

Logic Haskell Exercises

Young W. Lim

2018-09-11 Tue

- 1 Based on
- 2 Logic
 - Using TAMO.hs

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

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

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)
```

- 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
```

- Implication (\implies)

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

- Equivalence (\iff)

```
(<=>) :: 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 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 ***,///
```

```
(+++)  
a +++ b = a + 2*b
```

```
(***)  
a *** b = a - 4*b
```

```
(///)  
a /// b = 2*a - 3*b
```

$(1 \text{ +++ } (2 \text{ *** } 3)) = (1 \text{ +++ } (2 - 4*3)) = (1 \text{ +++ } -10) = 1 + 2*(-10) = -19$

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

Fixity of Connectives Definition

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

Connectives Examples

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

Evaluating Connectives

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 Connectives in Haskell

```
p = True
q = False
```

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