Exercise 1 (Quiz): 

Give a short proof sketch or a counterexample for each of the following statements:

a) Monotonic unary functions are always strict.

b) Strict unary functions on flat domains are always monotonic.

c) Let \( \mathbb{B} \) be the Boolean values true, false.

Is \( f : (\mathbb{B} \to \mathbb{B} \perp) \to \mathbb{Z} \) with \( f(g) = \begin{cases} 1 & \text{if } g(x) \neq \text{true} \text{ for all } x \in \mathbb{B} \\ 0 & \text{otherwise} \end{cases} \) monotonic?

d) Is \( \rightarrow_\alpha \) terminating?

e) Is \( \rightarrow_\alpha \) confluent?

Solution:

a) No. Consider \( f(x) = 0 \).

b) Yes. Let \( f : D \to D' \) be some strict function, \( D \) flat, \( d, d' \in D \) and \( d \subseteq d' \). As the domain is flat, we have two cases:

- \( d = d' \), which directly implies \( f(d) = f(d') \).
- \( d = \bot \), which due to \( f \)'s strictness implies \( f(d) = \bot \subseteq f(d') \).

c) No. Consider \( g_1(x) = \bot \) and \( g_2(x) = \begin{cases} \text{true} & \text{if } x = \text{true} \\ \bot & \text{otherwise} \end{cases} \). We have \( g_1 \subseteq g_2 \), but \( f(g_1) = 1 \not\subseteq 0 = f(g_2) \).

d) No. Consider the derivation \( \lambda x.x \to_\alpha \lambda y.y \to_\alpha \lambda x.x \) as counterexample.

e) Yes. Let \( t \) be some \( \lambda \)-term and \( t \to^*_\alpha q, t \to^*_\alpha p \) hold. Then, as \( q \) and \( p \) are just variable-renamed versions of \( t \), we can directly conclude that \( q \to^*_\alpha p \) and \( p \to^*_\alpha q \) holds.

Exercise 2 (Programming in Haskell): 

We define a polymorphic data structure \texttt{ZombieHalls} to represent a zombie-infested school whose classrooms contain different types of food:

```haskell
data ZombieHalls food =
    HallwayFork (ZombieHalls food) (ZombieHalls food)
  | HallwayClassroom (Int, food) (ZombieHalls food)
  | HallwayEnd
```

Here, we use three data constructors: One representing the case that the hallway forks and we can go in two directions, one for the case that we have a classroom on one side and can continue in the hallway and finally one case for the end of a hallway. The data structure \texttt{ZombieFood} is used to represent food for zombies. As example, consider the following definition of \texttt{exampleSchool} of type \texttt{ZombieLabyrinth ZombieFood}, corresponding to the illustration on the right:
data ZombieFood = Brains | Nuts deriving Show

eexampleSchool :: ZombieHalls ZombieFood
exampleSchool =
    HallwayClassroom (3, Nuts)
    (HallwayFork
        (HallwayClassroom (4, Brains)
            (HallwayFork HallwayEnd HallwayEnd))
        (HallwayClassroom (0, Brains) HallwayEnd))

a) Implement a function
   buildSchool :: Int -> ZombieHalls ZombieFood
such that for any integer
   number
   n ≥ 0, it returns a structure of hallways containing 2^n+1 classrooms in total. Of these, one half
   should each contain one brain and the other should each contain one nut.

b) Implement a fold function
   foldZombieHalls, including its type declaration, for the data structure
   ZombieHalls. As usual, the fold function replaces the data constructors in a ZombieHalls expres-
   sion by functions specified by the user. The first argument of foldZombieHalls should be the function
   for the case of a HallwayFork, the second argument should replace the HallwayClassroom construc-
   tor and the third argument should replace the HallwayEnd data constructor. As an example, con-
   sider the following function definition, which uses foldZombieHalls to determine the number of dead
   ends in a ZombieHalls structure, where a classroom does not count as dead end. Hence, the call
   numberOfDeadEnds exampleSchool returns 3.

   numberOfDeadEnds :: ZombieHalls food -> Int
   numberOfDeadEnds school = foldZombieHalls (+) (\_ r -> r) 1 school

c) Implement the function
   bcCounter :: ZombieHalls ZombieFood -> (Int, Int)
   which counts the
   number of brains and classrooms in a given school and returns the two numbers as a tuple of integers.
   The first part of the tuple should be the number of brains in the school and the second should be
   the number of classrooms. For the definition of bcCounter, use only one defining equation where the
   right-hand side is just one call to the function foldZombieHalls. However, you may use and define
   non-recursive auxiliary functions.

   For example, a call bcCounter exampleSchool should return the tuple (4, 3).

d) The infinite sequence of Fibonacci numbers fib_i is defined as fib_0 = 0, fib_1 = 1 and fib_i = fib_{i−1} + fib_{i−2}
   for all i > 1. The first elements of the sequence are 0, 1, 2, 3, 5, 8, 13, 21, ....

   Implement a cyclic data structure
   fibs :: [Int]
   that represents the infinite list of Fibonacci numbers.
   Do not use self-defined auxiliary functions and ensure that take n fibs has linear complexity.

Hints:
   • You should use use the function
     zipWith :: (a -> b -> c) -> [a] -> [b] -> [c], which ap-
    plies the function given as its first argument to combine the elements of two lists. For example
     zipWith (+) ["a","b"] ["c", "d", "e"] results in the list ["ac","bd"]. Note that the length
     of the resulting list is the smallest length of both input lists.
   • You may use the pre-defined function
     tail defined as tail (x:xs) = xs.

e) Write a function
   splits :: [a] -> [(a],[a])
   that computes all splits of a finite input list, i.e., a call
   splits xs should return all pairs (ys,zs) such that ys ++ zs is again xs. For example, we have
   splits "abc" = [([""],["abc"]),("a","bc"),("ab","c"),("abc","")].

   The right-hand side of your function should be just a list comprehension.

Hints:
   • Use length :: [a] -> Int, which returns the length of a given list.
   • Use take :: Int -> [a] -> [a], where take n xs yields the longest prefix of xs with length ≤ n.
   • Use drop :: Int -> [a] -> [a], where drop n xs returns the list obtained from xs by removing
     the first n elements.
Solution:

a) buildSchool :: Int -> ZombieHalls ZombieFood
   buildSchool n | n == 0 = HallwayClassroom (1, Nuts) (HallwayClassroom (1, Brains) HallwayEnd)
   | n > 0 = HallwayFork otherHall otherHall
   where otherHall = buildSchool (n-1)

b) foldZombieHalls :: (result -> result -> result)
   -> ((Int, food) -> result -> result)
   -> result
   -> ZombieHalls food
   -> result
   foldZombieHalls fHF fHC fHE = go
   where
     go (HallwayFork l r) = fHF (go l) (go r)
     go (HallwayClassroom c h) = fHC c (go h)
     go HallwayEnd = fHE

c) bcCounter :: ZombieHalls ZombieFood -> (Int, Int)
   bcCounter = foldZombieHalls (\(rB, rN) (cB, cN) -> (rB+cB, rN+cN)) cHelper (0,0)
   where
     cHelper (n, Brains) (rB, rN) = (rB+n, rN+1)
     cHelper (n, _) (rB, rN) = (rB, rN+1)

d) fibs :: [Int]
   fibs = 0:1:(zipWith (+) fibs (tail fibs))

e) splits :: [a] -> [(a[a], [a])]
   splits xs = [ (take i xs, drop i xs) | i <- [0 .. length xs] ]

Exercise 3 (Semantics): (22 + 10 + 5 + 4 = 41 points)

a)  i) Let \( \subseteq_{D_1} \) and \( \subseteq_{D_2} \) be complete partial orders on \( D_1 \) resp. \( D_2 \) and \( f : D_1 \rightarrow D_2 \) a function. Prove that \( f \) is continuous if and only if \( f \) is monotonic and for all chains \( S \) in \( D_1 \), \( f(\bigcup S) \subseteq_{D_2} \bigcup f(S) \) holds.

ii) Let \( D = \mathbb{N} \rightarrow \{1\} \), i.e., \( D \) is the set of all functions mapping the natural numbers to \( \bot \) or 1. Let \( \subseteq \) be defined as usual on functions.
   1) Prove that every chain \( S \subseteq D \) has a least upper bound w.r.t. the relation \( \subseteq \).
   2) Prove that \( \subseteq \) is a cpo on \( D \).
   3) Give an example for an infinite chain in \( (D, \subseteq) \).
   4) Give a monotonic, non-continuous function \( f : D \rightarrow D \). You do not need to prove that \( f \) has these properties.

b) i) Consider the following Haskell function \( \text{exp} \):

   \[
   \text{exp} :: (\text{Int}, \text{Int}) \rightarrow \text{Int} \\
   \text{exp} (x, 0) = 1 \\
   \text{exp} (x, y) = x \ast \text{exp} (x, y - 1)
   \]

   Please give the Haskell declaration for the higher-order function \( \text{f_exp} \) corresponding to \( \text{exp} \), i.e., the higher-order function \( \text{f_exp} \) such that the least fixpoint of \( \text{f_exp} \) is \( \text{exp} \). In addition to the function declaration, please also give the type declaration of \( \text{f_exp} \). You may use full Haskell for \( \text{f_exp} \).
ii) Let $\phi_{\text{f\_exp}}$ be the semantics of the function $f_{\text{exp}}$. Give the semantics of $\phi_{\text{f\_exp}}^n(\bot)$ for $n \in \mathbb{N}$, i.e., the semantics of the $n$-fold application of $\phi_{\text{f\_exp}}$ to $\bot$.

iii) Give the least fixpoint of $\phi_{\text{f\_exp}}$.

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

\[
\text{data Nats} = \text{Z} \mid \text{S Nats}
\]

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

\[
\begin{array}{ccc}
\text{S (S Z)} & \text{S (S (S \bot))} \\
\text{S Z} & \text{S (S \bot)} \\
\text{Z} & \text{S \bot} \\
\bot
\end{array}
\]

Now consider the following data type declarations:

\[
\begin{array}{l}
\text{data U = V} \\
\text{data T a = C | D (T a) | E a a}
\end{array}
\]

Give a graphical representation of the first three levels of the domain for the type $T \ U$. The third level contains the element $D \ C$, for example.

d) Consider the usual definitions for List a, i.e., data List a = Nil | Cons a (List a) and Nats from above.

Write a function $\text{length} :: \text{List a} \rightarrow \text{Nats}$ in Simple Haskell that computes the length of a list, i.e., $\text{length} \ (\text{Cons Z (Cons Z Nil)})$ should yield $\text{S(S(Z))}$. Your solution should use the functions defined in the transformation from the lecture such as $\text{sel}_{n,i}$, $\text{isa}_{\text{constr}}$, $\text{argof}_{\text{constr}}$, and $\text{bot}$. You do not have to use the transformation rules from the lecture, though.

Solution:

a) i) First, let $f$ be continuous. Then, for any chain $S$, we have $f(\sqcup S) = \sqcup f(S)$. Because $\sqsubseteq_{D_2}$ is reflexive, $f(\sqcup S) \sqsubseteq_{D_2} \sqcup f(S)$ follows. To prove monotonicity of $f$, let $d, d' \in D_1$ with $d \sqsubseteq_{D_1} d'$. Then $f(\sqcup \{d, d'\}) = f(d')$. Since $f$ is continuous, we also have $f(\sqcup \{d, d'\}) = \sqcup \{f(d), f(d')\}$. Consequently, we have $f(d) \sqsubseteq f(d')$.

Now, assume $f$ is monotonic and $f(\sqcup S) \sqsubseteq_{D_2} \sqcup f(S)$ holds. As $\sqsubseteq_{D_2}$ is antisymmetric, it suffices to prove $\sqcup f(S) \sqsubseteq_{D_2} f(\sqcup S)$, i.e., that for all $d \in S$, we have $f(d) \sqsubseteq_{D_2} f(\sqcup S)$. Obviously, $d \sqsubseteq_{D_1} \sqcup S$ holds. As $f$ is monotonic, $f(d) \sqsubseteq f(\sqcup S)$ follows and therefore, $\sqcup f(S) \sqsubseteq_{D_2} f(\sqcup S)$ holds.
ii) Let $S = \{ f_1, f_2, \ldots \}$ with $f_i \sqsubseteq f_{i+1}$ be a chain.

1) Let $M_i = \{ x \in \mathbb{N} \mid f_i(x) \neq \bot \}$. Then, by definition of $\sqsubseteq$, $M_i \sqsubseteq M_{i+1}$. Let $M = \bigcup M_i$. We define

$$\sqcup S = f \text{ with } f(x) = \begin{cases} 1 & \text{if } x \in M \\ \bot & \text{otherwise} \end{cases}.$$ 

First, we prove that $f$ is an upper bound. Assume $f_i \sqsubseteq f$ does not hold. Then, there is some $n \in \mathbb{N}$ with $f_i(n) = 1$ and $f(n) = \bot$. But then, we also have $n \in M_i$ and hence $n \in M$, so $f(n) = f_i(n) = 1$, which is a contradiction to our choice of $n$.

Now, we prove that $f$ is the least upper bound. Assume there is another bound $g \neq f$ with $g \sqsubseteq f$. Then, there is some $n \in \mathbb{N}$ with $g(n) = \bot$ and $f(n) = 1$. But then, there is also some $k$ such that $n \in M_k$, i.e., $f_k(n) = 1$, so $f_k(n) \not\sqsubseteq g(n)$ and hence, $g$ is not an upper bound for $S$.

2) In 1), we have proven that for every chain, there exists a lub. The constructed function is trivially again in $D$. We also have $c(x) = \bot \in D$ as obvious minimal element, hence, $\sqsubseteq$ is a cpo for $D$.

3) Let $N_i := \{ k \in \mathbb{N} \mid k \leq i \}$. Then, $N_i \sqsubseteq N_{i+1}$ holds. Let

$$f_i(x) = \begin{cases} 1 & \text{if } x \in N_i \\ \bot & \text{otherwise} \end{cases}$$

Then, $f_i \sqsubseteq f_{i+1}$ holds (see above) and hence, $\{f_1, f_2, \ldots \}$ is a chain.

4) 

$$f(g) = \begin{cases} h(y) = \bot & \text{if } \{ x \in \mathbb{N} \mid g(x) \neq \bot \} \text{ is finite} \\ h(y) = 1 & \text{otherwise} \end{cases}$$

b) i) $f_{\text{exp}} : ((\mathbb{N}, \mathbb{N}) \to \mathbb{N}) \to ((\mathbb{N}, \mathbb{N}) \to \mathbb{N})$

$f_{\text{exp}} \text{ exp } (x, 0) = 1$

$f_{\text{exp}} \text{ exp } (x, y) = x * \text{ exp } (x, y - 1)$

ii) 

$$(\phi_{\text{f}_{\text{exp}}}^n(\bot)) (x, y) = \begin{cases} 1 & \text{if } y = 0 \land n > 0 \\ x^y & 0 < y < n \land x \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

iii) The least fixpoint of $\phi_{\text{f}_{\text{exp}}}$ is the function

$$g(x, y) = \begin{cases} 1 & \text{if } y = 0 \\ x^y & 0 < y \land x \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

c)
d) \text{length} = \lambda xs \rightarrow \text{if } (\text{isa}\ \text{Nil} \ xs) \text{ then } Z \text{ else if } (\text{isa}\ \text{Cons} \ xs) \text{ then } S(\text{length} (\text{sel}_{2,2} (\text{argof}\ \text{Cons} \ xs))) \text{ else } \text{bot}

Alternative:
\text{length} = \lambda xs \rightarrow \text{if } (\text{isa}\ \text{Nil} \ xs) \text{ then } Z \text{ else } S(\text{length} (\text{sel}_{2,2} (\text{argof}\ \text{Cons} \ xs)))

Exercise 4 (Lambda Calculus):

\text{(4 + 6 = 10 points)}

a) Please translate the following Haskell expression into an equivalent lambda term (e.g., using \text{Let})

Translate the pre-defined function > to \text{GreaterThan}, + to \text{Plus}, * to \text{Times} and - to \text{Minus} (remember that the infix notation of >, +, * - is not allowed in lambda calculus). It suffices to give the result of the transformation:

\text{let \ sqrt = } \lambda x \ a \rightarrow \text{if } a \ast a > x \text{ then } a - 1 \text{ else } \sqrt x (a + 1) \text{ in } \sqrt u 0

b) Let \ t = \lambda \text{fromto}.\lambda x.\lambda y.\text{If} (\text{Eq} x y) \text{ Nil} (\text{Cons} x (\text{fromto} (\text{Plus} x 1 y))) \text{ and }

\delta = \{ \text{If True } \rightarrow \lambda x.\lambda y.x, \\
\text{If False } \rightarrow \lambda x.\lambda y.y, \\
\text{Fix } \rightarrow \lambda f.\lambda f (\text{Fix} f) \}
\cup \{ \text{Plus} x y \rightarrow z \mid x, y \in \mathbb{Z} \land z = x + y \}
\cup \{ \text{Eq} x y \rightarrow \text{False} \mid x, y \in \mathbb{Z} \land x \neq y \}
\cup \{ \text{Eq} x y \rightarrow \text{True} \mid x, y \in \mathbb{Z} \land x = y \}

Please reduce \text{Fix} t 1 2 by WHNO-reduction with the \rightarrow_{\beta\delta}-relation. List all intermediate steps until reaching weak head normal form, but please write “t” instead of the term it represents whenever possible. However, you may combine several subsequent \rightarrow_{\beta}-steps.

Solution:

a) (\text{Fix} (\lambda \text{sqrt} x a.\text{If} (\text{Greater} (\text{Times} a a) x) (\text{Minus} a 1) (\text{sqrt} x (\text{Plus} a 1))) u 0

b) \text{Fix} t 1 2
\rightarrow_{\delta} (\lambda f.\lambda f (\text{Fix} f)) t 1 2
\rightarrow_{\beta} t (\text{Fix} t) 1 2
\rightarrow_{\beta} (\lambda x.\lambda y.\text{If} (\text{Eq} x y) \text{ Nil} (\text{Cons} x ((\text{Fix} t) (\text{Plus} x 1 y))) 1 2
\rightarrow_{\beta} (\lambda y.\text{If} (\text{Eq} 1 y) \text{ Nil} (\text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 y))) 2
\rightarrow_{\beta} \text{If} (\text{Eq} 1 2) \text{ Nil} (\text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 2))
\rightarrow_{\beta} \text{If False} \text{ Nil} (\text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 2)))
\rightarrow_{\delta} (\lambda x.\lambda y.y) \text{ Nil} (\text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 2)))
\rightarrow_{\beta} (\lambda y.y) (\text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 2))
\rightarrow_{\beta} \text{Cons} 1 ((\text{Fix} t) (\text{Plus} 1 1 2))
Exercise 5 (Type Inference):  

Using the initial type assumption $A_0 := \{y :: \forall a.a \rightarrow a\}$ infer the type of the expression $\lambda x.(y x) x$ using the algorithm $W$.

Solution:

\[
W(A_0, \lambda x.(y x) x) \\
W(A_0 + \{x :: b_1\}, (y x) x) \\
\quad W(A_0 + \{x :: b_1\}, y) = (id, b_2 \rightarrow b_2) \\
\quad W(A_0 + \{x :: b_1\}, x) = (id, b_1) \\
\quad mgu(b_2 \rightarrow b_2, b_1 \rightarrow b_3) = \{b_1/b_2, b_3/b_2\} \\
\quad = (b_1/b_2, b_3/b_2) \\
\quad W(A_0 + \{x :: b_2\}, x) = (id, b_2) \\
\quad mgu(b_2, b_2 \rightarrow b_4) \rightharpoonup \text{occurrence failure}
\]