cc-2

Stan Rost (6290655) & Jordi Wippert (6303013)

To use our code, one should start at the root of the project and execute the following: `make cabal install`

`cd src ghci Dev.hs`

Now you're in interactive mode and able to start the analysis. This can be done with: `runAnalysis <programName>`

Where `programName` is the filename without extension from one of the files in the examples folder. This should generate output in which the various implementations of the analyses can be found. Also, the list of assigned labels is printed and a list that represents the flow of the given program.

MFP

In order to execute an analysis on a program we implemented the monotone framework (with the maximal fixed point solution) which can be used for both Strongly Live Variable and Constant Propagation analysis. The MFP code takes a few input variables in order to fit both analyses as they differ in various ways.

Strongly Live Variables

In our code, `slv` holds the top level variables for this analysis. It uses a reversed flow, and holds the final labels of the input program as extremal labels. The empty set is chosen as the extremal value. Together with a transferfunction, and two booleans to determine if union or intersection and superset of subset should be used, it forms the input for the MFP algorithm.

The transferfunction takes a list of variable names and a label which can be seen as a program point to start the execution from. The list that is applicable on this specific point is updated by the kill and gen sets that are generated here. The transferfunction combines these sets to find the variables that are live.

Constant Propagation

For `cp` the base instance is built with a normal flow as transitions and the init labels of the program as extremal labels. The extremal value is a list of the variables in the program and are mapped to a value called Top. This value can be updated throughout the program and determines if a constant is always the same. If a variable in the output isn't Top anymore at the end of a analysis, it means that it is static.

Here the transferfunction has the ability to update the value of a variable from Top to a Int value.

Examples and explaination

We've generated a few examples to show the working of our program. All examples can be found in the examples folder ranging from example1 to 5. In the programs there's made use of if-then-else, while and simple operations. There's a lack of examples that use procs as we found that our implementation for embellished monotone frameworks doesn't work properly.

All of our haskell code is in Dev.hs and other additions are made in AttributeGrammar.ag. This first file is split up by comment lines to differentiate the basic, mfp, slv and cp parts.

General remarks

We found it quite hard to implement all of the requested functionalities and are aware of the imcompleteness of it. Especially the conversion of the idea of the embellished framework into haskell code was undoable for us any further than the basic functions on call statements to implement procedures. Next to this, the output we've generated holds various values for the same expressions and statements including some wrong and right ones. Filtering this on the spot to only have the preferred end result didn't work out as expected.