Type introduction illustrated

for casual haskellers

to get over the Foldable

Takenobu T.

Rev. 0.01.2
WIP

"What is this description ?!"

```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```

NOTE

- This document shows one of the mental model.
- Please see also references.
- This is written for Haskell, especially ghc7.10/8.0 and later.

Contents

1. Introduction

1. Introduction

Values, Types, Type classes

Values

'a'

False

1.0

1

2

'h'

1.5

'5'

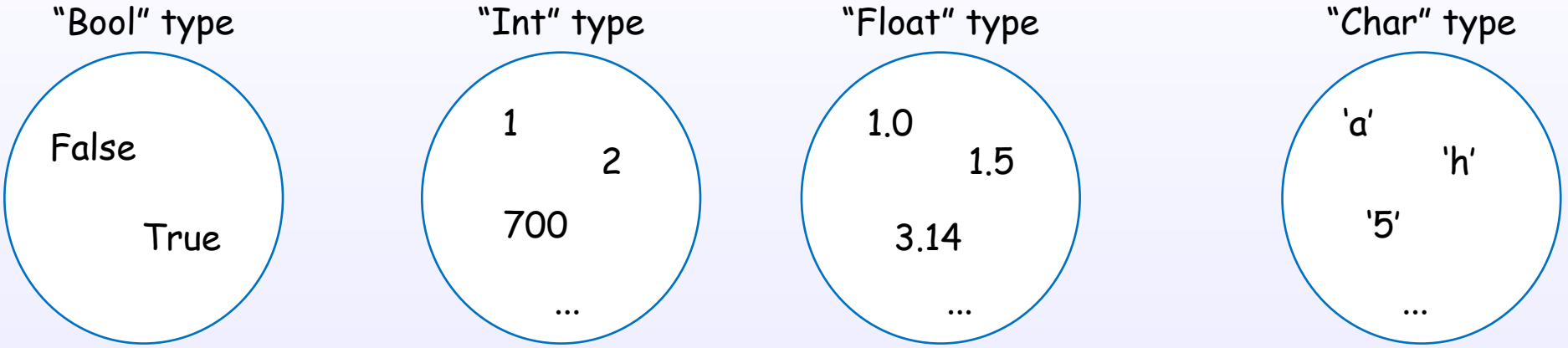
True

700

3.14

References : [B2] Ch.2, [B3] Ch.2

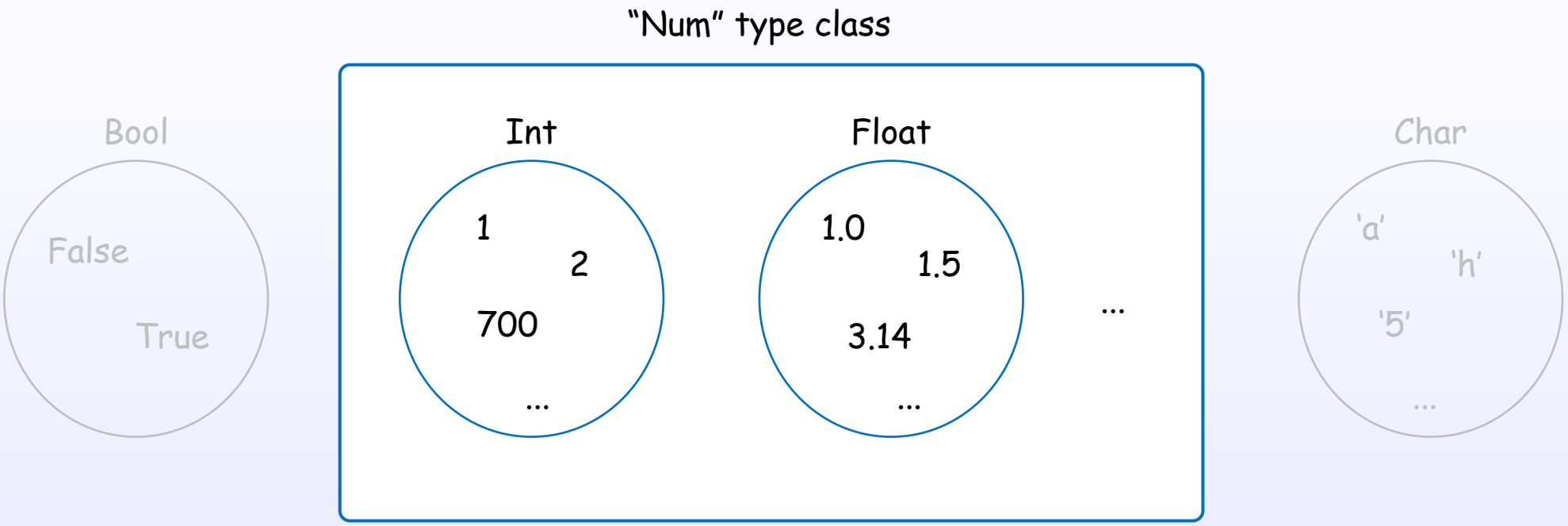
Types



A type is a collection of values which have common property.

References : [B2] Ch.2, [B3] Ch.2, [B1] Ch.2, [D2], [B5] Ch.8, [H1] Ch.4

Type classes



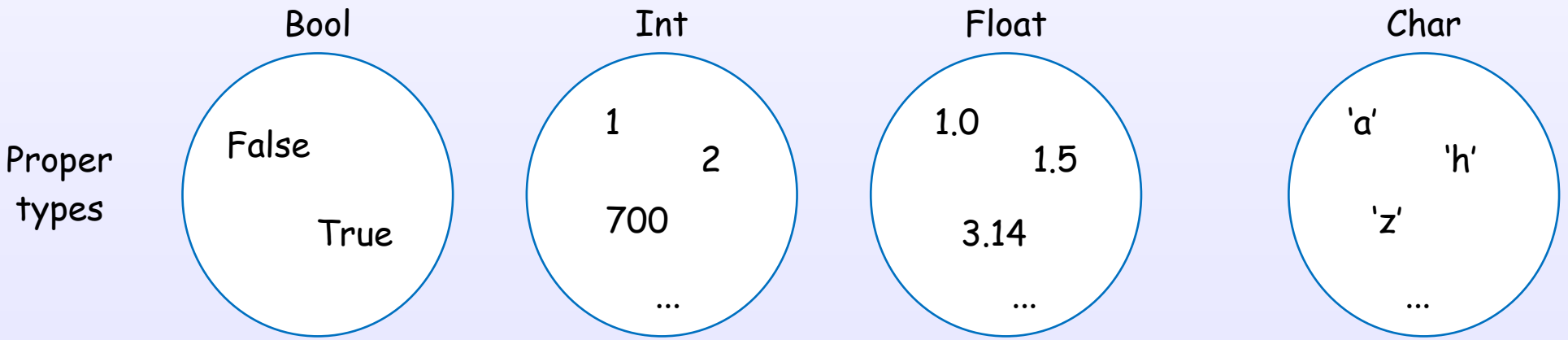
A type class is a collection of types which have common operations.

References : [B1] Ch.2, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6, [H1] Ch.4

1. Introduction

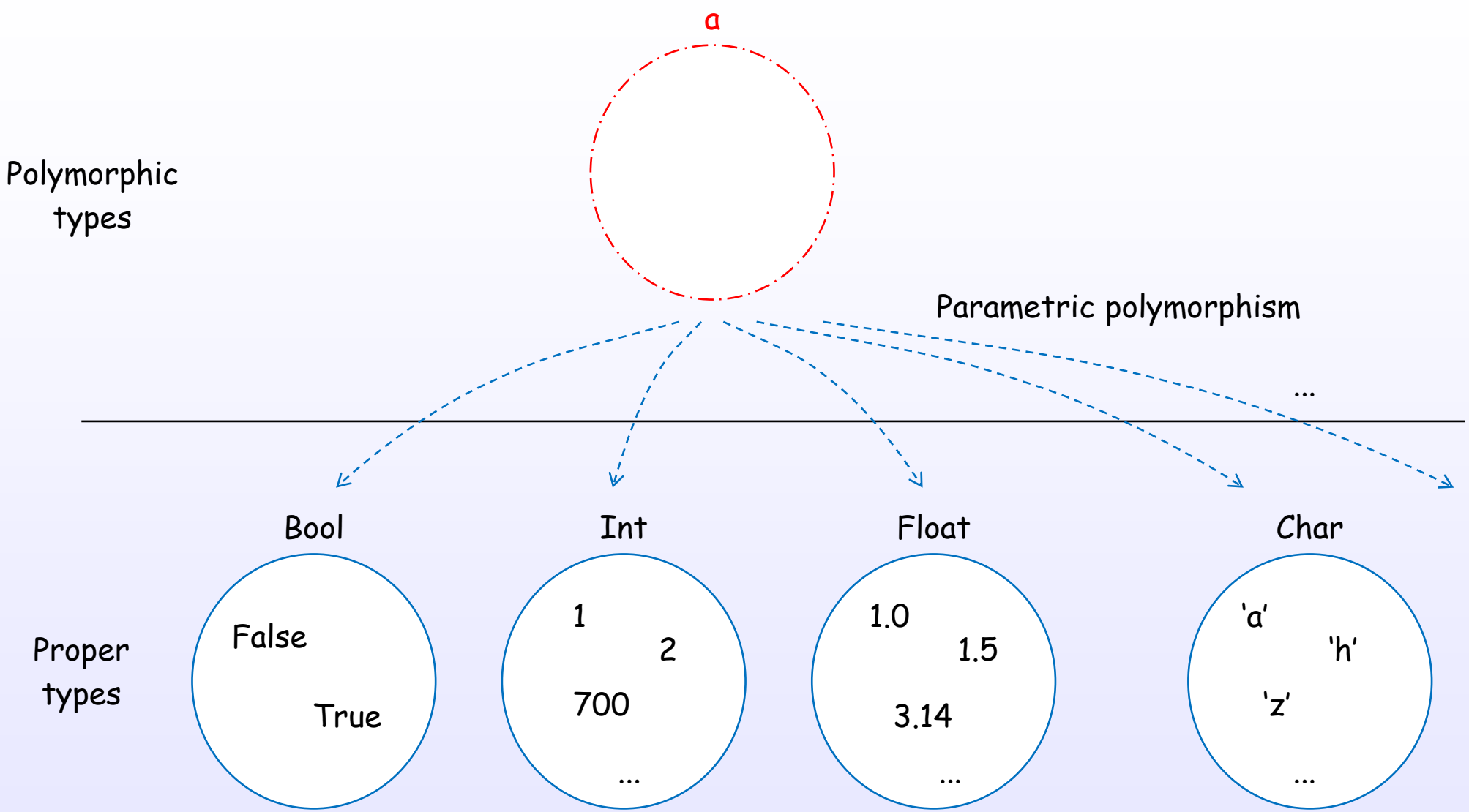
Polymorphic types

Proper types



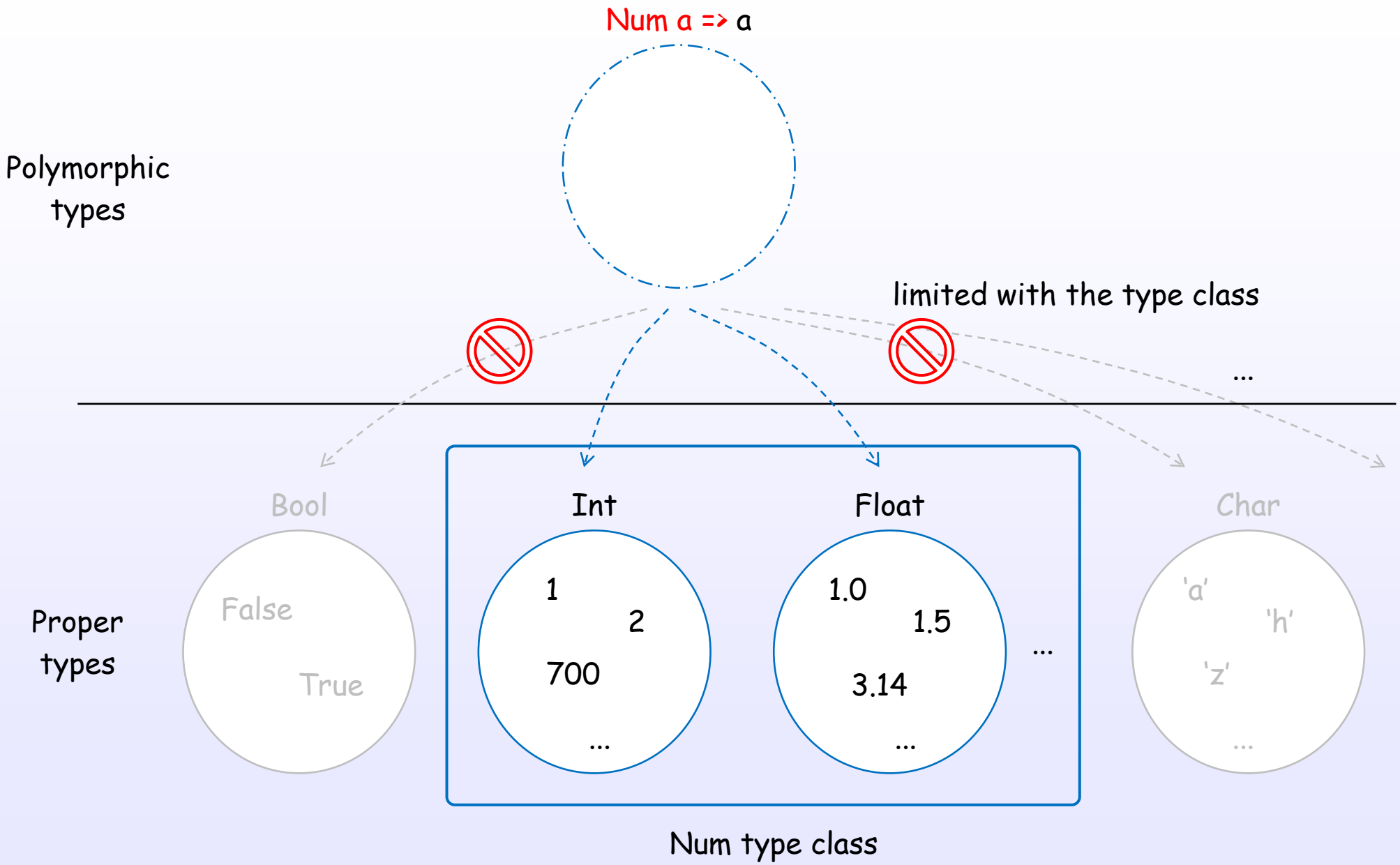
References : [B1] Ch.2, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Polymorphic types



References : [B1] Ch.7, [B3] Ch.3, [B4] Ch.6, [D1] Week 2, [H1] Ch.4

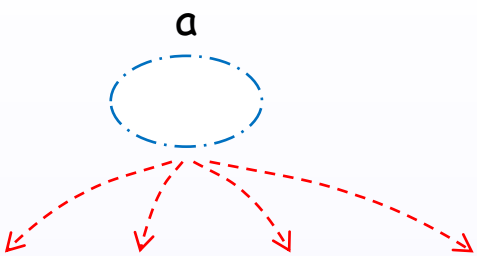
Polymorphic types restricted with type classes



References : [B1] Ch.2, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6, [D1] Week 4

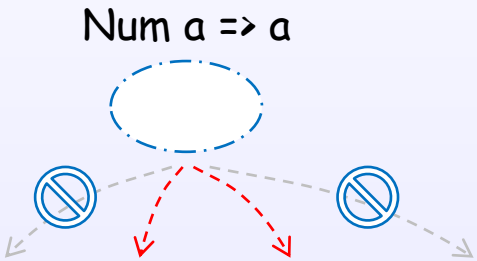
Polymorphic types

Polymorphic types

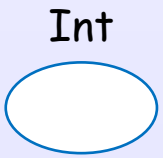


---

polymorphic types with type class



Proper types



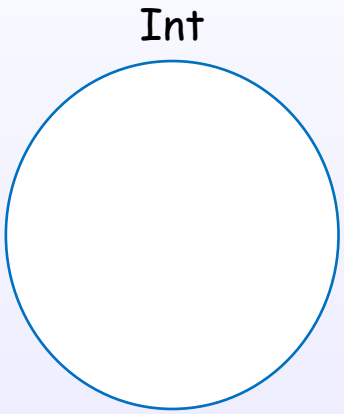
References : [B1] Ch.2, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6, [D1] Week 2, 4

1. Introduction

Type constructors

Type constructors

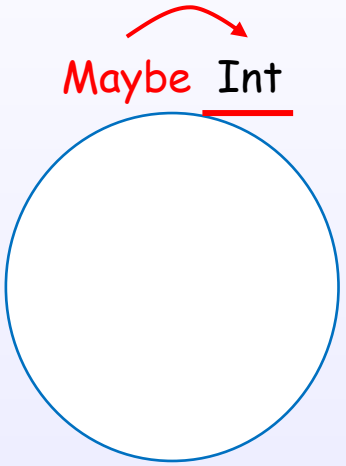
nullary type constructor



References : [B1] Ch.7

Type constructors

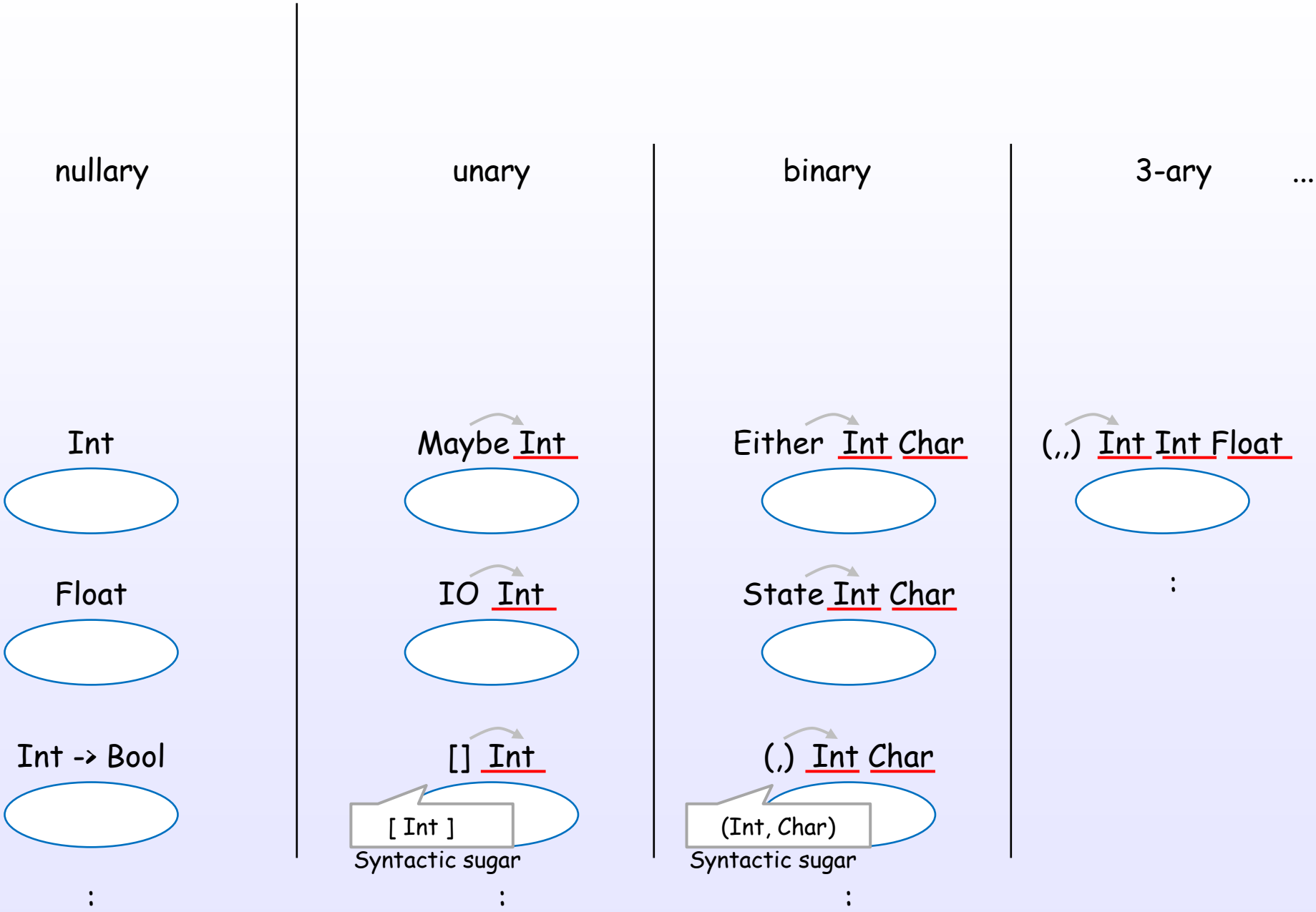
unary type constructor



"Maybe" type constructor takes one type argument (unary).

References : [B1] Ch.7

Type constructors

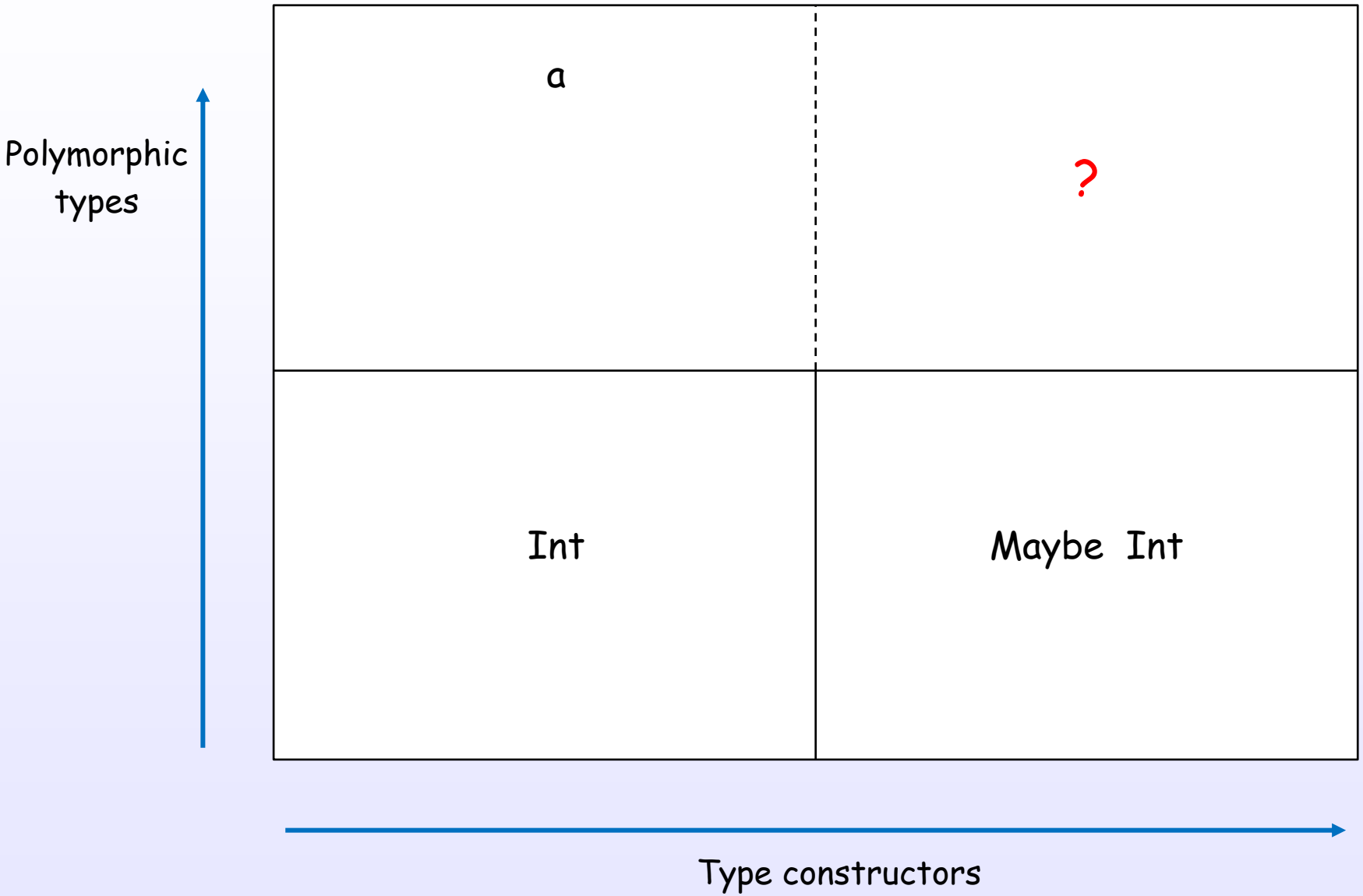


References : [B1] Ch.7, [H1] Ch.4

1. Introduction

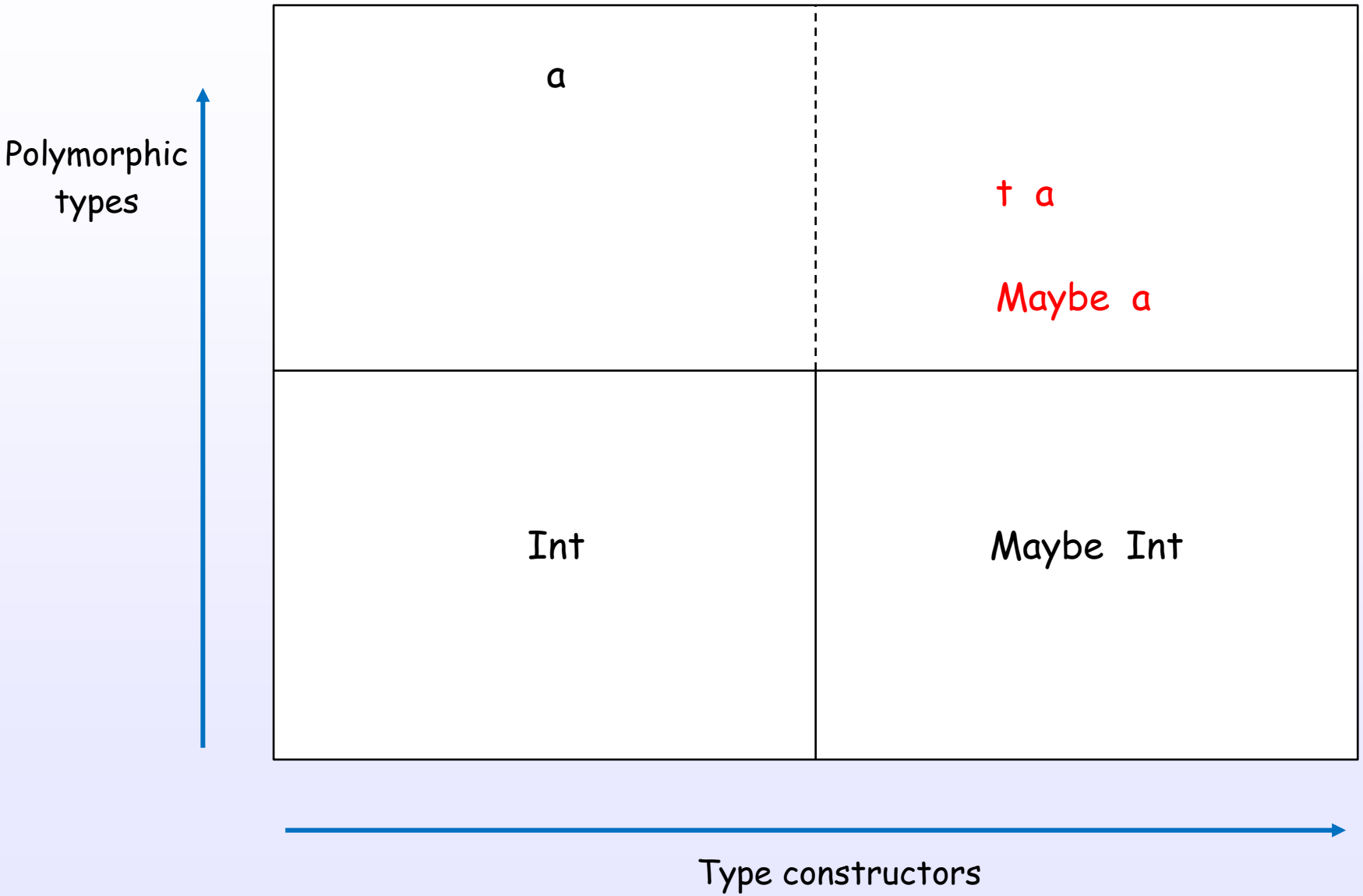
Polymorphic types and type constructors

Polymorphic types and type constructors



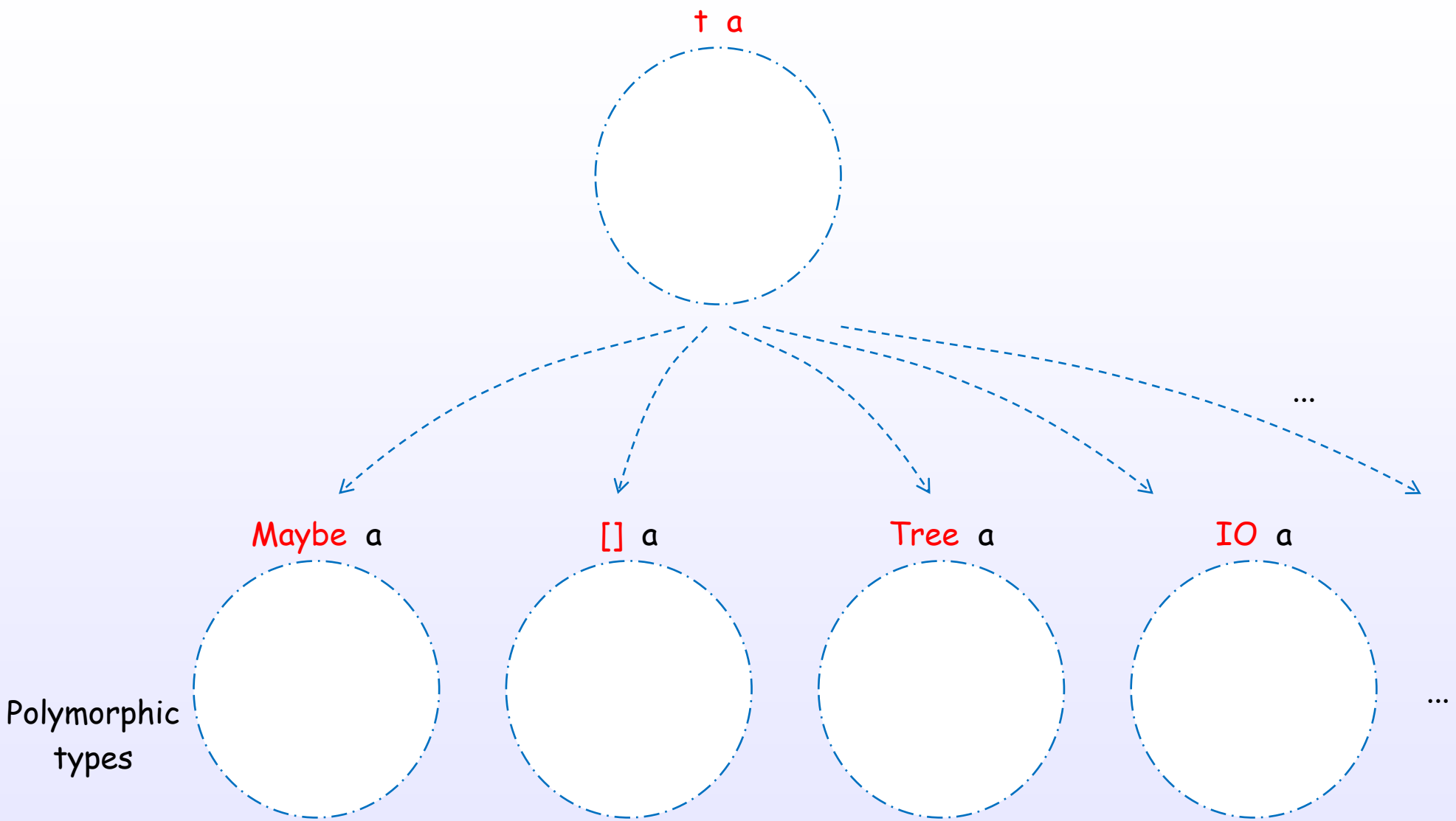
References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Polymorphic types and type constructors



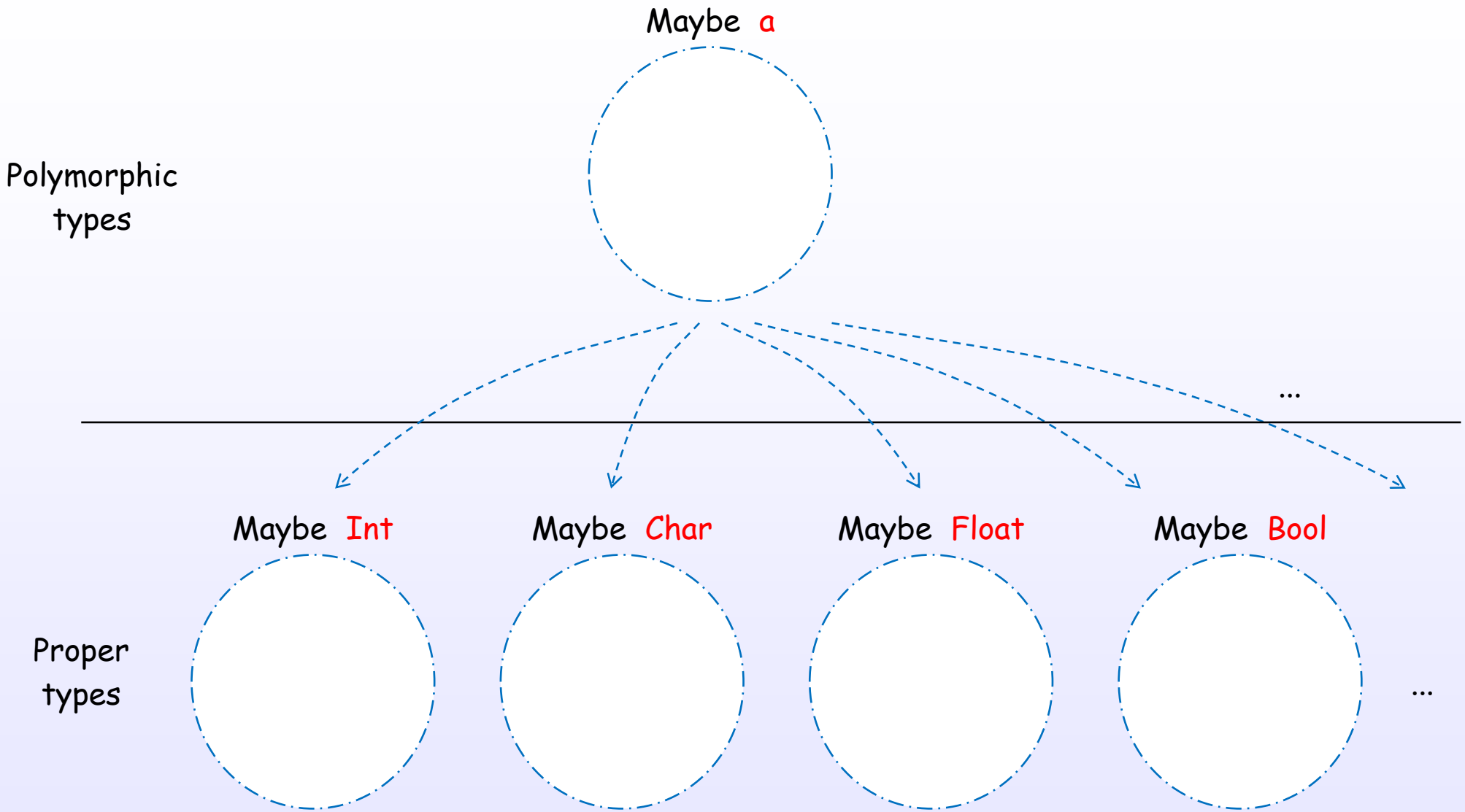
References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Polymorphic types and type constructors



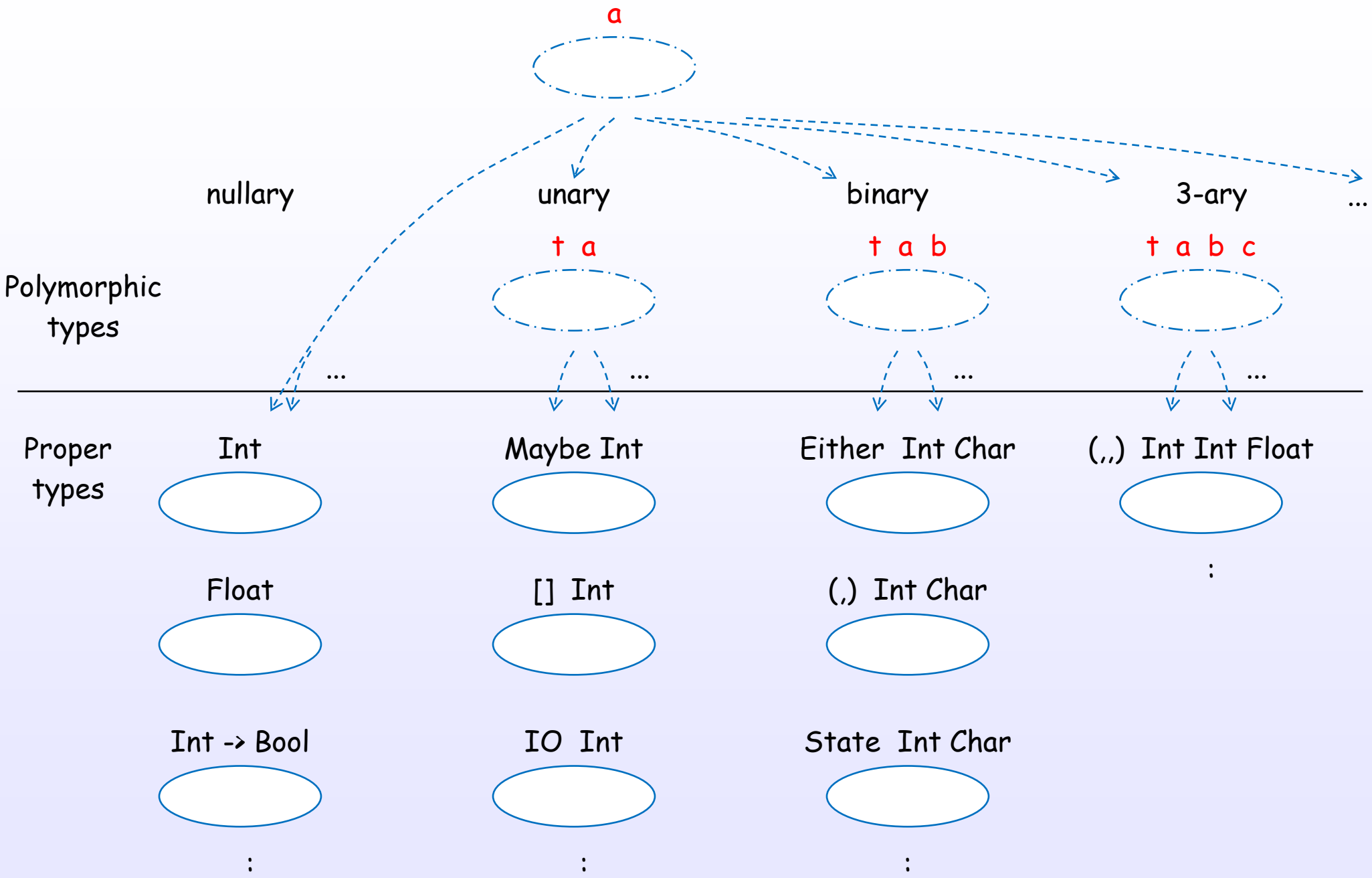
References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Polymorphic types and type constructors



References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Polymorphic types and type constructors



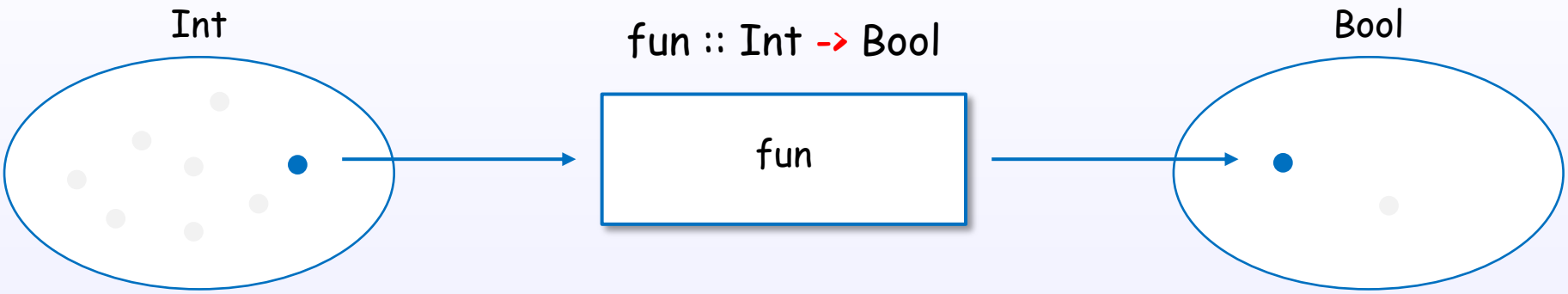
References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

2. more, Types and Type classes

2. more, Types and Type classes

Function types

Function type



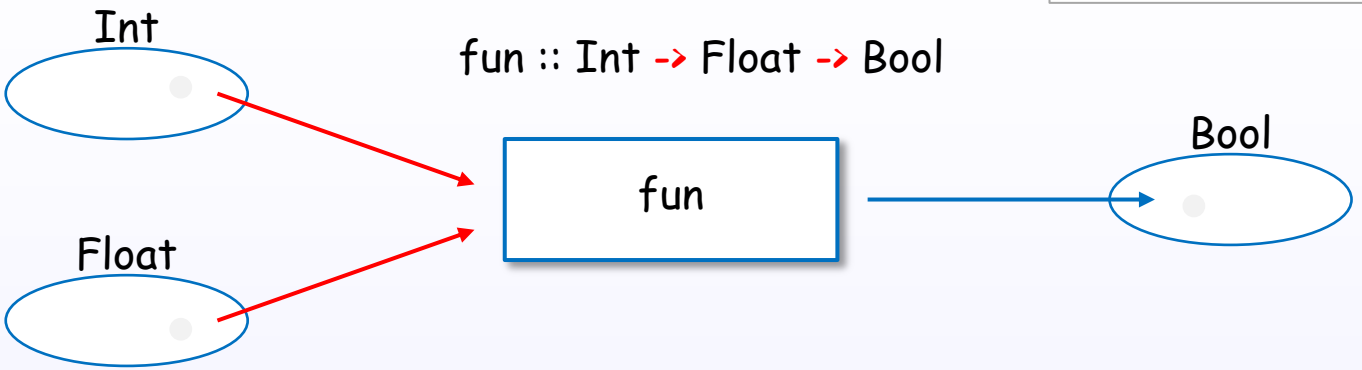
The "->" represents the function type.

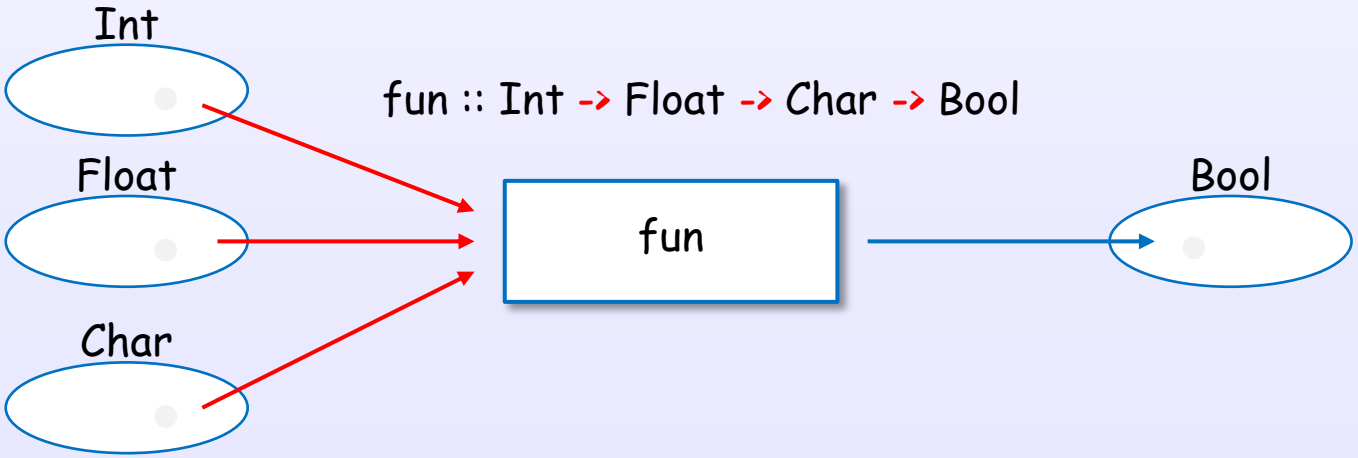
References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7, [B5] Ch.9, [H1] Ch.4

Function type with multiple arguments

equivalent to

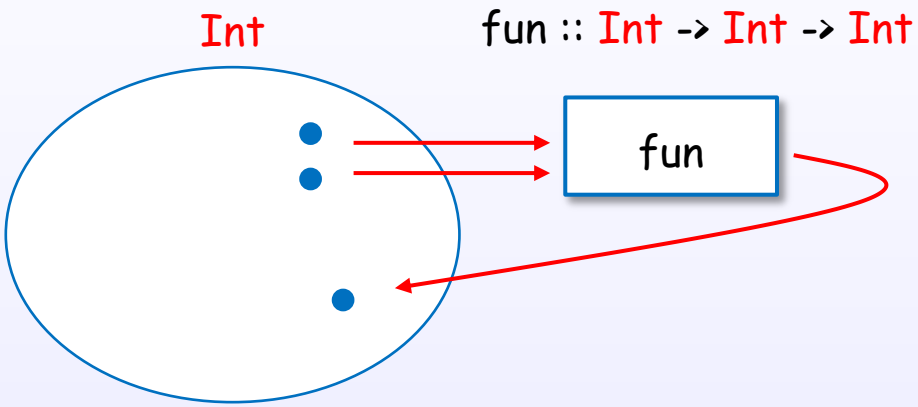
```
f :: Int -> (Float -> Bool)
```





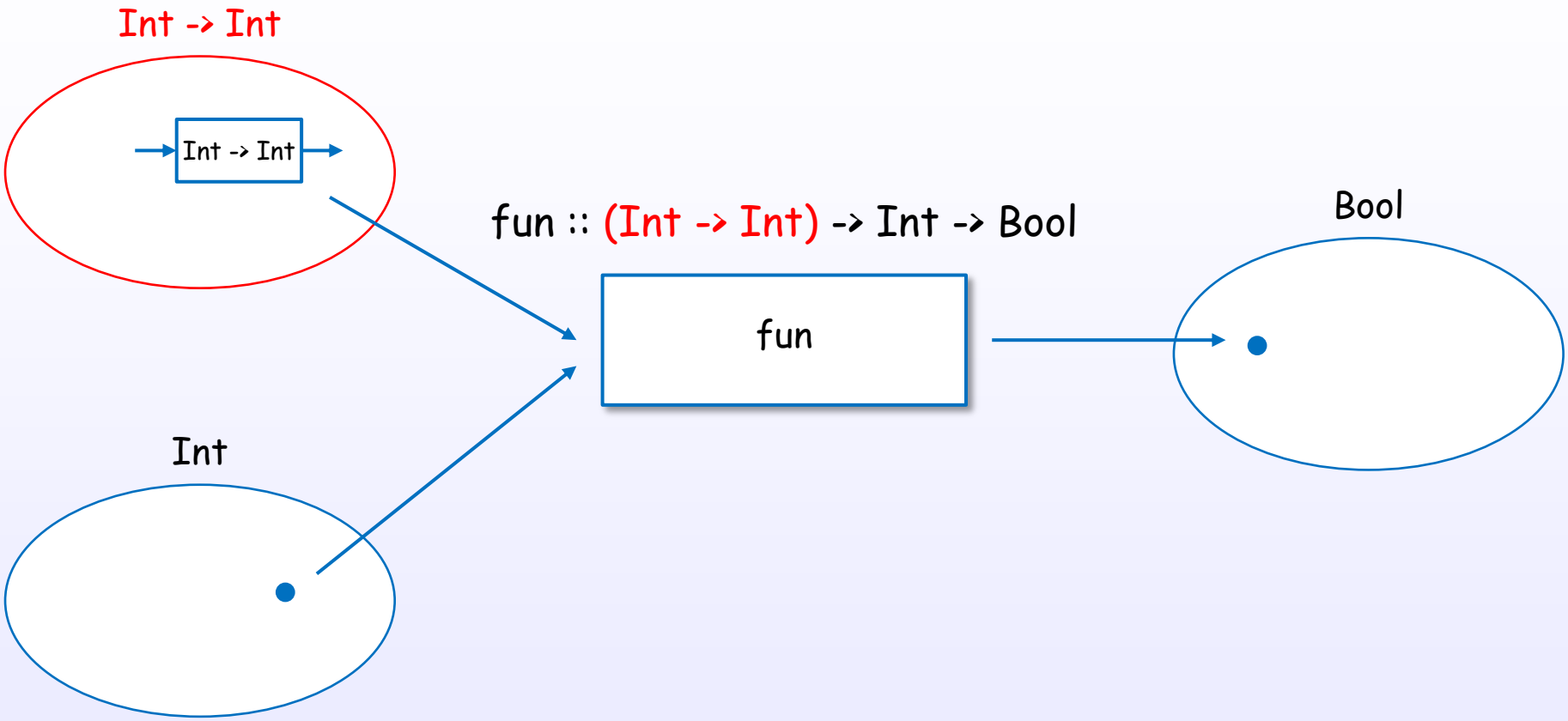
References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

Function type with same type



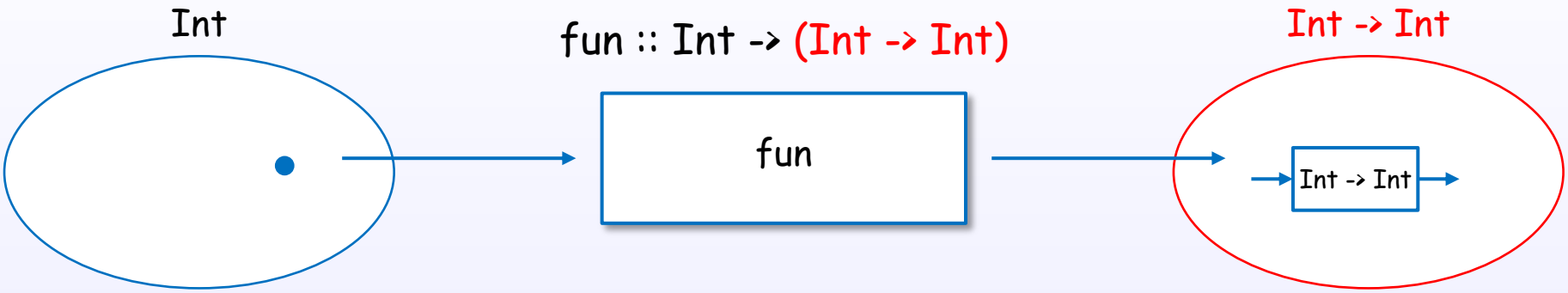
References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

Function type with function as argument



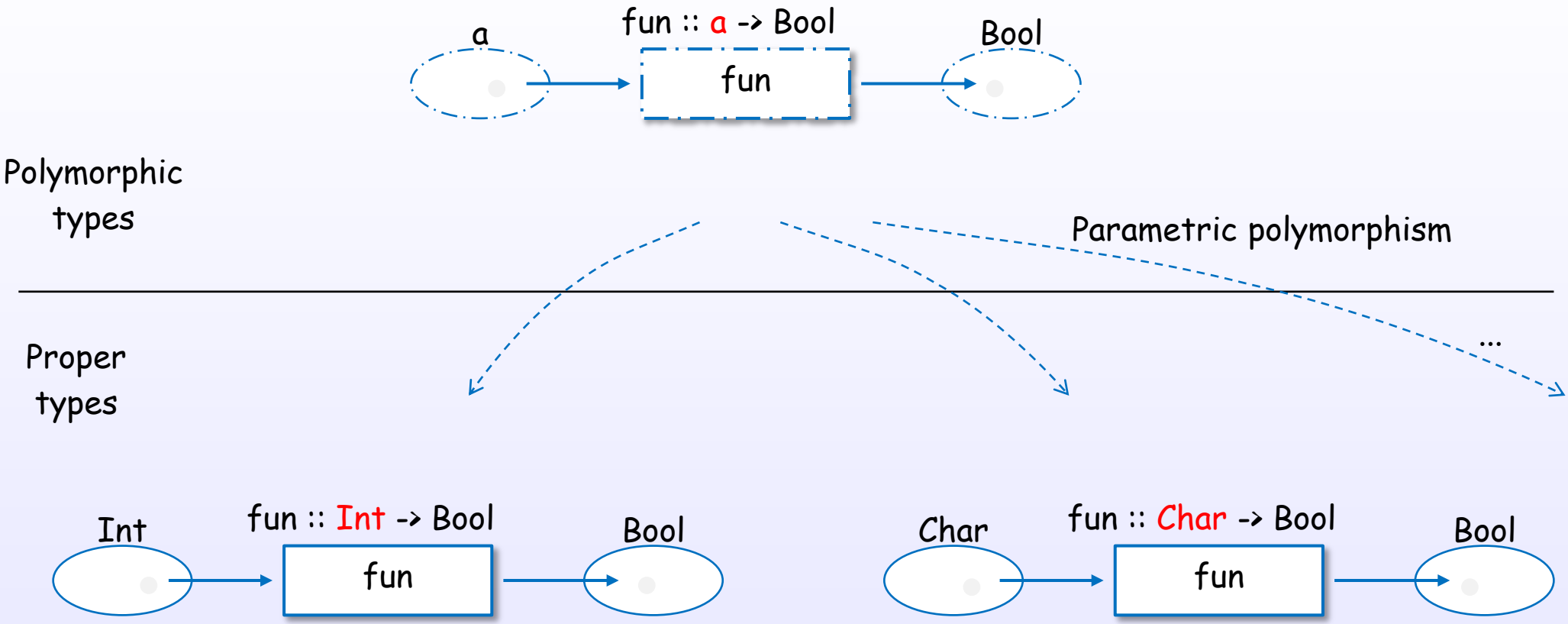
References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

Function type with function as result



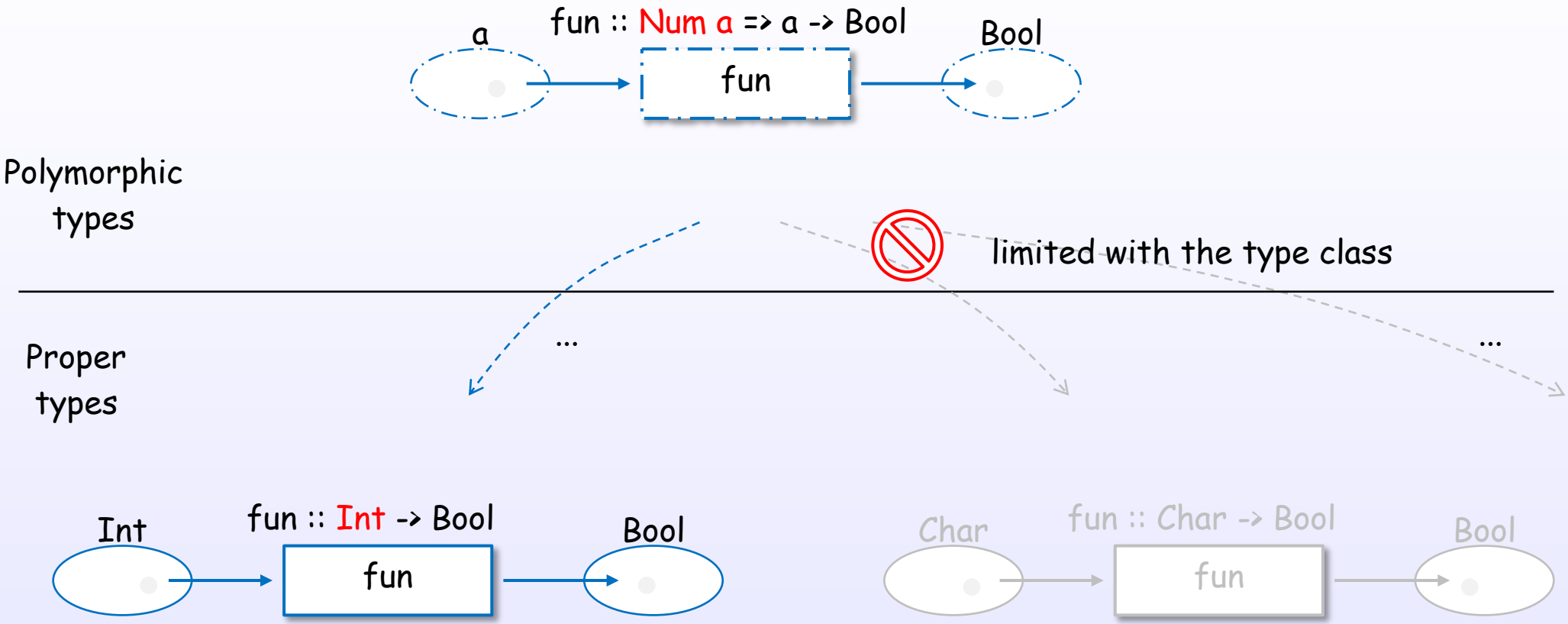
References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

Function type with polymorphic function



References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

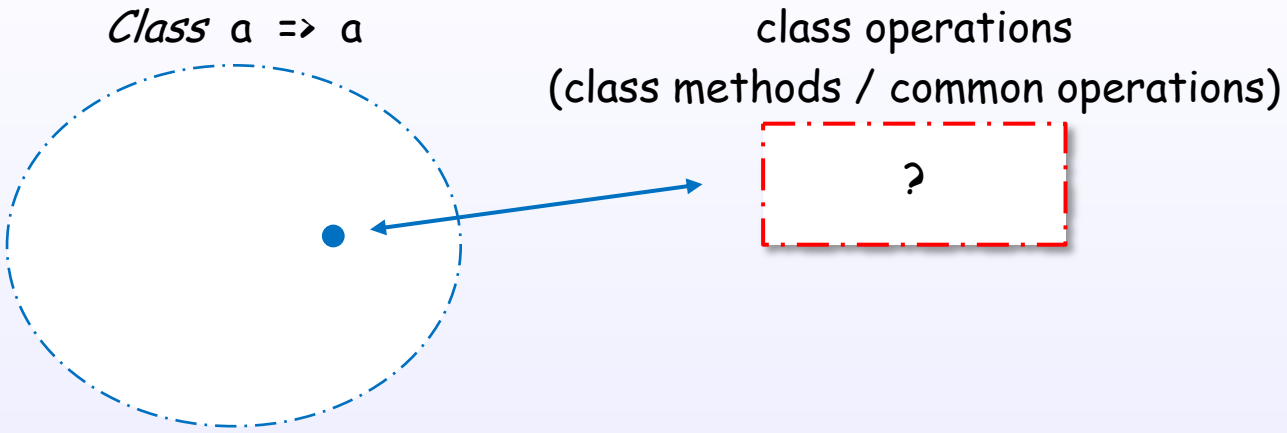
Function type for polymorphic function with type class



References : [B2] Ch.2, [B1] Ch.5, [B3] Ch.7

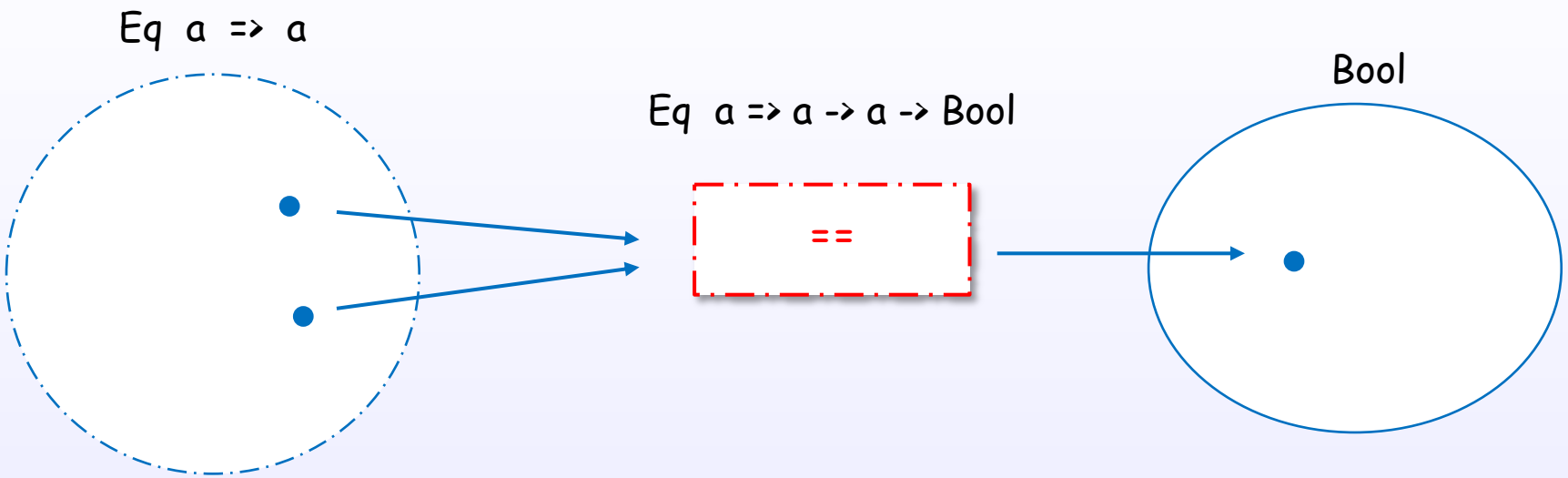
Type class operations

A type class has the class operations



References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

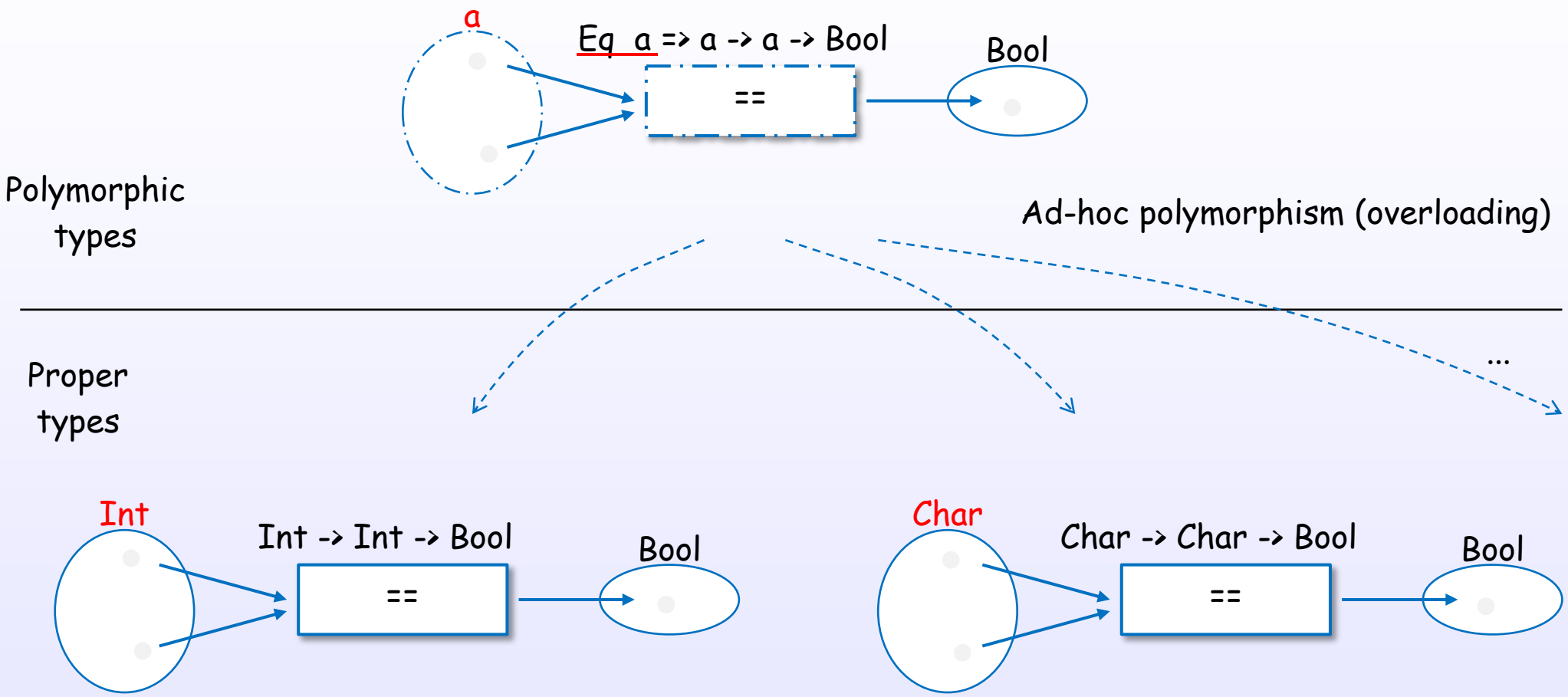
A type class has the class operations



Eq class has "==" (equality) operation.

References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

A type class has the class operations

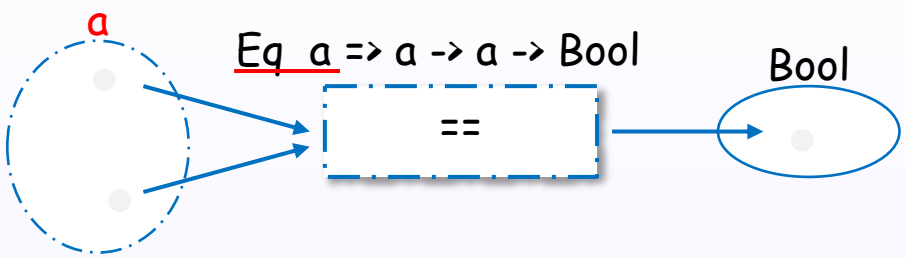


Each type, that belongs to the type class, must be support the overloaded operations.

References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

Declaration of a type class and instances

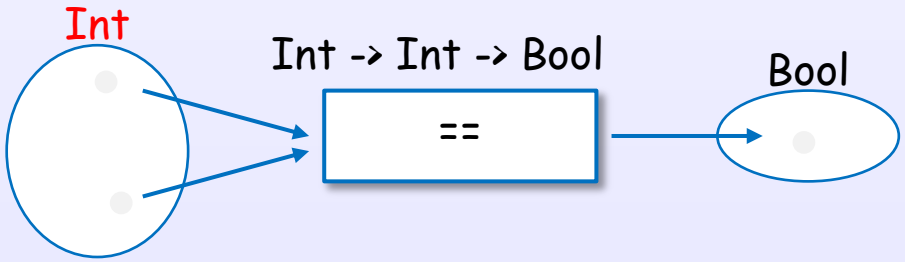
```
class Eq a where
    (==) :: a -> a -> Bool
```



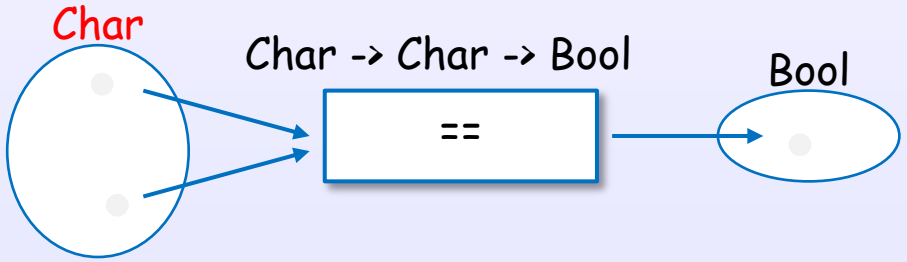
Polymorphic types

Proper types

...




```
instance Eq Int where
    (==) = eqInt
```




```
instance Eq  Char  where
    (==) = ...
```

References : [B1] Ch.2, 7, [B2] Ch.2, [B3] Ch.3, [B4] Ch.6

3. What is this?

What is this ?!

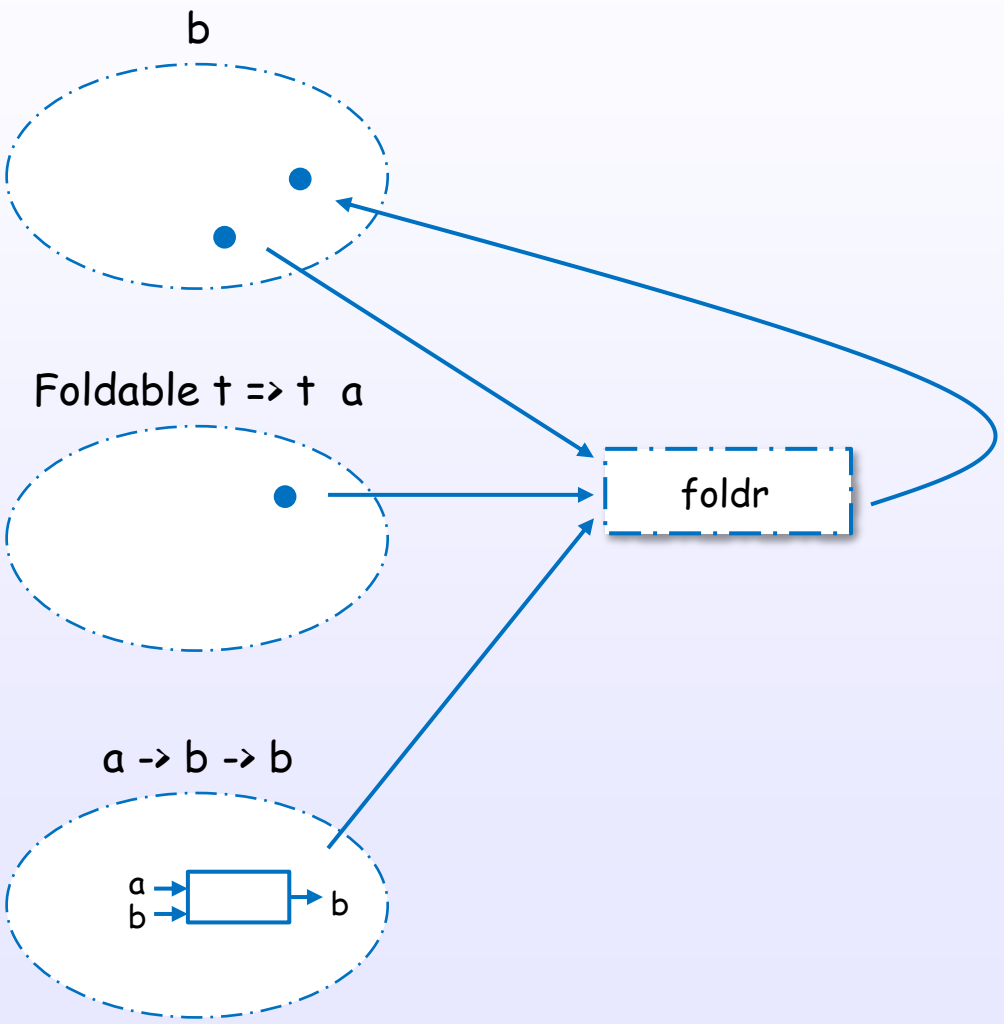
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
foldr = ...
```

?

References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

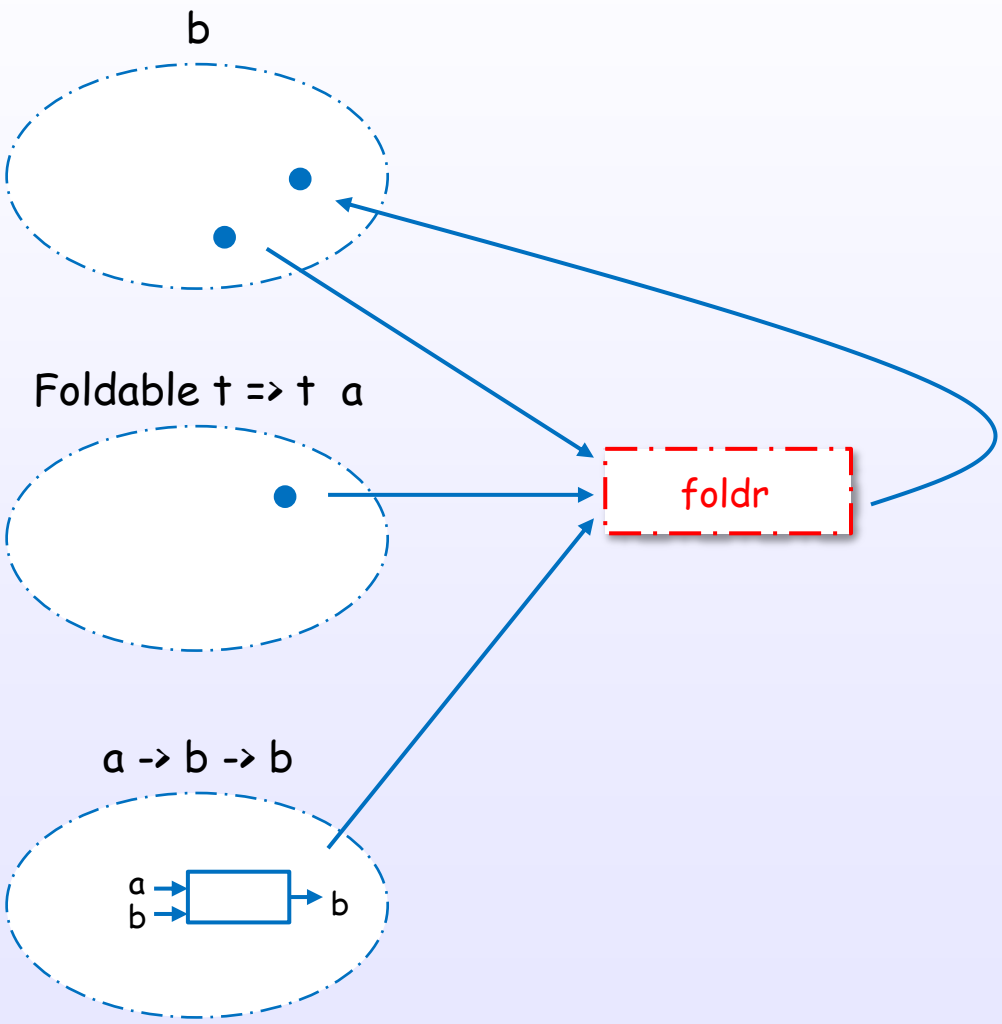
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

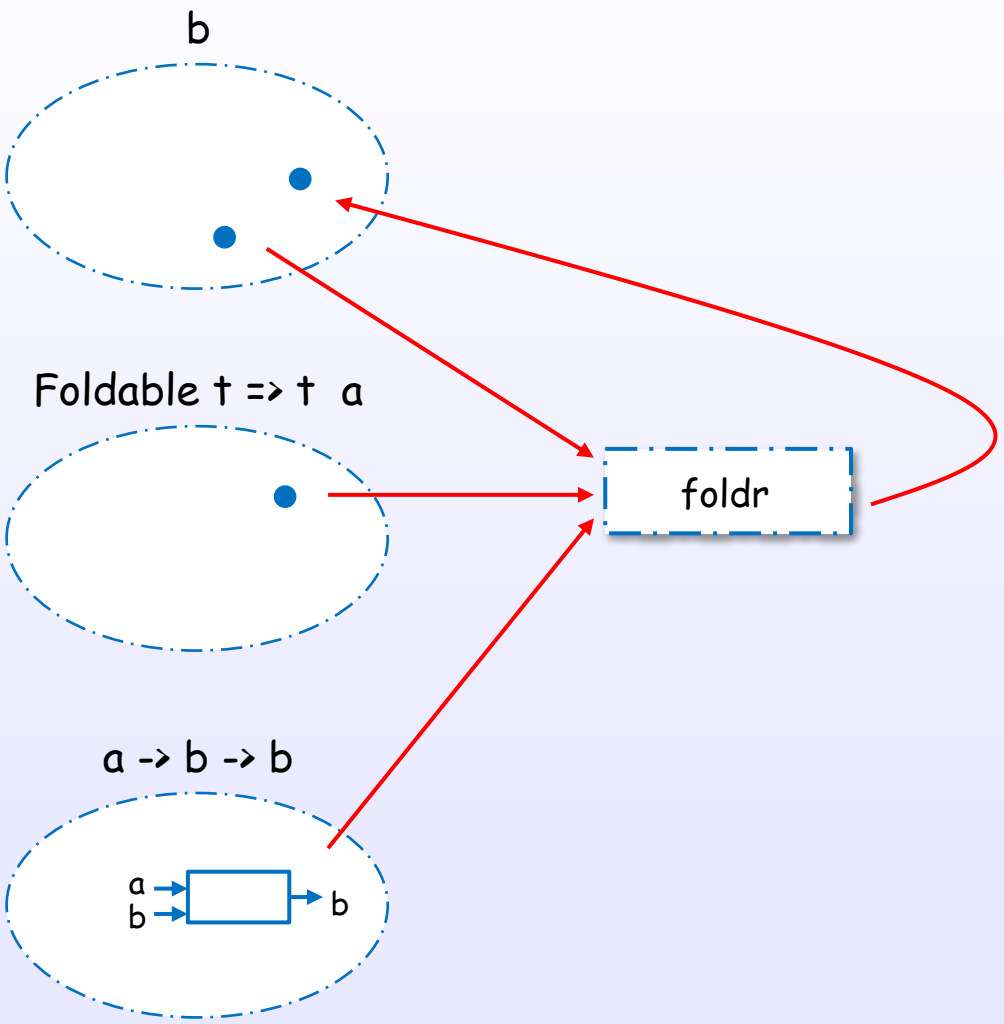
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

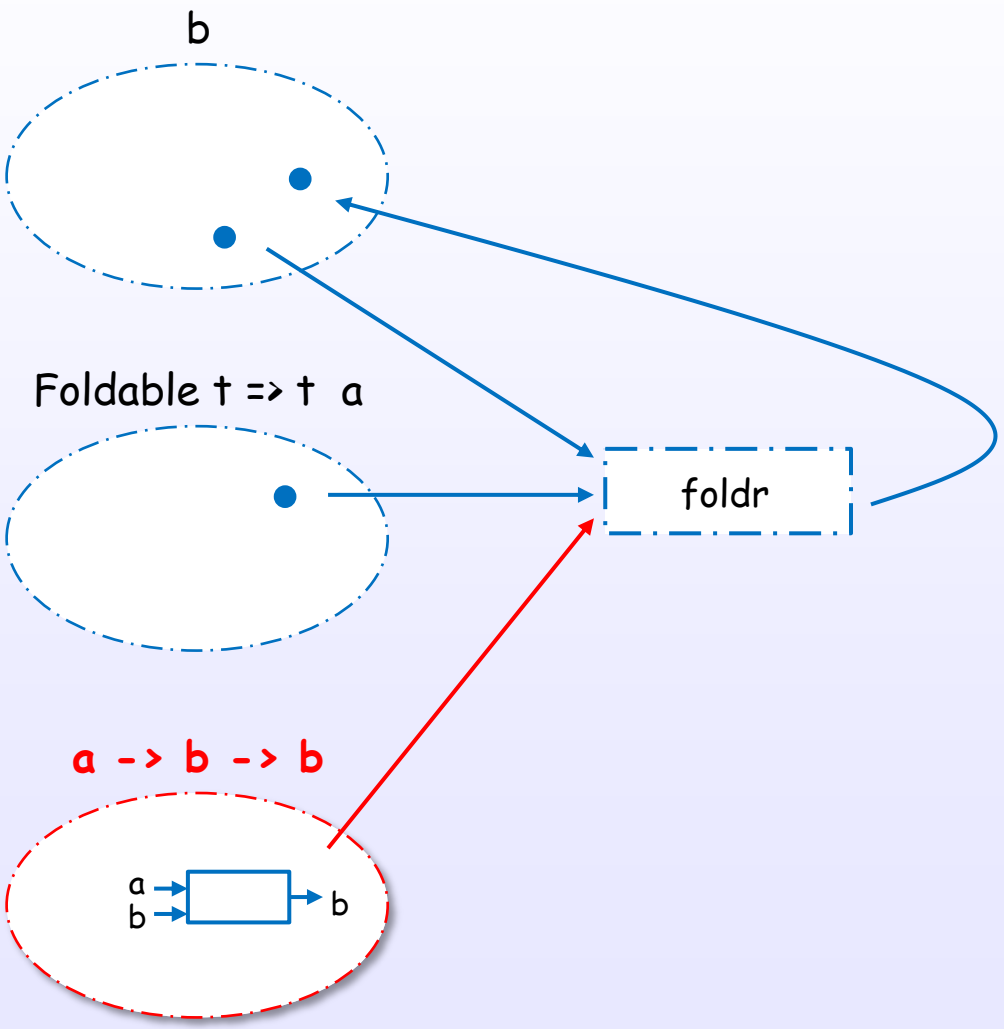
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

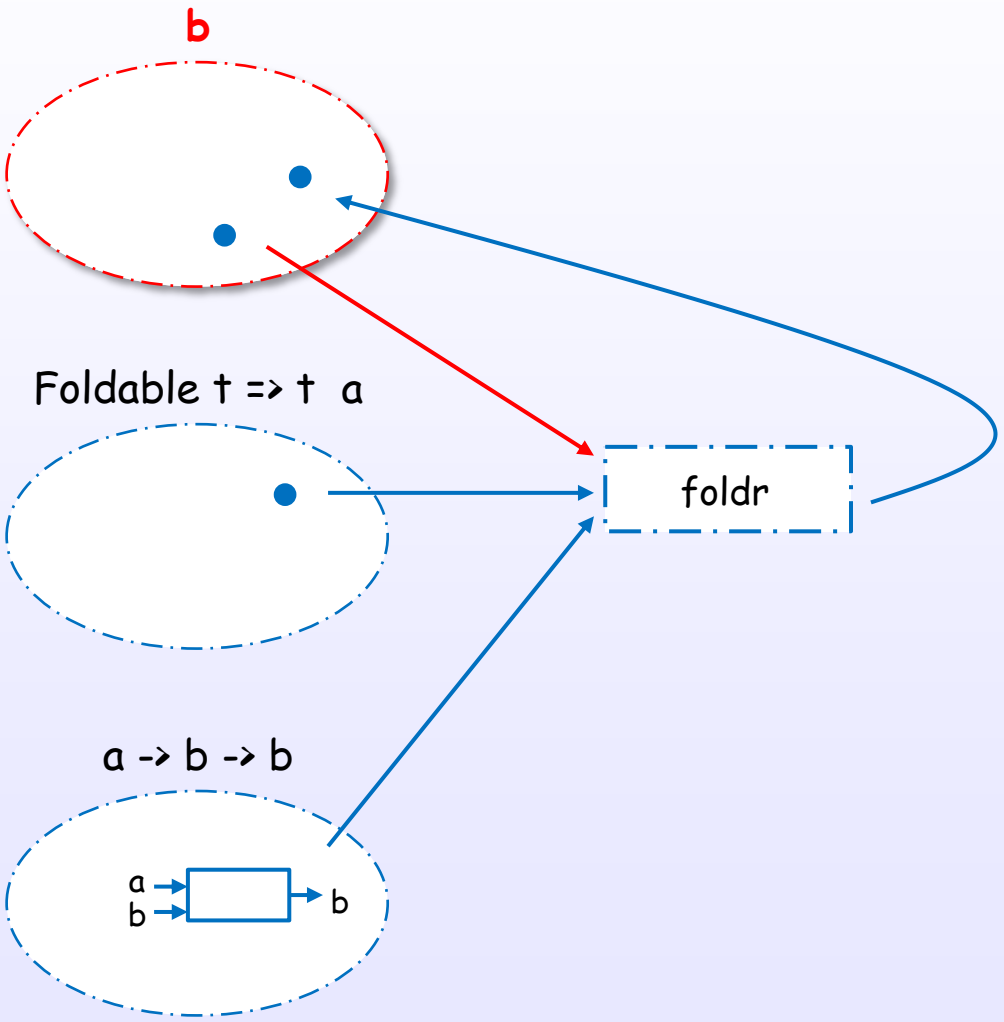
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

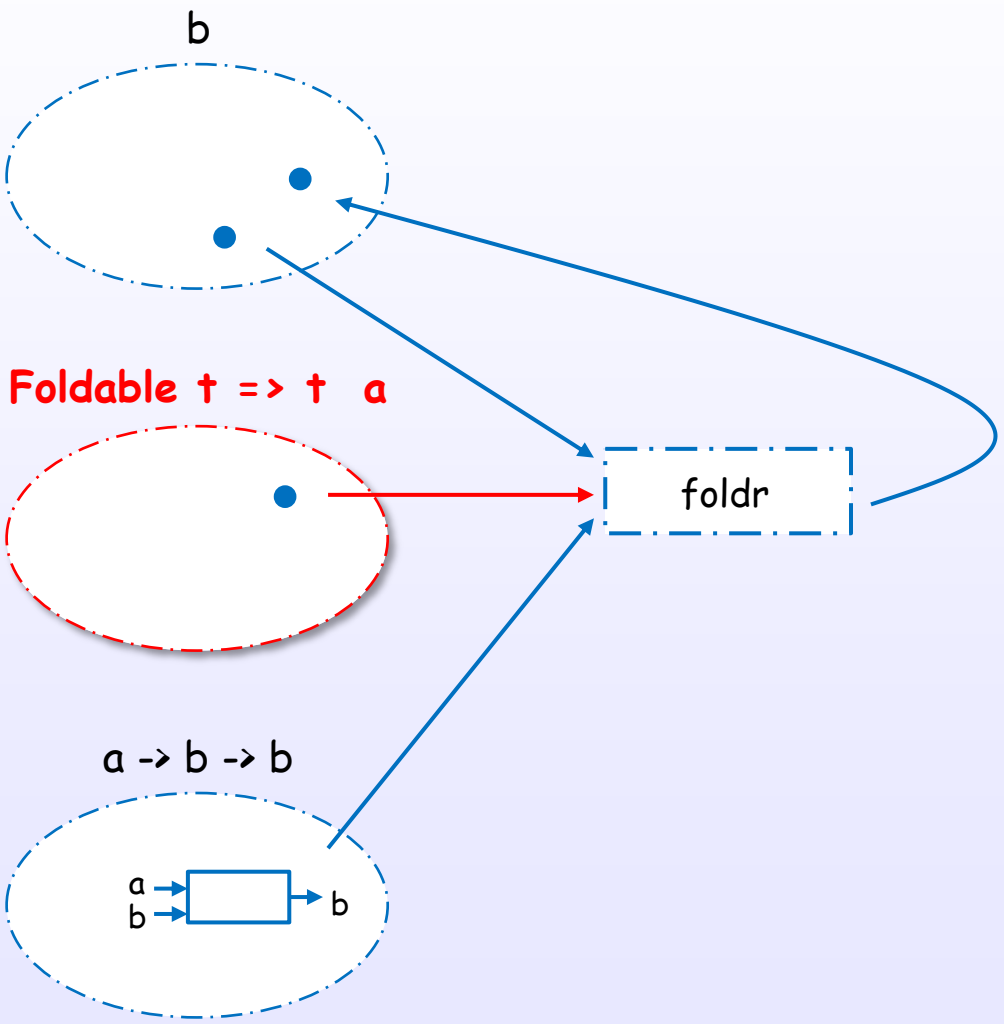
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

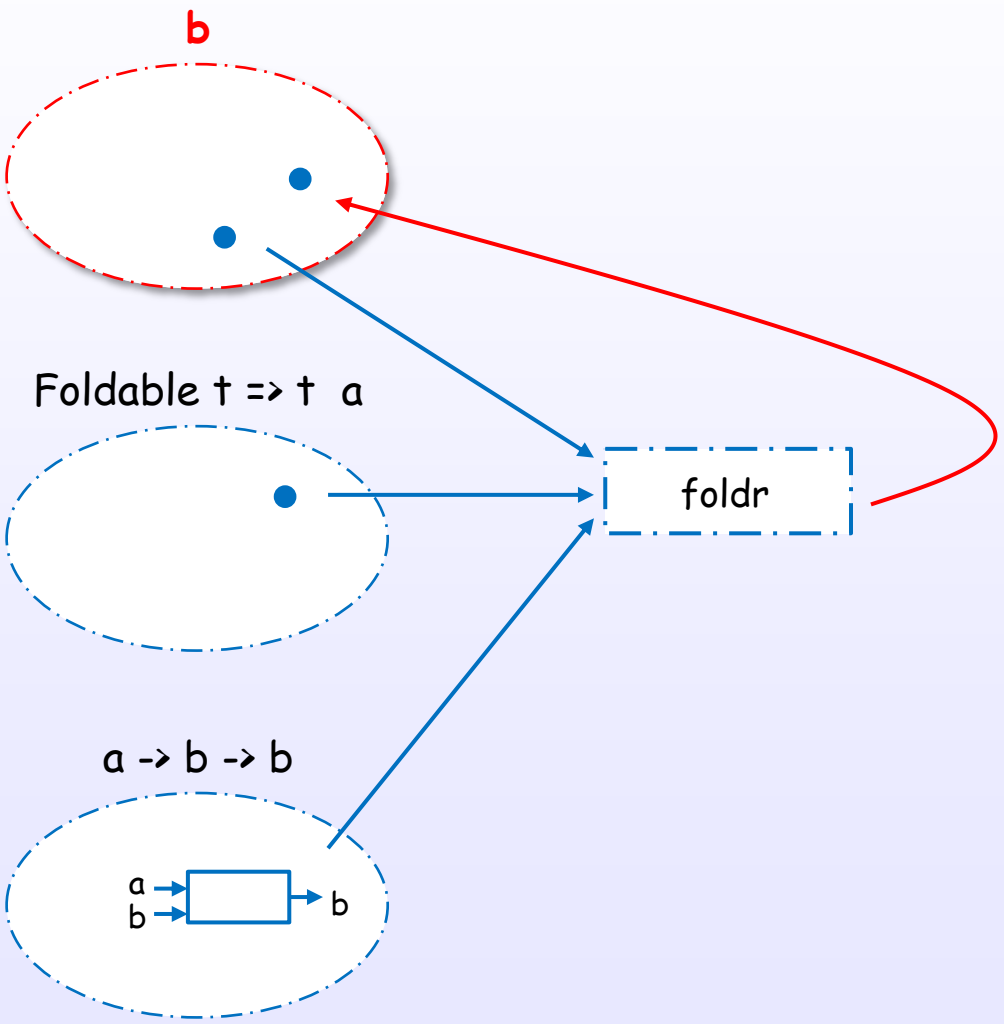
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

What is this ?!

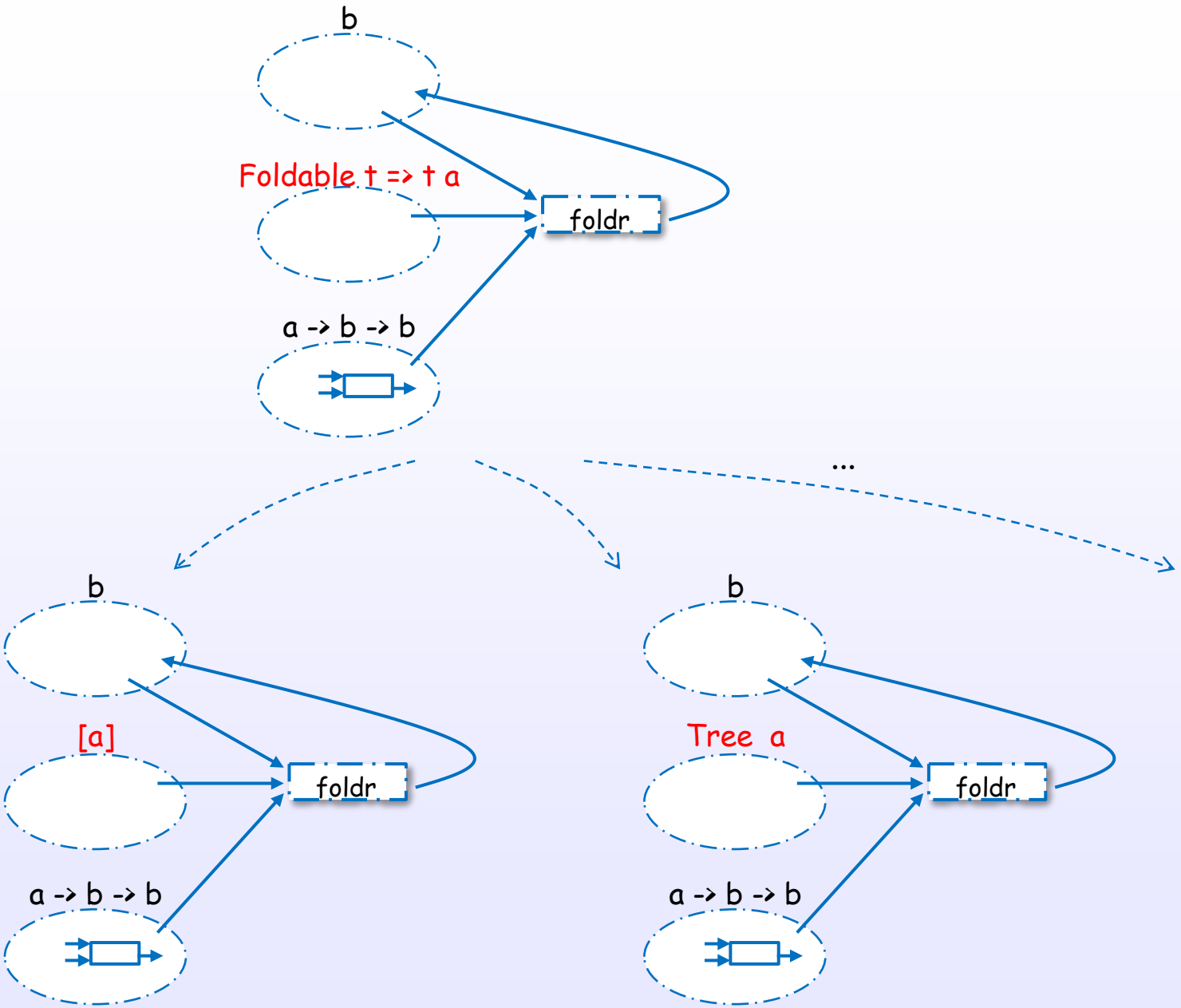
```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

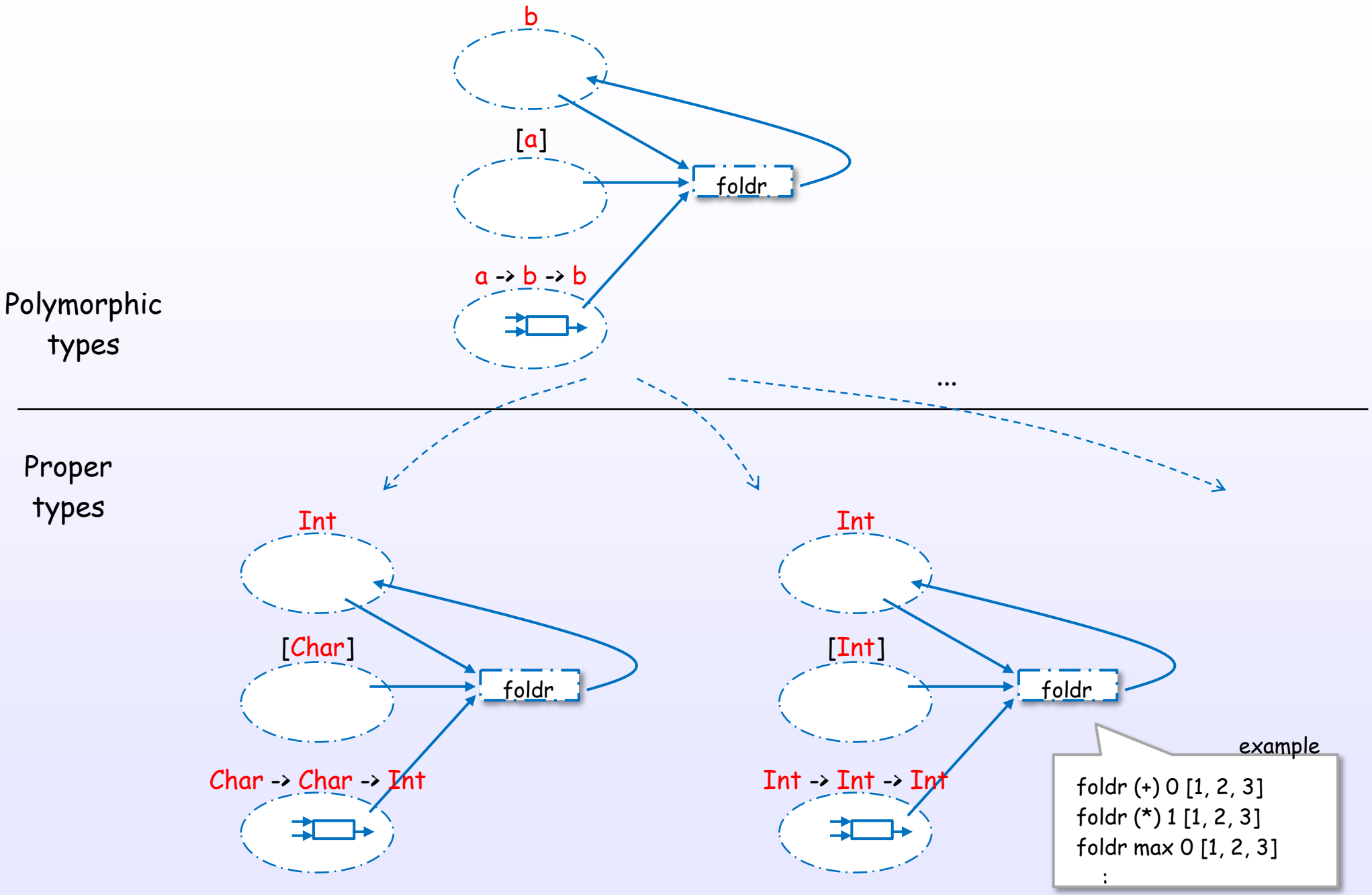
Example of polymorphism on foldr

Polymorphic types



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

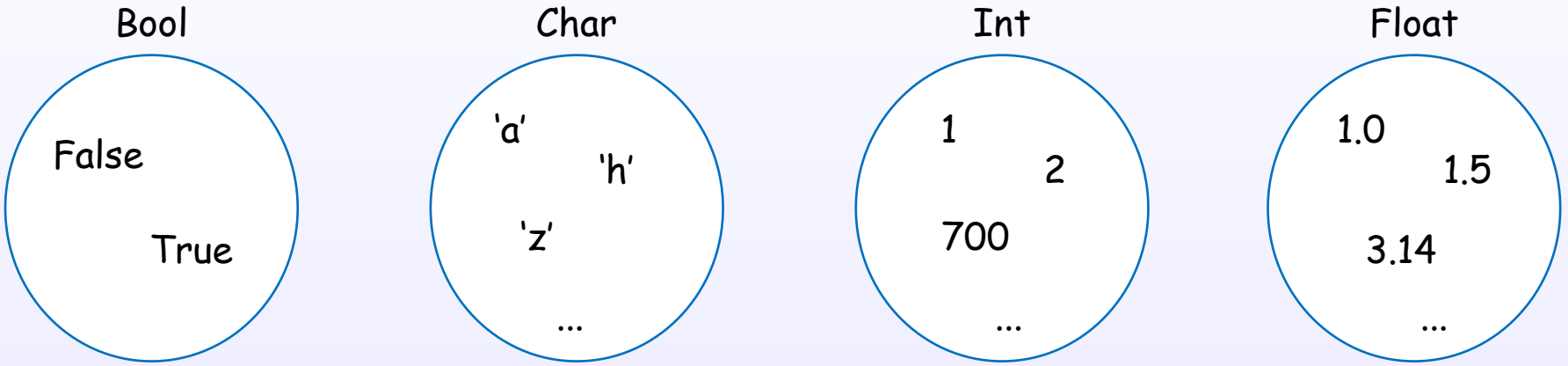
Example of polymorphism on foldr



References : [B1] Ch.12, 5, 7, [B2] Ch.6, 2, [B3] Ch.7, 3, [B4] Ch.4, 6

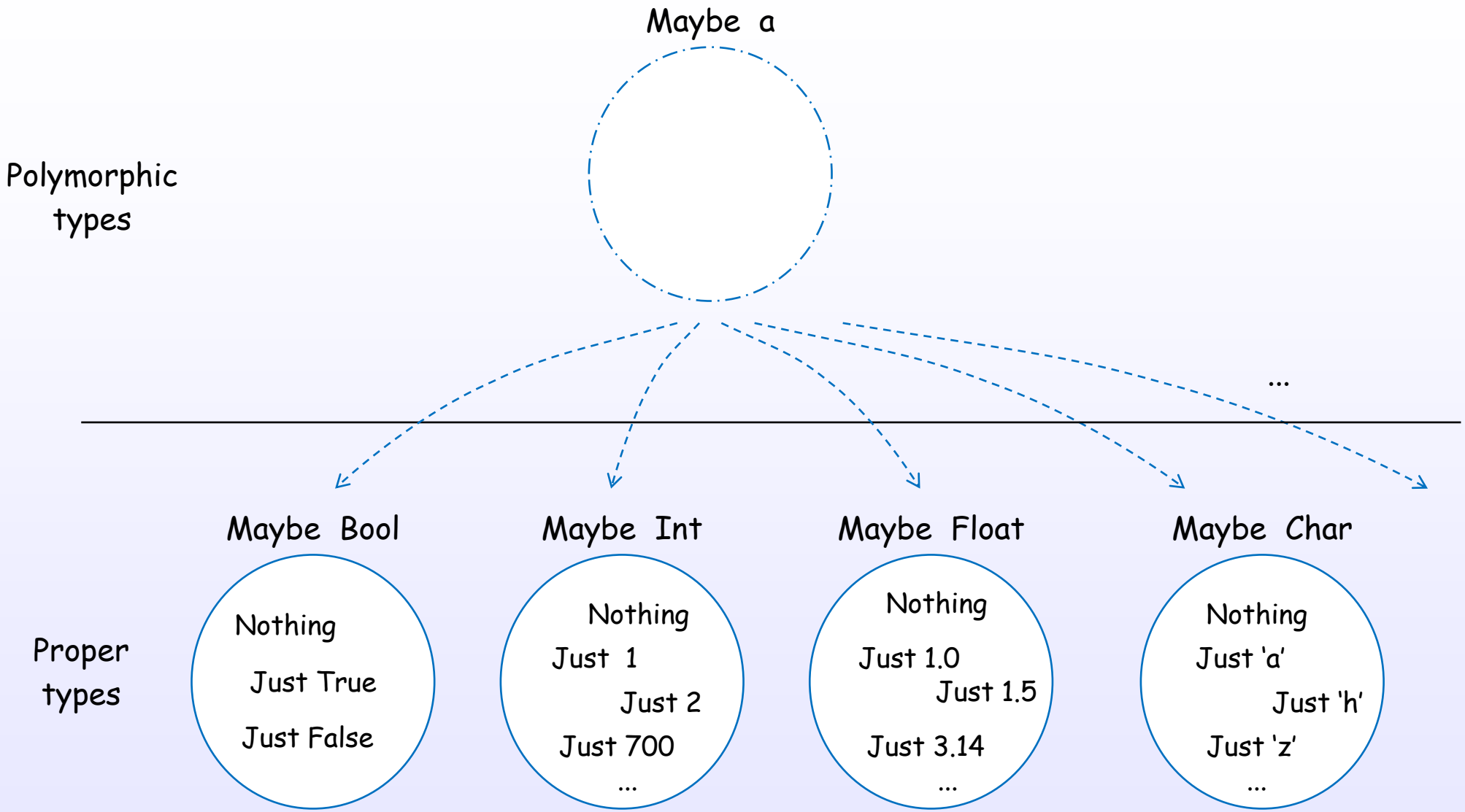
Appendix I – Various types

Bool, Char, Int, Float types



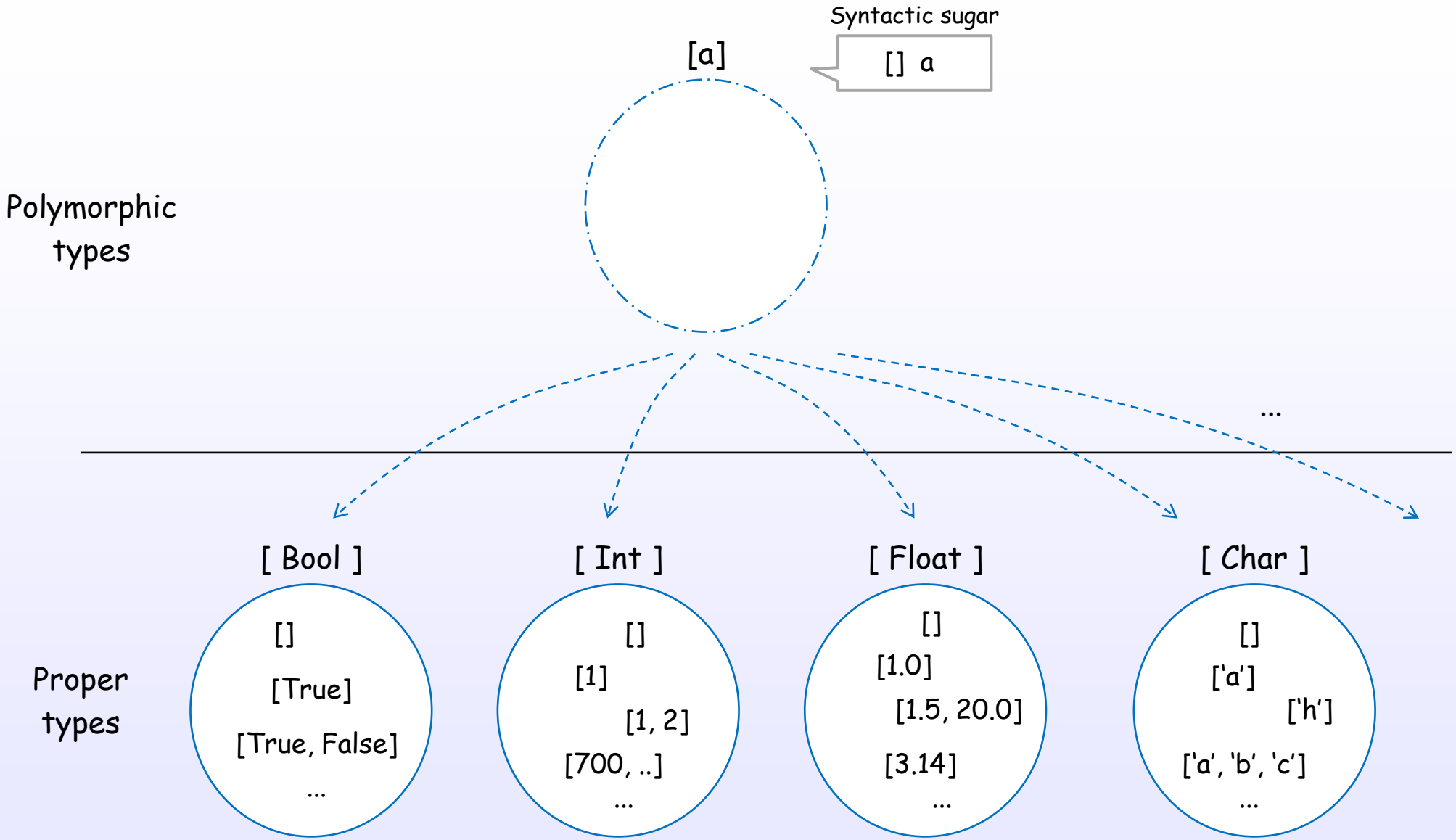
References : [B2] Ch.2, [B3] Ch.2, [B1] Ch.2, [H1] Ch.6, [S1]

Maybe type



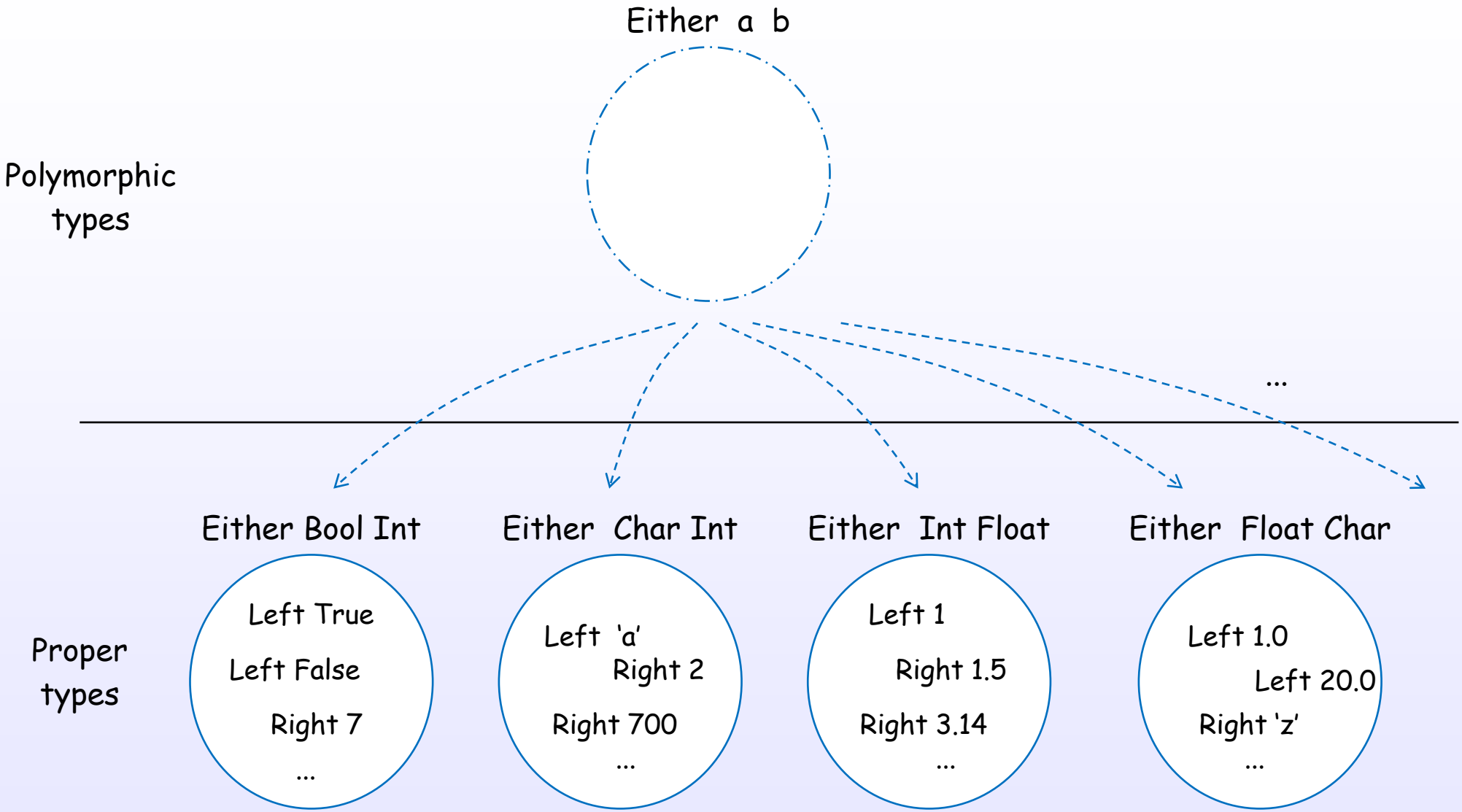
References : [B1] Ch.7, [S1]

List type



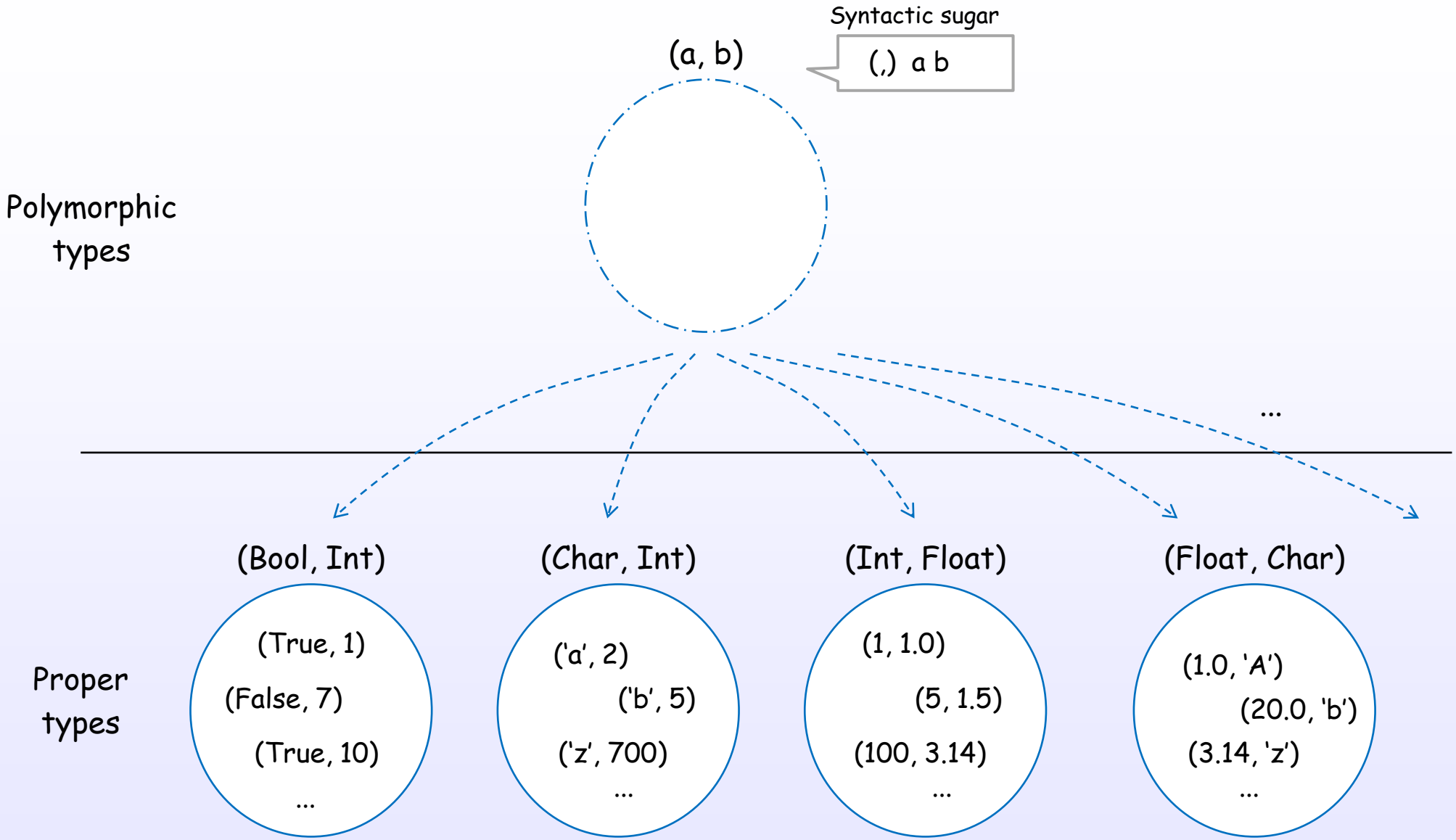
References : [B1] Ch.1, [B2] Ch.4, [B3] Ch.3, [H1] Ch.6, [S1]

Either type



References : [B1] Ch.7, [S1]

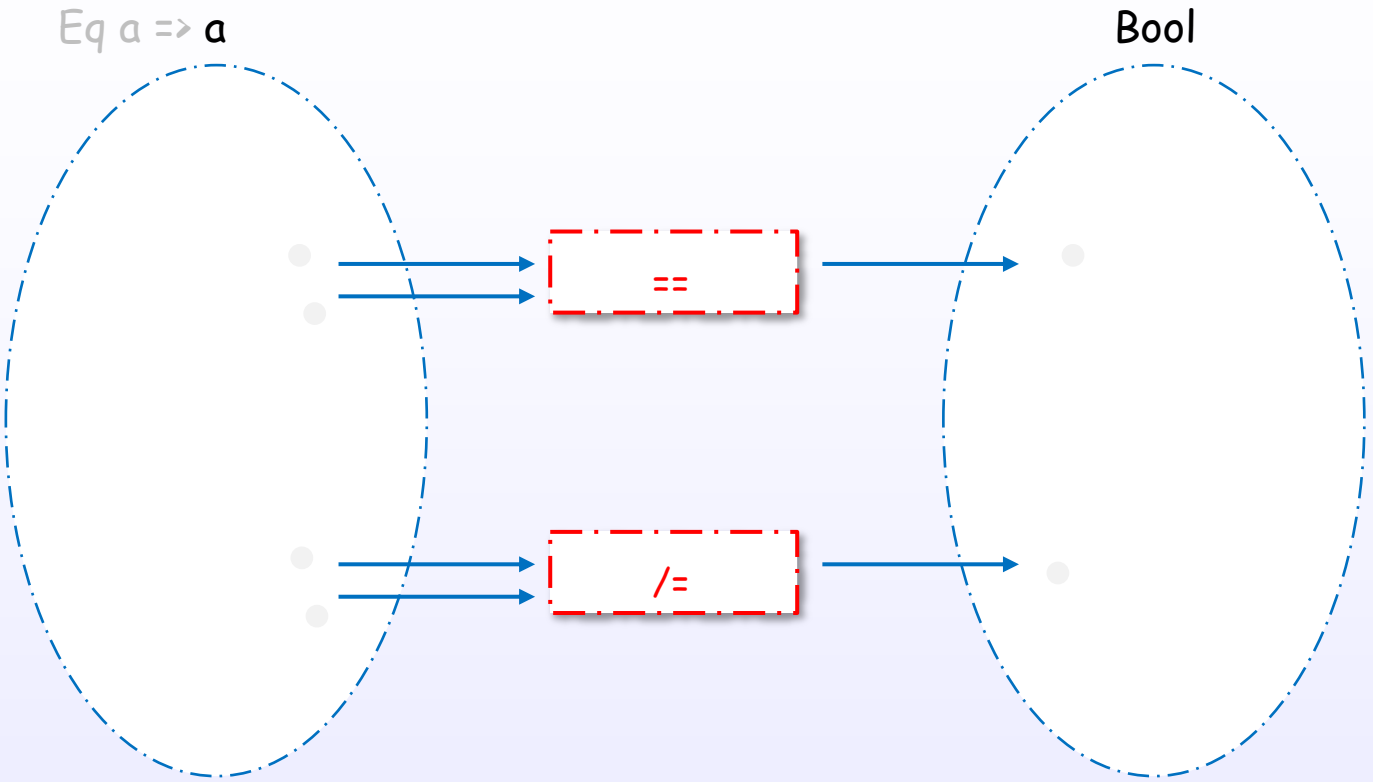
Tuple (pair) type



References : [B1] Ch.1, [B2] Ch.2, [B3] Ch.3, [H1] Ch.6, [S1]

Appendix II – Various type classes

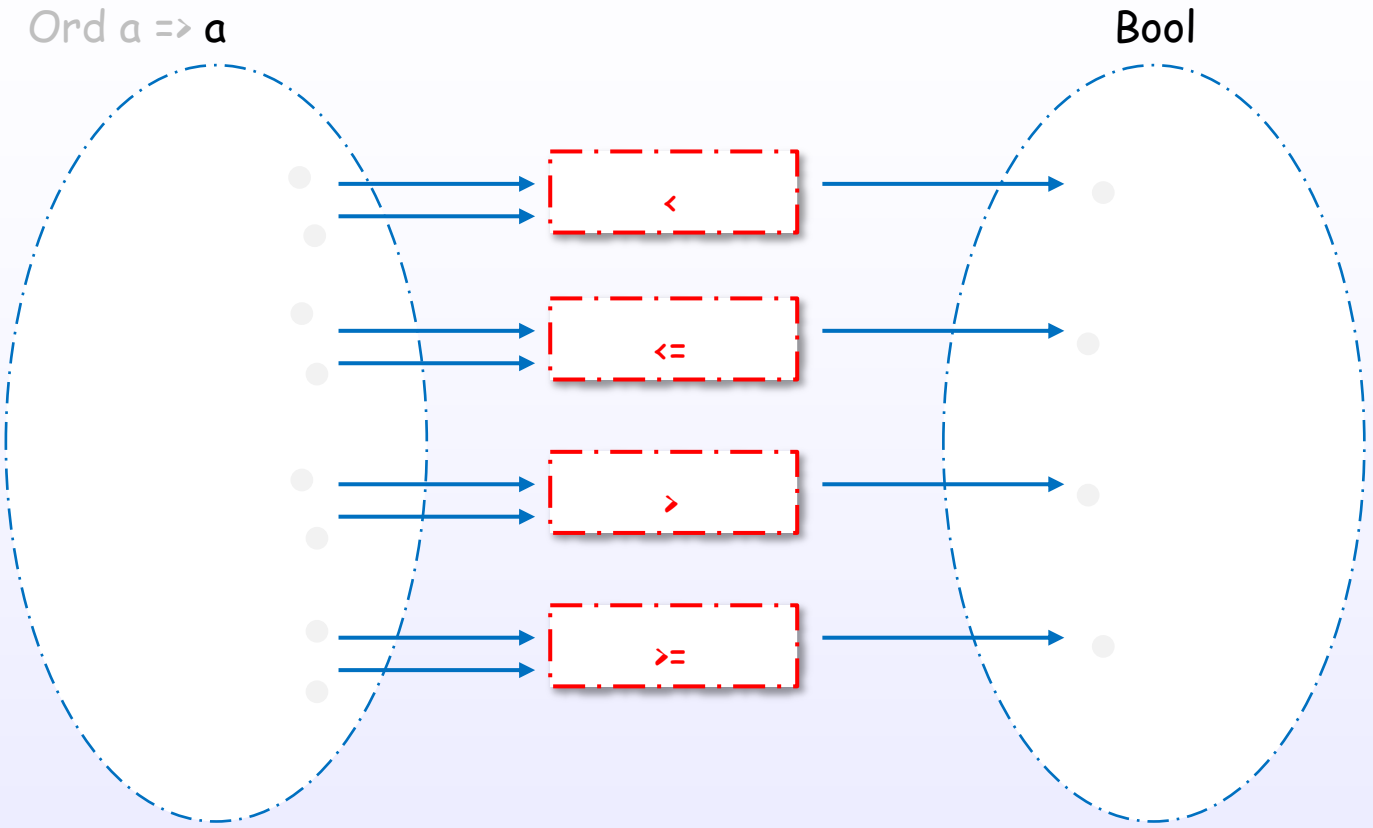
Eq class's characteristic operations



The Eq class has equality operations.

References : [B1] Ch.1, [B2] Ch.2, [B3] Ch.3, [H1] Ch.6, [S1]

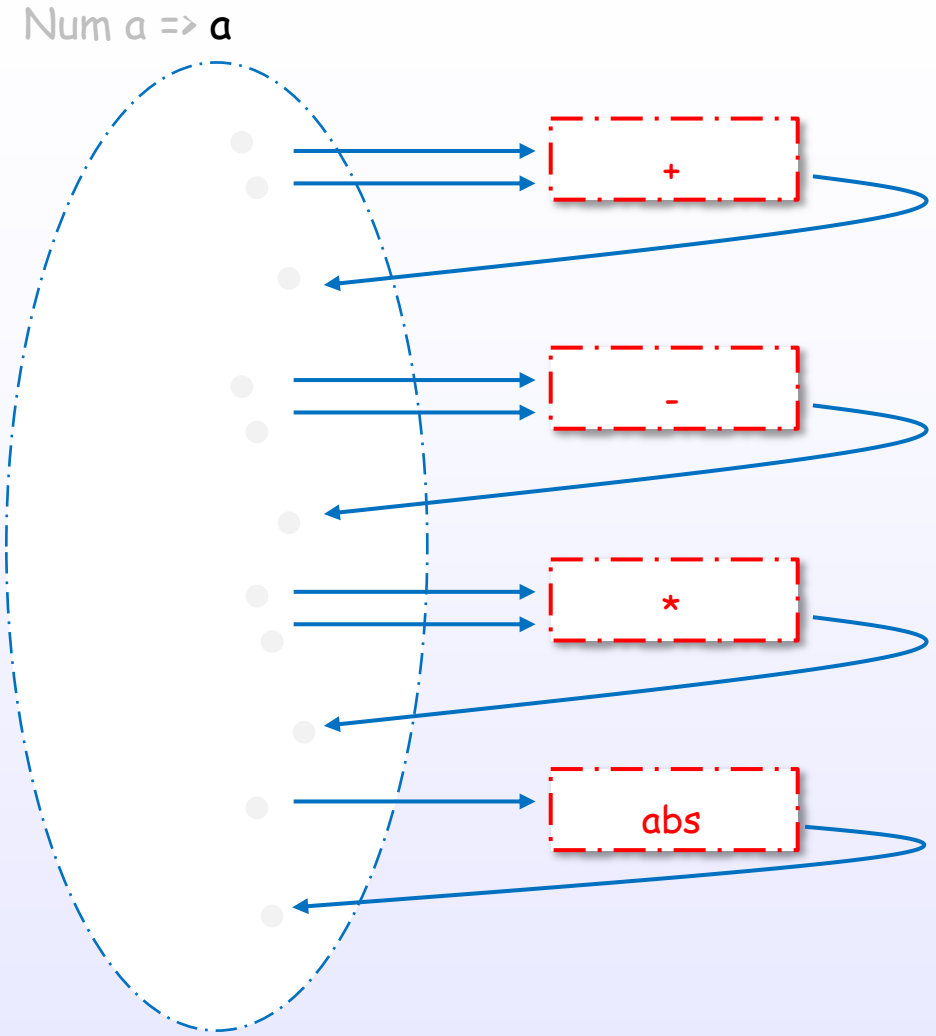
Ord class's characteristic operations



The Ord class has comparison operations.

References : [B1] Ch.1, [B2] Ch.2, [B3] Ch.3, [H1] Ch.6, [S1]

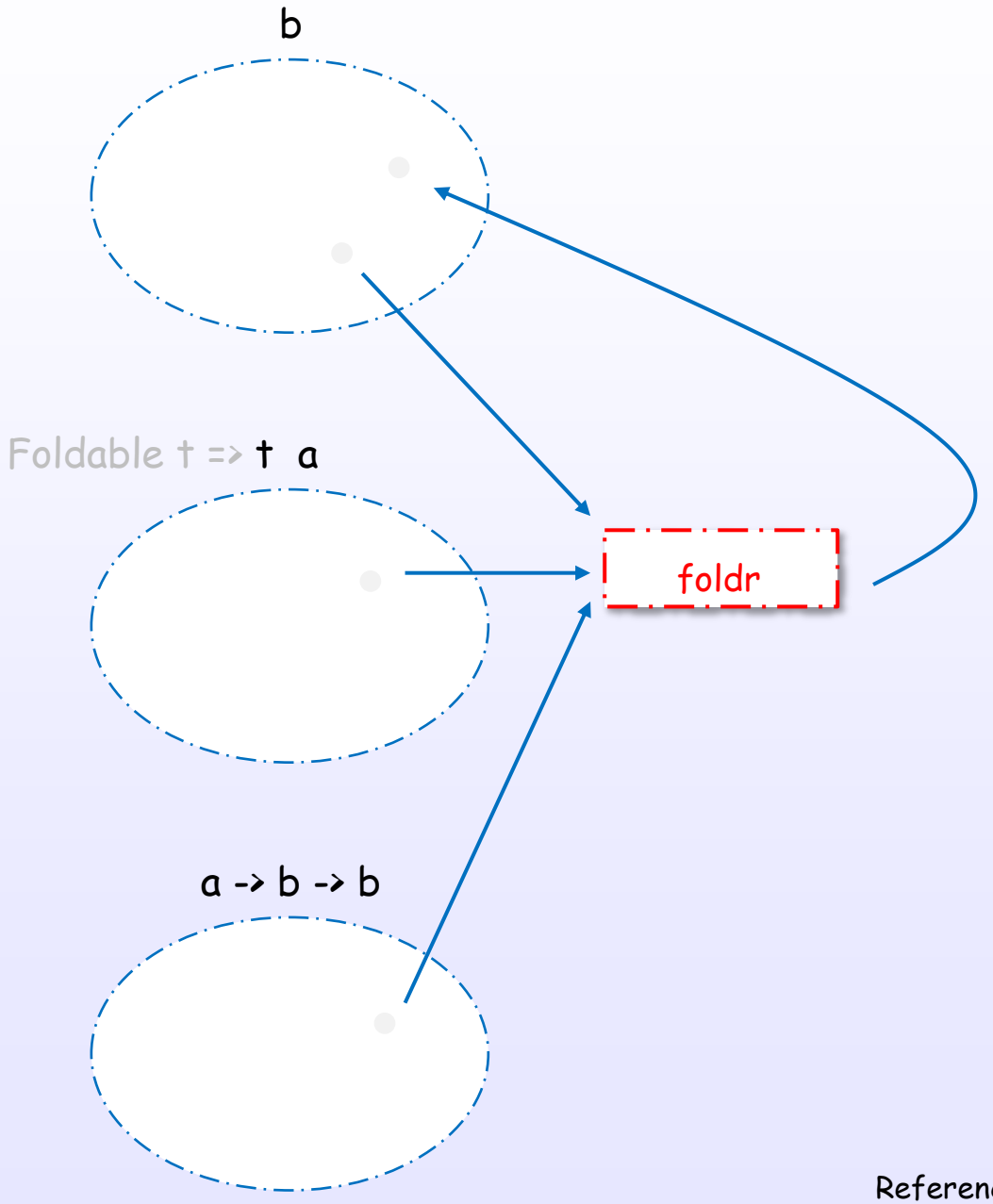
Num class's characteristic operations



The Num class has arithmetic operations.

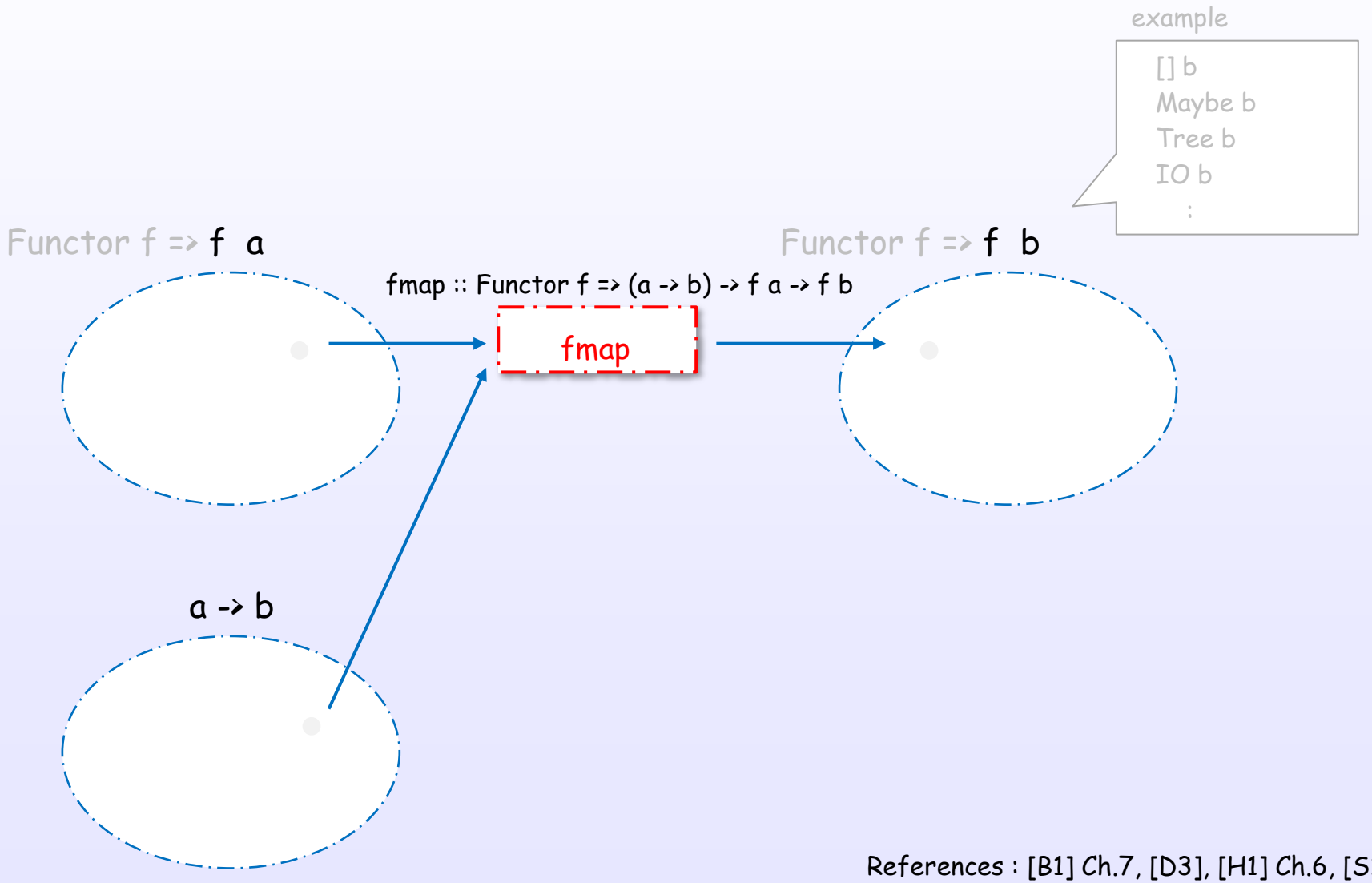
References : [B1] Ch.1, [B2] Ch.2, [B3] Ch.3, [H1] Ch.6, [S1]

Foldable class's characteristic operations



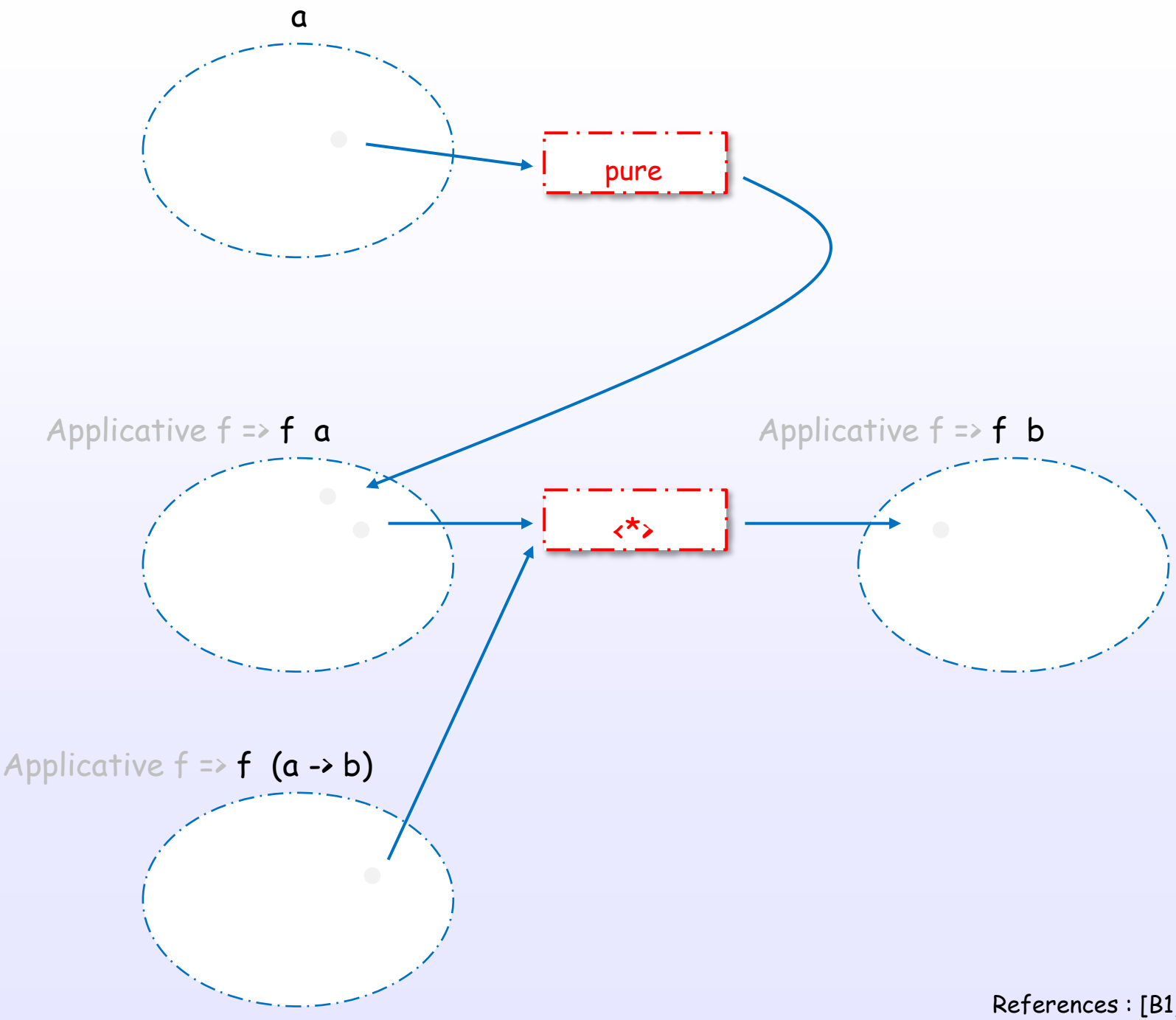
References : [B1] Ch.12, [B2] Ch.6, [B3] Ch.7, [D3], [S1]

Functor class's characteristic operations



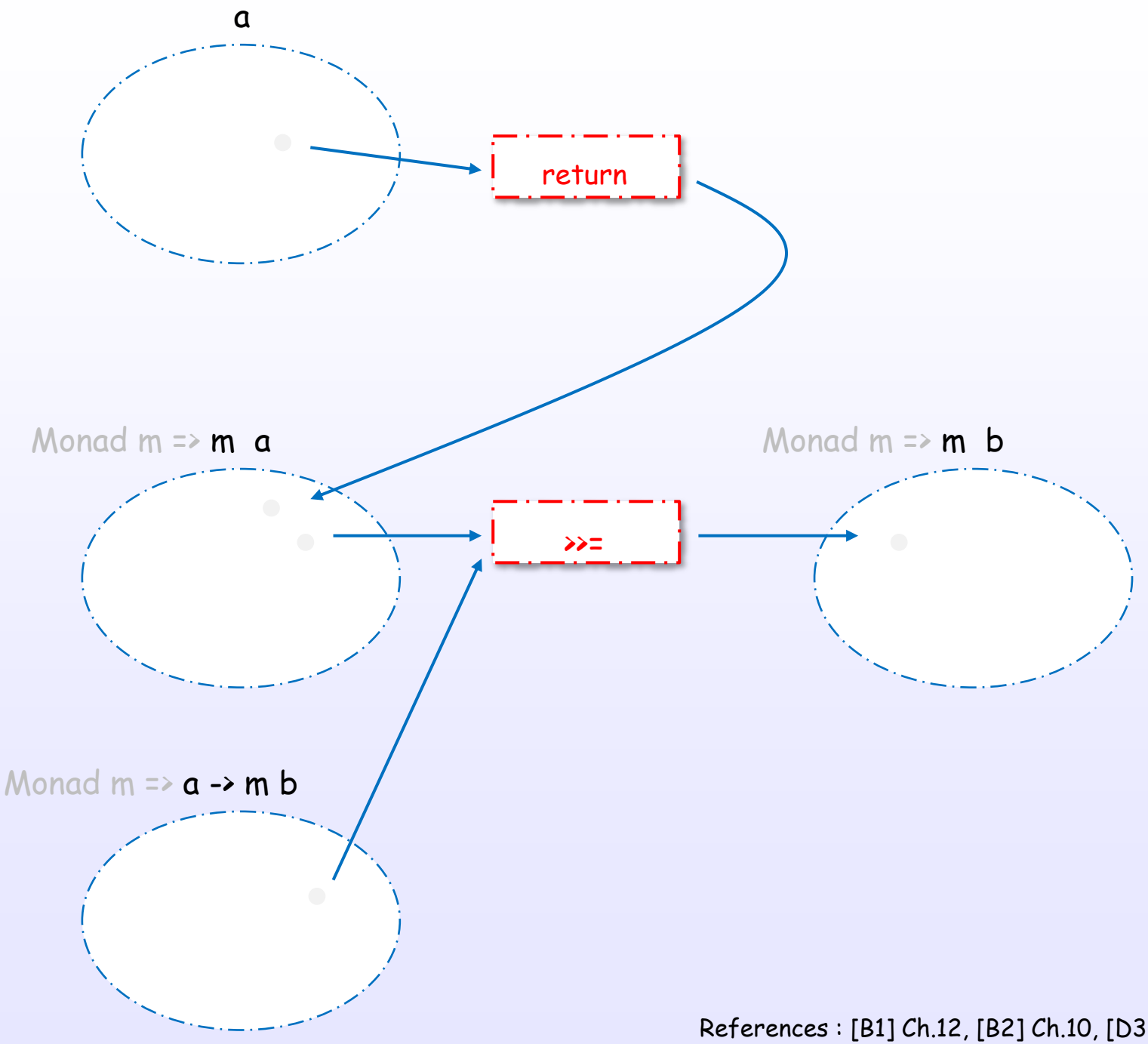
References : [B1] Ch.7, [D3], [H1] Ch.6, [S1]

Applicative class's characteristic operations



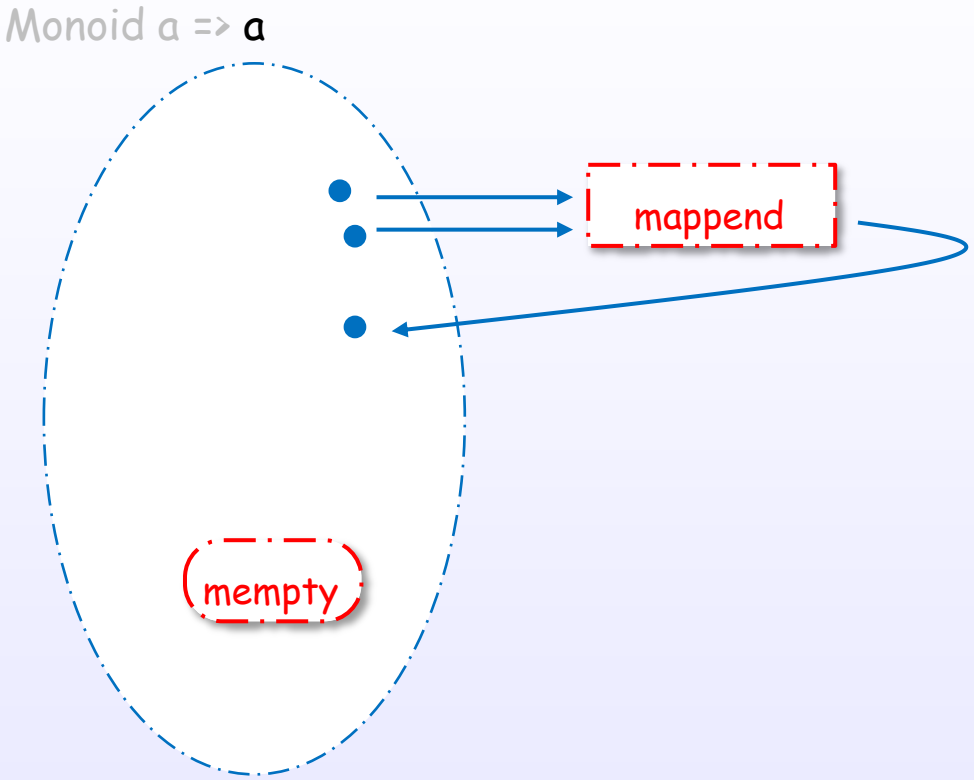
References : [B1] Ch.11, [D3], [S1]

Monad class's characteristic operations



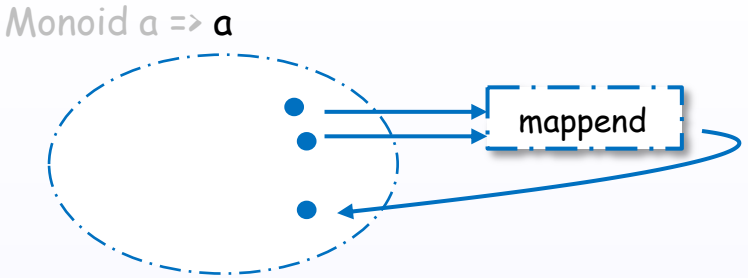
References : [B1] Ch.12, [B2] Ch.10, [D3], [H1] Ch.6, [S1]

Monoid class's characteristic operations

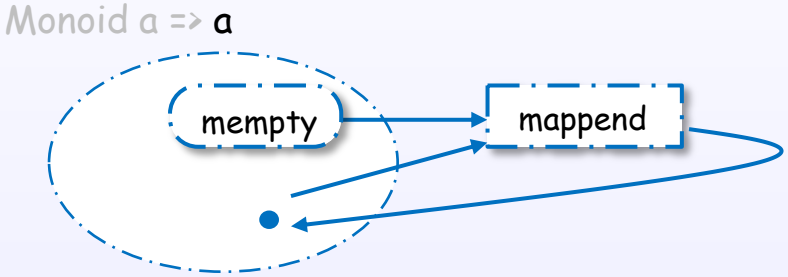


References : [B1] Ch.12, [D3], [S1]

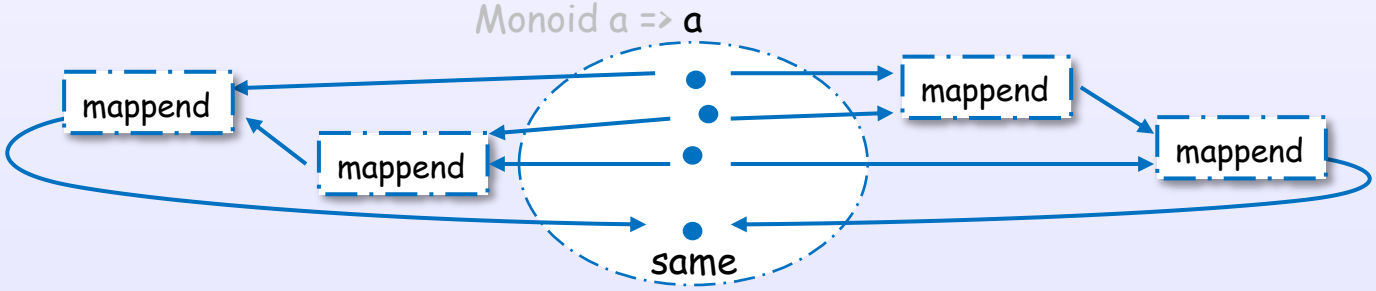
Related topics: monoid laws



binary operation



unit law

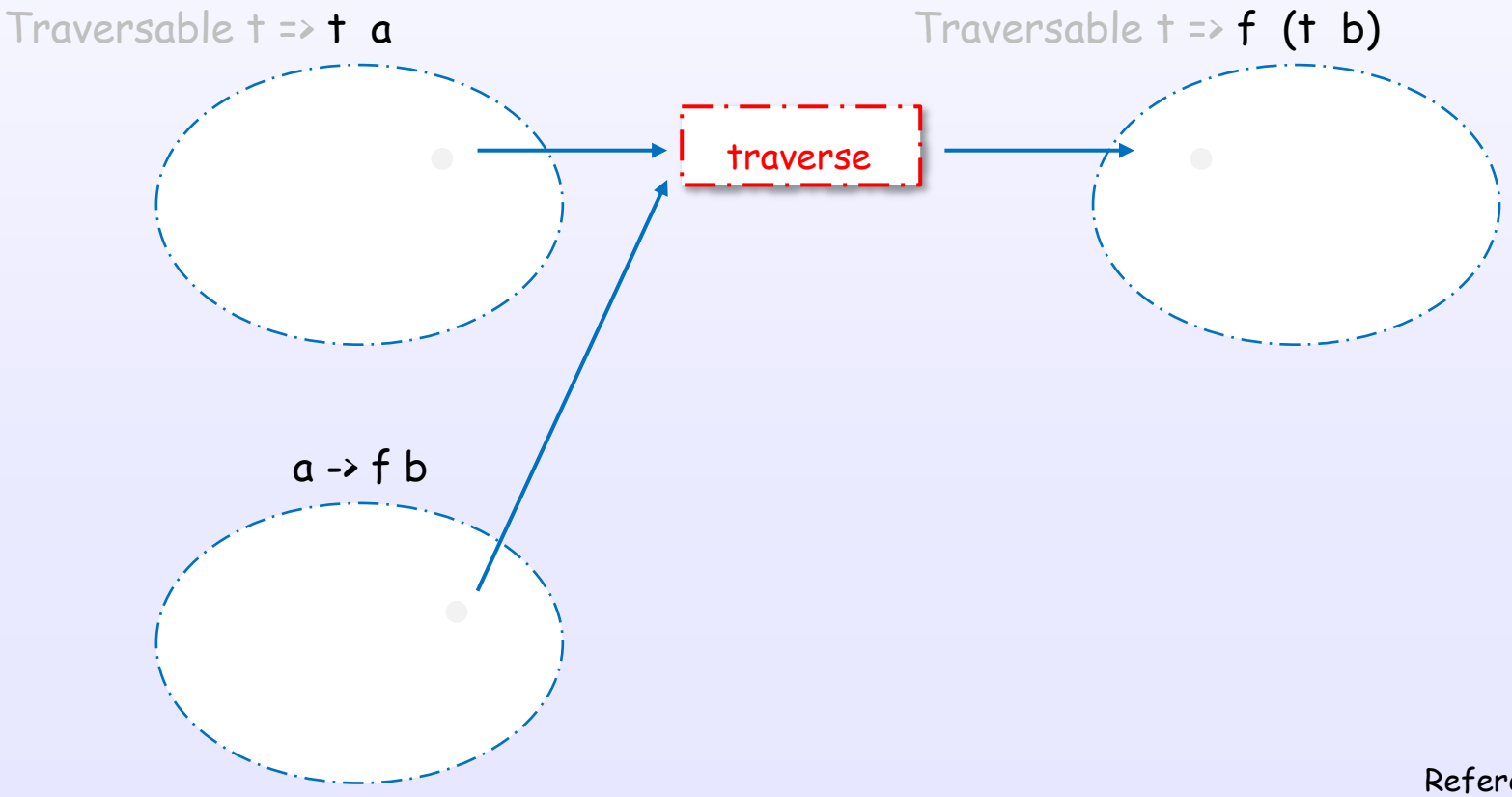


associative law

Programmers should satisfy the monoid laws.

References : [B1] Ch.12, [D3], [S1]

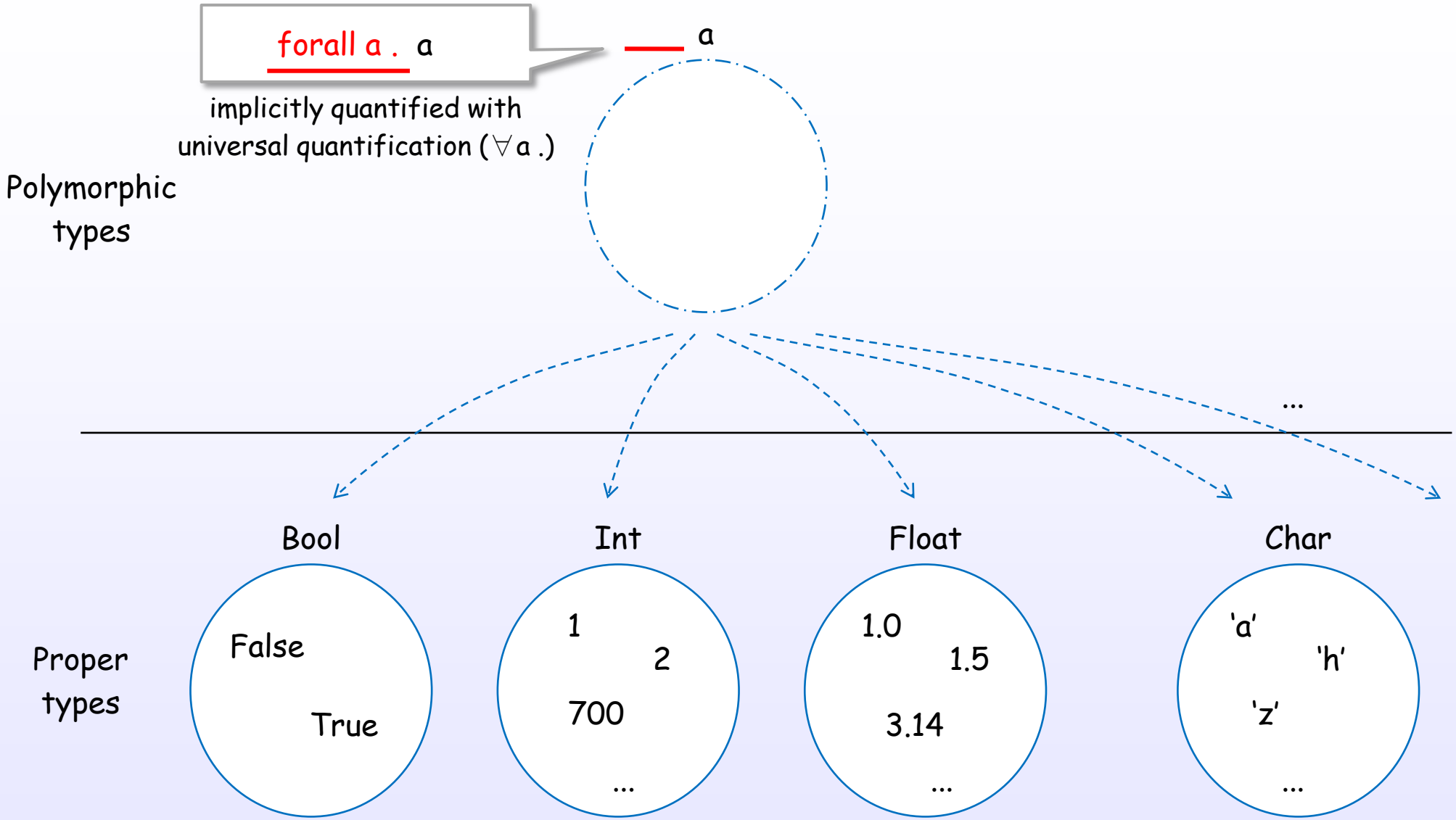
Traversable class's characteristic operations



References : [D3], [S1]

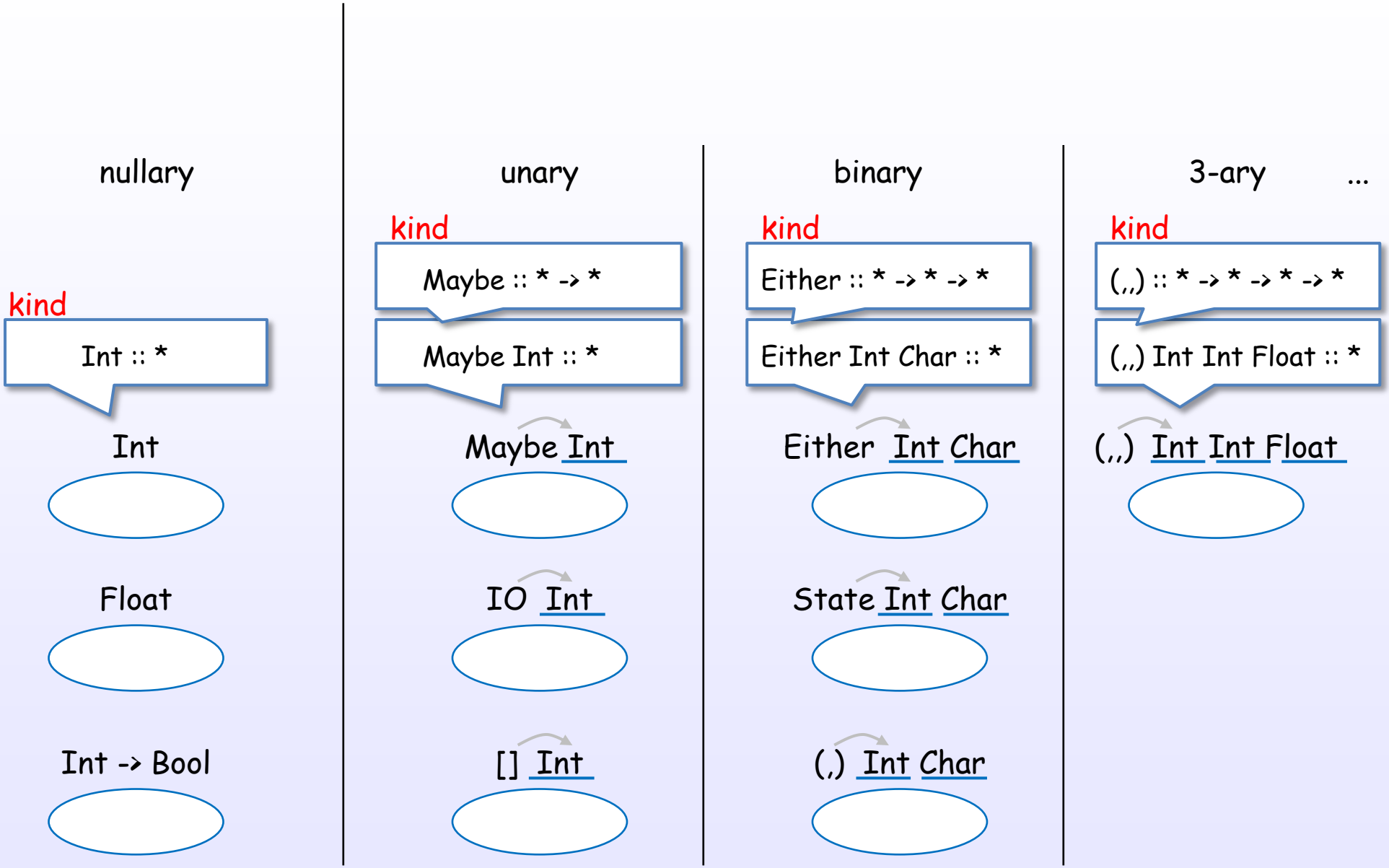
Appendix III – Advanced topics

Universally quantified types



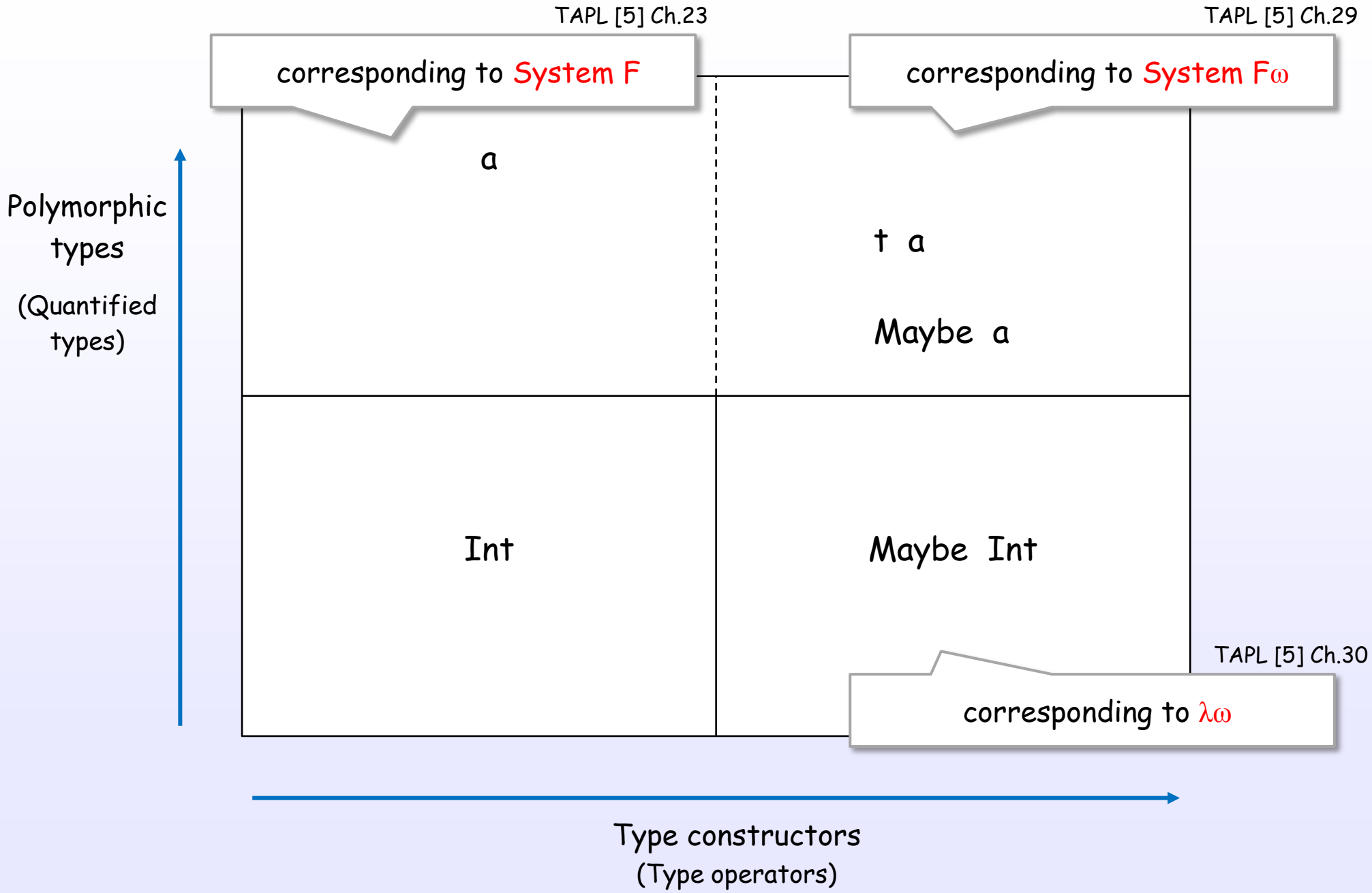
References : [B5] Ch.23, [H1] Ch.4, [H2] "GHC Language Features"

Kinds and type constructors



References : [B1] Ch.7, [B5] Ch.29, [H1] Ch.4

Type systems



References : [B5] Ch.23, 29, 30

References

References

Books

[B1] Learn You a Haskell for Great Good! (LYAH)
http://learnyouahaskell.com/

[B2] Thinking Functionally with Haskell (IFPH 3rd edition)
http://www.cs.ox.ac.uk/publications/books/functional/

[B3] Programming in Haskell
http://www.cs.nott.ac.uk/~pszgmh/book.html

[B4] Real World Haskell (RWH)
http://book.realworldhaskell.org/

[B5] Types and Programming Languages (TAPL)
https://mitpress.mit.edu/books/types-and-programming-languages

Documents

[D1] CIS 194: Introduction to Haskell
http://www.seas.upenn.edu/~cis194/lectures.html

[D2] Type Systems
http://dev.stephendiehl.com/fun/004_type_systems.html

[D3] Typeclassopedia
http://www.cs.tufts.edu/comp/150FP/archive/brent-yorgey/tc.pdf
https://wiki.haskell.org/Typeclassopedia

References

Search

[S1] Hoogle
https://www.haskell.org/hoogle

Specifications

[H1] Haskell 2010 Language Report
https://www.haskell.org/definition/haskell2010.pdf

[H2] The Glorious Glasgow Haskell Compilation System (GHC user's guide)
https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/index.html
https://downloads.haskell.org/~ghc/latest/docs/users_guide.pdf

References

Furthermore readings

[A1] What I Wish I Knew When Learning Haskell
http://dev.stephendiehl.com/hask/

[A2] How to learn Haskell
https://github.com/bitemyapp/learnhaskell

[A3] Documentation
https://www.haskell.org/documentation

[A4] A Haskell Implementation Reading List
http://www.stephendiehl.com/posts/essential_compilers.html

[A5] The GHC reading list
https://ghc.haskell.org/trac/ghc/wiki/ReadingList