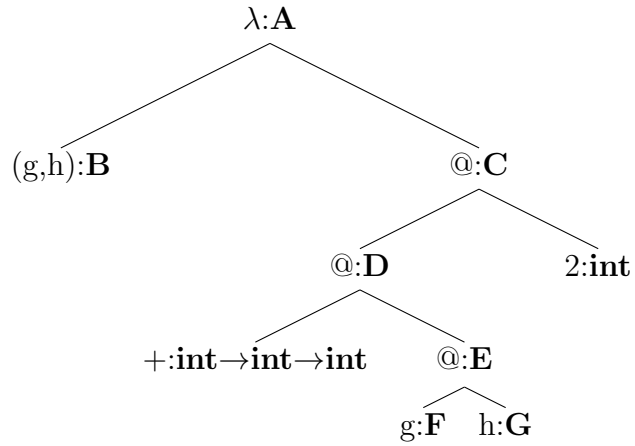
Assignment 5 (Type Inference)

Submission of assignments is optional. Submitted assignments will not be graded. If your solution differs from the posted solution, please submit it and indicate clearly why you think your solution is correct.

1. Mitchell, exercise 6.2 *Solution:* See HW4 □

2. Mitchell, exercise 6.5 *Solution:*



We generate these constraints:

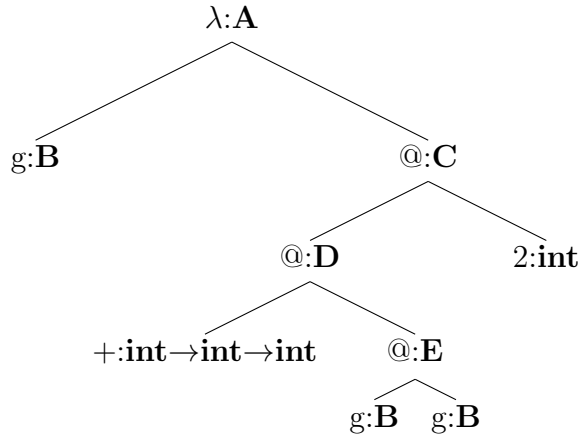
$$
\begin{aligned}
B &= F * G \\
B &= C \to A \\
D &= int \to C \\
(+) &= E \to D \\
F &= G \to E
\end{aligned}
$$

Solving these constraints yields:

$$
\begin{aligned}
E &= int \\
D &= int \to int \\
C &= int \\
F &= G \to int \\
B &= (G \to int) * G \\
A &= (G \to int) * G \to int
\end{aligned}
$$

□

3. Mitchell, exercise 6.6 *Solution:*



We generate these contraints:

$$\begin{aligned} A &= B \to C \\ D &= int \to C \\ int \to int \to int &= E \to D \\ B &= B \to E \end{aligned}$$

We cannot possibly unify the constraints for B, because of the self reference, so we get a type error. □

4. Mitchell, exercise 6.7 *Solution:* For case 1, we generate constraint:

```
'a list * 'b -> 'b
```

For case 2, we generate:

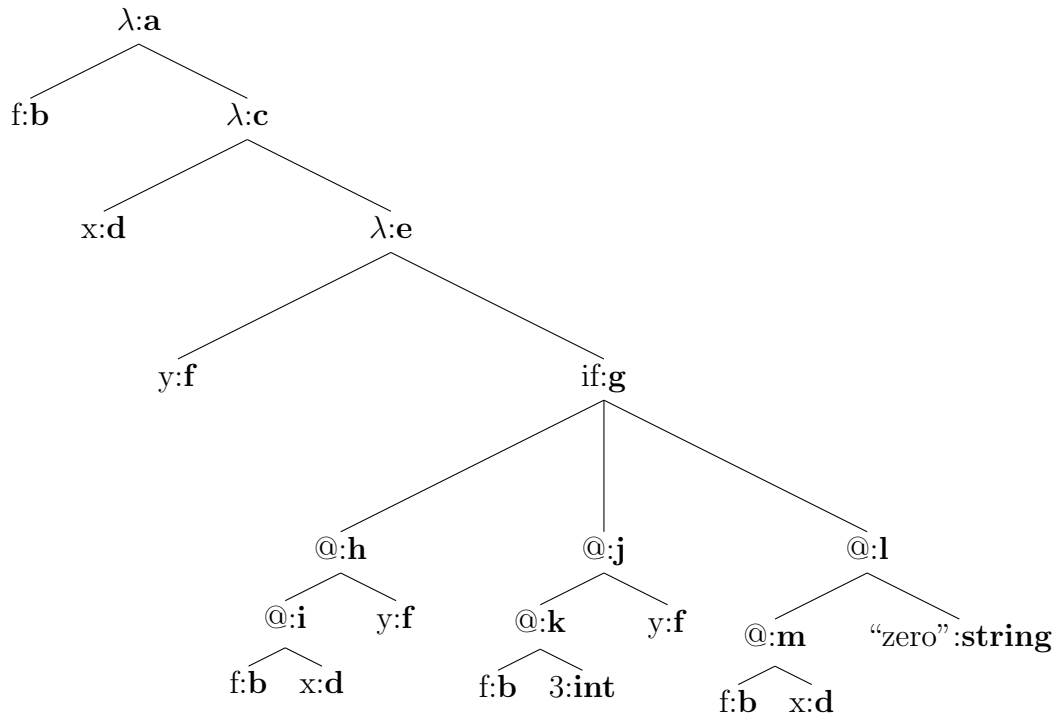
```
'c list * 'd -> 'd
```

□ To unify, we get $`b = `d$ and $`a = `c$ and so the type we get is $`a\ list * `b \to `b$. This suggests that one might be able to append a list and something that's not a list. However, we know that append should only work with two lists, and so in this sense, knowing the type of the function helps us find the bug in the function definition.

5. Derive the type of

```
fun ff f x y = if (f x y) then (f 3 y) else (f x "zero")
```

Solution:



We generate these constraints:

$$
\begin{aligned}
a &= b \to c \\
c &= d \to e \\
e &= f \to g \\
g &= j = l \\
h &= bool \\
i &= f \to h \\
b &= d \to i \\
k &= f \to j \\
b &= int \to k \\
m &= string \to l \\
b &= d \to m
\end{aligned}
$$

We solve these constraints to get:

$$
\begin{aligned}
a &= b \to (d \to (f \to g)) \\
i &= f \to bool \\
b &= d \to (f \to bool) \\
b &= int \to (f \to g) \\
b &= d \to string \to g
\end{aligned}
$$

Unifying these we get

$$
\begin{aligned}
d &= int \\
f &= string \\
g &= bool \\
b &= int \rightarrow string \rightarrow bool \\
a &= int \rightarrow (string \rightarrow bool) \rightarrow (int \rightarrow (string \rightarrow bool))
\end{aligned}
$$

Consequently, the required type of gg is:

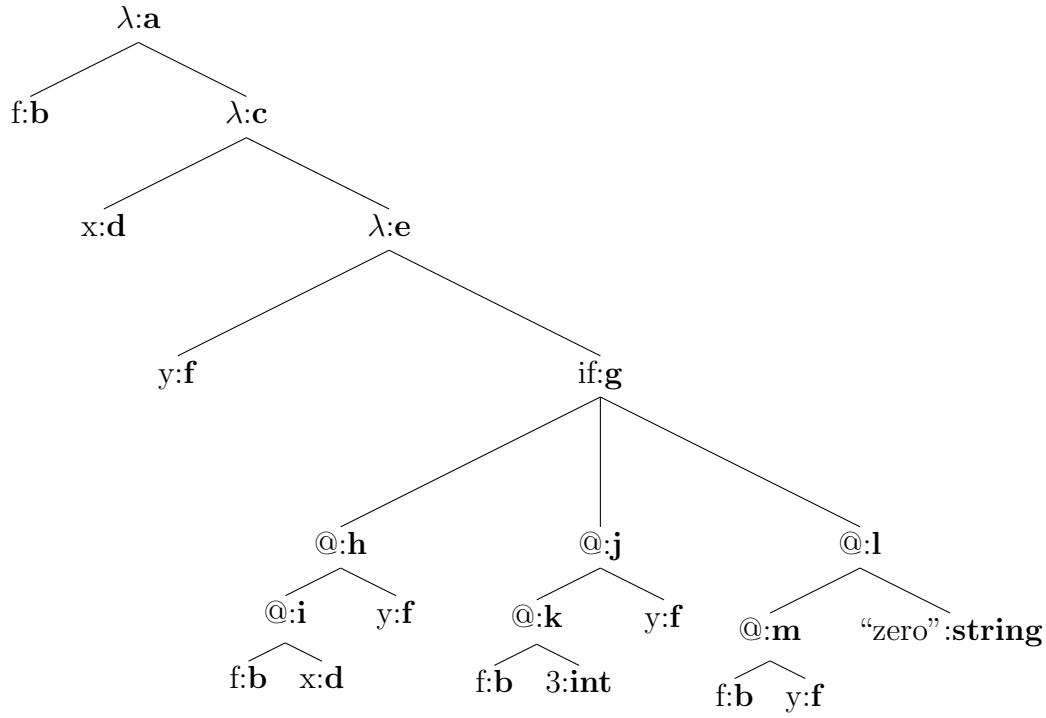
```
ff : (int -> string -> bool) -> int -> string -> bool
```

□

6. Derive the type of

```
fun gg f x y = if (f x y) then (f 3 y) else (f y "zero")
```

Solution:



We generate these constraints:

$$
\begin{aligned}
a &= b \to c \\
c &= d \to e \\
e &= f \to g \\
g &= j = l \\
h &= bool \\
i &= f \to h \\
b &= d \to i \\
k &= f \to j \\
b &= int \to k \\
m &= string \to l \\
b &= f \to m
\end{aligned}
$$

We solve these constraints to get:

$$
\begin{aligned}
a &= b \to (d \to (f \to g)) \\
i &= f \to bool \\
b &= d \to (f \to bool) \\
b &= int \to (f \to g) \\
b &= f \to string \to g
\end{aligned}
$$

Unifying these we get:

$$d = f = int \text{ But also} f = string$$

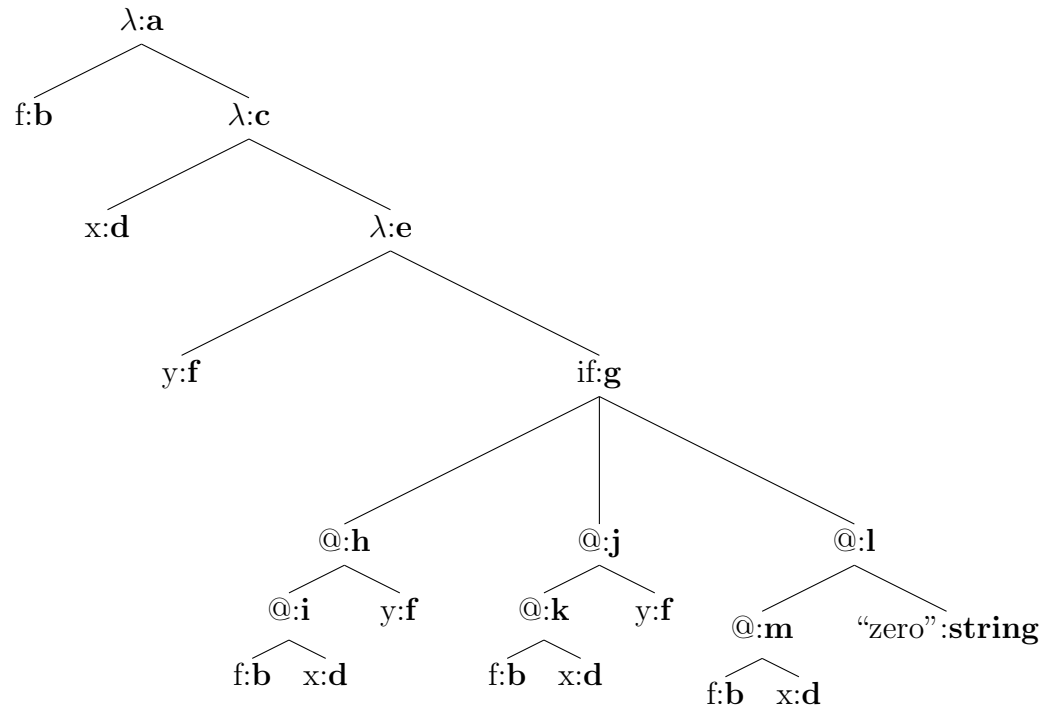
Therefore, we fail to unify these and get a type error.

□

7. Derive the type of

```
fun hh f x y = if (f x y) then (f x y) else (f x "zero")
```

Solution:



We generate these constraints:

$$
\begin{aligned}
a &= b \to c \\
c &= d \to e \\
e &= f \to g \\
g &= j = l \\
h &= \mathit{bool} \\
i &= f \to h \\
b &= d \to i \\
k &= f \to j \\
b &= d \to k \\
m &= \mathit{string} \to l \\
b &= d \to m
\end{aligned}
$$

We solve these to get:

$$
\begin{aligned}
a &= b \to (d \to (f \to g)) \\
i &= f \to \mathit{bool} \\
b &= d \to (f \to \mathit{bool}) \\
b &= d \to (f \to g) \\
b &= d \to \mathit{string} \to g
\end{aligned}
$$

Unifying these we get:

$$
\begin{aligned}
f &= string \\
g &= bool \\
b &= d \to string \to bool \\
a &= d \to (string \to bool) \to (d \to (string \to bool))
\end{aligned}
$$

Consequently, the required type of hh is :

```
hh : (d -> string -> bool) -> (d -> (string -> bool))
```

□