错误定位方法 / Fault Localization (FL)

感谢 张震宇等老师提供支持

错误定位

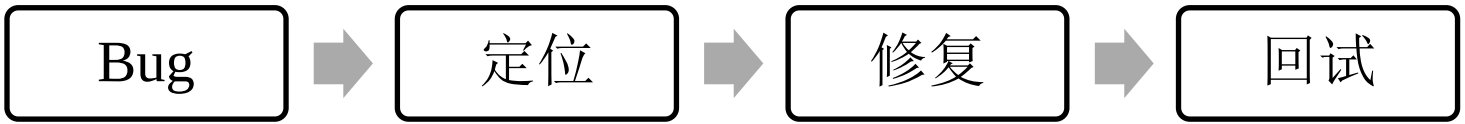
- 有错了，怎么办

```
require 'rubygems'
require 'atchoum'
class SneezyWeb < Atchoum::Website
  def layout
    html do
      head do
        title 'I sneeze'
      end
      body do
        self << yield
      end
    end
  end
  def poem_page
    h1 'The allergies season'
    p 'Flowers are red'
    p 'Buds open'
    p 'I sneeze a lot!'
  end
end
```

Bug/Fault

错误定位

- 软件调试（Debugging）的一般流程



一般认为，软件测试开销巨大。其中，**错误定位被**认为是最难的一个环节。

错误定位

- 程序“错误”的描述语

- **Error**: An error is an internal state of a program that deviates from the expected behavior of the program.
- **Fault/Bug/Flaw**: Fault is set of statements which their execution may perturb the state of program to error state. In other words, it may infects the state of program.
- **Failure**: Failure is manifestation of error to an external entity (i.e. oracle) which discern the program failure. In this case we say the program fails.

- **Error 程序运行时的内部状态**
- **Fault 或Bug是会引起Error的源码**
- **Failure 是Fault造成的观察到的结果**

错误定位

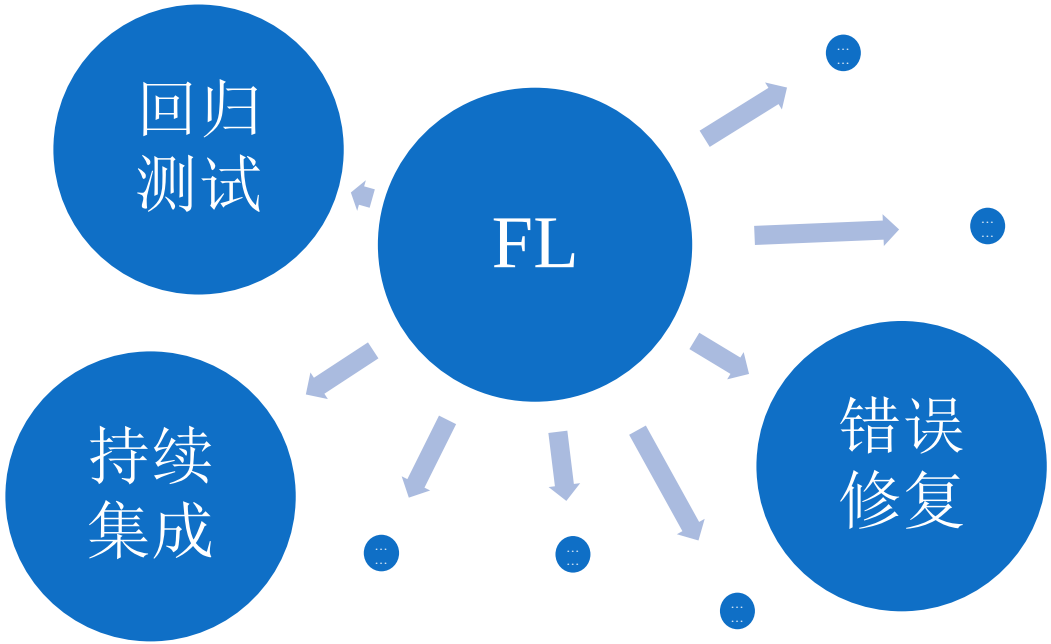
- 查找错误相关的模块

- 准确查找错误相关的语句

- 定位到错误的位置

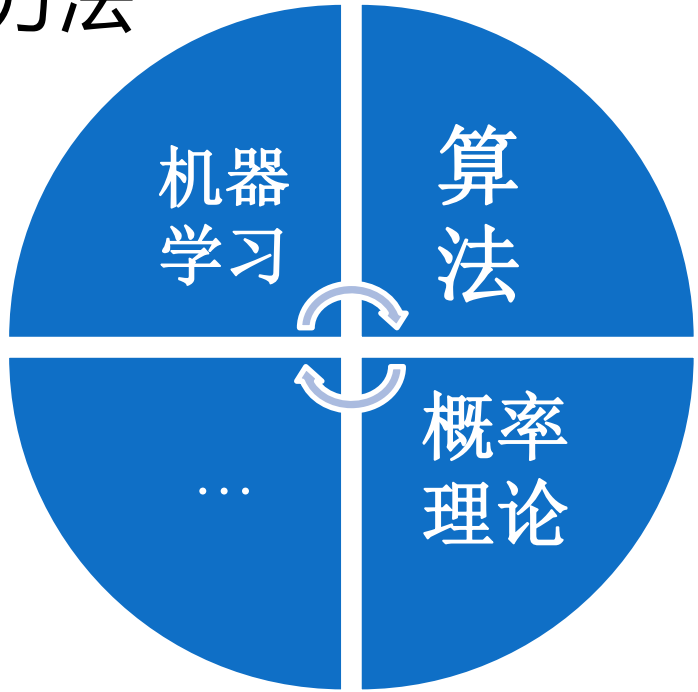
错误定位

- 关系



错误定位

- 相关的方法



相关研究分布

- 软件工程
- 编程语言
- 体系结构
- 人工智能

错误定位

- 错误定位方法
  - –Slice-Based Techniques
  - –**Program Spectrum-Based Techniques**
  - –Statistics-Based Techniques
  - –Program State-Based Techniques
  - –Machine Learning-Based Techniques
  - –Data Mining-Based Techniques
  - –Model-Based Techniques
  - –…

错误定位

- Program Spectrum-Based Techniques（基于频谱）
  –测试用例 + 疑似度公式

例子

		T15	T16	T17	T18
1	int count; int n; Ele *proc; List *src_queue, *dest_queue; if (prio >= MAXPRIO) /*maxprio=3*/	●	●	●	●
2	{return;}	●			
3	src_queue = prio_queue[prio]; dest_queue = prio_queue[prio+1]; count = src_queue->mem_count; *if (count > 1) /* Bug*//* supposed : count>0*/ {*		●	●	●
4	n = (int) (count*ratio + 1); proc = find_nth(src_queue, n); if (proc) {		●	●	
5	src_queue = del_ele(src_queue, proc); proc->priority = prio; dest_queue = append_ele(dest_queue, proc); }		●	●	
	Pass/Fail of Test Case Execution :	Pass	Pass	Pass	Fail

计算程序元素的怀疑度

- a_{ef}：执行语句a的失败测试的数量，a_{nf}：未执行语句a的失败测试的数量
- a_{ep}：执行语句a的通过测试的数量，a_{np}：未执行语句a的通过测试的数量
- Tarantula: $\frac{a_{ef}}{a_{ef}+a_{nf}} / \left(\frac{a_{ef}}{a_{ef}+a_{nf}} + \frac{a_{ep}}{a_{ep}+a_{np}}\right)$
- Jaccard: $\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$
- Ochiai: $\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})(a_{ef}+a_{ep})}}$
- D*: $\frac{a_{ef}^{*}}{a_{nf}+a_{ep}}$，*通常设置为2或者3
- Naish1: $\begin{cases} -1 & a_{nf} > 0 \\ a_{np} & a_{nf} = 0 \end{cases}$

https://xiongyingfei.github.io/SA/2020/19_fault_localization.pdf

错误定位

- 关键点

程序中的 Faults (root cause)一般并不和failure强相关

1、Fault 一般隐藏很深
2、统计定位中噪音大量存在
3、是否有最好的基于统计的比较方法？
4、Faults并非一定会导致failure

错误定位

- 关键点

程序中的 Faults (root cause)一般并不和failure强相关

1、Fault 一般隐藏很深

2、统计定位中噪音大量存在

3、是否有最好的基于统计的比较方法？

4、Faults并非一定会导致failure

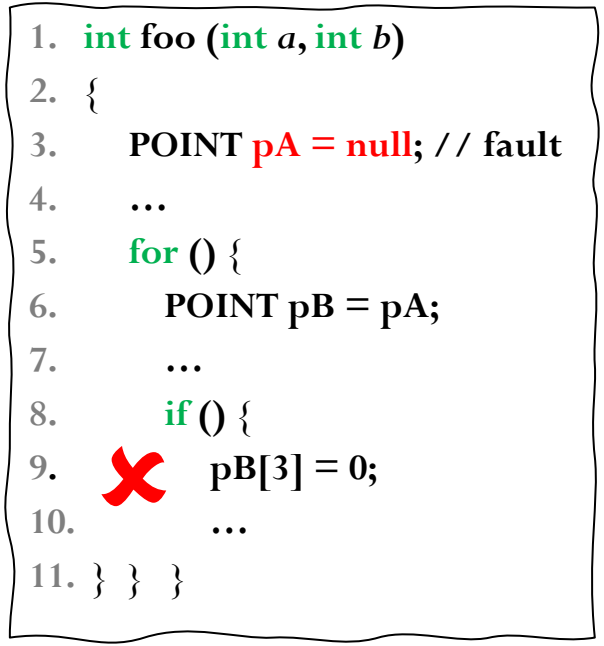
错误定位

- 错误隐藏

A pointer assigned to NULL

Another assignment

An invalid memory access



错误定位

- 错误隐藏

	Program states			
	Start	pA : N\A	pB : N\A	…
infect	Line 3	pA : null	pB : N\A	…
↓				
infect	Line 6	pA : null	pB : null	…
↓				
crash	Line 9	pA : null	pB : null	…

```
1.  int foo (int a, int b)
2.  {
3.      POINT pA = null; // fault
4.      …
5.      for () {
6.          POINT pB = pA;
7.          …
8.          if () {
9.              pB[3] = 0;
10.             …
11. } } }
```

错误定位

- 错误隐藏

Executing Line 3 may not cause a crash

Executing Line 9 always cause a crash

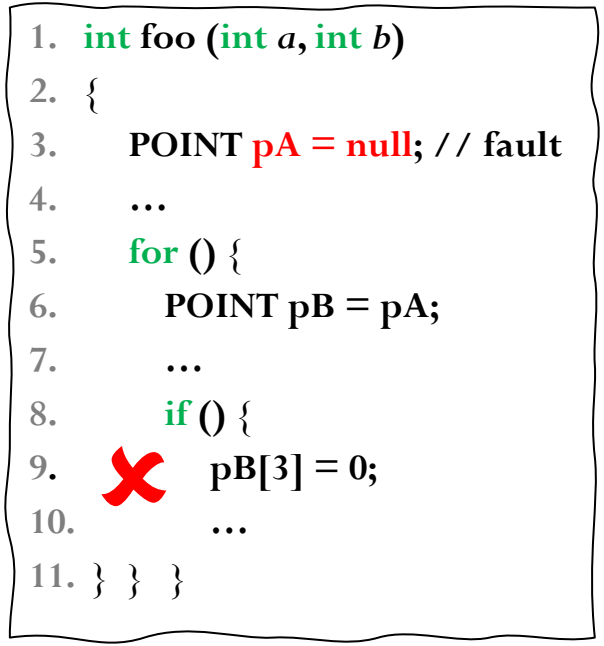
```
1.  int foo (int a, int b)
2.  {
3.      POINT pA = null; // fault
4.      …
5.      for () {
6.          POINT pB = pA;
7.          …
8.          if () {
9.              pB[3] = 0;
10.             …
11. } } }
```

错误定位

- 错误隐藏

Line 3 may weakly correlate to program failures

Line 9 strongly correlates to program failures



错误定位

- 如何定位

Locating statements correlating to failures will mislead us to Line 9 and miss Line 3

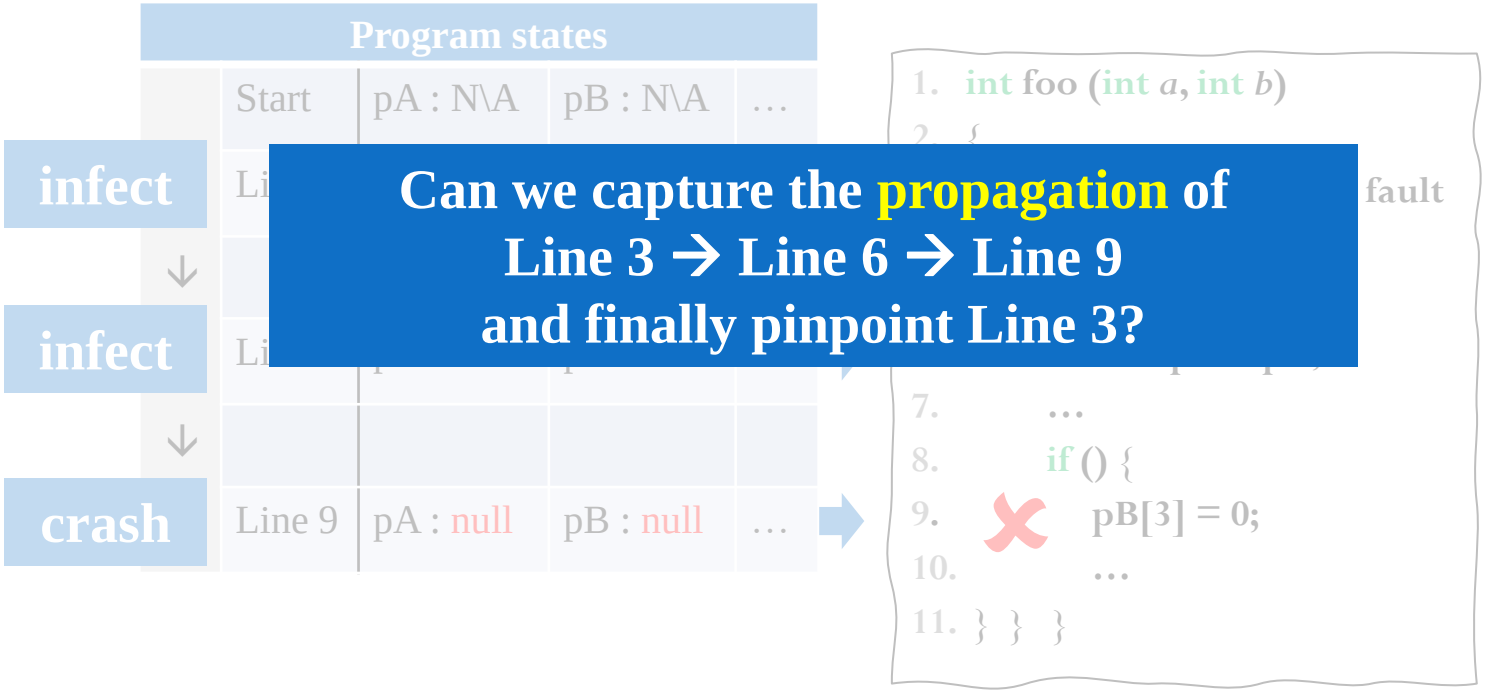
Strong correlation to failures ≠ faulty statements

How to pinpoint Line 3?

```
1.  int foo (int a, int b)
2.  {
3.      POINT pA = null; // fault
4.      …
5.      for () {
6.          POINT pB = pA;
7.          …
8.          if () {
9.              pB[3] = 0;
10.             …
11. } } }
```

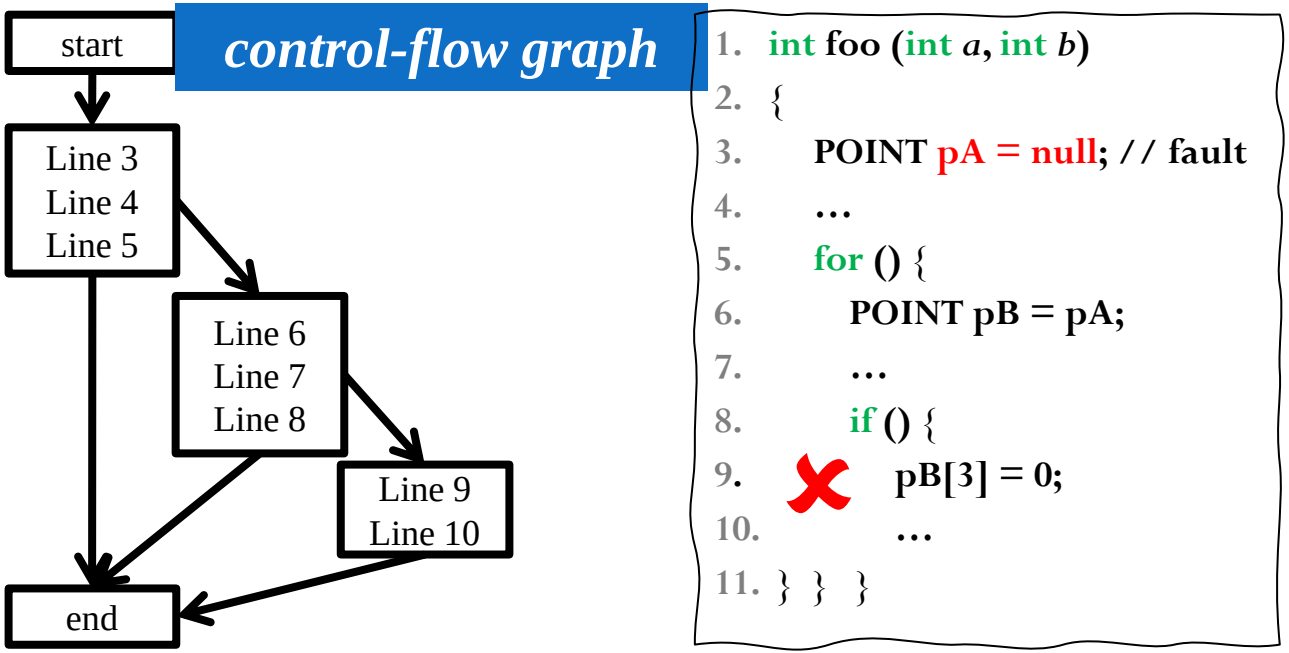
错误定位

- 如何定位


Program states
Start
pA : N\A
pB : N\A
…
infect
infect
crash
Line 9
pA : null
pB : null
…
Can we capture the propagation of
Line 3 → Line 6 → Line 9
and finally pinpoint Line 3?
1. int foo (int a, int b)
fault
7. …
8. if () {
9. pB[3] = 0;
10. …
11. } } }

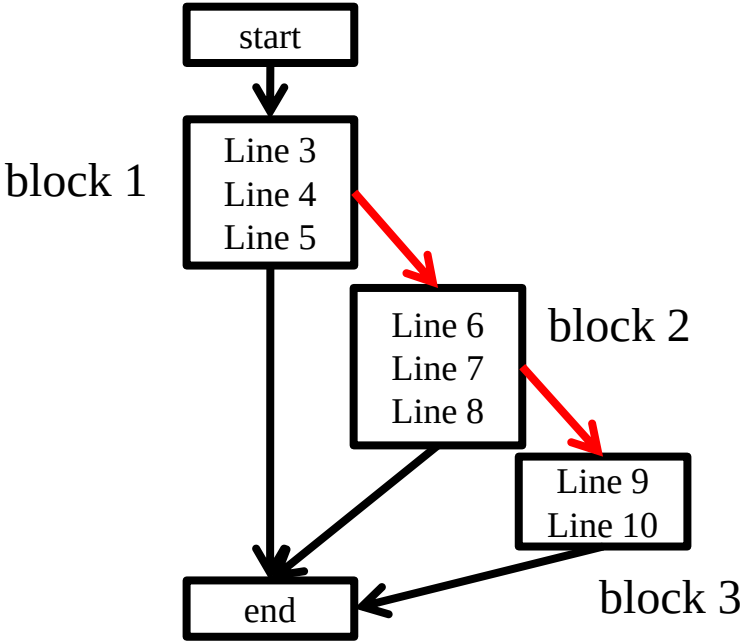
错误定位

- 基于控制流图


start
control-flow graph
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
end
1. int foo (int a, int b)
2. {
3. POINT pA = null; // fault
4. …
5. for () {
6. POINT pB = pA;
7. …
8. if () {
9. pB[3] = 0;
10. …
11. } } }

错误定位

- 基于控制流图
  - –Why use CFG



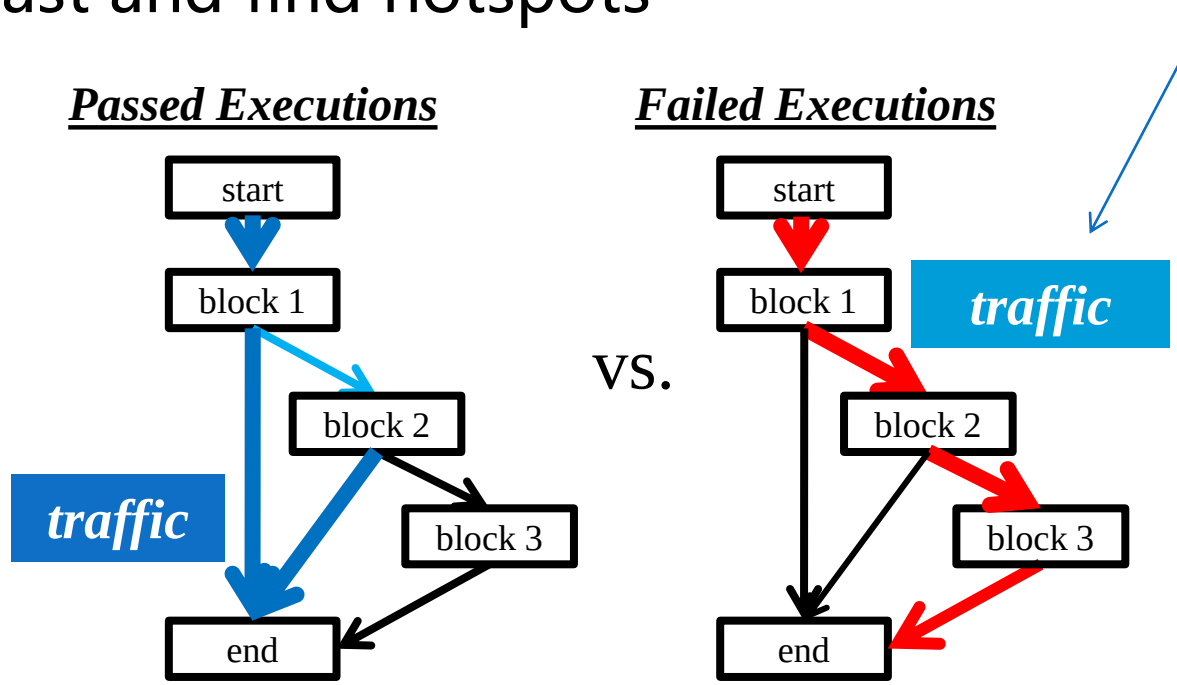
The infected program states are propagating via edges

```
1.  int foo (int a, int b)
2.  {
3.      POINT pA = null; // fault
4.      …
5.      for () {
6.          POINT pB = pA;
7.          …
8.          if () {
9.              pB[3] = 0;
10.             …
11. } } }
```

错误定位

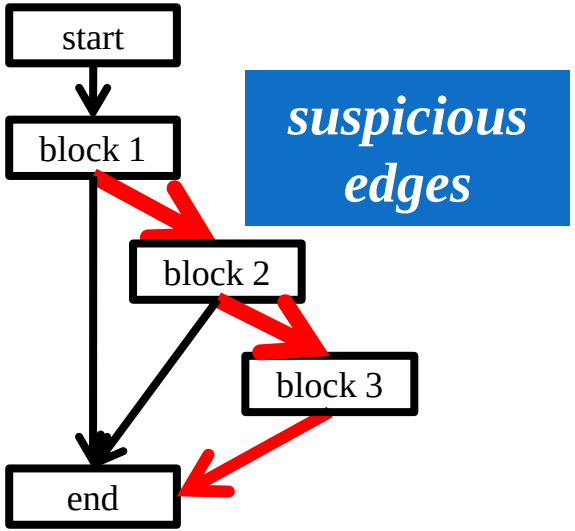
- 基于控制流图
  - –Contrast and find hotspots

Execution frequency of every edge in program execution



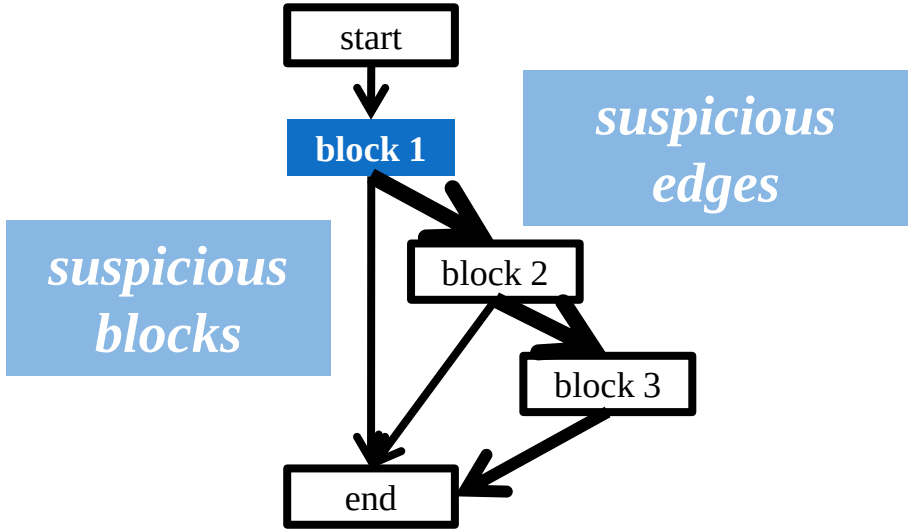
错误定位

- 简单算法
  - –We find the suspicious edges,



错误定位

- 简单算法
  - –We find the suspicious edges,
  - –Then, map to suspicious blocks



错误定位

- 简单算法
  - –Collect the traffic via each CFG edge
    - For passed test case side

$$\theta^{pass}(e): \text{the mean traffic via edge } e, \text{ by averaging all passed test cases}$$

    - E.g.,
      - –T1 executes edge *e* for 3 times
      - –T2 executes edge *e* for 7 times
      - –T3 executes edge *e* for 5 times

$$\theta^{pass}(e) = (3+7+5)/3 = 5$$

错误定位

- 简单算法
  - –Collect the traffic via each CFG edge
    - For passed test case side

$$\theta^{pass}(e): \text{the mean traffic via edge } e, \text{ by averaging all passed test cases}$$

    - For failed test case side

$$\theta^{fail}(e): \text{the mean traffic via edge } e, \text{ by averaging all failed test cases}$$

错误定位

- 简单算法
  - –Compute suspiciousness for each CFG edge
    - For each CFG edge

$$\theta^{delta}(e) = \frac{u \cdot \theta^{fail}(e)}{v \cdot \theta^{fail}(e) + u \cdot \theta^{pass}(e)}$$

of passed test cases

of failed test cases

    - Physical meaning:
      Maximum Likelihood of "passing through *e* causes failure"

错误定位

- 问题
  - –Since we need to map suspicious edges to suspicious blocks
    - What if one block has two edges?
  - –Since we use the edges and blocks of control-flow graph
    - What if loops exist in a control-flow graph?

错误定位

- 关键点

程序中的 Faults (root cause)一般并不和failure强相关

1、Fault 一般隐藏很深

2、统计定位中噪音大量存在

3、是否有最好的基于统计的比较方法？

4、Faults并非一定会导致failure

错误定位

- 举例

```
Program: sort.cpp
L1: void main(int argc, char** argv) {
L2:     int i=atoi(argv[1]);
L3:     int j=atoi(argv[2]);
L4:     if(i<j)
L5:         cout<<i<<" "<<j;
L6:     if(i>j || i=j)          /* faulty statement */
L7:         cout<<j<<" "<<i;
L8: }
```

Should be `i==j`

错误定位

- 举例 – short-circuiting rule
  - –Predicate: `i>j || i=j`
  - –Short-circuiting:
    When `i>j` is evaluated as True, `i=j` will not be executed

错误定位

- Not all Boolean expressions will be executed
  - Failed run T1: (input: 2 3 output: 2 3 3 3)

```
if(i<j)
    cout<<i<<" "<<j;
if(i>j || i=j)
    cout<<j<<" "<<i;
```

L6: if(i >j | | i=j)			
	False	True	Both i>j and i=j are executed

  - Passed run T3: (input: 3 2 output: 2 3)

L6: if(i>j | | i=j)			
	True	---	Only i>j is executed

- We count execution for statement, while actually some of its Boolean expressions are executed
- We want to distinguish executing i>j || i=j from executing i>j

错误定位

- Not all Boolean expressions will be executed
  - Failed run T1: (input: 2 3 output: 2 3 3 3)

L6:	if(	i>j	||	i=j	)
		False	True	Both i>j and i=j are executed	

  - Passed run T3: (input: 3 2 output: 2 3)

L6:	if(	i>j	||	i=j	)
		True	---	Only i>j is executed	

```
if(i<j)
    cout<<i<<" "<<j;
if(i>j || i=j)
    cout<<j<<" "<<i;
```

- We count execution for statement, while actually some of its Boolean expressions are executed
- We want to distinguish executing i>j || i=j from executing i>j

错误定位

- Not all Boolean expressions will be executed
  - Failed run T1: (input: 2 3 output: 2 3 3 3)

L6:	if(	i>j	||	i=j	)	
		False		True		Both i>j and i=j are executed

  - Passed run T3: (input: 3 2 output: 2 3)

L6:	if(	i>j	||	i=j	)	
		True		---		Only i>j is executed

```
if(i<j)
    cout<<i<<" "<<j;
if(i>j || i=j)
    cout<<j<<" "<<i;
```

- We count execution for statement, while actually some of its Boolean expressions are executed

- We want to distinguish executing i>j || i=j from executing i>j

错误定位

- Motivation
  - –fault-relevant statement is located

		Passed runs					Failed runs			
		T1	T2	T3	T4		T5	T6	T7	T8
L1	void main(…) {	0	0	0	0	←resemble→	0	0	0	0
L2	int i=atoi(argv[1]);	1	1	1	1	←resemble→	1	1	1	1
L3	int j=atoi(argv[2]);	1	1	1	1	←resemble→	1	1	1	1
L4	if(i<j)	1	1	1	1	←resemble→	1	1	1	1
L5	cout<<i<<“ ”<<j;	1	0	1	0	←resemble→	1	0	1	0
L6 (faulty)	if(i>j || i=j)	1	1	1	1	←resemble→	1	1	1	1
L7 (fault-relevant)	cout<<j<<“ ”<<i;	0	1	0	1	**←different→**	1	1	1	1
L8	}	0	0	0	0	←resemble→	0	0	0	0

错误定位

- Motivation

		Passed runs					Failed runs			
		T1	T2	T3	T4		T5	T6	T7	T8
L1	void main(…) {	0	0	0	0	←resemble→	0	0	0	0
L2	int i=atoi(argv[1]);	1	1	1	1	←resemble→	1	1	1	1
L3	int j=atoi(argv[2]);	1	1	1	1	←resemble→	1	1	1	1
L4	if(i<j)	1	1	1	1	←resemble→	1	1	1	1
L5	cout<<i<<“ ”<<j;	1	0	1	0	←resemble→	1	0	1	0
L6	i>j || i=j	0	0	0	0	**←very different→**	1	1	1	1
	i>j	1	1	1	1	**←very different→**	0	0	0	0
L7	cout<<j<<“ ”<<i;	0	1	0	1	**←different→**	1	1	1	1
L8	}	0	0	0	0	←resemble→	0	0	0	0

L6: if(i>j || i=j)

错误定位

- Evaluation sequence (求值序列)

Predicate: i>j || i=j				
	Input:	i>j	i=j	Evaluation Sequence:
Case 1	Input = 2 3	Evaluated as False	Evaluated as True	<False, True>
Case 2	Input = 0 0	Evaluated as False	Evaluated as False	<False, False>
Case 3	Input = 3 2	Evaluated as True	Short-circuited	<True>

错误定位

- Evaluation sequence (求值序列)
  - –Any evaluation on this predicate must fall into one of the three evaluation sequences
  - –Use **evaluation sequence** as unit of investigation
  - –Advantages
    - More scientific
    - Finer granularity, more precise
    - Reduce the chance of mislead

错误定位

- 关键点

程序中的 Faults (root cause)一般并不和failure强相关

1、Fault 一般隐藏很深

2、统计定位中噪音大量存在

3、是否有最好的基于统计的比较方法？

4、Faults并非一定会导致failure

错误定位

- 关于错误定位的一些思考

程序中的 Faults (root cause)一般并不和failure强相关

1、Fault 一般隐藏很深
2、统计定位中噪音大量存在
3、是否有最好的基于统计的比较方法？
4、Faults并非一定会导致failure

Q & A

问答