Функційне програмування мовою Haskell

Типи даних

Користувацькі типи даних

4.2 User-Defined Datatypes

https://www.haskell.org/definition/haskell2010.pdf

1. Алгебраїчні типи даних

   4.2.1 Algebraic Datatype Declarations

   data

2. Новий тип (ізоморфний)

   4.2.3 Datatype Renamings

   newtype

3. Синонімічні типи

   4.2.2 Type Synonym Declarations

   type

Алгебраїчні типи даних

https://www.haskell.org/definition/haskell2010.pdf стор. 40/60

topdecl → ***data*** *[context =>] simpletype [= constrs]* [deriving]*
simpletype → *tycon* $tyvar_1 \dots tyvar_k$ $(k \ge 0)$

data [Контекст] КонструкторТипу [ЗмінніТипу] [= КонструкториДаних] [ЗмінніТипу]

data ConstrType typeVar_1 typeVar_2 ... typeVar_k

= Constructor_1 $\text{typeVar}_{1,1}$ $\text{typeVar}_{1,2}$ ... $\text{typeVar}_{1,m1}$

| Constructor_2 $\text{typeVar}_{2,1}$ $\text{typeVar}_{2,2}$ ... $\text{typeVar}_{2,m2}$

...

| Constructor_p $\text{typeVar}_{p,1}$ $\text{typeVar}_{p,2}$ ... $\text{typeVar}_{p,mp}$

де k>=0, p>=0, m1..mp >=0)

Алгебраїчні типи даних

https://www.haskell.org/definition/haskell2010.pdf стор. 40/60

topdecl → ***data*** *simpletype = constrs [deriving]*
simpletype → *tycon* $tyvar_1 \dots tyvar_k$ *(k ≥ 1)*

data КонструкторТипу [ЗмінніТипу] **=** КонструкториДаних [ЗмінніТипу]

data ConstrType $typeVar_1$ $typeVar_2$ ... $typeVar_k$

= $Constructor_1$ $typeVar_{1,1}$ $typeVar_{1,2}$ ... $typeVar_{1,m1}$

| $Constructor_2$ $typeVar_{2,1}$ $typeVar_{2,2}$ ... $typeVar_{2,m2}$

...

| $Constructor_p$ $typeVar_{p,1}$ $typeVar_{p,2}$ ... $typeVar_{p,mp}$

де k>=0, p>=1, m1..mp >=0)

Алгебраїчні типи даних

https://www.haskell.org/definition/haskell2010.pdf стор. 40/60

topdecl → ***data*** *simpletype = constrs [deriving]*
simpletype → *tycon* $tyvar_1 \ldots tyvar_k$ $(k \geq 1)$

Ім'я/Назва типу

data КонструкторТипу [ЗмінніТипу] **=** КонструкториДаних [ЗмінніТипу]

data ConstrType $typeVar_1$ $typeVar_2$ ... $typeVar_k$

= $Constructor_1$ $typeVar_{1,1}$ $typeVar_{1,2}$ ... $typeVar_{1,m1}$

| $Constructor_2$ $typeVar_{2,1}$ $typeVar_{2,2}$ ... $typeVar_{2,m2}$

...

| $Constructor_p$ $typeVar_{p,1}$ $typeVar_{p,2}$ ... $typeVar_{p,mp}$

де k>=0, p>=1, m1..mp >=0)

Алгебраїчні типи даних

конструктори без параметрів

```
data Color = Red
           | Orange
           | Yellow
           | Green
           | Blue
           | Indigo
           | Violet
```

Алгебраїчні типи даних

конструктори без параметрів

```
data Color = Red
           | Orange
           | Yellow
           | Green
           | Blue
           | Indigo
           | Violet
```

```
*Main> :t Orange
Orange :: Color

*Main> Orange

<interactive>:3:1: error:
    * No instance for (Show Color) arising from a use of `print'
    * In a stmt of an interactive GHCi command: print it
```

Алгебраїчні типи даних

конструктори без параметрів

```
data Color = Red
           | Orange
           | Yellow
           | Green
           | Blue
           | Indigo
           | Violet
            deriving (Show)
```

Алгебраїчні типи даних

конструктори без параметрів

```
data Color = Red
           | Orange
           | Yellow
           | Green
           | Blue
           | Indigo
           | Violet
            deriving (Show)
```

```
*Main> :l dataColor.hs
[1 of 1] Compiling Main             ( dataColor.hs, interpreted )
Ok, 1 module loaded.

*Main> :t Orange
Orange :: Color

*Main> Orange
Orange
```

Алгебраїчні типи даних

конструктори без параметрів

```
 > Blue > Indigo
> Blue == Indigo
 > [Orange ..Indigo]
```

Алгебраїчні типи даних

конструктори без параметрів

```
data Color = Red
           | Orange
           | Yellow
           | Green
           | Blue
           | Indigo
           | Violet
            deriving (Eq, Show, Ord, Enum)
```

Алгебраїчні типи даних

конструктори даних з параметрами

```
data Shape = Ellipse Float Float
           | Square Float
           | Polygon [(Float, Float)]
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Point a = Pnt a a



dist :: Point Double -> Point Double -> Double

dist (Pnt x1 y1)  (Pnt x2 y2) =  sqrt ((x1-x2)^2+(y1-y2)^2)
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Point a = Pnt a a



dist :: Point Double -> Point Double -> Double

dist (Pnt x1 y1)  (Pnt x2 y2) =  sqrt ((x1-x2)^2+(y1-y2)^2)
```

```
> :type Pnt 2 3

Pnt 2 3 :: (Num t) => Point t


> dist ( Pnt 0 0) (Pnt 1 1)

1.4142135623730951
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Point a = Pnt a a

dyst :: Point Double -> Point Double -> Double

dyst (Pnt x1 y1)  (Pnt x2 y2) =  sqrt ((x1-x2)^2+(y1-y2)^2)
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Point a = Pnt a a

dist :: Point Double -> Point Double -> Double

dist (Pnt x1 y1)  (Pnt x2 y2) =  sqrt ((x1-x2)^2+(y1-y2)^2)
```

назва конструктора **може** збігатись з назвою типу

можна було визначити так:

```
data Point a = Point a a

dist :: Point Double -> Point Double -> Double

dist ( Point x1 y1) ( Point x2 y2) = sqrt ((x1-x2)^2+(y1-y2)^2)
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

```
 > head' []
Nothing

> head' [1,2,3]
Just 1

> head' "fgh"
Just 'f'
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
> head "fgh" : "12"
"f12"
```

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                | Nothing
```

```
> head "fgh" : "12"
"f12"
```

head "fgh" : "12" => ‘f’ : "12" => "f12"

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
> head "fgh" : "12"
"f12"
```

head "fgh" : "12" => 'f' : "12" => "f12"

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

```
> head' "fgh" : "12"
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
> head "fgh" : "12"
"f12"
```

head "fgh" : "12" => 'f' : "12" => "f12"

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

```
> head' "fgh" : "12"
<interactive>:25:15: error:
    * Couldn't match type `Char' with `Maybe Char'
      Expected type: [Maybe Char]
        Actual type: [Char]
```

Алгебраїчні типи даних

конструктори з параметрами

```
data Maybe a =    Just a
                  | Nothing
```

```
> head "fgh" : "12"
"f12"
```

head "fgh" : "12" => 'f' : "12" => "f12"

```
head':: [a]->Maybe a

head' [ ]=Nothing

head' (x:_) = Just x
```

```
> head' "fgh" : "12"
<interactive>:25:15: error:
    * Couldn't match type `Char' with `Maybe Char'
      Expected type: [Maybe Char]
        Actual type: [Char]
```

head’ "fgh" : "12" => Just 'f' : "12" => ⊥

Алгебраїчні типи даних

конструктори з параметрами

```
fromMaybe::(Num a)=>Maybe a -> a
fromMaybe Nothing = 0
fromMaybe (Just x) = x
```

```
> fromMaybe (Just 2)
2

> fromMaybe (head' [1,2,3,4])
1

> 2 + fromMaybe (head' [1,2,3,4])
3

> 2 + fromMaybe (head' [])
2
```

Алгебраїчні типи даних

конструктори з параметрами

```
fromMaybe2 :: Maybe Char -> Char
fromMaybe2 Nothing = ' '
fromMaybe2 (Just x) = x
```

```
> head' "fgh"
Just 'f'

> fromMaybe2 (head' "fgh")
'f'

> fromMaybe2 (head' "fgh") : "123"
"f123"

> fromMaybe2 (head' "") : "123"
" 123"
```

Алгебраїчні типи даних

конструктори з параметрами

```
fromMaybe2 :: Maybe Char -> Char
fromMaybe2 Nothing = ' '
fromMaybe2 (Just x) = x
```

```
fromMaybe::(Num a)=>Maybe a -> a
fromMaybe Nothing = 0
fromMaybe (Just x) = x
```

Prelude містить функцію *maybe*

```
maybe :: b -> (a -> b) -> Maybe a -> b
maybe n _ Nothing  = n
maybe _ f (Just x) = f x
```

Алгебраїчні типи даних

конструктори з параметрами

Prelude містить функцію *maybe*

```
maybe :: b -> (a -> b) -> Maybe a -> b
maybe n _ Nothing  = n
maybe _ f (Just x) = f x
```

```
> maybe 0 (*3) (Just 2)
6

> maybe 0 (*3)  Nothing
0

> maybe "" show (Just 5)
"5"

> maybe "" show  Nothing
""
```

Алгебраїчні типи даних

конструктори з параметрами

Prelude містить функцію *maybe*

```
maybe :: b -> (a -> b) -> Maybe a -> b
maybe n _ Nothing  = n
maybe _ f (Just x) = f x
```

```
> maybe 0 (*3) (Just 2)
6

> maybe 0 (*3)  Nothing
0

> maybe "" show (Just 5)
"5"

> maybe "" show  Nothing
""
```

```
> maybe 0 (*1)  Nothing
0

> maybe 0 (*1) (Just 2)
2

> maybe 0 id (Just 2)
2

> maybe 0 id Nothing
0
```

Алгебраїчні типи даних

Іменовані поля

```
data Configuration =
       ConsConfig{
         userName :: String,
         localHost :: String,
         remoteHost :: String,
         isGuest :: Bool,
         isSuperuser :: Bool,
         currentDirectory :: String,
         homeDirectory :: String,
         timeConnected :: Integer
         } deriving (Eq, Show)
```

Алгебраїчні типи даних

Іменовані поля

```
data Configuration =
       ConsConfig{
         userName :: String,
         localHost :: String,
         remoteHost :: String,
         isGuest :: Bool,
         isSuperuser :: Bool,
         currentDirectory :: String,
         homeDirectory :: String,
         timeConnected :: Integer
         } deriving (Eq, Show)
```

```
myconf :: Configuration
myconf = ConsConfig {
                  userName = "User1",
                  localHost = "myComp",
                  remoteHost = "asd12",
                  isGuest = False,
                  isSuperuser = False,
                  currentDirectory = "curUser1",
                  homeDirectory = "hmUser1",
                  timeConnected = 0
                  }
```

Алгебраїчні типи даних

Іменовані поля

```
data Configuration =
        ConsConfig{
          userName :: String,
          localHost :: String,
          remoteHost :: String,
          isGuest :: Bool,
          isSuperuser :: Bool,
          currentDirectory :: String,
          homeDirectory :: String,
          timeConnected :: Integer
          } deriving (Eq, Show)


> :t userName
userName :: Configuration -> String

> :t isSuperuser
isSuperuser :: Configuration -> Bool

> :t timeConnected
timeConnected :: Configuration -> Integer
```

```
myconf :: Configuration
myconf = ConsConfig {
                  userName = "User1",
                  localHost = "myComp",
                  remoteHost = "asd12",
                  isGuest = False,
                  isSuperuser = False,
                  currentDirectory = "curUser1",
                  homeDirectory = "hmUser1",
                  timeConnected = 0
```

Алгебраїчні типи даних

Іменовані поля

```
data Configuration =
        ConsConfig{
          userName :: String,
          localHost :: String,
          remoteHost :: String,
          isGuest :: Bool,
          isSuperuser :: Bool,
          currentDirectory :: String,
          homeDirectory :: String,
          timeConnected :: Integer
          } deriving (Eq, Show)
```

```
myconf :: Configuration
myconf = ConsConfig{
                  userName = "User1",
                  localHost = "myComp",
                  remoteHost = "asd12",
                  isGuest = False,
                  isSuperuser = False,
                  currentDirectory = "curUser1",
                  homeDirectory = "hmUser1",
                  timeConnected = 0
```

```
> :t userName
userName :: Configuration -> String

> :t isSuperuser
isSuperuser :: Configuration -> Bool

> :t timeConnected
timeConnected :: Configuration -> Integer
```

```
> userName myconf
"User1"

> isSuperuser myconf
False

> timeConnected myconf
```

Новий тип (ізоморфний)

4.2.3 Datatype Renamings
https://www.haskell.org/definition/haskell2010.pdf стор. 43/63

Єдиний конструктор з єдиним параметром

topdecl → ***newtype*** *[context =>] TypeName typeVars= Constr oneType [deriving]*

Новий тип (ізоморфний)

4.2.3 Datatype Renamings

https://www.haskell.org/definition/haskell2010.pdf стор. 43/63

Єдиний конструктор з єдиним параметром

topdecl → ***newtype*** *TypeName typeVars= Constr oneType [deriving]*

newtype MyInt = ConMyInt Int

Новий тип (ізоморфний)

4.2.3 Datatype Renamings
https://www.haskell.org/definition/haskell2010.pdf стор. 43/63

Єдиний конструктор з єдиним параметром

topdecl → newtype TypeName typeVars= Constr oneType [deriving]

```
newtype MyInt = ConMyInt Int


instance Eq MyInt where
    ConMyInt i == ConMyInt j
        | odd i && odd j = i == j
        | otherwise = False
```

Новий тип (ізоморфний)

4.2.3 Datatype Renamings
https://www.haskell.org/definition/haskell2010.pdf стор. 43/63

Єдиний конструктор з єдиним параметром

topdecl → newtype TypeName typeVars= Constr oneType [deriving]

```
newtype MyInt = ConMyInt Int


instance Eq MyInt where
    ConMyInt i == ConMyInt j
        | odd i && odd j = i == j
        | otherwise = False
```

```
> ConMyInt 2 == ConMyInt 2
False

> ConMyInt 3 == ConMyInt 3
True

> ConMyInt 3 == ConMyInt 5
False
```

Новий тип (ізоморфний)

4.2.3 Datatype Renamings
https://www.haskell.org/definition/haskell2010.pdf стор. 43/63

Єдиний конструктор з єдиним параметром

```
–– можна визначити і інші відношення
instance Ord MyInt where
     ConMyInt i < ConMyInt j
      | odd' i && odd' j = i < j
      | even' i && even' j = i < j
      | even' i = True
      | otherwise = False
            where odd' x = (x `mod` 2) == 1
                  even'  = not . odd'
```

Типи-синоніми

4.2.2 Type Synonym Declarations
https://www.haskell.org/definition/haskell2010.pdf стор. 42/62

topdecl → **type** simpletype = type
simpletype → tycon tyvar$_1$ . . . tyvar$_k$
(k ≥ 0)

Типи-синоніми

4.2.2 Type Synonym Declarations

https://www.haskell.org/definition/haskell2010.pdf стор. 42/62

type Position = (Float,Float)
type Angle = Float
type Distance = Float

Типи-синоніми

4.2.2 Type Synonym Declarations

https://www.haskell.org/definition/haskell2010.pdf стор. 42/62

```
type Position = (Float,Float)
type Angle = Float
type Distance = Float

move :: Distance->Angle->Position->Position
move d alpha (x0,y0) = (x0+d*cos(alpha),y0+d*sin(alpha))
```

Типи-синоніми

4.2.2 Type Synonym Declarations
https://www.haskell.org/definition/haskell2010.pdf стор. 42/62

```
type Position = (Float,Float)
type Angle = Float
type Distance = Float

move :: Distance->Angle->Position->Position
move d alpha (x0,y0) = (x0+d*cos(alpha),y0+d*sin(alpha))

>move 3 0.3 (0,0)
(2.8660095,0.8865607)
```