

# Constraint Solving in Program Repair

## 程序修复中的约束求解

玄跻峰

武汉大学 软件工程国家重点实验室

**Email:** [jxuan@whu.edu.cn](mailto:jxuan@whu.edu.cn)

**URL:** <http://cstar.whu.edu.cn/>



# 玄济峰

jxuan@whu.edu.cn

<http://cstar.whu.edu.cn/>

教授，博士生导师

## 经历

2015年至今， 武汉大学

2013至2015年， 法国国立计算机研究院 (INRIA) 博士后研究员

2007和2013年， 大连理工大学获本科和博士学位

## 研究方向 - 软件测试与分析

软件测试调试、软件数据分析、基于搜索的软件工程



内容概要

研究背景

代表成果

相关工作

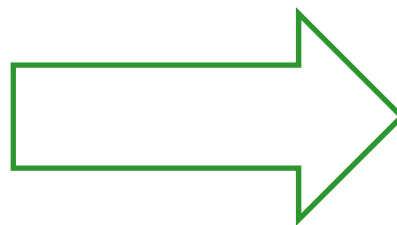
未来挑战



# 从Bug到Debug



软件程序中的缺陷

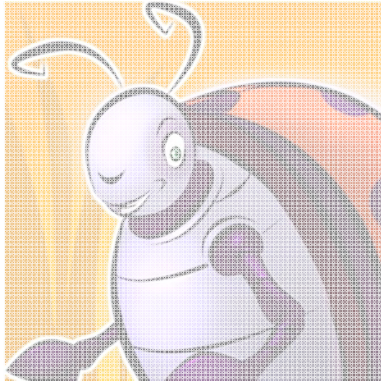


消灭缺陷



测试用例 – Test Case

# 从Bug到Debug



软件程序中的缺陷

## 测试用例

```
@Test  
public void testFoo ( )  
{  
    assert a.parent != null;  
    ...  
    assert a + b == 3 ;  
}
```

软件测试、软件调试、程序理解、软件维护、软件分析 ...

知晓  
缺陷

找到  
缺陷

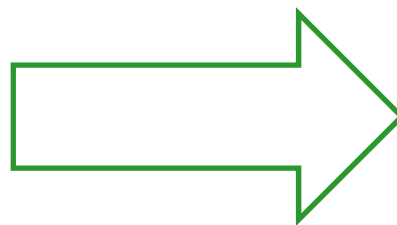
分析  
缺陷

测试用例 – Test Case

# 从Bug到Debug



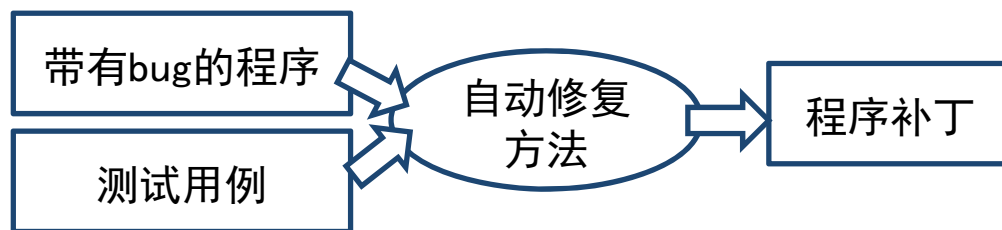
软件程序中的缺陷



消灭缺陷



# 基于测试的程序修复



# 背景 – 基于测试的程序修复

输入: Buggy Program + Test Cases

输出: Code Patch

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    ...  
}
```

Bug

Patch

Bug Math-238, Apache Commons Math, 2009





## 背景 – 基于测试的程序修复

### 困境

- 在大型程序中，如何找到缺陷位置？
- 若找到缺陷位置，如何生成补丁？
- 若生成补丁，如何确认补丁正确？

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    ...  
}
```

Bug

Patch

Bug Math-238, Apache Commons Math, 2009



# 背景 – 基于测试的程序修复

输入: Buggy Program + Test Cases

输出: Code Patch

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    }  
    ...  
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
...	...	...	...	...



# 背景 – 基于测试的程序修复

输入: Buggy

输出: Code

## 基于测试的程序修复

已知程序和测试用例集，能否自动生成代码补丁，并通过全部测试集用例？

计算u与v的最大公约数

```
int gcd (int u, int v) {
```

```
-   if (u * v == 0) {
```

```
+   if (u == 0 || v == 0) {
```

```
    return (Math.abs(u) + Math.abs(v));
```

```
...
```

```
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
...	...	...	...	...



# 背景 – 基于测试的程序修复

输入: Buggy Program + Test Cases

输出: Code Patch

应用场景:

测试驱动开发 (Test-Driven Development)

持续集成 (Continuous Integration)

回归测试 (Regression Testing)

...



MENU FORTUNE SUBSCRIBE

# Researchers create "skin grafts"

software



业界资讯 Win10之家 iPhone之家 安卓之家  
软件之家 WP之家 iPad之家 数码之家

首页 > IT资讯 > 网络

## MIT黑科技：任何程序都能自动修复BUG

2015-6-30 13:29:15 来源：威锋网 作者：TangoDown 责编：弥尘

搜狐 首页

用户名邮箱/手机号

登录

注册

if (!(((length MIT黑科技：自动修复BUG

PLDI 2015

## Automatic Error Elimination by Horizontal Code Transfer Across Multiple Applications

Stelios Sidiroglou-Douskos Eric Lahtinen Fan Long Martin Rinard

与计算机科学的很多分支不同，  
软件工程研究处理人类不易完成的任务



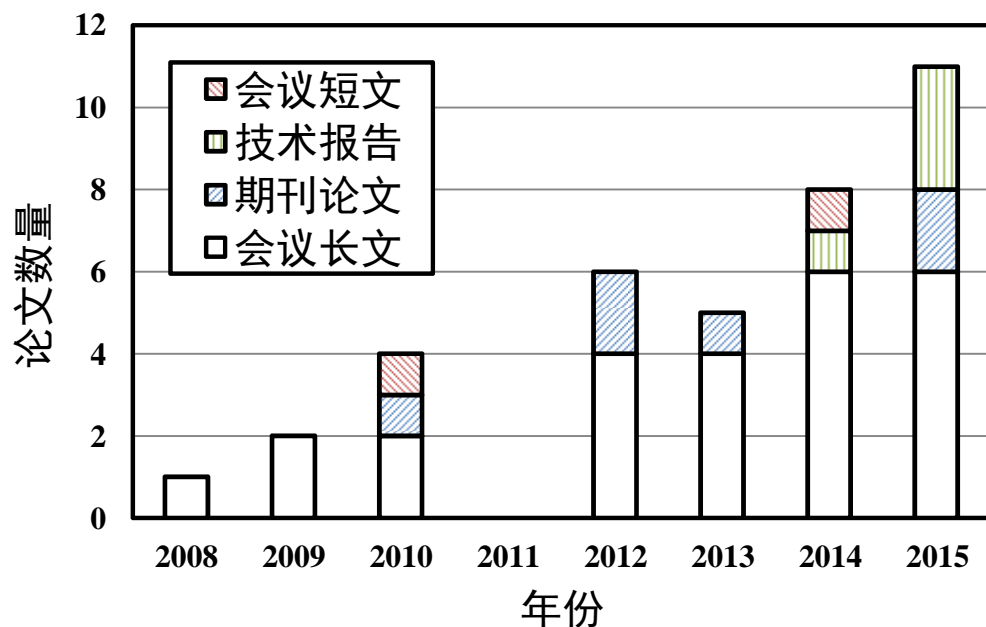
A team that fixes bugs by applying skin grafts to source code.



# 缺陷修复断代 “十年史”

## 研究历史

- 早期工作，2008年国际同行提出基于演化计算的自动修复算法
- 至今10年时间，发展了大量的高水平成果



玄跻峰 等. 自动程序修复方法研究进展. 软件学报, 27 (4), 2016.

类型	缩写	文章数
国际会议	ICSE	8
	FSE	4
	ISSTA	3
	ASE	3
	ICSM	3
	ICST	1
	GECCO	3
	CSTVA	1
	CEC	1
	TSE	2
国际期刊	CACM	1
	ESE	1
	SQJ	1
	SCIS	1
	TR	4
	技术报告	4
	合计	37



# 基于遗传规划的缺陷修复

基于遗传规划的修复, GenProg, by Weimer and Le Goues, ICSE 2009, TSE 2012.

程序转换为  
抽象语法树



执行测试用例  
查找bug可能的位置,  
用于指导变异



应用遗传规划  
生成新的  
语法树

重用已有的代码,  
变换位置和关系,  
生成新的代码

```
- if (u * v == 0) {  
+ if (u == 0 || v == 0) {  
    return (Math.abs(u) + Math.abs(v));  
}
```

...

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
2^30	2^29	2^29	2^30+2^29	Fail
...	...	...	...	...



# Example: GCD

```
/* requires: a >= 0, b >= 0 */
void print_gcd(int a, int b) {
    if (a == 0)
        printf("%d", b);
    while (b != 0) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    printf("%d", a);
    return;
}
```

Bug: when  
a==0 and b>0,  
it loops forever!

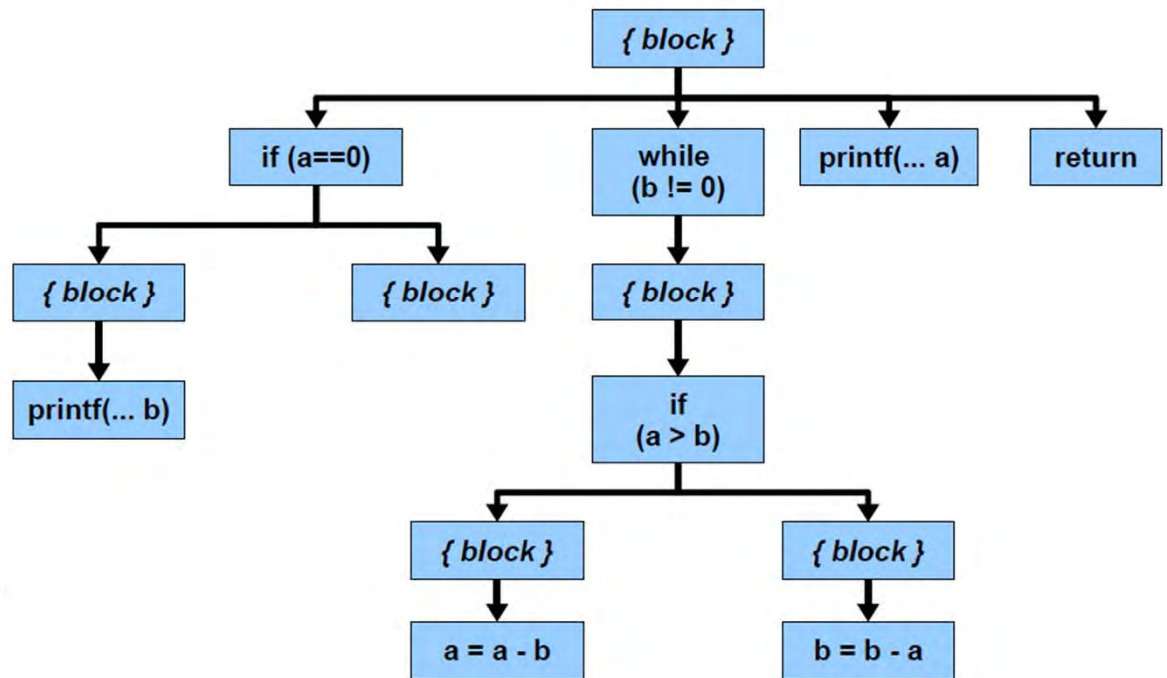




# Example: GCD

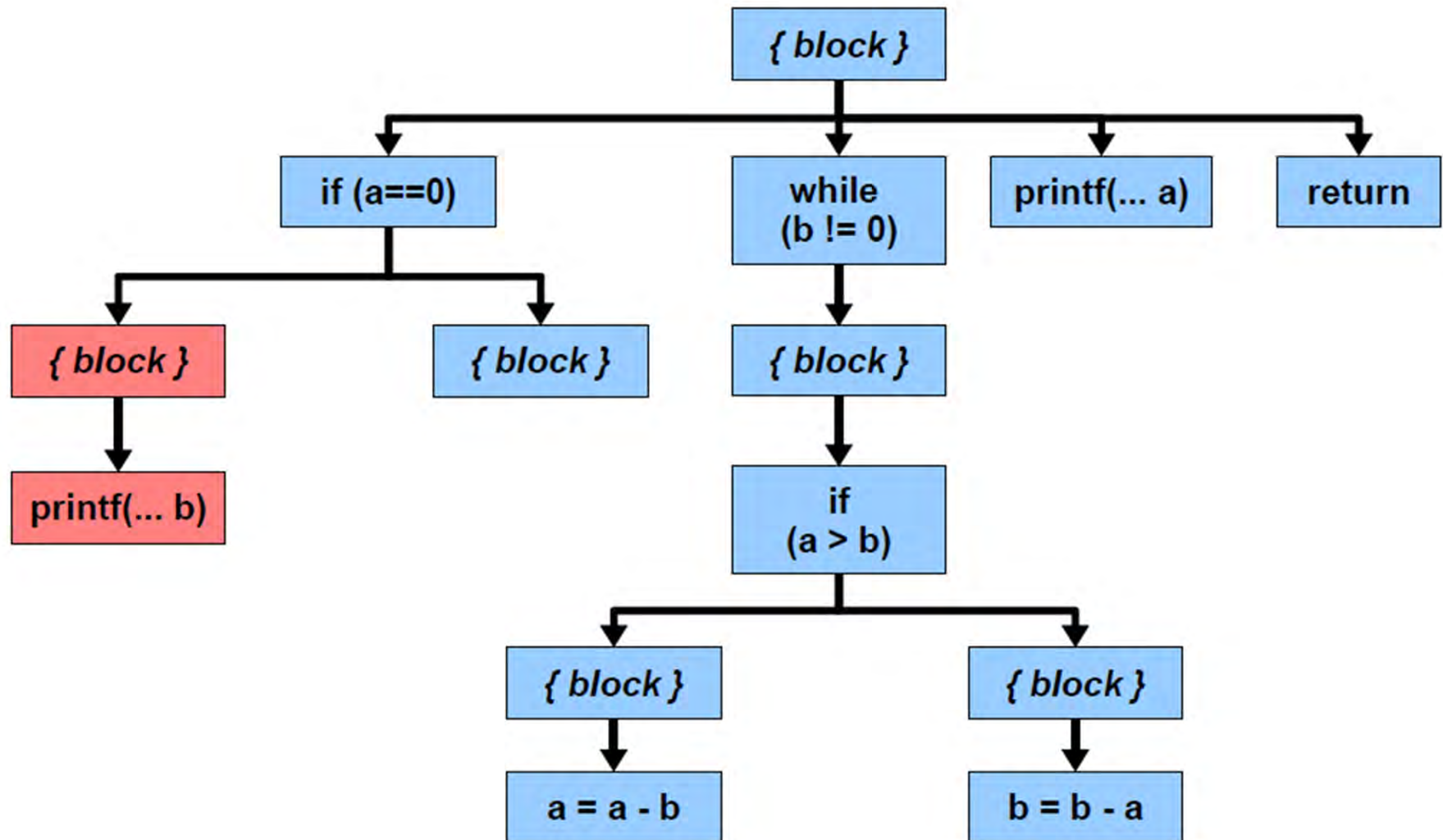
```
/* requires: a >= 0, b >= 0 */  
void print_gcd(int a, int b) {  
    if (a == 0)  
        printf("%d", b)  
    while (b != 0) {  
        if (a > b)  
            a = a - b;  
        else  
            b = b - a;  
    }  
    printf("%d", a);  
    return;  
}
```

## Example: Abstract Syntax Tree



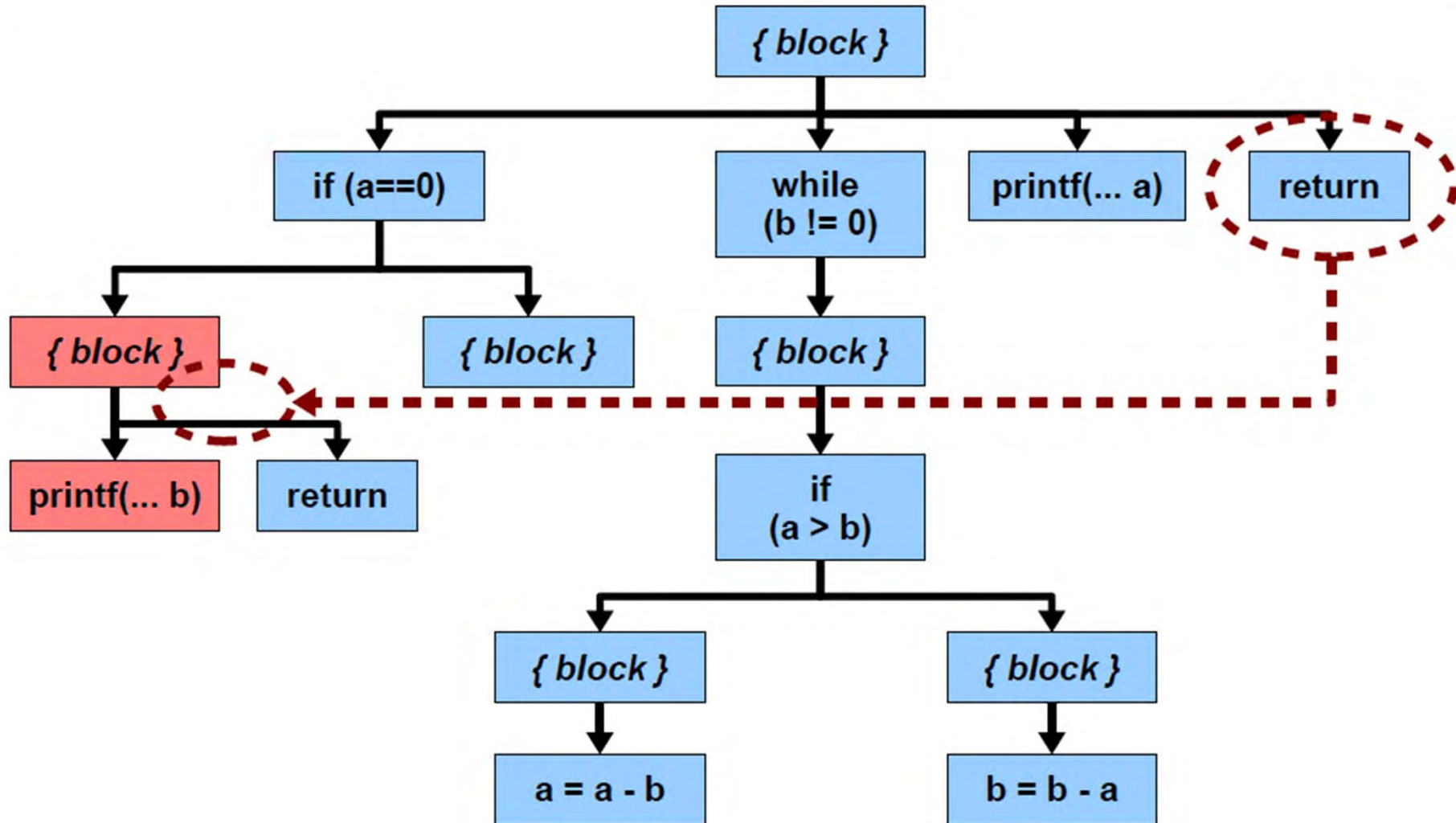
# Example: GCD

```
/* requires: a >= 0, b >= 0 */  
void print gcd(int a, int b) {
```



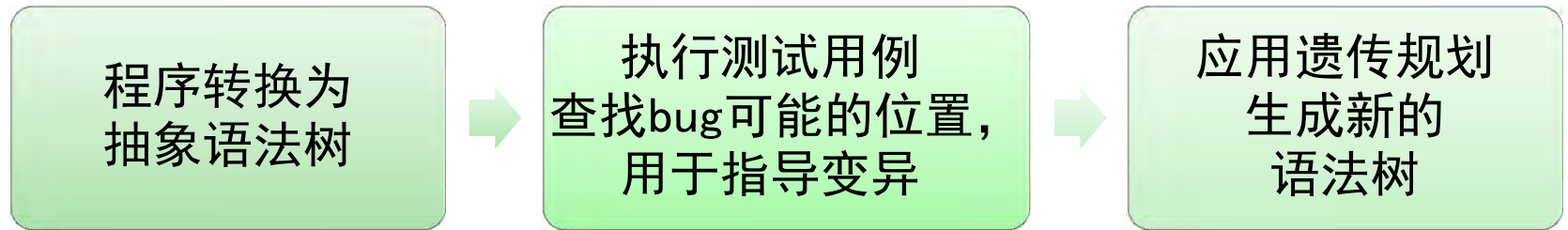
# Example: GCD

```
/* requires: a >= 0, b >= 0 */  
void print gcd(int a, int b) {
```



# 基于遗传规划的缺陷修复

基于遗传规划的修复, GenProg, by Weimer and Le Goues, ICSE 2009, TSE 2012.

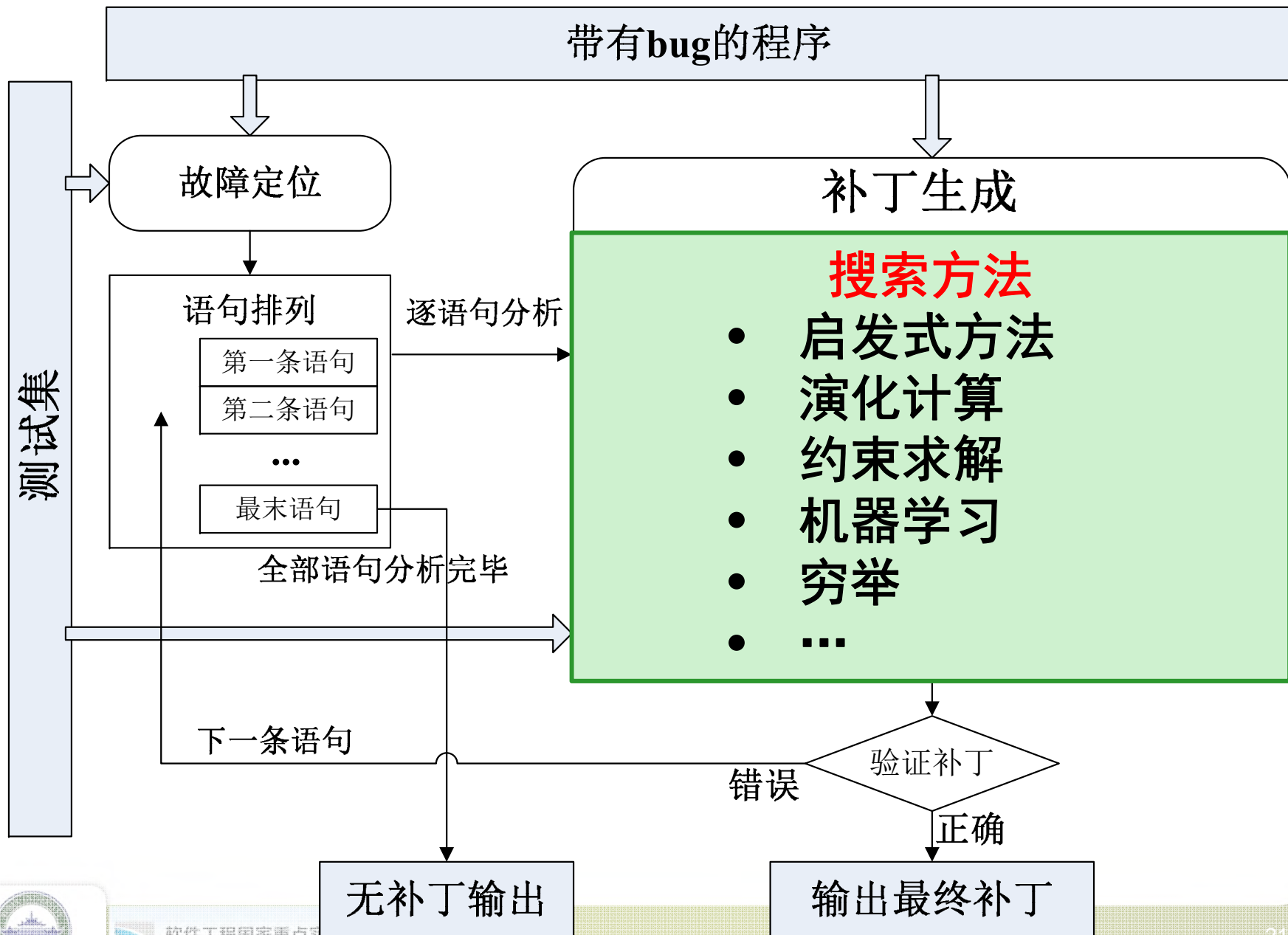


每个8美元, 修复105个bug中的55个, by Le Goues et al., ICSE 2012.

真实C程序bug实验, 印证了GenProg方法的实践效果  
实践上验证了自动方法行之有效, 并且很多后续方法依赖该实验数据集



# 基于测试的程序修复框架





# 基于测试的程序修复

## 演化算法成果现状

- 多数方法基于尝试并验证（generate-and-validate）的策略，算法精度低，时间消耗高，**难以用于真实程序**
- 多数方法局限于修复运算符和数值程序，**无法修复面向对象程序**



```
Object object = null;  
Object = new Object();  
Object.foo();
```



```
Object object = null;  
Object.foo();  
Object = new Object();
```

# 自动条件语句修复

## 自动缺陷修复

已知程序和测试用例集，能否自动生成代码补丁，并通过全部测试集用例？

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    }  
    ...  
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
$2^{30}$	2	2	2	Pass
...	...	...	...	...



# 自动条件语句修复

## 自动缺陷修复

已知程序和测试用例集，能否自动生成代码补丁，并通过全部测试集用例？

计算u与v的最大公约数

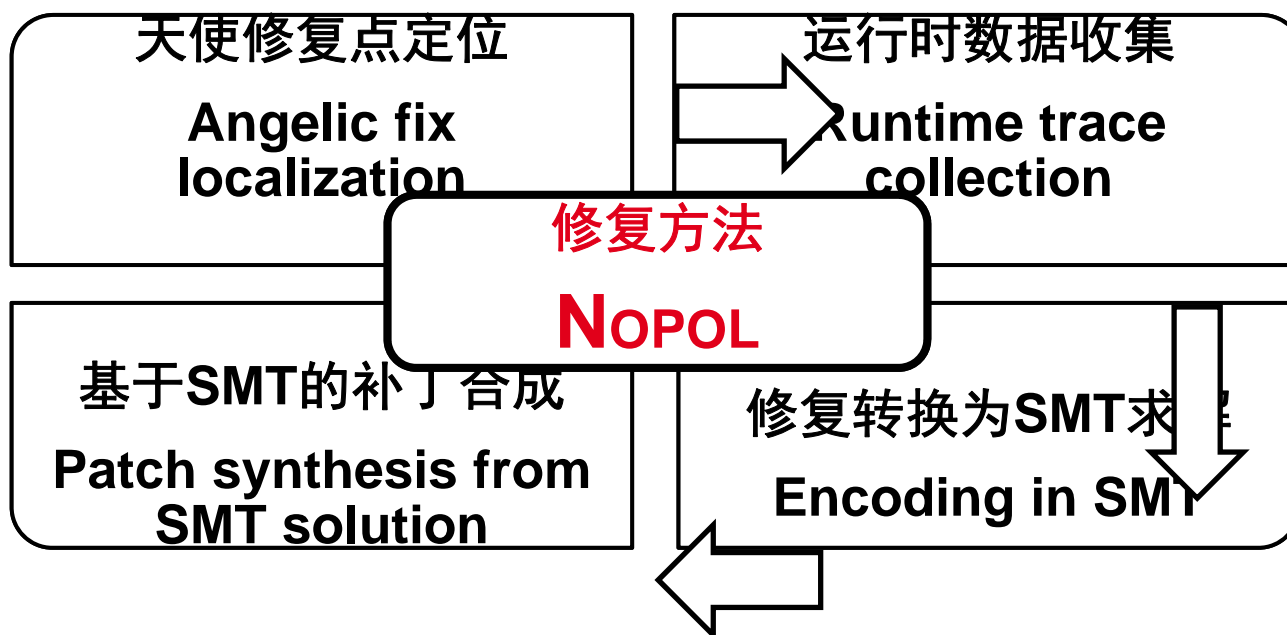
困难问题  $\Rightarrow$  问题转换 + 自动求解

生成一行补丁  $\Rightarrow$  SMT encoding + solving





# 条件语句的自动修复



Jifeng Xuan, et al. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. IEEE Trans. Software Engineering, 2017.



# 条件语句的自动修复1

## 思路

- 天使修复点定位 (Angelic fix localization), 为条件语句预估值

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    }  
    ...  
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
2^30	2^29	2^29	2^30+2^29	Fail
2^30	2	2	2	Pass
...	...	...	...	...



# 条件语句的自动修复1

## 思路

- 天使修复点定位 (Angelic fix localization)，为条件语句预估值

计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (u * v == 0) {
```

```
+   if (u == 0 || v == 0) {
```

```
    return (Math.abs(u) + Math.abs(v));
```

```
...  
}
```

问题：如何获得  
条件语句的值，  
true or false?

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
$2^{30}$	2	2	2	Pass
...	...	...	...	...



# 条件语句的自动修复1

## 思路

- 天使修复点定位 (Angelic fix localization), 为条件语句预估值



计算u与v的最大公约数

```
int gcd (int u, int v) {
```

```
- if (true) { ???
```

Not True !

```
+ if (u == 0 || v == 0) {
```

```
    return (Math.abs(u) + Math.abs(v));
```

```
...  
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
$2^{30}$	2	2	2	Pass
...	...	...	...	...



# 条件语句的自动修复1

## 思路

- 天使修复点定位 (Angelic fix localization), 为条件语句预估值



计算u与v的最大公约数

```
int gcd (int u, int v) {
```

```
- if (false) { ???
```

It's False !

```
+ if (u == 0 || v == 0) {
```

```
    return (Math.abs(u) + Math.abs(v));
```

```
...  
}
```

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
$2^{30}$	2	2	2	Pass
...	...	...	...	...

# 条件语句的自动修复1

## 思路

- 天使修复点定位 (Angelic fix localization), 为条件语句预估值



计算u与v的最大公约数

```
int gcd (int u, int v) {  
-   if (false) { ???  
+   if (u == 0 || v == 0) {  
        return (Math.abs(u) + Math.abs(v));  
    }  
    ...  
}
```

新问题：基于获得的  
条件值如何生成  
补丁？

输入 u	输入 v	预期输出	观测输出	测试结果
0	3	3	3	Pass
$2^{30}$	$2^{29}$	$2^{29}$	$2^{30}+2^{29}$	Fail
$2^{30}$	2	2	2	Pass

# 条件语句的自动修复2

## 思路

- **提高算法精度**，将被修复程序编码为约束求解问题（SMT），一次求解，避免多次尝试以降低时间消耗



# 条件语句的自动修复2

## 思路

- **提高算法精度**，将被修复程序编码为约束求解问题（SMT），一次求解，避免多次尝试以降低时间消耗

例1. 已知函数 $f(\text{int})$ 和 $g(\text{int}, \text{int})$ ，常数 $a$ 和 $b$ ，条件语句的目标值 $o$   
函数可改写为输入输出的管道， $O_1 := f(I_1)$  和  $O_2 := g(I_{2,1}, I_{2,2})$   
将上面 $O$ 与 $I$ 编码为整数索引，通过分析程序运行时提取索引的约束，  
可以用SMT获得基于约束的解，  
例如， $o = f(g(a, b))$  或  $o = g(f(b), a)$ ，进而转换为代码补丁





# 条件语句的自动修复2

## 思路

- **提高算法精度**，将被修复程序编码为约束求解问题（SMT），一次求解，避免多次尝试以降低时间消耗

例1. 已知函数 $f(int)$ 和 $g(int, int)$ ，常数 $a$ 和 $b$ ，条件语句的目标值 $o$   
函数可改写为输入输出的管道， $O_1 := f(I_1)$  和  $O_2 := g(I_{2,1}, I_{2,2})$   
将上面 $O$ 与 $I$ 编码为整数索引，通过分析程序运行时提取索引的约束，  
可以用SMT获得基于约束的解，  
例如， $o = f(g(a, b))$  或  $o = g(f(b), a)$ ，进而转换为代码补丁

- **面向对象程序**，收集运行时（runtime）面向对象域和方法返回值，作为程序补丁的一部分

包括收集对象是否初始化，和一些可取得的方法值，如`string.length()`等



## 运行时数据收集

被修复语句的输出：

$O_{loc,m,n}$  在代码段的loc位置的第n个测试用例第m次执行时的条件值

$$O_{loc,m,n} = \begin{cases} eval(loc), & \text{for passing test cases} \\ \text{angelic value} & \text{for failing test cases} \end{cases}$$

运行时的常数值：

int, Boolean, double, char, ...

运行时的对象状态：

$a = \text{obj.size}()$ ,  $b = \text{str.length}()$ , ...



## 生成语句条件的补丁

生成一个  $\text{if}(\text{cond}) \{ \dots \}$  中的  $\text{cond}$   
 $\Rightarrow$  生成一个布尔表达式  $\text{exp}$ , 使得

$$\forall_{loc,m,n} \exp(C_{loc,m,n}) = O_{loc,m,n}$$

其中,  $C_{loc,m,n}$  是运行时的收集值的集合。



# Building block

$$b_i = (\phi_i(I_i, r_i), I_i, r_i) \quad (b_i = (\phi_i, I_i, r_i) \text{ for short})$$



## 补丁合成

```
INT a = 3;
```

```
BOOL b = false;
```

```
INT input;
```

```
BOOL output;
```

```
BOOL = foo ( BOOL )
```

```
BOOL = bar ( INT, INT )
```

```
foo: ! x  
bar: y + z
```



## 补丁合成

```
INT a = 3;
```

```
BOOL b = false;
```

```
INT input;
```

```
BOOL output;
```

```
BOOL = foo ( BOOL )
```

```
BOOL = bar ( INT, INT )
```



合成条件语句

BOOL = ????????

猜测:

foo ( b )

bar ( input, a )

foo ( foo ( b )

foo ( bar ( input, a ) )

...



# 补丁合成

```
INT a = 3;  
BOOL b = false;  
INT input;  
BOOL output;  
BOOL = foo ( BOOL )  
BOOL = bar ( INT, INT )
```



合成条件语句  
BOOL = ????????

## 如何合成补丁?

猜测:  
foo ( b )  
bar ( input, a )  
foo ( foo ( b )  
foo ( bar ( input, a ) )  
...



## 基础约束

$$\phi_{FIXED}(I_0, r) = (\bigwedge_{i=1}^{|I_0|} l_{I_0, i} = i) \bigwedge l_r = p$$

$$\phi_{OUTPUT}(O) = \bigwedge_{i=1}^{|O|} (|I_0| + 1 \leq l_{r_i} \leq p)$$

$$\phi_{INPUT}(I) = \bigwedge_{x \in I} \bigvee_{y \in \text{type}(x), x \neq y} (l_x = l_y)$$

$$\phi_{CONS}(L, O) = \bigwedge_{x, y \in O, x \neq y} l_x \neq l_y$$

$$\phi_{ACYC}(B, L, I, O) = \bigwedge_{(\phi_i, I_i, r_i) \in B} \bigwedge_{x \in I_i} l_x < l_{r_i}$$





# 功能约束

$$\phi_{LIB}(B, V_{IO}) = \bigwedge_{(\phi_i, I_i, r_i) \in B, v_{r_i} \in V_{IO}} pb_i(\text{value}(I_i), v_{r_i})$$

$$\phi_{CONN}(L, V_{IO}) = \bigwedge_{S \in \{\text{BOOL}, \text{INT}, \text{REAL}\}} \bigwedge_{x, y \in S} l_x = l_y \Rightarrow v_x = v_y$$



$$\begin{aligned} \phi_{FUNC}(B, L, C_{loc, m, n}, O_{loc, m, n}) = \\ \exists V_{IO} \left( \phi_{LIB}(B, V_{IO}) \bigwedge \right. \\ \left. \phi_{CONN}(L, V_{IO}) [\text{value}(I_0) \leftarrow C_{loc, m, n}, v_r \leftarrow O_{loc, m, n}] \right) \end{aligned}$$

# 补丁合成

```
INT a = 3;  
BOOL b = false;  
INT input;  
BOOL output;  
BOOL = foo ( BOOL )  
BOOL = bar ( INT, INT )
```



合成条件语句  
BOOL = ????????

## 如何合成补丁?

猜测:  
foo ( b )  
bar ( input, a )  
foo ( foo ( b )  
foo ( bar ( input, a ) )  
...



## 补丁合成

u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOOL = bar ( INT, INT )

编码



```
loc (u) = 1;  
loc (v) = 2;  
loc (w) = 3;  
loc (x) = 4;
```



合成条件语句

BOOL = ????????

猜测:

foo ( b )

bar ( input, a )

foo ( foo ( b )

foo ( bar ( input, a ) )

...



## 补丁合成

u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOOL = bar ( INT, INT )



合成条件语句

BOOL = ????????

猜测:

foo ( b )

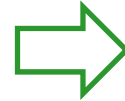
bar ( input, a )

foo ( foo ( b )

foo ( bar ( input, a ) )

...

编码



loc (u) = 1;

loc (v) = 2;

loc (w) = 3;

~~loc (x) = 4;~~



## 补丁合成

u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOOL = bar ( INT, INT )



合成条件语句

BOOL = ????????

猜测:

foo ( b )

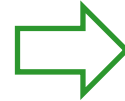
bar ( input, a )

foo ( foo ( b )

foo ( bar ( input, a ) )

...

编码



loc (u) = 1;

loc (v) = 2;

loc (w) = 3;

loc (temp) = 4 .. 5 ;

loc (x) = 6;

e.g.,  
foo (foo (bar ... ))



# 补丁合成

u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOOL = bar ( INT, INT )



合成条件语句  
BOOL = ????????

猜测:

foo ( b )  
bar ( input, a )  
foo ( foo ( b )  
foo ( bar ( input, a ) )  
...

编码



loc (u) = 1;
loc (v) = 2;
loc (w) = 3;
loc (temp) = 4 .. 5 ;
loc (x) = 6;
loc (y, i1)
= {loc (v), loc(temp)}
loc (z, i1)
= {loc(u), loc(w), loc(temp)}
loc (z, i2)
= {loc(u), loc(w), loc(temp)}
loc (y, o)
= { loc (temp), loc(x) }
loc (z, o)
= { loc (temp), loc(x) }

输入

输出



# 补丁合成

编码



u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOO

loc (u) = 1;
loc (v) = 2;
loc (w) = 3;
loc (temp) = 4 .. 5 ;
loc (x) = 6;
loc (y, i1)
= {loc (v), loc(temp)}
loc (z, i1)
= { loc (temp), loc(x) }
loc (z, o)
= { loc (temp), loc(x) }



猜测  
foo (  
bar (  
foo (  
foo (  
...

如何获得一组解  
即下面变量的值

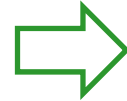
loc (y, i1) =  
loc (y, o) =  
loc (z, i1) = ?  
loc (z, i2) = ?  
loc (z, o) = ?

通过SMT  
约束求解



# 补丁合成

编码



u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOO



猜测

foo (  
bar (  
foo (  
foo (  
...

如何获得一组解  
即下面变量的值

loc (y, i1) = 2

loc (y, o) = 6

loc (z, i1) = 1

loc (z, i2) = 3

loc (z, o) = 5

loc (u) = 1;
loc (v) = 2;
loc (w) = 3;
loc (temp) = 4 .. 5 ;
loc (x) = 6;
loc (y, i1)
= {loc (v), loc(temp)}
loc (y, i2)
= {loc (temp), loc(x)}
loc (z, o)
= {loc (temp), loc(x)}

Patch:  
foo( b )





# 补丁合成

编码



u	INT a = 3;
v	BOOL b = false;
w	INT input;
x	BOOL output;
y	BOOL = foo ( BOOL )
z	BOO

loc (u) = 1;
loc (v) = 2;
loc (w) = 3;
loc (temp) = 4 .. 5 ;
loc (x) = 6;
loc (y, i1)
= {loc (v), loc(temp)}
loc (z, i1)
= { loc (temp), loc(x) }
loc (z, o)
= { loc (temp), loc(x) }



猜测  
foo (  
bar (  
foo (  
foo (  
...

如何获得一组解  
即下面变量的值

loc (y, i1) = 5

loc (y, o) = 6

loc (z, i1) = 1

loc (z, i2) = 3

loc (z, o) = 5

**Patch:**  
foo( bar(a, input))



# 实验结果

## 基于22个真实bug的程序修复结果

- 每个被修复程序**平均包含2.5万行代码**，399个类，1780个测试用例
- 17 / 22 个bug可以被修复

补丁

例2. 计算最大公约数（真实bug来自Apache Commons Math）

```
-   if (u * v == 0)
+   if (u == 0 || v == 0)
    return (Math.abs(u) + Math.abs(v));
```

补丁

例3. 匹配字符串模式（真实bug来自Apache Commons Math）

```
+   if (specific != null)
    stringBuilder.append(": ");
```



## 源自Apache Commons Lang, commit=230921

```

1  int indexOf(String substr, int startIndex) {
2      startIndex = (startIndex < 0 ? 0 : startIndex);
3      // FIX: if (substr == null || startIndex >= size) {
4      if (startIndex >= size) {
5          return -1;
6      }
7      int strLen = substr.length();
8      if (strLen > 0 && strLen <= size) {
9          if (strLen == 1)
10             return indexOf(substr.charAt(0), startIndex);
11         char[] thisBuf = buffer;
12         outer:
13         for (int i = startIndex; i < thisBuf.length
14             - strLen; i++) {
15             for (int j = 0; j < strLen; j++) {
16                 if (substr.charAt(j) != thisBuf[i + j])
17                     continue outer;
18             }
19             return i;
20         }
21     } else if (strLen == 0) {
22         return 0;
23     }
24     return -1;
25 }

```

Input			Output, indexOf(substr, startIndex)		Test result
parent	substr	startIndex	Expected	Observed	
abab	z	2	-1	-1	Pass
abab	(String) null	0	-1	NullPointerException	Fail
xyzabc	(String) null	2	-1	NullPointerException	Fail



## 现实并不那么美

不成功的例子，源自Apache Commons Math  
Bug:

```
v < knots[0] || v >= knots[n]
```

人工补丁:

```
v < knots[0] || v > knots[n]
```

我们的补丁:

```
if (v <= -1 || knots.length != v && n < v + 1) { ...
```



# 还有更多不足

## SMT求解器的性能缺陷

不能够处理带有参数的method call  
会导致组合爆炸

存在没有angelic value的条件语句



# 相关结果

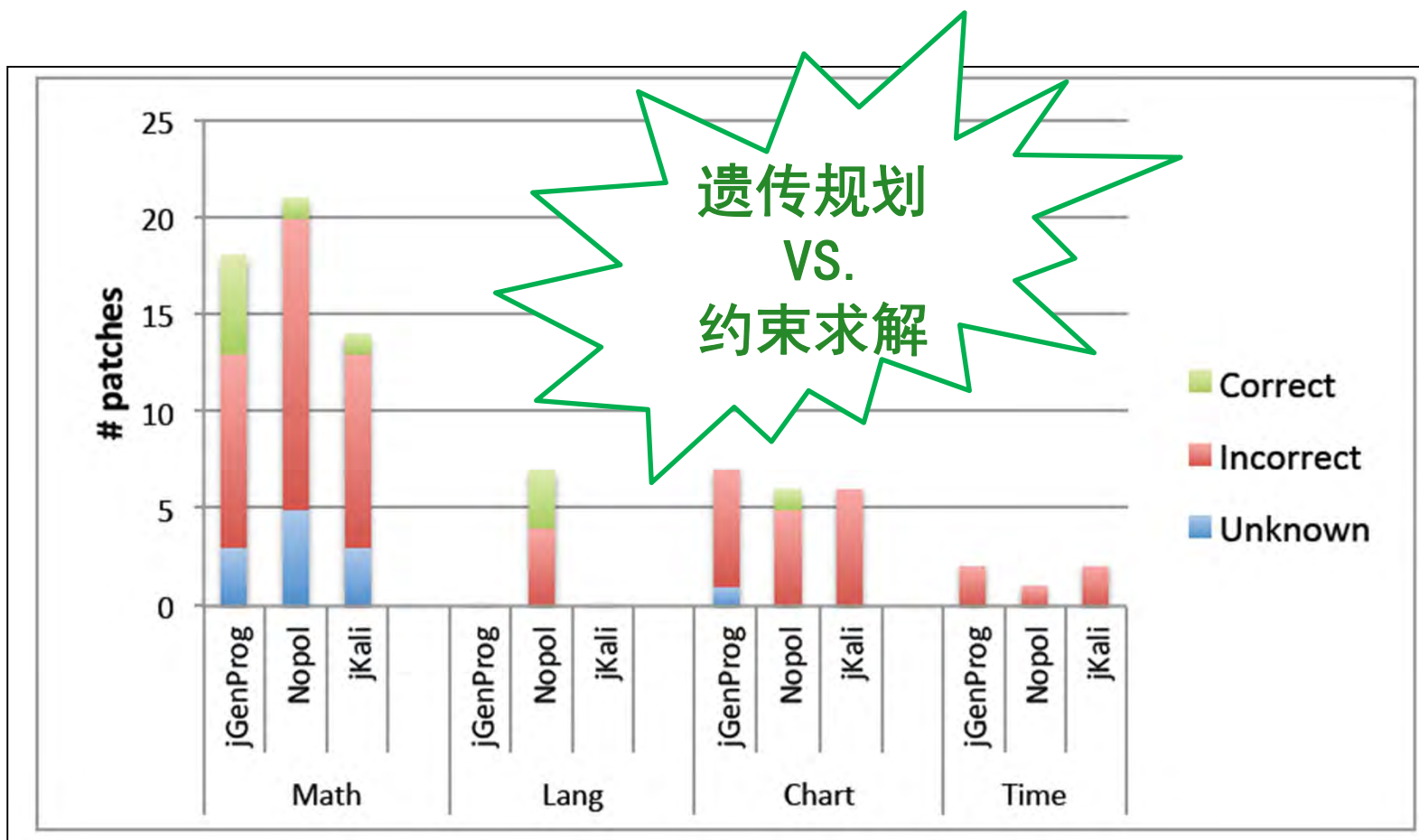
## 实验输出

- 22个真实的条件bug的数据集
  - 22 Bug dataset in GitHub,  
<https://github.com/SpoonLabs/nopol-experiments/>
- 修复方法已开源
  - Nopol in GitHub,  
<http://github.com/SpoonLabs/nopol>

**Jifeng Xuan**, et al. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. IEEE Trans. Software Engineering, 2017.



# 修复方法有效性的实证评估



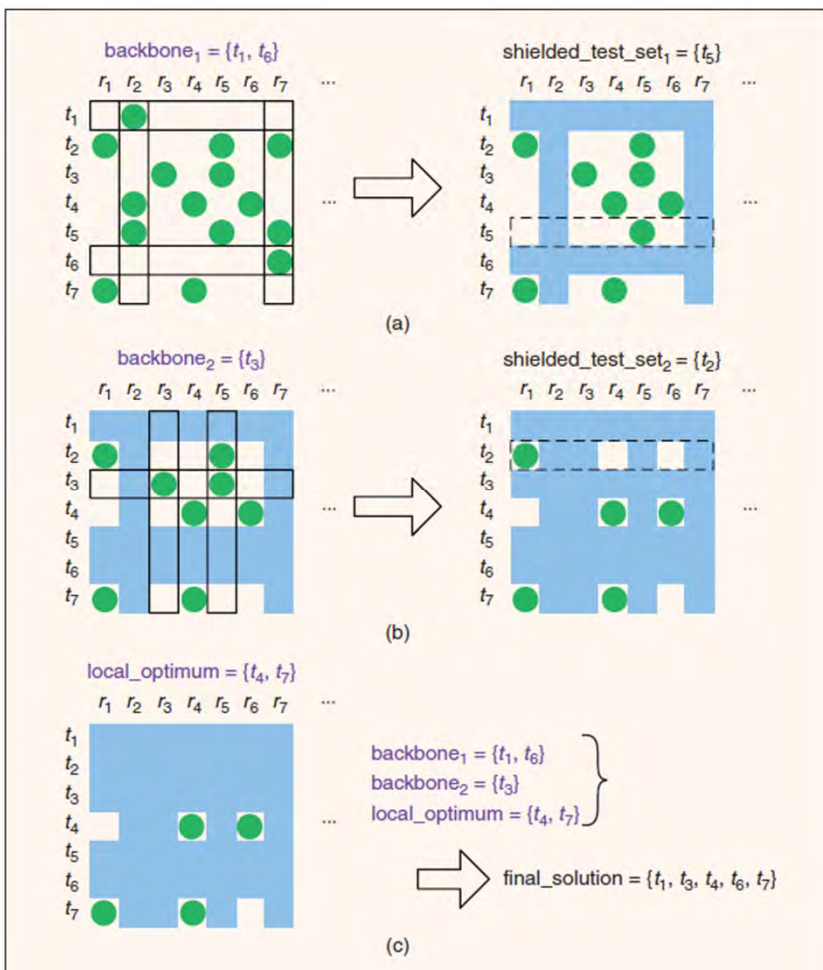
Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset. Empirical Software Engineering, 2017.





# 面向快速缺陷发现的测试用例集合规约

我们不能总是运行全部测试用例，所以 - 运行它的子集



Zongzheng Chi, Jifeng Xuan, Zhilei Ren, et al. Multi-Level Random Walk for Software Test Suite Reduction. IEEE Computational Intelligence Magazine, 2017.





## 测试用例重构

动机 - 很多动态分析技术以测试用例为输入，包括程序修复，而测试用例具有多样性。

例如 - 前面的条件语句的修复，需要测试用例能够只覆盖条件语句的一个分支(**then分支或else分支**)



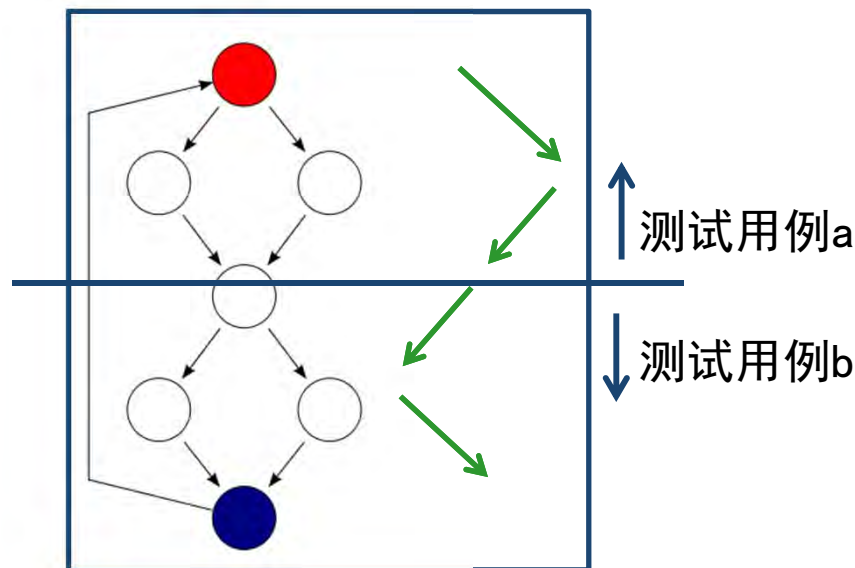
# 测试用例重构

## 方法

针对动态分析任务（包括程序修复），自动切分测试用例，减少多样性；切分后的测试用例，作为动态分析任务的输入

## 实验结果

包括自动修复在内的动态分析技术，可以借此提高分析结果



**Jifeng Xuan**, et al. Dynamic Analysis can be Improved with Automatic Test Suite Refactoring. Information and Software Technology, 2016.

# 基于测试用例提纯的测试增强

Original test case, t1	
1	public class IEEE754rUtilsTest {
2	@Test
3	void test_t1() {
4	...
Passing	assertEquals(1.2f, IEEE754rUtils.min(1.2f, 2.5f, Float.NaN));
Failing	assertEquals(2.5f, IEEE754rUtils.max(1.2f, 2.5f, Float.NaN));
7	...
8	float[] aF = new float[] {1.2f, Float.NaN, 3.7f, 27.0f, 42.0f, Float.NaN};
Ignored	assertEquals(42.0f, IEEE754rUtils.max(aF));
10	...
11	}
12	}



# 基于测试用例提纯的测试增强

Original test case, t1



```
1 public class IEEE754rUtilsTest {  
2     @Test  
3     void test_t1() {  
4         ...  
7  
8  
10  
11     }  
12 }
```



增强现有测试用例的使用效果



Passing

Failing

Ignored

@Test  
void test\_a1(){  
  
 assertEquals();  
 assertEquals();  
 float[] aF =...;  
 assertEquals();  
}

@Test  
void test\_a2(){  
  
 assertEquals();  
  
 assertEquals();  
  
 float[] aF =...;  
 assertEquals();  
}

@Test  
void test\_a3(){  
  
 assertEquals();  
  
 assertEquals();  
  
 float[] aF =...;  
 assertEquals();  
}

# 故障定位辅助技术可能伤害Debugging...

The screenshot shows a code editor window titled "Triangle.java" containing the following Java code:

```
/**
 *
 * @param triangle
 * @return type of triangle
 */
public String getType(Triangle triangle) {
    String strType = "Illegal";

    // Is it a Triangle
    if (isTriangle(triangle)) {
        // Is it regular
        if (triangle.lborderA == triangle.lborderB
            && triangle.lborderB == triangle.lborderC) {
            strType = "Regular";
        }
        // Is it scalene
        else if ((triangle.lborderA == triangle.lborderC)
            && (triangle.lborderB != triangle.lborderC)
            && (triangle.lborderA != triangle.lborderC)) {
            strType = "Scalene";
        }
        // Is it isosceles
        else {
            strType = "Isosceles";
        }
    }

    return strType;
}
```

Annotations on the code:

- "Second Suspicious" points to the condition `triangle.lborderA == triangle.lborderB`.
- "Most Suspicious" points to the condition `triangle.lborderA == triangle.lborderC`.
- "Highlight Overview" points to the right-hand side of the code editor.
- "Score Panel" points to the bottom panel of the IDE.

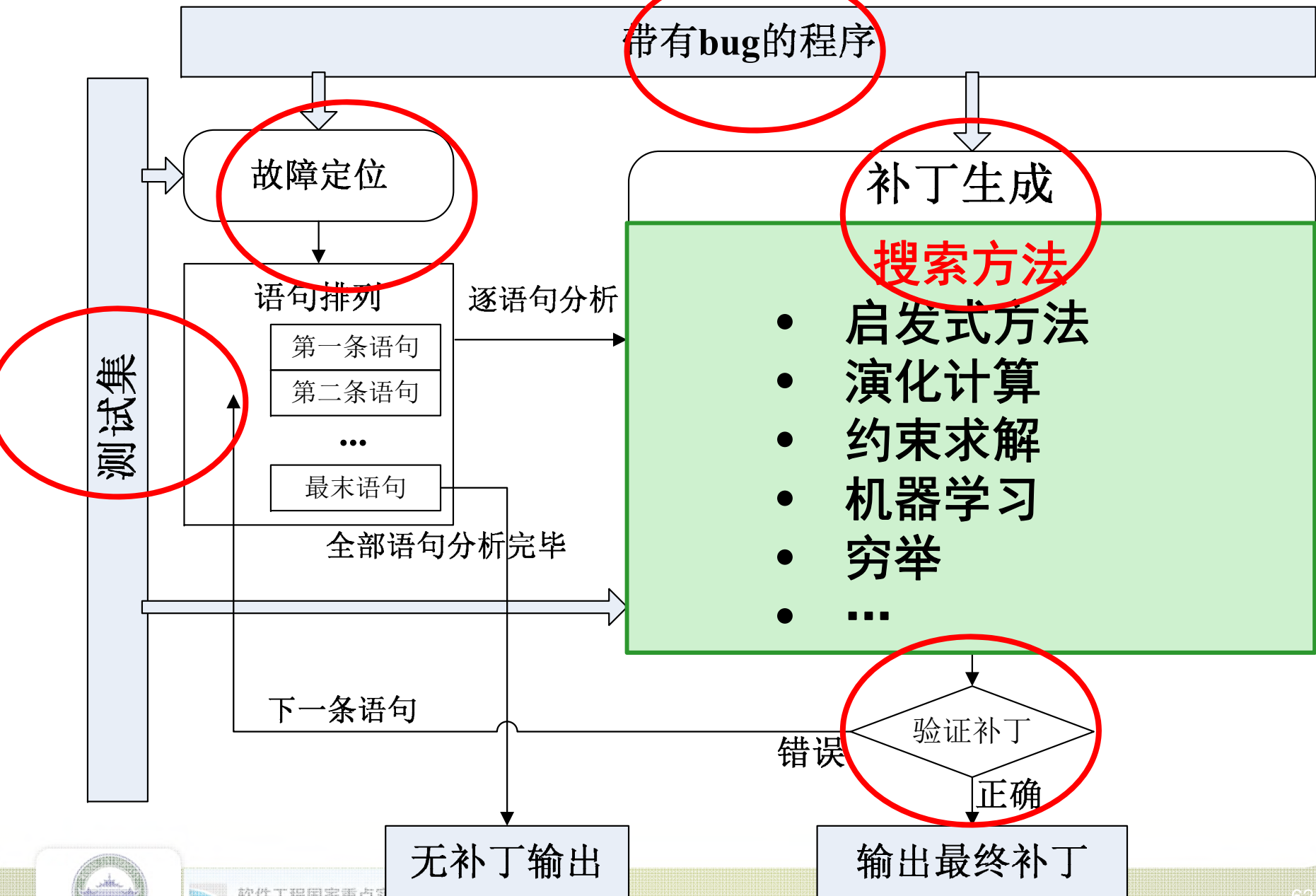
The bottom panel, titled "MoocTest", displays a table with the following data:

Name	Total Test Cases	Initial Failed Test Cases	Now Failed Test Cases	Score
Triangle	11	4	2	50.0

Revisit of Automatic Debugging via Human Focus-tracking Analysis. ICSE 2016.



# 未来与挑战 - 基于测试的程序修复





## 部分相关工作

- **Jifeng Xuan**, et al. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. *IEEE Transactions on Software Engineering*, 2017.
- Matias Martinez, Thomas Durieux, Romain Sommerard, **Jifeng Xuan**, Martin Monperrus. Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset. *Empirical Software Engineering*, 2017.
- Zongzheng Chi, **Jifeng Xuan**, et al. Multi-Level Random Walk for Software Test Suite Reduction. *IEEE Computational Intelligence Magazine*, 2017.
- **Jifeng Xuan**, et al. B-Refactoring: Automatic Test Code Refactoring to Improve Dynamic Analysis. *Information and Software Technology*, 2016.
- **Jifeng Xuan**, et al. Towards Effective Bug Triage with Software Data Reduction Techniques. *IEEE Transactions on Knowledge and Data Engineering*, 2015.
- **Jifeng Xuan**, et al. Crash Reproduction via Test Case Mutation: Let Existing Test Cases Help. *FSE-NIER*, 2015.
- **Jifeng Xuan**, et al. Test Case Purification for Improving Fault Localization. *FSE* 2014.
- 玄跻峰, 任志磊, 王子元, 谢晓园, 江贺. 自动程序修复方法研究进展. 软件学报, 2016.

课题组主页, 软件测试分析研究组 (CSTAR), <http://cstar.whu.edu.cn/cn/>





# 报告内容

程序修复中的约束求解

Q & A

欢迎批评指正

玄跻峰

**Email:** [jxuan@whu.edu.cn](mailto:jxuan@whu.edu.cn)

**URL:** <http://cstar.whu.edu.cn/>

