**Report on KLEE**

KLEE: Unassisted and Automatic Generation of High-Coverage

Tests for Complex Systems Programs

本文主要描述了一种符号执行的工具KLEE。用以解决/弥补之前的算法（如：EXE）在覆盖率和外部环境影响分析上的缺陷。

**本文的主要创新点如下：**

（1）在EXE的基础上设计了KLEE工具。

1. KLEE采用了多种约束条件解决优化算法，有较大的代码覆盖范围；

2. KLEE采用了一个简单直接的处理外部环境影响的方法，使得KLEE性能提高了一个数量级。

（2）在大量的测试程序集GNU COREUTILS上运行KLEE，验证了其性能。

（3）使用KLEE在GNU COREUTILS上发现了很多bug。

1. 包括3个自95年至今都没发现的bug。

**本文的主要解决思路如下：**

**1. KLEE产生的背景：符号执行**

首先本文介绍了什么是符号执行（Symbol Execution），即，在程序运行之前，使用符号来代替变量值，经由不同的分支及其分支约束条件，对于遇到的bug产生一组有效的测试输入。在KLEE以前，符号执行的问题是覆盖率较低，也即，无法广泛的适用于不同类型的程序。这个问题的主要挑战在于：

（1）代码会产生指数级的路径（path）数量；

（2）代码会收到了环境的影响，如：操作系统，网络等，对于输入很敏感。

**2. KLEE是什么**

文中提出的解决方案是KLEE，其主要目标是执行到了每一个执行代码（汇编），并检测每一个危险操作（dangerous operation），如assert。与一般方案不同的是，KLEE直接给出出现错误的条件，而非具体的输入参数。在程序运行结束时，KLEE才会根据错误条件，给出出现错误的具体测试输入参数（test case）。

运行KLEE时，需要首先将原始的代码编译到汇编代码（使用LLVM工具-GNU C）。然后对于汇编代码的语句进行检查。其基本思路是：

（1）初始的输入为“无约束条件”的输入；

（2）每当遇到分支跳转（branch）时，根据当前的数据约束条件判断跳转分支的有效性。

（3）对于有效的分支，在当前的约束条件中加入分支条件，形成路径条件（path condition）。

（4）重复（2）（3）直到遇到程序终止。执行解路径条件（solve path condition），生成测试输入。

因此在KLEE中，程序错误的粒度会被定义在每一条汇编执行指令上。

**3. KLEE的执行细节：**

KLEE的基本架构如下所示。程序中的所有变量都会由输入数据的符号进行表示。在运行的过程中，KLEE将维护一个state树结构。其中，树的中间节点是由程序的中间语句产生的；叶子节点则是每一条路径完成时的路径约束条件。

对于大部分语句，如赋值，计算，KLEE直接完成变量的符号表示。

对于分支语句，如前文所述，先判断分支的有效性，然后fork出多个有效分支，并维护每一个分支的路径条件。

对于潜在危险指令，如除法。首先KLEE会判断是否会出错。无论出错与否，程序都会在给出正确约束条件的基础上继续执行代码。换句话说，程序的错误判断和分支在这一部分是完全分开，互不影响的。这样，一方面可以找到完整的错误（不被前一个错误屏蔽）；另一方面程序可以单独处理state分支和错误判定，程序结构简单。

对于其他的危险指令，如内存访问。则会在解约束程序STP中首先对程序进行存储地址映射。

KLEE的符号表示过程由于会产生分支，所以会导致较多的冗余。因此KLEE会采用COW（copy on write）技术——需要修改时，才真正的完成复制，否则只是用虚拟的映射。

KLEE的解约束算法优化包括几个方面。表达公式的简化，约束条件的简化，潜在数值的具体化（const），约束条件独立，反例的缓存（用于建立减少冗余的查询）。

还有一个重要的部分就是state的调度。即在branch或其他会产生多个子state的时候，需要判断先执行哪一个。文中给出了两个策略：

（1）随机路径选择（Random Path Selection）

随机选择一个路径进行执行。算法简单，代价低，同时会避免死循环，或者有路径不被执行。缺点是空间复杂度比较高，会产生比较多的路径数据复制。

（2）优化覆盖率选择（Coverage-Optimized Search）

对于不同的路径增加一个权重，凡是路径层次越高的结点权值高，反之则低。这样，高权重的结点的约束条件比较少，有更大的可能存在更多的子分支，从而遍历更多的情况。

问题：这里没有说清楚的是，如果算法执行的时间是充裕的，那就是全部的path都被执行，那么这种情况下，就会是全部的路径都被遍历。上述两个算法就没有什么意义了。

**4. 环境建模**

这一部分是创新的部分。环境主要指的是各种环境的参数，包括命令行的参数，环境变量，文件数据，网络数据包等等。换句话说，就是输入参数和配置信息。

（1）文件系统

文件系统操作，包括真实的文件和符号文件。对于前者，调用相关的系统调用；对于后者，则需要对于每一个符号文件系统仿真操作的影响。

（2）失败的系统调用

KLEE可以完成对系统调用失败情况的仿真。

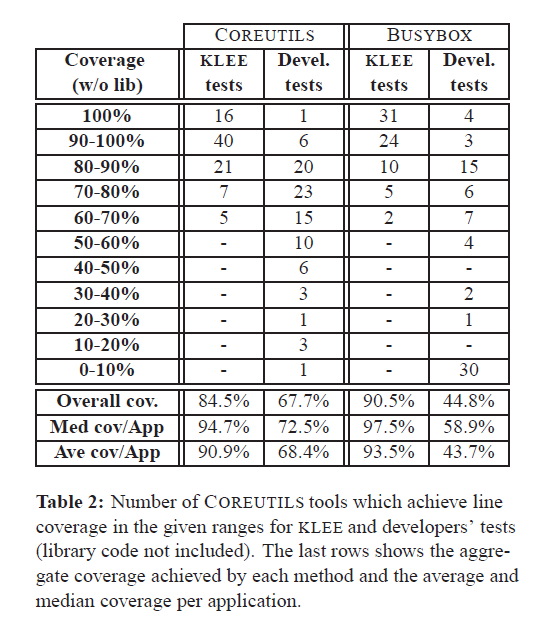
（3）**Rerunning test cases**

这部分没看懂。

**本文的实验验证如下：**

**1. 覆盖率实验**

目标程序：COREUTILS和BUSYBOX。



上图是对KLEE和Developers’ test程序之间的对比图。KLEE的覆盖率在不同的测试程序上都远远优于Developers’ test。

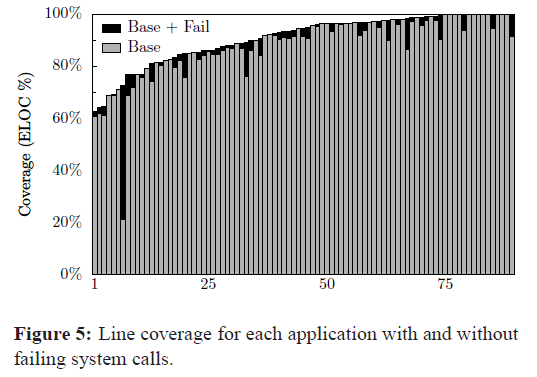
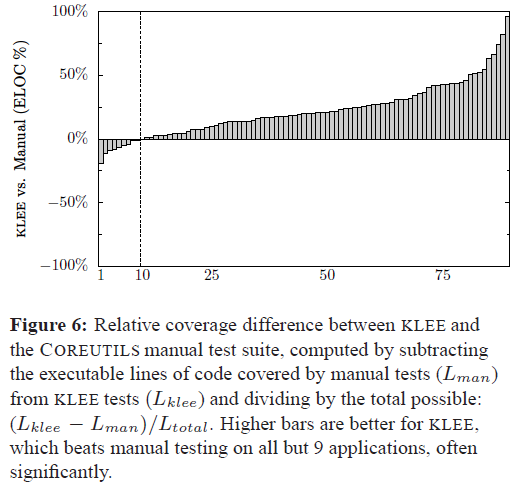
 

图5展示的是环境因素的影响，即系统调用失败时的影响，其中黑色的带是增加了对系统调用失败处理的代码的提升。

图6 是覆盖率对于KLEE和Manual test的比较，数值是KLEE的覆盖率减去后者的覆盖率的百分比。可见，对于大部分的testbench，KLEE都比较优越。