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 \( \text{Tree Int} \) in Haskell would be:

\[
\text{Node (Node (Leaf 1) (Leaf 2)) (Leaf 3)}
\]

Implement the following functions in Haskell.

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

\[
\text{foldTree} \, f \, g \, (\text{Leaf} \, x) = g \, x\\
\text{foldTree} \, f \, g \, (\text{Node} \, l \, r) = f \, (\text{foldTree} \, f \, g \, l) \, (\text{foldTree} \, f \, g \, r)
\]

(b) Use the \( \text{foldTree} \) function from (a) to implement the \( \text{maxTree} \) 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 \( \text{maxTree} \).

\[
\text{maxTree} \, :: \, \text{Ord a} \Rightarrow \text{Tree a} \to a\\
\text{maxTree} \, = \, \text{foldTree} \, \text{max} \, \text{id}
\]
(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:

```
Succ (Succ Zero)  Succ (Succ (Succ ⊥))

Succ Zero  Succ (Succ ⊥)

Zero  Succ ⊥

⊥
```

Sketch a graphical representation of the first three levels of the domain $D_{Tree Bool}$ for the data type `Tree Bool`.
Exercise 2 (2+3 points)

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

```haskell
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.

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

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

\[
(f\_\text{double}^n)(\bot)(x) = \begin{cases} 
2 \cdot x, & \text{if } 0 < x < n \\
0, & \text{if } x \leq 0 \land n > 0 \\
\bot, & \text{if } n = 0 \lor x = \bot \lor x \geq n
\end{cases}
\]
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.

We must prove $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$ for all $n \in \{0, 1, 2, \ldots\}$.

- $n = 0$: By definition we have $\bot \sqsubseteq f(\bot)$

- $n \to n + 1$: The function $f$ is continuous and therefore also monotonic. Thus, $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$ implies $f^{n+1}(\bot) \sqsubseteq f^{n+2}(\bot)$.

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

$$
\begin{align*}
f(\sqcup \{ f^n(\bot) \mid n \in \{0, 1, 2, \ldots\} \}) & \overset{f \text{ continuous}}{=} \sqcup f(\{ f^n(\bot) \mid n \in \{0, 1, 2, \ldots\} \}) \\
& = \sqcup \{ f^{n+1}(\bot) \mid n \in \{0, 1, 2, \ldots\} \} \\
& = \sqcup \{ f^n(\bot) \mid n \in \{1, 2, \ldots\} \} \\
& = \sqcup (\{ f^n(\bot) \mid n \in \{1, 2, \ldots\} \} \cup \{ \bot \}) \\
& = \sqcup (\{ f^n(\bot) \mid n \in \{1, 2, \ldots\} \} \cup f^0(\bot)) \\
& = \sqcup \{ f^n(\bot) \mid n \in \{0, 1, 2, \ldots\} \}
\end{align*}
$$
Exercise 4 (3 points)

We define the following algebraic data type for lists:

\[
\text{data List } a = \text{Nil} \mid \text{Cons } a (\text{List } a)
\]

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

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

\[
\text{let sum } = \lambda l \rightarrow \text{if (isa}_1 l) \text{ then } 0 \text{ else (sel}_{2,1} (\text{argof}_{\text{Cons}} l)) + (\text{sum} (\text{sel}_{2,2} (\text{argof}_{\text{Cons}} l)))
\]
Exercise 5 (2+3 points)

Consider the following data structure for natural numbers:

\[
data \text{Nats} = \text{Succ Nats} \mid \text{Zero}
\]

Let \( \delta \) be the set of rules from Definition 3.3.5, i.e., \( \delta \) contains at least the following rules:

\[
\begin{align*}
\text{fix} & \rightarrow \lambda f. f (\text{fix } f) \\
\text{if False} & \rightarrow \lambda x \, y. \, y \\
\text{isa}_{\text{zero}} (\text{Succ (Succ Zero)}) & \rightarrow \text{False}
\end{align*}
\]

(a) Please translate the following Haskell-expression into a lambda term using Lam. It suffices to give the result of the transformation.

\[
\begin{align*}
\text{let } g &= \lambda x \rightarrow \text{if (isa}_{\text{Zero}} x) \text{ then Zero else Succ (g (argof}_{\text{Succ}} x)) \\
in g \, (\text{Succ (Succ Zero))}
\end{align*}
\]

\[
(fix (\lambda g \, x. \text{if (isa}_{\text{Zero}} x) \text{ Zero (Succ (g (argof}_{\text{Succ}} x)))))) (\text{Succ (Succ Zero))}
\]

(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).

Let \( A = \lambda g \, x. \text{if (isa}_{\text{Zero}} x) \text{ Zero (Succ (g (argof}_{\text{Succ}} x))))

\[
\begin{align*}
\text{fix (} \lambda g \, x. \text{if (isa}_{\text{Zero}} x) \text{ Zero (Succ (g (argof}_{\text{Succ}} x)))) & (\text{Succ (Succ Zero))} \\
& = \text{fix } A \, (\text{Succ (Succ Zero))} \\
& \rightarrow_{\delta} (\lambda f. f (\text{fix } f)) \, A \, (\text{Succ (Succ Zero))} \\
& \rightarrow_{\beta} (\lambda x. \text{if (isa}_{\text{Zero}} x) \text{ Zero (Succ (fix A (argof}_{\text{Succ}} x)))))) (\text{Succ (Succ Zero))} \\
& \rightarrow_{\beta} (\text{if (isa}_{\text{Zero}} \text{ Succ (Succ Zero))}) \text{ Zero (Succ (fix A (argof}_{\text{Succ}} \text{ Succ (Succ Zero)))))) \\
& \rightarrow_{\delta} (\lambda x \, y. \text{Zero (Succ (fix A (argof}_{\text{Succ}} \text{ Succ (Succ Zero)))))) \\
& \rightarrow_{\delta} (\lambda y. \text{Succ (fix A (argof}_{\text{Succ}} \text{ Succ (Succ Zero)))))) \\
& \rightarrow_{\beta} (\text{Succ (fix A (argof}_{\text{Succ}} \text{ Succ (Succ Zero))))})
\end{align*}
\]
Exercise 6 (4 points)

Use the type inference algorithm $W$ 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 $W$-algorithm detects this.

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

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

\[
\begin{align*}
A_0(\text{Succ}) &= \text{Nats} \rightarrow \text{Nats} \\
A_0(\text{fix}) &= \forall a. (a \rightarrow a) \rightarrow a
\end{align*}
\]

\[
W(A_0, \text{fix}(\lambda x. \text{Succ } x))
\]

\[
W(A_0, \text{fix}) = (id, (b_0 \rightarrow b_0) \rightarrow b_0)
\]

\[
W(A_0, \lambda x. \text{Succ } x)
\]

\[
W(A_0 + \{x : b_1\}, \text{Succ } x)
\]

\[
W(A_0 + \{x : b_1\}, \text{Succ}) = (id, \text{Nats} \rightarrow \text{Nats})
\]

\[
W(A_0 + \{x : b_1\}, x)
\]

\[
mgu((\text{Nats} \rightarrow \text{Nats}), (b_1 \rightarrow b_0)) = [b_1/\text{Nats}, b_2/\text{Nats}]
\]

\[
= ([b_1/\text{Nats}, b_2/\text{Nats}], \text{Nats})
\]

\[
mgu(((b_0 \rightarrow b_0) \rightarrow b_0), ((\text{Nats} \rightarrow \text{Nats}) \rightarrow b_3)) = [b_0/\text{Nats}, b_3/\text{Nats}]
\]

\[
= ([b_1/\text{Nats}, b_2/\text{Nats}, b_0/\text{Nats}, b_3/\text{Nats}], \text{Nats})
\]

Resulting type: Nats