缺陷修复技术和北京大学近期进展

熊英飞

2018

报告人介绍 – 熊英飞

- 2000~2004，电子科技大学本科
- 2004~2006，北京大学研究生
  - 导师：梅宏、杨芙清
- 2006~2009，日本东京大学博士
  - 导师：胡振江、武市正人
- 2009~2011，加拿大滑铁卢大学博士后
  - 导师：Krzysztof Czarnecki
- 2012~，北京大学“百人计划”研究员（Tenure-Track）
- 研究方向：软件分析、编程语言设计

缘起

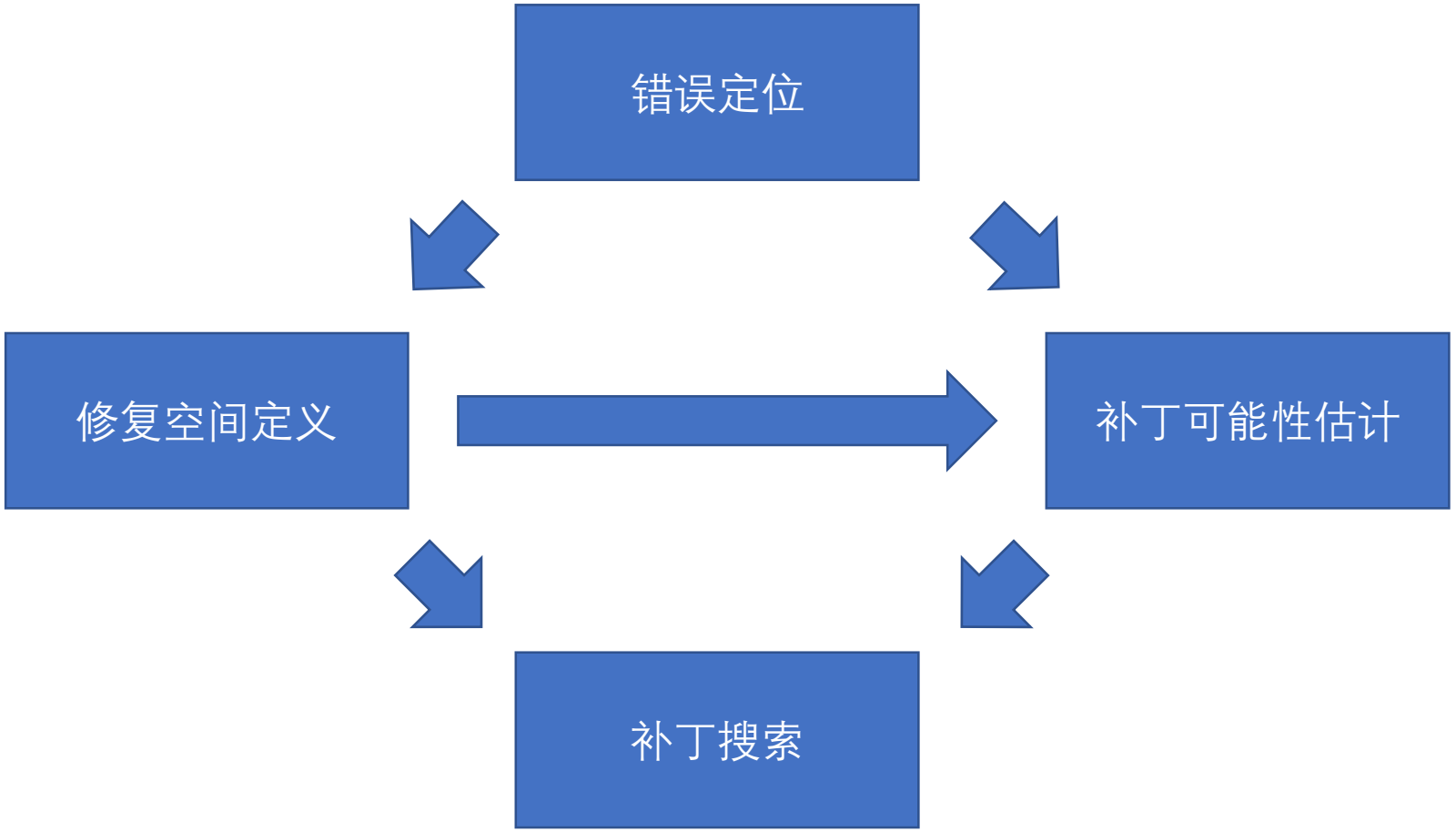
- 人和Bug的斗争从来没有停止过
- 缺陷检测：到底有没有Bug
  - 从上世纪60年代开始
  - 代表技术：软件测试、软件验证
- 缺陷定位：Bug在哪里
  - 从上世纪90年代开始
  - 代表技术：基于频谱的缺陷定位、统计性调试
- 缺陷修复：自动消除Bug
  - 约从2000年之后开始
  - 代表技术：生成-验证缺陷修复技术

缺陷自动修复定义

输入：一个程序和其正确性约束，并且程序不满足正确性约束
输出：一个补丁，可以使程序满足约束

研究和实践中考虑最广泛的正确性约束——
软件项目中的测试

缺陷修复四大支柱技术

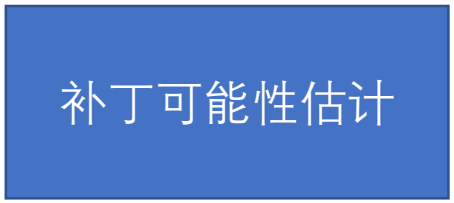


典型缺陷修复技术: GenProg

基于频谱的错误定位

以下三种操作的组合:

- 在错误语句前插入一条同项目任意语句
- 将错误语句替换成同项目任意语句
- 删除任意语句



通过测试越多越有可能

遗传算法

缺陷修复三大挑战

- 正确率：由于测试的不完备性，通过测试的缺陷未必是正确的，导致缺陷修复技术很难达到较高的准确率
- 召回率：由于补丁的多样性，很难定义出准确的修复空间，或者在修复空间中很难定位到准确的补丁。
- 修复效率：目前技术修复一个缺陷往往需要数小时

北京大学的近期工作

错误定位

集成错误定位方法
大幅提升错误定位正确率
[arXiv:1803.09939]

修复空间定义

通过分析历史补丁和项目代码来准确刻画修复空间
显著提升修复召回率
[ISSTA18]

补丁可能性估计

通过将生成过程分解来应用机器学习
显著提升修复正确率
[ICSE17(引用第二多),GI18,ICSE18]

通过计算重用减少重复计算
显著提升修复效率
[ISSTA17(Distinguished Paper)]

补丁搜索

北京大学的近期工作

错误定位

集成错误定位方法
大幅提升错误定位正确率
[arXiv:1803.09939]

修复空间定义

通过分析历史补丁和项目代码来准确刻画修复空间
显著提升修复召回率
[ISSTA18]

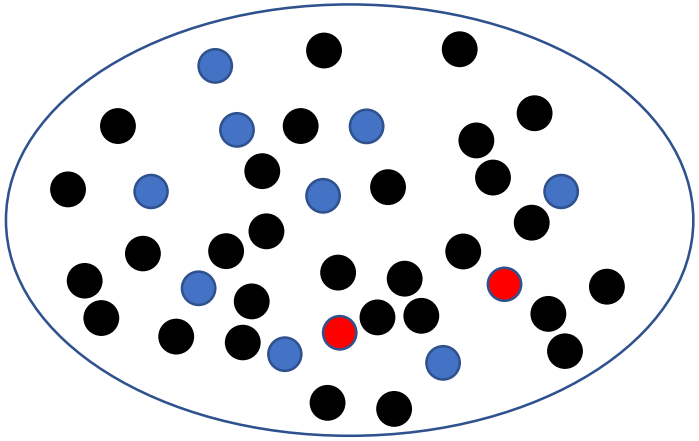
补丁可能性估计

通过将生成过程分解来应用机器学习
显著提升修复正确率
[ICSE17(引用第二多),GI18,ICSE18]

通过计算重用减少重复计算
显著提升修复效率
[ISSTA17(Distinguished Paper)]

补丁搜索

典型修复空间



不满足规约的补丁

满足规约但错误的补丁

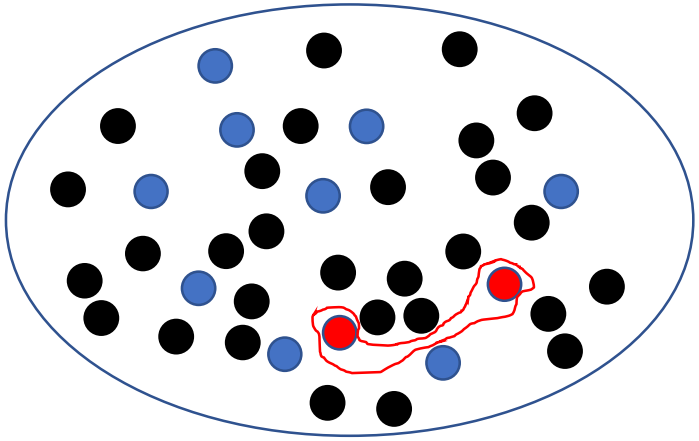
正确的补丁

空间较大

更可能包含正确补丁（召回率↑↑） 包含正确补丁时，更难定位到正确补丁（正确率↓↓召回率↓）	更不可能包含正确补丁（召回率↓↓） 包含正确补丁时，更容易定位到正确补丁（正确率↑↑召回率↑）

空间较小

理想修复空间



不满足规约的补丁

满足规约但错误的补丁

正确的补丁

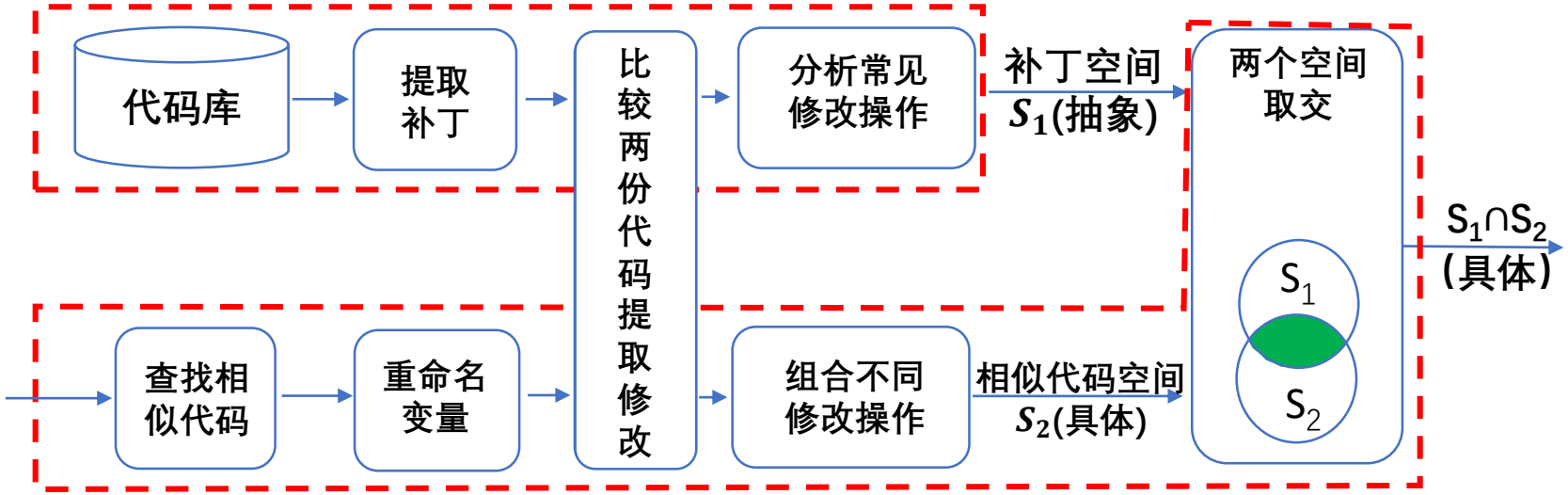
- 尽可能包含正确的补丁
- 尽可能不包含错误补丁，特别是能满足规约的错误补丁

SimFix——基本思路

- 从两个数据源刻画修复空间
- 已有补丁
  - 某些操作比其他操作更可能是修复操作
    - 更可能是修复操作：将<改成<=
    - 更不可能是修复操作：删除整个for语句块
  - 从已有补丁中可以统计出操作的信息
- 项目其他代码
  - 相似逻辑往往在一个项目中多次实现
  - 寻找错误代码和相似代码之间的差异可以更好的修复缺陷

SimFix——方法概览

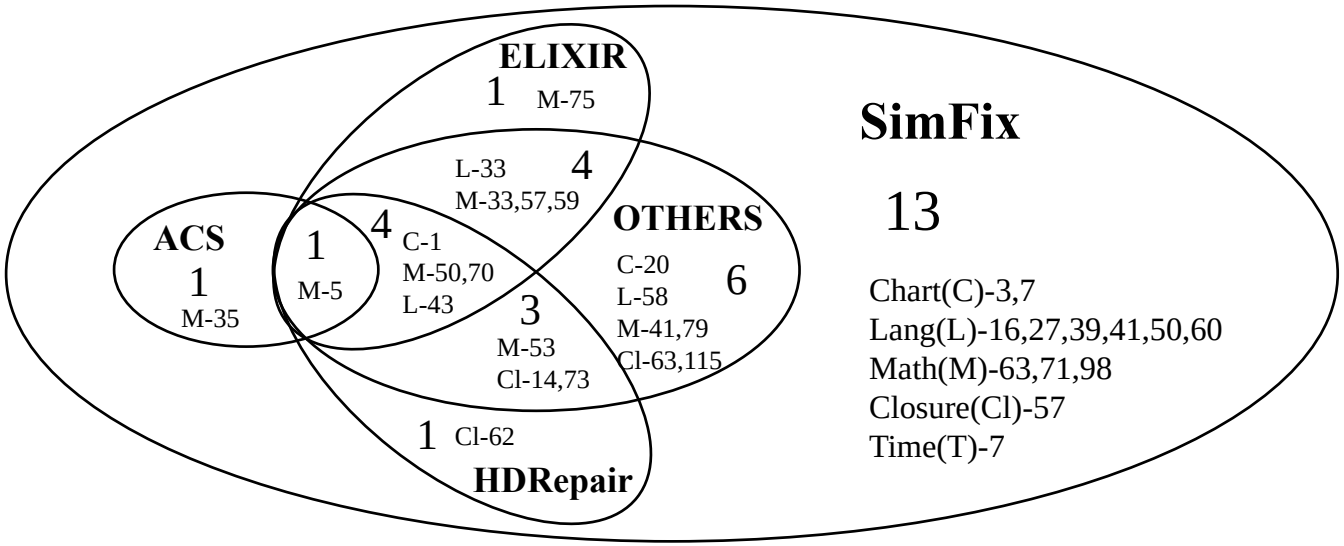
1. 离线挖掘阶段



2. 在线修复阶段

SimFix——方法效果

- 召回率：成功在Defects4J数据集上修复34个缺陷，包含13个首次被修复的缺陷，在所有方法中最多
- 正确率：达到60.7%



SimFix修复缺陷和已有方法的关系

北京大学的近期工作

错误定位

集成错误定位方法
大幅提升错误定位正确率
[arXiv:1803.09939]

修复空间定义

通过分析历史补丁和项目代码来准确刻画修复空间
显著提升修复召回率
[ISSTA18]

补丁可能性估计

通过将生成过程分解来应用机器学习
显著提升修复正确率
[ICSE17(引用第二多),GI18,ICSE18]

通过计算重用减少重复计算
显著提升修复效率
[ISSTA17(Distinguished Paper)]

补丁搜索

补丁正确性估计

- 已有工作：设计启发式规则
  - 通过测试的数量
  - 与原来补丁的语法距离
  - 与原来补丁的语义距离
  - 准确率较低
- 已有工作：应用机器学习
  - 传统机器学习
    - 补丁空间很大且不固定，无法应用
  - Learning-to-rank
    - 补丁复杂多变，难以准确提取特征

我们的工作: Learning to synthesize

- 将原来补丁的生成分成若干步骤
- 采用机器学习估计每一步正确的概率
- 最后概率是所有概率的乘积

例子：修复条件错误

条件错误是很常见的

```
   lcm = Math.abs(a+b);
+ if (lcm == Integer.MIN_Value)
+   throw new ArithmeticException();
```

缺少边界检查

```
-  if (hours <= 24)
+ if (hours < 24)
       withinOneDay=true;
```

条件过强或者过弱

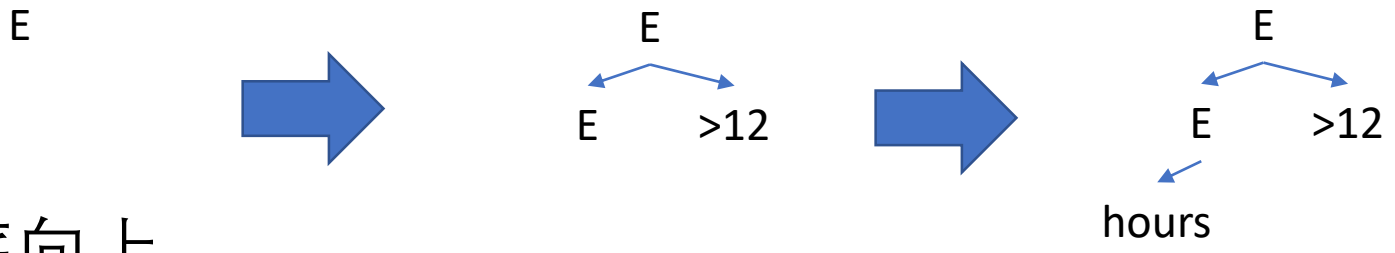
能否生成正确的条件来替换错误的条件？

搜索空间

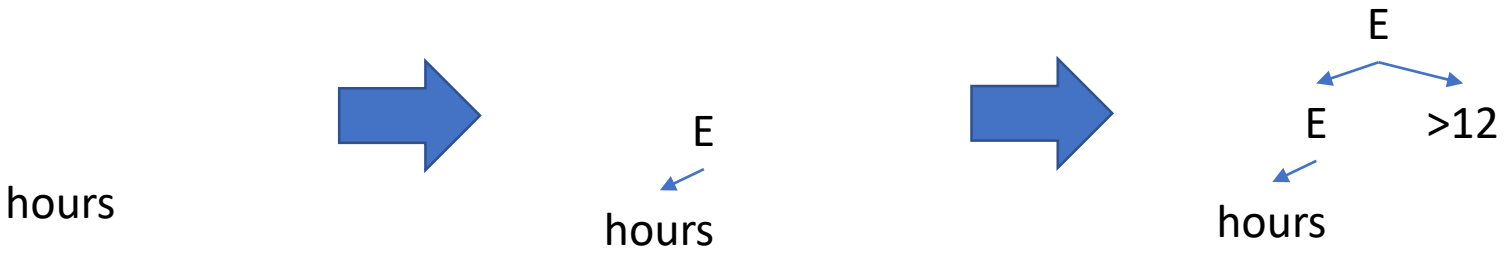
- 通过条件表达式的语法来定义
- E → E “>12”
  | E “>0”
  | E *“+”* E
  | *“hours”*
  | *“value”*
  | …

搜索过程

- 搜索过程是按某种顺序展开语法树的过程
- 自顶向下



- 自底向上



通过转换语法来表达搜索顺序

Grammar	Top-down Rules	Bottom-up Rules
$E \rightarrow E$ “+” E	$E^D \Rightarrow E \rightarrow E^D$ “+” E^D	$E^U \Rightarrow E^U \rightarrow E$ “+” E^D $E^U \Rightarrow E^U \rightarrow E^D$ “+” E
$E \rightarrow E$ “>12”	$E^D \Rightarrow E \rightarrow E^D$ “>12”	$E^U \Rightarrow E^U \rightarrow E$ “>12”
$E \rightarrow$ “hours”	$E^D \Rightarrow E \rightarrow$ “hours”	“hours”$^D \Rightarrow E^D \rightarrow$ “hours”

Creation Rules	
$\Rightarrow E^D$	// starting from the root
$\Rightarrow E^{DU}$	// starting from a middle node
$\Rightarrow$ “hours”^{U}	// starting from a leaf

Ending Rule	$E^U \Rightarrow E$

基于机器学习的概率估计

- 给定一个符号，训练机器学习模型估计该符号按不同方式展开的概率
  - $\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow \mathrm{E}^{\mathrm{D}}$ “+” E^{D}
  - $\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow \mathrm{E}^{\mathrm{D}}$ “>12”
  - $\mathrm{E}^{\mathrm{D}} \Rightarrow \mathrm{E} \rightarrow$ “hours”
- 特征提取可以针对不同符号单独设计

- 定理：当规则集合无歧义时，补丁的概率等于组成补丁的所有重写规则各自概率的乘积

在条件修复上的初步结果[ICSE17]

- 数据集：Defects4J上的四个项目
  - Time, Lang, Math, Chart
  - 总共224个缺陷

85%[ICSE18，基于补丁性质的过滤]

Approach	Correct	Incorrect	Precision	Recall
ACS	18	5	**73.9%**	**7.5%**
jGenProg	5	22	18.5%	2.2%
Nopol	5	30	14.3%	2.2%
xPAR	3	–[4]	–[4]	1.3%[2]
HistoricalFix[1]	10(16)[3]	–[4]	–[4]	4.5%(7.1%)[2,3]

北京大学的近期工作

错误定位

集成错误定位方法
大幅提升错误定位正确率
[arXiv:1803.09939]

修复空间定义

通过分析历史补丁和项目代码来准确刻画修复空间
显著提升修复召回率
[ISSTA18]

补丁可能性估计

通过将生成过程分解来应用机器学习
显著提升修复正确率
[ICSE17(引用第二多),GI18,ICSE18]

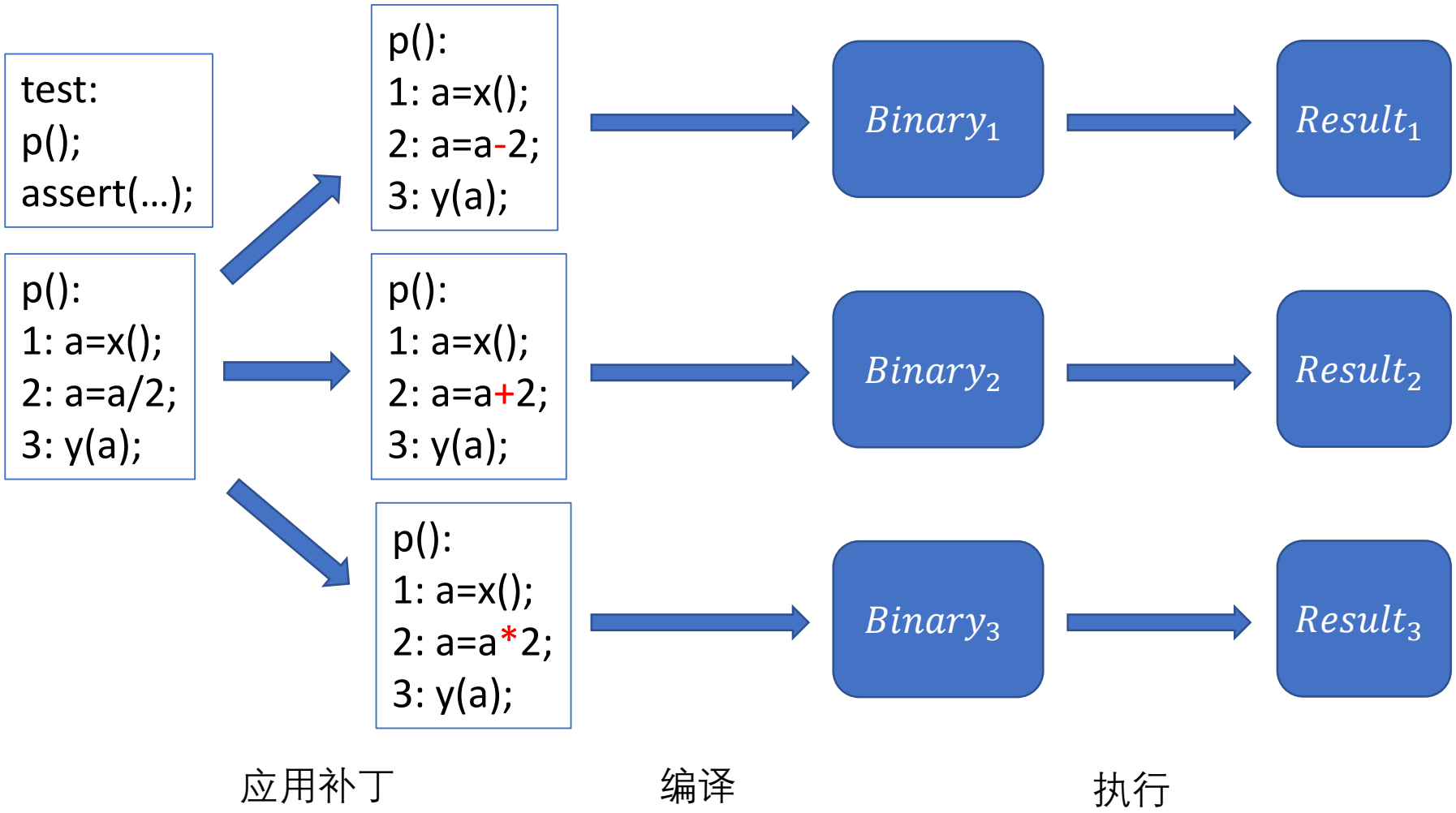
通过计算重用减少重复计算
显著提升修复效率
[ISSTA17(Distinguished Paper)]

补丁搜索

补丁搜索

- 现有缺陷修复技术常常需要花费数小时搜索补丁
- 补丁确认：补丁搜索过程中最耗时的部分
  - 应用补丁，检查是否所有测试都通过
- 能否加速该过程？

补丁确认过程


test:
p();
assert(…);
p():
1: a=x();
2: a=a/2;
3: y(a);
p():
1: a=x();
2: a=a-2;
3: y(a);
p():
1: a=x();
2: a=a+2;
3: y(a);
p():
1: a=x();
2: a=a*2;
3: y(a);
$Binary_1$
$Binary_2$
$Binary_3$
$Result_1$
$Result_2$
$Result_3$
应用补丁
编译
执行

编译的冗余

```
p():
1: a=x();
2: a=a-2;
3: y(a);
x():
…
y():
…
```

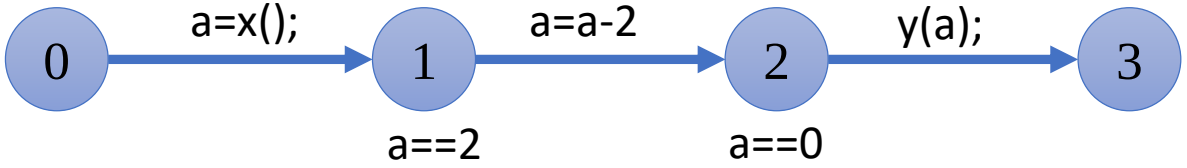
```
p():
1: a=x();
2: a=a+2;
3: y(a);
x():
…
y():
…
```

```
p():
1: a=x();
2: a=a*2;
3: y(a);
x():
…
y():
…
```

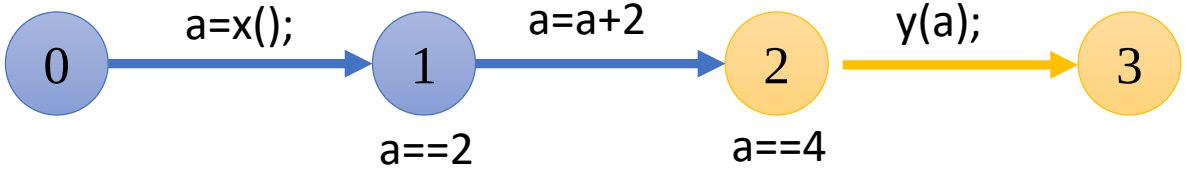
- x()和y()在三个版本中完全一样，但是被编译了三次

运行的冗余

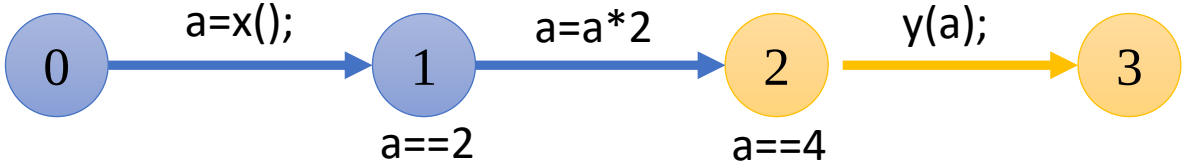
```
1: a=x();
2: a=a-2;
3: y(a);
```



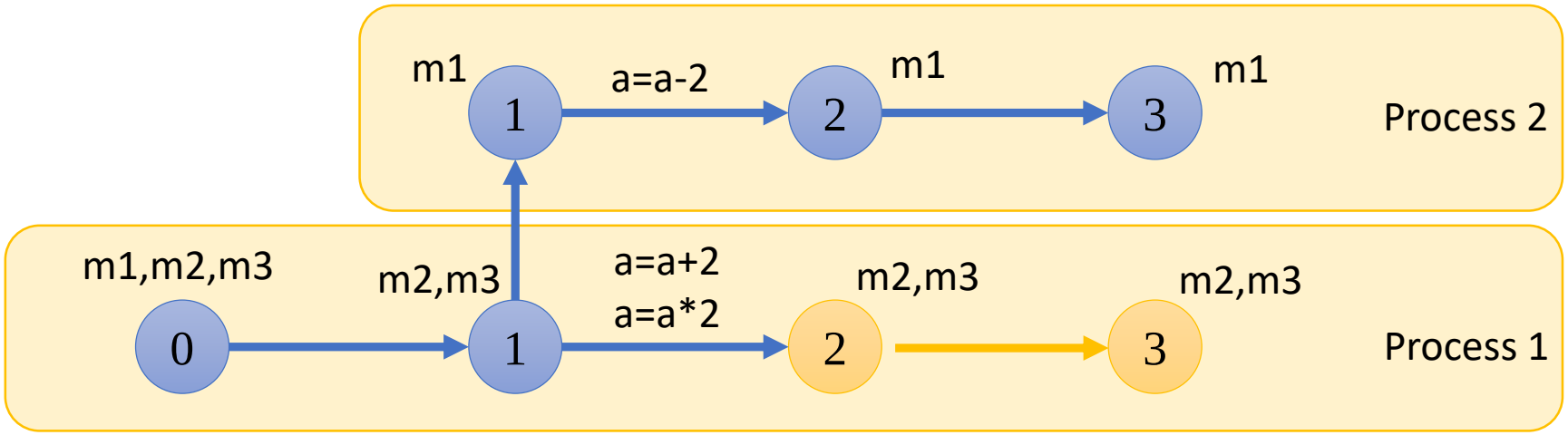

```
1: a=x();
2: a=a+2;
3: y(a);
```




```
1: a=x();
2: a=a*2;
3: y(a);
```



我们的工作：AccMut

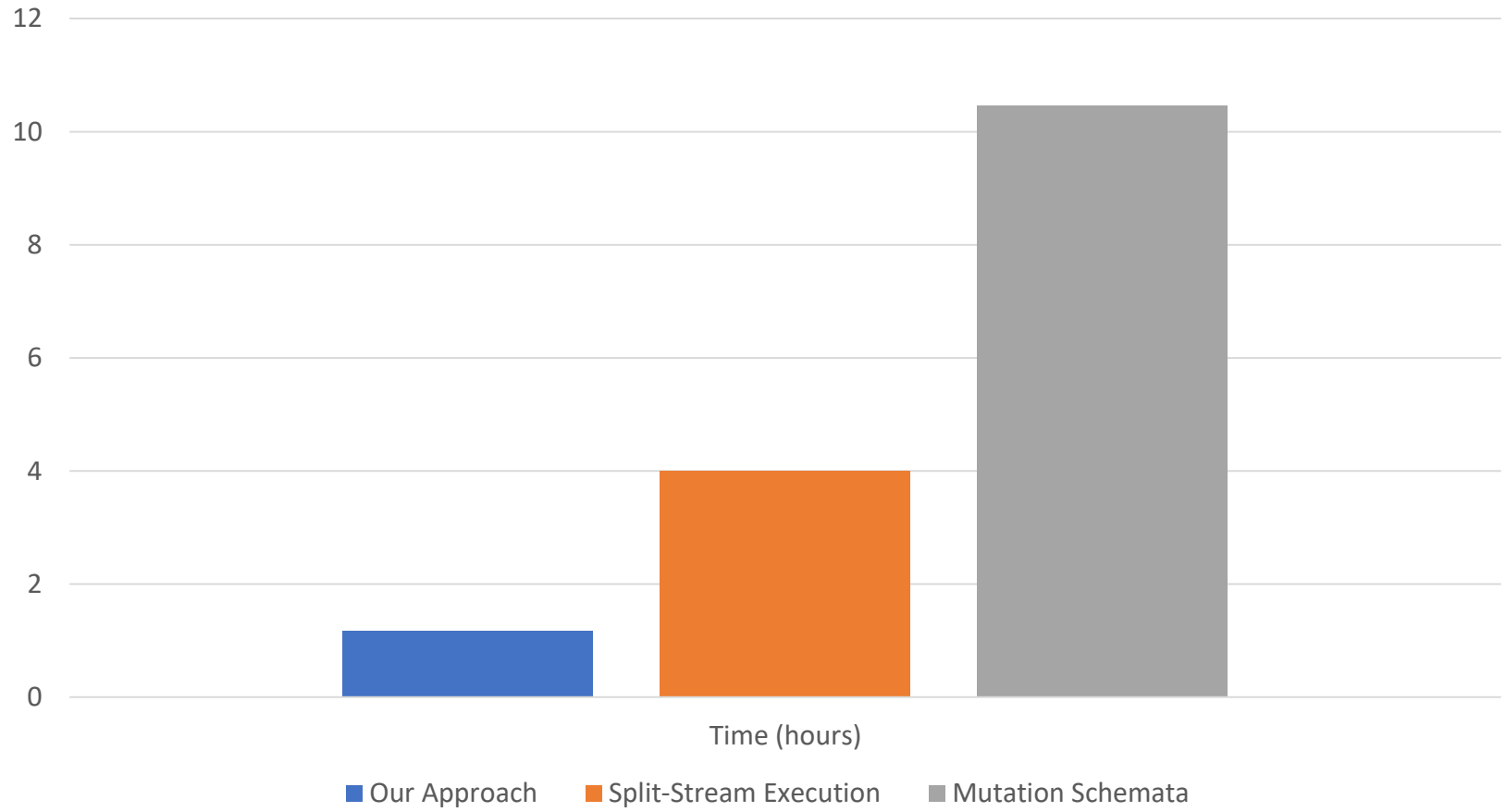


- 将所有补丁编译成一个超程序
- 超程序从一个进程开始执行，代表所有补丁
- 碰见有不同之处，超程序检查并分类不同语句的执行结果
- 对于每个等价结果类创建一个进程执行

如何实现超程序

- 需要保证 额外开销 << 去掉的重复计算
- 降低等价状态检查的额外开销
  - 运用抽象解释技术，将检查映射到一个开销较小的抽象域进行
- 降低结果分类的开销
  - 通过设计算法，使得分类算法依赖于一些上限较低的项

方法效果



在变异上模拟的效果：相比已有方法，我们比SSE有2.56倍加速，比MS有8.95倍加速

未来工作：缺陷修复开发框架

- 观察：
  - 学术界大量缺陷修复工作，不断重复实现基本模块
  - 学术界大量的小改进论文，互相之间的关系讨论、效果比较难以完成
  - 工业界实现缺陷修复工具缺乏参考实现
- 目标：
  - 帮助研究人员快速实现缺陷修复工具
  - 快速实验各种技术的不同组合方式
  - 为工业界提供参考实现
  - 集成今天所讲的各项工作