Zipper.
Laborator 11

Arbori binari

Exercițiul 0.1. Modificati functiile `change`, `goLeft`, `goRight` si `goUp` pentru a intoarce `Maybe Zipper`.

Exercițiul 0.2. Folositi `>>=` în câteva exemple de înlănțuiri de apeluri de `change`, `goLeft`, `goRight` si `goUp`.

Liste

Exercițiul 0.3. Modificati functiile `change`, `goLeft`, `goRight` si `goUp` pentru a intoarce `Maybe Zipper`.

Exercițiul 0.4. Folositi `>>=` în câteva exemple de înlănțuiri de apeluri de `change`, `goLeft`, `goRight` si `goUp`.

Arbori generali

Fie următoarea structură de date:

```
import Prelude hiding (Left, Right)

data Tree = Node Int [Tree] deriving (Show, Eq)
```

Valorile de tip `Tree` reprezintă arbori în care fiecare nod are un număr arbitrar de subarbori.
Fiecare nod (inclusiv frunzele) are o informație de tip `Int`.
Frunzele sunt reprezentate folosind constructorul `Node` aplicat la lista vidă de subarbori.

Exercițiul 0.5. Proiectati un *zipper* pentru arborii de mai sus.

Indicații pe paginile următoare.

Indicație 1.

```
data Dir = Down | Right deriving (Show, Eq)

t = Node 10 [Node 3 [],
Node 4 [Node 1 [], Node 2 [], Node 22 [], Node 33[] ],
Node 5 [], Node 6 []]

atPos :: [Tree] -> [Dir] -> Int

> atPos [t] [Down, Right, Down, Right, Right]
```

Indicație 2.

```
goDown :: ([Tree], [Crumb]) -> ([Tree], [Crumb])
goRight :: ([Tree], [Crumb]) -> ([Tree], [Crumb])
change :: Int -> ([Tree], [Crumb]) -> ([Tree], [Crumb])
goBack :: ([Tree], [Crumb]) -> ([Tree], [Crumb])
```