Laboratorul 5 - Codări Church

În acest laborator vom implementa Codări Church pentru mai multe tipuri de date.

În cele din urmă, pentru a realiza un interpretor de λ-calcul cu tipuri de date, va trebui să implementați aceste codări direct în λ-calculul pe care l-ați implementat deja.

Dar, pentru moment, și pentru a păstra un mediu cât de cât familiar, cât și pentru a beneficia de sistemul de tipuri foarte puternic al limbajului Haskell, vom implementa aceste codări în Haskell.

De asemeni, pentru ca implementarea fiecărui tip să fie minimală, vom defini niște clase de tipuri conținând această definiție minimală, vom defini o instanță pentru tipul corespunzător din Haskell (pentru a putea testa mai ușor), apoi folosind doar această definiție minimală, veți defini mai multe funții utile folosind acel tip de date. În final, vom defini un tip de date care încapsulează un tip de funcție și veți arăta că acest tip poate fi făcut instanță a clasei de tipuri.

Pentru a ne asigura că nu putem folosi nimic din funcțiile predefinite, nu se importă (aproape) nimic din Prelude, folosind

```
import Prelude ()
```

Mai mult decât atât, tipurile de date din Haskell se importă `qualified` pentru a putea refolosi numele funcțiilor; de asemenea, nu importăm decât minimul necesar pentru a putea defini instanța; ideea fiind de a încerca să definim funcțiile dependente de clasă doar folosind funcțiile specificate în clasă. De exemplu,

```
import qualified Data.Bool as Bool (Bool (..), bool)
```

0. `MyPrelude.hs`

Citiți definițiile din `MyPrelude.hs`; ele redefinesc câteva din funcțiile generale (independente de tipul de date) care erau disponibile în Prelude-ul din Haskell. Aceste funcții le puteți folosi oricând pentru a vă ușura definițiile.

1. `BoolClass.hs`

Toate fișierele următoare au același tipic:

- Introduc o clasă cu operațiile de bază care "definesc" tipul de date, descriind și regulile care le guvernează

```
-- | The class of Boolean-like types (types having a notion of
-- 'true' and 'false' and 'bool'ean choice).
-- Instances should satisfy the following:
--
-- [Bool True] @'bool' f t 'true' = t@
-- [Bool False]  @'bool' f t 'false' = f@
class BoolClass b where
  false :: b
  true :: b
  bool :: a -> a -> b -> a
```

- Urmează o instanță a acestei clase dată de tipul standard corespunzător din Haskell.

```
instance BoolClass Bool.Bool where
  true = Bool.True
  false = Bool.False
  bool = Bool.bool
```

- Urmează o listă de funcții care folosesc tipul respectiv, pe care va trebui să le implementați. Pentru a vă ajuta, funcțiile au comentariile funcțiilor corespunzătoare din Haskell și teste care folosesc instanța tipului standar corespunzător din Haskell.

```
-- | if-then-else
ite :: BoolClass b => b -> a -> a -> a
ite = undefined

-- >>> ite (true :: Bool.Bool) 1 2
-- 1

-- | Boolean "and"
(&&) :: BoolClass b => b -> b -> b
(&&) = undefined

-- >>> true && false :: Bool.Bool
-- False

-- | Boolean "or",
(||) :: BoolClass b => b -> b -> b
(||) = undefined
```

```
-- >>> true || false :: Bool.Bool
-- True

-- | Boolean "not"
not :: BoolClass b => b -> b
not = undefined

-- >>>  not true :: Bool.Bool
-- False
```

- Urmează o definiție a codării Church corespunzătoare tipului de date definit

  ```
  newtype CBool = CBool { getCBool :: forall a. a -> a -> a}
  ```

  Pentru `Bool`, tipul acesta corespunde tipului codărilor Church pentru `true` și `false` definite la curs.

  Pentru a putea însă lucra mai ușor cu acest tip în Haskell, el este încapsulat într-un record.

- Urmează o instanță a clasei pentru tipul reprezentând codarea Church, în care va trebui să implementați constructorii:

  ```
  instance BoolClass CBool where
    true = undefined
    false = undefined
    bool f t b = getCBool b f t
  ```

  In fiecare exemplu, funcția care "agreghează" tipul este definită astfel încât rezultatul să fie "aplicarea" elementului la celelalte argumente ale funcției. Definiți constructorii rămași astfel încât să fie respectate ecuațiile "funcției de agregare" specificate în comentariul de la definiția clasei.

- Urmează o funcție care poate converti între instanțe ale clasei și o instanță a clasei `Show` care convertește valorile codării Church la valorile din tipul intern Haskell pentru a le putea afișa, precum și niște teste pentru a verifica că instanța corespunzătoare codării Church se comportă cum trebuie.

  ```
  -- >>> ite (true :: CBool) 1 2
  -- 1

  -- | converting between different instances of 'BoolClass'
  fromBoolClass :: (BoolClass a, BoolClass b) => a -> b
  fromBoolClass = bool false true
  ```

```
-- | 'Show' instance for 'CBool' (via transformation into Haskell
Bool)
instance Show CBool where
  show cb = "C" <> show (fromBoolClass cb :: Bool.Bool)

-- >>> true && false :: CBool
-- CFalse

-- >>> true || false :: CBool
-- CTrue

-- >>> not true :: CBool
-- CFalse
```

2. `PairClass.hs`

Repetați pașii de mai sus pentru tipul pereche și codarea sa Church.

3. `MaybeClass.hs`

Repetați pașii de mai sus pentru tipul opțiune (`Maybe`) și codarea sa Church.

4. `EitherClass.hs` (optional)

Repetați pașii de mai sus pentru tipul sumă (`Either`) și codarea sa Church.

5. `NatClass.hs`

Repetați pașii de mai sus pentru tipul numerelor naturale și codarea sa Church.

6. `ListClass.hs`

Repetați pașii de mai sus pentru tipul listelor și codarea sa Church.