Automatic Program Analysis

Mini Project

April 20, 2022

Submit all code, examples and documentation (see website for how). Do this assignment in pairs.

Static Analysis with Embellished Monotone Frameworks

The aim of this mini project is to implement Embellished Monotone Frameworks as explained in Chapter 2 of Nielson, Nielson and Hankin (NNH) and in the slides. On top of this, you should create two analyses instances of the framework, and create example programs to convince yourself and the reviewers that your implementation is correct. Your code should be easy to understand, and where necessary include code comments.

The basis of the language to analyze is the While language with procedures.

Hint: in our experience students realize too late that this assignment is harder than it may seem, and that it helps to join to practice sessions to ask questions.

What to Do

Before you proceed, read the material from NNH.

1. Get the template from the website. It already parses (an extended variant of) the While language and builds a labelled AST (do *not* change the labelling, to make it easier to compare your output). The project can be compiled and executed with `stack run -- name`, where *name* is the name of an example file. The template already provides two example files, `cp1` and `fib`.
2. Note that the (Alex/Happy) parser supports much more than you need to at first. Just implement the rules for the constructs that you know from the book, at first. At a later stage you can add support for some of the other constructs in the language. The generated parser and lexer should suffice, but you can change the language if you have to.
3. You should only change the files in `src` and `examples`. The only exception is adding a dependency. Please do this in the usual way and make sure that `stack build` is still working.
4. The starting template already converts a `Program` to a labelled `Program'`. Inspect this to familiarize yourself with the attribute grammar system.
5. Implement functionality inspired by the definitions in NNH to compute administrative information from the AST like the init, the finals, the flow etc. Do this using attribute grammars on the labelled data types (`Program'`, `Proc'`, `Stat'`).

6. Implement Monotone Frameworks (for intraprocedural analysis) using the Maximal Fixed Point algorithm from the book. Convince yourself with example instances for the following two analyses that it works:

   - Constant Propagation
   - Strongly Live Variable Analysis: consider the following program:
     $$x := 1; x := x - 1; x := 2;$$
     A Live Variables analysis will consider x to be live between the first and second assignment, implying that the first assignment is not dead code. But the first assignment is useless since we compute the value of a variable that is only used for computing the value of a dead variable.

     This motivates the notion of a faint variable: a variable is a faint variable if it is dead or if it is only used to calculate new values for faint variables; otherwise it is strongly live. In the above program, x is always faint (and hence not Strongly Live). The purpose of the Strongly Live Variable Analysis is to compute in each program point the set of strongly live variables.

   Note: instances are usually best created with attribute grammars, but the Maximal Fixed Point algorithm is typically not implemented with attribute grammars.

7. Add interprocedural flow and binary transfer functions.

8. Extend your implementation to Embellished Monotone Frameworks, making sure to decouple context from transfer functions. You need only implement bounded call strings but the value of k should be a parameter. With $k = 0$ it should give the same result as the non-embellished version.

   For Constant Propagation implement the embellished montone framework instance, correctly handling scoping issues that may arise. It is essential that the embellished instance reuse the non-embellished one, as explained in the slides.

9. Create example programs, at least 5 for every analysis, ranging from simple ones without procedures to complicated ones with procedures to show that your implementation is correct.

   Make sure your implementation displays the results of the analysis in an easy to read fashion. This may range from generating graphical cfg's with the code and results in them, to listing analysis results for each label (the pretty printer provided already displays your program with the labels attached; do not change the pretty printer for existing cases).

10. Write documentation in which, for each implemented analysis, you walk through an example and explain why what your analysis computes is correct. One of these should have procedures and explain how the context depth influences the outcome. Choose a suitable example.

    In the documentation you should also explain how to compile your source code, what the purpose of each module is, how to run the analyses, and what, in mathematical notation, the transfer functions are that you have

implemented for each analysis (similar to the slides and book). Finally, you should list explicitly any extensions you have implemented on top of the minimal requirements.

11. Doing the above reasonably well and you are on your way to a grade between 7 and 8. To improve your grade:

    - Also create embellished instances for the other analysis.
    - Implement program transformations which optimize the program based on the results from the analyses. For instance, replace variables by constant values using constant propagation or remove unused assignments.
    - Add Shape Analysis as discussed in NNH.
    - Add one more analysis of *your own design* (also put the transfer functions in your documentation and add five example programs per analysis).
    - Add new language constructs such as *break*, *continue*, *simultaneous assignments*, a *case/switch*-statement, *return* and/or a *print/echo*-statement. (You should then also add them to your example programs, and explain the transfer functions for these statements in your documentation). Note that some constructs are easier to handle than others.