COMP 621 Program Analysis and Transformations
Assignment #2

Using the McLab Framework for Analysis and Profiling

Due: Tuesday October 25, 2016

Overview:

The purpose of this assignment is to give you some practice designing flow analyses, and to familiarize you with the McLab analysis framework, so you will be comfortable with it for your course project. You have been given an introduction to the system during the lecture. Chapter 5 of Jesse Doherty's master's thesis (`http://www.sable.mcgill.ca/mclab/projects/mcsaf/`) contains a detailed description of the analysis framework.

Please feel free to ask questions to the graduate students in my research lab, the TA during his office hours, and also feel free to share information with your classmates on the MyCourses discussion board. The objective of this assignment is to get everyone comfortable with the environment - so don't spend a lot of time getting stuck on some small technical issue.

Some program examples which are known to run correctly with McLab in addition to the rest of the code provided in the demo are available in a Git repository here `https://github.com/Sable/mcsaf-intro`. You are invited to provide additional examples, the examples provided represent a minimum amount of features to support. You will have to update the Makefile to use the examples that have been added since the demo.

Question 1 - *Measuring code coverage*

Code coverage is a metric which measures roughly how much (and which parts) of the source code of a program was exercised in a given execution of that program. There exist many different flavors of coverage data, for example describing which functions were called, which statements which executed, which branches or control flow paths were taken.

MATLAB's profiler, which you used in assignment 1, does provide some limited coverage information. It can tell you, for each function, which lines were executed. Here's an example. Suppose you have a function test:

```
function test()
  x = 1;
  y = 2;
  if x < y
    disp('then branch')
  else
```

```
    disp('else branch')
  end
end
```

Then you can get coverage information like this:

```
>> profile on
>> test()
>> profile off
>> stats = profile('info')
>> stats.FunctionTable(1).ExecutedLines

ans =

     2     1     0
     3     1     0
     4     1     0
     5     1     0
     9     1     0
```

Here the left column contains line numbers, the middle column corresponds to how many times that line was executed, and the third column corresponds to how much time was spent on that line (we don't care about this). Lines 2 and 3 correspond to the assignments, line 4 was the comparison, line 5 is the then branch, and line 9 is the end of the function.

Notice that lines that could have been executed but weren't, like line 7, aren't included in the table. This is bad. Typically you want to know which lines were executable, so that you have some meaningful measure of how much of the code was executed, e.g. 4/5 lines. You can't just take the number of lines in the file because you don't want to count blank lines, comments, and otherwise meaningless lines (like lines 6 and 8, which just consist of the 'else' and 'end' keywords). (There's an undocumented function, callstats, which among other things lets you get at an array of executable lines for a file, but it's brittle, and doesn't handle subfunctions and nested functions well.)

For this question, you should implement your own line coverage collection mechanism using McLab. Each node in the Natlab AST contains the line number of the first token corresponding to it in the original file (it's not perfect information, but in practice it tends to be good enough). You can get at the line number by calling the `getStartLine()` method, which is defined on all nodes.

This information can then be "injected" in the program by instrumenting each statement so that, if it's executed, the fact that that line was executed is recorded somewhere (perhaps in some global data structure). Before the program exits, it can use this information to generate a coverage report. For our purposes, this can just be a plain text file, each line consisting of filename, line number, and 1 or 0 for executed or not executed.

For simplicity, you can use the same overall structure we saw in class for the profiler example – a driver function written in MATLAB, and a source to source transformation which is aware of what that driver function expects.

If you like (i.e. optional), you can enrich your tool to profile more information, such as the frequency of each instruction, or the loop depth of each instruction.

In the report, you should submit a short description of your approach, how you are handling each MATLAB language feature with the smallest example that illustrates each, the resulting instrumented code, and the coverage report obtained on that example.

You should also submit the code to reproduce your experiments. It should automatically compile your profiler code written in Java, runs it on the examples to obtain instrumented code, and run the instrumented code on MATLAB to obtain the coverage report. It will be used by the TA for grading. You may reuse everything in the demo repository (mcsaf-intro) for that purpose.

Question 2 - *Implementing an Analysis in McLab*

McLab provides a structured flow analysis framework that can be used to implement both forward and backward analyses. For this question you can choose `one` of the following analyses to implement. However, you are strongly encouraged to suggest your own analysis, just give a clear definition of it, and make sure it is not an analysis that already exists in McSaf. The problems below are listed in increasing levels of difficulty.

- **Possibly Not-Defined Analysis:** In MATLAB variables may not be defined before they are used. For example, in the following programming fragment `a` is defined only on the if branch and `b` is defined only on the else branch.

  ```
  if (exp)
    a = ones(3);
  else
    b = ones(3);
  endif
  S: a(1) = b(1) // both a and b are possibly not defined
  ```

  Try this program and see what it does when `exp` is true, and when `exp` is false. This should show you that knowing whether a variable is possibly not defined is useful for developing back-ends and other analyses.

- **IsInteger Analysis:** The base type of variables in MATLAB is double, however often variables will only contain integer values (and hence can be stored more efficiently). Write an analysis, that determines at each program point, if a variable is: (1) integer (all values stored in this variable are integer at this program point) or (2) top (the types of the values stored in this variable are unknown).

- **Range Analysis:** For array-based programs it is very useful to know the range of values which summarizes the values a variable may have at a program point. Ranges are usually approximated as a pair [low, high], which indicates that the variable must have a value $\geq low$ and $\leq high$.

In the report, submit a definition of the analysis following the Laurie's 6 steps methodology. For each statement rule, provide both the general rule as well as some small illustrative example(s). Choose the examples to illustrate the subtle points of your analysis and to show that it is correct for these cases.

You should also submit the code to reproduce your experiments. A script should compile you Java code, perform the analysis on all examples, and produce text reports that show the analysis results on each statement of the program. You may add more examples in code than what is covered in the report to show that your analysis works on bigger examples. You are encouraged to reuse the same examples as for Question 1, and supplement them with additional cases that may be tricky to handle. You can reuse everything that is provided in the demo repository.

Question 3 - *Suggest and define a useful analysis*

In this question you do not need to implement the analysis, but you do need to invent the analysis and give some guidelines as to how it should be implemented (of course, you can implement it if you like).

The challenge is to suggest an analysis that could be used to either enable some optmizing transformation in Matlab; or could be used to provide information to the programmer that would be useful for program understanding, program debugging, or refactoring.

Motivate the need for your analysis with an example program.

Describe your analysis using the 6 steps for an analysis. If you implement the analysis, show the results of running the analysis.

What to hand in

Hand in a print-out of your answers for each question at the beginning of class.

Submit a zip archive with the code to reproduce your experiments in Question 1 and 2 by email to the TA at 'erick.lavoie@mail.mcgill.ca'. If you are using the repository created for the demo, change the value of the NAME variable in the Makefile to your last name and type 'make release'. The release target will remove intermediary files and the McLabCore jar archive from the directory and create a zip archive with the remaining files in the directory. Make sure typing 'make clean' than 'make' automatically performs all the reproduction steps for both questions correctly, then create the release with 'make release' and send the created zip archive by email.