Matvey Stepanov
Mitchell Fanger
Yueshan Wang

CMSC 430 Final Project

Our project involved implementing “apply” and partial application to our compiler (we didn’t, unfortunately, get to work on the interpreter, but the implementations would be much simpler in it anyway). BTW: sorry that the tests are separate files. I could not get the unit tester to work with our code for some reason.

Apply:

One of the more interesting findings was that we, theoretically, could apply on an arbitrarily sized list, and still maintain stack alignment. However, our implementation of apply does not do so. Our implementation of apply also does not allow partial application when using apply, but the process for partial application in apply would be quite similar to the process of partial application in a regular call. See compile-apply and compile-call for details (there is a ton of commenting in our code). One thing we found of interest right off the bat, is that apply is very different from a normal call, because we cannot know statically the size of the amount of arguments provided to the function. Thus, after discussion with Professor Lampropoulos, we were encouraged to continue and disregard stack alignment for the time being.

Partial Application:

Similarly to apply, partial application posed a problem with alignment because there would be no way for the compiler to know at compile time the amount of arguments that have already been applied to a closure. Therefore, one of the interesting things we did, was modify the way we compiled lambdas (the definitions themselves), to take one more thing on the env, which in the code would be the runtime number of arguments that were added to the passed in arguments. (One we know statically and the other we know at runtime). The process of partial application was really cool to us and we had a fun time playing around with the feature we had just created. Another interesting process that we had to incorporate was that we had to compile the statically known arguments first, then shift them on the stack, then add the partially applied arguments behind the statically known arguments, and then add the free variables. This turned out to be a solution which we thought was pretty gross, so we thought about potential other solutions and came up with the idea of reversing the order of compilation of arguments (instead of left to right, we could have done right to left). This was an interesting concept, but we knew that we would have to change much more to actually get that working, so we stuck with shifting the stack.

Overall, this project was really eye opening on the difficulties of having matching (or at least everything in the same relative positioning) stack and compile time environment!