Structures algébriques / Classes en Haskell

INFO1 - Semaine 48

Guillaume CONNAN

IUT de Nantes - Dpt d'informatique

Dernière mise à jour : 24 novembre 2013 à 23:23

Sommaire

1 Calcul modulaire sans arithmétique...

2 Rationnels

Sommaire

1 Calcul modulaire sans arithmétique...

2 Rationnels

```
Prelude> data Romain = O | I | II | III | IV | V deriving (Enum,
    Bounded, Eq, Show)
```

```
Prelude> minBound :: Romain
O
```

```
Prelude> maxBound :: Romain
V
```

```
Prelude> minBound :: Int
-9223372036854775808
Prelude> maxBound :: Int
9223372036854775807
```

```
Prelude> data Romain = O | I | II | III | IV | V deriving (Enum,
    Bounded, Eq, Show)
```

```
Prelude> minBound :: Romain
O
```

```
Prelude> maxBound :: Romain
V
```

```
Prelude> minBound :: Int
-9223372036854775808
Prelude> maxBound :: Int
9223372036854775807
```

```
Prelude> data Romain = O | I | II | III | IV | V deriving (Enum,
    Bounded, Eq, Show)
```

```
Prelude> minBound :: Romain
O
```

```
Prelude> maxBound :: Romain
V
```

```
Prelude> minBound :: Int
-9223372036854775808
Prelude> maxBound :: Int
9223372036854775807
```

```
Prelude> data Romain = O | I | II | III | IV | V deriving (Enum,
    Bounded, Eq, Show)
```

```
Prelude> minBound :: Romain
O
```

```
Prelude> maxBound :: Romain
V
```

```
Prelude> minBound :: Int
-9223372036854775808
Prelude> maxBound :: Int
9223372036854775807
```

```
Prelude> succ II
III
```

```
Prelude> pred V
IV
```

```
Prelude> succ V
*** Exception: succ{Romain}: tried to take ‘succ’ of last tag in
    enumeration
```

```
Prelude> [II .. V]
[II,III,IV,V]
```

```
Prelude> succ II
III
```

```
Prelude> pred V
IV
```

```
Prelude> succ V
*** Exception: succ{Romain}: tried to take 'succ' of last tag in
    enumeration
```

```
Prelude> [II .. V]
[II,III,IV,V]
```

```
Prelude> succ II
III
```

```
Prelude> pred V
IV
```

```
Prelude> succ V
*** Exception: succ{Romain}: tried to take ‘succ’ of last tag in
    enumeration
```

```
Prelude> [II .. V]
[II,III,IV,V]
```

```
Prelude> succ II
III
```

```
Prelude> pred V
IV
```

```
Prelude> succ V
*** Exception: succ{Romain}: tried to take ‘succ’ of last tag in
    enumeration
```

```
Prelude> [II .. V]
[II,III,IV,V]
```

```
Prelude> :t fromEnum
fromEnum :: Enum a => a -> Int
```

```
Prelude> fromEnum II
2
```

```
Prelude> toEnum 5 :: Romain
V
```

```
Prelude> :t fromEnum
fromEnum :: Enum a => a -> Int
```

```
Prelude> fromEnum II
2
```

```
Prelude> toEnum 5 :: Romain
V
```

```
Prelude> :t fromEnum
fromEnum :: Enum a => a -> Int
```

```
Prelude> fromEnum II
2
```

```
Prelude> toEnum 5 :: Romain
V
```

```haskell
data ClasseMod t = (Eq t, Show t, Enum t, Bounded t) => Zn t
```

```haskell
data LesZ5 = O5 | I5 | II5 | III5 | IV5 deriving (Eq,Show, Bounded,
    Enum)

type Z5 = ClasseMod LesZ5
```

```haskell
instance (Show t) => Show (ClasseMod t) where
  show (Zn x) = show x
```

```haskell
data ClasseMod t = (Eq t, Show t, Enum t, Bounded t) => Zn t
```

```haskell
data LesZ5 = O5 | I5 | II5 | III5 | IV5 deriving (Eq,Show, Bounded,
    Enum)

type Z5 = ClasseMod LesZ5
```

```haskell
instance (Show t) => Show (ClasseMod t) where
  show (Zn x) = show x
```

```haskell
data ClasseMod t = (Eq t, Show t, Enum t, Bounded t) => Zn t
```

```haskell
data LesZ5 = O5 | I5 | II5 | III5 | IV5 deriving (Eq,Show, Bounded,
    Enum)

type Z5 = ClasseMod LesZ5
```

```haskell
instance (Show t) => Show (ClasseMod t) where
  show (Zn x) = show x
```

```
*CalMod> Zn II5
II5
```

```
succZn :: (ClasseMod t) -> (ClasseMod t)
succZn (Zn x)          = if (x == maxBound) then (Zn minBound) else (
    Zn (succ x))
```

```
*CalMod> succZn (Zn III5)
IV5
```

```
*CalMod> succZn (Zn IV5)
05
```

predZn ??

```
succZn :: (ClasseMod t) -> (ClasseMod t)
succZn (Zn x)          = if (x == maxBound) then (Zn minBound) else (
    Zn (succ x))
```

```
*CalMod> succZn (Zn III5)
IV5
```

```
*CalMod> succZn (Zn IV5)
05
```

predZn ??

```
succZn :: (ClasseMod t) -> (ClasseMod t)
succZn (Zn x)          = if (x == maxBound) then (Zn minBound) else (
    Zn (succ x))
```

```
*CalMod> succZn (Zn III5)
IV5
```

```
*CalMod> succZn (Zn IV5)
O5
```

predZn ? ?

```
succZn :: (ClasseMod t) -> (ClasseMod t)
succZn (Zn x)          = if (x == maxBound) then (Zn minBound) else (
    Zn (succ x))
```

```
*CalMod> succZn (Zn III5)
IV5
```

```
*CalMod> succZn (Zn IV5)
O5
```

predZn ? ?

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
```

```
*CalMod> succ (Zn III5)
IV5
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
```

```
*CalMod> succ (Zn III5)
IV5
```

```haskell
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
```

```
*CalMod> pred (Zn III5)
II5
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
```

```
*CalMod> pred (Zn III5)
II5
```

```haskell
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
  fromEnum (Zn x) = fromEnum x
```

```
*CalMod> fromEnum IV5
4
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
  fromEnum (Zn x) = fromEnum x
```

```
*CalMod> fromEnum IV5
4
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
  fromEnum (Zn x) = fromEnum x
  toEnum n = Zn (toEnum n)
```

```
*CalMod> toEnum 4 :: Z5
IV5
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Enum (ClasseMod t)
    where
  succ = succZn
  pred = predZn
  fromEnum (Zn x) = fromEnum x
  toEnum n = Zn (toEnum n)
```

```
*CalMod> toEnum 4 :: Z5
IV5
```

```
instance (Eq t) => Eq (ClasseMod t) where
  (Zn x) == (Zn y) = x == y
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Bounded (ClasseMod t)
    where
  maxBound = Zn (maxBound)
  minBound = Zn (minBound)
```

```
instance (Eq t) => Eq (ClasseMod t) where
  (Zn x) == (Zn y) = x == y
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Bounded (ClasseMod t)
    where
  maxBound = Zn (maxBound)
  minBound = Zn (minBound)
```

```
plusZn  x y =
  if (fromEnum x == 0) then y
  else
    succ (plusZn (pred x) y)
```

Multiplication ? ?

```
plusZn  x y =
  if (fromEnum x == 0) then y
  else
    succ (plusZn (pred x) y)
```

Multiplication ? ?

```
foisZn x y =
  if (fromEnum x == 0) then (toEnum 0)
  else
    if (fromEnum x == 1) then y
    else plusZn y (foisZn (pred x) y)
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Num (ClasseMod t)
    where
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Num (ClasseMod t)
    where
  (+) = plusZn
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Num (ClasseMod t)
    where
  (+) = plusZn
  (*) = foisZn
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Num (ClasseMod t)
    where
  (+) = plusZn
  (*) = foisZn
  fromInteger  = classeOfInt
```

```
instance (Eq t, Show t, Enum t, Bounded t) => Num (ClasseMod t)
    where
  (+) = plusZn
  (*) = foisZn
  fromInteger  = classeOfInt
  negate x =  toEnum (- (fromEnum x) + (fromEnum (maxBound ::
      ClasseMod t)) + 1)
```

```
*CalMod> 2 :: Z5
II5
```

```
*CalMod> 2 :: Z5
II5
*CalMod> 17 :: Z5
II5
```

```
*CalMod> 2 :: Z5
II5
*CalMod> 17 :: Z5
II5
*CalMod> (-3) :: Z5
II5
```

```
*CalMod> 2 :: Z5
II5
*CalMod> 17 :: Z5
II5
*CalMod> (-3) :: Z5
II5
*CalMod> (-3) + 9 :: Z5
I5
```

```
*CalMod> 2 :: Z5
II5
*CalMod> 17 :: Z5
II5
*CalMod> (-3) :: Z5
II5
*CalMod> (-3) + 9 :: Z5
I5
*CalMod> 3 * 7 :: Z5
I5
```

```
*CalMod> 2 :: Z5
II5
*CalMod> 17 :: Z5
II5
*CalMod> (-3) :: Z5
II5
*CalMod> (-3) + 9 :: Z5
I5
*CalMod> 3 * 7 :: Z5
I5
*CalMod> 301 * 7157 :: Z5
II5
```

Sommaire

1 Calcul modulaire sans arithmétique...

2 Rationnels