PMATH911: Assignment 2

(Adapted from the solutions of Forte Shinko)

Question 1

We use the functional forms of the inference rules.

Part a

The following expression is of the desired type:

$$\lambda f.(\lambda x.\,\mathrm{pr}_1(f(x)), \lambda x.\,\mathrm{pr}_2(f(x)))$$

In order to see this, note that the following judgements are derivable:

$$f : \prod_{x:A} P(x) \times Q(x),\ x : A \vdash \mathrm{pr}_1(f(x)) : P(x), \qquad f : \prod_{x:A} P(x) \times Q(x),\ x : A \vdash \mathrm{pr}_2(f(x)) : Q(x).$$

By λ-abstraction and the introduction rule for cartesian products, one gets

$$f : \prod_{x:A} P(x) \times Q(x) \vdash (\lambda x.\,\mathrm{pr}_1(f(x)), \lambda x.\,\mathrm{pr}_2(f(x))) : \left(\prod_{x:A} P(x)\right) \times \left(\prod_{x:A} Q(x)\right),$$

and another λ-abstraction results in the desired

$$\vdash \lambda f.(\lambda x.\,\mathrm{pr}_1(f(x)), \lambda x.\,\mathrm{pr}_2(f(x))) : \left(\prod_{x:A} P(x) \times Q(x)\right) \to \left(\prod_{x:A} P(x)\right) \times \left(\prod_{x:A} Q(x)\right).$$

Part b

The following expression is of the desired type:

$$\begin{aligned}\mathrm{ind}_{(\sum_{x:A} P(x))+(\sum_{x:A} Q(x))}\Bigg(&\sum_{x:A} P(x) + Q(x),\\ &\mathrm{ind}_{\sum_{x:A} P(x)}\Big(\sum_{x:A} P(x) + Q(x), \lambda(a:A).\lambda(u:P(a)).(a, \mathrm{inl}(u))\Big),\\ &\mathrm{ind}_{\sum_{x:A} Q(x)}\Big(\sum_{x:A} P(x) + Q(x), \lambda(a:A).\lambda(v:Q(a)).(a, \mathrm{inr}(v))\Big)\Bigg).\end{aligned}$$

In all three inductions, the last argument has been left unspecified since we still want to obtain functions. We omit the verification that this expression has the correct type, i.e. the proof tree.
Informally speaking, we construct the desired function with codomain $T :\equiv \sum_{x:A} P(x) + Q(x)$ by first doing induction over the coproduct, which splits the problem into defining functions $(\sum_{x:A} P(x)) \to T$ and $(\sum_{x:A} Q(x)) \to T$. Constructing both of these is done via another induction over the dependent pair, which reduces the problem to construct dependent functions of types $\prod_{a:A} P(a) \to T$ and $\prod_{a:A} Q(a) \to T$, which can be taken to be $\lambda a.\lambda u.(a, \mathrm{inl}(u))$ and $\lambda a.\lambda u.(a, \mathrm{inr}(u))$, respectively.

Question 2

Part a

Define $\texttt{leaves} : \texttt{BinTree} \to \mathcal{U}$ inductively by the following:

$$\begin{aligned}\texttt{leaves}(\texttt{treeroot}) &:\equiv \mathbf{1} \\ \texttt{leaves}(\texttt{join}(x, y)) &:\equiv \texttt{leaves}(x) + \texttt{leaves}(y)\end{aligned}$$

where $\mathbf{1}$ denotes the unit type and $+$ denotes the coproduct.

Part b

Define the inductive type $\texttt{BinTree}(A)$ with the following two constructors:

- $\texttt{treeroot} : A \to \texttt{BinTree}(A)$
- $\texttt{join} : \texttt{BinTree}(A) \to \texttt{BinTree}(A) \to \texttt{BinTree}(A)$

This guarantees that every leaf—which comes with a label attached to it, which is the element of A with which it was originally constructed as a $\texttt{treeroot}$ before having joined it with other trees. This is similar to the definition of $\texttt{List}(A)$, where the A-dependence is in the constructor $\texttt{cons}$.
Now for $A : \mathcal{U}$, define $\texttt{fold} : \texttt{Magma}(A) \to \texttt{BinTree}(A) \to A$ again by induction as follows:

$$\begin{aligned}\texttt{fold}(m, \texttt{treeroot}(a)) &:\equiv a \\ \texttt{fold}(m, \texttt{join}(x, y)) &:\equiv m(\texttt{fold}(m, x), \texttt{fold}(m, y))\end{aligned}$$

Alternatively, one could have defined $\texttt{BinTree}(A)$ to be the pair type corresponding to a binary tree together with a labeling of its leaves, which is a function to A:

$$\texttt{BinTree}(A) :\equiv \sum_{T:\texttt{BinTree}} \texttt{leaves}(T) \to A.$$

In this approach, $\texttt{fold} : \texttt{Magma}(A) \to \texttt{BinTree}(A) \to A$ is more difficult to define. By induction over $\sum_{T:\texttt{BinTree}} \texttt{leaves}(T) \to A$, it is sufficient to define $\texttt{fold}$ on $m : A \to A \to A$ for an explcitly given pair $T : \texttt{BinTree}$ and $l : \texttt{leaves}(T) \to A$ only; so it is sufficient to define $\texttt{fold}(m, (T, l)) : A$. In this case, we can use induction on T to put

$$\begin{aligned}\texttt{fold}(m, (\texttt{treeroot}, l)) &:\equiv l(*) \\ \texttt{fold}(m, (\texttt{join}(T_1, T_2), l)) &:\equiv m(\texttt{fold}(m, (T_1, l \circ \mathrm{inl})), \texttt{fold}(m, (T_2, l \circ \mathrm{inr}))),\end{aligned}$$

where we use $\texttt{leaves}(\texttt{join}(T_1, T_2)) \equiv \texttt{leaves}(T_1) + \texttt{leaves}(T_2)$, so that $l \circ \mathrm{inl} : \texttt{leaves}(T_1) \to A$ and $l \circ \mathrm{inr} : \texttt{leaves}(T_2)$ are indeed well-typed labellings of the constituent subtrees.

Question 3

The following expression is of the desired type:

$$\lambda\left(g : \prod_{x:A} \sum_{y:B} R(x, y)\right). (\lambda x.\, \mathrm{pr}_1(g(x)), \lambda x.\, \mathrm{pr}_2(g(x))).$$

In contrast to Question 1, the pair $(\lambda x.\, \mathrm{pr}_1(g(x)), \lambda x.\, \mathrm{pr}_2(g(x))) : \sum_{f:A\to B} \prod_{x:A} R(x, f(x))$ is now a *dependent* pair, so we need to verify that the types match accordingly,

$$\begin{aligned}\lambda x.\, \mathrm{pr}_1(g(x)) &: A \to B, \\ \lambda x.\, \mathrm{pr}_2(g(x)) &: \prod_{x:A} R(x, (\lambda x.\, \mathrm{pr}_1(g(x)))(x)).\end{aligned}$$

The first typing statement follows from $g(x) : \sum_{y:B} R(x,y)$, so that $\mathrm{pr}_1(g(x)) : B$. In the second typing statement, we can use $(\lambda x.\, \mathrm{pr}_2(g(x)))(x) \equiv \mathrm{pr}_2(g(x))$, so that the typing judgement

$$\mathrm{pr}_2(g(x)) \; : \; R(x, \mathrm{pr}_1(g(x)))$$

matches exactly with the type of the second projection of a dependent pair type.

Question 4

Part a

Define `pred` $: \mathbb{N} \to \mathbb{N}$ by induction as follows:

$$\begin{aligned} \texttt{pred}(0) &:\equiv 0 \\ \texttt{pred}(\texttt{succ}(n)) &:\equiv n \end{aligned}$$

Part b

Define `map` $: (A \to B) \to \texttt{List}(A) \to \texttt{List}(B)$ by induction over $\texttt{List}(A)$ as follows:

$$\begin{aligned} \texttt{map}(f, \texttt{nil}) &:\equiv \texttt{nil} \\ \texttt{map}(f, \texttt{cons}(x,l)) &:\equiv \texttt{cons}(f(x), \texttt{map}(f,l)) \end{aligned}$$

Part c

Unlike in Question 2, we now should consider all nodes of a binary tree to be labelled by elements of A. As in Question 2, there are two ways of defining $\texttt{BinTree}(A)$, resulting in equivalent types. Taking $\texttt{BinTree}(A)$ to be an inductive type defined "in one go" is preferably over defining it as a dependent pair type with one component in `BinTree` and the other component representing the labelling, since it allows for more direct constructions of functions out of $\texttt{BinTree}(A)$. So we take $\texttt{BinTree}(A)$ to be the inductive type with constructors

- `treeroot` $: A \to \texttt{BinTree}(A)$
- `join` $: A \to \texttt{BinTree}(A) \to \texttt{BinTree}(A) \to \texttt{BinTree}(A)$.

Now define a function which concatenates two lists `append` $: \texttt{List}(A) \to \texttt{List}(A) \to \texttt{List}(A)$ as follows:

$$\begin{aligned} \texttt{append}(\texttt{nil}, l) &:\equiv l \\ \texttt{append}(\texttt{cons}(x,l), m) &:\equiv \texttt{cons}(x, \texttt{append}(l,m)) \end{aligned}$$

Now define `labels` $: \texttt{BinTree}(A) \to \texttt{List}(A)$ as follows:

$$\begin{aligned} \texttt{labels}(\texttt{treeroot}(a)) &:\equiv a \\ \texttt{labels}(\texttt{join}(a,x,y)) &:\equiv \texttt{append}(\texttt{labels}(x), \texttt{cons}(a, \texttt{labels}(y))). \end{aligned}$$

This lists first the list of labels on the left subtree, then the label of the root node, and finally the list of labels on right subtree.

$\square$