

Applicative (2A)

Copyright (c) 2016 - 2017 Young W. Lim.

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".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

<http://learnyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>

<http://learnyouahaskell.com/functors-applicative-functors-and-monoids>

Haskell in 5 steps

https://wiki.haskell.org/Haskell_in_5_steps

Currying

Currying recursively transforms
a function that takes multiple arguments
into a function that takes just a single argument and
returns another function if any arguments are still needed.

$f :: a \rightarrow b \rightarrow c$

$f\ x\ y$

$f :: a \rightarrow b \rightarrow c$

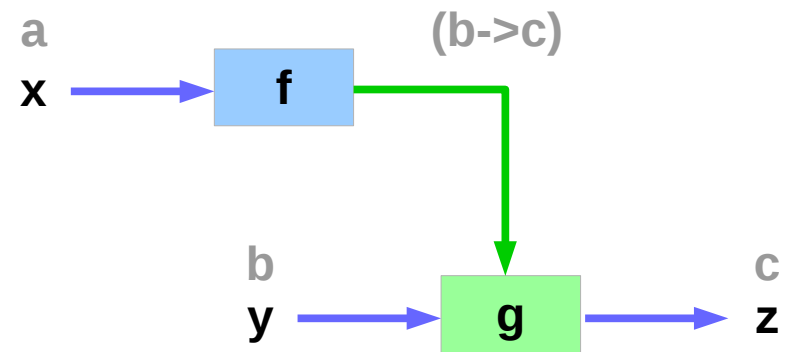
$(f\ x)\ y$

$f :: a \rightarrow (b \rightarrow c)$

$g\ y$

$g :: b \rightarrow c$

$f :: a \rightarrow b \rightarrow c$



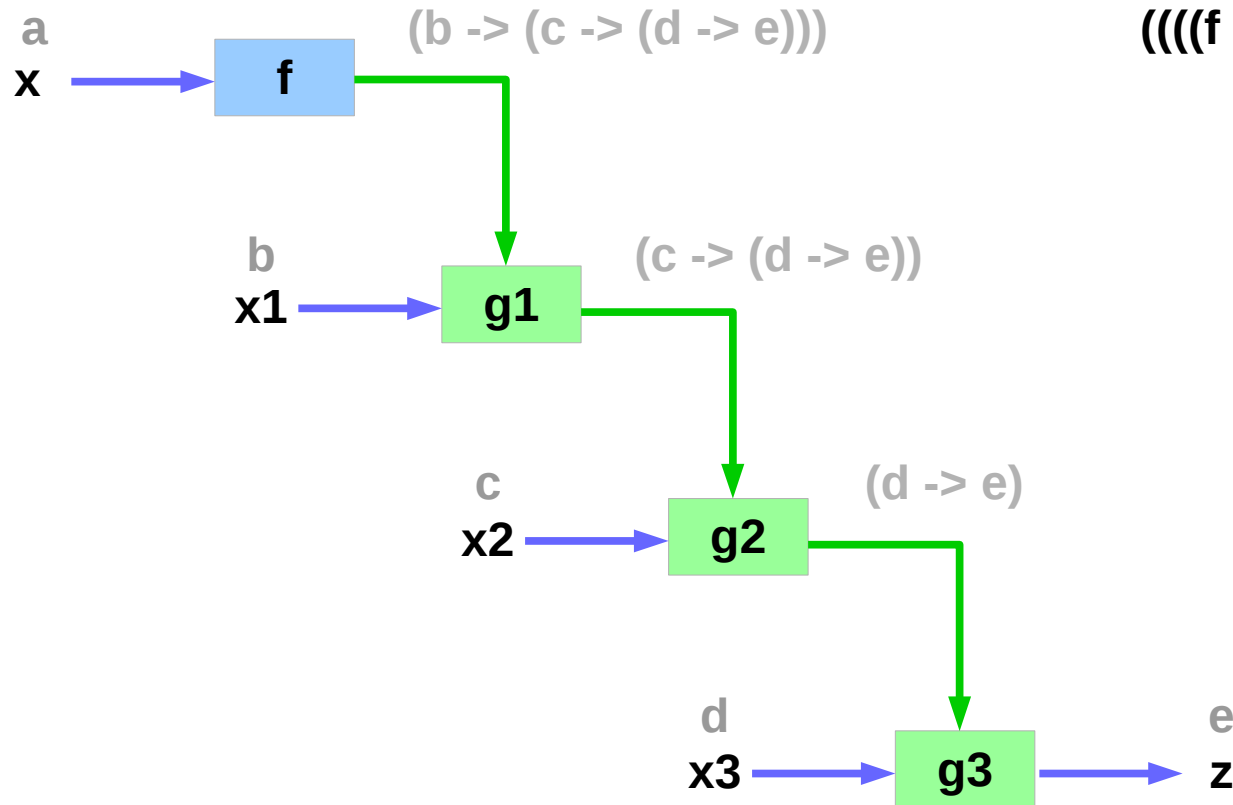
<https://wiki.haskell.org/Currying>

<http://learnyouahaskell.com/functors-applicative-functors-and-monoids>

Currying Examples

$f :: a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$

$f :: a \rightarrow (b \rightarrow (c \rightarrow (d \rightarrow e)))$



$(((((f\ x)\ x1)\ x2)\ x3))$

<http://learnyouahaskell.com/functors-applicative-functors-and-monoids>

Curry & Uncurry

$f :: a \rightarrow b \rightarrow c$ the curried form of $g :: (a, b) \rightarrow c$

$f = \text{curry } g$
 $g = \text{uncurry } f$

$f \ x \ y = g \ (x,y)$

the curried form is usually more convenient because it allows **partial application**.

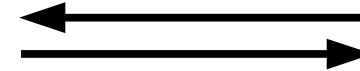
all functions are considered **curried**

all functions take **just one argument**

the curried form

$f :: a \rightarrow b \rightarrow c$

currying



$g :: (a, b) \rightarrow c$

uncurrying

$f \ x \ y$

$g \ (x,y)$

<https://wiki.haskell.org/Currying>

<https://www.schoolofhaskell.com/user/EFulmer/currying-and-partial-application>

<http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/>

Curry & Uncurry

$f :: a \rightarrow b \rightarrow c$ the curried form of $g :: (a, b) \rightarrow c$

$f = \text{curry } g$
 $g = \text{uncurry } f$

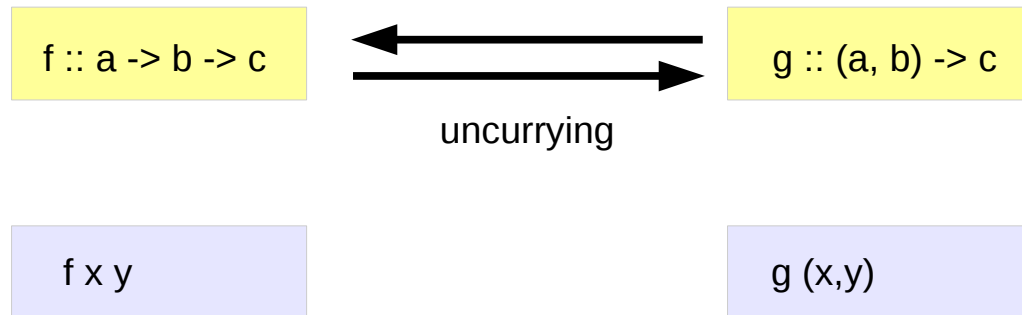
$f \ x \ y = g \ (x,y)$

the curried form is usually more convenient because it allows **partial application**.

all functions are considered **curried**

all functions take **just one argument**

the curried form



<https://wiki.haskell.org/Currying>

Curry & Uncurry

$f :: a \rightarrow b \rightarrow c$ the curried form of $g :: (a, b) \rightarrow c$

$f = \text{curry } g$
 $g = \text{uncurry } f$

$f \ x \ y = g \ (x,y)$

the curried form is usually more convenient because it allows **partial application**.

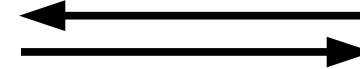
all functions are considered **curried**

all functions take **just one argument**

the curried form

$f :: a \rightarrow b \rightarrow c$

currying



$g :: (a, b) \rightarrow c$

uncurrying

$f \ x \ y$

$g \ (x,y)$

<https://wiki.haskell.org/Currying>

References

- [1] <ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf>
- [2] <https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf>