Rao [Naish et al. 2011]	$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$
	Binary [Naish et al. 2011]	$\begin{cases} 0 & \text{if } a_{ef} < F \\ 1 & \text{if } a_{ef} = F \end{cases}$

$a_{ef}$ : 执行s语句, fail  
 $a_{ep}$ : 执行s语句, pass  
 $a_{nf}$ : 不执行s语句, fail  
 $a_{np}$ : 不执行s语句, pass

## 2.2 基于程序特征谱的故障定位


### ■ 基于集合运算的故障定位技术 (coverage based information)

#### ✓ Set Union

$$E_{initial} = E_f - \bigcup_{p \in P} E_p$$


fail用例执行过，而所有的pass用例都没有执行过

#### ✓ Set intersection

$$E_{initial} = \bigcap_{p \in P} E_p - E_f$$


每一个pass用例都执行过，但是fail用例没有执行过

## 2.2 基于程序特征谱的故障定位

### ■ Set Union 举例 $E_{initial} = E_f - \bigcup_{p \in P} E_p$

mid() { int x,y,z,m;	Test Cases						suspiciousness	rank
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		
1: read("Enter 3 numbers:",x,y,z);	●	●	●	●	●	●	0.5	7
2: m = z;	●	●	●	●	●	●	0.5	7
3: if (y<z)	●	●	●	●	●	●	0.5	7
4:     if (x<y)	●	●			●	●	0.63	3
5:         m = y;		●					0.0	13
6:     else if (x<z)	●				●	●	0.71	2
7:         m = y;   // *** bug ***	●					●	0.83	1
8: else			●	●			0.0	13
9:     if (x>y)			●	●			0.0	13
10:        m = y;			●				0.0	13
11:     else if (x>z)				●			0.0	13
12:        m = x;							0.0	13
13: print("Middle number is:",m);	●	●	●	●	●	●	0.5	7
}								
Pass/Fail Status		P	P	P	P	F		

- 仅考虑T2-T6:

$f = \{1, 2, 3, 4, 6, 7, 13\}$

$U_p = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13\}$

$f - U_p = \{7\}$

Figure 1: Example of Tarantula technique.

## 2.2 基于程序特征谱的故障定位

### ■ 另一个mid.c的错误版本例子

程序	语句	T1	T2	T3	T4	T5
#include<stdio.h>	1					
main (int argc, char *argv[])	2					
{ int x, y, z, m;	3					
if (argc <4)	4	•	•	•	•	•
{ fprintf(stderr,"Error\n");	5					
exit (1); }	6					
x=(short int)atoi (argv[1]); //截断错误	7	•	•	•	•	•
y=atoi (argv[2]);	8	•	•	•	•	•
z=atoi (argv[3]);	9	•	•	•	•	•
m=z;	10	•	•	•	•	•
if (y<z)	11	•	•	•	•	•
{if (x<y)	12		•		•	•
m=y;	13		•		•	
else if (x<z)	14					•
m=x; }	15					•
else if (x>y)	16	•		•		
m=y;	17	•		•		
else if (x>z)	18					
m=x;	19					
printf ("%d\n", m); }	20	•	•	•	•	•
<hr/>						
T1:x=7,y=5,z=2						
T2:x=1,y=2,z=3						
T3:x=-317 696,y=2,z=-63 579		Result	P	P	F	P
T4:x=2,y=5,z=7						
T5:x=-327 696,y=-65 579,z=-3						

引自：  
基于程序特征谱整数  
溢出错误定位技术研  
究(2012)，  
计算机学报

## 2.2 基于程序特征谱的故障定位

### ■ 基于分支特征谱的定位（控制流）

控制依赖(分支)	T1	T2	T3	T4	T5	LOUPE 模型分支可疑度/CP 模型分支可疑度
(4,5)						0/-1
(4,7)	•	•	•	•	•	0.632/0
(11,12)		•		•	•	0.408/-0.143
(11,16)	•		•			0.5/0.2
(12,13)		•		•		0/-1
(12,14)					•	0.707/1
(14,15)					•	0.707/1
(16,17)	•		•			0.5/0.2
(16,18)						0/-1
(18,19)						0/-1

引自:

基于程序特征谱整数溢出错误定位技术研究(2012), 计算机学报

## 2.2 基于程序特征谱的故障定位

### ■ 基于数据定义使用对特征谱的定位 (数据流)

数据依赖(定义使用对)	T1	T2	T3	T4	T5	LOUPE 模型定义使用对可疑度
(7,12,x)		•		•	•	0.408
(7,14,x)					•	0.707
(7,15,x)					•	0.707
(7,16,x)	•		•			0.707
(7,18,x)						0
(7,19,x)						0
(8,11,y)	•	•	•	•	•	0.632
(8,12,y)		•		•	•	0.408
(8,13,y)		•		•		0
(8,16,y)	•		•			0.5
(8,17,y)	•		•			0.5
(9,10,z)	•	•	•	•	•	0.632
(9,11,z)	•	•	•	•	•	0.632
(9,10,z)						0
(9,14,z)					•	0.707
(9,18,z)						0
(10,20,m)	•	•	•	•	•	0.632
(13,20,m)		•		•		0
(15,20,m)					•	0.707
(17,20,m)	•		•			0.5
(19,20,m)						0
Result	P	P	F	P	F	注:“•”表示程序分支或定义使用对被测试用例覆盖. 测试用例与表 1 相同.

引自:

基于程序特征谱整数溢出错误定位技术研究(2012), 计算机学报



## 2.2 基于程序特征谱的故障定位

### ■ 基于多种特征谱的定位 (1)

		Fault 1: Statement 7 change to: m = y							Fault 2: Statement 3 change to: if (y<z-1)							Fault 3: Statement 2 change to: m = x						
		t1	t2	t3	t4	t5	t6	suspiciousness	t1	t2	t3	t4	t5	t6	suspiciousness	t1	t2	t3	t4	t5	t6	suspiciousness
mid() { int x,y,z,m; read(x,y,z); m = z; if (y<z) if (x<y) m = y; else if (x<z) m = x; else if (x>y) m = y; else if (x>z) m = x; print(m); }	Statements	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	
	1	●	●	●	●	●	●	0.41	●	●	●	●	●	●	0.58	●	●	●	●	●	●	0.41
	2	●	●	●	●	●	●	0.41	●	●	●	●	●	●	0.58	●	●	●	●	●	●	0.41
	3	●	●	●	●	●	●	0.41	●	●	●	●	●	●	0.58	●	●	●	●	●	●	0.41
	4	●	●			●	●	0.50	●	●			●	●	0.00	●	●			●	●	0.50
	5		●			●	●	0.00		●			●	●	0.00		●			●	●	0.00
	6	●				●	●	0.58		●			●	●	0.00	●				●	●	0.58
	7	●				●	●	0.71	●				●	●	0.00	●				●	●	0.00
	8			●	●	●	●	0.00		●	●	●	●	●	0.71			●	●	●	●	0.00
	9			●	●	●	●	0.00		●	●	●	●	●	0.71			●	●	●	●	0.00
	10			●		●	●	0.00			●		●	●	0.50			●		●	●	0.00
	11				●		●	0.00		●		●		●	0.50				●		●	0.00
	12					●	●	0.00					●	●	0.00					●	●	0.00
	13	●	●	●	●	●	●	0.41	●	●	●	●	●	●	0.58	●	●	●	●	●	●	0.41
	Branches																					
	Entry	●	●	●	●	●	●	0.41	●	●	●	●	●	●	0.58	●	●	●	●	●	●	0.41
	3 True	●	●			●	●	0.50	●	●			●	●	0.00	●	●			●	●	0.50
	3 False			●	●		●	0.00		●	●	●	●	●	0.71			●	●		●	0.00
	4 True		●					0.00		●					0.00		●					0.00
	4 False	●				●	●	0.58		●			●	●	0.00	●				●	●	0.58
	6 True	●				●	●	0.71	●				●	●	0.00	●				●	●	0.00
	6 False					●	●	0.00		●			●	●	0.00					●	●	1.00
	9 True			●				0.00		●		●		●	0.50			●				0.00
	9 False				●			0.00		●		●		●	0.50				●			0.00
	11 True					●		0.00		●		●		●	0.00					●		0.00
	11 False					●		0.00		●		●		●	0.50					●		0.00
Pass/Fail Status		P	P	P	P	P	F		P	F	P	P	F	P		P	P	P	P	F	P	

引自:  
 Lightweight Fault-  
 Localization Using  
 Multiple Coverage  
 Types (ICSE 2009)

red: fail test case

## 2.2 基于程序特征谱的故障定位



### ■ 基于多种特征谱的定位 (1)

		Fault 1: Statement 7 change to: m = y										Fault 2: Statement 3 change to: if (y<z-1)										Fault 3: Statement 2 change to: m = x									
		t1	t2	t3	t4	t5	t6	suspiciousness				t1	t2	t3	t4	t5	t6	suspiciousness				t1	t2	t3	t4	t5	t6	suspiciousness			
		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3					3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3					3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3				
mid() {	Statements																														
int x,y,z,m;	1	●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
read(x,y,z);	2	●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
m = z;	3	●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
if (y<z)	4	●	●			●	●	0.50	●	●							0.00	●	●			●				●	0.50	●	●		
if (x<y)	5		●				●	0.00									0.00		●								0.00		●		
m = y;	6	●				●	●	0.58	●	●							0.00	●				●				●	0.58	●	●		
else if (x<z)	7	●					●	0.71	●	●							0.00	●								●	0.00		●		
m = x;	8			●	●			0.00		●	●	●	●				0.71			●	●		●	●			0.00		●		
else	9			●	●			0.00		●	●	●	●				0.71			●	●		●	●			0.00		●		
if (x>y)	10			●				0.00				●					0.50				●						0.00		●		
m = y;	11				●			0.00		●		●					0.50					●					0.00		●		
else if (x>z)	12							0.00									0.00										0.00		●		
m = x;	13	●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
print(m);																															
}																															
DU-Pairs																															
1, 2, z		●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
1, 3, y		●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
1, 3, z		●	●	●	●	●	●	0.41	●	●	●	●	●	●	●	●	0.58	●	●	●	●	●	●	●	●	●	0.41	●	●		
1, 4, x		●	●			●	●	0.50	●	●							0.00	●	●			●				●	0.50	●	●		
1, 4, y		●	●			●	●	0.50	●	●							0.00	●	●			●				●	0.50	●	●		
1, 5, y								0.00									0.00										0.00		●		
1, 6, x		●				●	●	0.58	●	●							0.00	●				●				●	0.58	●	●		
1, 6, z		●				●	●	0.58	●	●							0.00	●				●				●	0.58	●	●		
1, 7, x		●					●	0.71	●	●							0.00	●				●				●	0.00		●		
1, 9, x				●	●			0.00		●	●	●	●				0.71			●	●		●	●			0.00		●		
1, 9, y				●	●			0.00		●	●	●	●				0.71			●	●		●	●			0.00		●		
1, 10, y				●				0.00				●					0.50				●						0.00		●		
1, 11, x					●			0.00		●		●					0.50					●					0.00		●		
1, 11, z					●			0.00		●		●					0.50					●					0.00		●		
1, 12, x								0.00									0.00										0.00		●		
2, 13, m					●	●		0.00		●		●					0.50				●	●					0.71		●		
5, 13, m			●					0.00									0.00		●								0.00		●		
7, 13, m		●					●	0.71	●	●							0.00	●				●				●	0.00		●		
10, 13, m				●				0.00				●					0.50				●						0.00		●		
12, 13, m								0.00									0.00										0.00		●		
Pass/Fail Status		P	P	P	P	P	F		P	F	P	P	F	P		P	P	P	P	F	P		P	P	P	P	F		P		

引自:

Lightweight Fault-  
Localization Using  
Multiple Coverage  
Types (ICSE 2009)



## 2.2 基于程序特征谱的故障定位

### ■ 基于多种特征谱的定位（1）

max-SBD: 最大值（语句+分支+定义使用对）  
avg-SBD: 平均值（语句+分支+定义使用对）  
avg-BD: 平均值（分支+定义使用对）

引自:

Lightweight Fault-  
Localization Using  
Multiple Coverage  
Types (ICSE 2009)

#### 实验结论:

1. max-SBD并不比使用单一覆盖类型的定位技术有效;
2. avg-SBD和avg-BD都要比单一覆盖类型的定位技术有效;

## 2.2 基于程序特征谱的故障定位

### ■ 基于多种特征谱的定位 (2) -Loupe模型

✓思想:

分别使用CDBug模型和DDBug模型来建模控制流和数据流上错误的异常行为, 选出相应的合适模型。

引自:

Locating faults  
using multiple  
spectra-specific  
models (SAC 2011)

$$Loupe(s) = \max(susp_{CDBug}(s), susp_{DDBug}(s)).$$

•CDBug:

$$susp_c(s) = |sim(\epsilon_c(s, \mathbf{true})) - sim(\epsilon_c(s, \mathbf{false}))|$$

计算每一个与语句S相关的true和false的可疑度, 计算其差值

•DDBug:

$$susp_d(s) = (\sum_{\epsilon \in \epsilon_d(s)} sim(\epsilon)) / |\epsilon_d(s)|$$

计算每一个与语句S相关的数据依赖的平均值

## 2.2 基于程序特征谱的故障定位

### ■ 常见的public数据集

✓ Siemens (西门子套件)

<http://sir.unl.edu/portal/>

✓ Space

表 3 实验程序信息

程序名	错误版本数	代码行数	测试用例数	简要描述
print_tokens	7	344	4 130	词法分析程序
print_tokens2	10	355	4 115	词法分析程序
schedule	9	292	2 650	优先级调度
schedule2	10	262	2 710	优先级调度
replace	32	512	5 542	正则表达式替换
tcas	41	135	1 608	海拔高度分离
tot_info	23	273	1 052	信息度量
space	38	9 562	13 525	语言注释器

注:前面 7 个是西门子程序集中包含的程序,它们的规模相对较小,space 程序的规模更大.

Table 2 Coverage types of the Siemens test suites

表 2 西门子用例集代码覆盖率类型

覆盖率类型	描述
testplans-cov	用例达到分支覆盖,并且无冗余
testplans-rand	根据 testplans-cov 中用例集所包含用例的平均数量 $A$ ,随机选取 $A/2$ 到 $2 \times A$ 数量的用例
testplans-rand-covsize	按照 testplans-cov 中每个用例集中用例的数量,随机选取相同数量的用例构成每个用例集
testplans-bigcov	用例达到分支覆盖,但其中包含冗余
testplans-bigrand	按照 testplans-bigcov 中每个用例集中用例的数量,随机选取相同数量的用例构成每个用例集
universe	所有测试用例

## 2.2 基于程序特征谱的故障定位



### ■ 故障定位方法的准确性度量

#### ● 单故障版本程序

定义错误定位方法的准确性为

$$q = 1 - \frac{rank}{total}$$

其中,  $rank$  表示诊断报告中语句的秩,  $total$  表示可执行语句总数.

#### ● 多故障版本程序

准确率: 找到首个错误语句前必须检查的语句个数, 占可执行语句总数的百分比

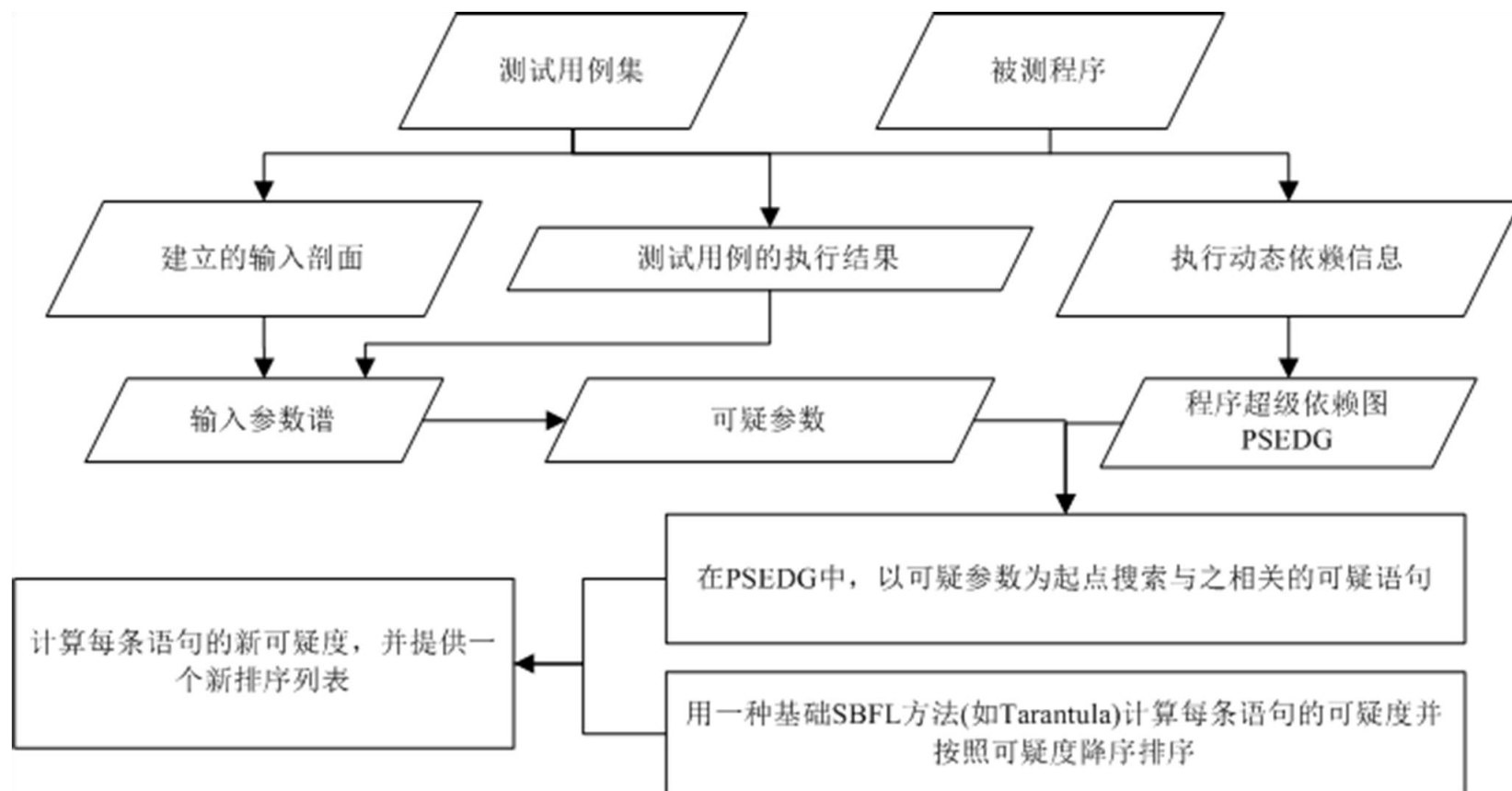
定义漏报率为

$$NP = 1 - \frac{D}{E}$$

其中,  $D$  为诊断报告中注入错误的数量,  $E$  为注入错误的总数量.

## 2.2 基于程序特征谱的故障定位

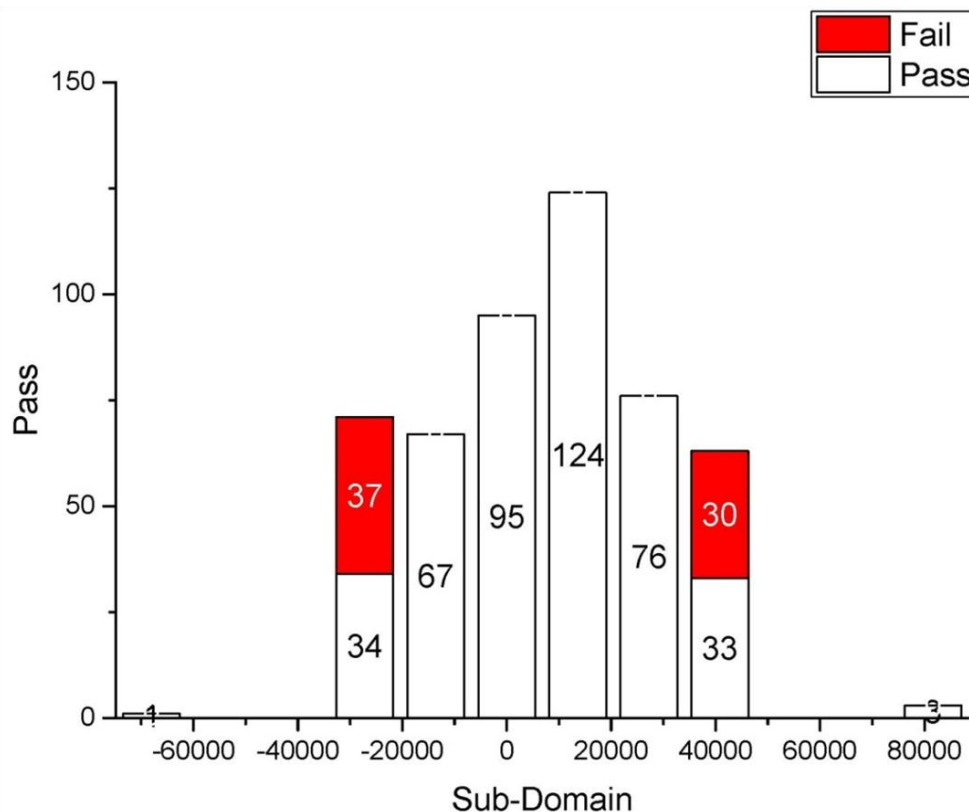
### ■ 部分研究工作介绍



## 2.2 基于程序特征谱的故障定位

### ■ 基于可疑参数的FL

1. 构建剖面模型：即将程序输入分为若干等价类，然后分析各个等价类和运行失效的测试用例的关系。



某段程序中的参数1构建的操作剖面实例。

每一根柱型都表示对该参数取值所划分的一个等价类。

横坐标表示等价类的取值范围。

纵坐标表示这个等价类上运行成功（使用白色标记）和运行失效（使用红色标记）的测试用例的个数。

由此，我们可以清晰的看出有两个等价类和程序运行错误密切相关。



## 2.2 基于程序特征谱的故障定位

### ■ 基于可疑参数的FL

根据构建的剖面分析可疑参数和可疑区间

$$Density_d = \frac{fail_d}{fail} \quad (1) \quad \text{错误密度, 表示该等价类上所占的错误比重}$$

$$Sensitivity_d = \frac{fail_d / total_d}{fail / total} \quad (2) \quad \text{错误敏感度, 该等价类错误密度与全局错误密度的比值}$$

$$Suspiciousness_d = Density_d * Sensitivity_d \quad (3) \quad \text{等价类可疑度, 该等价类的可疑度, 综合考虑等价类上的错误密度和错误敏感度}$$

$$Suspiciousness_P = \text{Max}(Suspiciousness_d) \quad (4) \quad \text{输入元可疑度, 该输入元的可疑度, 取值为这个输入元上所有等价类的最大可疑度}$$

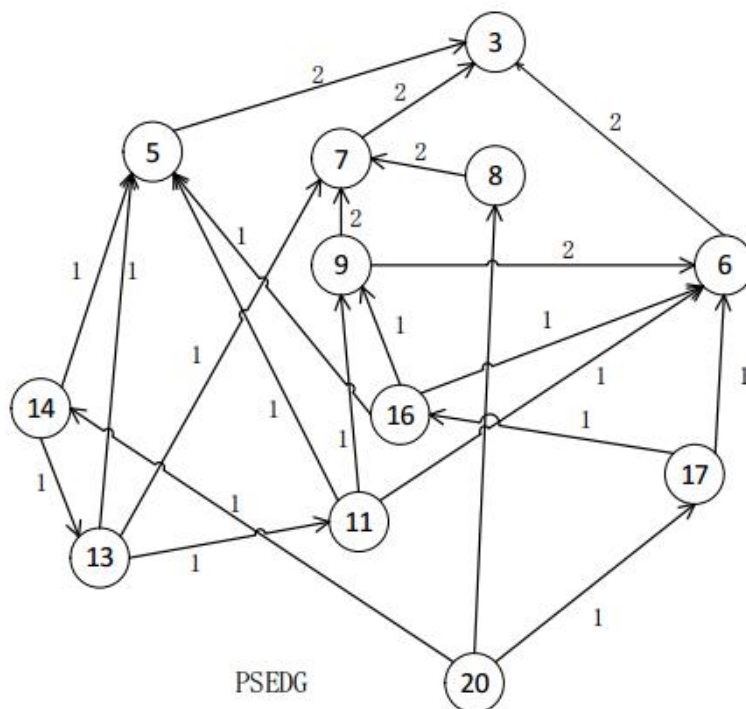
根据输入元可疑度, 设定一个**阈值**后, 可判断这个输入元**是否可疑**。

## 2.2 基于程序特征谱的故障定位



### ■ 基于可疑参数的FL

2. 构建依赖图：构建fail测试用例的依赖图





## 2.2 基于程序特征谱的故障定位

### ■ 基于可疑参数的FL

3. 搜索 $S_{SPN}$  : 以可疑参数为起点进行广度优先的搜索。

```
Input:
 $G_{super}$ : PSDG constructed with execution slicing
 $S_{para}$ : the set of suspicious parameters
Output:
 $S_{SPN}$ : the set of suspicious program entities
begin
  n=0;
  Create a new empty set  $S_{SPN}$ ;
  Create an empty set:  $S_n$ ;
  for each node node in  $G_{super}$ 
    that directly depends on any element  $\in S_{para}$ 
  do
    if node has the maximum weight
    then
      put node into  $S_0$ ;
    end
  end
   $S_{SPN} = S_0$ ;
  while  $|S_{SPN}| < |G_{super}| \times 0.1$  do
    n = n + 1;
    Create a new empty set  $S_n$ ;
    for each node in  $G_{super}$  that directly depends on
      or is depended by any element from  $S_{n-1}$ 
    do
      if node has the maximum weight
      then
        put node into  $S_n$ ;
      end
    end
     $S_{SPN} = S_{SPN} \cup S_n$ ;
  end
end
```

## 2.2 基于程序特征谱的故障定位

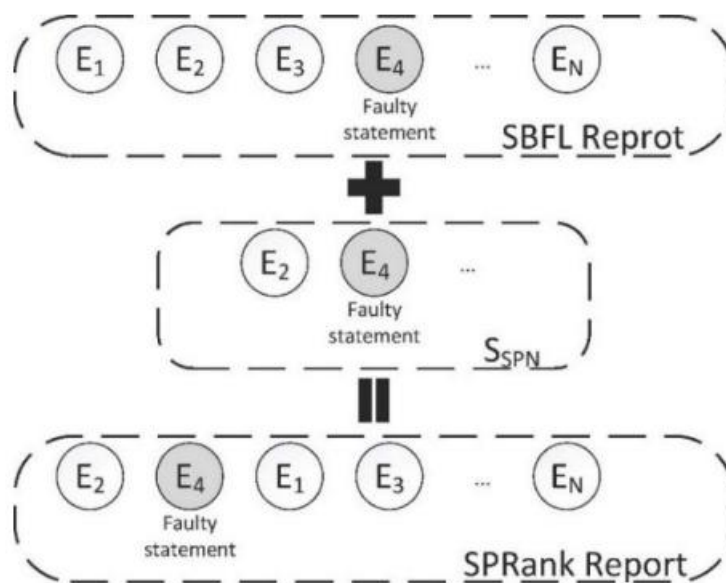


### ■ 基于可疑参数的FL

4. 重新计算语句的可疑度:

$$Susp_e = \frac{1}{2} \times SuspBasicSBFL_e + \frac{1}{2} \times flag \times Max(SuspBasicSBFL)$$

(若 $e$ 属于 $S_{SPN}$ ,  $flag = 1$ ; 不属于 $S_{SPN}$ ,  $flag = 0$ )



1. 程序切片技术

2. 基于程序特征谱的故障定位

3. 故障上下文定位

4. 测试用例约简



## 2.3 故障上下文定位

### ■ 基于图的故障上下文定位

- ✓ 程序流图
- ✓ 控制流图 (CFG)
- ✓ 程序依赖图 (PDG)
- ✓ 方法调用图

图挖掘的工具: gSpan, GAIA等

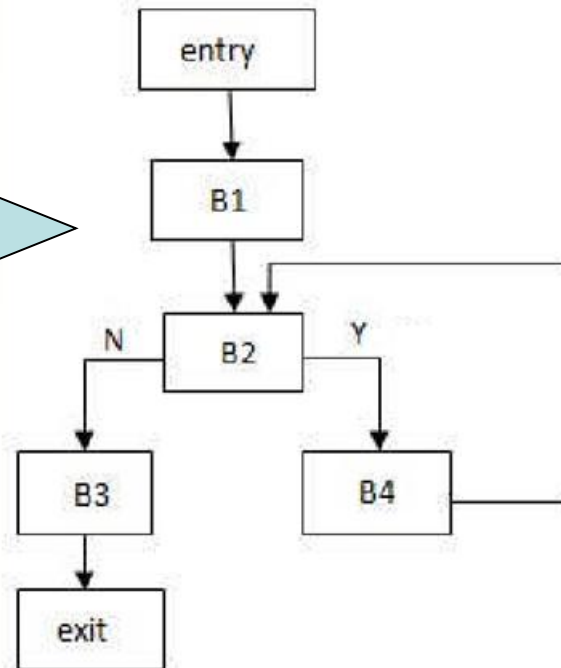
## 2.3 故障上下文定位

### ■ 程序流图

$G=(N, E)$ , 以基本块 (Block) 为节点的有向图

```
(1)  read(n);  
(2)  i := 1;  
(3)  sum := 0;  
(4)  product := 1;  
(5)  while i <= n do  
    begin  
(6)      sum := sum + i;  
(7)      product := product * i;  
(8)      i := i + 1  
    end;  
(9)  write(sum);  
(10) write(product)
```

基于块的程序流图



## 2.3 故障上下文定位

### ■ 程序控制流图 (CFG)

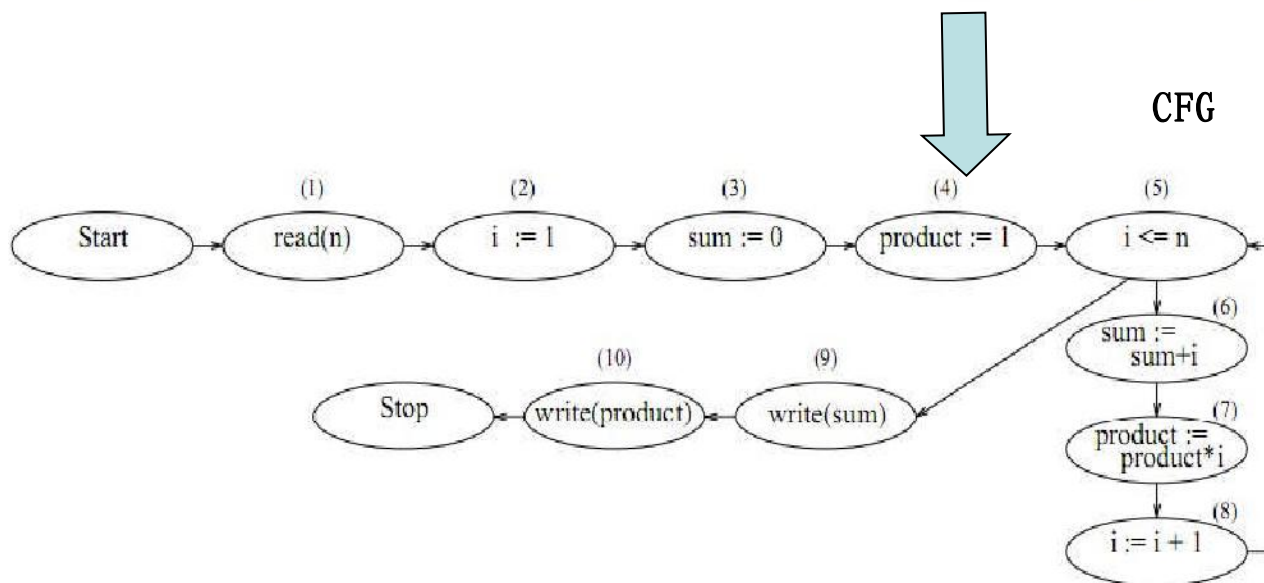
$G = (N, E, s, e)$ ,

N: 点集合; E: 边的集合;

s: 起点      e: 终点

```

(1)  read(n);
(2)  i := 1;
(3)  sum := 0;
(4)  product := 1;
(5)  while i <= n do
      begin
(6)    sum := sum + i;
(7)    product := product * i;
(8)    i := i + 1;
      end;
(9)  write(sum);
(10) write(product)
  
```

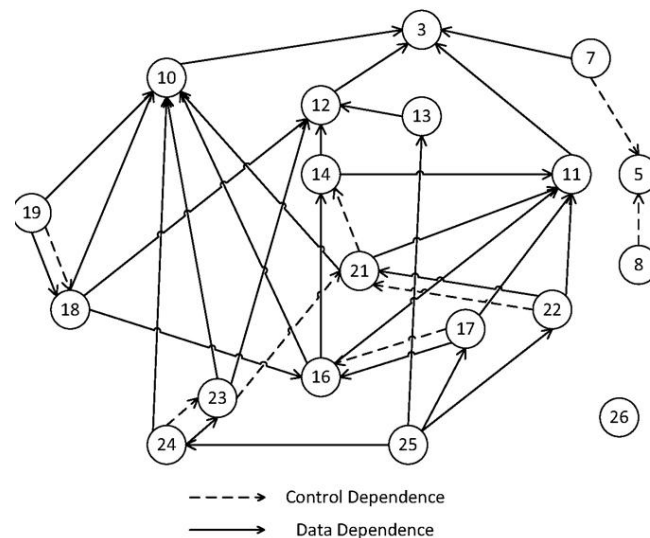


## 2.3 故障上下文定位

### ■ 程序依赖图 (PDG)

程序Mid.c(Last-Line)	T1	T2	T3	T4	T5
	(-5,3,-14)	(-4,7,2)	(2,5,7)	(-32800,5,2)	(-327696,-7,-23)
1 #include<stdio.h>					
2 main(int argc, char *argv[])					
3 {	.	.	.	.	.
4 int x,y,z,m;					
5 if(argc<4)	.	.	.	.	.
6 {					
7 fprintf(stderr, "Error\n");					
8 exit(1);					
9 }					
10 x = (short int)atoi(argv[1]);//截断错误	.	.	.	.	.
11 y = atoi(argv[2]);	.	.	.	.	.
12 z = atoi(argv[3]);	.	.	.	.	.
13 m = z;	.	.	.	.	.
14 if( y<z)	.	.	.	.	.
15 {					
16 if (x<y)		.			
17 m = y;					
18 else if (x<z)		.			
19 m = x;		.			
20 }					
21 else if(x>y)	.		.	.	
22 m = y;				.	
23 else if( x>z)	.		.		.
24 m = x;	.		.		.
25 printf( "%d\n", m);	.	.	.	.	.
26 }	.	.	.	.	.
Result	Passed	Passed	Passed	Failed	Failed

- 静态程序依赖图
- 动态程序依赖图



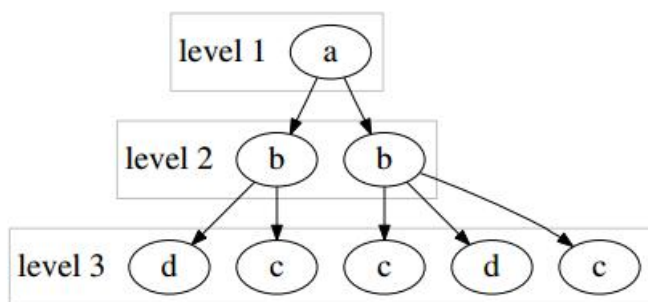
mid.c的动态程序依赖图示例



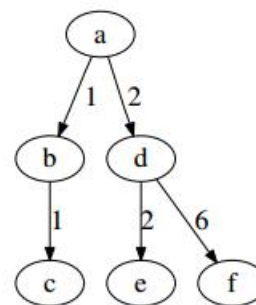
## 2.3 故障上下文定位

### ■ 程序调用图

- ✓ 分层：包/类/文件/方法
- ✓ 权重：调用次数/参数与返回值



分层调用图



带权重的调用图





## 2.3 故障上下文定位

### ■ 基于CFG的故障上下文定位 - 最大区分子图

Identifying Bug Signatures Using Discriminative Graph Mining, ISSTA2009

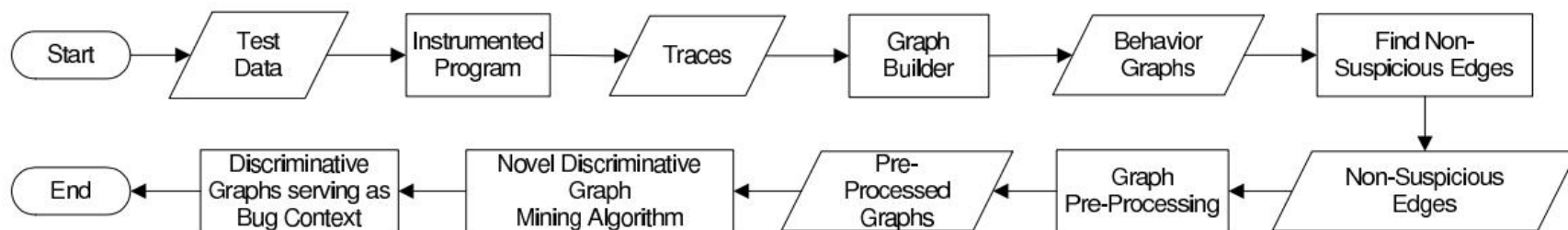


Figure 2: Context-Based Bug Localization Framework

✓ 核心理念:

将所有pass的CFG和fail的CFG分为两组，找出能两组图的最大不同所在即找出最大区分子图。

## 2.3 故障上下文定位

### ■ 基于CFG的故障上下文定位 - 最大区分子图

Identifying Bug Signatures Using Discriminative Graph Mining, ISSTA2009

#### ✓ 关键步骤:

1. 插桩获得程序的CFG (如何插桩? 相关的工具? )

gcov、wet等



工具尝试: 尝试使用一种插桩工具。

2. 在构建好的CFG图中去除不可疑边, 仅保留可疑边  
(如何定义可疑边)

$$susp_{edg} = \frac{failed(edg)}{passed(edg)} > \frac{totalfailed}{totalpassed}$$

3. 在fail和pass两种图中挖掘最大区分子图 (方法? Leap Search)

## 2.3 故障上下文定位

### ■ 基于CFG的故障上下文定位 - 最大区分子图

```

1: void replaceFirstOccurence (string arr [], int len, string sx, string sy, string sz) {
2:   for (int i=0;i<len;i++) {
3:     if (arr[i]==sx){
4:       arr[i] = sz;
5:       // a bug, should be a break;
6:     }
7:     if (arr[i]==sy)){
8:       arr[i] = sz;
9:       // a bug, should be a break;
10:    }
11:  }
12: }

```

程序功能：当sx字符串或者sy字符串第一次出现时，用sz字符串代替。

**错误：**sz不仅代替了第一次出现，代替了sx或sy所有的出现。

Table 1: Buggy Code

测试用例T1-T4

No	arr	sx	sy	sz	
1	{a, b}	a	g	1	pass
2	{a, b}	g	a	1	pass
3	{a, g}	a	g	1	fail
4	{a, g}	g	a	1	fail

No	Trace
1	<1, 2, 3, 4, 7, 10, 2, 3, 7, 10>
2	<1, 2, 3, 7, 10, 2, 3, 7, 8, 10>
3	<1, 2, 3, 4, 7, 10, 2, 3, 7, 8, 10>
4	<1, 2, 3, 7, 8, 10, 2, 3, 4, 7, 10>

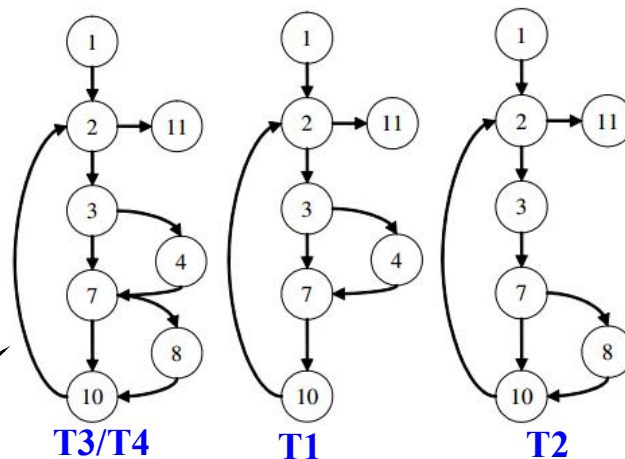


Figure 3: Control Flow Graph (CFG) of Code in Table 1 and Behavior Graph of Traces 3 & 4 (left), Behavior Graph of Trace 1 (middle), and Behavior Graph of Trace 2 (right). All edges in the behavior graphs are labeled as trans.

## 2.3 故障上下文定位

### ■ 基于CFG的故障上下文定位 - 最大区分子图

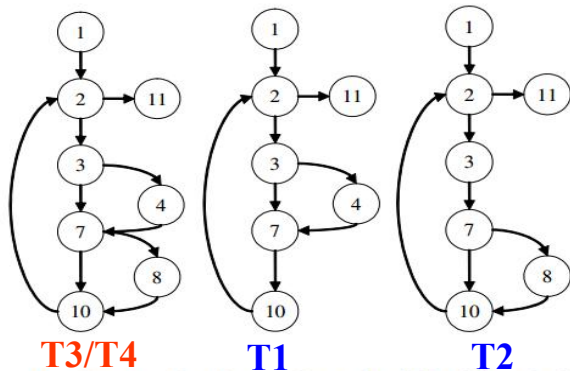


Figure 3: Control Flow Graph (CFG) of Code in Table 1 and Behavior Graph of Traces 3 & 4 (left), Behavior Graph of Trace 1 (middle), and Behavior Graph of Trace 2 (right). All edges in the behavior graphs are labeled as trans.

计算边可疑度



$$\frac{total\ failed}{total\ passed} = 2/2 = 1$$

例如:

$$susp_{1-2} = 2/2 = 1$$

$$susp_{3-4} = 2/1 = 2 > 1 \text{ (可疑)}$$

• • •

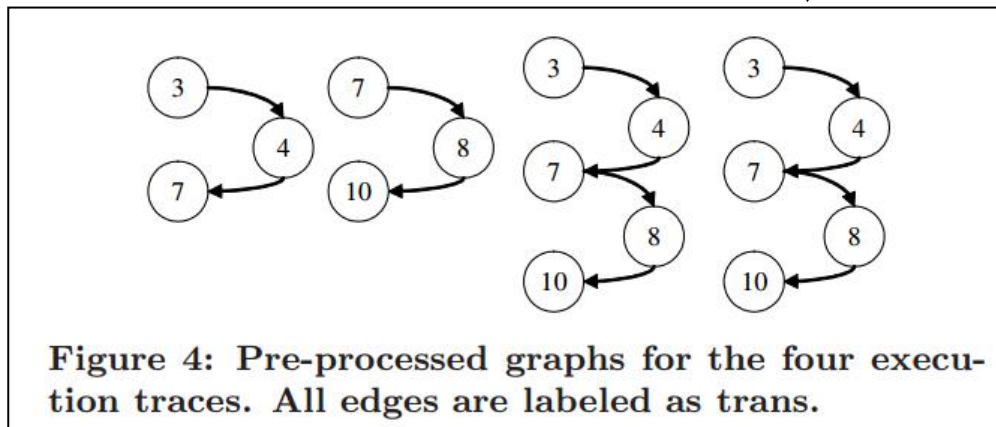
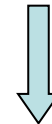
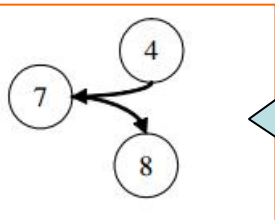


Figure 4: Pre-processed graphs for the four execution traces. All edges are labeled as trans.



最大区分子图

仅保留可疑  
边和节点



## 2.3 故障上下文定位

### ■ 基于CFG的故障上下文定位 - 最大区分子图

Prog.	RAPID			Top-K LEAP		
	Pre.	Rec.	Size	Pre.	Rec.	Size
tcas	82.9	82.9	8.0	85.9	95.1	5.0
ptok	71.4	71.4	4.0	85.7	100	4.3
ptok2	20.0	20.0	2.7	36.0	60.0	2.9
sched	33.3	33.3	2.3	54.1	66.7	3.6
sched2	0.0	0.0	N/A	24.2	30.0	2.2
tinfo	21.7	21.7	2.5	69.6	78.3	2.4
rep	53.1	53.1	5.1	54.4	81.3	2.9
<b>Avg.</b>	40.4	40.4	4.1	58.5	73.0	3.3

Table 2: Result - Method Level

Prog.	RAPID			Top-K LEAP		
	Pre.	Rec.	Size	Pre.	Rec.	Size
tcas	90.2	90.2	11.3	88.3	100	3.8
ptok	100	100	9.7	85.7	100	4.8
ptok2	65.0	70.0	7.2	74.0	100	3.4
sched	75.0	77.8	5.1	86.7	88.9	3.2
sched2	40.0	40.0	2.6	52.0	80.0	2.8
tinfo	56.5	56.5	15.4	55.0	87.0	3.6
rep	80.5	81.3	20.7	78.1	81.3	4.9
<b>Avg.</b>	72.5	73.7	10.3	74.3	91.0	3.8

Table 3: Result - Basic Block Level

方法拓展思考:

1. CFG → 其他图

2. 无权图 → 加权图

1. 程序切片技术
2. 基于程序特征谱的故障定位
3. 故障上下文定位
4. 测试用例约简



## 2.4 测试用例约简

### ■ 测试用例约简

✓使用场景：回归测试、软件故障定位等

✓意义：提高测试用例的执行效率

✓重点：软件故障定位领域的测试用例约简技术

目标：不降低(提高)软件故障定位准确性的同时  
约简测试用例数量



## 2.4 测试用例约简

### ■ 基于覆盖的测试用例约简

**核心思想：**选择部分测试用例，使得约简后的用例集与原用例集保相同的语句/分支/MCDC覆盖率。

int mid() { int x, y, z, m;  1: read ( x, y, z ); 2: m = z ; 3: if ( y < z ) 4:   if ( x < y ) 5:     m = y ; 6:   else if ( x < z ) 7:     m = y ;   /**bug** 8: else 9:   if ( x > y ) 10:    m = y ; 11:   else if ( x > z ) 12:    m = x ; 13: return   m ; }	测试用例							
	t <sub>1</sub>	t <sub>2</sub>	t <sub>3</sub>	t <sub>4</sub>	t <sub>5</sub>	t <sub>6</sub>	t <sub>7</sub>	t <sub>8</sub>
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	7,5,4	2,1,3	4,3,5
1: read ( x, y, z );	●	●	●	●	●	●	●	●
2: m = z ;	●	●	●	●	●	●	●	●
3: if ( y < z )	●	●	●	●	●	●	●	●
4:   if ( x < y )	●	●			●		●	●
5:     m = y ;		●						
6:   else if ( x < z )	●				●		●	●
7:     m = y ;   /**bug**	●						●	●
8: else			●	●		●		
9:   if ( x > y )			●	●		●		
10:    m = y ;			●			●		
11:   else if ( x > z )				●				
12:    m = x ;								
13: return   m ;	●	●	●	●	●	●	●	●
Pass / Fail Status	P	P	P	P	P	P	F	F

1. S5仅被t2覆盖，所以t2保留
2. S11仅被t4覆盖，所以t4保留
3. 覆盖完全相同可以的仅保留1个，如t1, t7, t8
4. 其余的可按不同策略进行保留

也可将pass和fail的  
分组进行约简

## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简

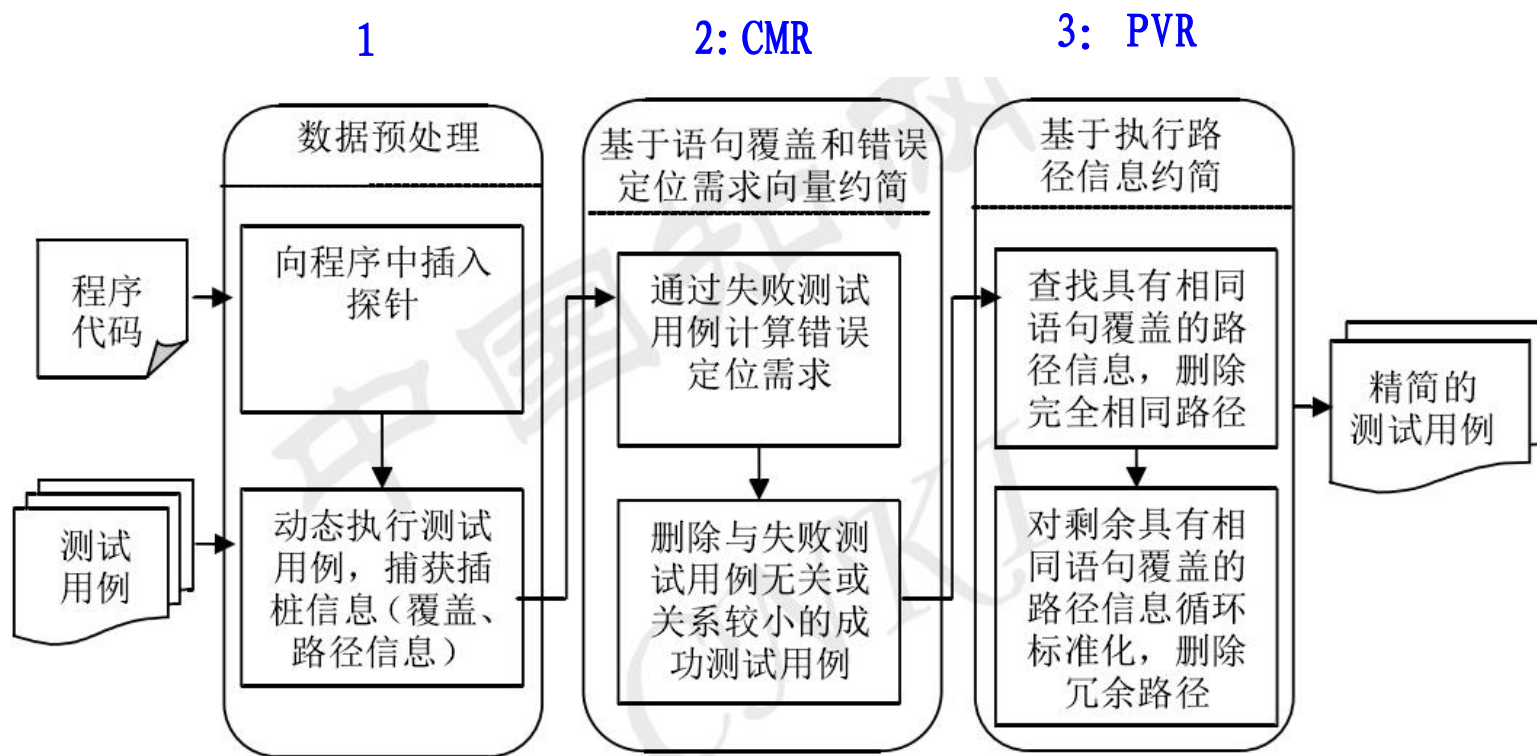


图 2-1 面向错误定位需求的测试用例约简流程

引自：结合测试用例约简和概率图建模的软件错误定位方法研究，博士论文，2014，哈工大，龚丹丹

## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简

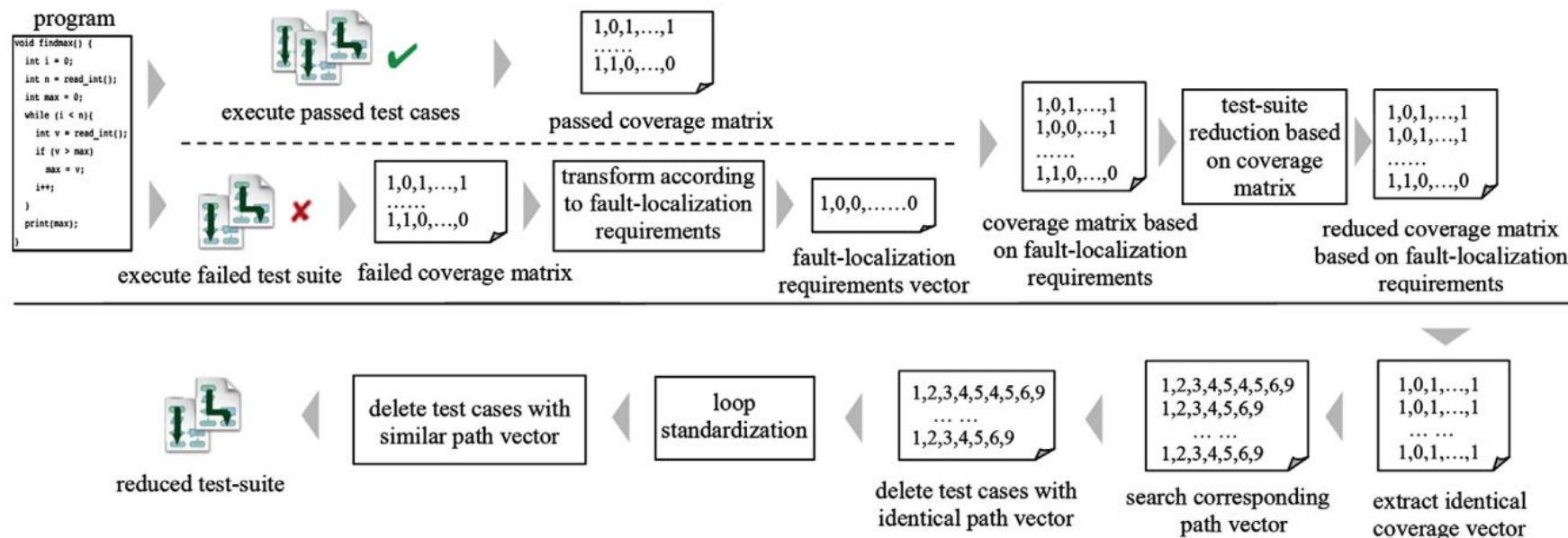


Fig. 1. The framework of two-step test-suite reduction.

引自: A test-suite reduction approach to improving fault-localization effectiveness, 2013,  
Comput. Lang. Syst. Struct

错误定位需求:

## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - 例子

<pre>mid() {   int x, y, z, m;</pre>	Test Cases							
	t <sub>1</sub>	t <sub>2</sub>	t <sub>3</sub>	t <sub>4</sub>	t <sub>5</sub>	t <sub>6</sub>	t <sub>7</sub>	t <sub>8</sub>
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	7,5,4	2,1,3	4,3,5
1: read ( x, y, z );	●	●	●	●	●	●	●	●
2: m = z ;	●	●	●	●	●	●	●	●
3: if ( y < z )	●	●	●	●	●	●	●	●
4:   if ( x < y )	●	●			●		●	●
5:     m = y ;		●						
6:   else if ( x < z )	●				●		●	●
7:     m = y ;   /**bug**	●						●	●
8: else			●	●		●		
9:   if ( x > y )			●	●		●		
10:     m = y ;			●			●		
11:   else if ( x > z )				●				
12:     m = x ;								
13: printf ( m ) ;	●	●	●	●	●	●	●	●
} Pass / Fail Status	P	P	P	P	P	P	F	F

Fig. 2. Example program and its test cases.

## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - CMR

#### 1. 构建覆盖matrix, 消除弱相关列 (语句)

	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$	$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$	$s_{11}$	$s_{12}$	$s_{13}$	
$COVER(t_1)$	1	1	1	1	0	1	1	0	0	0	0	0	1	$\rightarrow RCOV(t_1)$
$COVER(t_2)$	1	1	1	1	1	0	0	0	0	0	0	0	1	$\rightarrow RCOV(t_2)$
$COVER(t_3)$	1	1	1	0	0	0	0	1	1	1	0	0	1	$\rightarrow RCOV(t_3)$
$COVER(t_4)$	1	1	1	0	0	0	0	1	1	0	1	0	1	$\rightarrow RCOV(t_4)$
$COVER(t_5)$	1	1	1	1	0	1	0	0	0	0	0	0	1	$\rightarrow RCOV(t_5)$
$COVER(t_6)$	1	1	1	0	0	0	0	1	1	1	0	0	1	$\rightarrow RCOV(t_6)$
$COVER(t_7)$	1	1	1	1	0	1	1	0	0	0	0	0	1	$\rightarrow RCOV(t_7)$
$COVER(t_8)$	1	1	1	1	0	1	1	0	0	0	0	0	1	$\rightarrow RCOV(t_8)$

弱相关语句
剩余覆盖向量
弱相关语句

覆盖向量



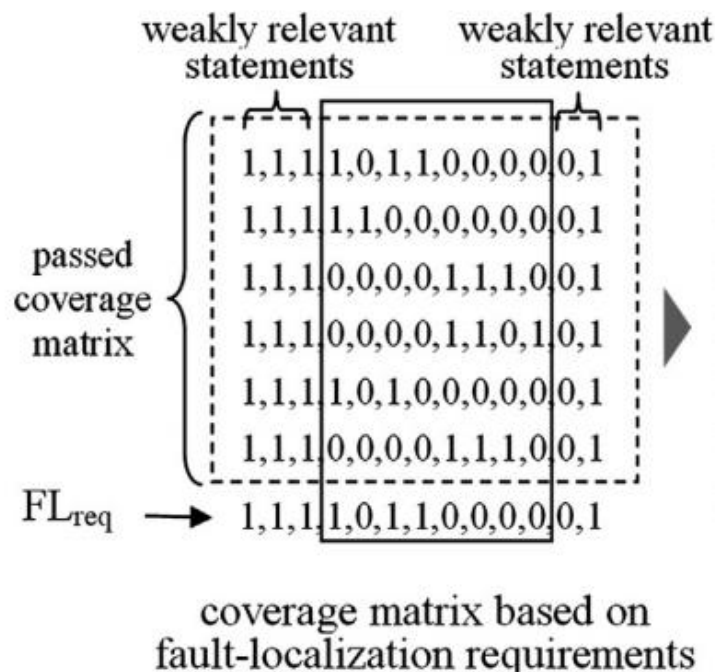
## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - CMR

#### 2. 计算错误定位需求向量 $FL_{req}$ ，并构成新矩阵

$FL_{req}$ : 单错误时，多个用例的覆盖结合交集  
多错误时，多个用例的覆盖结合的并集

T7和T8: fail,  
且覆盖完全相同



## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - CMR

#### 3. 删除弱相关行，构造最精简的覆盖矩阵

**弱相关行**：将每行与 $FL_{req}$ 相与，向量内积为0时，为弱相关，可以删除该用例

例如：t1:  $\langle 1, 0, 1, 0, 1 \rangle$   
 $FL_{req}$ :  $\langle 0, 1, 0, 1, 0 \rangle$

则  $t1 * FL_{req} = \langle 0, 0, 0, 0, 0 \rangle$ ，称t1与 $FL_{req}$ ，即说明t1与 $FL_{req}$ 关系不大

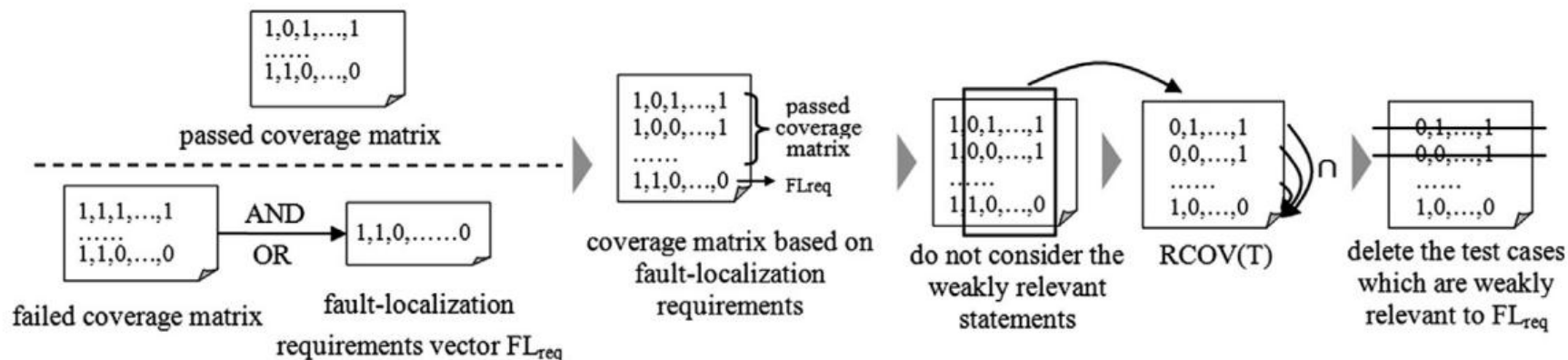


Fig. 4. The model of coverage matrix based reduction.



## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - PVR

1. 从CMR得到的矩阵中，抽取完全相同的向量，找出其对应的path  
(重点研究覆盖相同，Path不同的情况，如下面的t1-t5)
2. 删除完全相同的path
3. 进行循环标准化，判断相似路径，仅保留一组相似路径中的一个

4 for .....			
5 .....	PATH(t <sub>1</sub> )=<4,5,6,7,4,5,6,7,4,5,8,9>	PATH'(t <sub>1</sub> )=<4,5,6,7,4,5,8,9> series:4,5,6,7 times:2	} similar paths
6 if (1)	PATH(t <sub>2</sub> )=<4,5,6,7,4,5,8,9>	PATH'(t <sub>2</sub> )=<4,5,6,7,4,5,8,9>	
7 .....	PATH(t <sub>3</sub> )=<4,5,6,7,4,5,8,9,4,5,8,9>	PATH'(t <sub>3</sub> )=<4,5,6,7,4,5,8,9> series:4,5,8,9 times:2	
8 if (2)	PATH(t <sub>4</sub> )=<4,5,8,9,4,5,6,7>	} identical paths	
9 .....	PATH(t <sub>5</sub> )=<4,5,8,9,4,5,6,7>		

Fig. 7. The example of path vector based reduction approach.

## 2.4 测试用例约简

### ■ 面向错误定位需求的测试用例约简 - 实验分析

#### ✓ 度量:

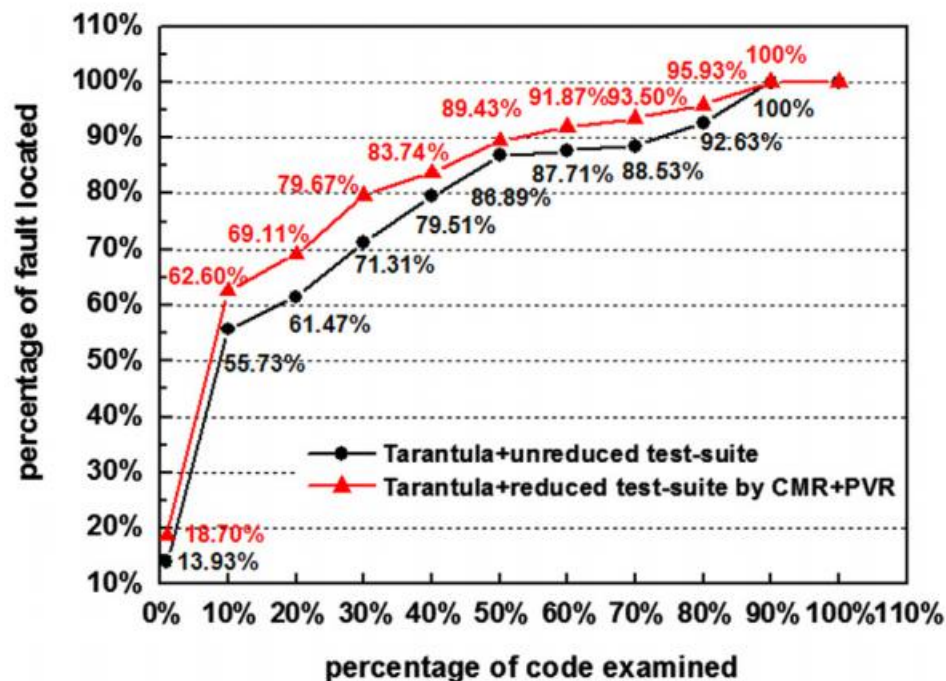
- 测试用例约简比率

$$Reduction = \left( 1 - \frac{\text{size of reduced test - suite}}{\text{size of unreduced test - suite}} \right) \times 100\%$$

- 故障定位准确性变化率

$$EffectivenessChange = \frac{\text{UnreducedRank} - \text{ReducedRank}}{\text{number of executable statements}} \times 100\%$$

#### ✓ 实验结果:



## ■ An Empirical Study of Fault Localization Families (TSE2019)

### 智能软件的软件质量保证（设计、测试等）

- Software Engineering for Machine Learning
- Metamorphic Testing
- Deep learning
- Large-scale empirical study on machine learning related questions on stack overflow
- 微软: <https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview>
- ...

---

**业精于勤，荒于嬉！**

---