Project in Program Analysis 236???

Nimrod Partush and Eran Yahav

Winter 2012-13

1 General Information

1.1 Project Spirit

In this project you will implement and experiment with research results in the area of program analysis. The general idea is to implement some variant of a well known program analysis technique or use an existing tool towards some special purpose. You will usually start with a research paper, describing some technique or tool, and use it to familiarize with the problem at hand. This knowledge will later be used to implement the techniques presented in the paper or use the knowledge of the existing technique towards a certain purpose (for papers that describe tools).

Simplifying Assumptions The projects listed below provide a lot of freedom and you are encouraged to make intelligent choices of simplifying assumptions. Our only requirements are that your simplifying assumptions will be justifiable, consistent, and documented. Example: assuming that there is no need to deal with pointer arithmetic may be justifiable if you and that your technique is useful in a reasonable class of benchmarks despite not handling this feature.

Getting Started The input to your program analysis is going to be (not surprisingly) a program. All of our projects rely on existing frameworks for the fairly standard problem of parsing a program and transforming it to some intermediate representation that is designed for analysis and transformation. Therefore, the first step in your project should be to create a few example programs and make sure that you can run the basic framework on them to produce the intermediate representation.

Working on the Project

- For your own sake, you should use a revision control system. We can setup an SVN repository for your project per your request.
- To prevent environment related issues, and allow us to use your results more easily, you are required to do all your work within a VM image and

submit it as part of your final submission. In case you are unable to create a Vmware image, An initial image will be provided for you.

Project Presentation When the project is completed, you will be required to present it in a short evaluation meeting. While you will be running the project for the demonstration, we should be able to run your project independently, so please provide the appropriate documentation for doing so as part of your submission.

Time Evaluation You will evaluate the amount of time spent on the project yourselves (the honor system) by documenting how much time is spent on each task.

1.2 Project Schedule

The tentative schedule for the project is:

- Week 2: submit an outline of your planned project
- Week 7: submit a progress report
- Week 14: submit a final report and present your project

This tentative schedule is flexible. In particular, the final deadline for presenting your project may be extended until after the exam period.

2 Available Topics

Project 1: Dataflow Analysis of Intermediate Representation Language

Goal Given a LLVM bitcode program, statically (at compile time) produce the set of values and constraint for each integer variable, at each program point.

Main Idea The main idea is to statically track the possible values and constraints of integer variables in the program. Since integer variables may (conceptually) take unbounded values, and the number of program states may (conceptually) be unbounded, you will employ abstraction to represent the potentially unbounded program state space in a bounded way. As the collected data will refer to the variables in the intermediate representation, you will use debug information to translate code locations in the IR back to the source code.

Scope 300 hours.

Output In-code printing of abstract state, in *source code level*.

Evaluation Criteria Your project will be evaluated according to the following criteria:

- **Documentation and Code Style!**: Your project is part of active research and we aim to use your results (assuming they are good), therefore your code *must be clean, structured, documented and as simple as possible (without hurting performance)*.
- **Benchmark**: The set of programs you use for collecting the histogram should be versatile. You are encouraged to consider all types of structures in a programs, including loops and recursion (of course coverage for these types of programs can never be complete).
- **Completeness**: You analysis should handle as many language features as possible. For example: arithmetic shifting, modulo operation, etc.
- **Soundness**: Your analysis should always report an over-approximation of the possible values. False positives are permitted, false negatives are not.
- **Precision**: The less false positives - the better.
- **Performance**: The faster your analysis - the better.
- **Presentation**: The analysis results should be replayed in a readable manner. You are encouraged to use CLang's in-code diagnostics mechanism for this.

Don't be discouraged by these criteria, we are well aware of how challenging it is to implement a successful analysis and present these criteria just to describe how a good analysis is measured.

Infrastructure The infrastructure that you will use for this project is the LLVM low-level virtual machine and the APRON numerical abstract domain library. You will implement the analysis as an LLVM pass and keep variable information and constraints in an abstract state provided by the APRON library.

References

- Eran's course slides.
- Writing a LLVM pass.
- APRON's C++ API Test file (the best way to learn how to use APRON).
- CLang's diagnostics.

Project 2: Differential Path Profiling

Goal Given two versions of a C program (P,P'), provide a histogram describing path executions in the two versions, classified by input.

Main Idea We want to describe how the execution of a program changes due to a patch. You will first use a standard algorithm to uniquely tag all the paths in the two versions (this algorithm is guaranteed to tag re-occuring paths in P and P' with the same ID). Once the tagging is complete you will dynamically profile the programs, by running them on different inputs, and produce a histogram describing on which inputs the two programs differ in path and on which they are equivalent. One of the challenges here is producing a test set that will cover as many paths as possible.

Scope 150 hours.

Output

1. A table showing the overall paths run, describing the path index, inputs that invoked the path, and path outputs from each program. The table will be divided into three sections: (i) joint paths (ii) added paths (iii) removed paths.
2. A histogram of the collected data in the table.

Evaluation Criteria Your project will be evaluated according to the following criteria:

- **Documentation and Code Style!**: Your project is part of active research and we aim to use your results (assuming they are good), therefore your code *must be clean, structured, documented and as simple as possible (without hurting performance).*
- **Correctness**: Your profiling must be consistent in the way it tags paths and must correlate and differentiate paths correctly.
- **Coverage**: You should try and cover as many paths as possible.
- **Benchmark**: The set of programs you use for collecting the histogram should be versatile. You are encouraged to consider all types of structures in a programs, including loops and recursion (of course coverage for these types of programs can never be complete).
- **Performance**: The faster your analysis - the better.

Don't be discouraged by these criteria, we are well aware of how challenging it is to implement a successful analysis and present these criteria just to describe how a good analysis is measured.

Infrastructure The infrastructure that you will use for this project is the LLVM low-level virtual machine. You will implement the analysis as an LLVM pass and may build upon existing path profiling passes (preliminary work for this has been done, consult the staff as to how it can be used). Alternatively you may create an entirely new pass.

References

- A paper about path profiling and tagging
- Writing a LLVM pass.
- Existing Path Profiling in the Low-Level Virtual-Machine.
- KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
- Using KLEE.

Project 3: Differential Binary Analysis for Exploit Generation

Goal Given two versions of a binary program, use dynamic analysis methods to produce inputs that exploit security vulnerabilities in either of the programs.

Main Idea As many patches made to security critical systems (operating systems, etc.) are given in binary form (in an attempt to thwart attackers from locating the vulnerability patched), we are interested in analyzing program binaries to produce input that leads to different behaviors in these binaries. We will use dynamic analysis methods to symbolically execute the two versions and lead the execution towards different behaviors. To encapsulate the two versions into one program, we will use a *correlating program* construct that combines two versions of a program into one binary. This binary will be dynamically analyzed using the S^2E selective symbolic execution engine to direct the execution towards paths where output variables differ (provided by assertions in the correlating programs that check for variable equality).

Scope 300 hours.

Output Concrete inputs that generate different output in each program.

Evaluation Criteria Your project will be evaluated according to the following criteria:

- **Precision**: The closer you get to covering the set of paths which the program differ in output - the better (a good benchmark is required to correctly evaluate this).
- **Performance**: The faster your analysis - the better.
- **Benchmark**: The set of programs you use for collecting the histogram should be versatile. You are encouraged to consider all types of structures in a programs, including loops and recursion (of course coverage for these types of programs can never be complete).
- **Documentation and Code Style!**: Your project is part of active research and we aim to use your results (assuming they are good), therefore your code *must be clean, structured, documented and as simple as possible (without hurting performance).*

Don't be discouraged by these criteria, we are well aware of how challenging it is to implement a successful analysis and present these criteria just to describe the dimensions of the problem.

Infrastructure The infrastructure that you will use for this project is the S2E tool and the correlating compiler (provided by the course staff).

References

- S2E Paper. Read First!
- S2E Site.

- (Hard Copy) Abstract Semantic Differencing for Numerical Programs (describes the correlating program).

Project 4: Roll your own

The project list above is by no means exhaustive. If you have an idea for a project in the area of program analysis that you would like to implement, let us know.