Adam Holt
CSCI 3155
Lab 3 pdf

1. 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 in your write-up.

const x=1 const add=function(n) {return function f(x) {return x+n}; }; add(5)

With static scoping, the function call of add(5) will set the value of x=1 from the previous scope, while with dynamic scoping, the x in the function call may be any other x used in a previous scope.

2. Explain whether the evaluation order is deterministic as specified by the judgment form e -> e'.

Yes, the evaluation order is deterministic as specified by the judgement form e -> e'. The function will always evaluate in the same order and result in the same output, assuming the same input is entered.

3. Consider the small-step operational semantics for JAVASCRIPTY shown in Figures 7, 8, and 9. What is the evaluation order for e1 + e2? Explain. How do we change the rules obtain the opposite evaluation order?

For the operation e1 + e2, e1 is first fully evaluated prior to evaluating e2. The small-step judgments show that whenever there is an e2, there is no e1, only v1. Because of this in order to evaluate e2, e1 must first be evaluated until the form v1 + e2 is reached. This is illustrated in the judgment **SearchBinaryArith$_2$**.
In order to reverse the evaluation order, you would need to substitute e1 for v1 and v2 for e2 in the **SearchBinaryArith$_2$** judgment. This would lead to e2 being evaluated before e1. Any other judgments that rely on both e1 and e2, such as the logical AND and OR operators would need to be reversed in order to ensure that the logical still operates as expected.

4. Give an example that illustrates the usefulness of short-circuit evaluation. Explain

a) Example:

```
int denominator
int num
if (denominator != 0 && num / denominator){
        println("hello world")
}
```

The short-circuit occurs in line 3 where the "and" is checked. If the denominator is zero (something that would in this case throw an error, if the second condition was evaluated) the first case in the and will be false. Because it is an "and" check there is no way the rest of the check can evaluate to true, so the "and" check may as well return false without spending the time and space evaluating the second condition. This is useful because if you have something that will throw an error if it is evaluated (like dividing by 0), you can avoid executing it in the one specific case that will throw the error.

(b) JavaScripty:

Yes, e1 && e2 will short-circuit in lab 3. The specific forms we are interested in are the **DoAndTrue** and the **DoAndFalse**. The **DoAndTrue** will evaluate e2, but the **DoAndFalse** will not, because since the first expression is false it will just return the current value of false. This is implemented in the code as recursing on e2 if e1 is true, else just returning false.