CS453 Homework #4: Due on Oct 30 23:59 (200 points).

Building LLVM Branch Condition Coverage Tool

Overview

In this homework, you will build an LLVM code analyzer that instruments a target C program to record branch coverage in LLVM IR achieved in executions. Your task is to implement the code analyzer as an LLVM pass that transforms the target program in IR level. You need to measure the coverage of grep 1.2 to demonstrate that your tool works correctly.

LLVM branch coverage

LLVM branch coverage checks if every branch condition (conditional jump) in a target program was ever evaluated as true and false in test executions. In this homework, we define a branch condition as the conditions of br instructions and switch instructions in LLVM IR. For each br instruction in LLVM IR, branch coverage generates two test requirements: one that is satisfied when the evaluation result is true, and the other that is satisfied when the evaluation result is false.

Consider that branch coverage generates one test requirement for each case statement under a switch statement. For example, for the switch statement switch(x){case 0:… case 1:… case 2:… default: …}, there are 4 test requirements each of which is satisfied when x is 0, 1, 2, and one of the other values, correspondingly (you must count for “implicit default” as you did in homework#3).

You do not need to consider other types of branching instructions. Note that the relation between branching statements in source code and branching instructions in LLVM IR is often **non-trivial** because one condition in source code can be translated into multiple branching instructions with logical short-circuit evaluation.

Tool design

In this homework, you need to write an LLVM Pass that inserts probes to a target program, which monitors and checks the runtime information related to condition coverage. To check each test requirement of branch condition coverage is covered (achieved) in a test, we need to monitor which condition was evaluated and what was the result in an execution. A coverage tool can achieve this goal by inserting a probe to every branch condition related instruction to extract runtime information (actual values) on the condition evaluation.

You need to write down an LLVM Pass that automatically inserts the probe to proper locations in a target program. An LLVM Pass can traverse and modify the IR of a target program, generated by an LLVM front-end. Once your LLVM Pass generates a modified IR by inserting the probes, an LLVM back-end generates the binary code whose execution produces the data of the coverage achievement as well.

You should implement your own LLVM Pass based on the given template (see “LLVM How-to for Homework#4”). The source code of a sample LLVM Pass IntWrite is also given to you. You can reuse the source code of the LLVM Pass and the runtime module.

Tasks

You need to complete the following specific tasks, and **describe the results clearly** in your reports:

1. Explain how branching statements (e.g., if, for, switch) and logical operations in C are translated into LLVM IR instructions (30 points)
2. Describe at which location your tool need to insert a probe and what type of information a probe monitors from a target program (30 points)
3. Write your LLVM Pass and the probe program. You should construct the LLVM Pass based on the given template CCov.cpp (70 points). You should write the probe program as well. Your tool should meet the following requirements:
	1. Your code should have sufficient comments that document your design.
	2. At the end of instrumentation, your tool must report how many branch conditions exist in the target program.
	3. An instrumented target program should produce a file coverage that reports how many times each test requirement is covered in an execution. A branch condition is identified as a pair of line number and an integer index that differentiate the condition from the other conditions of the same line number.
	4. If an instrumented target program is executed multiple times, coverage must report the accumulated result.
	5. coverage must look like the following:

…

311.0 -> 20, 2

311.1 -> 20, 0

311.2 -> 18, 2

…

657.0 -> 0

657.1 -> 0

…

657.10 -> 1

…

657.33 -> 0

…

Total: XXX, Covered: YYY

This is from the result with grep.c. At line, “X.Y -> n, m” means that a branch condition located at line X with an index Y was evaluated as true for n times and evaluated as false for m time in the previous test executions. Note that line 311 in grep.c has three branch conditions.

“X.Y -> n” represents coverage related to a switch statement, and it means that for the switch statement at line X, an execution took the case of index Y for n times in previous executions. Note that line 657 is a switch statement whose body contains total 33 case and 1 default statements. The last line reports total number of test requirements (i.e. double of the number of branch conditions) in a target program, and how many of these were ever covered in executions.

1. Apply your tool to grep.c (not grep\_prep.c), and submit the output (coverage) after executing the instrumented grep with the following test cases (50 points):

./grep –n "if" grep.c
./grep –E "[0-9][0-9]+" grep.c
./grep -E “[[:digit:]][[:alpha:]]” grep.c

5. Discuss advantages/disadvantages of instrumenting LLVM IR over instrumenting source code (as you did in homework#3) (20 points).

Submission

Send your report and implementation to TAs via e-mail (hongshin@gmail.com). The report should clearly describe all results of the tasks. Your implementation should include the source code of both your LLVM Pass class and the runtime module.

The homework is evaluated primary based on your softcopy. Your softcopy should contain the answer for every question explicitly. If your report and softcopy are difficult to understand, you will not get full points. The source code must be able to be compiled with LLVM and understandable (i.e., you must write comments to explain your code).

You should submit the hardcopy of your report to the homework box at 4th floor, N1.