EECE 513: Assignment 3 – Symbolic Fault-injection using SymPLFIED

In this assignment, you will use the SymPLFIED framework for injecting faults into a program. The paper below describes the design and implementation of SymPLFIED.

http://synergy.ece.ubc.ca/karthik/2009/11/12/symplfied-symbolic

In this assignment, you will NOT need to write code within the SymPLFIED framework, but you will use SymPLFIED to inject faults and analyze the results.

The assignment consists of three phases as follows:

Phase 1: Install SymPLFIED

In this phase, you will need to download SymPLFIED from the course homepage and unzip and untar it. SymPLFIED has two subdirectories, namely, *comprehensive* and *mips2maude*. *comprehensive* has the Maude files that form the core of SymPLFIED and *mips2maude* has the Perl scripts to translate MIPS assembly code to Maude.

You will also need to install the supporting tools for SymPLFIED, namely:

1. Maude rewriting logic framework: Download and install it from the website: http://maude.cs.uiuc.edu
2. SimpleScalar simulator: You can download the SimpleScalar simulator from the following website: http://www.simplescalar.com/
3. SimpleScalar-gcc: Make sure you download the gcc cross-compiler for SimpleScalar also from the same website. Follow the installation instructions. We will assume that both SimpleScalar and SimpleScalar-gcc are installed under the same parent directory.

You will need both Perl and Python (v 2.4 or above) to run the scripts.

We will refer to the following paths throughout the document. **You need to set the environment variables in your shell (e.g., bash) to reflect these entries.**

1. SYM_HOME – The parent directory in which you installed SymPLFIED
2. SS_HOME – The parent directory in which you installed SimpleScalar and SimpleScalar-gcc (i.e., the common parent of both these installations).
3. MAUDE_HOME – The directory in which you installed the Maude executable

Phase 2: Compile and run the tcas program using SymPLFIED

In this phase, you will compile the tcas program using the SimpleScalar gcc simulator that you installed in the previous phase. The goal is to generate the assembly code of the program, translate it to Maude equivalent and then run it under Maude. You would need to compile the program using the –S option of gcc to generate the .s files, and then translate it to Maude using our custom scripts. You would then need to specify an input (given to you as the file input.txt) for executing the program. Finally, you would execute it in SimpleScalar and compare the output. However, these steps have been automated for you in the form of a script, doAll.sh, which calls a series of scripts. You need to do the following before executing it

1. Create a new directory under the mips2maude directory for tcas

2. Copy the file tcas.c to this directory
3. Copy the file input.txt to this directory

Once you do the above steps, cd to the newly created directory and type exactly:

```
../scripts/doAll.sh tcas
```

When this step is done, it should have created a file called tcas.maude. You are strongly encouraged to go over the tcas.maude file and verify that all the functions in tcas.c are present (with a –func added after their names and '_' converted to '–'). Also, ensure that the input and output of the program are specified correctly in the file. SymPLFIED uses the '&' symbol to separate the elements in an input/output stream. If you are wondering how the script knows what the output is, it actually executes the program using the SimpleScalar simulator to generate the output.

Once the tcas.maude file is generated, switch to the SymPLFIED/comprehensive directory and go to the scripts sub-directory. Ensure that the name of the file loaded in the script load.maude to tcas.maude (after in SymPLFIED). Finally, type

```
$(MAUDE_HOME)/maude.$(PLATFORM) scripts/normal.maude
```

where platform = IntelDarwin/Windows/Linux

and normal.maude is the name of the script file that executes the program. The output produced by Maude should match the one expected in the program. Note that the way SymPLFIED displays its output may be different from the one produced by SimpleScalar, but you should be able to match the outputs nonetheless.

Phase 3: Inject faults into the tcas program and analyze the results

The final and most important phase is to perform a systematic fault-injection campaign into tcas using the SymPLFIED framework. Again, you'll use the scripts to automate most of the fault-injection and analysis parts. However, you will need to interpret the results of the analysis based on your understanding of the tcas code.

To inject the faults, we will first generate fault-injection scripts using an automated script. The FI scripts split the tasks of injecting faults into multiple processes which can be launched in parallel on a single machine or on a cluster (this way you'll not risk overrunning the memory capacity of a single machine).

To generate the fault-injection scripts, first create the sub-directories fiScripts and fiOutputs, both under the tcas directory. Then launch the script as follows:

```
../scripts/generateErrorScripts.py tcas <step size> 0
```

where <step size> indicates the granularity at which you want to decompose the injections. In other words, it is the number of consecutive instructions injected in the program by the script. Futher, the third argument 0 represents register errors.

The script will generate a series of scripts to inject faults into the tcas program. These scripts will be in the fiScripts directory. It will also create a single master script called runAllSearches.py in the tcas directory to launch all the scripts in the fiScripts directory. You can also modify this script to submit the jobs to a cluster.

The scripts will take a long time to run, with some even going into an infinite loop. It all depends on how much memory your machine has - SymPLFIED is highly memory intensive. I suggest you set a time out of say 4-5 hours and kill tasks if they exceed this time limit. For example, on my core i7 machine with 8GB of RAM, I decompose the search into 20 tasks of which only about 15 tasks complete in a span of 5 hours.

The scripts upon termination (or not) write their outputs to the fiOutputs directory. Take a look at the output files and see if you can understand their format (it is based on what we saw in class). To make your job easier however, we have another script to analyze the output files and to create a summary of the result. To run this script,

```
../scripts/analyzeOutputs.py tcas <Number of output files>
```

where the second argument is dictated by how many scripts you have generated.

The format of the file produced by the above script is as follows. It first identifies all distinct outputs that do not result in crashes in the tcas program due to the injected fault. For each output, it identifies the name of the function, the line number at which the fault was injected (corresponding to the instruction no in tcas.maude), and the number of solutions that map to the fault. It also identifies the total number of solutions and number of distinct outputs.

You will need to produce a report based on your analysis of the outputs as follows:

1. Group the outputs and injected faults into semantically meaningful categories. For example, the output 2 produced by tcas is a catastrophic error for reasons explained in class and is produced by a specific kind of error. You will need to perform a similar analysis for the other categories and come up with explanations as to why the faults result in the particular failures.
2. Create a table summarizing the results, i.e., each semantic category is a row in the table, and you should report both the no of outputs and solutions in the category. Note that a single category may encompass more than one output. For example, both 0 and 2 are valid, but incorrect outputs for the input (although only 2 results in the catastrophic outcome described earlier).
3. How long did the experiments take to run ? How many tasks completed ?
4. What were the major stumbling blocks in running SymPLFIED. How will you rate the user experience ? Give some concrete suggestions for improvement.
5. Based on the analysis you did above, what kinds of error detectors will you consider adding to the program. You need not add the detectors, but they should be chosen to mitigate as many failures as possible.

Submission instructions:

Your report should be no more than four pages in length (including tables and figures), and written in 12 point font, single line spacing and single column. There is no need to include any code or screenshots in your report though you are welcome to do so if it helps you to answer the questions. Please be brief in your answers.

The report is due in class on December 1st in hard-copy form. Late submissions will not be accepted for this assignment.