Final Project Writeup

Extension: Lexical Scoping Environment

For the extension portion of the CS51 final project, I have decided to incorporate a lexically scoped evaluation function for MiniML. The first two sections of the project implemented the substitution model and the dynamic scoped environment model of a MiniML interpreter.

I was unable to do a thorough testing of the different models, but the conceptual and evaluation differences of each will be described instead.

Substitution Model

The first section of the implementation was for the substitution model. This is the basic notion of evaluation. In this implementation, the variables are replaced by the values that are defined in the profram. This follows the basic mathematic property of transition. The greatest flaw of the substitution evaluation may be that the program must produce a final value and that it will stop running once a value is found. To do the substitution model, each separate expression from the given expression must be evaluated according to the values given by the expression. In this case, numbers, booleans, and functions merely evaluate to themselves. However, the "Let" function must substitute the variable that is equated to expression one in expression three to produce a value. the "Letrec" function follows a similar action except that the expression must be evaluated multiple times according to the program. The functions created in the substitution model all rely on the subst function defined in expr.ml

Dynamic Scoping Environment

Dynamic scoping is when the value of a variable for the body of a function is evaluate in the current environment in which the function or variable was called rather than defined. This allows the the function to use the latest variation of the variable, rather than the static variation. Dynamic scoping is helpful when updating variables. Thus, references and pointers are used to reference the location of the variable rather than the variable itself to allow for updated runs. For instance, in the Let function for the dynamic scope, the variable "v" is added to the environment that keeps the updated values of the variables. This is important as the second expression, the one being evaluated according to the first expression, must be evaluated against the most recent variable value. Although a useful counterpart of various programs, most languages do not use it as the primary scoping mechanism.

Lexical Scoping

For the final extension of the final project, a lexically scoped evaluator was created. Lexical scoping, also called Static Scoping, is when the variable definition only exists in the context of the function. Most of the code for the dynamic scope was used for the lexical scope. The biggest change is the evaluation of the function which now returns a closure.

Same functions with different scopes.

Example 1

let x = 1 in let f = fun y -> x + y in let x = 2 in f 3;; - substitution: 4 - lexical: 4 - dynamic: 5 In the dynamic scope, the most recent assignment for x is 3 so therefore, the function evaluates 3 + 2, which is 5. In the lexical scoping however, the value evaluates to 4 because the first time x in defined in with x = 1. Due to this, the function evaluates 3 + 1 = 4.

Example 2

let f = fun x -> x in f f 3;; - substitution: 3 -lexical: 3 -dynamic: 3