Logic Haskell Exercises

Young W. Lim

2018-09-11 Tue

Outline

1. Based on

2. Logic
   - Using TAMO.hs

Based on

"The Haskell Road to Logic, Maths, and Programming",
K. Doets and J. V. Eijck

Using TAMO.hs

```
module TAMO

where
```

```
:load TAMO
```

Quantifiers as Functions

```
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude> :load TAMO
[1 of 1] Compiling TAMO             ( TAMO.hs, interpreted )
Ok, modules loaded: TAMO.
*TAMO> any (>3) [0 ..]
True
*TAMO> any (<3) [0 ..]
True
*TAMO> any (<-1) [0 ..]

<interactive>:5:6:
    parse error on input ‘<-’
    Perhaps this statement should be within a ’do’ block?
*TAMO> any (< -1) [0 ..]

^CInterrupted. (takes too long)
```

Predefined Connectives

- Datatype Bool

```
data Bool = False | True
```

- Negation (not)

```
not :: Bool -> Bool
not True = False
not False = True
```

- Conjunction (&&)

```
(&&) :: Bool -> Bool -> Bool
False && x = False
True && x = x
```

- Disjunction (||)

```
(||) :: Bool -> Bool -> Bool
False && x = x
True && x = True
```

Connectives Definition

- Implication (==>)

```
(==>) :: Bool -> Bool -> Bool~
 x ==> y = (not x) || y~
```

- Equivalence (<=>)

```
(<=>) :: Bool -> Bool -> Bool
x <=> y = x == y
```

- Exclusive or (<+>)

```
(<+>) :: Bool -> Bool -> Bool
x <+> y = x /= y
```

Fixity Declaration in Haskell

- `infix` non-associativity
- `infixl` left-associativity
- `infixr` right-associativity
- specifies a a precedence level from 0 to 9
  - 0 (weakest)
  - 9 (strongest)
  - 10 (normal application)

`http://zvon.org/other/haskell/Outputsyntax/fixityQdeclaration_reference.html`

Fixity Declaration Example

```
main = print (1 +++ 2 *** 3)

infixr  6 +++
infixr  7 ***,///

(+++) :: Int -> Int -> Int
a +++ b = a + 2*b

(***) :: Int -> Int -> Int
a *** b = a - 4*b

(///) :: Int -> Int -> Int
a /// b = 2*a - 3*b


(1 +++ (2 *** 3)) = (1 +++ (2 - 4*3)) = (1 +++ -10) = 1 + 2*(-10) = -19

http://zvon.org/other/haskell/Outputsyntax/fixityQdeclaration_reference.html
```

Fixity of Connectives Definition

- `infix  1 ==>`
- `infix  1 <=>`
- `infixr 2 <+>`

Conncectives Examples

```
*Main> True ==> True
True
*Main> True ==> False
False
*Main> False ==> True
True
*Main> False ==> False
True
----------------------
*Main> (==>) True True
True
*Main> (==>) True False
False
*Main> (==>) False True
True
*Main> (==>) False False
True
```

```
*Main> True <=> True
True
*Main> True <=> False
False
*Main> False <=> True
False
*Main> False <=> False
True
----------------------
*Main> (<=>) True True
True
*Main> (<=>) True False
False
*Main> (<=>) False True
False
*Main> (<=>) False False
True
```

```
*Main> True <+> True
False
*Main> True <+> False
True
*Main> False <+> True
True
*Main> False <+> False
False
------------------------
*Main> (<+>) True True
False
*Main> (<+>) True False
True
*Main> (<+>) False True
True
*Main> (<+>) False False
False
```

Evaluating Conncectives

```
P=True, Q=False

~ P ^ ((P -> Q) <-> ~(Q ^ ~ P))
: T :   T  : F   :  : F : : T
F   :      F     :  :   : F
    :            :  :   F
    :            :  T
    :            F
    F

~ P ^ ((P -> Q) <-> ~(Q ^ ~ P))
F T F   T F  F   F  T F F F T
```

No Multiple Declarations in Haskell

- in ghci, can change the value of a binding

```
P=True
Q=False

P=False
Q=True
```

- in ghc, cannot change the value of a binding

```
Error
Multiple declarations of 'P'
Multiple declarations of 'Q'
```

Evaluating Conncectives in Haskell

```
p = True
q = False

formula1 = (not p) && (p ==> q) <=> not (q && (not p))
                 T      T     F           F         T
             F             F                 F
                   F                  T
                                F

formula2 p q = ((not p) && (p ==> q) <=> not (q && (not p)))
```

```
*Main> formula1
False
*Main> formula2 True True
False
*Main> formula2 True False
False
*Main> formula2 False True
False
*Main> formula2 False False
True
```