CSCI 3155 Spring 2018 Lab 3 Writeup
Kayla Gehring (02/24)
Partner: Cody Luhmann

---

1. **(Question 2a in lab3.pdf) Give one test case that behaves differently under dynamic scoping versus static scoping (and does not crash). Explain the test case and how they behave differently.**

```
const x = 3;
const f1 = function(filler){ return x; }
const f2 = function(filler){ const x = 5; return f1(1); }
console.log(f2(1)); //prints 5 in dynamic, 3 in static
```

   (a) **In dynamic scoping:**
   f2() is called, which itself calls f1(). f1() returns the inner-most binding of x, which is in f2 (const x = 5).

   In other words, using big-step operations, the dynamic scoping prints 5.

   (b) **In static scoping:**
   f2() is called, which calls f1(). f1() returns the outer-most binding of x, which is in line 1 (const x = 3).

   In other words, using small-step operations, the static scoping prints 3.

2. **(Question 3d in lab3.pdf) Explain whether the evaluation order is deterministic as specified by the judgment form $e \longrightarrow e'$.**

   For the small-step operational semantics employed in 2a-c, the evaluation order is deterministic. As discussed in the notes 3.2.2 (p 48), each reduction step is deterministic, as each step is defined as having at most one step of evaluation at a time. These deterministic steps are shown in figures 7-9 of lab3.pdf.

   For big-step operational semantics, the evaluation order is non-deterministic.

3. **(Question 4 in lab3.pdf) Consider the small-step operational semantics for Javascripty shown in Figures 7, 8, and 9. What is the evaluation order for $e_1 + e_2$? Explain. How do we change the rules to obtain the opposite evaluation order?**
   The evaluation order is left to right (eval e_1 then eval e_2). To change this order, the SearchBinary rules would have to be reversed, so that SearchBinary1 evaluates $e_2 \longrightarrow e_2'$ and SearchBinaryArith2 evaluates $e_1 \longrightarrow e_1'$.

   The rules would become:

$$SearchBinary_1 : \frac{e_2 \longrightarrow e_2'}{e_1 bop e_2 \longrightarrow e_1 bop e_2'}$$

$$SearchBinaryArith_2 : \frac{e_1 \longrightarrow e_1' bop \in \{+, -, *, /, <\ .\ <=, >, >=\}}{e_1 bop v_2 \longrightarrow e_1' bop v_2}$$

4. **(Question 5 in lab3.pdf) In this question, we will discuss some issues with short-circuit evaluation.**

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

   Consider the following case, which avoids calling foo() if b happens to be zero.

   ```
   if (b != NaN && foo(b))
   ```

   This can be quite helpful if foo() has a large time-complexity, for example. If b is NaN, foo() is never called and the program can run faster, thanks to the short-circuiting.

   (b) **Javascripty: Consider the small-step operational semantics for Javascripty shown in figures 7-9. Does $e_1 \&\& e_2$ short circuit? Explain.**

   Yes, it does short circuit. SearchBinary1 ensures that e_1 is evaluated first. Once e_1 has been assigned a value, the DoAnd rules apply. If v_1 evaluates as false, DoAndFalse is applied, and e_2 is never evaluated.