1 Abstract

The purpose of this extension of Miniml is to implement lexical, environment semantics based scoping so that variables will get their values from declarations in their own lexical scope, rather than just the last declaration to be evaluated. One of the advantages of this change is that code will now be easier to read, since the scoping rules follow directly from the structure of the written code. In Dynamic scoping, by contrast, a programmer has to have an intuitive grasp of what order code will run in in order to predict what declaration a variable will use. Another major advantage of lexical scoping is that it allows for the design of modular code: code that exists in separate, abstracted modules that cannot interfere with one another except through defined, controlled channels. Thus, this implementation of lexical scoping could be a building block for future expansions of Miniml, since it is a necessary prerequisite for object oriented and modular programming constructs. The following section will explain how lexical scoping was implemented in this version of Miniml.

2 Implementation

Lexical scoping in Miniml can be implemented two different ways. The first is to use substitution semantics, which is inherently lexical since it substitutes expressions one level at a time. The second method uses environment semantics and closures to 'remember' the scoping environment that a value was in when it was declared. The concept of environment semantics will be discussed first, followed by closures. In order to remember what variables translate to which values, one needs to store all such value-variable pairs in some kind of data structure. In this implementation, the Ocaml Set module was used to create a set of these pairs. The set implementation ensures that the same variable name won't ever be linked to two different values.

Three key functions are used to interact with the environment set: extend, lookup, and close. The extend function is called when a new variable is given a value, so that that value-variable pair can be added to the set. If the variable name is already in the set, the value of that variable is simply updated. The lookup function is called when an evaluation function needs to retrieve the value of a variable from within it's environment. Finally, the close function has a very important purpose that will be explained later, in the discussion of closures.

The eval-l function is used to convert an expression of arbitrary complexity to the most simplified possible representation of its value. It uses an extensive match case statement to determine what to do in all cases currently implemented in Miniml. The primary difference between eval-l and eval-d is its use of the close function in its solution to a function definition. When it sees a function definition (fun x -¿ P), it calls the close function, which creates a new instance of the type Closure, which contains the function expression, as well as the current environment. Now, when the function is applied, it can be called within the scope of the environment in which it was originally defined.

In this way, Miniml can now evaluate expressions lexically, gaining all of the advantages enumerated in the abstract.