
Reading

1. Read Section 4.2 on lambda calculus, except pages 64-66.
2. Chapter 5 on the Algol family and ML, except section 5.4.5.
3. Read Chapter 6 on types and type inference, except for section 6.5.

Problems

1. Symbolic Evaluation

The Algol-like program fragment

```
function f(x)
  return x+4
end;
function g(y)
  return 3-y
end;
f(g(1));
```

can be written as the following lambda expression:

$$\left(\underbrace{(\lambda f. \lambda g. f(g\ 1))}_{\text{main}} \underbrace{(\lambda x. x + 4)}_f \right) \underbrace{(\lambda y. 3 - y)}_g$$

Reduce the expression to a normal form in two different ways, as described below.

- (a) Reduce the expression by choosing, at each step, the reduction that eliminates a λ as far to the *left* as possible.
- (b) Reduce the expression by choosing, at each step, the reduction that eliminates a λ as far to the *right* as possible.

2. Translation into Lambda Calculus

A programmer is having difficulty debugging the following C program. In theory, on an “ideal” machine with infinite memory, this program would run forever. (In practice, this program crashes because it runs out of memory, since extra space is required every time a function call is made.)

```
int f(int (*g)(...)){ /* g points to a function that returns an int */
  return g(g);
}
int main(){
  int x;
  x = f(f);
  printf("Value of x = %d\n", x);
  return 0;
}
```

Explain the behavior of the program by translating the definition of f into lambda calculus and then reducing the application $f(f)$. This program assumes that the type checker does not check the types of arguments to functions.

3. ML Reduce for Trees

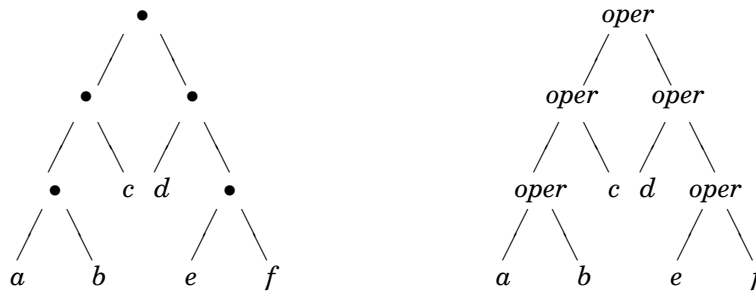
The binary tree datatype

```
datatype 'a tree = LEAF of 'a |
                  NODE of 'a tree * 'a tree;
```

describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node). Write a function

```
reduce : ('a * 'a → 'a) * 'a tree → 'a
```

that combines all the values of the leaves using the binary operation passed as a parameter. In more detail, if $oper : 'a * 'a \rightarrow 'a$ and t is the nonempty tree on the left in this picture,



then $reduce\ oper\ t$ should be the result obtained by evaluating the tree on the right. For example, if f is the function

```
fun f(x : int, y : int) = x + y;
```

then $reduce\ f\ (NODE(NODE(LEAF\ 1,\ LEAF\ 2),\ LEAF\ 3)) = (1 + 2) + 3 = 6$. Explain your definition of $reduce$ in one or two sentences.

4. ML Types

Explain the ML type for each of the following declarations:

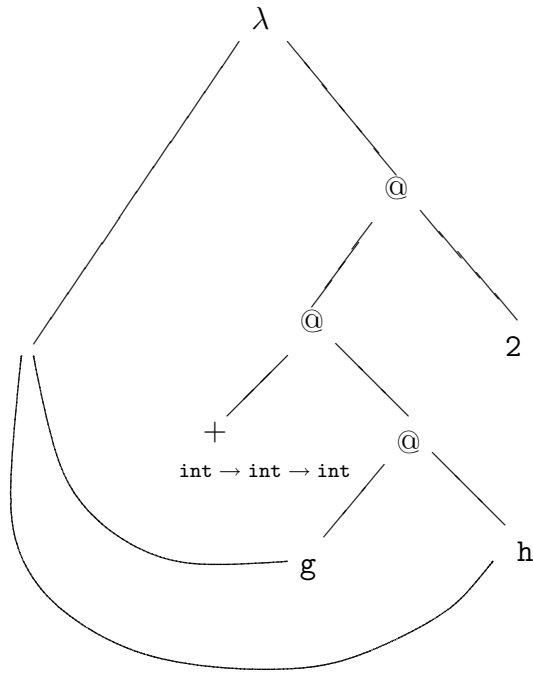
- (a) $fun\ a(x, y) = x * y - 3;$
- (b) $fun\ b(x, y) = x / 7.0 + y;$
- (c) $fun\ c(x, y, b) = if\ b(x)\ then\ x\ else\ y;$
- (d) $fun\ d(f) = fn\ y => f(y + 3);$
- (e) $fun\ e(f, g, x) = f(g(x));$

Since you can simply type these expressions into an ML compiler to determine the type, be sure to write a short *explanation* to show that you understand why the function has the type you give.

5. Parse Graph

Use the parse graph below to calculate the ML type for the function

```
fun f(g, h) = g(h) + 2;
```



6. Type Inference and Bugs

What is the type of the following ML function?

```
fun append(nil, l) = l
  | append(l, x::m) = x::append(l,m);
```

Write one or two sentences to explain succinctly and informally why `append` has the type you give. This function is intended to append one list onto another. However, it has a bug. How might knowing the type of this function help the programmer to find the bug?