Abstract Data Types

An abstract Data Type: A type together with a collection of operations, where

- the representation of values is hidden.

An abstract data type for sets must have:

- Operations to generate sets from the elements. Why?
- Operations to extract the elements of a set. Why?
- Standard operations on sets.

Sets in F#

The `Set` library of F# supports finite sets. An efficient implementation is based on a balanced binary tree.

Examples:

```
set ["Bob"; "Bill"; "Ben"];;
val it : Set<string> = set ["Ben"; "Bill"; "Bob"]

set [3; 1; 9; 5; 7; 9; 1];;
val it : Set<int> = set [1; 3; 5; 7; 9]
```

Equality of two sets is tested in the usual manner:

```
set["Bob";"Bill";"Ben"] = set["Bill";"Ben";"Bill";"Bob"];;
val it : bool = true
```

Sets are order on the basis of a lexicographical ordering:

```
compare (set ["Ann";"Jane"]) (set ["Bill";"Ben";"Bob"]);;
val it : int = -1
```

Selected operations (1)

- `ofList: 'a list -> Set<'a>`,
  where `ofList` $[a_0; \ldots; a_{n-1}] = \{a_0; \ldots; a_{n-1}\}$
- `toList: Set<'a> -> 'a list`,
  where `toList` $\{a_0, \ldots, a_{n-1}\} = [a_0; \ldots; a_{n-1}]$
- `add: 'a -> Set<'a> -> Set<'a>`,
  where `add` $a\ A = \{a\} \cup A$
- `remove: 'a -> Set<'a> -> Set<'a>`,
  where `remove` $a\ A = A \setminus \{a\}$
- `contains: 'a -> Set<'a> -> bool`,
  where `contains` $a\ A = a \in A$
- `minElement: Set<'a> -> 'a)`
  where `minElement` $\{a_0, a_1, \ldots, a_{n-2}, a_{n-1}\} = a_0$ when $n > 0$

Notice that minElement is well-defined due to the ordering:

```
Set.minElement (Set.ofList ["Bob"; "Bill"; "Ben"]);;
val it : string = "Ben"
```

Selected operations (2)

- `union: Set<'a> -> Set<'a> -> Set<'a>`,
  where `union` $A\ B = A \cup B$
- `intersect: Set<'a> -> Set<'a> -> Set<'a>`,
  where `intersect` $A\ B = A \cap B$
- `difference: Set<'a> -> Set<'a> -> Set<'a>`,
  where `difference` $A\ B = A \setminus B$
- `exists: ('a -> bool) -> Set<'a> -> bool`,
  where `exists` $p\ A = \exists x \in A.p(x)$
- `forall: ('a -> bool) -> Set<'a> -> bool`,
  where `forall` $p\ A = \forall x \in A.p(x)$
- `fold: ('a -> 'b -> 'a) -> 'a -> Set<'b> -> 'a`,
  where

$$\begin{aligned}&\texttt{fold}\ f\ a\ \{b_0, b_1, \ldots, b_{n-2}, b_{n-1}\}\\ &= f(f(f(\cdots f(f(a, b_0), b_1), \ldots), b_{n-2}), b_{n-1})\end{aligned}$$

These work similar to their List siblings, e.g.

`Set.fold (-) 0 (set [1; 2; 3])` $= ((0 - 1) - 2) - 3 = -6$

where the ordering is exploited.