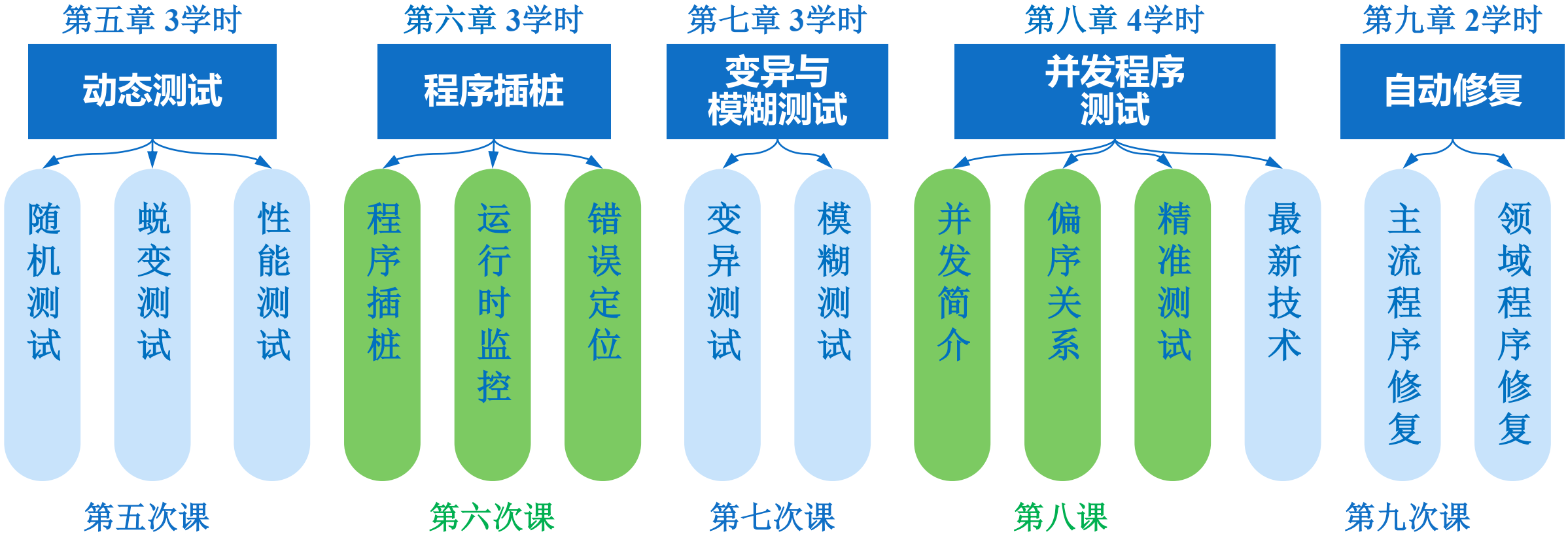
《软件分析与测试》第二部分



第二次大作业	发布日期：第8周~第9周	截止日期：第12周

第九章　自动修复方法

蔡彦
中国科学院软件研究所

第八章 自动修复方法（2学时）

- 基于动态测试的程序修复
- 领域程序自动修复
  - 并发程序修复
  - 循环(for-loop)修复
  - 浮点数错误修复

参考文献：

1. 王赞, 郜健, 陈翔, et al. 自动程序修复方法研究述评[J]. 计算机学报, 2017, 41(03): 588-610.
2. LIU Y, ZHANG L, et al. A Survey of Test Based Automatic Program Repair[J/OL]. Journal of Software, 2018: 437-452.
3. GAZZOLA L, MICUCCI D, MARIANI L. Automatic software repair: a survey[C/OL]//Proceedings of the 40th International Conference on Software Engineering. 2018.
4. 其它（PPT中相应地方标出的参考）

为什么需要自动修复

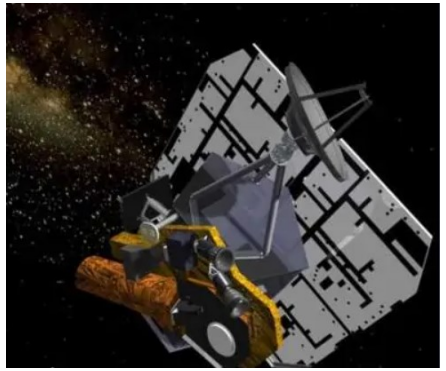
- 软件错误会造成损失
  - 数据丢失、隐私泄露、系统崩溃
- 软件维护的成本不足
  - 占开发成本的50%～75%（主要为错误定位和修复）
- 安全漏洞的严重挑战
  - 仅2022年，新增漏洞近2万5千个，达到历史新高，并保持连年增长态势

IT公司纷纷出台BUG赏金计划，以助于自身软件的维护与修复

- Microsoft Bug Bounty Program：https://www.microsoft.com/en-us/msrc/bounty
- Google’s Bug Hunting community：https://bughunters.google.com

为什么需要自动修复

- 传统的维护方式不仅耗时耗力、
  而且**经常无法解决问题**，特别是在处理复杂的程序错误时
  - 航天等领域软件中，软件逻辑和运行环境等因素，人工（远程）修复无法实现。



2005 年,NASA 发射的“深度撞击”号探测器由于重置探测器计算机遇到一个软件通讯的小故障，使得太阳能电池板方向指向错误。失去联系一个月以来，项目科学家多次尝试激活该探测器上的系统，但均以失败告终。最终探测器失去电力，寒冷的气温破坏机载设备，它的电池和推进系统等全被“冻死”。

自动程序修复

自动程序修复 APR（Automated/Automatic Program Repair/Fixing）

- 自动化的软件维护方法，利用**程序分析**技术和**人工智能**来发现、定位和修复程序中的错误
- 可以帮助开发人员快速修复大量缺陷报告，减少程序调试时间，提高软件的可信赖性和安全性

程序错误 (program error)

- 程序的预期行为和实际执行时所发生情况之间存在的一种偏差
- 程序的预期行为：又称为**程序规约**(program specification)，可以是自然语言书写的文本、形式化的逻辑公式，或者测试集等

自动程序修复就可以解释为将不符合程序规约的错误语义自动转换为符合程序规约的一种技术

李斌,贺也平,马恒太.程序自动修复:关键问题及技术.软件学报,2019,30(2):244−265

自动程序修复

自动程序修复的过程一般分为三个阶段

- **缺陷定位**：通过静态分析或动态测试方法确定程序可能的缺陷位置
- **补丁生成**：选择一个缺陷位置，使用特定的补丁生成技术尝试生成修复补丁
- **补丁评估**：利用测试集等验证生成补丁的正确性

姜佳君, 陈俊洁, and 熊英飞. "软件缺陷自动修复技术综述." 软件学报 32.9(2021):26.

APR的分类

按照自动化程度

- 全自动修复
- 半自动修复（为开发人员提供候选补丁）

按照修复策略

- 基于状态的修复（修改程序运行时的寄存器等执行状态）
- 基于行为的修复（直接改变程序代码）

按照程序规约

- 不完全规约的修复（基于测试集）
- 完全规约的修复（对特定错误的完全修复）
- 半完全规约的修复（手工编写规约等）

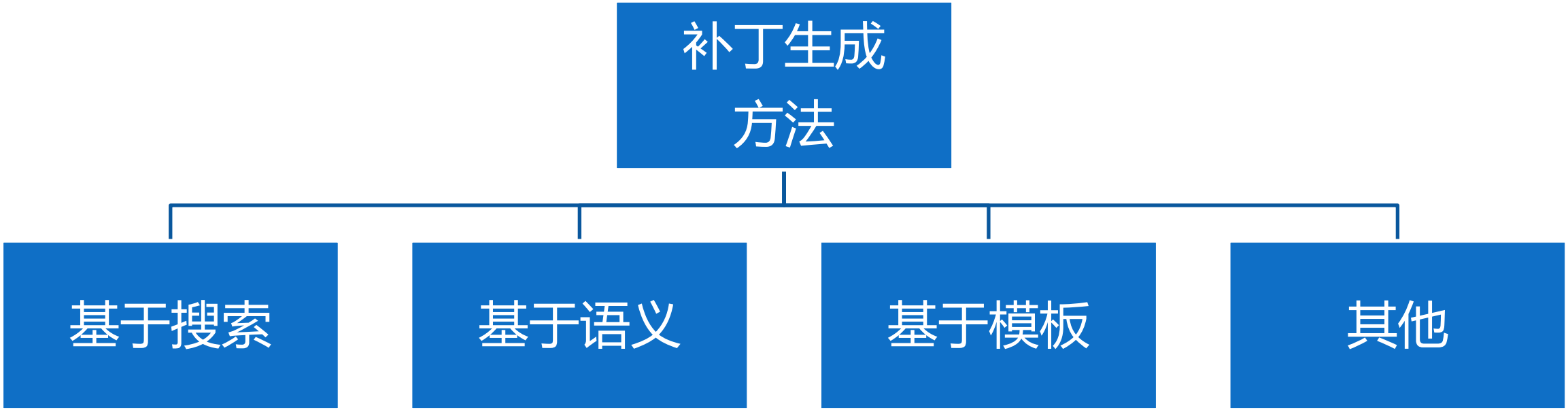
经典的APR方法是基于动态测试的程序自动修复

缺陷定位

基于 错误定位方法

- 通过执行预先选择的测试用例来获得程序的执行信息，并通过分析程序的执行流程来定位测试程序中**缺陷语句**的位置
- 利用成功测试用例的个数和失败测试用例的个数，以及程序语句被覆盖的次数，可以计算得到程序语句的**可疑度**
- 修复工具按照可疑度列表依次检查待检测语句并**生成候选补丁**，直到找到真正有缺陷的语句完成补丁生成
- 可疑度的计算有多种方式，包括简单的0-1分类和各种公式计算得到的可疑度具体值等

补丁生成



基于搜索的补丁生成

基本假设：补丁代码可以由程序其他位置的代码生成

- 通过对程序的整体分析，就建立了一个补丁搜索空间，可以利用各种**启发式算法**进行补丁生成

- GenProg （APR的经典技术之一）
  - 基于遗传编程(Genetic Programming)：提取程序的抽象语法树(AST)，在AST上，迭代应用交叉算子和变异算子，修改AST中的节点
  - 操作包括：删除、增加或替换，将修改后的AST转换成对应的程序补丁

基于搜索的补丁生成

- RSRepair
  - 将GenProg的遗传算法替换为更简单的随机搜索(Random Research)，只使用变异操作来生成可用的补丁
- 其它
  - 在GenProg的基础上，许多团队进行了各种优化提升工作，包括添加“染色体”概念、优化交叉变异细节、选择初始化种群、产生新语句扩展搜索空间等等
- 除了现有的程序语句之外，**历史的补丁修复语句**和当前程序甚至其他程序中的相似度大的语句也可以被添加进补丁搜索空间中，来提高补丁生成能力

基于搜索的补丁生成

- 辗转相除过程
  - (a, b) = (0, 100),程序陷入死循环

```
1   /* requires: a >= 0, b >= 0 */
2   void gcd(int a, int b) {
3       if (a == 0) {
4           printf("%d", b);
5       }
6       while (b != 0) {
7           if (a > b)
8               a = a - b;
9           else
10              b = b - a;
11      }
12      printf("%d", a);
13      exit(0);
14  }
```

- 缺陷定位
  - 构造测试集，检测执行路径，发现成功测例（a, b均为正数）执行3,6-12行，而失败测例（a=0）执行3-7,9-12行，根据失败测例覆盖到的而成功测例没有执行的（4-5，8），我们初步将缺陷定位在4-5行。

https://zhuanlan.zhihu.com/p/267223614

基于搜索的补丁生成

- 补丁生成
  - 采用变异方法，对4-5行进行代码替换、删除和插入
  - 显然对printf进行替换或删除会使程序失去结果输出，在不改动第四行的前提下，尝试插入程序中其他部分的代码，发现插入exit(0); 失败测例通过
- 以整个测试集对候选补丁进行检查，确定补丁可行性

```
1   /* requires: a >= 0, b >= 0 */
2   void gcd(int a, int b) {
3       if (a == 0) {
4           printf("%d", b);
5           exit(0);    // insert
6       }
7       while (b != 0) {
8           if (a > b)
9               a = a - b;
10          else
11              b = b - a;
12      }
13      printf("%d", a);
14      exit(0);
15  }
```

基于语义的补丁生成

- 动态测试条件下，所有的测试集可以看作是对程序进行了一个约束：需要满足所有测例的通过条件。
- 基于搜索的方法是在给定的搜索空间中来寻找满足约束的一个解
- 基于**语义**的方法则是利用约束求解器，通过程序合成（program synthesis technique）直接合成出一个解
  - 该类方法依赖于符号执行和约束求解技术，把补丁生成问题转化为一个SMT问题，使用现有的SMT求解器（比如Z3）来进行求解

基于语义的补丁生成

- 模拟修复方法Angelix的修复过程：
  - 有一个求**max**(x, y)的程序发生了错误，对于两个测例(1, 2)和(2, 1),前者输出了正确结果2但执行流错误，后者输出了错误结果1
  - Anglix猜测有两处错误，将两处表达式替换为未定义符号α, β

```
if (x < y)  // if (x > y)
    ret = x + 1;  // ret = x
else
    ret = y;
return ret;
```

```
if (α)  // if (x > y)
    ret = β;  // ret = x
else
    ret = y;
return ret;
```

Angelix: scalable multiline program patch synthesis via symbolic analysis. ICSE'16
https://zhuanlan.zhihu.com/p/572208327

基于语义的补丁生成

- Anglix猜测有两处错误，将两处表达式替换为未定义符号α, β。
- 记$\sigma = \{x \to 1, y \to 2\}, \theta = \{x \to 2, y \to 1\}$
- 利用符号执行技术，将两个测例抽象为约束表示:

```
if (α)  // if (x > y)
    ret = β;  // ret = x
else
    ret = y;
return ret;
```

- Test1:
  - Path1: $\{(\alpha, True, \sigma), (\beta, 2, \sigma)\}$
  - Path2: $\{(\alpha, False, \sigma)\}$
- Test2:
  - Path1: $\{(\alpha, True, \theta), (\beta, 2, \theta)\}$
  - Path2: $\emptyset$

- **每个Test至少有一个通路**，所有Test的约束构成一个未定义符号α,β求解问题，利用求解器，结合函数语义给出最终的补丁。

Angelix: scalable multiline program patch synthesis via symbolic analysis. ICSE'16
https://zhuanlan.zhihu.com/p/572208327

基于模板的补丁生成

- 基于模板
  - 指根据开发者或研究人员的经验预定义一些补丁模板或者补丁生成策略用于指导修复的过程
- 相比搜索方法，其修复能力由人工定义的模板数量与范围直接决定，因此每项技术一般特定针对某些特定类型的缺陷开展
  - 科研人员从大范围的开源项目中挖掘开发人员所打过的补丁，总结出常用的修复模板，整合到修复程序中来进行自动化修复
- 代表性的PAR方法定义了10个修复模板，包括替换函数参数、添加空值检查等等

其他修复方法

- 基于错误报告的自动程序修复
- 基于人工智能自动程序修复
  - 深度学习（DL-Based）
  - 大模型

补丁评估

- 补丁的评估是使用测试用例集验证候选补丁质量的过程
- 验证补丁是否能通过所有测试样例即可确定补丁是否正确，但是这样产生的很多都是疑似正确的补丁，而非可以真正应用的正确补丁
- 测试样例**过拟合**
  - 补丁过于针对测例而进行了非常规的功能代码删除等操作
  - 由于补丁生成阶段往往会生成大量候选补丁，对所有补丁进行验证也是一个费时费力的过程。因此有很多方法致力于优化验证过程和减小过拟合程度

补丁评估

- 虽然单纯增加测试样例数目并不一定能提高补丁精度，但是通过**更完善的样例**来提高对整个程序的覆盖率对补丁评估有益。因此在测例选择上，要尽可能多的构造覆盖较多语句的错误测例而非一味增加通过测例
- 借助机器学习领域的概念，将测例分为训练集和测试集，来**检验**过拟合问题
- 利用**程序距离**的计算，对于多个通过测试的候选补丁进行排序，找出更接近源程序的补丁，设为更优结果
- 在关注测例的输入输出之外，对执行过程同样进行监控，利用执行**行为**的**相似度**进行补丁分类择优

第八章 自动修复方法（2学时）

- 基于动态测试的程序修复
- 领域程序自动修复
  - 并发程序修复
  - 循环(for-loop)修复
  - 浮点数错误修复

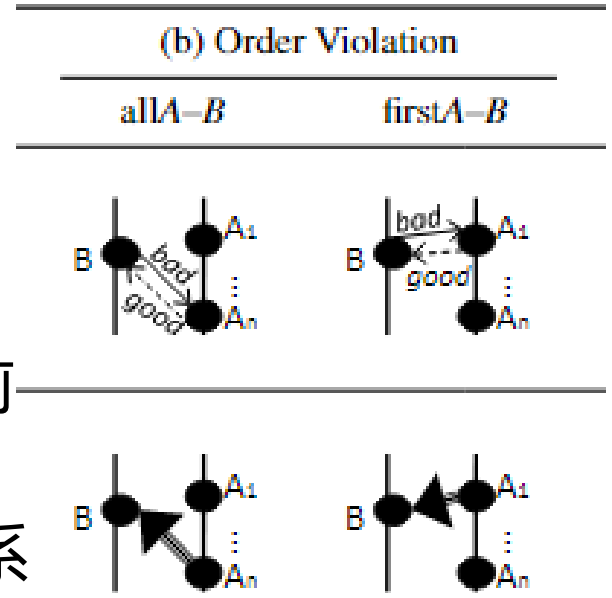
并发领域的程序修复

- 并发缺陷
  - 由于多个线程在不符合预期的时机对共享数据进行访问造成的
  - 相比于一般的程序错误，由于线程的交错执行，并发错误具有不确定性，需要在修复原有缺陷的同时避免引入新的缺陷，因此该领域的自动修复更具有挑战性

- 常见的几种并发缺陷
  - 数据竞争、原子性违背、以及死锁

并发领域的程序修复

- 数据竞争
  - 两个或以上线程对同一内存进行并发访问，且至少有一个是写访问。
  - 修复技术CFix提出了两种策略：为指令设置一种确定的顺序关系，或是添加互斥锁
- 顺序违背
  - 指各个线程的某些指令并未按照预期顺序执行
  - Cfix分析了两种顺序违背导致的数据错误：资源悬挂（数据使用在数据回收之后）和未初始化的读（读之前没有任何写操作）
  - 对于二者，Cfix分别强制执行allA-B和firstA-B顺序关系



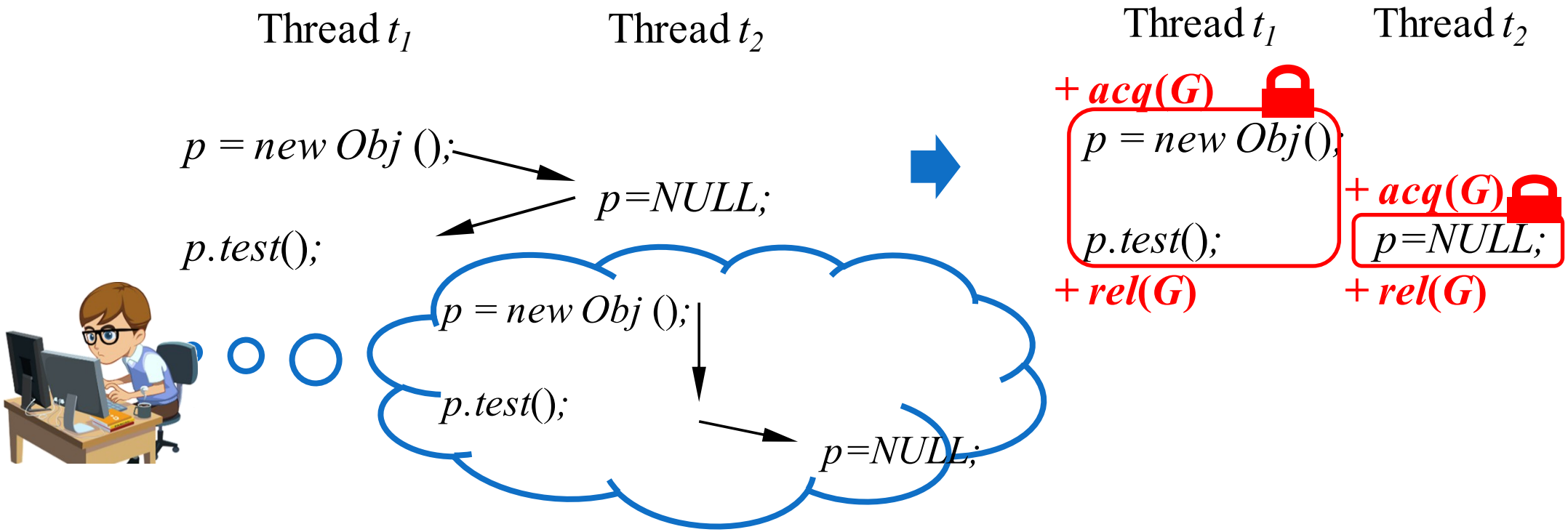
Automated concurrency-bug fixing. OSDI'12

并发领域的程序修复

- 原子性违背
  - 指对某一共享变量的一系列访问序列，被来自其他线程的访问打破原子性。
  - 原子性违背的主要修复方法是通过添加同步设施，为某一可能发生原子性违背的代码区域添加锁保护或形成某种确定的顺序关系
- 死锁
  - 指至少两个线程在执行过程中由于资源竞争造成彼此互相等待的问题。
  - 死锁修复要考虑的问题是在修复的过程中不要引入新的死锁
  - 修复技术DFixer 选择一个线程进行修复，通过控制流分析提前获取锁而不引入新锁，破坏了死锁的形成条件

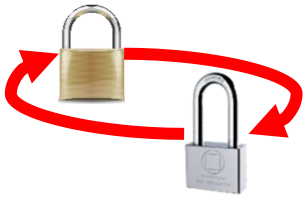
并发领域的程序修复

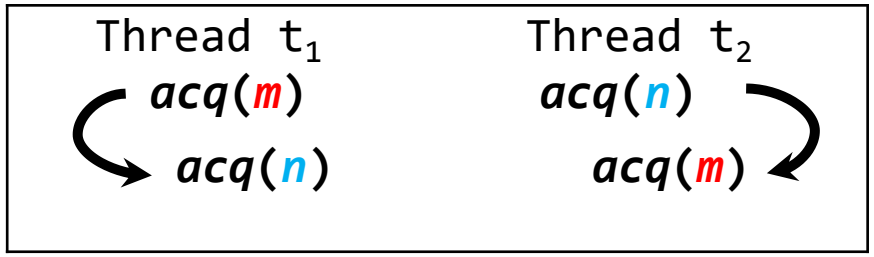
- Gate Lock Algorithms (GLA)
  - *AFix* (*PLDI, OSDI*), *Axis* (*ICSE*), *Grail* (*FSE*), *Gadara* (*ODSI*)
- Use a gate lock to serialize two or more threads



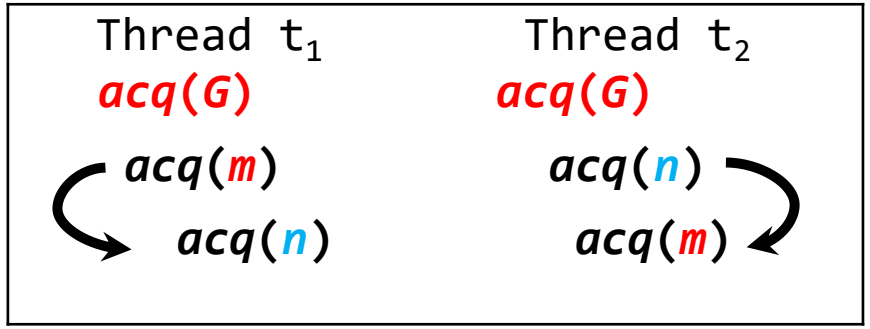
并发领域的程序修复

- Deadlocks in programs





- Existing Generic Approaches (GA)
  - Introduce new lock (**G**ateLock) to serialize two threads
  - *AFix* (*PLDI, OSDI*), *Axis* (*ICSE*), *Grail* (*FSE*), *Gadara* (*ODSI*), ...



如何避免引入新的死锁

并发领域的程序修复

- 如何提高修复代码的可读性

```
+ Global G;
...
ap_buffered_log_writer(...){
+  acq(G);
    idx = buf->outcnt;
    s = &buf->output[idx];
    buf->outcnt += len;
+  rel(G);
}
```

Basic GLA

```
ap_buffered_log_writer(...) {
+ G = contextL(
          hash(&(buf->outcnt)),
          hash(&(buf->output)));
+ acq(G);
    idx = buf->outcnt;
    s = &buf->output[idx];
    buf->outcnt += len;
+ rel(G);
}
```

Grail

```
struct buffered_log{
    apr_size_t outcnt;
    char outbuf[…];
+ Lock G;
}

ap_buffered_log_writer(...){
+  acq(buf->G);
    idx = buf->outcnt;
    s = &buf->output[idx];
    buf->outcnt += len;
+  rel(buf->G);
}
```

Our AlphaFixer

循环结构的程序修复

- 循环结构：循环条件+循环体
  - 任何一处的错误都会导致整个循环结构发生错误，甚至导致无限循环的产生
- LoopFix (JSS 2019)：
  - 采用了基于语义的修复方法，考虑了循环条件和循环体分别出错的情况。
  - (1) 循环条件出现错误
    - LoopFix首先进行**循环测试，每次进行一轮循环迭代，直到出现正确输出**，找到应该停止的循环迭代次数N。
    - 对于每次迭代，将所有可达变量和其值作为key，将循环是否终止作为value，产生N个键值对，由这N个对来生成约束条件，进行约束求解。

LoopFix: An Approach to Automatic Repair of Buggy Loops. Journal of Systems and Software, 2019.

循环结构的程序修复

- (2) 循环体错误
  - 那么按照正确的循环次数进行**循环展开**，就将修复循环转换为**修复序列程序错误**，按照符号执行树进行程序合成修复
- (3) 二者均错误
  - **优先修复循环条件**
  - 根据输出变量的预期值和初始值是否相等来判断循环体是否被执行了，从而确定循环条件要满足的值的范围
  - 当条件确定后，再去进行循环体的修复即可

```
1.int func (int x)
2.{
3.    int y = 0;
4.    while (x > 0)//fix: while (x > 5)
5.    {
6.        y=y+1;   //fix: y=2*x-10
7.        x--;
8.    }
9.    return y;
10.}
```

LoopFix: An Approach to Automatic Repair of Buggy Loops. Journal of Systems and Software, 2019.

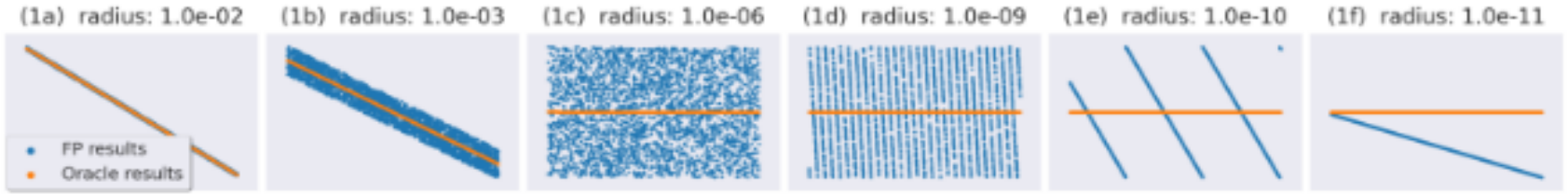
浮点数误差的程序修复

- 浮点数
  - 计算机为了近似表示任意实数所设计的表示方法，然而由于指数、尾数的限制，任何浮点数标准都会存在精度问题，从而在计算时产生舍入误差。
  - 主要来源：（1）输入数据的精度、（2）计算过程中发生的舍入
- 修复思路
  - 已有的部分修复方法依赖于一个用于判断和指导修复过程的**准确结果或模型**
  - 通过比较浮点程序的不精确输出与一个更高精度的准确的数学函数的输出，可以得到待修复程序的误差信息，从而指导缺陷定位以及补丁生成向着减小误差的方向进行
  - 补丁生成阶段有着许多方法，包括对一些可能产生误差的表达式进行数学重写（如平方差公式的左右转换）、使用一阶、二阶分段表达式进行误差区间上的近似以减小误差等等，主要是基于搜索的方法

浮点数误差的程序修复

- ACESO（OOPSLA'22)
  - 不需要获得浮点程序(FP)所对应的准确数学结果，且使用基于语义的方法来生成补丁
  - 由于没有一个准确的结果，其首先要解决的问题时如何获取误差信息
  - ACESO提出了FP错误的微观结构的概念，即在一定的精度（观测半径）下，在某一点的邻域内，FP的结果受误差影响产生了不连续的现象，而且某些特定的半径下微观结构呈现出规律性的变化（如1d和1e）。因此，选择一个合适的半径后，可以**利用统计学定律**，得出FP结果的均值（可近似为准确的数学结果）和FP误差的方差的估计值

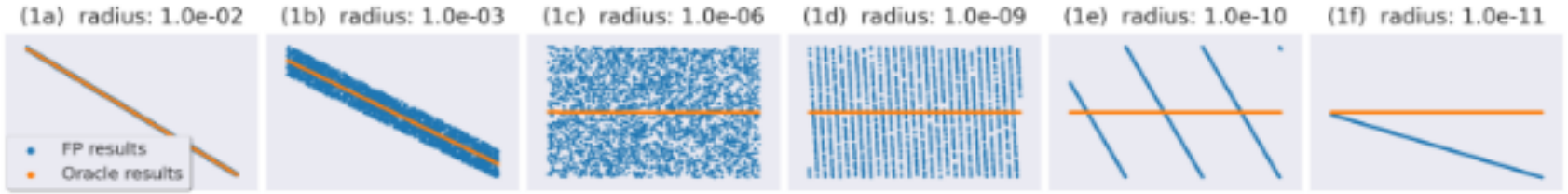
- $f(x) = \frac{1-\cos(x)}{x^2}$ with example center point $x_c = 1.2345 \times 10^{-3}$



浮点数误差的程序修复

- 对于缺陷定位得到的待修复区间，ACESO在区间内计算切比雪夫节点来作为采样点，在**采样点**上得到上述的微观结构信息。之后，ACESO将幂级数和切比雪夫级数结合，构造一个待定系数的**多项式通用修复模型**。再使用加权最小二乘方法，最小化修复多项式与实际值的差值，从而得到修复模型的系数值，合成出修复补丁。
- 由于没有准确结果，补丁的**验证**也需要微观结构的帮助。ACESO在待修复区间中选择一组待测点，在每一点的合适半径的邻域内观测出FP的最大值和最小值，要求补丁多项式的近似结果要落在两者之间，才能被确认为正确的补丁。

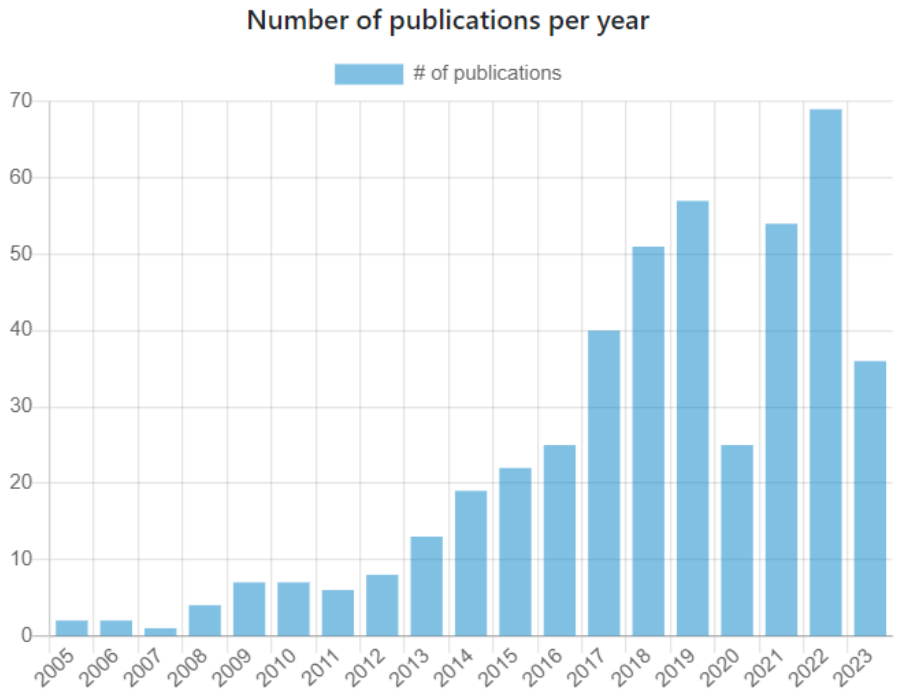
- $f(x) = \frac{1-\cos(x)}{x^2}$ with example center point $x_c = 1.2345 \times 10^{-3}$



自动修复的发展现状

- 程序修复社区 program-repair.org
  - http://program-repair.org

- 程序缺陷修复的研究已经吸引了软件工程、程序语言、人工智能、形式化验证等四个社区的大量研究人员投入



自动修复的发展现状

- 随着大模型LLMs的流行，越来越多的研究团队将其和APR领域相结合

- ICSE 2023录用相关文章
  - 有数篇文章均研究了LLM在程序修复领域的作用效果，纷纷得到了大模型能力不逊于甚至强于各类APR方法的结论，指出以LLM为基础进行APR的后续研究是APR领域的未来趋势
  - 经典APR方法的研究也在进行中，ICSE 2023一篇文章提出了将错误报告和测例结合起来的缺陷定位方法来提高补丁生成效果

自动修复的挑战与展望

1. 缺陷定位精度
   a) 更灵活的缺陷定位
   b) 将缺陷定位和补丁生成结合
2. 过拟合的影响
   a) 优化补丁生成策略
   b) 减小对测试集的依赖 ……
3. 修复效率问题
4. 工业场景中的实用性问题
   a) 如何不依赖失败测例
   b) 继续提高自动补丁的正确率和可用性
5. 进一步和大模型相结合
   a) 充分利用大模型的强大能力
   b) 能否同时找到传统的APR方法的优化方向