The Typeability of Matrix Arithmetic in R with Liquid Haskell Refinements

Alexa Van Hattum (with partners Anjali Pal & Jack Wrenn)

May 2016

Capstone Project Abstract

For our final project in an independent study in advanced topics in Programming Languages, we created a tool with the intention of analyzing matrix arithmetic in R in order to statically surface dynamic errors and warnings. R is a popular programming language used primarily for data-intensive scientific computing, and the basic data type in R is a vector. R is dynamic, thus errors from incorrect vector and matrix operations (such as dimension mismatches) occur at run-time. Interestingly, the semantics for matrix operations in R are more nuanced than basic mathematical convention — for example, vectors with different lengths can be added without error if one's length is a factor of the other, and attempting to add vectors whose lengths do not have this relation produces a run-time warning but not a halted execution. Matrices in R, on the other hand, must be of the same dimensions to be added (with a run-time error produced otherwise).

We set out to determine whether an R program would produce such errors and warnings through static analysis rather than at run-time. Our general approach is to take advantage of the refinement types provided by Liquid Haskell, a static verifier for Haskell. Using refinement types, we can restrict the input and output types of vector and matrix operations to "match" R's run-time behavior for matrix arithmetic. Our task is thus to create a tool that parses R source code into an abstract syntax tree (AST), transpiles the R AST into an equivalent Haskell AST, and finally augments the resulting Haskell module with Liquid Haskell refinements. At this time, we have generated the Haskell functions and Liquid Haskell refinements for most common matrix operations in R. In addition, we have created a tool that can successfully parse a small but reasonable subset of R into a Haskell module. In doing so, we are able to statically reason about the behaviour of a highly dynamic language by leveraging the robust type system and type inference of Haskell. However, we are currently unable to run the full pipeline of this tool, because Liquid Haskell does not support refinements for type class instance declarations, which are necessary in our current iteration in order to handle overloaded R operators such as "+". We are exploring this issue with the developers of Liquid Haskell as well as considering alternative paths forward that do not require refinements on type class instances.