Zadanie F: SML - 2-3 drzewa : funktory

2-3 drzewa reprezentujemy w SML-u przez następujący typ danych:

```
datatype 'a node = Two of 'a node * 'a * 'a node |
                   Three of 'a node * 'a * 'a node * 'a * 'a node |
                   Empty;
```

Liście 2-3 drzewa mają się znaleźć na jednym poziomie.

Rozważmy następujące sygnatury

```
signature COMPARABLE= sig
        type t;
        val cmp: t*t -> order
end;

signature DICT= sig
        structure Key:COMPARABLE;

        type 'vt dict;

        val empty: 'vt dict;
        val insert: (Key.t * 'vt) * 'vt dict -> 'vt dict;
        val lookup: (Key.t * 'vt dict) -> 'vt option;
end;

signature OSET= sig
        structure Key:COMPARABLE;

        type oset;

        val empty: oset;
        val insert: Key.t * oset -> oset;
        val member: Key.t * oset -> bool;
end;
```

COMPARABLE specyfikuje sposób porównywania elementów danego typu, zakładamy że implementacja definiuje preporządek liniowy na elementach typu t. DICT to sygnatura dla słowników których klucze można ze sobą porównywać. OSET to sygnatura dla zbiorów których elementy można ze sobą porównywać.

Dla dowolnej struktury C:COMPARABLE słownik oraz zbiór o kluczach (elementach) typu C.t można zaimplementować za pomocą 2-3 drzewa.

Pierwsza część zadania polega na implementacji funktora

```
functor TFrame(structure Spec:SPEC)= struct
        type 'vt frame= 'vt Spec.entryT node;
        val empty = ...
        fun lookup (key,  tree) = ...
        fun insert (entry,  tree) = ...
end;
```

który dla odpowiednich specyfikacji implementujących SPEC wygeneruje implementację słowników i zbiorów za pomocą 2-3 drzew (z dokładnością do nazw funkcji).

Specyfikacja ma spełniać poniższą sygnaturę

```
datatype 'b Propagate= Good of 'b | PropagateUp of 'b;

signature SPEC = sig
        structure Key:COMPARABLE;
        type 'vT entryT;
        type 'vT resultT;


        val extractKey: 'vT entryT -> Key.t;
        val updateE: 'vT entryT node * 'vT entryT -> 'vT entryT node Propagate;
        val lookupE: 'vT entryT option -> 'vT resultT;
end;
```

‘vT entryT to typ którym ma być sparametryzowane drzewo. Wykorzystywane 2-3 drzewo powinno być typu ‘vT entryT node. Czyli obiekty przechowywane w drzewie są typu ‘vT entryR, będziemy je nazywać wpisami.

Funkcja extractKey ma za zadanie wyłuskać klucz (typu Key.t) z elementu typu ‘vT entryT, wartości kluczy determinują ułożenie wpisów w drzewie.

Funkcja updateE definiuje w jaki sposób ma być uaktualniony bieżący wierzchołek drzewa tak aby od tej chwili zawierał wpis przekazany w drugim argumencie. Zakładamy że funkcja updateE będzie aplikowana do drzewa pustego lub do wierzchołka który już zawiera element o kluczu równym kluczowi nowego elementu. Ponieważ modyfikacja pustego drzewa generuje poddrzewo o większej głębokości wynik funkcji jest dodatkowo opatrzony informacją Good/PropagateUp która mówi czy konieczne jest przywrócenie warunku głębokości.

Funkcja lookupE wyciąga informację z danego wpisu. Funkcja ma być aplikowana do wpisów wyszukiwanych w drzewie na podstawie podanego klucza. Ponieważ wpis o danym kluczu może w drzewie nie występować argument jest w postaci ‘vT entryT option. Wartość NONE dla argumentu oznacza że wpis o zadanym kluczu nie został odnaleziony.

Druga część zadania polega na implementacji funktorów

```
functor DSpec (structure KeyS:COMPARABLE):SPEC = struct
        ...
end;


functor SSpec (structure KeyS:COMPARABLE):SPEC = struct
        ...
end;
```

które dla zadanych komparatorów wygenerują specyfikacje odpowiednio dla słownika oraz zbioru tak aby słowniki i zbiory dało się generować następująco

```
functor TDict(structure KeyS:COMPARABLE):>DICT where type Key.t=KeyS.t = struct
        structure Spec:SPEC=DSpec(structure KeyS=KeyS);
        structure Frame= TFrame(structure Spec= Spec);

        structure Key:COMPARABLE=KeyS;
        type 'vt dict= 'vt Frame.frame;

        val empty= Frame.empty;
        val insert= Frame.insert;
        val lookup= Frame.lookup;
end;

functor TSet(structure KeyS:COMPARABLE):>OSET where type Key.t=KeyS.t = struct
        structure Spec:SPEC=SSpec(structure KeyS=KeyS);
        structure Frame= TFrame(structure Spec= Spec);

        structure Key:COMPARABLE=KeyS;
        type oset= unit Frame.frame;

        val empty= Frame.empty;
        val insert= Frame.insert;
        val member= Frame.lookup;
end;
```

Rozwiązanie

Należy przesłać jeden plik zawierający implementację funktorów TFrame, DSpec, SSpec w opisanym powyżej kształcie. W dołączonym pliku template.tar.gz. znajduje się szablon rozwiązania. Rozwiązanie powinno się znaleźć w 'sol-main.sml' kompilacja: mlton sol.mlb .

Przykład

Następujący kod wygeneruje implementację słownika TD o kluczach typu int oraz zbioru TS elementów typu int.

```
structure cInt: COMPARABLE= struct
        type t= int;
        val cmp = Int.compare;
end;

structure TD= TDict(structure KeyS=cInt);
structure TS= TSet(structure KeyS=cInt);
```