Connor Adsit
Professor Fluet
Software Foundations
22 May 2014

Independent Study Write-up

The original purpose of my project is to define flow soundness of the STLC extended with polymorphism in Coq. Initially, the language that I used as the basis for my project was the STLC defined in Software Foundations extended with integers and let expressions. Stepping was performed with the substitution semantics that was also defined in the text.

When I first attempted to define the control flow relation, I took inspiration from Jan Midtgaard's subcubic control flow algorithm in defining two relations: an input relation, which finds all possible subexpressions of a program; and a primitive flow relation, which holds whenever an expression flows to another expression based upon a program. Although this representation was able to correctly identify flows in basic programs, it was very difficult to use this interpretation to reason about any benefits of performing a control flow analysis. For example, proving flow safety requires two expressions -- a start expression and another expression to which the former can multistep. Because they are not necessarily the same expression, the input relation fails to provide any information that can be used to relate the flows in both programs.

The next attempt was modeled around more traditional control flow analysis definitions, such as those described in "Principles of Program Analysis". I extended the language to have labeled expressions, which could be used in an abstract cache to find a set of values that the enclosed expression could potentially evaluate to. Additionally, I added in an abstract variable environment, but both of these were initially hard to reason with because there is no built in set theory module in Coq, so I ended up using ListSets after defining a decision procedure of equality for expressions. However, the labels ended up convoluting my proofs a little bit -- initially I had a translation function that would produce a labelled expression from an unlabelled expression, which obfuscated reasoning about flows with regard to how an expression steps.

And finally, to aid in the simplification, I remolded the language to make reasoning a little bit easier. Instead of using the provided STLC syntax and semantics, I wrote an environment substitution semantics and restricted the programs to those that were in anormalized form, very similar . As a result, I no longer needed a cache, and was able to successfully define a flow-safe relation which took in a predefined flow context. From there, I was able to prove flow preservation, ie. for all expressions, if the expression is safe under a flow context and if the expression takes a step, the resulting expression is also safe under the same flow context; and flow safety, cleanly and simply. I went on to try and prove flow soundness, using a concrete variable environment, but was unable to make significant progress in the time provided.