**Report on S2E**

S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems

本文主要描述了一个面向于大规模程序的软件系统分析平台S2E。S2E是基于符号执行算法的一个工具。可以降低现有算法在处理大型程序是的复杂度，并根据需求在结果的精确度和运行性能之间找到折中。

**本文的主要创新点如下：**

（1）在符号执行的基础上设计了S2E平台。

1. 有选择的符号执行算法

2. 设计了执行一致性模型——根据不同的模型有不同的符号执行选择算法

3. 设计了相应于S2E的API

（2）完成了平台的实现，并对其进行运行测试。

**本文的主要解决思路如下：**

**1. S2E面向的问题：大型程序的执行**

（1）现有的算法处理软件分析时，是对于所有可能的路径进行分析。对于大型的程序，其复杂度是指数的。因此需要对算法进行简化。

（2）软件分析时，可以将软件系统分为两个部分，程序和环境条件。前者就是待分析的代码，后者是代码使用的库函数，文件系统，操作系统调用等。现有算法在处理环境条件的思路是对环境条件进行建模。如果环境比较复杂，如window操作系统，那么建模的复杂度是很大的。

（3）目前现有的程序来源比较复杂，大部分都是闭源或者半开源的。没法获得实际的源代码。所以现有的分析工具都无法实现。

**2. S2E如何解决这些问题？**

（1）面对算法复杂度的问题，S2E采用的办法就是剪枝。去除冗余的或者影响较小的路径来降低计算复杂度。因此，S2E实现了可选择的符号执行（Selective Symbolic Execution，S2E）。和现有的符号执行方法相比，这种算法根据实际需求精简符号执行的路径，将在环境中运行的部分大大的缩减。

（2）面对环境建模的问题，S2E给出的方法不是建模，而是选择实际参数直接执行。 一方面，直接执行可以跳过建模的过程；另一方面，直接执行可以避免建模错误带来的误差。直接执行的缺点是不能遍历所有的情况，使得最终的结果不能完全的覆盖全部的变量域。为此，S2E给出的解决方案是针对不同的需求设计了6种固定的执行模型，分别给出了不同选择真实参数的策略，最大限度的降低直接执行的片面性。

（3）S2E给出了可以在二进制代码上直接运行的API。

**3. S2E的执行细节：**

S2E把面向的代码系统包括处理单元和环境条件两个部分：系统=处理单元+环境条件。处理单元是需要被分析的代码段，环境条件是其他的代码段。

一个使用S2E处理的例子如下所示。程序分为三个部分，app层调用lib程序，lib层调用内核的程序。当kernel的程序返回值后，lib经过一系列处理再返回app层。在这个过程中，lib层是处理单元，app层和kernel层都是环境条件。

**（1）可选择的符号执行**

如前文所述，可选择的符号执行就是对于环境条件中产生的分支路径进行剪枝。这里的剪枝并不是删掉应用的分支，而是将环境条件中的分支打包起来。具体的打包方法就是使用使用一组真实数据来运行这段代码。而这组真实的数据实在环境条件被调用时的路径约束所约束的。S2E处理真实数据的数据执行模式和符号执行模式之间的切换主要分为四个过程。

例如，在图中例子里，app代码是环境条件，需要进入数据执行的模式。其之前没有路径约束（app层是路径的起点），那么对于一个变量x，可以选择一个合理的x=5来执行app层。

**阶段一：环境条件调用处理单元**

当app层调用lib层时，lib层是处理单元，因此需要将变量符号表示，进入符号执行的模式。此时，x需要从真实数据5变为符号a。其中a的约束条件由当前保存的路径约束条件得到——例子中是运行的开始，因此条件应为全体实数。由于x在lib中完成了符号执行。因此，路径在lib的代码段中产生了分叉。

需要说明的是，与此同时根据x=5的真实数据也会并行的写入lib层代码中，进行真实数据执行。下面的阶段中，这一个真实数据的路径会始终保存，且唯一。

**阶段二：处理单元调用环境条件**

当lib代码中开始调用kernel层时，变量x再一次进入了环境条件中。此时需要根据x所处于的路径约束构建一个x的值。因此，在环境条件kernel层中，用于测试的是数据执行模式。

**阶段三：环境条件返回处理单元**

Kernel层完成数据处理，将数据返回lib层。代码从环境条件返回到处理单元中，测试模式需要从数据执行模式跳转回符号执行模式。此时需要根据x的值构建一个x的约束条件。再将新的变量约束条件加入lib层的符号执行中。

**阶段四：处理单元返回环境条件**

处理单元处理好数据，将数据返回给环境条件。此时的路径约束条件会保存在环境条件处。而环境条件中所使用的真实数据来源于最早的x=5数据执行到此处的结果。实际上，这组结果一定是满足对应的约束条件的。这组约束条件会在下一个阶段一出现时用于构建变量所需的约束条件。

以上的四个阶段就是S2E进行剪枝（打包）工作所面临的四个阶段。总结的来说，就是在环境中用真实数据执行，在单元中用符号执行产生新的路径。因此路径只会在处理单元中产生。大大降低了计算的复杂度。

以上的四个阶段中，其中两个阶段是真实数据执行模式🡪符号执行模式，两个阶段是符号执行模式🡪真实数据执行模式。在前者中，真实数据需要根据当前的结果产生一组合理的约束条件，例如x=5。但是这样的约束条件过紧，会导致“过约束”。此外，如果真实数据转化为符号执行时产生新变量的话，需要给出合理的新变量的约束条件，而这是很难的。（在此前的算法中，由于环境可以确定性的建模，因此新变量均可以由初始变量表示。但是在S2E中，环境产生的变量可能是新变量）

为了应对“过约束”的问题，或者说是为了应对运行精确度和运行性能的折中问题，S2E给出了6中不同强度的一致性模型，来确定如何定界。

**（2）执行一致性模型**

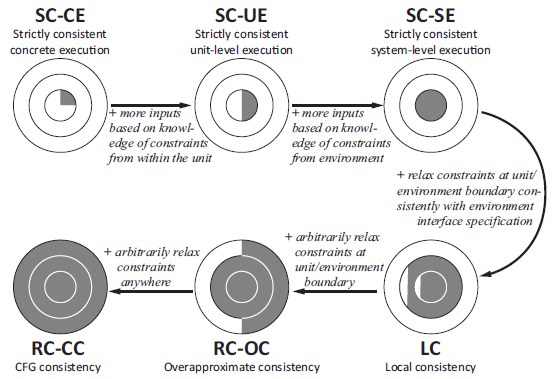
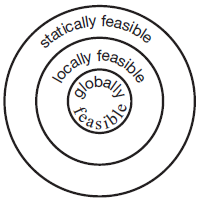
对于程序执行路径，有三种不同层次：

1. 静态可执行路径（statically feasible）：满足系统控制流图

2. 局部可执行路径（locally feasible）：以上且满足处理单元变量约束的流图

3. 全局可执行路径（global feasible）：以上且满足环境约束的流图

根据不同的需求关系，得到6个不同强度的一致性模型，如下图：



（1）SC-CE：

只接受全局可行路径。不分析系统。

不会产生新的分支。只能通过随机产生输入数据产生不同路径

（2）SC-UE：（KLEE）

只接受全局可行路径。只分析处理单元，不分析环境。

处理单元中产生新的分支，环境中不追踪路径约束。

（3）SC-SE：

只接受全局可行路径。分析处理单元和环境。

会得到全部分支，但存在计算复杂度问题。

（3）LC：

只接受局部可行路径。分析处理单元。只分析环境输出的取值范围。

由于不考虑输入数据在环境中的影响，会导致某些路径不满足环境条件的CFG

（5）RC-OC：

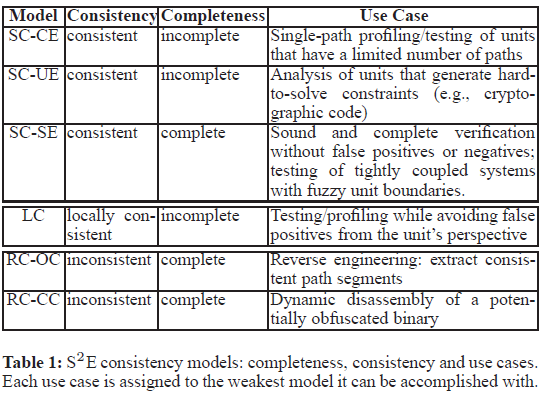
接受所有可行路径。忽视环境给出的约束条件。

可以适用于逆向工程。

（6）RC-CC：

接受所有可行路径。忽视环境给出的约束条件。

可以适用于反汇编或加密代码。



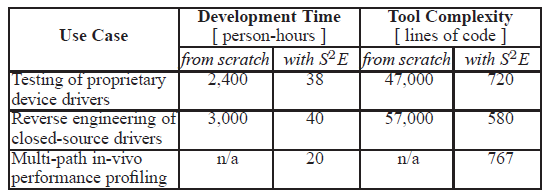
**（3）给出了S2E的原型系统**

如下图。自上而下是程序运行的层次，分别代表着不同的区域，每一个区域都可以能设定为环境或处理单元。其中符号执行的处理核心采用的是KLEE的基本方式。

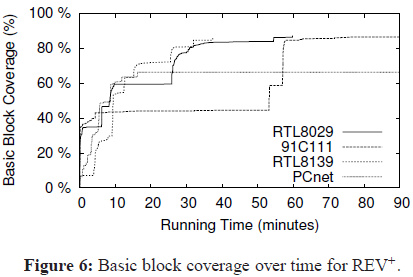
**本文的实验验证如下：**

**1. 基于S2E的测试器**

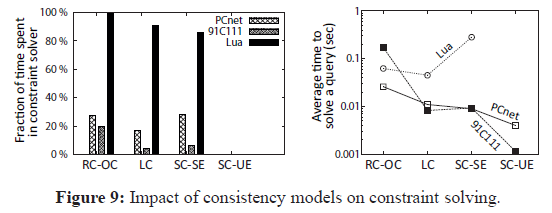
主要有三个测试器，分别是专用硬件驱动程序测试器，反向工程模块和性能分析模块



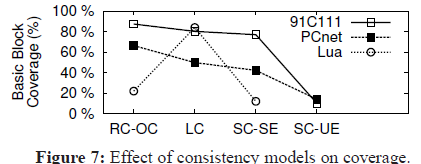
上图是三种不同模块的生成和工具复杂度。

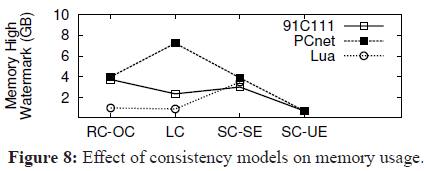


上图是使用反向工具时，随时间增长，覆盖率的变化，时间变成以后，覆盖率趋于饱和。



上图是不同的模型处理常数分析的时间，越复杂的模型的处理时间越长。





上图是不同的模型处理时，覆盖率和内存空间利用率的曲线。