CS 164 Programming Language and Compilers **Fall 2015**

Section 5: Scope and Introduction to Type

GSI: Melanie Cebula, Ben Mehne, Matt Miller

- **Announcement**
  - PA4 is out (due 10/28 at 11:59pm)
  - WA5 out Wed 10/14
- **Outline**
  1. Scope.
  2. Type.
  3. Cool Type Inference.

---

1 Scope

1. What does a call fo `f()` return in Cool?

```
x : Int <- 1;

addx(y : Int) : Int {
    x + y
};

f() : Int {
    let x <- 2 in addx(2)
};
```

   Answer:

2. What would it return if Cool were dynamically scoped?

   Answer:

2 Type

A **type** is a collection of values that have some common operations (e.g., integers and addition). We want to make sure that the run-time values used by a program correspond to the compile-time types assigned to the program's variables. This process is called **type checking**.

1. Explain in english what are the meaning of these inference rules:

$$\frac{}{\vdash 3 : \text{Int}}$$

Answer:

$$\frac{\vdash e1 : \text{Int}, \vdash e2 : \text{Int}}{\vdash e1 + e2 : \text{Int}}$$

Answer:

2. Consider the following toy language, and give a complete set of inference rules for booleans:

```
e ::= true | false                  boolean constructs
   | n                             all integers n
   | x                             all variables (x, y, z, ...)
   | not e
   | e1 = e2
   | e1 < e2
   | let x : T <- e1 in e2;
T ::= Int | Bool;
```

How to handle the types of variables? Write inference rules for the `let` expression.

Answer:

3 Cool Type Inference

1. How does the code below fail to pass type checking? Introduce changes to the code to fix this problem.

```
class A {
instance(): A {new A};
}

class B inherits A {
...
}
...(somewhere else in the code)
b:B <- (new B).instance();
```

Answer:

2. True/False SELF_TYPE is a dynamic type. **Answer:**

3. Select which of the following are needed as context for type inference in Cool?

   (a) A collection of known method signatures (mapping components of method signature to types)

   (b) The variable type environment you are in (mapping variable identifiers to types)

   (c) The body of which class you are in

   (d) A mapping of scope to all identifiers (for both methods and objects)

   Answer:

4. Why is the following rule unsound? What's one easy fix?

$$\frac{O, M, C \vdash e1 : Bool, O, M, C \vdash e2 : T2, O, M, C \vdash e3 : T3, T2 \geq T3}{O, M, C \vdash \text{if e1 then e2 else e3 : T3}}$$

   Answer:

5. Perform type inference on the let statement (near the end of the code).

```
class BinaryOp {
  do(a:Int, b:Int) : Int { {
    abort(); 0;
  }};
};

class Mult inherits BinaryOp {
  do(a:Int, b:Int): Int {
    a*b
  };
};

class Add inherits BinaryOp {
  do(a:Int, b:Int): Int {
    a+b
  };
};

....
bar(doMult: Bool): Int {
  let z:BinaryOp <- if doMult then new Mult else new Add in z.do(5*3,2)
};
```

Answer: