软件分析技术 2018 课程大作业一

小组成员： 黄杨洋 张博洋 刘渊强

一、目标：实现一个Java上的指针分析系统

- 基于 **SOOT** 框架

二、代码目录及使用环境

- `src/main/java`为分析程序源代码目录
- `code/benchmark`中保存了评测类及被测类的实现
- `code/test`中保存了测试代码（被分析的代码）
- 请使用 **Intellij** 打开 project，并将`code`和`src/main/java`目录标为源代码目录
- 命令行参数：最后一个类名被视为`main()`方法所在类，修改该参数即可分析不同的测试代码；示例：`-w -cp out/production/pointerAnalysis -pp test.Hello`
- **jar包使用**：命令行参数：`java -jar analyzer.jar -pp -cp BENCHMARK_CODE_PATH test.Hello`

三、算法主要设计思想及代码实现

1. `src/main/java`中基于 **SOOT** 框架实现了一个 **Anderson** 风格的指针分析算法，即基于约束(constraint-based)或基于子集(subset-based)的指针分析方法
2. 由于java中的指针均以引用形式存在，故而只存在基本约束（`a = new A()`)和简单约束（`a = b`）这两种约束类型
3. `src/main/java/RunPointerAnalysis.java`为分析入口，`src/main/java/WholeProgramTransformer.java`为基于SOOT框架实现的转换函数，`src/main/java/AnswerPrinter.java`作用为输出分析结果，`src/main/java/AndersonAnalysis.java`为实现的分析算法
4. 分析难点： 流敏感分析、域敏感分析、上下文敏感分析、数组分析、递归分析
5. 数据流分析框架：
    1. 正向分析
    2. 半格元素：一个字典集合，每个键代表一个变量，值为该变量可能指向的内存位置集合
    3. 交汇操作：并
    4. 变换函数：
        1. Benchmark.alloc(id): 获取id作为下一个new语句的内存位置

2. Benchmark.test(testcnt,variable):在当前的半格元素中查找variable，找到即打印其可能指向的位置；并将答案记录下来，若答案集中已经存在该变量，则合并其位置集

3. New语句：在当前的半格元素中新建一个键值对，将变量名作为 key，将之前获取的id初始化为该变量可能指向的位置集合

4. 赋值语句：存在两种情况，变量赋值`a = b`和域赋值`a.f = b.f`

    - kill 集合是左值对应的内存位置集，gen 集合是右值对应的内存位置集
    - 若右值是域 `base.field`，此时的 gen：先求出 base 所指向的内存位置（如 [1,2]），查询当前集合中是否有 1.f,2.f 的位置，若存在则加入 gen 中
    - 若左值是域`base.field`，此时的 kill：同样先求出 base 所指向的内存位置，查询当前集合中是否有该域的位置，若存在则加入 kill 中
    - 执行半格元素的并操作

5. 数组

    - 数组元素指向位置的集合表示有`#location.num`、`#location._`和`#location.*`，其中`location`是一个数字，代表数组被New时的id，`num`为一个数字，代表数组的某个具体下标，所以`#location.num`代表该内存位置上的数组的某一具体下标的位置，而`#locaiton._`代表未知下标的位置，`#location.*`代表所有数组元素可能指向的位置
    - 取值时，若是具体数字下标索引，则返回`#location.num`和`#location._`的并作为其位置集合（`#location._`中可能也包含该下标指向的位置）；若是变量下标索引，则返回`#location.*`作为其位置集合（即此时可能指向任意一个数组元素指向的位置）
    - 赋值时，若是具体数字下标索引，则向`#location.num`和`#location.*`中并入此时等式右值的位置集；若是未知下标索引，则向`#location._`和`#location.*`中并入此时等式右值的位置集

6. 函数调用语句（过程间）：

    - 维护一个函数调用栈：每次调用一个函数将其入栈，函数返回将其出栈
    - 新建一个 AndersonAnalysis 类的实例，传入被调用函数和当前堆上的分析结果进行同样的分析
    - 分析完毕后，把分析结果中堆上的结果和对返回值分析的结果取出，放入当前实例的分析结果中
    - 若发现递归调用（即调用的函数仍然在栈中未被出栈），将所有堆上和返回值的分析结果置为`#unk`(Unkown)，目前无法处理递归调用