基于频繁挖掘的代码自动合成方法

朱正楠, 吉如一, 熊英飞, 张路

问题的提出

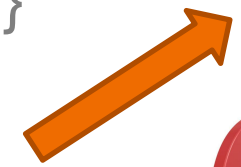
语言描述

求绝对值

可编译运行的代码

```
int getabs(int x){
  return Math.abs(x);
}
```

Deep API

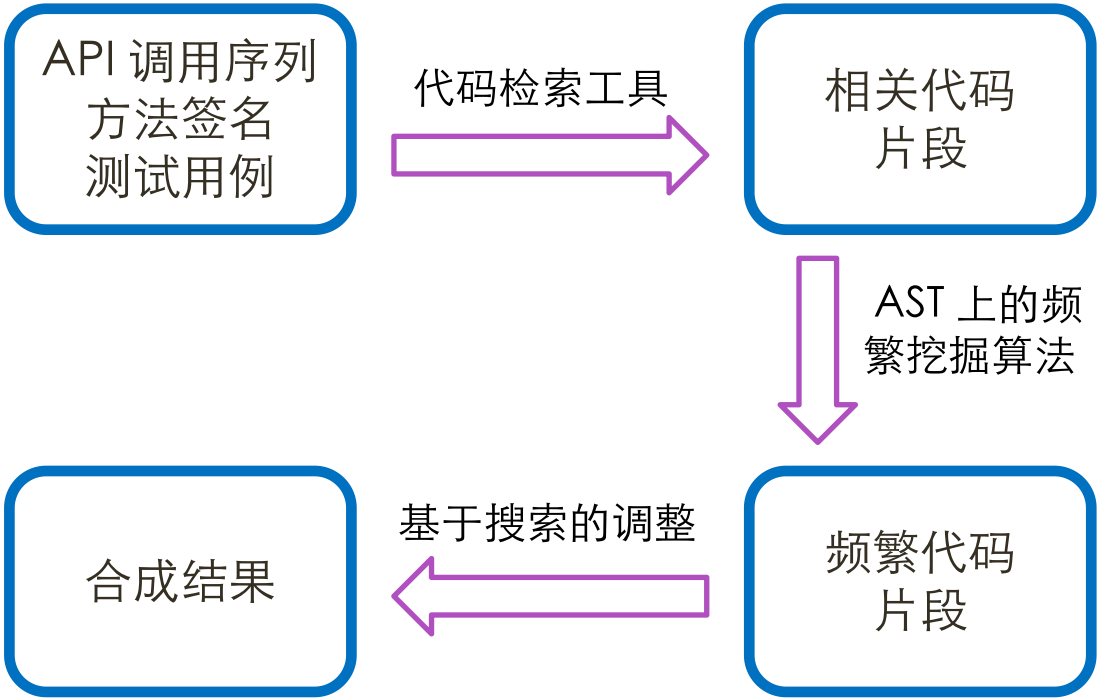


API 调用序列

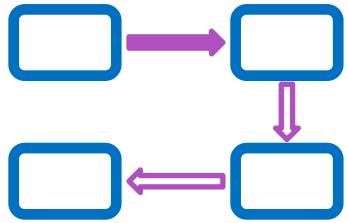
Math.abs()

Deep API Learning [Gu X et al. 2016]

方法流程


API 调用序列
方法签名
测试用例
代码检索工具
相关代码
片段
AST 上的频
繁挖掘算法
频繁代码
片段
基于搜索的调整
合成结果

代码检索工具



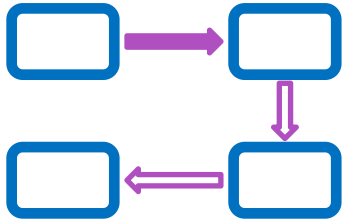
FileInputStream.new FileOutputStream.new
FileInputStream.read FileOutputStrem.write
FileInputStream.close FileOutputStream.close

复制文件

```
222         byte[] bytes = new byte[2];
223         FileInputStream istream = new FileInputStream("测试.txt");
224
225         //获得读取的字节数
226         int read = istream.read(bytes);
…
249         FileOutputStream stream1 = new FileOutputStream("复制1.png");
250         while (stream.read(bytes) != -1) {
251             stream1.write(bytes);
252         }
253
254         stream.close();
```

```
27          FileOutputStream fos = null;
28          try {
29              fis = new FileInputStream("E:\\upload\\必看.txt");        //
    创建输入流对象,关联txt
…
32              int b;
33              while((b = fis.read()) != -1) {
    //在不断的读取每一个字节
34                  fos.write(b);
    //将每一个字节写出
```

代码检索工具



API调用序列

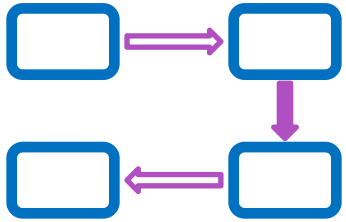
Math.abs

```
int b = Math.abs(c);
int d = Math.abs(c);
```

相关代码片段

频繁挖掘算法

频繁挖掘算法

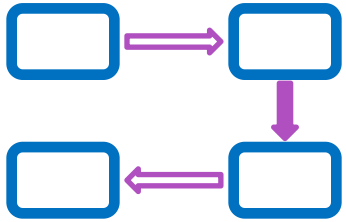


输入：检索得到代码的AST
输出：频繁出现的联通结构

```
int b = Math.abs(c);
int d = Math.abs(c);
```

```
int _ = Math.abs(c);
```

频繁挖掘算法



Apriori 算法 基于合并

问题区别：

频繁项数量可能很多：只需要找到包含所有API的频繁集

需要处理AST本身的复杂结构

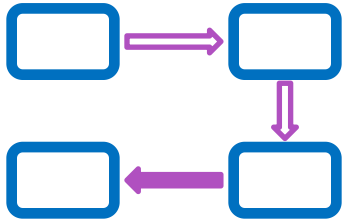
基于扩展

f(_,b); ≠ f(b,_)

{a(); b(); _;} =

{a(); _; b();}

搜索调整



```
int _ = Math.abs(c);
int getabs(int x);
unit tests
```

```
int getabs(int x) {
    return Math.abs(x);
}
```

对缺失的变量名重新命名

　搜索相同类型变量的划分

把代码片段变成函数

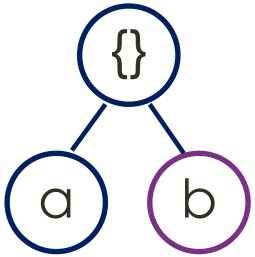
　根据类型搜索选择输入/输出变量

频繁挖掘算法

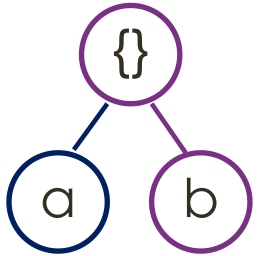
对象: 抽象语法树
起点: API 调用
增广方式: 选择当前点的相邻点进行扩展
注意点: AST 本身的结构

向上扩展

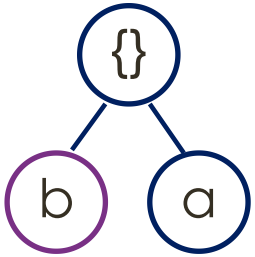
{a(); b();}

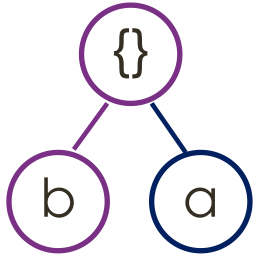


向上



{b(); a();}



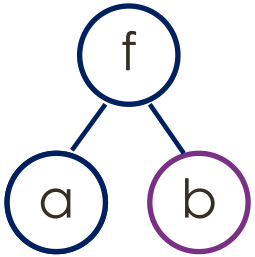


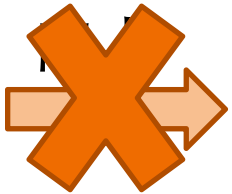
依据：

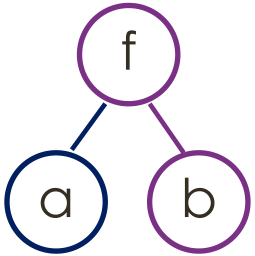
双亲节点类型

向上扩展

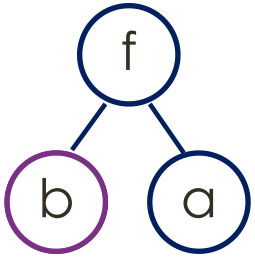
f(a,b);

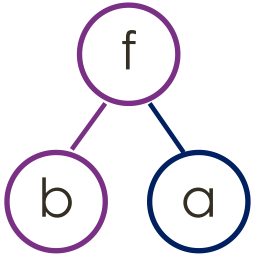






f(b,a);





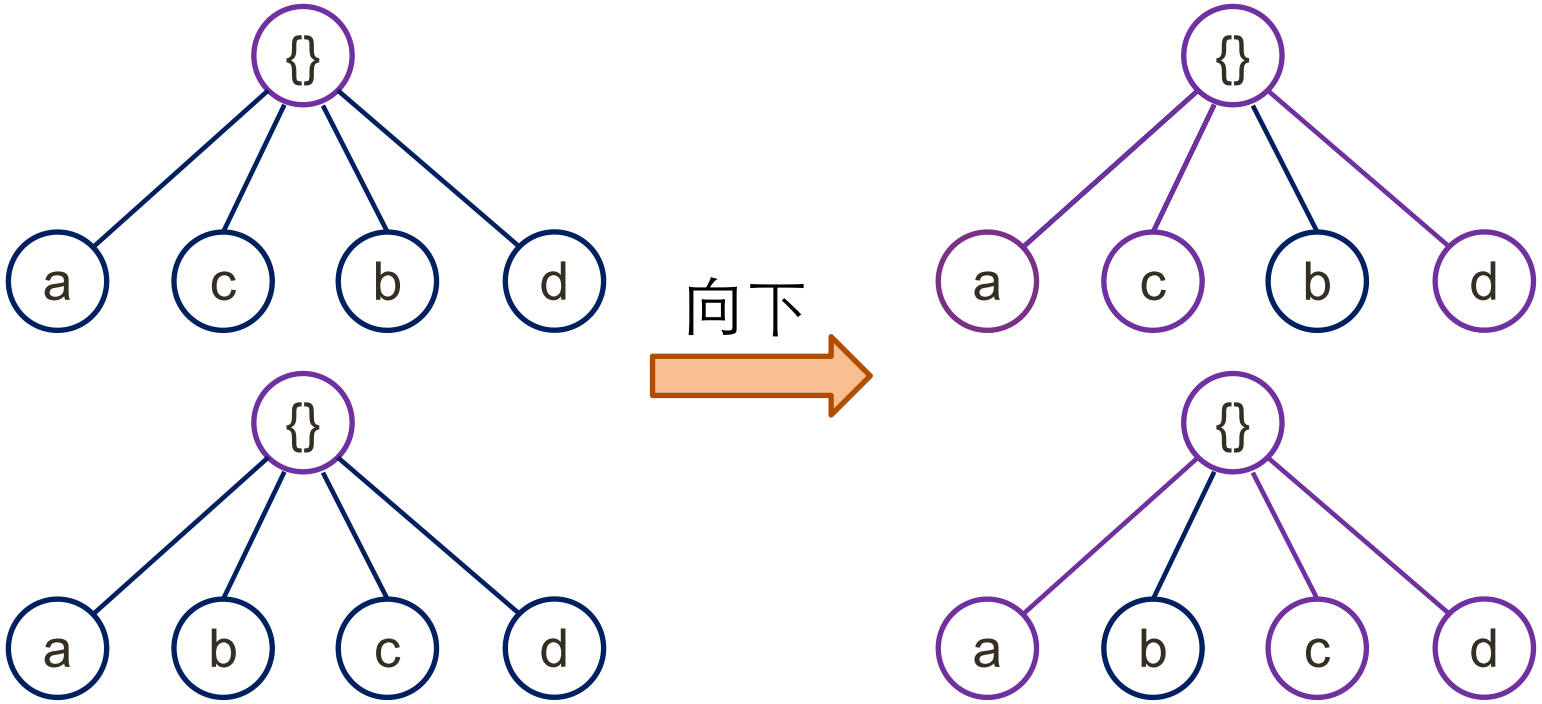
依据：

双亲节点类型

当前节点位置

向下扩展

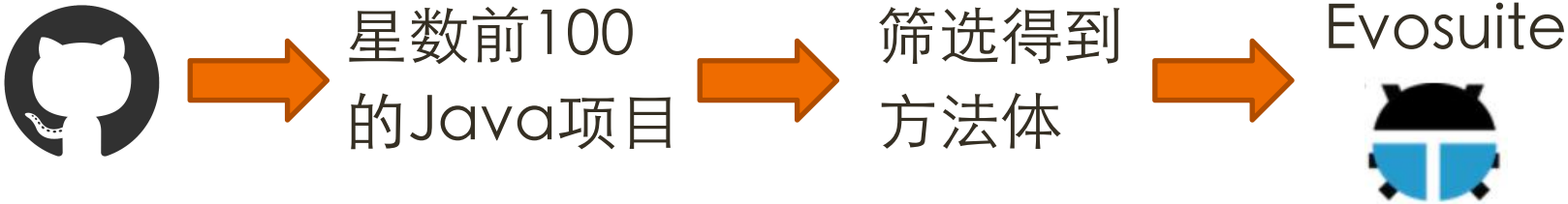
把子树的顺序看成序列
Apriori algorithm



测试

两个测试集

Dateset1: 从 DeepAPI 的测试集中得到（大小 29）
　DeepAPI 人工选择了30个常用询问作为测试集
Dataset2: 利用 GitHub 的代码自动生成（大小 88）

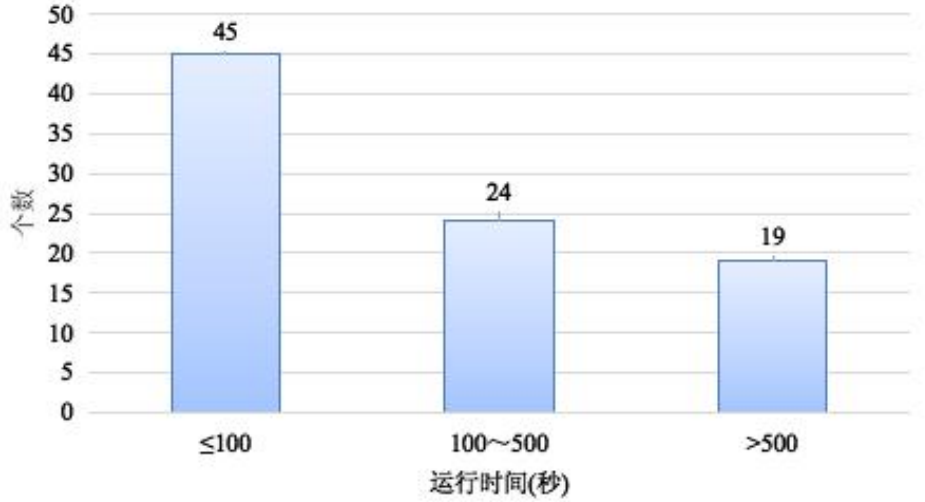


结果

两个测试集

Dateset1: 从 DeepAPI 的测试集中得到

Dataset2: 利用 GitHub 的代码自动生成



局限性

检索结果不够精确

Dataset2 的询问搜索结果前 300 的代码中
35(37.5%)组干扰项超过50%
7(8%)组干扰项超过80%

难以处理多功能的API调用序列

一个例子

Query: Math.abs+Math.sqrt
Task: 二维平面计算两点的欧式距离

```
34                  double delta = mid * mid + 8 * left;
35                  if (delta >= 0) {
```

```
14      public static boolean collisionCircle(int x1, int y1, int r1, int x2, int y2, int
    r2)
15      {
16          double a,b, dis = 0;
17          a = (double)Math.abs(x2 - x1);
18          b = (double)Math.abs(y2 - y1);
19          dis = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
20
21          if( r1 + r2 >= dis)return true;
```

```
128                 x1 = (-2 * Math.signum(R()) * Math.sqrt(Math.abs(Q())) * Math.sinh(phi1)
    - a / 3) + "";
```

把 dis 加入搜索关键字

结论

关键字: 自动生成，API调用序列

三个部分:

- 代码搜索工具
- AST 上的频繁挖掘算法
- 数据生成器

正确率: Dataset1 (12/29), Dataset2(21/88)


thank
you