CS51 Final Project - Mini-ML

Joe O'Keefe

The CS51 final project tasks students with implementing a semantic interpreter for the OCaml / ML languages. More specifically, I had to create a number of helper functions in a file "expr.ml" which allowed any generic expression to be broken down into its component parts. This included the free_vars / to_concrete_string functions as well as the subst function which was critical for implementing the various evaluators in "evaluation.ml"

The core of my implementation in "evaluation.ml" consisted of a large match statement for all three evaluators. All three of these functions took an expression as an input, mapped that expression to all possible cases, and then returned a value or continued the recursive deconstruction of the expression.

The key distinction between these functions was the concept of lexical and dynamic scoping. In lexical programming, the function works using the scope within which it was defined (essentially the broadest scope), while in dynamic programming a function will refer back to the most recent declaration of a variable. Both the pure lexical

and substitution evaluators I implemented used lexical scoping. However, the substitution model only works for pure programming languages and can not handle a changing environment.

I chose to create a lexical evaluator as an extension to my substitution and dynamic evaluators because I felt that the lexical model best fit the purpose of the project. Moreover, OCaml is a lexical programming language and a lexical evaluator would be the best suited to parse an OCaml expression.

To test my evaluators I created “tests.ml” and ran unit tests on various types of expressions. Two examples of these are:

1) (eval_l (Num(4)) (Env.create ()) = Env.Val (Num(4)))

2) (eval_s (Binop(LessThan, Num(2), Binop(Plus, Num(2), Num(2)))) (Env.create ()) = Env.Val (Bool(true)))

Both of these tests worked on all three of my evaluators.