[Type here]

Jacob Berman CSCI 3155

Lab 3 Write-up

1. Dynamic Scoping vs. Static Scoping

Dynamic scoping means that when a symbol is referenced, the compiler/interpreter will walk up the symbol-table stack to find the correct instance of the variable to use.
def function2(): Int = { int b = 6; // Dynamic scoping finds this value of b at execution and updates b to 6 return function1(); }

const int b = 10; def function1(): Int = { int a = b + 9; // Static scoping looks for b and finds it as 11 return a; }

2. Yes, the evaluation order specified by the e → e' judgement form is deterministic. As an example, if we look at SearchBinary, we can definitively say that e1 is evaluated first until it is a value, in which case e2 is then evaluated, it is left associative because of this.

3. e1 + e2will be evaluated as followed.

   a. First: e1 will be evaluated recursively until it reaches a base case.

   b. Second: e2 will then be evaluated until IT reaches a base case.

   c. Third: Based on the type of each of those values it will be evaluated either as a number (in which case they will be added) or as a string (in which case they will be concatenated). If we wanted to change the way they are evaluated we would need to change the SearchBinary so that it is e2 → e2' and also SearchBinaryArith so that it is **e1 bop v2** → **e1' bop v2**, this would make it right associative.

4. Short circuiting is useful in that it allows the interpreter to only evaluate what is necessary to get an answer. For example, in **OR** we see that it evaluates e1 all the way to a base case first, then only if e1 is false does it need to evaluate e2, if e1 evaluates to true, then e2 never gets evaluated.

5. Yes, in our small-step interpreter (as well as big step) the **AND** operator does short circuit. This is because it is left associative, so it evaluates e1 down to a value, if that value is true it then evaluates e2, if it is false then e2 is never evaluated, thus it only evaluates when it needs to.