Homework 2 Dataflow Analysis

15-819O: Program Analysis
Claire Le Goues clegoues@cs.cmu.edu

Due: February 9, 2016, 11:59 pm
150 points total, up to 10 extra credit points available

Assignment Objectives:

- Precisely define an analysis using a lattice and flow functions.
- Implement a dataflow analysis on top of an existing framework based on the concepts of flow functions and lattices.

Notes: This homework is to be done individually. If you would like a private GitHub repos for this assignment, contact me ASAP, and I can give you one.

Submission. For questions 1–3, turn in a PDF file containing your responses via Blackboard, under Homework 2 (a). Note that a .docx is not a .pdf. Include your name at the top of the PDF. For questions 5–8, submit a single zip or tar.gz file on blackboard under Homework 2 (b). Follow the instructions for which files to include where given with question 5–8. Name your files andrewid-hw2a.pdf and andrewid-hw2b.[zip|tar.gz] respectively.

Assignment context. *Integer sign analysis* tracks whether each integer in the program is positive, negative, or zero. The results of this analysis can be used to optimize a program or to circumvent errors like using a negative index into an array. The analysis is broadly similar to the zero analysis discussed in class. For the purposes of this assignment, ignore the possibility of integer overflow.

Dataflow Abstractions, Written

Question 1, Precise abstraction, *(20 points).* Design a "precise" lattice for a single variable. Your lattice should track whether a value is less than zero, greater than zero, equal to zero, greater than or equal to zero, less than or equal to zero, non-zero, or unknown.

> Give (a) the set of lattice elements (b) the ordering relation between them, (c) the top element, and (d) the bottom element.

Question 2, Less precise abstraction, *(10 points).* Design a "less precise" lattice for a single variable. This lattice should only track whether a value is less than zero, greater than zero, equal to zero, or unknown (which in this case will include cases like greater than or equal to zero).

> Define the lattice by giving (a) the set of lattice elements, (b) the ordering relation between them, (c) the top element and (d) the bottom element.

Question 3, Flow function, *(25 points).* For the second, simpler lattice:

> Define a flow function for a multiplication of two variables assigned to a third variable, i.e. of the form $x := y * z$. It should be as precise as possible given the analysis information available. Use precise, unambiguous notation.

Sign Analysis Implementation

In this part of the assignment, you will *implement* an intraprocedural dataflow sign analysis for some real-world imperative programming language, *demonstrate* that it can identify potentially dangerous array accesses, and *document* it such that an informed user can reproduce your results. You can use the equivalent of the less-precise abstraction from above applied to your target language of choice; you will need to think about the abstraction function, additional flow functions, etc for instructions in the target language, but you do not need to document these design decisions precisely as you did above.

Use an existing framework to parse programs in the target language and create the control-flow graph,[1] and then write code that performs the analysis over it. Your analysis code should therefore take as input code in the target language and produce as output some indication either of potential errors or that no potential errors have been detected. This output can either be textual or in the form of "markers" or "tags" in an IDE, depending on what your framework supports.

You may target a language of your choice using the framework of your choice, subject to constraints. The purpose of these constraints is to help ensure that you are performing this task in a way that supports the learning goals of the assignment *and* that I will be able to run your analysis for grading. Thus, the target language must:

- Be imperative (Java, C, C++, etc),
- Be widely used (e.g., by more than 100 projects on GitHub).
- Require no special treatment or coding conventions for the purposes of your analysis. It's OK if I need to run a preprocessor; it's not OK if I need to annotate my code manually with contracts or rewrite everything to be Java 1.4 compliant.

The framework you choose must:

- Be usable/installable and runnable in either a Linux (I'm running Ubuntu) or OS X-based environment. If all you have is Windows (?), either do the assignment in Java/Eclipse and ensure that it's cross-platform, use a virtual machine (VirtualBox: `www.virtualbox.org`) with a Linux install, or make use of campus cluster resources.
- *Not* require me to downgrade from the latest LTS or El Capitan,
- *Not* require me to purchase proprietary software or tools.

You may assume I am competent with a variety of systems and languages. If you are unsure about a given choice or have a compelling reason to violate one of the above-listed constraints, feel free to ask me for feedback before doing the assignment.

Finally, your analysis must use the framework to analyze some control flow graph representation of the input code (bytecode in CFG form is fine, any abstract syntax tree walker-like implementation is not). You must implement the core of the analysis yourself (including but perhaps not limited to the lattice, join, and flow functions; you may also need to turn Kildall's worklist algorithm into real code). For example, if your framework provides something like "runSignAnalysis(CFG graph)", simply calling it and returning the result is insufficient for the purposes of this assignment.

[1] I mean, you can do this "by hand" if you *really* want, but you'll easily quadruple the amount of work you need to do for the assignment.

Suggestions. Here are some options if you don't want to dig: If you are looking to write/target Java, consider either using Soot (`http://sable.github.io/soot/`) or writing a FindBugs plugin (`http://findbugs.sourceforge.net/`). I have succeeded in dropping the Soot jars into a project and writing code against it; the source checkout did not trivially compile last I checked. If you try FindBugs, make sure you're referencing documentation that is current to the version you use, because otherwise setup can get unpleasant.

If you are looking to write OCaml or target C, consider the C Intermediate Language from UC Berkeley (`https://github.com/cil-project/cil`). CIL is easily the best framework for writing new program analyses and transformations; One reason I'm not just mandating everyone use it is that, as of my last check, it is not compatible with the latest version of OCaml (4.02.3, released 07/15; bitrot is a problem). It does work with 4.01.0, which I have installed, so you can count on my machine to have everything set up properly if you go this route. If you want to target C but not write OCaml, LLVM/Clang is also presumably a viable option; I personally have never written this kind of analysis using it.

Note that getting existing frameworks set up and running properly and then learning the API to extend them is an important skill in conducting or using analysis research. You may struggle a bit if you haven't done this very much previously; this is normal.

Additional notes on analysis requirements. Real-world languages typically include a variety of types that correspond to integer. You need only track information for variables that correspond to type `int` in Java or C. Regardless of language, your analysis should cover variable copies, integer constants, addition, subtraction, multiplication, and division as precisely as possible given your lattice. Your analysis should reason about function parameters and local variables; you may assume that globals or fields or similar, including the results of function calls or array accesses, are unknown. You are not required to correctly analyze other operations, though your analysis should not crash on code that includes them.

Question 4, Write test code, *(20 points).*

> Produce one or more standalone pieces of code in the language you are targeting to test your analysis (e.g., TestSign.java, testSign.c). You must include at least one method that includes a potential negative array index operation and at least one function with safe array accesses. You may include more than two functions (and should probably use more than two methods to test your code anyway); if your code enables me to find bugs in your fellow students' implementations, I will give you *extra credit*(1–10 points, depending on how tricksy it is). Submit the test code in a directory named test/ at the top level of the zip file you submit for question 8. If your analysis framework requires that the target language be preprocessed, submit both the preprocessed and unpreprocessed versions of this code.

Question 5, Run your analysis on the test code, *(10 points).*

> Run your analysis on the test code you wrote for question 5. Capture the output of your analysis (as one or more text logs or screenshots, depending on how you wrote it) to show the results of the analysis. If you take a screenshot, resize the window as necessary to show all the relevant output. Submit this record as files in a directory named testOutput at the top level of the zip you submit for question 8. Name your files something intelligible, and use a standard/reasonable file format.

Question 6, Document how to setup and run your analysis, *(10 points).*

Provide a README file at the top level of your code that explains how to set up your analysis and run it on a file, project, or similar written in the target language. Specify dependencies as necessary. If you use GitHub to manage this part of the assignment, a complete README.md will suffice.

Question 7, Submit your analysis code, *(60 points).*

Turn in your analysis code. Include files necessary to get it set up and running, but do *not* include byproducts of the build process or dependencies that I can or should install myself. I reserve the right to inspect or run your analysis code on examples other than the test code you provide, looking both for accuracy and robustness (i.e. it should not throw unexpected exceptions when run on a larger codebase). Use reasonable coding style and commenting such that I can figure out which part of your code does what and convince myself of its correctness.