Eric Minor

Writeup Lab 3

2.a

```
  "DoStatic" should "perform Static" in {


    assertResult(N(2)) {
      iterateStep(parse("const x =1;const f =function f(z) {return z+x}; const x=2; f(1)"))
    }
  }


  "EvalDynamic" should "perform Dynamic" in {


    assertResult(N(3)) {
      evaluate(parse("const x =1;const f =function f(z) {return z+x}; const x=2; f(1)"))
    }
  }
```

Both cases run the program const x =1;const f =function f(z) {return z+x}; const x=2; f(1), however the first program uses static typing while the second uses dynamic. In dynamic typing, variable assignments are stored in the environment, and earlier assignments get overwritten by later assignments. When f(1) gets called, the variable x has been assigned to 2, so f(1) evaluates to 3. In static typing, the first assignment of x substitutes the assignment value into all non-bound instances of x, so the function becomes 1+z. The second assignment to x does not change this since x no longer is inside the function, and f(1) returns 1+1=2.

3. d. The judgement form e->e’ is deterministic for the judgement forms implemented in the this lab, since our reduction system has the determinism property which means that if e->e’ and e->e’’ then e’’=e’. Essentially, there can only ever be one next step. Since no possible e in our system fits multiple rules, our system is deterministic.

4. The evaluation order of e1+e2 is left associative, meaning e1 will be evaluated before e2. The searchBinary rule explicitly tells us that any e1 bop e2 will step to e1’ bop e2, where e1->e1’. The rules could be changed to be right associative by making cases where both e1 and e2 are

not values evaluate e2 first. For bop, we change the rule to make e1 bop e2 step to e1 bop e2’ where e2->e2’.

5. a. The statement true or !(!(!(!(false)))) benefits from short circuit evaluation by evaluating directly to true because the first argument is true, saving it from wasting several cycles evaluating the value of !(!(!(!(false))))

5.b yes, e1&&e2 short circuits. Our semantics is left associative, so e1 will step until it reaches a value. If that value is false the evaluation of e2 is skipped, short circuiting and saving cycles.