

Symbolic Execution Notes

Jerry



1 Symbolic Execution

之前已经学习了把程序转成命题逻辑表达式的方法，类似地，可以把程序转成谓词逻辑表达式。例如下面这个函数。

```
void swap(int& a, int& b) {  
    a = a ^ b;  
    b = a ^ b;  
    a = a ^ b;  
}
```

证明这个函数交换了 a 和 b 的值，就是证明下面的谓词逻辑表达式永真。

$$\begin{aligned}
 &(\forall A0)(\forall A1)(\forall A2)(\forall A3)(\forall B0)(\forall B1)(\forall B2)(\forall B3)(\\
 &\quad (A1 = xor(A0, B0) \wedge \\
 &\quad B1 = B0 \wedge \\
 &\quad A2 = A1 \wedge \\
 &\quad B2 = xor(A1, B1) \\
 &\quad B3 = B2 \\
 &\quad A3 = xor(A2, B2) \\
 &\quad) \rightarrow \\
 &\quad ((A3 = B0) \wedge (B3 = A0)) \\
 &\quad)
 \end{aligned}$$

也就是证明下面的表达式不可满足。

$$\begin{aligned}
 &(\exists A0)(\exists A1)(\exists A2)(\exists A3)(\exists B0)(\exists B1)(\exists B2)(\exists B3)(\\
 &\quad (A1 = xor(A0, B0) \wedge \\
 &\quad B1 = B0 \wedge \\
 &\quad A2 = A1 \wedge \\
 &\quad B2 = xor(A1, B1) \\
 &\quad B3 = B2 \\
 &\quad A3 = xor(A2, B2) \\
 &\quad) \wedge \\
 &\quad \neg((A3 = B0) \wedge (B3 = A0)) \\
 &\quad)
 \end{aligned}$$

但是这种方法有一个问题，如果程序非常长或者有很多的 if 语句，那么就会生成一个很长的公式。

符号执行技术为程序的每一条可能的执行路径生成一个公式，分别求解。和上面的方法一样，符号执行也是用符号而不是具体的数值表示当前程序状态。符号执行过程中维护 3 块数据。

- 当前将要执行的语句
- Symbolic storage 用来记录每个变量的值，这些值实际就是符号组成的表达式
- Path constraint 记录到达当前语句的条件，因为每条执行路径会生成一个公式，Path constraints 就反映出之前遇到的每个 if 语句是 true 还是 false

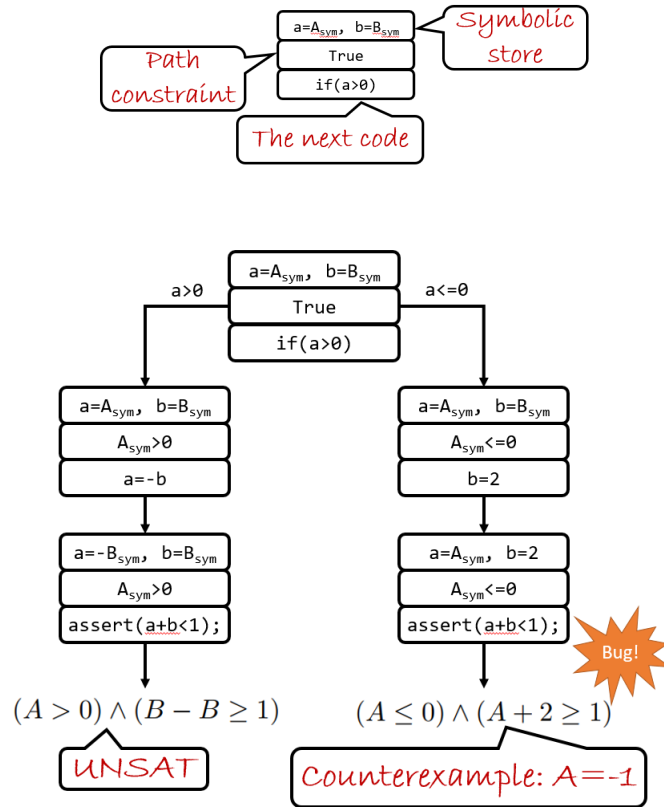
以下面这个程序为例讲解符号执行的流程。

```

void func(int a, int b) {
    if(a > 0)
        a = -b;
    else
        b = 2;
    assert(a + b < 1);
}

```

初始状态下，如下图所示，当前将要执行的语句是 $\text{if}(a>0)$ ，symbolic storage 记录 a 的值是符号 A ， b 的值是符号 B ，path constraint 是 true ，因为这条语句是一定要执行的。



整个符号执行的流程如上图所示。因为当前遇到了 if 语句，符号执行会分成两路，左边的子树对应 $a>0$ 成立的情况，右边的子树对应 $a>0$ 不成立的情况。现在考虑 $a>0$ 成立的情况，也就是左子树的根节点。将要执行的语句是 $a=-b$ ， if 语句不修改变量的值，所以 symbolic storage 还是 $\{a=A, b=B\}$ ，因为要求 if 语句成立才能执行到 $a=-b$ ，所以现在 path constraint 是 $A>0$ 。 $a=-b$ 执行后，接下来将执行 $\text{assert}(a+b<1)$ ，由于 $a=-b$ 修改了 a 的值，所以 symbolic storage 是 $\{a=-B, b=B\}$ ，path constraint 是 $A>0$ 。现在判断 assert 语句是否会报错，已知条件是 path constraint 成立，要证明当前 $a+b<1$ 永远成立，就是证明 $(A > 0) \rightarrow (B - B < 1)$ 永远成立，就是证明 $(A > 0) \wedge (B - B \geq 1)$ 不可满足，接下来就交给 SMT solver 了。右子树对应 $a \leq 0$ 的情况，流程同上。最后，如果每个执行路径生成的逻辑表达式都是不可满足的，那么说明 assert 不会报错，但是在这个例子中， $a \leq 0$ 对应的执行路径得到了可满足的表达式， $A=-1$ 让 $(A \leq 0) \wedge (A + 2 \geq 1)$ 为 T ，说明输入 a

为-1 时会报错。

假如出现多个 if 语句，例如下面这个函数

```
void func(int a, int b) {  
    if(a > 0) {  
        if(a < 5)  
            a = -b;  
        a = 3;  
    }  
    else  
        b = 2;  
    assert(a + b < 1);  
}
```

执行到 `if(a < 5)` 的 path constraint 是 $A > 0$ ，执行到 `a=-b` 时的 path constraint 就是 $(A > 0) \wedge (A < 5)$ ，也就是两个 if 条件的与。