Prof. Dr. J. Giesl

M. Brockschmidt, F. Emmes

Exercise 1 (Quiz): (4 + 4 + 4 + 4 + 4 = 20 points)

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

a) Is $\sqsubseteq$ always a complete partial order for flat domains like $\mathbb{Z}_\bot, \mathbb{B}_\bot, \ldots$?

b) Can the function $f : \mathbb{Z}_\bot \to \mathbb{Z}$ with $f(x) = \begin{cases} 1 & \text{if } x \in \mathbb{Z} \text{ and } x \leq 0 \\ 0 & \text{otherwise} \end{cases}$ be implemented in Haskell?

c) Is $g : (\mathbb{Z} \to \mathbb{Z}_\bot) \to \mathbb{Z}_\bot$ with $g(h) = \begin{cases} 0 & \text{if } h(x) \neq \bot \quad \text{for all } x \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases}$ continuous?

d) If a lambda term t can be reduced to s with $\to_{\beta\delta}$ using an outermost strategy, can t also be reduced to s with $\to_{\beta\delta}$ using an innermost strategy? Here, you may choose an arbitrary delta-rule set δ.

e) The $\to_{\beta\delta}$ reduction in lambda calculus is confluent. Is Simple Haskell also confluent?

Solution:

a) Yes, because flat domains only have chains of finite length and a minimal element. Hence, by Theorem 2.1.13(a), $\sqsubseteq$ is a complete partial order.

b) No, as f is not continuous and thus, not computable: Consider the chain $S = \{\bot, 0\}$. There exists no least upper bound for $f(S) = \{0, 1\}$, and hence $f(\sqcup S) = \sqcup f(S)$ does not hold.

c) No. For a counterexample, let $f_i(x) = \begin{cases} 0 & \text{if } x \leq i \\ \bot & \text{otherwise} \end{cases}$.

Then for the chain $S = \{f_1, f_2, \ldots\}$, we have $\sqcup S = f_\infty$ with $f_\infty(x) = 0$ for all $x \in \mathbb{Z}$. Then $\sqcup g(S) = \bot \neq 0 = g(\sqcup S)$.

Alternatively, a more intuitive solution: If g were continuous, it would be computable. As it implicitly solves the halting program for an input function, it is known to be uncomputable, hence, we have a contradiction.

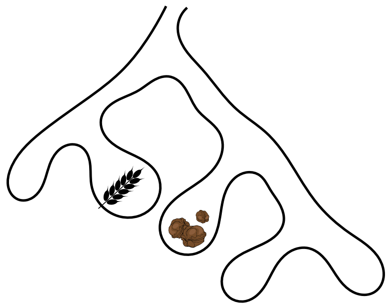
d) No. Consider the term $(\lambda x.42)$ `bot` as example (with the usual δ-rule `bot` $\to$ `bot` for `bot`), which is reduced to 42 using an outermost strategy and does not have a normal form when reducing according to an innermost strategy.

e) Yes, as Simple Haskell can be implemented using the $\to_{\beta\delta}$ reduction.

Exercise 2 (Programming in Haskell): (10 + 10 + 8 + 10 = 38 points)

We define a polymorphic data structure `HamsterCave` to represent hamster caves which can contain different types of food.

```
data HamsterCave food
    = EmptyTunnel
    | FoodTunnel food
    | Branch (HamsterCave food) (HamsterCave food)
    deriving Show
```



The data structure `HamsterFood` is used to represent food for hamsters. For example, `exampleCave` is a valid expression of type `HamsterCave HamsterFood`.

```
data HamsterFood = Grain | Nuts deriving Show

exampleCave :: HamsterCave HamsterFood
exampleCave = Branch
    (Branch EmptyTunnel (FoodTunnel Grain))
    (Branch (FoodTunnel Nuts) (Branch EmptyTunnel EmptyTunnel))
```

a) Implement a function `digAndFillCave :: Int -> HamsterCave HamsterFood`, such that for any integer number `n` > 1, `digAndFillCave n` creates a hamster cave without empty tunnels of depth `n`, such that the number of `FoodTunnels` containing `Grain` equals the number of `FoodTunnels` containing `Nuts`. Here, the depth of a cave is the maximal number of "nodes" on any path from the entry of the cave to a dead end. Thus, `exampleCave` has depth 4.

b) Implement a fold function `foldHamsterCave`, including its type declaration, for the data structure `HamsterCave`. As usual, the fold function replaces the data constructors in a `HamsterCave` expression by functions specified by the user. The first argument of `foldHamsterCave` should be the function for the case of the empty tunnel, the second argument the function for the case of the food tunnel, and the third argument the function for the case of a branch. As an example, the following function definition uses `foldHamsterCave` to determine the number of dead ends (either with or without food) in a cave, such that the call `numberOfDeadEnds exampleCave` returns 5.

```
numberOfDeadEnds :: HamsterCave food -> Int
numberOfDeadEnds cave = foldHamsterCave 1 (\_ -> 1) (+) cave
```

c) Implement the function `collectFood :: HamsterCave food -> (HamsterCave food, [food])`, which returns a tuple for a given hamster cave. The first argument of the tuple is the same hamster cave as the one given to the function, but without any food (i.e., every `FoodTunnel` is replaced by an `EmptyTunnel`). The second argument is a list of all the food that was removed from the cave. For the definition of `collectFood`, use only one defining equation where the right-hand side is a call to the function `foldHamsterCave`.

For example, a call `collectFood exampleCave` should return the following tuple:

```
(Branch (Branch EmptyTunnel EmptyTunnel)
        (Branch EmptyTunnel (Branch EmptyTunnel EmptyTunnel))
,[Grain,Nuts])
```

d) In this part of the exercise, you should create a program that navigates through a hamster cave. To do this, implement a function `exploreHamsterCave :: Show food => HamsterCave food -> IO ()` that, given a `HamsterCave` data structure, prints the kind of food it found at the current position if the `HamsterCave` is a `FoodTunnel`, prints `"Didn't find any food"` for an `EmptyTunnel`, and asks the user whether one should descend into the left or right part of the cave in case of a `Branch`. The user should answer with either `l` for the left cave or `r` for the right cave. If the answer of the user is valid, the function should continue in the corresponding part of the hamster cave. Otherwise, the question should be repeated after a short explanation about the possible options.

A successful run for the `exampleCave` might look as follows:

```
*Main> exploreHamsterCave exampleCave
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? up
Sorry, I did not understand you. Please answer with l for left or r for right.
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? r
Yay! Found Grain!
```

In the following run, the user does not manage to find any food in the cave:

```
*Main> exploreHamsterCave exampleCave
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
Didn't find any food.
```

Hint: You should use the function `getLine :: IO String` to read the input from the user and the function `putStr :: String -> IO ()` to print a `String`. To save space, you may assume that the following declarations exist in your program:

```
question, mistake :: String
question = "You are standing at a fork in the tunnel. "
    ++ "Proceed to the (l)eft or to the (r)ight?"
mistake = "Sorry, I did not understand you. "
    ++ "Please answer with l for left or r for right.\n"
```

Solution:

a)
```
digAndFillCave :: Int -> HamsterCave HamsterFood
digAndFillCave n | n > 1 = Branch (cave Nuts (n-1)) (cave Grain (n-1))
  where
    cave food 1 = FoodTunnel food
    cave food n = Branch (cave food (n-1)) (cave food (n-1))
```

b)
```
foldHamsterCave
    :: result
    -> (food -> result)
    -> (result -> result -> result)
    -> HamsterCave food
    -> result
foldHamsterCave fET fTWF fTB = go
  where
    go EmptyTunnel = fET
    go (FoodTunnel f) = fTWF f
    go (Branch left right) = fTB (go left) (go right)
```

c)
```
collectFood :: HamsterCave food -> (HamsterCave food, [food])
collectFood = foldHamsterCave
        (EmptyTunnel, [])
        (\x -> (EmptyTunnel, [x]))
        (\(tl, fl) (tr, fr) -> (Branch tl tr, fl ++ fr))
```

d)
```
exploreHamsterCave :: Show food => HamsterCave food -> IO ()
exploreHamsterCave EmptyTunnel = putStr "Didn't find any food.\n"
exploreHamsterCave (FoodTunnel food) = putStr ("Yay! Found "
                                       ++ show food ++ "!\n")
exploreHamsterCave (Branch left right) = do
    putStr ("You are standing at a fork in the tunnel. "
            ++ "Proceed to the (l)eft or to the (r)ight? "
    c <- getLine
    case c of
        "l" -> exploreHamsterCave left
        "r" -> exploreHamsterCave right
        _ -> do
```

```
putStr ("Sorry, I did not understand you. "
        ++ "Please answer with l for left or r for right.\n")
exploreHamsterCave (Branch left right)
```

Exercise 3 (Semantics): (21 + 10 + 5 = 36 points)

a) i) Let $\sqsubseteq$ be a cpo on D and $f : D \to D$ be continuous. Prove the fixpoint theorem, i.e., that $\sqcup\{f^i(\bot) \mid i \in \mathbb{N}\}$ exists and that this is the least fixpoint of f. You may use all other results from the lecture in your proof.

ii) Let $D = 2^{\mathbb{N}}$, i.e., D is the set of all sets of natural numbers and let $\subseteq$ denote the usual subset relation.

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 $f : D \to D$. You do not need to prove that f has these properties.

b) i) Consider the following Haskell function `mult`:

```
mult :: (Int, Int) -> Int
mult (0, y) = 0
mult (x, y) = y + mult (x - 1, y)
```

Please give the Haskell declaration for the higher-order function `f_mult` corresponding to `mult`, i.e., the higher-order function `f_mult` such that the least fixpoint of `f_mult` is `mult`. In addition to the function declaration, please also give the type declaration of `f_mult`. You may use full Haskell for `f_mult`.

ii) Let $\phi_{\texttt{f\_mult}}$ be the semantics of the function `f_mult`. Give the semantics of $\phi^n_{\texttt{f\_mult}}(\bot)$ for $n \in \mathbb{N}$, i.e., the semantics of the n-fold application of $\phi_{\texttt{f\_mult}}$ to $\bot$.

iii) Give all fixpoints of $\phi_{\texttt{f\_mult}}$ and mark the least fixpoint.

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

```
data Nats = Z | S Nats
```

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

$$\begin{tikzcd}
 & & \texttt{S (S Z)} & & \texttt{S (S (S }\perp\texttt{))} \\
 & \texttt{S Z} & & \texttt{S (S }\perp\texttt{)} \arrow[ul] \arrow[ur] & \\
\texttt{Z} & & \texttt{S }\perp \arrow[ul] \arrow[ur] & & \\
 & \perp \arrow[ul] \arrow[ur] & & &
\end{tikzcd}$$

Now consider the following data type declarations:

```
data X = A X Y | B Y
data Y = E Y | H
```

Give a graphical representation of the first three levels of the domain for the type `X`. The third level contains the element `A (A` $\perp$ $\perp$`)` $\perp$, for example.

Solution:

a) i) We first prove that $f^i(\perp) \sqsubseteq f^{i+1}(\perp)$ holds for all $i \in \mathbb{N}$ by induction. As base case, we consider $i = 0$ and of course, $f^0(\perp) = \perp \sqsubseteq f^1(\perp)$ holds.

In the induction step, we assume that for some $i > 0$, $f^{i-1}(\perp) \sqsubseteq f^i(\perp)$ holds. Then, because f is continuous, f is also monotonic, hence $f(f^{i-1}(\perp)) \sqsubseteq f(f^i(\perp)) \Leftrightarrow f^i(\perp) \sqsubseteq f^{i+1}(\perp)$ holds.

Thus, $\{f^i(\perp) \mid i \in \mathbb{N}\}$ is a chain and because $\sqsubseteq$ is a cpo on D, $\sqcup\{f^i(\perp) \mid i \in \mathbb{N}\}$ exists. We now need to prove that this is the least fixpoint of f. First, we prove that this is indeed a fixpoint:

$$\begin{aligned}
f(\sqcup\{f^i(\perp) \mid i \in \mathbb{N}\}) &= \sqcup f(\{f^i(\perp) \mid i \in \mathbb{N}\}) && (f \text{ continuous})\\
&= \sqcup\{f^{i+1}(\perp) \mid i \in \mathbb{N}\}\\
&= \sqcup(\{f^{i+1}(\perp) \mid i \in \mathbb{N}\} \cup \{\perp\})\\
&= \sqcup\{f^i(\perp) \mid i \in \mathbb{N}\}
\end{aligned}$$

Now assume there is another fixpoint d of f. We need to prove $\sqcup\{f^i(\perp) \mid i \in \mathbb{N}\} \sqsubseteq d$ and do this by inductively proving $f^i(\perp) \sqsubseteq d$. In the base case, $f^0(\perp) = \perp \sqsubseteq d$ obviously holds. In the induction step, assume $f^i(\perp) \sqsubseteq d$ already holds. Then, because f is monotonic, we have $f(f^i(\perp)) \sqsubseteq f(d)$. But as d is a fixpoint of f, we can conclude that $f^{i+1}(\perp) \sqsubseteq d$.

ii) Let $S = \{M_1, M_2, \ldots\}$ with $M_i \subseteq M_{i+1}$.

1) We have $\sqcup S = \bigcup M_i$. Obviously, $M_i \subseteq \bigcup M_i$. Now assume that there is some other upper bound B with $\bigcup M_i \not\subseteq B$. Then there is some $e \in \bigcup M_i \setminus B$ and by construction, there is some k with $e \in M_k$. As $e \notin B$, we have $M_k \not\subseteq B$ and hence, B is not an upper bound of S w.r.t. $\subseteq$. Thus, we have a contradiction.

2) In 1), we have proven that for every chain, there exists a lub. Obviously, we have $\bigcup M_i \in D$. With $\emptyset$ as the minimal element, $\subseteq$ is a cpo for D.
3) Let $N_i := \{k \in \mathbb{N} \mid k \leq i\}$. Then, $N_i \subseteq N_{i+1}$ holds and hence, $\{N_1, N_2, \ldots\}$ is a chain.
4)

$$f(M) = \begin{cases} \emptyset & M \text{ is finite} \\ \{42\} & \text{otherwise} \end{cases}$$

Alternative: The function g from Ex. 1 c).

b) i)

```
f_mult :: ((Int, Int) -> Int) -> ((Int, Int) -> Int)
f_mult mult (0, y) = 0
f_mult mult (x, y) = y + mult (x - 1, y)
```

ii)

$$(\phi^n_{\texttt{f\_mult}}(\bot))(x,y) = \begin{cases} 0 & \text{if } x = 0 \land n > 0 \\ x \cdot y & \text{if } 0 < x < n \land y \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

iii) The least fixpoint of $\phi_{\texttt{f\_mult}}$ is the function

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

Another fixpoint is the function

$$h(x,y) = \begin{cases} 0 & \text{if } x = 0 \\ x \cdot y & \text{if } x \neq \bot \land y \neq \bot \\ \bot & \text{if } x = \bot \lor (x \neq 0 \land y = \bot) \quad \text{(this is ``otherwise'')} \end{cases}$$

To be a fixpoint, a function f has to satisfy the equality $f(c_1, c_2) = \phi_{\texttt{f\_mult}}(f)(c_1, c_2)$, which is equivalent to $f(c_1, c_2) = 0$ for $c_1 = 0$ (this is the first case in the definitions above).

For $c_1 \neq 0$, we have $f(c_1, c_2) = c_2 + f(c_1 - 1, c_2)$. This implies that for $c_1 = \bot$, the result has to be $\bot$, as $c_1 - 1$ is not well-defined in that case. For $c_2 = \bot$ (and $c_1 \neq 0$, as that case was handled above), the result also has to be $\bot$, as $c_2 + f(c_1 - 1, c_2)$ is not well-defined in that case. This corresponds to the last case in the definition of h.

So finally, we are left with the cases for $c_1, c_2 \in \mathbb{Z}$, for which $f(c_1, c_2) = c_2 + f(c_1 - 1, c_2)$ has to hold, which is exactly the condition for multiplication, yielding the middle case.

c)

$$\begin{tikzcd}
\texttt{A (A } \bot\ \bot\texttt{) } \bot & \texttt{A (B } \bot\texttt{) } \bot & \texttt{A } \bot \texttt{ H} & \texttt{A } \bot \texttt{ (E } \bot\texttt{)} & \texttt{B H} & \texttt{B (E } \bot\texttt{)} \\
 & & \texttt{A } \bot\ \bot \arrow[ull] \arrow[ul] \arrow[u] \arrow[ur] & & \texttt{B } \bot \arrow[u] \arrow[ur] & \\
 & & & \bot \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Exercise 4 (Lambda Calculus): (4 + 6 + 10 = 20 points)

a) Please translate the following Haskell expression into an equivalent lambda term (e.g., using $\mathcal{L}am$). Translate the pre-defined function `<` to `LessThan`, `+` to `Plus` and `-` to `Minus` (remember that the infix notation of `<`, `+`, `-` is not allowed in lambda calculus). It suffices to give the result of the transformation:

```
let quot = \x y -> if x < y then 0 else 1 + quot (x-y) y in quot v w
```

b) Let $t = \lambda fact.(\lambda x.(\texttt{If}\ (\texttt{LessThanOrE}\ x\ 1)\ 1\ (\texttt{Times}\ x\ (fact\ (\texttt{Minus}\ x\ 1)))))$ and

$$\begin{aligned}
\delta = \ & \{\ \texttt{If True} \to \lambda x\ y.x,\\
& \quad \texttt{If False} \to \lambda x\ y.y,\\
& \quad \texttt{fix} \to \lambda f.f(\texttt{fix}\ f)\}\\
\cup\ & \{\ \texttt{Minus}\ x\ y \to z \mid x, y \in \mathbb{Z} \wedge z = x - y\}\\
\cup\ & \{\ \texttt{Times}\ x\ y \to z \mid x, y \in \mathbb{Z} \wedge z = x \cdot y\}\\
\cup\ & \{\ \texttt{LessThanOrE}\ x\ y \to b \mid x, y \in \mathbb{Z} \wedge ((x \le y \wedge b = \texttt{True}) \vee (x > y \wedge b = \texttt{False}))\}
\end{aligned}$$

Please reduce $\texttt{fix}\ t\ 1$ by WHNO-reduction with the $\to_{\beta\delta}$-relation. List **all** intermediate steps until reaching weak head normal form, but please write "t" instead of the term it represents whenever possible.

c) We use the representation of natural numbers in the pure λ-calculus presented in the lecture. So, n is represented as $\lambda f\ x.f^n\ x$, i.e., $\mathsf{0}$ is represented as $\lambda f x.x$, $\mathsf{1}$ as $\lambda f x.f x$ and so on. Using this representation, give pure λ-terms for the following functions:

- $\overline{\texttt{Succ}}$, which increases a natural number n to $n+1$. Thus, $\overline{\texttt{Succ}}\ (\lambda f\ x.t) \to_\beta^* \lambda f\ x.f\ t$, where t is the term $f^n\ x$.
- $\overline{\texttt{Plus}}$, which adds two natural numbers n, m. Here, you may use the λ-term $\overline{\texttt{Succ}}$. Recall that $n + m$ is the same as the m-th successor of n.

Solution:

a) $(\texttt{fix}\ (\lambda quot\ x\ y.\texttt{If}\ (\texttt{LessThan}\ x\ y)\ 0\ (\texttt{Plus}\ 1\ (quot\ (\texttt{Minus}\ x\ y)\ y))))\ v\ w$

b)

$$\begin{aligned}
& \texttt{fix}\ t\ 1\\
\to_\delta\ & (\lambda f.(f\ (\texttt{fix}\ f)))\ t\ 1\\
\to_\beta\ & t\ (\texttt{fix}\ t)\ 1\\
\to_\beta\ & (\lambda x.(\texttt{If}\ (\texttt{LessThanOrE}\ x\ 1)\ 1\ (\texttt{Times}\ x\ (\texttt{fix}\ t\ (\texttt{Minus}\ x\ 1)))))\ 1\\
\to_\beta\ & \texttt{If}\ (\texttt{LessThanOrE}\ 1\ 1)\ 1\ (\texttt{Times}\ 1\ (\texttt{fix}\ t\ (\texttt{Minus}\ 1\ 1))) \qquad (*)\\
\to_\delta\ & \texttt{If True}\ 1\ (\texttt{Times}\ 1\ (\texttt{fix}\ t\ (\texttt{Minus}\ 1\ 1)))\\
\to_\delta\ & (\lambda x.(\lambda y.x))\ 1\ (\texttt{Times}\ 1\ (\texttt{fix}\ t\ (\texttt{Minus}\ 1\ 1)))\\
\to_\beta\ & (\lambda y.1)\ (\texttt{Times}\ 1\ (\texttt{fix}\ t\ (\texttt{Minus}\ 1\ 1)))\\
\to_\beta\ & 1
\end{aligned}$$

[The original exam had a mixed use of `If` and `if`, so technically, it was OK to stop after reaching the term marked with $(*)$.]

c)
- $\overline{\texttt{Succ}} = \lambda n\ f\ x.f\ (n\ f\ x)$
- $\overline{\texttt{Plus}} = \lambda n\ m\ f\ x.n\ f\ (m\ f\ x)$ or
  $\overline{\texttt{Plus}} = \lambda n\ m.n\ \overline{\texttt{Succ}}\ m$

Exercise 5 (Type Inference): (6 points)

Using the initial type assumption $A_0 := \{x :: \forall a.a \to \texttt{Int}\}$ infer the type of the expression $\lambda y.y\,x$ using the algorithm $\mathcal{W}$.

Solution:

```
$\mathcal{W}(A_0, \lambda y.y\,x)$
    $\mathcal{W}(A_0 + \{y :: b_1\}, y\,x)$
        $\mathcal{W}(A_0 + \{y :: b_1\}, y) = (id, b_1)$
        $\mathcal{W}(A_0 + \{y :: b_1\}, x) = (id, b_2 \to \texttt{Int})$
    $mgu(b_1, (b_2 \to \texttt{Int}) \to b_3) = [b_1/(b_2 \to \texttt{Int}) \to b_3])$
    $= ([b_1/(b_2 \to \texttt{Int}) \to b_3], b_3)$
$= ([b_1/(b_2 \to \texttt{Int}) \to b_3], ((b_2 \to \texttt{Int}) \to b_3) \to b_3)$
```