AFP 2021-2022 Take home question

Finitely branching trees

1. In the course of the lectures, we saw how to define a *universe* for generic programming closed under products (pairs), coproducts (*Either*), and constant types. To do so, we defined a data type to represent the types in the universe U, and a function mapping elements of U to the pattern functor they represent.

$$
\begin{array}{l}
\textbf{data}\ U : \mathit{Set}\ \textbf{where} \\
\quad \_\oplus\_ : U \to U \to U \\
\quad \_\otimes\_ : U \to U \to U \\
\quad I \;\; : U \\
\quad K \;\; : \mathit{Set} \to U \\
\quad \ldots \\
\mathit{elU} : U \to \mathit{Set} \to \mathit{Set} \\
\ldots
\end{array}
$$

   In this question, we will explore an alternative representation of such types, namely as *finitely branching trees*. We will begin by defining the following data type:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Tree}\ (S : \mathit{Set})\ (P : S \to \mathit{Nat}) : \mathit{Set}\ \textbf{where} \\
\quad \mathit{Node} : (s : S) \to (\mathit{Fin}\ (P\ s) \to \mathit{Tree}\ S\ P) \to \mathit{Tree}\ S\ P
\end{array}
$$

   Values of type $\mathit{Tree}\ S\ P$ are (recursive) tree-like data structures. Here the 'shape' S describes the (non-recursive) parts of the data type. For every shape $s : S$, there are $P\ s$ recursive subtrees. The inhabitants of the type $\mathit{Tree}\ S\ P$ correspond to a choice of constructor from S and a function $\mathit{Fin}\ (P\ s) \to \mathit{Tree}\ S\ P$, that can be used to consult the i-th subtree. Remember that the $\mathit{Fin}\ n$ data type, corresponding to a type with precisely n inhabitants, is defined as follows:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Fin} : \mathit{Nat} \to \mathit{Set}\ \textbf{where} \\
\quad \mathit{fzero} : \mathit{forall}\ \{n\} \to \mathit{Fin}\ (\mathit{succ}\ n) \\
\quad \mathit{fsucc} : \mathit{forall}\ \{n\} \to \mathit{Fin}\ n \to \mathit{Fin}\ (\mathit{succ}\ n)
\end{array}
$$

   (a) (2 points) We will begin by modelling lists of natural numbers as Tree-types. Given the following data type:

$$
\begin{array}{l}
\textbf{data}\ \mathit{ListS} : \mathit{Set}\ \textbf{where} \\
\quad \mathit{Nil} : \mathit{ListS} \\
\quad \mathit{Cons} : \mathit{Nat} \to \mathit{ListS}
\end{array}
$$

   Define a function $\mathit{ListP} : \mathit{ListS} \to \mathit{Nat}$, such that $\mathit{Tree}\ \mathit{ListS}\ \mathit{ListP}$ is isomorphic to $\mathit{List}\ \mathit{Nat}$.

   Also define the constructor functions:

$$
\begin{array}{l}
\mathit{nil} : \mathit{Tree}\ \mathit{ListS}\ \mathit{ListP} \\
\mathit{cons} : \mathit{Nat} \to \mathit{Tree}\ \mathit{ListS}\ \mathit{ListP} \to \mathit{Tree}\ \mathit{ListS}\ \mathit{ListP}
\end{array}
$$

   (b) (2 points) Define TreeS and TreeP such that $\mathit{Tree}\ \mathit{TreeS}\ \mathit{TreeP}$ is isomorphic to the following tree type:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Tree} : \mathit{Set}\ \textbf{where} \\
\quad \mathit{Node} : \mathit{Tree} \to \mathit{Tree} \to \mathit{Tree} \\
\quad \mathit{Leaf} : \mathit{Nat} \to \mathit{Tree}
\end{array}
$$

   (c) (4 points) Write a generic size function that counts the number of Node constructors in its argument:

$$
\mathit{gsize} : \mathit{Tree}\ S\ P \to \mathit{Nat}
$$

   You may find it helpful to define an auxiliary function of the following type:

$$
\mathit{sumFin} : (n : \mathit{Nat}) \to (\mathit{Fin}\ n \to \mathit{Nat}) \to \mathit{Nat}
$$

   Such that $\mathit{sumFin}\ n\ f$ sums all the natural numbers in the image of f.

(d) (4 points) Let $M : Set$ be a monoid, that is, we have a zero element $z : M$ and 'addition' operation $\_+\_ : M \to M \to M$. Define an analogue of Haskell's *foldMap* function on finitely branching trees:

$$foldMap : (S \to M) \to Tree\ S\ P \to M$$

(e) (5 points) Show that *Tree* types are closed under coproducts. That is, define operations:

$$\begin{aligned}
&_ \oplus_S _ \quad : (S : Set) \to (S' : Set) \to Set \\
&_ \oplus_P _ \quad : (P : S \to Nat) \to (P' : S' \to Nat) \to (S \oplus_S S' \to Nat) \\
&inl : Tree\ S\ P \to Tree\ (S \oplus_S S')\ (P \oplus_P P') \\
&inr : Tree\ S'\ P' \to Tree\ (S \oplus_S S')\ (P \oplus_P P')
\end{aligned}$$

(f) (5 points) Given a choice of $S : Set$ and $P : S \to Nat$, we can compute a the pattern functor corresponding to *Tree* $S\ P$ as follows:

$$\begin{aligned}
&\textbf{data}\ DPair\ (S : Set)\ (B : S \to Set) : Set\ \textbf{where} \\
&\quad _,_ : (s : S) \to B\ s \to DPair\ S\ B \\
&toPF : (S : Set) \to (P : S \to Nat) \to Set \to Set \\
&toPF\ S\ P\ a = DPair\ S\ (\lambda s \to Fin\ (P\ s) \to a)
\end{aligned}$$

Use this definition to formulate properties showing your definitions for coproducts is correct. *Hint:* define conversions between the pattern functors arising between coproducts and the usual coproduct of pattern functors using *Either*. Prove that these conversions are mutual inverses.