# CSCI 3155 LAB 3 WRITEUP

Lubrano, Jason & Abiel Fattore

February 24, 2018

# 1 FEEDBACK

# 2 Question 2

**Problem a).** Hi my name is Jason Lubrano and I'm Abiel Fattore and this is our lab writeup.

```
/* jason lubrano and abiel fattore's jsy tests
        ->> notes are for STATIC */
const x = 5
        // make x = 5
const y = function(x) { return x }
        /* wouldnt get replaced in static,
    replaced by whatever gets passed */
const z = function(a) { return x }
        /* would get replaced in static */
const foo = fucntion(boo) { return y(boo) + z(boo) }
        /* y(boo) returns 10, z(boo) returns 5
            foo(10) = 15 */


jsy.print(a) /* prints 5 */
jsy.print(y(3)) /* prints 3 */
jsy.print(z(4)) /* prints 5 */
jsy.print(foo(10)) /* prints 15 */
```

This test is one of the many test cases we wrote for our program. It is simple, yet defines the biggest difference between (and issues that may arise) from static and dynamic scoping. In **dynamic scoping**, the environment would be updated such that every $x$ would be equal to 5. This is unfortunate because no matter what we pass through our functions $y$ and $z$, we will always print 5 because they want to return $x$.
**Static scoping** is what is seen in most languages such as C++ and Java. Even though we have $const\ x = 5$ written, function $y$ will return whatever the parameter is equal to.
Then we have the function foo. In dynamic scoping: foo would return 10; however in static scoping foo would return 15 because: the $y$ function would return 10, the $z$ function would return 5, so 15.

**Problem b).** Done.

# 3 Question 3

**Problem d).** Determinism is shown in small step semantics by its inference rules. For example, when adding two expressions,$e_1, e_2$, the program has three steps. First it must evaluate the first expression, $e_1$ to a value. Next, the program must evaluate the second expression $e_2$ to a value. Lastly, it must do the addition. This inference rule shows that order matters and thus shows evaluation order is deterministic. In big step semantics, the inference rules don't specify any order when evaluating an expression saying that evaluation order is arbitrary, (or doesn't matter) so this doesn't show determinism.

# 4 Question 4

**Problem T.** he evaluation order for $e_1 + e_2$ starts form the left and works to the right. The inference rule for binary plus in small step semantics enforces this by making it so that expression $e_2$ can't be evaluated until $e_1$ has been evaluated to some value. To obtain the opposite evaluation order, you would make it so expression $e_1$ can't be evaluated until $e_2$ has been evaluated to some value. This would be done in the SearchBinary1 inference rule.

# 5 Question 5

**Problem a).** Example: $e_1 \| e_2$. This example uses short-circuit evaluation by checking if $e_1$ is true, then returns true if so. The second expression, $e_2$ won't be evaluated which would save time in the long run. Short-circuiting still stays true to the logical or operation.
Consider the truth table:

| $e_1$ | $e_2$ | returned |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

For the first two expressions, since $e_1$ is true, it doesn't matter whats the value of $e_2$ because true will be returned irregardless. For the third and fourth expression, $e_1$ gets evaluated to false, so it relies on $e_2$'s value for the return. In each case, $e_2$ gets evaluated.

**Problem b).** The figures are logical and says that if both expression aren't true the false is returned. The expression: $e1$ && $e2$ uses short circuiting because if first expression is true then, the second expression will be returned. If the second expression was true then the returned value would be true. Otherwise, the returned value would be false. An evaluation step is skipped.
Consider the truth table:

| $e_1$ | $e_2$ | returned |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

For the last two expressions, since $e_1$ is false, it doesn't matter whats the value of $e_2$ because false will be returned irregardless. For the first and second expression, $e_1$ gets evaluated to true, so it relies on $e_2$'s value for the return. In each case, $e_2$ gets evaluated.