Programming Assignment 2

Overview

The goal of this assignment is to learn how to create a path profiler using LLVM. A path profiler can find which path get executed most frequently and it contains two key components: a instrumentation tool which will insert path profiling code to original program and a runtime library which keep counting path and produce profile output. The high level idea of a path profiler is similar to the edge profiler we have discussed on class.

Detailed Instructions

1. First, read this paper on efficient path profiling. Specifically, read Section 3 and you are required to implement the path profiling algorithm covered in 3.1, 3.2 and 3.4. Section 3.3 mentioned an optimization for 3.2 but this optimization is optional.
2. You will be implementing a instrumentation pass for path profiling, but you will only have to do so for the bodies of innermost loops within a function. These are loops that do not contain any loops within themselves. However, if one function contains multiple loops, you will profile the innermost loop in each loop. As a result, a path is uniquely identified as a LoopId and PathId. You can assume they are both integers.
3. Since you will be dealing with innermost loop, it will not contain any other loop hence your Entry node will be loop header and your Exit node is the loop latch, i.e., last basic block before the backedge of the loop that returns to the loop header. However, since your loop may contain goto and break, the body of a loop with multiple exits should be modeled as though all actual exits (including the latch) are immediately followed by a unique pseudo-exit for the loop body. You do not have to consider return or switch statements inside the loop, as well as the indirect jump.
4. Information about loops inside LLVM is available via the LoopInfo class, which is enabled in the template provided to you. You will be particularly interested in the begin(), end(), and empty() methods method of LoopInfo as well as the methods of the same name and contains, getLoopLatch, and getExitingBlocks or isExitingBlock for Loop.
5. After an instrumentation is done, you need to implement a runtime library which your instrumented program will call. In the skeleton code, we provided you four function prototypes and you are expected to implement them. Your instrumentation pass need to only call init_path_reg(), inc_path_reg() and finalize_path_reg(). You can assume dump_path_regs() is already being inserted in all test programs. The goal of inc_path_reg() is to increase the path register itself (r in the paper) and finalize_path_reg() is used to update the path counter (count[r] in the paper).
6. You can modify everything in the provided skeleton, including the running script and runtime library header. However, in your report, please justify your modification and

explain how to test it. Also, you can implement your runtime in both C or C++ (I recommend you to use C++ since you can use STL). Remember to change your Makefile if you decide to use C.

7. You may assume all loops are in canonical loop simplified form and you will not need to instrument critical edges in the graph

Example Output

We have given you an example test program in the skeleton code. Since the skeleton code do not work with the running script, here we give you the example output for the test program. When the simpleloop.c is instrumented correctly and called with argc=2 and argv[1]="10", I expect output like this:

```
1       0       for.cond->Exit
1       1       for.cond->for.body->if.else->if.end->for.inc->Exit
1       2       for.cond->for.body->if.then->if.end->for.inc->Exit
1       0       1
1       1       3
1       2       7
```

The first column is the LoopId (started with 1) and the second column is PathId (started with 0). The first three line show that simpleloop.c has one loop with three path and it prints a human readable format for each path (you can use getName() to find the name of each basic block). Notice the basic block for.inc is the loop latch and for.cond is the loop header. Also, All three path are connected to a pseudo Exit block.

The second three line showed that that path 0 executed 1 time, path 1 executed 3 times and path 2 executed 7 times. Your program must produce the exactly the same output for the simpleloop.c. And we will be testing your program with other test cases. When you have multiple loops, I expect all paths within the same loop are printed together.

As you can see from our running script, we actually produce tow files, one for path description (path-desc.ll) and one for profile output (path-prof.ll).

Submission Guideline

Programming Assignment 2 is due on Tuesday, Feb 28th, 11:59pm. You will be submitting all your code, your own test cases (if there is any) and a short report describing your solution and the structure of your archive (less than 1 page). If you made any changes that is different from the provided running script, explain them in the report and how to test your code.