CSCI 3155 Spring 2018 Lab 3
Kyle Gronberg
Noah Leuthaeuser

---

2. **Javascripty Interpreter: Tag Testing, Recursive Functions, and Dynamic Scoping.**

   (a) *Write a test case that behaves differently under dynamic scoping versus static scoping:*

```
const x = 7;
const f = function(y) {return x;}
const g = function(z) {const x = 11; return f(1);}
console.log(g(2));
```

   With dynamic scoping, the above code will print out 11. With static scoping, the code will instead print 7. With dynamic scoping, the most recent declaration of x (in terms of code execution) will replace older declarations. With static scoping, when x is used, it will search for the innermost declaration of x. Because x is used in function f, it checks inside f for a declaration; not finding one, it checks the main body of the program, finds a declaration of x = 7, and uses it.

3. **Javascripty Interpreter: Substitution and Evaluation Order**

   (d) *Explain whether the evaluation order is deterministic as specified by the judgment form* $e \to e'$

   The evaluation order for the small-step semantics is deterministic as there is a specified order of evaluation within the inference rules. For example, for any binary operation, e1 is iteratively stepped until it arrives at a value, then e2 is iteratively stepped until it arrives at a value. Then the binary operation is done on the two values. In big-step semantics, there is no rules specifying the order in which expressions ought to be evaluated, so it is not deterministic.

4. **Evaluation Order.** *What is the evaluation order for $e_1 + e_2$? Explain. How do we change the rules to obtain the opposite evaluation order?*

   According to the inference rules, the evaluation order of $e_1 + e_2$ is left to right. First, e_1 is evaluated to a value, then e_2 is evaluated to a value, then the addition is done. This is enforced by requiring e_1 to be a value before e_2 can begin being stepped towards a value, and the addition operation requiring both expressions to be values. To reverse the order, make it so e_2 is stepped to a value first and e_1 cannot begin to be evaluated until it is done.

5. **Short-Circuit Evaluation.**

   (a) **Concept.** *Give an example that illustrates the usefulness of short-circuit evaluation.*

   Consider:

   ```
   true || ((90-(24/8))*512^2)
   ```

   To evaluate e_2 would be relatively expensive. However, because we know e_1 is true, the whole statement will be true regardless of what e_2 evaluates as. Thus we can skip the evaluation of e_2 with short-circuit evaluation and simply return true. If e_1 had been false, then we could have just returned e_2 without evaluating it any further.

   (b) **Javascripty.** *Consider the small-step operational semantics for Javascripty. Does e_1&&e_2 short-circuit?*

   Yes, it does. Instead of evaluating e_2, it only steps e_1 to a value before performing the binary operation. If e_1 was true, then its entirely up to e_2 as to what return value of the operation is, so we can just return e_2 without evaluating it further. If e_1 is false, then we know that the whole operation will be false regardless of what e_2 evaluates to, so we can return the value v_1 that e_1 stepped to.