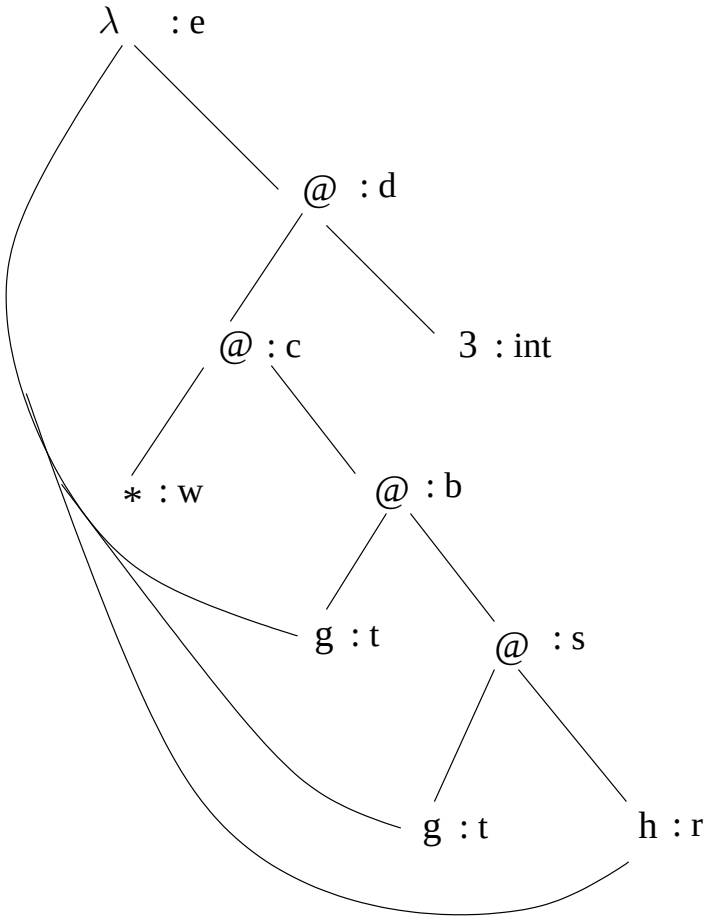
1 ML

1.1 Exercise 1

Infer the type of the following function:

```
fun f(g,h) = g(g(h)) * 3;
```

Complete the parse tree by following the steps of the ML type-inference algorithm, explicitely describing the 3 steps of the algorithm.



1.2 Exercise 2

Higher-order programming is one of the remarkable features of ML. Some of ML's higher-order functions operating over lists are `map`, `foldr` and `foldl`, which are defined as follows:

```
fun map f []      = []
  | map f (x::xs) = (f x) :: map f xs;

fun foldr f e []      = e
  | foldr f e (x::xs) = f (x, foldr f e xs);

fun foldl f e []      = e
  | foldl f e (x::xs) = foldl f (f(x, e)) xs;
```

1. Using higher-order programming, define a function `firstElems` with type

   ```
   val firstElems = fn : 'a list list -> 'a list
   ```

   such that when applied to a list of lists of integers, it gives a list containing only the first elements of each sublist. For example, if `a` is defined to be the following list:

   ```
   val a = [[2,5,76,8], [3,23,45], [5,27,1,43], [34,2,56]];
   ```

   then the result of applying firstElems to `a` will be:

   ```
   - firstElems a;
    > val it = [2,3,5,34] : int list
   ```

   (Hints: Use the `map` function. Remember that lists have an operation `hd` to obtain the first element of a list, and `tl` to obtain the tail of a list.)

2. Using higher-order programming, define a function `prodFirstElems`

   ```
   val prodFirstElems = fn : int list list -> int
   ```

such that when applied to a list of lists of integers, it gives the product of the first elements of each sublist. For example, if `a` is defined as in item 1. above, then:

```
- prodFirstElems a;
 > val it = 1020 : int
```

since $2 * 3 * 5 * 34 = 1020$.

(Hints: Use `foldr` or `foldl`. Remember that an infix operator may be transformed into prefix form by using the keyword `op`; for example, instead of writing `3*5`, you may write `op*(3,5)`.)

1.3 Exercise 3

We say that a function (program) $f_1 : int * real \rightarrow real$ is defined for given arguments $x : int$ and $y : real$ if there is a value $z : real$, such that $f_1(x, y) = z$; otherwise we say that the function is *undefined* for the given arguments. We can define the following three undefined values:

- "uncaught exception": this happens when an exception is raised without being handled;
- nan: this is a result given by ML when you perform "0.0/0.0", for instance;
- inf: this is the result given by division by zero ("1.0/0.0"), for instance.

We say that two functions $f_1 : int*real \rightarrow real$ and $f_2 : int*real \rightarrow real$ are equivalent if and only if (a) when one of the functions is defined the other one is also defined and they give the same result (i.e., $f_1(x, y) = f_2(x, y)$), and (b) both are undefined for the same values of the arguments.

Let `f` be the following ML function:

```
exception OddNum;
fun f(0,count) = count
     |  f(1,count) = raise OddNum
     |  f(x,count) = f(x-2, count+1.0) handle OddNum => ~1.0;
```

Write an ML function `eqTof` equivalent to `f`, without using exceptions.

2 Prolog

2.1 Exercise 1

Let the following be a partial family database:

```
father(jon,mikael).
father(jon,juan).
father(jon,per).
father(mikael,carlos).
father(per,karl).
father(per,anne).
father(per,sofia).

male(jon).
male(mikael).
male(juan).
male(per).
male(carlos).
male(karl).

female(sarah).
female(anne).
female(sofia).
```

1. Add 4 facts defining a `mother` relation between the three female persons and other persons (chosen by you) in the above database, (e.g. `mother(sarah,carlos)`).

2. Add rules for defining the following relationships: `son`, `daughter`, `uncle`, `brother` and `parent`. Some rules might use (some of) the facts `male`, `female`, `mother` and `father`, as well as some of the rules you are defining here.

3. Write two rules `atleastonebrother` and `unclefemale` in order to be able to ask queries for getting the following information:

   (a) All the persons who have at least one brother (use `atleastonebrother`)

   (b) All the persons who are uncle of a female (use `unclefemale`).

2.2 Exercise 2

Natural numbers may be defined as follows in Prolog:

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

The first fact asserts that "0" is a natural number while the second one says that if X is a natural number, then the s(X) is also a natural number.

The sum and product of two natural numbers may be defined as follows:

```
plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

prod(0,X,0) :- natural_number(0).
prod(s(X),Y,Z) :- prod(X,Y,XY), plus(XY,Y,Z).
```

Write a Prolog program `exp(M,X,Y)` which computes the mathematical exponentiation: $X^M = Y$. For example, writing the query

```
exp(s(s(0)), s(s(s(0))), Y).
```

(which represents $3^2 = Y$), it would produce the answer

```
Y = s(s(s(s(s(s(s(s(s(0)))))))))
```

(which represents $Y = 9$).