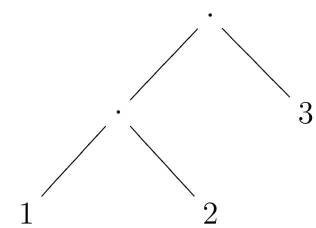
First name	Last name	Matriculation number

Exercise 1 (2+2+2 points)

The following data structure represents binary trees only containing values at the leaves:
`data Tree a = Node (Tree a) (Tree a) | Leaf a`

Consider the tree `t` of integers on the right-hand side. The representation of `t` as an object of type `Tree Int` in Haskell would be:

`Node (Node (Leaf 1) (Leaf 2)) (Leaf 3)`



Implement the following functions in Haskell.

(a) The function `foldTree` of type `(a -> a -> a) -> (b -> a) -> Tree b -> a` works as follows: `foldTree n l t` replaces all occurrences of the constructor `Node` in the tree `t` by `n` and it replaces all occurrences of the constructor `Leaf` in `t` by `l`. So for the tree `t` above, `foldTree (+) id t` should compute `(+) ((+) (id 1) (id 2)) (id 3)` which finally results in `6`. Here, `Node` is replaced by `(+)` and `Leaf` is replaced by `id`.

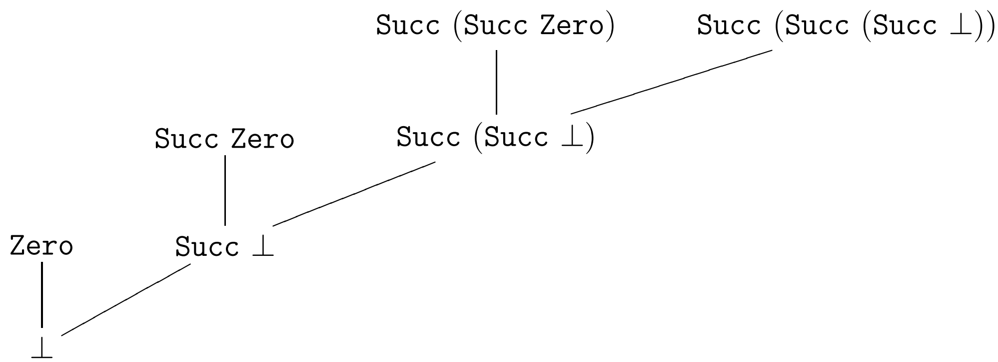
(b) Use the `foldTree` function from (a) to implement the `maxTree` function which returns the largest (w.r.t. `>`) element of the tree. Apart from the function declaration, also give the most general type declaration for `maxTree`.

First name	Last name	Matriculation number

(c) Consider the following data type declaration for natural numbers:

```
data Nats = Zero | Succ Nats
```

A graphical representation of the first four levels of the domain for `Nats` could look like this:



Sketch a graphical representation of the first three levels of the domain $D_{\texttt{Tree Bool}}$ for the data type `Tree Bool`.

First name	Last name	Matriculation number

Exercise 2 (2+3 points)

Consider the following Haskell declarations for the `double` function:

```
double :: Int -> Int
double (x+1) = 2 + (double x)
double _ = 0
```

(a) Give the Haskell declarations for the higher-order function `f_double` corresponding to `double`, i.e., the higher-order function `f_double` such that the least fixpoint of `f_double` is `double`. In addition to the function declaration(s), also give the type declaration of `f_double`. Since you may use full Haskell for `f_double`, you do not need to translate `double` into simple Haskell.

(b) We add the Haskell declaration `bot = bot`. For each $n \in \mathbb{N}$ determine which function is computed by $\texttt{f\_double}^n$ `bot`. Here "$\texttt{f\_double}^n$ `bot`" represents the n-fold application of `f_double` to `bot`, i.e., it is short for $\underbrace{\texttt{f\_double (f\_double} \ldots \texttt{(f\_double}}_{n \text{ times}}\ \texttt{bot)}\ldots)$. Give the function in closed form, i.e., using a non-recursive definition.

First name	Last name	Matriculation number

Exercise 3 (3+3 points)

Let $\sqsubseteq$ be a complete order and let f be a function which is continuous (and, therefore, also monotonic).

Prove or disprove the following statements:

(a) $\{\ f^n(\bot)\ \mid\ n \in \{0, 1, 2, \ldots\}\ \}$ is a chain.

(b) $\sqcup\{\ f^n(\bot)\ \mid\ n \in \{0, 1, 2, \ldots\}\ \}$ is a fixpoint of f.

First name	Last name	Matriculation number

Exercise 4 (3 points)

We define the following algebraic data type for lists:

```
data List a = Nil | Cons a (List a)
```

Write a program in simple Haskell which computes the function `sum :: List Int -> Int`. Here, `sum` adds all integers in a list of integers. For example, `sum (Cons 1 (Cons (-2) Nil))` should return `-1`.

Your solution should use the functions defined in the transformation from the lecture such as $\texttt{sel}_{n,i}$, $\texttt{isa}_{\underline{\texttt{constr}}}$, and $\texttt{argof}_{\underline{\texttt{constr}}}$. You do not have to use the transformation rules from the lecture, though.

First name	Last name	Matriculation number

Exercise 5 (2+3 points)

Consider the following data structure for natural numbers:

```
data Nats = Succ Nats | Zero
```

Let δ be the set of rules from Definition 3.3.5, i.e., δ contains at least the following rules:

$$\begin{aligned} \texttt{fix} &\rightarrow \lambda f.\ f\ (\texttt{fix}\ f) \\ \texttt{if False} &\rightarrow \lambda x\ y.\ y \\ \texttt{isa}_{\texttt{Zero}}\ (\texttt{Succ}\ (\texttt{Succ Zero})) &\rightarrow \mathit{False} \end{aligned}$$

(a) Please translate the following Haskell-expression into a lambda term using $\mathcal{L}am$. It suffices to give the result of the transformation.

```
let g = \x -> if (isa_Zero x) then Zero else Succ (g (argof_Succ x))
    in g (Succ (Succ Zero))
```

(b) Reduce the lambda term from (a) by WHNO-reduction with the $\rightarrow_{\beta\delta}$-relation. You do not have to give the intermediate steps but only the **weak head normal form** (which is not the same as the normal form).

First name	Last name	Matriculation number

Exercise 6 (4 points)

Use the type inference algorithm $\mathcal{W}$ to determine the most general type of the following λ-term under the initial type assumption A_0. Show the results of all sub-computations and unifications, too. If the term is not well typed, show how and why the $\mathcal{W}$-algorithm detects this.

$$\texttt{fix}\ (\lambda x.\ \texttt{Succ}\ x)$$

In this exercise, please use the initial type assumption A_0 as presented in the lecture. This type assumption contains at least the following:

$$\begin{array}{lcl} A_0(\texttt{Succ}) & = & \texttt{Nats} \to \texttt{Nats} \\ A_0(\texttt{fix}) & = & \forall a.\ (a \to a) \to a \end{array}$$