IPA Basic Course Software Technology 2001
Functional Data Structures: Exercises

Ralf Hinze
Institute of Information and Computing Sciences, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
ralf@cs.uu.nl

May 28, 2001

Consider the following 'set' data type.

$$
\begin{array}{lll}
\textbf{data}\ \mathit{Set}\ a & & \\
\emptyset & :: & (\mathit{Ord}\ a) \Rightarrow \mathit{Set}\ a \\
\{\cdot\} & :: & (\mathit{Ord}\ a) \Rightarrow a \rightarrow \mathit{Set}\ a \\
(\uplus) & :: & (\mathit{Ord}\ a) \Rightarrow \mathit{Set}\ a \rightarrow \mathit{Set}\ a \rightarrow \mathit{Set}\ a \\
\mathit{max} & :: & (\mathit{Ord}\ a) \Rightarrow \mathit{Set}\ a \rightarrow a
\end{array}
$$

The constructor $\emptyset$ denotes the empty set, $\{\cdot\}$ forms a singleton set, and $s_l \uplus s_r$ takes the disjoint union of s_l and s_r under the proviso that the elements in s_l precede the elements in s_r.

For each constructor there is a corresponding destructor (typeset with a bar) that can be used in patterns: $\bar{\emptyset}$ matches the empty set, $\bar{\{}\cdot\bar{\}}$ matches singleton sets, and $s_l \mathbin{\bar{\uplus}} s_r$ matches sets with at least two elements. In the latter case, we may assume that $\mathit{max}\ s_l < \mathit{min}\ s_r$ and furthermore that both s_l and s_r are non-empty. Thus, the patterns $\bar{\emptyset}$, $\bar{\{}a\bar{\}}$, and $s_l \mathbin{\bar{\uplus}} s_r$ are exhaustive and exclusive.

The operation max is used to determine the maximum element of a non-empty set.

All operations, constructors as well as destructors, are guaranteed to have a running time that is bounded by a constant.

Exercise 1 Implement the following set operations using the above interface.

$$
\begin{array}{lll}
\mathit{min} & :: & (\mathit{Ord}\ a) \Rightarrow \mathit{Set}\ a \rightarrow a \\
\mathit{member} & :: & (\mathit{Ord}\ a) \Rightarrow a \rightarrow \mathit{Set}\ a \rightarrow \mathit{Bool} \\
\mathit{insert} & :: & (\mathit{Ord}\ a) \Rightarrow a \rightarrow \mathit{Set}\ a \rightarrow \mathit{Set}\ a \\
\mathit{delete} & :: & (\mathit{Ord}\ a) \Rightarrow a \rightarrow \mathit{Set}\ a \rightarrow \mathit{Set}\ a \\
\mathit{delete\text{-}max} & :: & (\mathit{Ord}\ a) \Rightarrow \mathit{Set}\ a \rightarrow \mathit{Set}\ a
\end{array}
$$

The function min determines the least element of a set; $\mathit{delete\text{-}max}$ removes the maximum element from a set.

Exercise 2 Implement the following higher-order functions that fold a set in a list-like fashion (again, building upon the above interface).

$$
\begin{aligned}
\mathit{foldr} &:: (a \to b \to b) \to b \to (\mathit{Set}\ a \to b) \\
\mathit{foldl} &:: (b \to a \to b) \to b \to (\mathit{Set}\ a \to b)
\end{aligned}
$$

Use *foldr* and *foldl* to define the following observer functions.

$$
\begin{aligned}
\mathit{size} &:: \mathit{Set}\ a \to \mathit{Int} \\
\mathit{inorder} &:: \mathit{Set}\ a \to [\,a\,]
\end{aligned}
$$

The function *inorder* that converts a set to an ordered list should have a linear running time.

Exercise 3 Implement the above 'set' interface using binary search trees as the underlying data structure. Use 'smart constructors' to define the constructors $\emptyset$, $\{\cdot\}$, and '$\uplus$' and a 'view' to define the corresponding destructors $\bar{\emptyset}$, $\bar{\{}\cdot\bar{\}}$, and '$\bar{\uplus}$'. Note that all operations must have a running time that is bounded by a constant. [*Hint:* Restrict binary search trees so that *max* can be made to run in constant time.]

Exercise 4$*$ Modify the implementation so that *min* can also be supported in constant time.