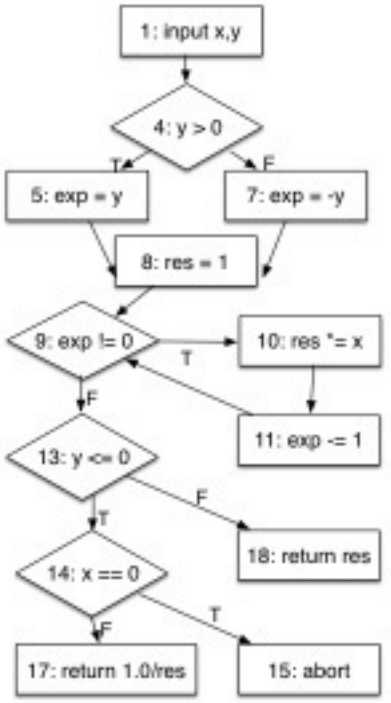
Combining Symbolic Execution and Model Checking for Data Flow Testing

Overview

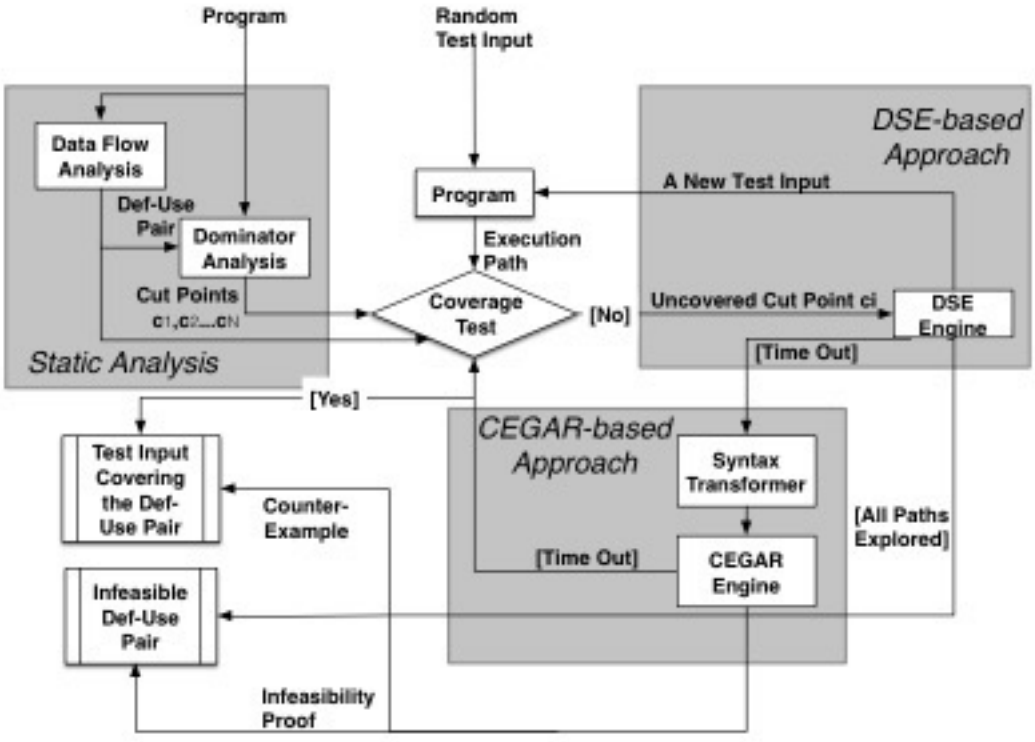
- Data Flow Testing(DFT)的目的是通过观察程序中变量所有被使用的情况以证明其被使用正确
- 问题：
  1. 现存的 data flow coverage 工具很少，已知的只有20年前的ATAC
  2. 数据流的test data的生成复杂度高；要生成覆盖变量定义、使用的测试用例比只覆盖程序语句要复杂（路径爆炸）
  3. 不可达的测试目标使得DFT更加困难
- 方法：
  - Dynamic Symbolic Execution (DSE)
  - Counter example-Guided Abstraction Refinement(CEGAR)
    - 给定源码和假设，静态分析证明程序满足假设、或者给出反例

An Illustrative Example

```
1  double power(int x,int y){
2      int exp;
3      double res;
4      if (y>0)
5        exp = y;
6      else
7        exp = -y;
8      res=1;
9      while (exp!=0){
10       res *= x;
11       exp -= 1;
12     }
13     if (y<=0)
14       if(x==0)
15         abort;
16       else
17         return 1.0/res;
18     return res;
19 }
```



Approach



DSE-based Data Flow Testing

- 如上例，假设起始输入为 x=0, y=42，则有以下执行路径：

$$p = l_4, l_5, l_8, \underbrace{l_9, l_{10}, l_{11},\ l_9, l_{10}, l_{11}, \ldots,}_{\text{repeated 42 times}} l_9, l_{13}, l_{18}$$

- 首先剪枝去除非法的分支节点。*res* 在语句 l_{10}被修改，所以在此之后的分支都没有被搜索的必要了，则有

$$p = l_4, l_5, l_8, \underbrace{l_9, l_{10}, l_{11},\ \cancel{l_9}, l_{10}, l_{11}, \ldots,}_{\text{repeated 42 times}} \cancel{l_9}, \cancel{l_{13}}, l_{18}$$

- 使用 **Cut Point-Guided Search** 策略以确定首先应该被搜索的节点。在 l_8 到 l*{17}* 的所有路径上，必须经过的分支点集合为 *{l_4, l_8, l_9, l*{13}, l*{14}, l*{17}} ，而在以上 *p* 中，l_4, l_8, l_9 都已经被覆盖，则下一个被搜索的目标是 l_{13} ，所以接下来被置反的点是 l_9 .
- 结果可生成一个新的输入 x = 0, y = 0，获得新路径：$p' = l_4, l_6, l_7, l_8, l_9, l_{13}, l_{14}, l_{15}$ 而后继续以上步骤

CEGAR-based Data Flow Testing

```
1  double power(int x, int y){
2     bool cover_flag = false;
3     int exp;
4     double res;
5     ...
6     res=1;
7     cover_flag = true;
8     while (exp!=0){
9        res *= x;
10       cover_flag = false;
11       exp -= 1;
12    }
13    ...
14    if(cover_flag) check_point();
15    return res;
16 }
```

- （源码插桩，之后符号执行检测flag即可）

Implementation

- 基于作者在 *SERE'14* 上发表的文章中开发的动态符号执行引擎 *CAUT*
- （是一个基于CIL的源码分析引擎）
- 求解器是 **Z3**

Evaluation

- 和现存的符号执行工具 *CREST* 和 *KLEE* 比较
- benchmarks来自 *Software-artifact Infrastructure Repository*
- 比较的搜索策略包括：
  - Random Input
  - Random Path Search
  - CFG-Directed Search(CREST)
  - RP-MD2U Search(KLEE)：基于宽搜的最小距离策略
  - Shortest Distance Guided Search
- 结果现实，在 *def-use* 覆盖率、处理时间上，文章的方法都优于以上搜索策略和工具