Programowanie Funkcyjne 2018

Lista zadań nr 8

5 grudnia 2018

Zadanie 1 (8p). Rozważmy sygnaturę dla funkcyjnych kolejek priorytetowych:

```
module type PQUEUE =
sig
  type priority
  type 'a t

  exception EmptyPQueue

  val empty : 'a t
  val insert : 'a t -> priority -> 'a -> 'a t
  val remove : 'a t -> priority * 'a * 'a t
end
```

1. Zdefiniuj moduł `PQueue : PQUEUE`, przyjmując typ `priority = int`. Reprezentacja kolejki może być dowolna.
2. Wykorzystaj moduł `PQueue` do napisania funkcji sortowania list liczb typu `int`.
3. Uogólnij rozwiązanie punktów 1 i 2 definiując funktor, który dla zadanego modułu OrdType : ORDTYPE zwraca moduł o sygnaturze PQUEUE, gdzie

   ```
   module type ORDTYPE =
   sig
     type t
     type comparison = LT | EQ | GT

     val compare : t -> t -> comparison
   end
   ```

   Zmodyfikuj odpowiednio funkcję sortowania list z p. 2 i przetestuj ją.
4. Przy pomocy *modułów pierwszego rodzaju* zdefiniuj funkcję `sort`, która dla dowolnego modułu implementującego sygnaturę `ORDTYPE` dla pewnego typu i dla dowolnej listy elementów tego typu posortuje listę zgodnie z porządkiem definiowanym przez moduł. Użyj funktora z poprzedniego punktu. Poza modułami pierwszego rodzaju będzie potrzebne jeszcze jedno rozszerzenie języka.

Zadanie 2 (12p). Chcemy stworzyć sygnaturę dla funkcyjnych reprezentacji grafów skierowanych, sparametryzowanych przez abstrakcyjne typy wierzchołków i krawędzi. W tym celu tworzymy sygnaturę dla typu wierzchołków:

```
module type VERTEX =
sig
  type t
  type label

  val equal : t -> t -> bool
  val create : label -> t
  val label : t -> label
end
```

gdzie typ `t` reprezentuje abstrakcyjny typ wierzchołków, typ `label` reprezentuje abstrakcyjny typ etykiet wierzchołków, a funkcja `equal` pozwala porównywać wierzchołki. Funkcja `create` tworzy nowy wierzchołek na podstawie etykiety, a funkcja `label` zwraca etykietę wierzchołka.

1. Napisz analogiczną sygnaturę dla typu krawędzi przyjmując, że krawędzie również mogą być etykietowane, porównywane, oraz dla każdej krawędzi powinna istnieć możliwość wyznaczenia jej wierzchołka początkowego i końcowego.

2. Zaimplementuj moduł `Vertex` spełniający sygnaturę `VERTEX`, a także moduł `Edge` spełniający sygnaturę `EDGE`.

3. Rozważmy następnie sygnaturę dla grafów skierowanych:

```
module type GRAPH =
sig
  (* typ reprezentacji grafu *)
  type t

  module V : VERTEX
  type vertex = V.t

  module E : EDGE with type vertex = vertex

  type edge = E.t

  (* funkcje wyszukiwania *)
  val mem_v : t -> vertex -> bool
  val mem_e : t -> edge -> bool
  val mem_e_v : t -> vertex -> vertex -> bool
  val find_e : t -> vertex -> vertex -> edge
  val succ : t -> vertex -> vertex list
  val pred : t -> vertex -> vertex list
  val succ_e : t -> vertex -> edge list
  val pred_e : t -> vertex -> edge list

  (* funkcje modyfikacji *)
  val empty : t
  val add_e : t -> edge -> t
  val add_v : t -> vertex -> t
  val rem_e : t -> edge -> t
  val rem_v : t -> vertex -> t

  (* iteratory *)
  val fold_v : (vertex -> 'a -> 'a) -> t -> 'a -> 'a
  val fold_e : (edge -> 'a -> 'a) -> t -> 'a -> 'a
  end
```

   Wykorzystując moduły `Vertex` i `Edge` z punktu 2., zaimplementuj moduł `Graph` zgodny z sygnaturą `GRAPH` dla dowolnie wybranej reprezentacji funkcyjnej grafu. (Funkcje `succ` i `pred` wyznaczają odpowiednio listę następników i poprzedników danego wierzchołka, a funkcje `succ_e` i `pred_e` wyznaczają odpowiednio listę krawędzi wychodzących i wchodzących do danego wierzchołka.)

4. Przetestuj działanie swojej implementacji na przykładowych danych.

5. Napisz funktor, który przyjmując jako argumenty moduły `V:VERTEX` oraz `E:EDGE` zwraca moduł zgodny z sygnaturą `GRAPH`.

6. Korzystając z sygnatury `GRAPH` napisz funkcje przechodzenia grafu w głąb i wszerz. Przetestuj te funkcje na swojej implementacji.