CS 182 Project Guidelines

This **team-based** course project focuses on software testing and aims to provide you with an opportunity to explore and experience program testing in practice. You may work as **a team of three**, and there are multiple project options. Please choose **only one** option from the list below.

Project Options

Option 1: Classical Software Testing

JQF is a mature academic testing tool for Java programs. Gaining hands-on experience with JQF can go a long way toward understanding software testing.

For this option, you will need to:

1. Download and set up JQF.
2. Fuzz the program under test (PUT, available here) using JQF.
   2.1. Compile the PUT with essential dependencies using `javac -cp`.
   2.2. Run `jqf-random` on the PUT bytecode for 10 iterations.
3. Print out the coverages (i.e., trace events in JQF) of the PUT.
   3.1. Modify the source code of JQF to print out the branch decision for every event in the format of `Event ID: <event-id>, branch arm: <branch-arm>`.
   3.2. Build JQF again.
   3.3. Run `jqf-random` on the PUT bytecode for 10 iterations and print out the branch decisions.
4. Change test inputs as needed and print out the coverages.
   4.1. Modify the source code of JQF to generate the integer 200 for every iteration.
   4.2. Modify the source code of JQF to print out all the trace events in the format of `Thread <thread-name> produced an event <event-info>`.
   4.3. Build JQF again.
   4.4. Run `jqf-random` on the PUT bytecode for 10 iterations and print out the branch decisions.

Here are some resources that may be of help to you.

1. JQF codebase: https://github.com/rohanpadhye/JQF.
2. JQF tool paper: https://dl.acm.org/doi/10.1145/3293882.3339002.
3. JQF guidance interface: https://github.com/rohanpadhye/JQF/wiki/The-Guidance-interface.

Option 2: AI-based Input Generation

Pre-trained large language models (LLMs) have recently emerged as a breakthrough technique, and many researchers have been exploring how LLMs can be applied to software testing for generating structured data, such as XML files.

For this option, you will use the pre-trained GPT2 as the input generator to generate XML files and check if the generated files are syntactically valid. Specifically, you will need to:

1. Prepare a valid XML file as the example input for GPT2.
    a. Here are some examples for your reference: example 1, example 2.
    b. Please generate another valid XML file according to XML syntax rules.
2. Write a program to use libxml2 to parse XML files and to check if they are syntactically valid.
    a. Here are some examples for your reference.
3. Design at least five prompt templates to query the GPT2 model.
    a. For example, you can prompt GPT2 in this way: 'Here is an example XML file. Please generate another one. <sample-xml>'.
    b. You can either provide the example input or not.
4. Prompt GPT2 for XML file generation.
    a. Import the Hugging Face checkpoint GPT2.
    b. Prompt GPT2 using your designed prompt templates.
    c. Collect the response from GPT2 (i.e., generated XML files).
5. Check syntactic validity.
    a. Use the program in Step 2 to check the syntactic validity of the generated XML files and record the results.
6. Compare prompt templates.
    a. Which kinds of prompts perform well in generating valid XML files?
    b. Which performs badly?
    c. Why?

Option 3: AI-based Coverage Analysis

Pre-trained large language models (LLMs) have recently emerged as a breakthrough technique, and many researchers have been exploring how LLMs can be applied to software testing.

For this option, you will use the pre-trained transformer model, BERT, to predict the coverage of a word frequency counting program, wf, when processing a given text file.

More detailed requirements are available here in a Google Colab Jupyter Notebook. There are a number of TODO tasks left for you to complete, and you are also very encouraged to create your own pipeline.

Here are some resources on BERT that may be of help to you.

1. Source code: https://github.com/google-research/bert.

2. Research paper: https://arxiv.org/abs/1810.04805.
3. Hugging Face checkpoint: https://huggingface.co/bert-base-uncased.
4. Model documentation on Hugging Face: https://huggingface.co/docs/transformers/model_doc/bert.

Assessment and Deliverables

We require each team to (1) submit a short **proposal** to specify your project option, (2) do a **demonstration** of your project, and (3) document your design and results in a **report**.

The project counts **35%** of the final score. Below is a breakdown of the above three tasks. Detailed requirements and grading guidelines for the demonstration and report will come out shortly.

Task	Description	Weight	Deadline (Tentative)
Proposal	The team formation and the project option.	5%	11/09/2023
Demonstration	Live demo of your implementation.	15%	Lecture / discussion
Report	Details of your implementation and analysis of your results.	15%	12/08/2023