Contents

1 Preamble 2
2 Signature 3
3 Documentation 4
  3.1 Constructing a Set ........................................ 4
  3.2 Destructing a Set ........................................ 5
  3.3 Element Operations ........................................ 6
  3.4 Bulk Operations ........................................... 7
1 Preamble

The SET signature outlines the functions necessary for basic set functionality. Given a structure that lets us compare elements for equality, we can perform all basic set operations such as inserting, removing, checking for membership, union, intersection, and set difference.

Note that this structure uses the EQ signature, which outlines what it means for a type t to be comparable for equality:

```ocaml
signature EQ =
  sig
    type t
    val equal : t * t -> bool
  end
```

In order to make use of the set signature, we have provided a functor MkSet, which accepts as an argument a structure Elt which ascribes to the EQ signature. You will have to define your own structure Elt. It looks something like this:

```ocaml
functor MkSet (Elt : EQ) :> SET where type Elt.t = Elt.t =
```
2 Signature

signature SET =

sig

(* The EQ structure to use for element comparison *)
structure Elt : EQ

(* The type of the set *)
type t

(* These functions give the capability to create a set *)
val empty : t
val singleton : Elt.t -> t
val fromSeq : Elt.t Seq.t -> t

(* These functions give information about a set (destructors) *)
val size : t -> int
val toSeq : t -> Elt.t Seq.t

(* Element related functions *)
val insert : t -> Elt.t -> t
val remove : t -> Elt.t -> t
val member : t -> Elt.t -> bool

(* Bulk Operations *)
val union : t * t -> t
val intersection : t * t -> t
val difference : t * t -> t

end
3 Documentation

3.1 Constructing a Set

<table>
<thead>
<tr>
<th>empty : t</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSURES: empty is a set with no elements, {}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>singleton : Elt.t -&gt; t</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSURES: singleton x returns a set containing only x as an element, {x}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fromSeq : Elt.t Seq.t -&gt; t</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSURES: fromSeq⟨x₀, x₁, ..., xₙ₋₁⟩ returns a set containing each of x₀, x₁, ..., xₙ₋₁, {x₀, x₁, ..., xₙ₋₁}. If there exist duplicate elements, the first one is chosen and the rest are discarded.</td>
</tr>
</tbody>
</table>
3.2 Destructing a Set

\[
\text{size} : t \rightarrow \text{int}
\]
ENSURES: \text{size} \ S returns the number of elements in S

\[
\text{toSeq} : t \rightarrow \text{Elt} . t \ \text{Seq} . t
\]
ENSURES: \text{toSeq} \ S returns a sequence containing every element in S
3.3 Element Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Signature</th>
<th>Ensures Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>$t \rightarrow \text{Elt} \rightarrow t$</td>
<td>$\text{insert } S \ x$ returns a set $S'$ that contains all of the elements of $S$ and $x$, $S \cup {x}$</td>
</tr>
<tr>
<td>remove</td>
<td>$t \rightarrow \text{Elt} \rightarrow t$</td>
<td>$\text{remove } S \ x$ returns a set $S'$ that contains all of the elements of $S$ except for $x$ if $x \in S$, and $S$ otherwise, $S \setminus {x}$</td>
</tr>
<tr>
<td>member</td>
<td>$t \rightarrow \text{Elt} \rightarrow \text{bool}$</td>
<td>$\text{member } S \ x$ returns $\text{true}$ if $x$ is a member of $S$ and $\text{false}$ otherwise.</td>
</tr>
</tbody>
</table>
3.4 Bulk Operations

union : t * t -> t
ENSURES: union (S1, S2) returns a set S’ containing all of the elements contained within either S1 or S2, i.e. S1 ∪ S2

intersection : t * t -> t
ENSURES: intersection (S1, S2) returns a set S’ containing all of the elements contained within both S1 and S2, i.e. S1 ∩ S2

difference : t * t -> t
ENSURES: difference (S1, S2) returns a set S’ containing all of the elements contained within S1 but not within S2, i.e. S1 \ S2