6.1 ML Types

Explain the ML type for each of the following declarations:

(a) `fun a(x,y) = x+2*y;`

(b) `fun b(x,y) = x+y/2.0;`

(c) `fun c(f) = fn y => f(y);`

(d) `fun d(f,x) = f(f(x)));`

(e) `fun e(x,y,b) = if b(y) then x else y;`

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

a) $(int * int) \rightarrow int$

Because * applies to arguments with the same type and outputs that same type too, so y is int and similar reasoning is used for t.

b) $(real * real) \rightarrow real$

Like a $\qquad a \rightarrow b$

c) $\forall a, b \ (a \rightarrow b) \rightarrow a \rightarrow b$

fun c(f) : fn y => f(y) is same as $c = \lambda f.\ \lambda y.\ f(y)$

d) $(a \rightarrow a) * a \rightarrow a$ because the output of f is given as argument to input of outsider f.

e) $a * a * (a \rightarrow bool) \rightarrow a$ both branches must have the same type and b is applied to y and its output must be bool because it's given to if clause.

6.2 Polymorphic Sorting

This function performing insertion sort on a list takes as arguments a comparison function less and a list l of elements to be sorted. The code compiles and runs correctly:

```
fun sort(less, nil) = nil   |
    sort(less, a : : l) =
      let
        fun insert(a, nil) = a : : nil   |
            insert(a, b : : l) = if less(a,b) then a : : (b : : l)
                                 else b : : insert(a, l)
      in
        insert(a, sort(less, l))
      end;
```

a function that gets 2 arg and outputs a bool

from list

l → list

same type as b (list entries type)

b : : l → list

What is the type of this sort function? Explain briefly, including the type of the subsidiary function insert. You do not have to run the ML algorithm on this code; just explain why an ordinary ML programmer would expect the code to have this type.

sort : $((a * a) \rightarrow bool) * a\ list \rightarrow a\ list$

insert : $(a * a\ list) \rightarrow a\ list$

6.4 Polymorphic Fixed Point

A *fixed point* of a function f is some value x such that $x = f(x)$. There is a connection between recursion and fixed points that is illustrated by this ML definition of the factorial function factorial : int ⟶ int:

```
fun Y f x = f (Y f) x;
fun F f x = if x=0 then 1 else x*f(x-1);
val factorial = Y F;
```

The first function, Y, is a fixed-point operator. The second function, F, is a function on functions whose fixed point is factorial. Both of these are curried functions; using the ML syntax fn x ⇒ . . . for λx . . . , we could also write the function F as

```
fun F(f) = fn x =>
        if x=0 then 1 else x*f(x-1)
```

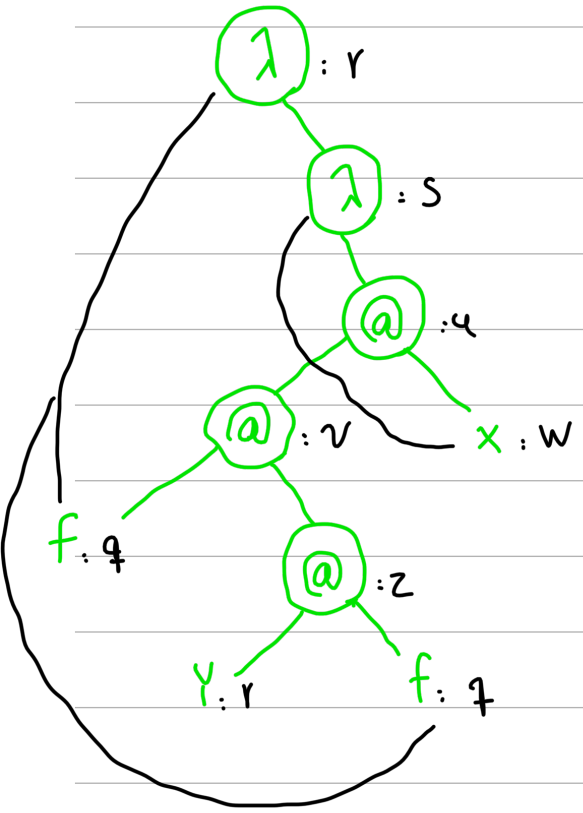
This F is a function that, when applied to argument f, returns a function that, when applied to argument x, has the value given by the expression if x=0 then 1 else x*f(x-1).

(a) What type will the ML compiler deduce for F?

(b) What type will the ML compiler deduce for Y?

a) $(int \to int) \to int \to int$

b) Using parse tree and type inference algorithm:



$r = q \to z$

$q = z \to v$

$v = w \to u$

$s = w \to u$

$r = q \to s$

$s = z$, $s = w \to u$, $q = s \to w \to u$

$= w \to u \to w \to u$

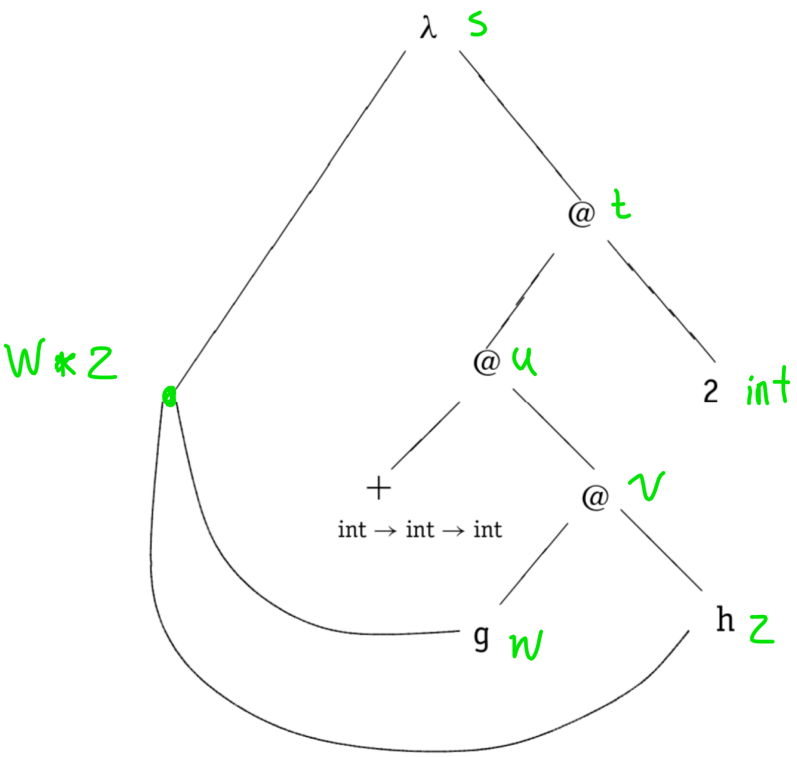
$r = q \to s$

$r = w \to u \to w \to u \to w \to u$

6.5 Parse Graph

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

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



$W = Z \to V$

$int \to int \to int = V \to U \Rightarrow t = int,\ u = int \to int,\ v = int$

$U = int \to t$

$S : W * Z \to t$

$W = Z \to int$

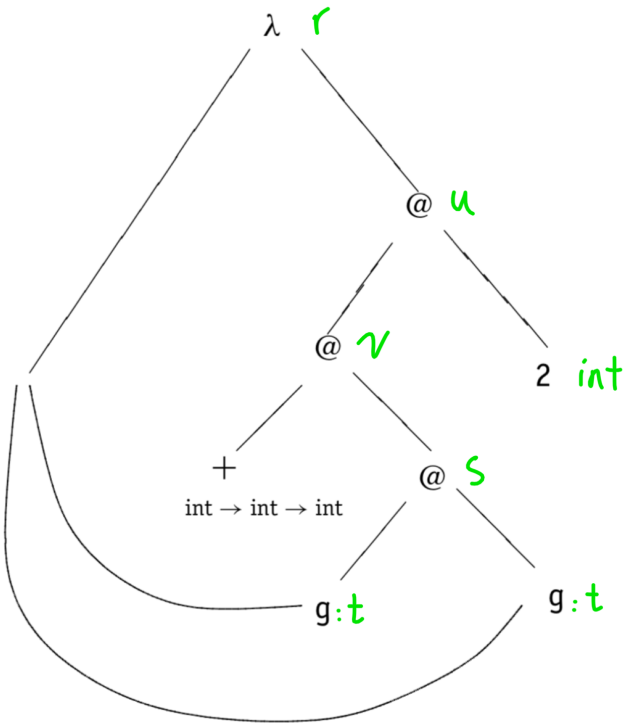
$S = (Z \to int) * Z \to int$

6.6 Parse Graph

Use the following parse graph to follow the steps of the ML type-inference algorithm on the function declaration

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

What is the output of the type checker?



$t = t \to s$

$int \to int \to int = s \to v$ $\Rightarrow$ $v = int \to int$, $u = int$, $s = int$

$v = int \to u$

$t = t \to int$ ⟶ there is a loop ⟶ No type compile error

$r = t \to u$

6.7 Type Inference and Bugs

What is the type of the following ML function?

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

First clause: a list * b → b } consistent
Second clause: c list * x → x }

The output type must be a list of some type but it's not. Type checker can help us with showing a general type for output and not necessarily a list.

6.8 Type Inference and Debugging

The reduce function takes a binary operation, in the form of a function f, and a list, and produces the result of combining all elements in the list by using the binary operation. For example;

```
reduce plus [1,2,3] = 1 + 2 + 3 = 6
```

if plus is defined by

```
fun plus (x, y : int) = x+y
```

A friend of yours is trying to learn ML and tries to write a reduce function. Here is his incorrect definition:

```
fun reduce(f, x) = x
|   reduce(f, (x : : y)) = f(x, reduce(f, y));
```

He tells you that he does not know what to return for an empty list, but this should work for a nonempty list: If the list has one element, then the first clause returns it. If the list has more than one element, then the second clause of the definition uses the function f. This sounds like a reasonable explanation, but the type checker gives you the following output:

```
val reduce = fn : ((('a * 'a list) -> 'a list) * 'a list) -> 'a list
```

How can you use this type to explain to your friend that his code is wrong?

Expected type: $((a * a) \to a) * a\ \text{list} \to a$

of

The problem is that because first clause, (reduce) always output x

6.10 Typing and Run-Time Behavior

The following ML functions have essentially identical computational behavior,

```
fun f(x) = not f(x);
fun g(y) = g(y) * 2;
```

because except for typing differences, we could replace one function with the other in any program without changing the observable behavior of the program. In more detail, suppose we turn off the ML type checker and compile a program of the form $\mathcal{P}$[fun f(x) = not f(x)]. Whatever this program does, the program $\mathcal{P}$[fun f(y) = f(y) * 2] we obtain by replacing one function definition with the other will do exactly the same thing. In particular, if the first does not lead to a run-time type error such as adding an integer to a string, neither will the second.

(a) What is the ML type for f?

(b) What is the ML type for g?

(c) Give an informal explanation of why these two functions have the same run-time behavior.

(d) Because the two functions are equivalent, it might be better to give them the same type. Why do you think the designers of the ML typing algorithm did not work harder to make it do this? Do you think they made a mistake?

a) a → bool

b) a → int

c) Both have self-referential and create an infinite loop

d) Because halting problem is undecidable