CS 164 Programming Languages and Compilers **Spring 2019**

Discussion Week of 4/01: Scope and Type Unification

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

1 Lexical and Dynamic Scope

Consider the following program:

```
x = 161
def add(y)
  return x + y
def f(x)
  return add(3)
```

Recall that a lexically scoped variable is looked up according to the textual arrangement of its definition. By contrast, a dynamically scoped variable is looked up in the call stack that is active at the moment when a name needs to be resolved and found in the most recent stack frame.

1. What does `f(5)` return under lexically scoped rules? What about dynamically scoped rules?
   Answer: Lexical: 164 Dynamic: 8

2 Type Unification

1. In this problem, we will perform type inference in order to determine the type of the following `foldl` program.

```
def foldl f z l = if l ≡ [] then z else foldl f (f z (head l)) (tail l)
```

   (a) Derive type constraints for the above program.

   (b) Based on the type constraint equations from the previous part, use the unification algorithm to derive the type of the entire program.

   (c) Suppose we want to count the number of `a`'s in a string. Let the string have type *List* Char. Write an appropriate function involving `foldl` that computes this quantity.

   Answer: Examine the `TypeUnification.pdf` file.

2. What are the types of the empty lists in the following expression?

```
if x = [] then [] else x::y fi
```

Answer: The type of the first empty list is *list* a and the type of the second empty list is *list* b. The variables a and b must be distinct; otherwise, the expression will not type check. This is because if the first empty list is of type *list* a, then inside the else statement, `x::y` has a type of *list* list a. Clearly list a $\neq a$.