15-745 Optimizing Compilers
Project Proposal

Symbolic Execution in Difficult Environments

David Renshaw
renshaw@cmu.edu

Soonho Kong
soonhok@cs.cmu.edu

March 18, 2011

This project will be hosted at `www.cs.cmu.edu/~dwrensha/15-745/`.

1 Introduction

Symoblic execution is a static analysis technique that allows programs to be run on input that is not fully specified. Program data is kept *abstract* as long as possible. At branch points, all possible paths are taken, and the corresponding contraints on the abstract data are propagated. Symbolic execution has been used for generating high-coverage test cases [2], for performance profiling and reverse engineering [5], and for verification of remote client behavior in online games [1].

KLEE [2] is a symbolic execution engine built on top of LLVM. The primary goal of our project will be to add new features to KLEE.

2 Description

Goal Our tentative goal is to improve KLEE to overcome the following limitations of the way that it models file systems.

1. KLEE does not handle file operations on a concrete file symbolically (§4.1 of [2]). If a target program does an operation on an actually existing file (called concrete file), KLEE just executes that operation. This limits the possible test coverage of generated test cases.

   For example, consider a program which uses "`config.txt`" file as a configuration file and do operations based on the content of this file.

```
...
if(read("config.txt") == "A") {
      doA();
} else {
      doB();
}
...
```

Because KLEE treats "`config.txt`" as a concrete file, KLEE generates test cases only covering either doA() part or doB() part, not the both of them.

2. As indicated in §4.1 of [2], KLEE's symbolic file sysem is *crude*, containing a single directory with N symbolic files in it. We will extend this to be a richer symbolic file system which supports multiple directories and nested directories.

Metric We are going to run our version of KLEE and compare the result with original KLEE's result. The same `coreutils` will be used as a benchmark. We expect our version generates test cases which are better than KLEE in terms of *line coverage* with not severly degenerating performance (time).

Specific Goals

- 75%: Only achieve item 1.
- 100%: Achieve item 1 & 2.
- 125%: Have more test benchmarks and show that our extended KLEE performs better than the original.

3 Logistics

3.1 Plan of Attack

	Week	David Renshaw	Soonho Kong
1	(3/18 - 3/22)	Figure out KLEE internals, settle on goals	
2	(3/23 - 3/29)	Write simple program analyses	
3	(3/30 - 4/5)	Design analysis	
4	(4/6 - 4/12)	Implement analysis	
5	(4/13 - 4/19)	find / create interesting code for analysis	tune analysis
6	(4/20 - 4/26)	Write documentation	Perform Experiments

3.2 Milestone

For the milestone, we hope to have a working analysis so that in the remaining two weeks we can concentrate on tuning it and achieving interesting experimental results.

3.3 Literature Search

Basically, KLEE paper [2] is the start point. Tkachuk and et al. [4] try to address similar problem for compositional model checking of Java programs. Kong et al. [3] present a parameterized model of the file system that can be used in conjunction with Pex, an automated test generation tool for .NET framework.

3.4 Resources Needed

We have all the resources needed, including LLVM suite, KLEE source code, and `coreutils` benchmark.

3.5 Getting Started

We have installed KLEE on our machines and followed tutorials.

References

[1] Darrell Bethea, Robert A. Cochran, and Michael K. Reiter. Server-side verification of client behavior in online games. In *Network and Distributed System Security Symposium (NDSS)*, 2010.

[2] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2008.

[3] Soonho Kong, Nikolai Tillmann, and Jonathan de Halleux. Automated testing of environment-dependent programs - a case study of modeling the file system for pex. *Information Technology: New Generations, Third International Conference on*, 0:758–762, 2009.

[4] O. Tkachuk, M.B. Dwyer, and C.S. Pasareanu. Automated environment generation for software model checking. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 116 – 127, 2003.

[5] Volodymyr Kuznetsov Vitaly Chipounov and George Candea. S2e: A platform for in-vivo multi-path analysis of software systems. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.