# 笔记模板2

## 1. 文章解决的问题

用Perl来实现程序合成，用Z3 SMT solver来解决修复约束。

## 2. 解决的思路

### 1. 产生修复约束

本文只针对赋值语句和分支语句的右侧修复。不会更改任何左侧的修复。以下为两种修复的例子：

$$\mathrm{x} = f_{buggy}(\ldots) \rightarrow \mathrm{x} = f(\ldots)$$
$$\mathrm{if}(f_{buggy}(\ldots)) \rightarrow \mathrm{if}(f(\ldots))$$

求出的f不会修改任何变量

假设bug语句为s（在一次符号执行中，s最多执行一次），T是测试用例集，有n个，$t_i$是测试用例中的一个。

设在s之前执行的程序状态为$\xi_i$，程序状态保存程序变量以及路径条件

$\tau_i$为s中f(...)的输出符号值，$C_i$指每个测试用例在运行程序P时的约束

对于每一条路径$\pi_i$,相关的路径条件为$pc_i$,它的符号表达式的输出为$O_i$。那么可以求得$C_i$这个约束公式

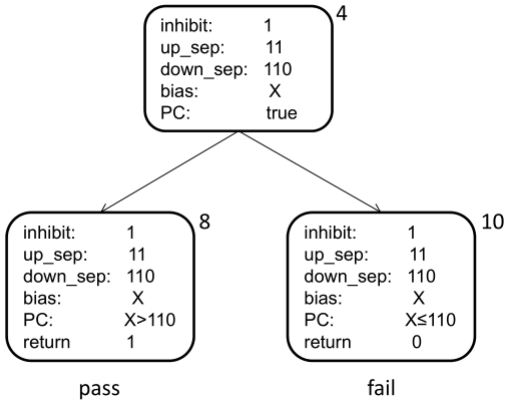
$$C_i := (\bigvee_{j=1}^{m}(pc_j \wedge O_j == O(t_i))) \wedge (f(\xi_i) == \tau_i)$$

第一部分说明至少有一条路径使得程序的输出等于预期的输出。第二部分说明将$\tau_i$必须满足第一部分

当s这条bug语句在$t_i$测试时没有运行时如果执行通过了，那么$C_i$为true

举个例子:

```
1  int is_upward_preferred(int inhibit, int up_sep,
       int down_sep) {
2    int bias;
3    if(inhibit)
4      bias = down_sep; //fix: bias=up_sep+100
5    else
6      bias = up_sep;
7    if (bias > down_sep)
8      return 1;
9    else
10     return 0;
11 }
```




即$pc_i$有两个$X > 110$或$X <= 110$.那么
$C_i$的第一部分为$(X > 110 \wedge 1 = 1) \vee (X \leq 110 \wedge 0 = 1)$。由于0！=1，所以$C_i$第一部分可以简化为$X > 110$。$C_i$的第二部分为$f(1, 11, 110) = X$.即$f(1, 11, 110) > 110$.

产生了每一个$C_i$后就能得到一个C来表示程序的约束

$$C := \bigwedge_{i=1}^{n}((\bigvee_{j=1}^{m}(pc_j \wedge O_j == O(t_i))) \wedge (f(\xi_i) == \tau_i))$$

现在讨论bug语句如果执行多次

设在执行$t_i$个测试用例时,$\tau_i^k$来表示$f(\dots)$在第k次时的输出符号。

$$C_i := (\bigvee_{j=1}^{m}(pc_j \wedge O_j == O(t_i))) \wedge (\bigwedge_{k=1}^{w} f(\xi_i^k) == \tau_i^k)$$

$$C := \bigwedge_{i=1}^{n} C_i$$

2. **产生修复**

   利用程序合成

3. **如何修复一个程序**

   RC（repair candidate）：一个bug语句按照可疑度排名的list。每次修复针对一条语句。

   S最初只包含失败的测试用例。当一个修复产生时，就会测试不在S中的成功的测试用例t。如果没有通过，则将t加入到S。

   在Repair函数中C是指基于当前集合S的约束。level指的是复杂度，即为了是修复满足约束C要用到的组件。最低等级的就是只用一些常量就能修复

# 3. 核心知识点

1. 基于组件的程序合成部分（Lval2Prog 函数）
   - 组件(component):从这些组件中生成一个函数f

     假设现在有一个组件N, 里面的组件都是一个个函数 $\{f_1, f_2 \dots f_N\}$ 。对于第i个函数，用$\vec{\chi}_i$来表示它的输入，用$r_i$来表示输出。Q是输入的集合，R是输出的集合。

     现在用$\vec{\chi}$来表示目标函数$f$的输入，用$r$来表示$f$的输出

- $l_x$位置变量(location variable): 一个变量x在Q，R，$\vec{\chi}$，$r$中的一个变量。$l_x$是x在程序中出现的行号
- $L$是$l_x$的集合
- 良好结构性约束(well-formedness constraint)：$\psi_{wfp}$ 这个只保证对L进行约束，$\psi_{cons}$指R中的元素都不在程序的同一行出现，一行只有一个组件。$\psi_{acyc}$指对于$\vec{\chi}_i$和$r_i$来说，前者的$l_x$小于后者的$l_x$,即输入的变量出现在输出变量的前面。
- $f$要满足以下约束：输入为$\vec{\chi}$，输出必须为$r$.当$l_x = l_y \Longrightarrow x = y$.
- $\phi_{func}$指程序f必须满足实际输出等于预期输出

# 4.程序功能说明

```
P : The buggy program
T : A test suite
RC : A ranked list of potential bug root-cause
Output:
r: A repair for P

while RC is not EMPTY and not TIMEOUT  do
    rc = Shift(RC) // A repair candidate
    S = ∅          // A test suite for repair generation
    T_f = ExtractFailedTests(T, P);
    while T_f ≠ ∅  do
        S = S ∪ T_f
        new_repair = Repair(P, S, rc)
        if new_repair == null then
            break
        end if
        P' = ApplyRepair(P, new_repair)
        T_f = ExtractFailedTests(T, P');
    end while
    if new_repair not null then
        return new_repair
    end if
end while

function Repair(P, S, rc)
    C = GenerateRepairConstraint(P, S, rc);
    level = 1  // The complexity of a repair
    new_repair = Synthesize(C, level);
    while new_repair == null and level ≤ MAX_LEVEL do
        level = level + 1
        new_repair = Synthesize(C, level);
    end while
    return new_repair
end function
```

# 5. 存在的问题

# 6. 改进的思路