# 符号测试

## 1.基本概念

**符号执行** （Symbolic Execution）是一种程序分析技术，它可以通过分析程序来得到让特定代码区域执行的输入。顾名思义，使用符号执行分析一个程序时，该程序会**使用符号值作为输入，而非一般执行程序时使用的具体值。**在达到目标代码时，分析器可以得到相应的**路径约束**，然后通过约束求解器来得到可以触发目标代码的具体值。

**软件测试中的符号执行**主要目标是：

在给定的探索尽可能多的、不同的程序路径(program path)。对于每一条程序路径，
1) 生成一个具体输入的集合(主要能力)；
2) 检查是否存在各种错误(errors，包含assertion violations, uncaught exceptions, security vulnerabilities, and memory corruption)。

从测试生成的角度，符号执行可以生成高覆盖率的测试用例。
从bug finding的角度，符号执行可以提供一个具体的输入用于触发bug（过去常常用于调试bug）。

## 2.主要技术

符号执行的主要思想就是将输入(input)用符号来表征而不是具体值，同时将程序变量表征成符号表达式。因此，程序的输出就会被表征成一个程序输入的函数，即fun(input)。在软件测试中，符号执行被用于生成执行路径(execution path)的输入 。

**执行路径(execution path)**：一个true和false的序列seq={p0,p1,...,pn}。其中，如果是一个条件语句，那么pi=ture则表示这条条件语句取true，否则取false。
**执行树(execution tree)**：一个程序的所有执行路径则可表征成一棵执行树。

现代符号执行技术的特点是同时执行精确(Concrete)执行和符号(Symbolic)执行。

## 3.重要挑战和解决方案

### 3-1. 路径爆炸(Path Explosion)

**描述**:
首先，要知道，符号执行implicitly过滤两种路径：

1. 不依赖于符号输入的路径；
2. 对于当前的路径约束，不可解的路径。
   但是，尽管符号执行已经做了这些过滤，路径爆炸依旧是符号执行的最大挑战。

**解决方案**:

1. 利用启发式搜索搜索最佳路径

   目前，主要的启发式搜索主要focus在对语句和分支达到

   高覆盖率

   ，同时他们也可被用于优化理想的准则。

   - 方法1：利用控制流图来guideexporation。
   - 方法2：interleave 符号执行和随机测试。
   - 方法3（more recently）：符号执行结合演化搜索(evolutionary search)。其中，fitness function用于drive the exploration of the input space。

2. 利用可靠的(sound)程序分析技术来减小路径爆炸的复杂度
    - 方法1：静态地合并路径，然后再feed solver。尽管这个方法在很多场合都有效，但是他把复杂度转移给了solver，从而导致了下一个challenge，即约束求解的复杂度。
    - 方法2： 在后续的计算中，记录并重用low-level function的分析结果。
    - 方法3： 自动化剪枝

## 3-2. 约束求解(Constraint Solving)

**描述**:
约束求解是符号执行的技术瓶颈。因此，对于solver的优化（提高solver的求解能力）成了解决这个技术瓶颈的手段。

**解决方案**:

1. **去除不相关的约束**
   一般来说，程序分支主要依赖于一小部分的程序变量。也就是说，程序分支依赖于一小部分来自于路径条件(path condition)的约束。因此，一种有效的方法就是去掉那些与当前分支的输出不相关的路径条件。例如，现有路径条件：`(x+y>10)^(z>0)^(y<12)^(z-x=0)`。假设我们现在想生成满足`(x+y>10)^(z>0)^¬(y<12)`，其中我们想建立对`¬(y<12)`（与y有关）的feasibility。那么，`(z>0)`和`(z-x=0)`这两个约束都可以去掉，因为与y不相关。
2. **递增求解**
   核心思想就是缓存已经求解过的约束，例如`(x+y<10)^(x>5)=>{x=6,y=3}`。对于新的约束，首先判断这个新约束的搜索空间是缓存里约束的超集还是子集。如果是新的约束的搜索空间是缓存的约束的子集，那么，就把缓存中的约束去掉多余的条件后继续求解。如果是超集，那么直接把解代入去验证。

## 3-3. 内存建模(Memory Modeling)

**描述**:
程序语句转化成符号约束的精度对符号执行的覆盖率有很大的影响。例如，内存建模是用一个具体的数值去近似一个固定位数的整数变量可能会很有效，但是另一方面，它也会导致imprecision，比如丢失一些符号执行的路径，或探索一些无用的解。另一个例子就是指针，一些工具如DART[3]只能处理精确的指针。

**解决方案**:
precision和scalability是一个trade-off。需要考虑的因素有：

1. 代码是high level的application code还是low-level的system code。
2. 不同的约束求解器的实际效果。

## 3-4. 并发控制(Handling Concurrency)

**描述**:
很多现实世界中的程序是并发的，这也意味着他们很多都是不确定的(non-deteminism)。尽管如此，符号执行已经被有效地运用在测试并发系统，分布式系统。