Artificial Intelligence for Software Testing and Reverse Engineering Final Lab Assignment

INTRODUCTION

In this course, you have learned the state-of-the-art methods for automated software testing and reverse engineering using artificial intelligence. In the final assignment, you have to show that you understand these methods, know how to implement them, and can draw valid conclusions from their use. In addition, you have to show that you understand and can use research tools from testing and reverse engineering.

Model/machine learning is a key technology for reverse engineering. The basic task in machine learning is to uncover a model from data that explains the observed behavior, which is indeed very similar to reverse engineering. The key difference is the presence of data, but we have an executable that can be used to generate it! This is what active state machine learning does. Moreover, it uses the current hypothesized model to minimize the amount of required data.

Genetic programming is the key technique used for automated program testing. The key reason for its success is the recombination scheme it uses to create new programs, while not requiring symbolic reasoning on the outcome of a program. This is all done concretely, i.e., by running the software. Difficulties are coming up with a good fitness function that determines the value of a generated input program. The enormous search space (set of all possible programs) also does not help of course. Currently, there are no techniques that perform such program generation symbolically.

LEARNING OUTCOMES

In this Lab, you will explain the different algorithms you implemented in labs 1, 2, 3 and 4, and study more advanced topics in testing and reverse engineering. You must show that you understand how the implemented algorithms work, how to use them, and understand their strengths and weaknesses. After completing this assignment, you will be able to:

- Develop algorithms for automated reversing and testing.
- Perform research using state-of-the-art reversing and testing tools.

THIS LAB WILL BE GRADED (RUBRIC IN THE BACK OF LAB DESCRIPTION) AND WILL BE YOUR TOTAL GRADE FOR THE COURSE. THE NUMBER OF POINTS AVAILABLE ARE LISTED AT EVERY QUESTION.

INSTRUCTIONS

Task 1: Recap Labs 1, 2, 3, 4 (4 A4 - 40 points)

In the first task, you are asked to demonstrate what you learned during labs 1, 2, 3, and 4. In **max 4 A4 pages** (each worth 10 points), please provide the following:

- A description of the used AI (hill-climbing, satisfiability solving, genetic programming, active learning) technique, explain how the algorithm aims to triggering all code branches/behaviors. (approx. 1/2 A4)
- A run of your solution run on one of the RERS problems, provide snippets of output, and explain the results it gives, highlight the effect of choices you made in the design. You may use your improved solutions if you have made any improvements. (approx. 1/2 A4)

Task 2: AFL (1 A4 - 10 Points)

For this task, you are asked to apply AFL to the same set of reachability problems that were given in Task 1. To keep it fair, also run AFL for five minutes for each reachability problem. AFL is a state-of-the-art fuzzer that uses genetic algorithms and branch coverage for guidance. Instructions on how to set up AFL for the RERS problems are available on GitHub:

- https://github.com/apanichella/JavaInstrumentation/blob/main/docs/fuzzing_rers.md

Compare the performance of AFL against both the random fuzzer and your smart fuzzer. The findings/crashes directory contains all crashes AFL found, in this case there are the reachability errors. By running these through the normal (uninstrumented) code, you can obtain all the found reachability statements. To make it easier for you, we have written an script that collects the error codes that AFL managed to trigger for a given reachability problem. The script can be found here:

- https://github.com/apanichella/JavaInstrumentation/blob/main/docs/fuzzing_rers.md (Specifically "Retrieving Results" section)

In your comparison answer the following questions:

- Does AFL reach more unique reachability statements (error codes) than your own smart fuzzer on the same set of problems and time contraint? What about the number of unique branches? (approx. 1/2 A4)
- Investigate the traces that were used by AFL to find the error codes. Did your own fuzzer also generate the same traces? (approx. 1/2 A4)

For the comparison, you can again consider to plot the convergence graph. Write down all your findings!

Task 3: KLEE vs AFL (1 A4 - 10 Points)

KLEE is a state-of-the-art symbolic execution engine based on LLVM. We have already installed KLEE for you on the docker container.

For instructions on how to get KLEE working on the RERS problems, please have a look at this document in the repository:

- https://github.com/apanichella/JavaInstrumentation/blob/main/docs/symbolic_rers.md

Your task is to compare the performance of KLEE and AFL by running them on the reachability problems used in the first assignment (11-15 and 17). Give them the same amount of runtime and analyze the obtained reachability targets. Your focus is to highlight and explain the differences between fuzzing and concolic execution. Address the following elements in your report:

1. Descriptions of AFL and KLEE. How do they work? What are they good at? (approx. 1/2 A4)
2. What are the results that you got for each state-of-the-art tool on problems 11-15 and 17?
3. An analysis of the results. (approx. 1/2 A4)

The Driller paper (see Resources) contains good examples of result presentations. Use these for inspiration, although some will be too detailed to include in your report, be selective!

Task 4: Genetic Programming using ASTOR (1 A4 - 10 Points)

You are asked to use the tool ASTOR (https://github.com/SpoonLabs/astor) to find a patch for buggy versions of the RERS problems ("Buggy_RERS_ASTOR.zip"). ASTOR is already available in the docker container we provided for this course. Detailed instructions on how to run ASTOR on the three RERS problems above are also posted on Brightspace.

Step 1: First, you have to generate test cases for the original version (without bugs) of the three RERS problems above. You can use the test cases we provide. These are included in the packages on Brightspace, generated using EvoSuite.

Step 2: Your job is to run ASTOR to find a patch to the target bug. For the assignment, you need to run ASTOR using Genetic Programming as patch engine. You will set this engine using the parameter -mode jgenprog. More details information on how to run ASTOR are available on Brightspace and on GitHub.

Step 3: For each buggy version of the RERS problem, run ASTOR once. Then, report on statistics regarding:

- The time it needed to find the patches (if any).
- How many patches were found by ASTOR.

Then, manually analyze the generated patches and answer the following questions:

1. Does ASTOR generate a meaningful patch? Use a diff-tool to analyze the code between the original RERS problems, their buggy versions, and the patched versions. (approx. 1/2 A4)
2. Why are/aren't the patches that were generated meaningful. (approx. 1/2 A4)

Task 5: Comparing to LearnLib (1 A4 - 10 Points)

LearnLib is considered to be the state-of-the-art tool for learning state machine models using active learning. For this task, you are asked to compare your implementation of your active learning algorithm to LearnLib.

To run the state-of-the-art Learning with L*, you can use the following code:

```
public static void runLearnLib() {
    LearnLibRunner llr = new LearnLibRunner();
    llr.start(1);
}
```

You can specify the w parameter as the argument to the `start(int w)` method. This corresponds to the "w" parameter in your own W-method. In the file `LearnLibRunner.java` you can look at the code for setting up LearnLib. We have chosen the L* algorithm by default, but LearnLib includes more efficient methods for learning models from software. One of these methods is the TTT algorithm. You can change the line `MealyLearner<String, String> learner = lstar;` to `MealyLearner<String, String> learner = ttt;` Compare the state of the art TTT algorithm with your own implementation of L* and answer the following:

- Is your method able to find the same models as TTT? Why, or why not? (approx. 1/2 A4)
- Compare the runtime, as well as the number of membership queries of your method to the state-of-the-art. How does your implementation compare to the state-of-the-art? To keep track of the number of queries for your own implementation, you can modify the LearningTracker to increment a counter for every time the runNextTrace method is called. (approx. 1/2 A4)

ASSESSMENT CRITERIA

All slides from lectures, all literature from previous lab assignments.

PRODUCTS

A report (max 8 A4 pages). The report can be extended with at most three A4 pages for visualizations (no text). Also provide an archive (tar/zip) containing all your code

Make sure you have also provided some instructions on how to run your code.

ASSESSMENT CRITERIA

The assignment will be assessed by the teachers.

Knockout criteria (will not be evaluated if unsatisfied):
Over 10 pages, code not executing.

Task 1 will be assessed using these criteria:

Criteria	*Bad*	*Good*	*Evaluation*
Run	*Shows wrong behavior, bad design choices*	*Good working solution, solid algorithm design*	*0-3 points*
Properties	*Incorrect or highlighting side issues*	*Core strengths and weaknesses listed*	*0-3 points*
Depth	*Code and run are superficial*	*Both code and run explain and demonstrate core behavior of the algorithm*	*0-4 points*

10 points can be ontained in total for each topic (lab 1,2,3,4), 40 in total for task 1.

Task 2, 3, 4 and 5 will be assessed using these criteria:

Criteria	*Bad*	*Good*	*Evaluation*
Correctness	*Techniques not explained, incorrectly used*	*Implemented correctly, explanation accurate*	*0-3 points*
Validity	*Analysis missing or wrong*	*Analysis is insightful, conclusion supported*	*0-3 points*
Depth	*Results missing or no reasonable explanation for the results*	*Thorough explanation of the obtained results*	*0-4 points*

10 points can be ontained in total for each task

Your total score will be determined by summing up all points. In total 80 points can be obtained. Your total grade for this course will be calculated as min(10, $(1 + points)/8$*).*

SUPERVISION AND HELP

We use Mattermost for this assignment. Under channel Final Lab, you may ask questions to the teacher, TAs, and fellow students. It is wise to ask for help when encountering start-up problems related to loading the virtual machine or getting a tool to execute. Experience teaches that students typically answer within an hour, TAs within a day, and the teacher the next working day. When asking a question to a TA or teacher, your questions may be forwarded to the channel to get answers from fellow students.

SUBMISSION AND FEEDBACK

Submit your work in Brightspace, under assignments.