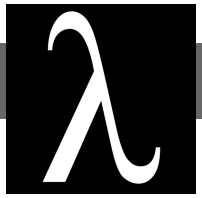

Contents

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:

```
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:

```
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

`empty : t`

ENSURES: `empty` is a set with no elements, {}

`singleton : Elt.t -> t`

ENSURES: `singleton x` returns a set containing only `x` as an element, {`x`}

`fromSeq : Elt.t Seq.t -> t`

ENSURES: `fromSeq`$\langle x_0, x_1, ..., x_{n-1} \rangle$ returns a set containing each of $x_0, x_1, ..., x_{n-1}$, $\{x_0, x_1, ..., x_{n-1}\}$. If there exist duplicate elements, the first one is chosen and the rest are discarded.

3.2 Destructing a Set

```
size : t -> int
```

ENSURES: `size S` returns the number of elements in `S`

```
toSeq : t -> Elt.t Seq.t
```

ENSURES: `toSeq S` returns a sequence containing every element in `S`

3.3 Element Operations

```
insert : t -> Elt.t -> t
```

ENSURES: `insert S x` returns a set `S'` that contains all of the elements of `S` and `x`, $S \cup \{x\}$

```
remove : t -> Elt.t -> t
```

ENSURES: `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, $S \setminus \{x\}$

```
member : t -> Elt.t -> bool
```

ENSURES: `member S x` returns `true` if `x` is a member of `S` and `false` otherwise.

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` $\cup$ `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` $\cap$ `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` $\setminus$ `S2`