Software Testing Project: Phase 1 – Random Testing on LLVM IR

1 Introduction

There are several goals for this assignment:

- Designing a simple random testing tool.
- Gaining exposure to LLVM in general and the LLVM IR which is the intermediate representation used by LLVM.
- Using LLVM to perform a sample testing.

This assignment is a group assignment. Each team should have **2 members**

1.1 Random Testing

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail. Random testing for hardware was first examined by Melvin Breuer in 1971 and initial effort to evaluate its effectiveness was done by Pratima and Vishwani Agrawal in 1975. In software, Duran and Ntafos had examined random testing in 1984. Source: Wikipedia

2 Task 1: Implementing Random Testing in LLVM

In this task, you will design a random testing tool using LLVM. LLVM is set of compiler infrastructure tools. You have seen `clang` and `clang++`, the LLVM C and C++ compilers in the demo. In this task, you will write an LLVM pass to perform the analysis on loop-free programs.

Example 1: For example, consider the c program below:

```
int myAbs(int x) {
    if (x > 0) {
        return x;
    }
    else {
        return x; // bug: should be '-x'
    }
}
```

The program contains a bug. Now the random tests for this function could be {123, 36, -35, 48, 0}. Only the value '-35' triggers the bug.

The generated LLVM IR will have four basic blocks with labels: entry, if.then, if.else and if.end. As explained int the demo, a random tester will randomly pick a path in the program. Your random tester should return a value for the input argument x such that the running the program with the value of x would cause the corresponding path to be executed:

- value of x: 123 {true path}
- value of x: -35 {false path}

Note 1: For simplicity you can consider the arguments are defined in the beginning of the program using *alloca* instruction. Moreover, input arguments are always marked with a_1, a_2, etc. For example, the above example program would be changed to the following and provided to your random tester:

```
int main() {
    int a1;
    int x = a1;
     if (x > 0) {
        return x;
    }
    else {
        return x; // bug: should be '-x'
    }
}
```

Note 2: Your LLVM pass should provide the value for $a1$ and the sequence of traversed LLVM basic blocks as output:

- $a1 = 123$
- Sequence of Basic Blocks:
- entry
- if.then,
- if.end,

3 Task 2: Adding Support for Loops to the Random Tester

In order to handle loops, the designed tester should be continued until it can jump out of the loop. Extend your analysis from task 1 to support loops. Your tester should pick a random path such that the starting point

of the path is the **entry** basic block and the end of the path is a basic block with no outgoing edge (the last instruction is a *ret* instruction).

Example 3: In this example, consider the c program with a loop below and its respective CFG in Figure 1:

```
int main() {
  int a1, a2;
  int b,c;
  int i = 0;
  while (i < 2) {
    if (a1 > 0){
      b = 7;
      a1 = -1;
    } else
      c = 12;
    i++;
  }
}
```

Applying the random tester from task 2 (assuming the chosen path goes through the if.then basic block in the first iteration) the output value would be $a1 = 12$ (12 as a positive number) and the traversed path would be:

- $a1 = 12$
- Sequence of Basic Blocks:
- %0,
- %2,
- %5,
- %8,
- %10,
- %2,
- %5,
- %9,
- %10,
- %2,
- %13

4 Submission - Due Date: Wed 24th Farvardin 1401, 11.59pm

Please submit the following in a single archive file (`zip` or `tgz`):

1. A report in PDF (`asg1.pdf`). It should describe your design for task 1 and task 2, the implementation details, the algorithm used, and the steps to build and run your code, and finally the output of examples programs tested.
2. Your source code.
3. Your test C files with corresponding `ll` files.

Make sure your reports and source files contain information about your name, matric number and email. Your zip files should have the format *Surname-Matric*-asg1.zip (or `tgz`).

Please email the submissions to (`sajjadrsm@gmail.com`).

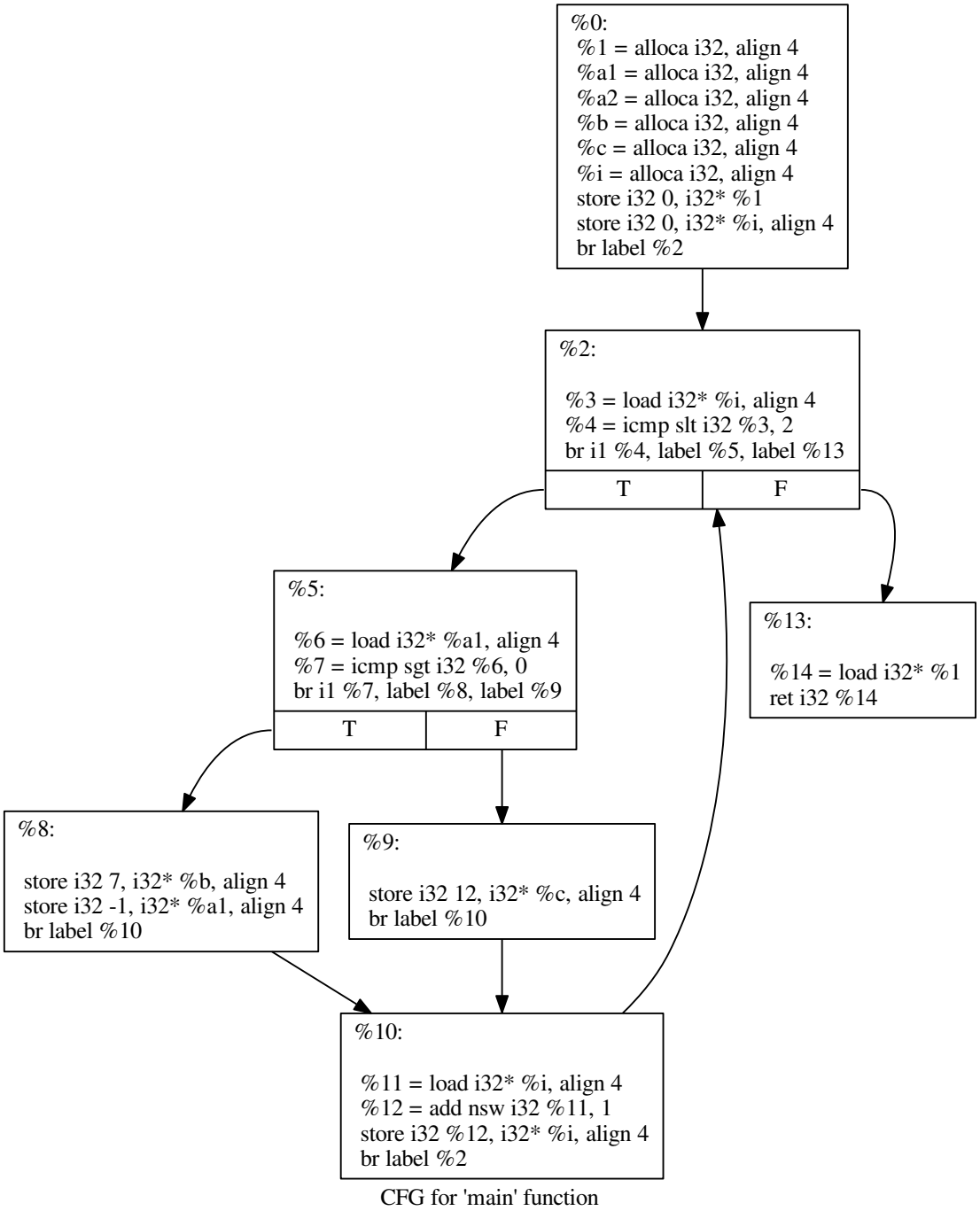

Figure 1: CFG of C Program with Loop