

Type introduction illustrated

for casual haskellers

to get over the Foldable

Takenobu T.

“What is this description ?!”

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

NOTE

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

Contents

1. Introduction

- Values, Types, Type classes
- Polymorphic types
- Type constructors
- Polymorphic and type constructors

2. more, Types and Type classes

- Function types
- Type class operations

3. What is this?

Appendix I - Various types

- Bool, Char, Int, Float
- Maybe, List, Either, Tuple

Appendix II - Various type classes

- Eq, Ord
- Num
- Foldable
- Functor, Applicative, Monad
- Monoid
- Traversable

Appendix III - Advanced topics

References

1. Introduction

1. Introduction

Values, Types, Type classes

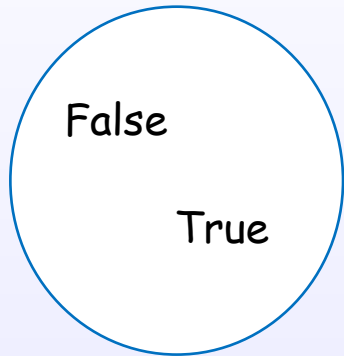
Values

False 1 2 1.0 'a' 'h'

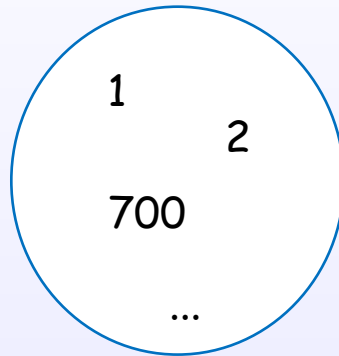
True 700 1.5 3.14 '5'

Types

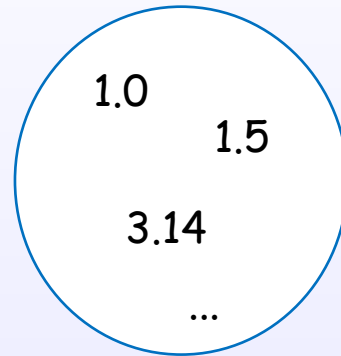
"Bool" type



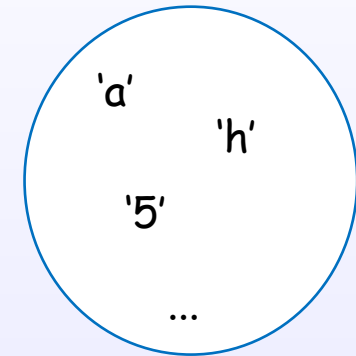
"Int" type



"Float" type

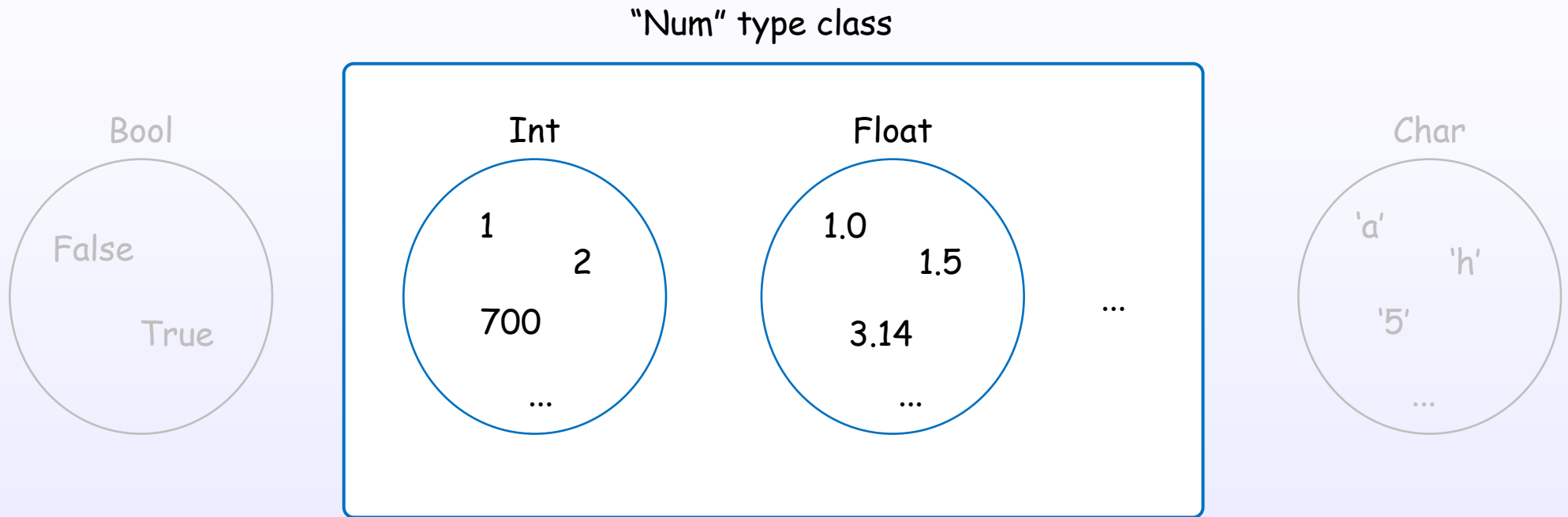


"Char" type



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

Type classes

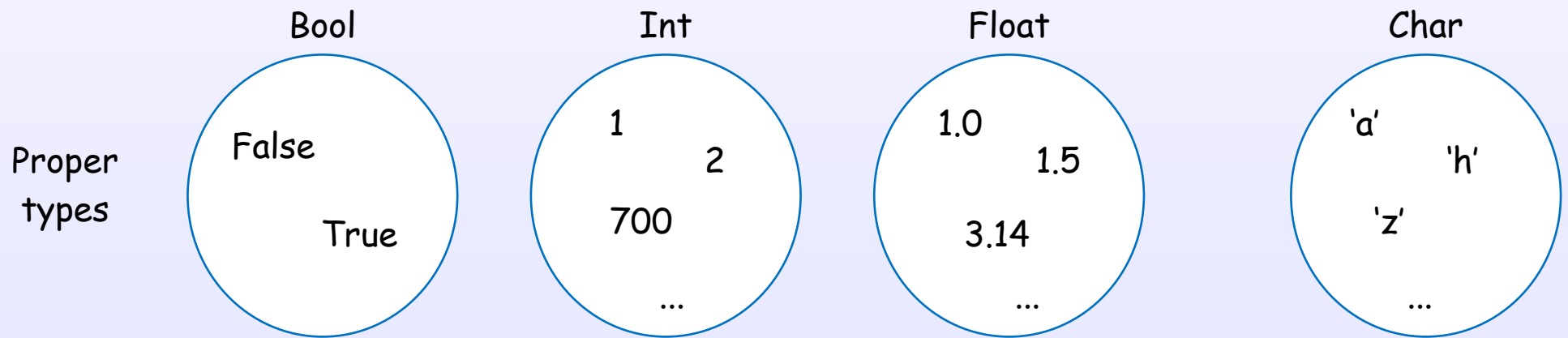


A type class is a collection of types which have common operations.

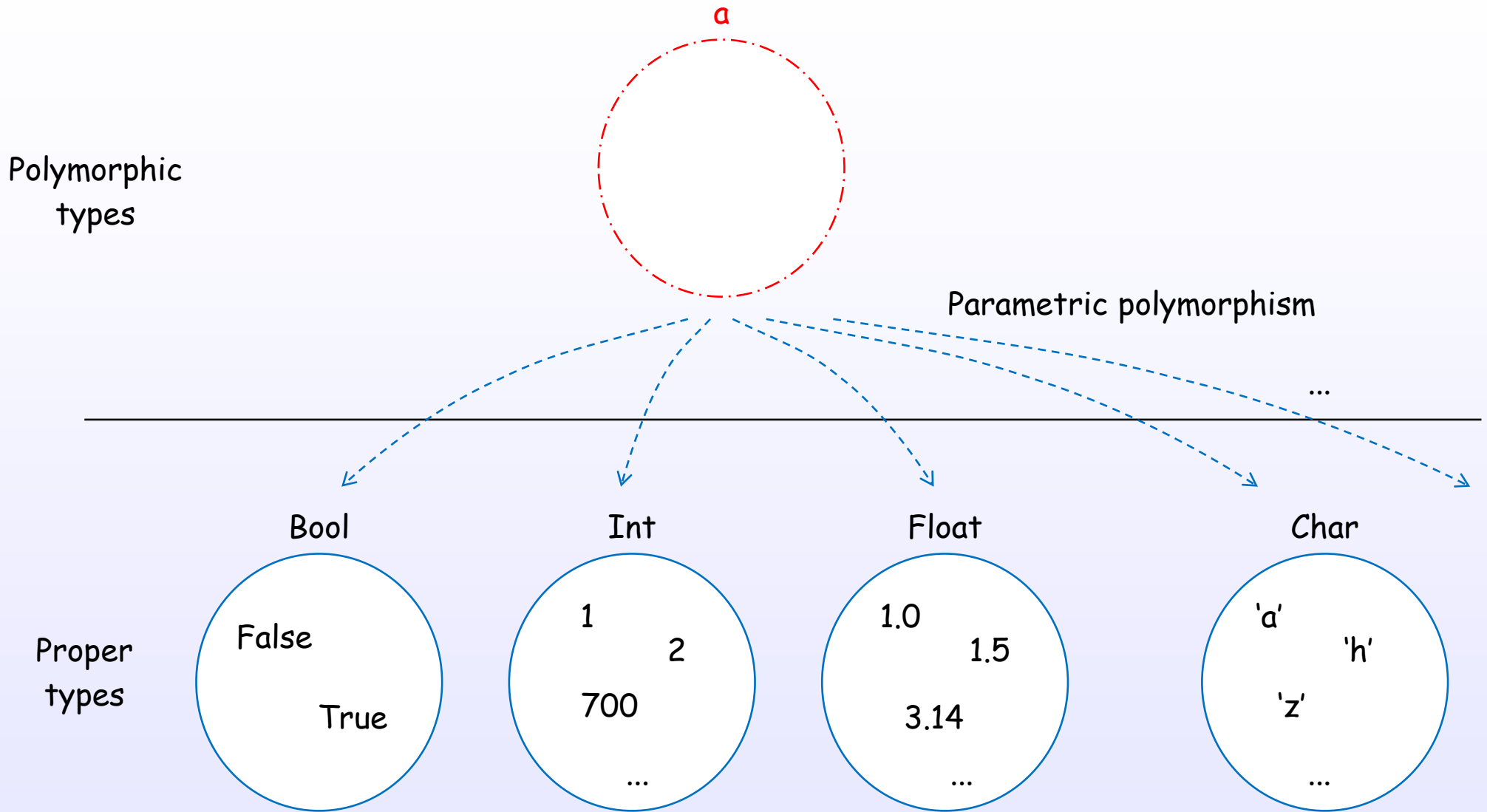
1. Introduction

Polymorphic types

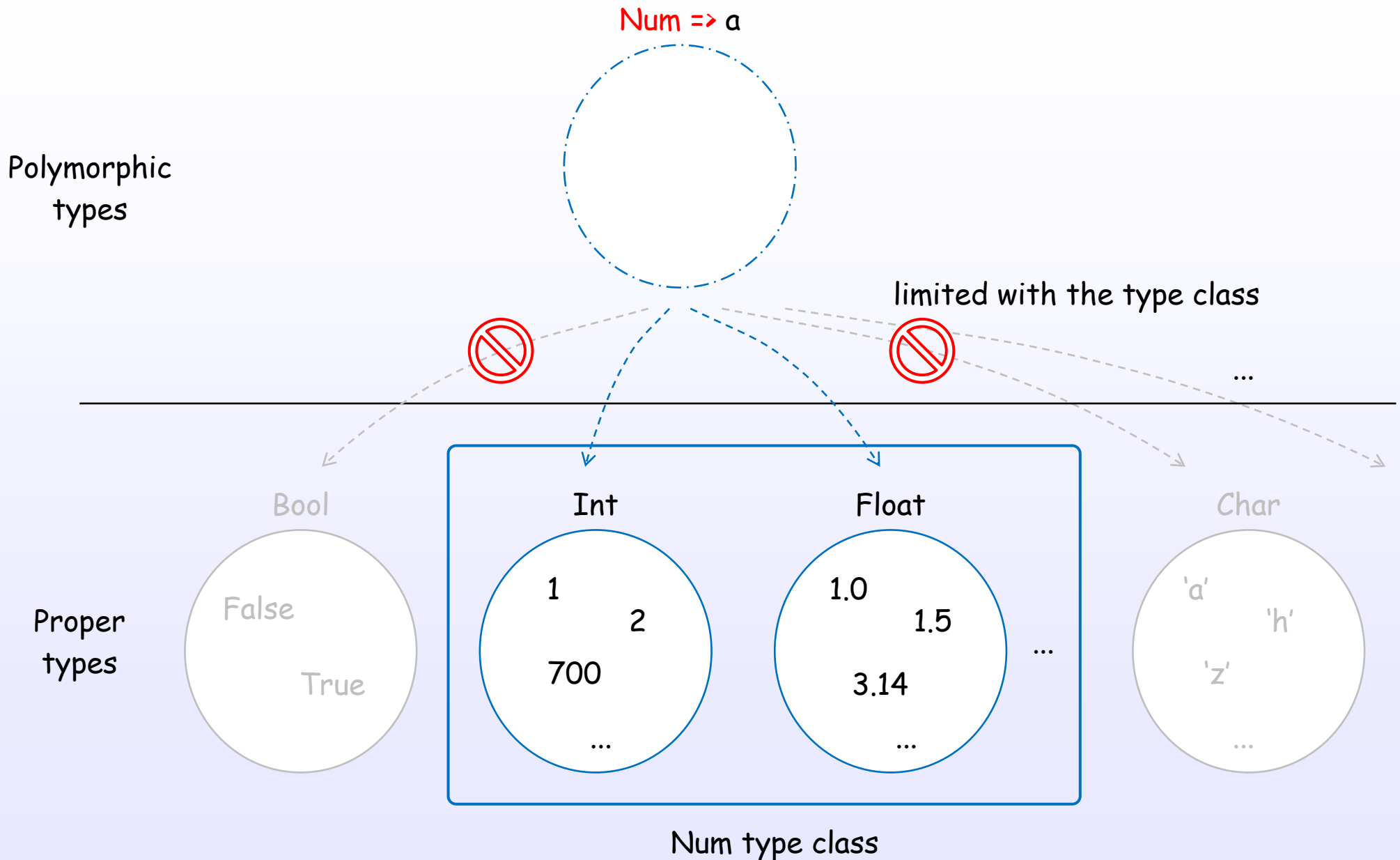
Proper types



Polymorphic types

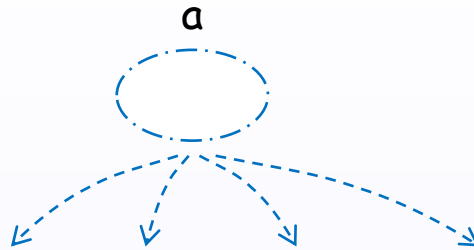


Polymorphic types restrained with type classes

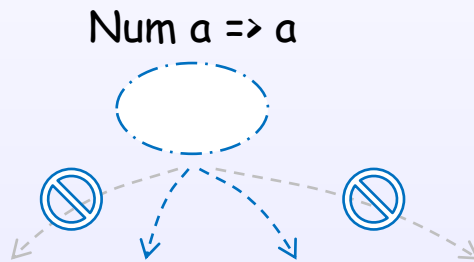


Polymorphic types

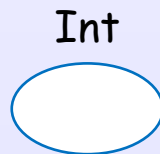
Polymorphic
types



polymorphic
types
with type class



Types



1. Introduction

Type constructors

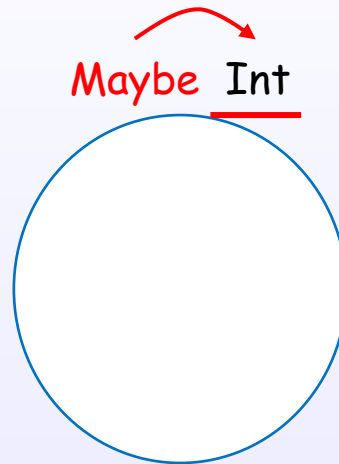
Type constructors

nullary type constructor



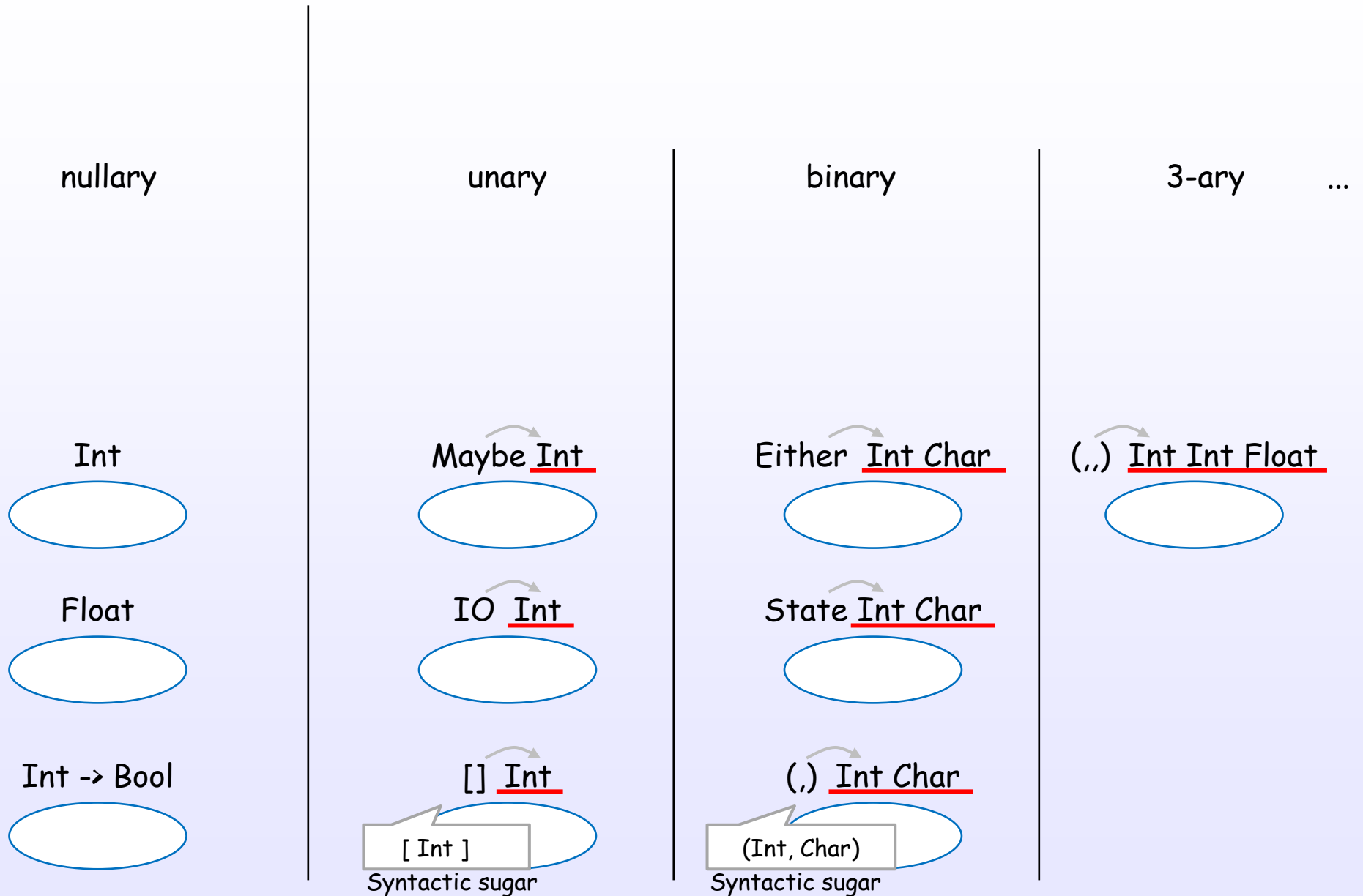
Type constructors

unary type constructor



"Maybe" type constructor takes one type argument.

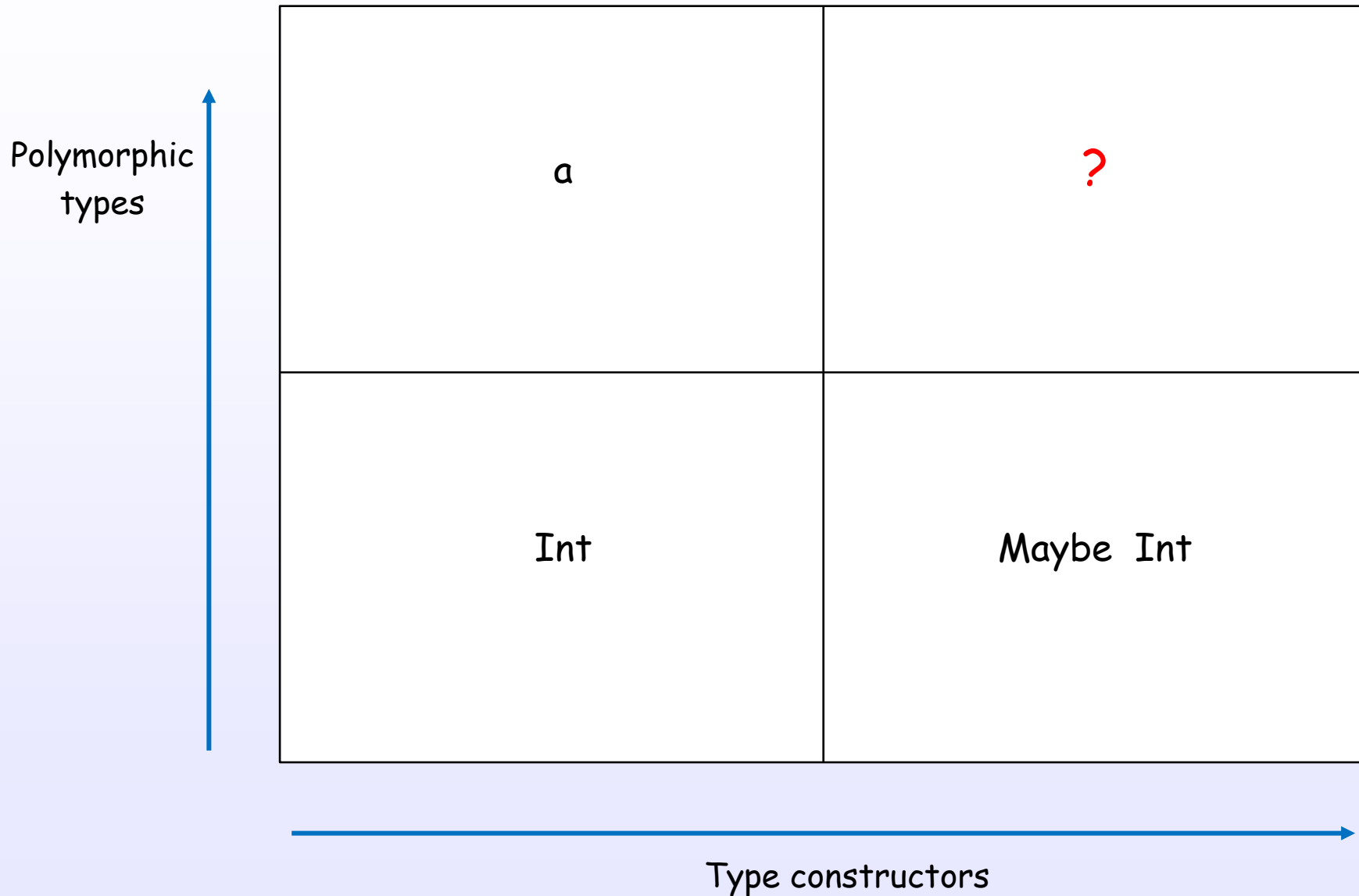
Type constructors



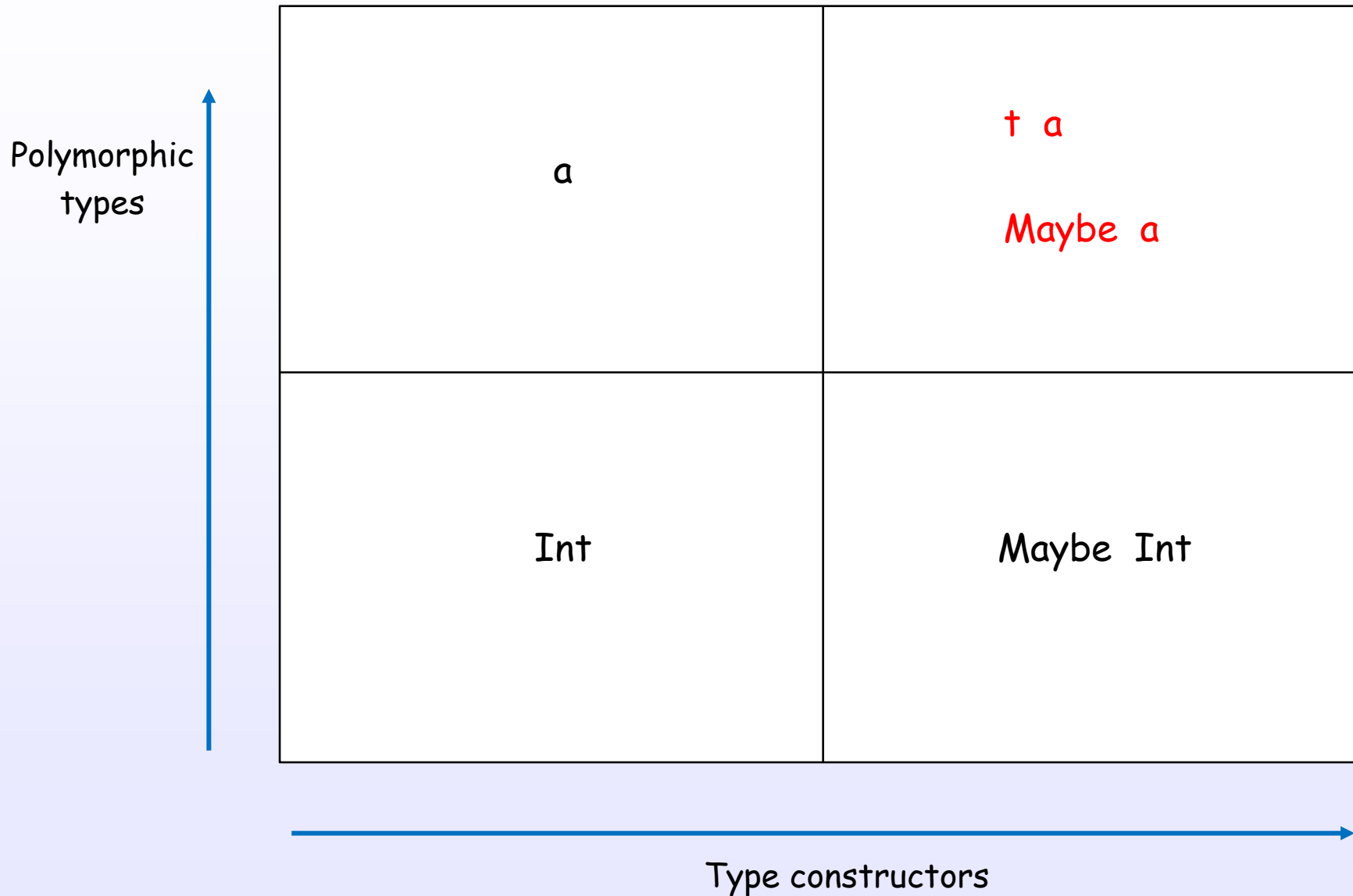
1. Introduction

Polymorphic types and type constructors

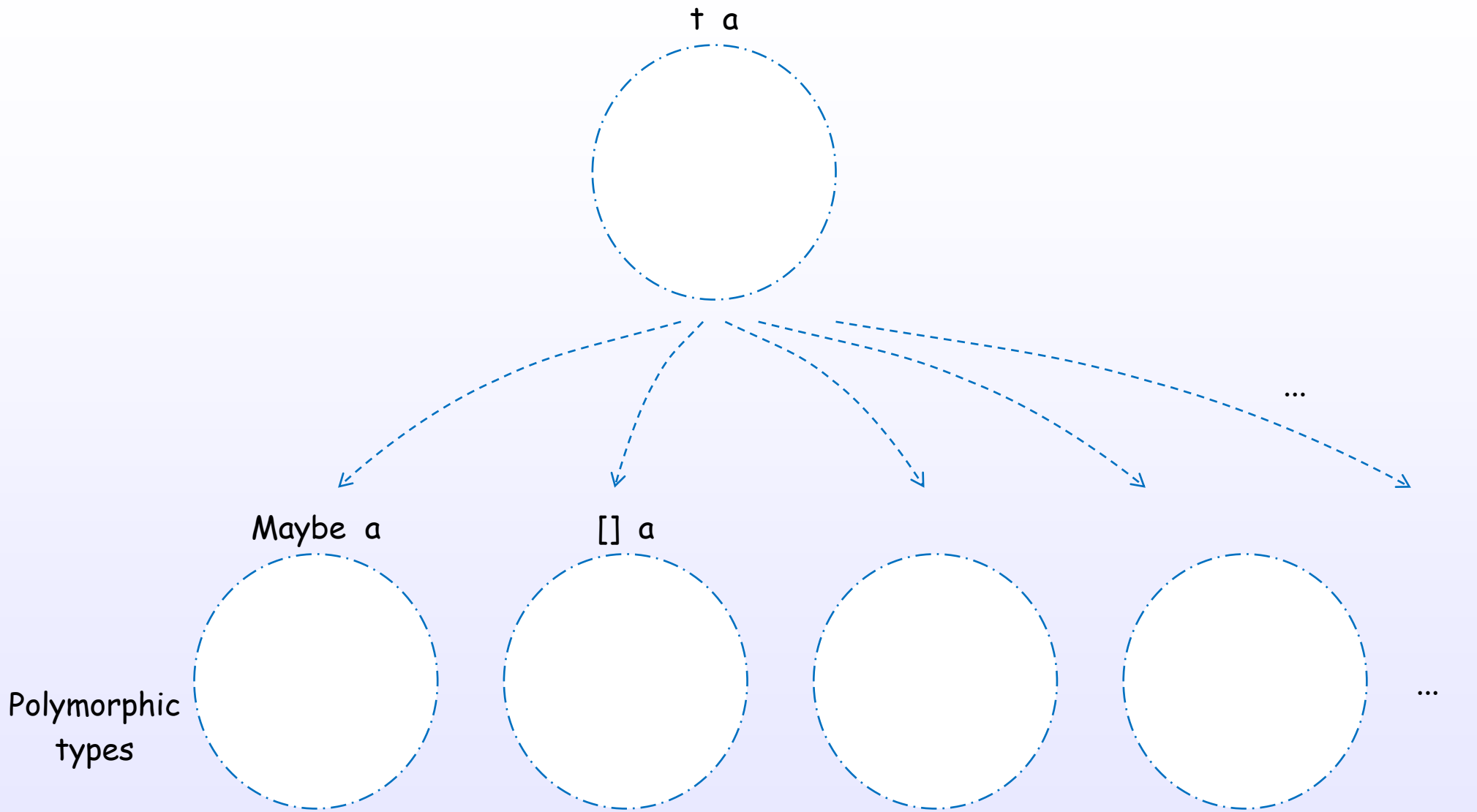
Polymorphic types and type constructors



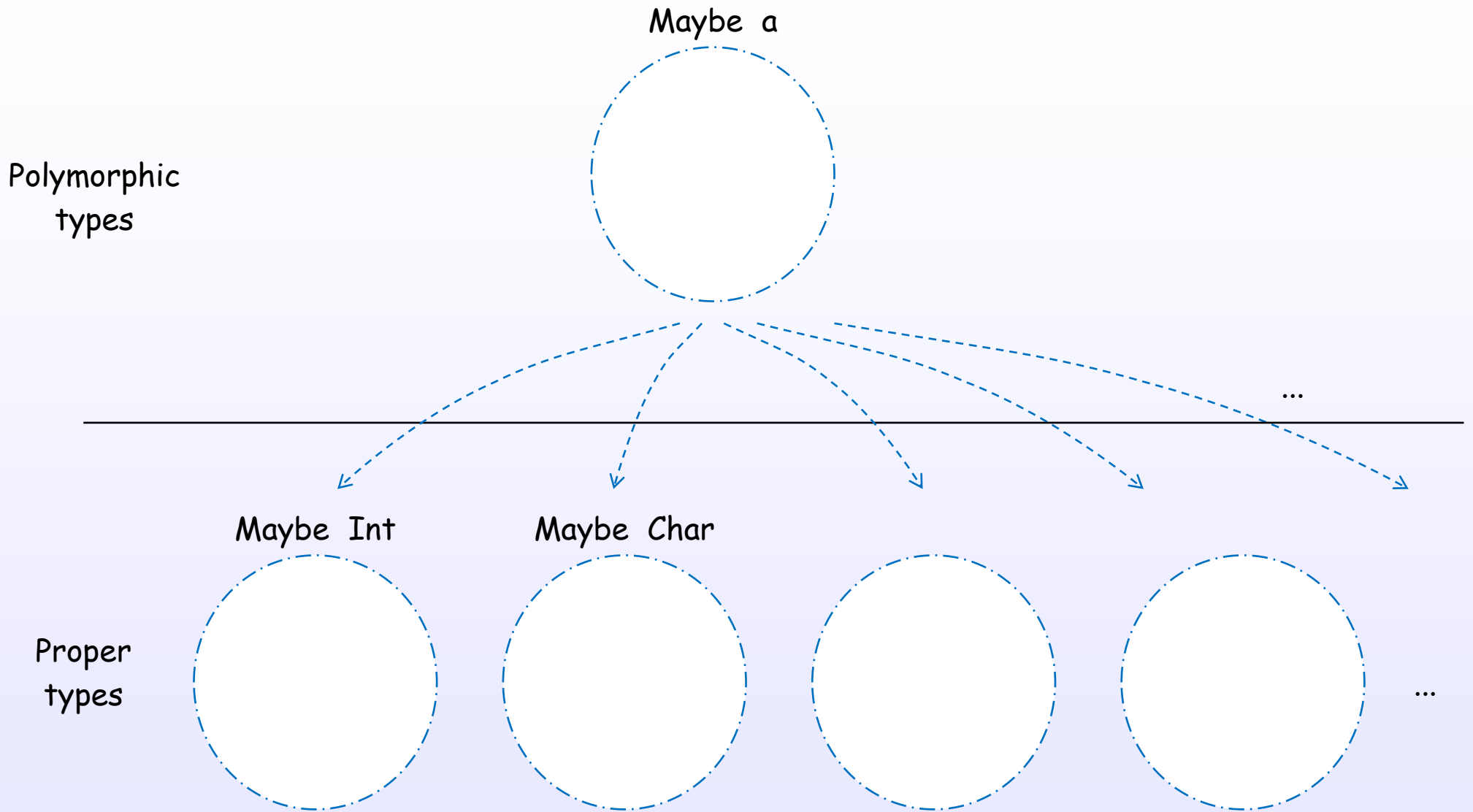
Polymorphic types and type constructors



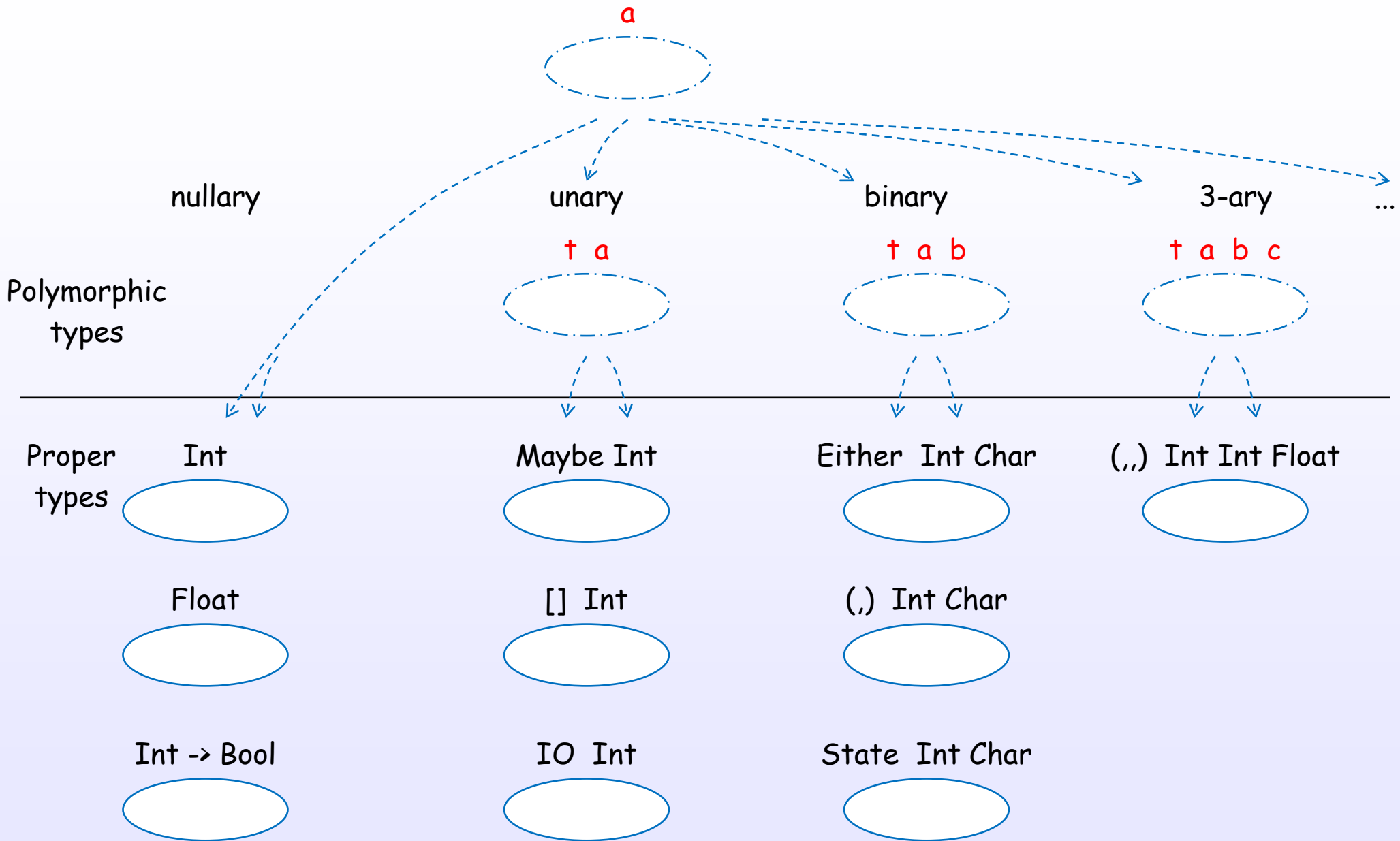
Polymorphic types and type constructors



Polymorphic types and type constructors



Polymorphic types and type constructors

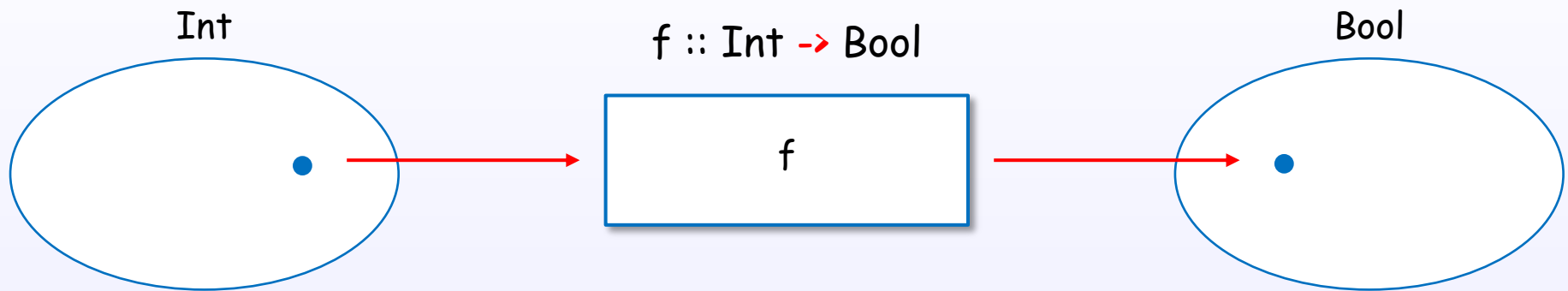


2. more, Types and Type classes

2. more, Types and Type classes

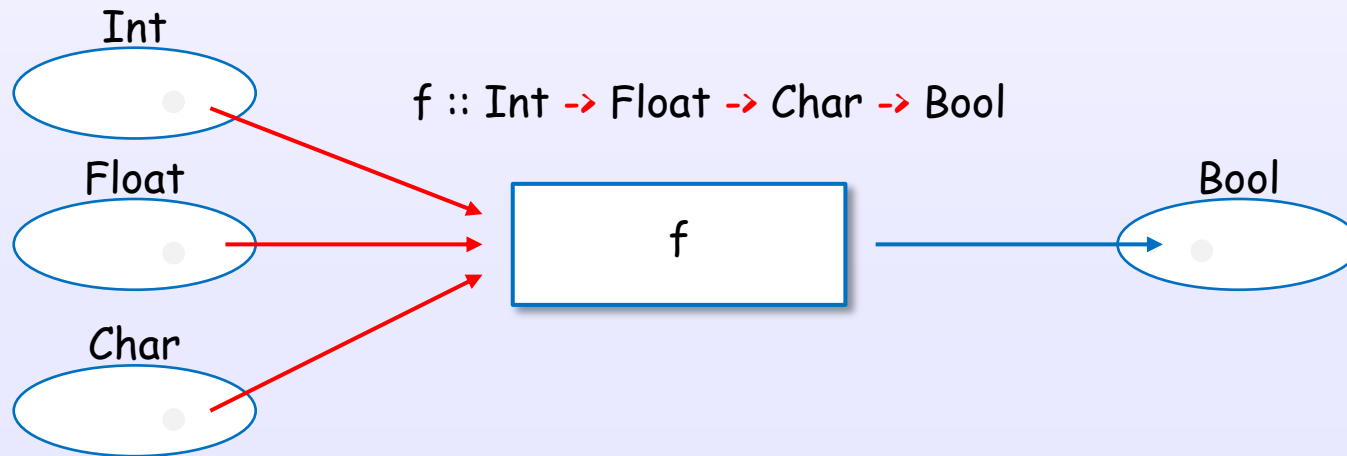
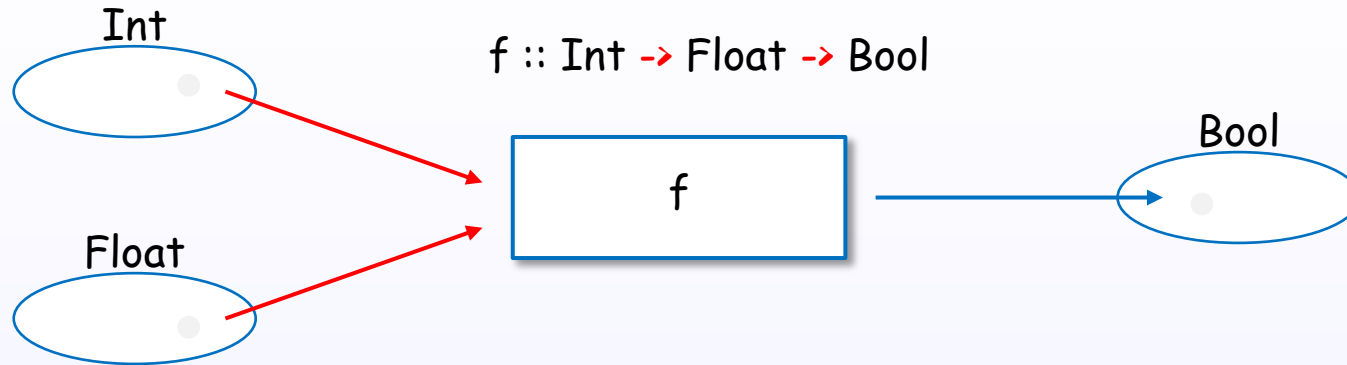
Function types

Function type

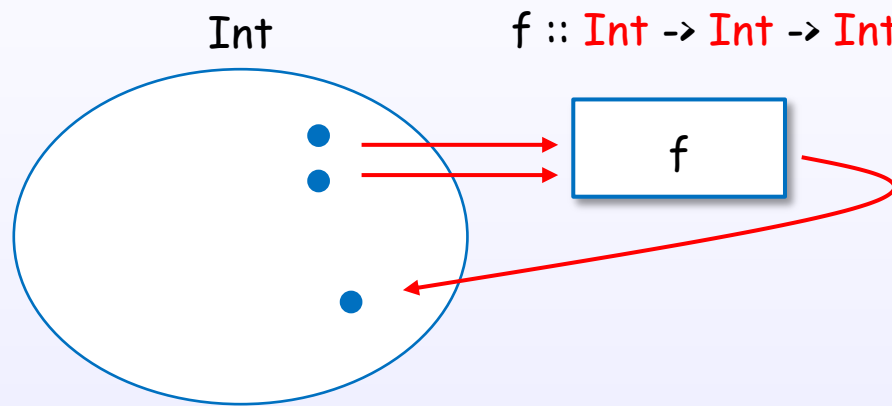


" \rightarrow " represents the function type.

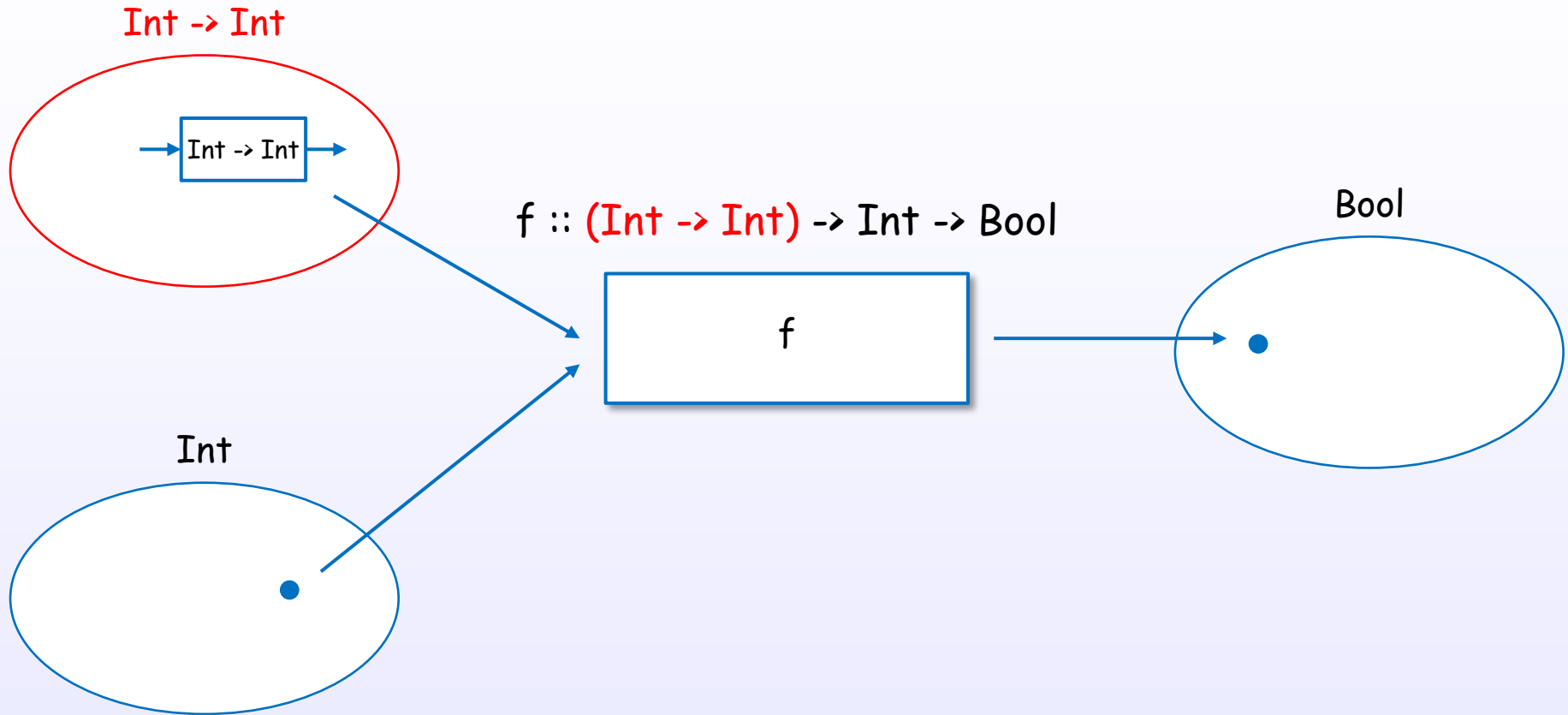
Function type for multiple arguments



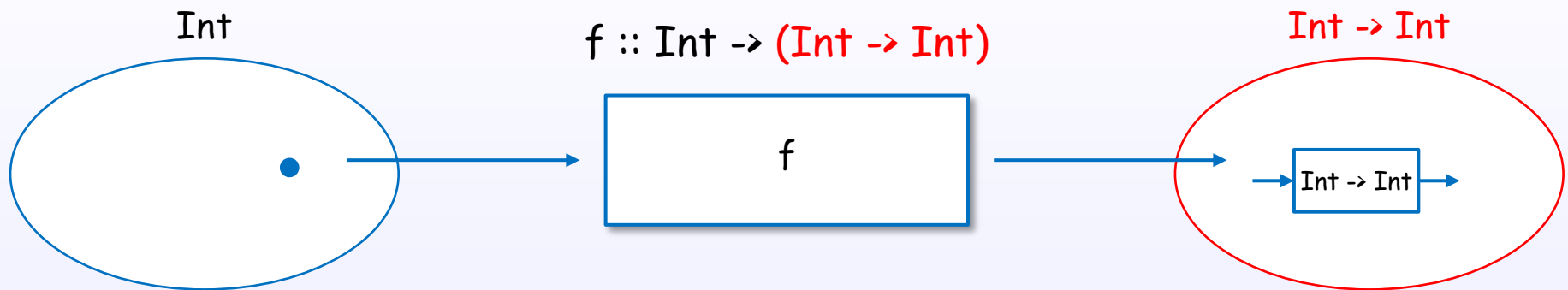
Function type for same type



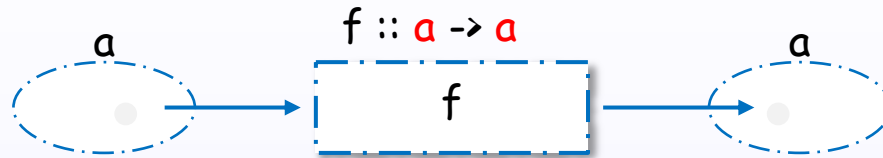
Function type for function argument



Function type for function result



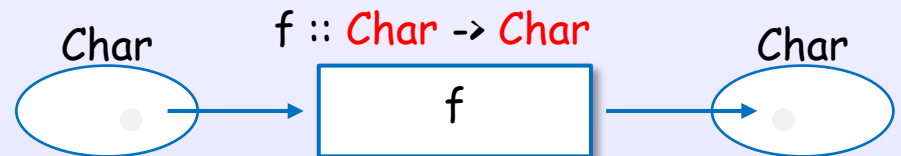
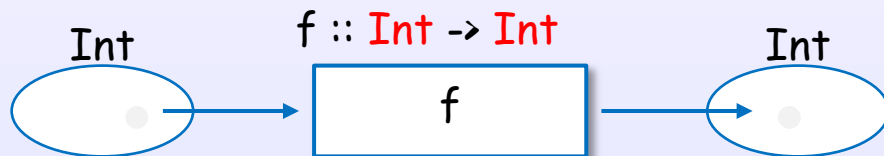
Function type for polymorphic function



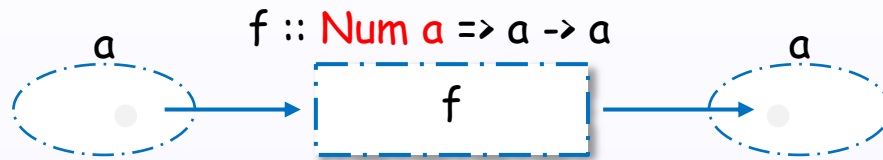
Polymorphic
types

Parametric polymorphism

Proper
types



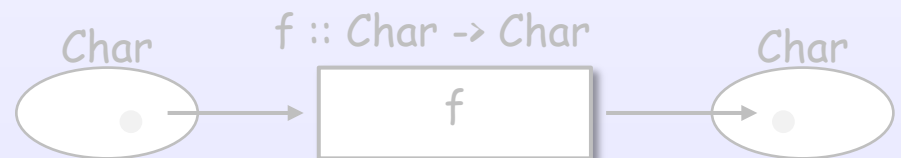
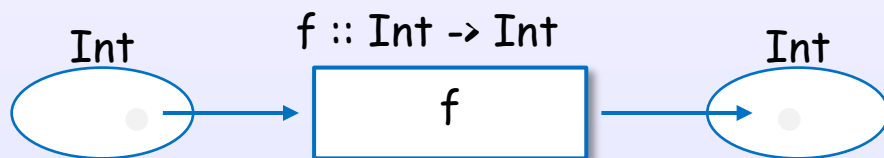
Function type for polymorphic function with type class



Polymorphic
types

Proper
types

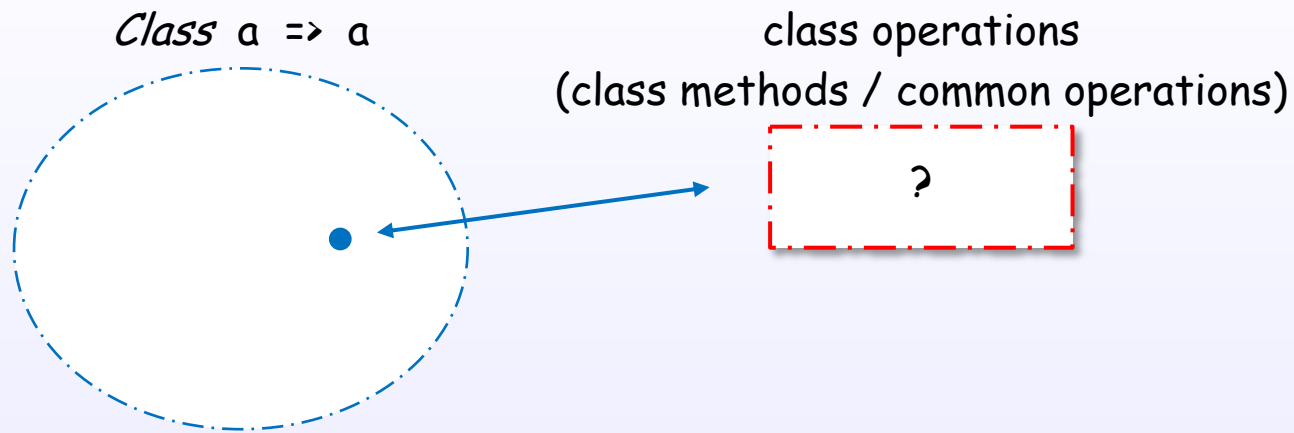
limited with the type class



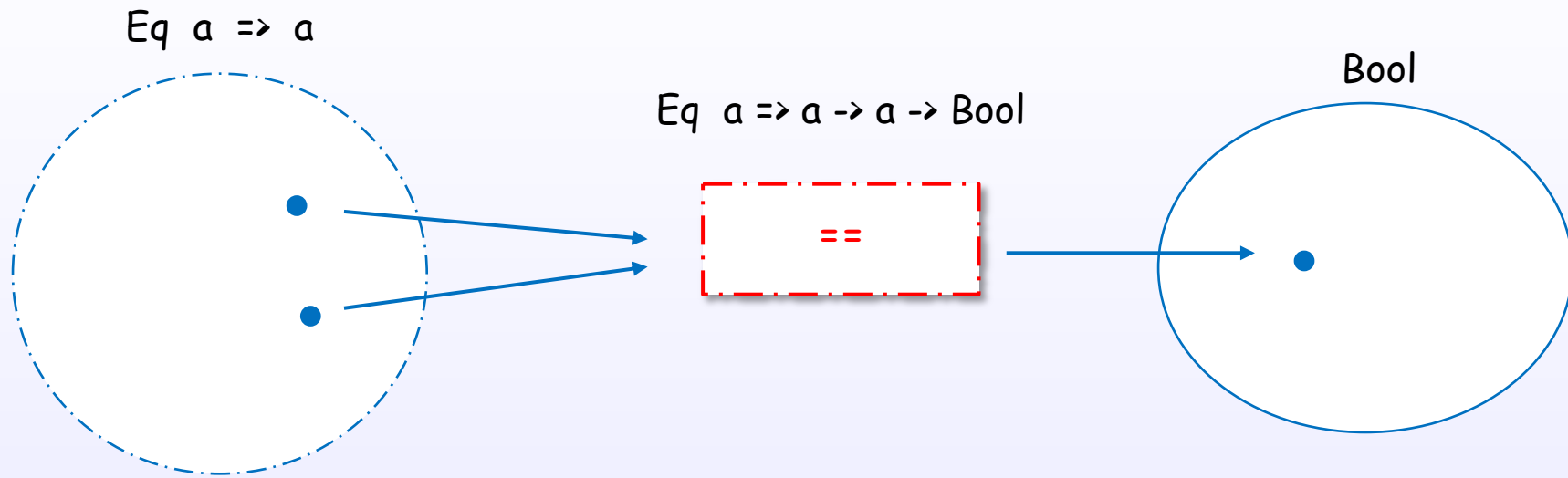
2. more, Types and Type classes

Type class operations

A type class has the class operations

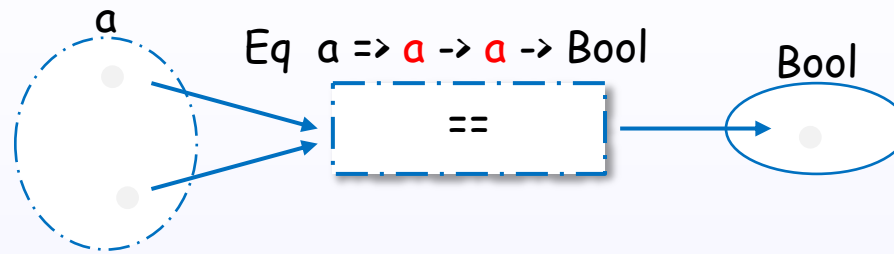


A type class has the class operations

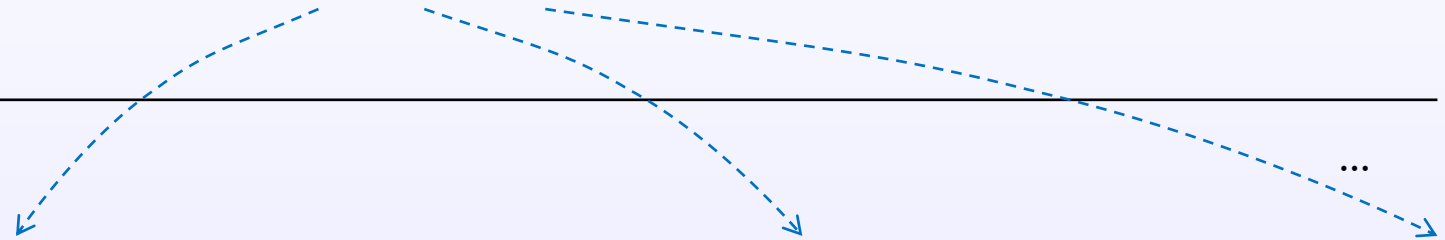


`Eq` class has `(==)` operation.

A type class has the class operations

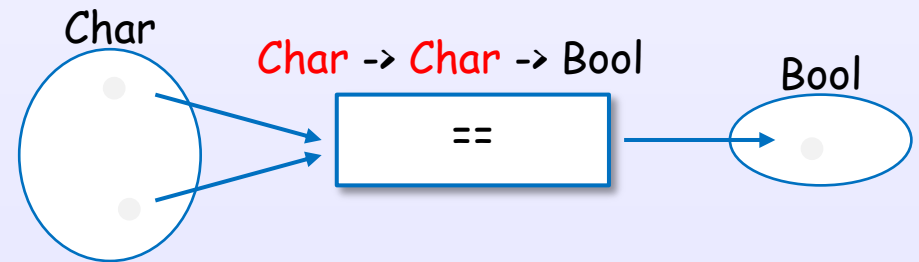
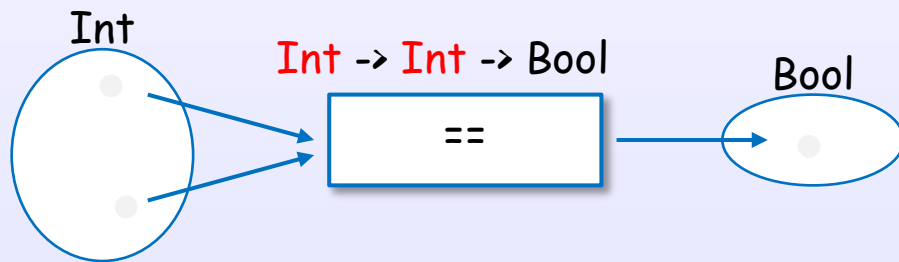


Ad-hoc polymorphism (overloading)



Polymorphic
types

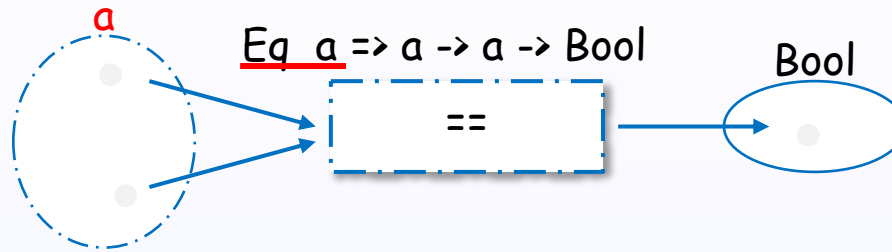
Proper
types



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

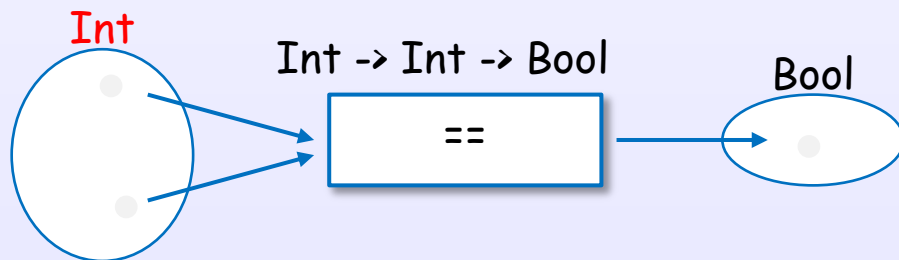
Definition of a type class and instances

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

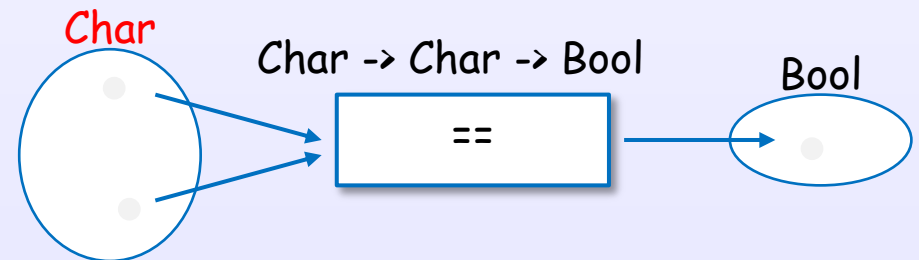


Polymorphic
types

Proper
types



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



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

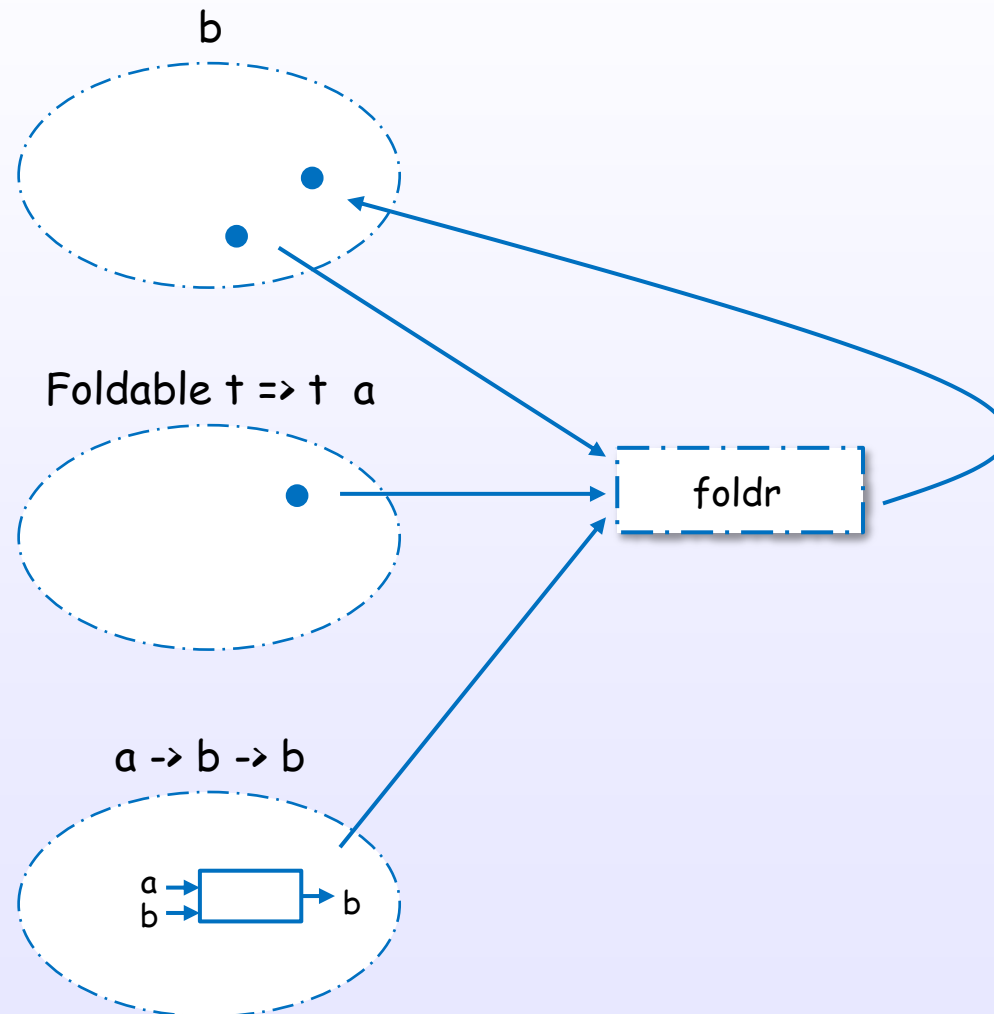
3. What is this?

What is this ?!

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

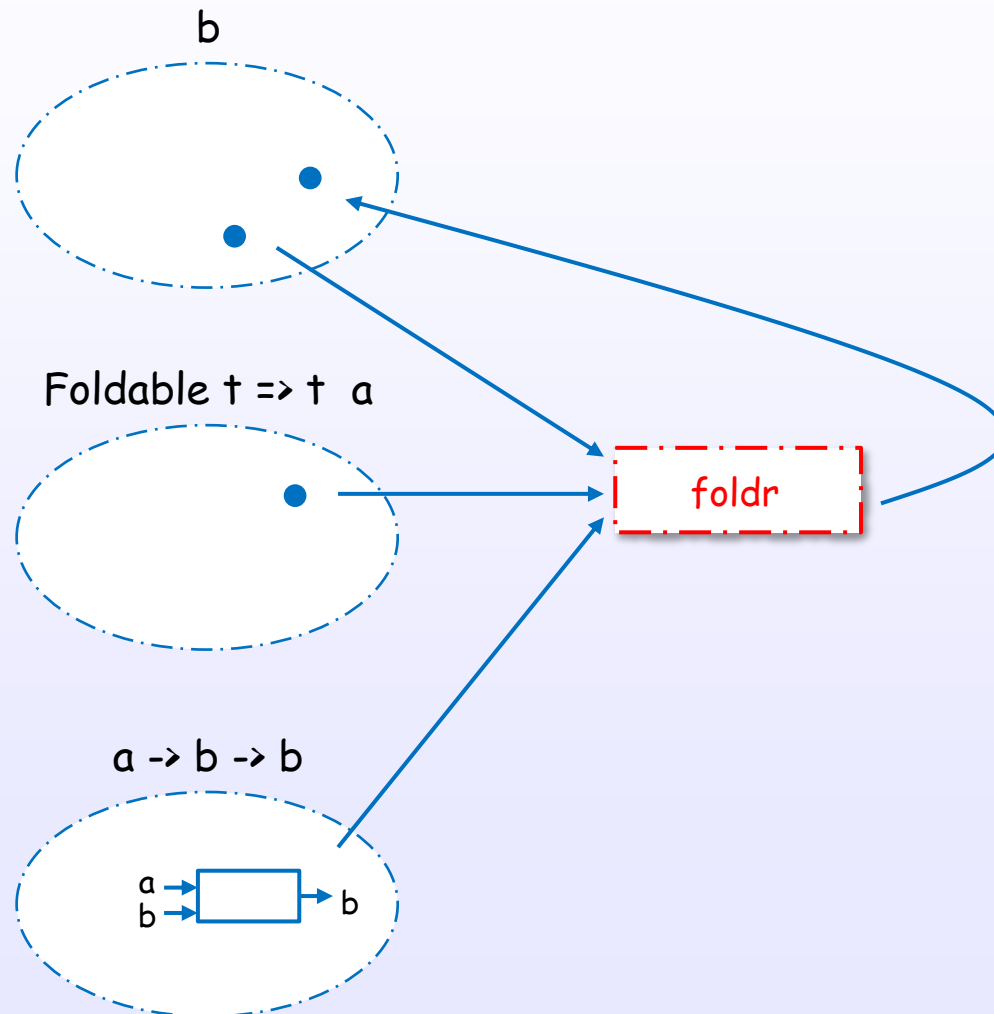
What is this ?!

$\text{foldr} :: \text{Foldable } t \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow b$



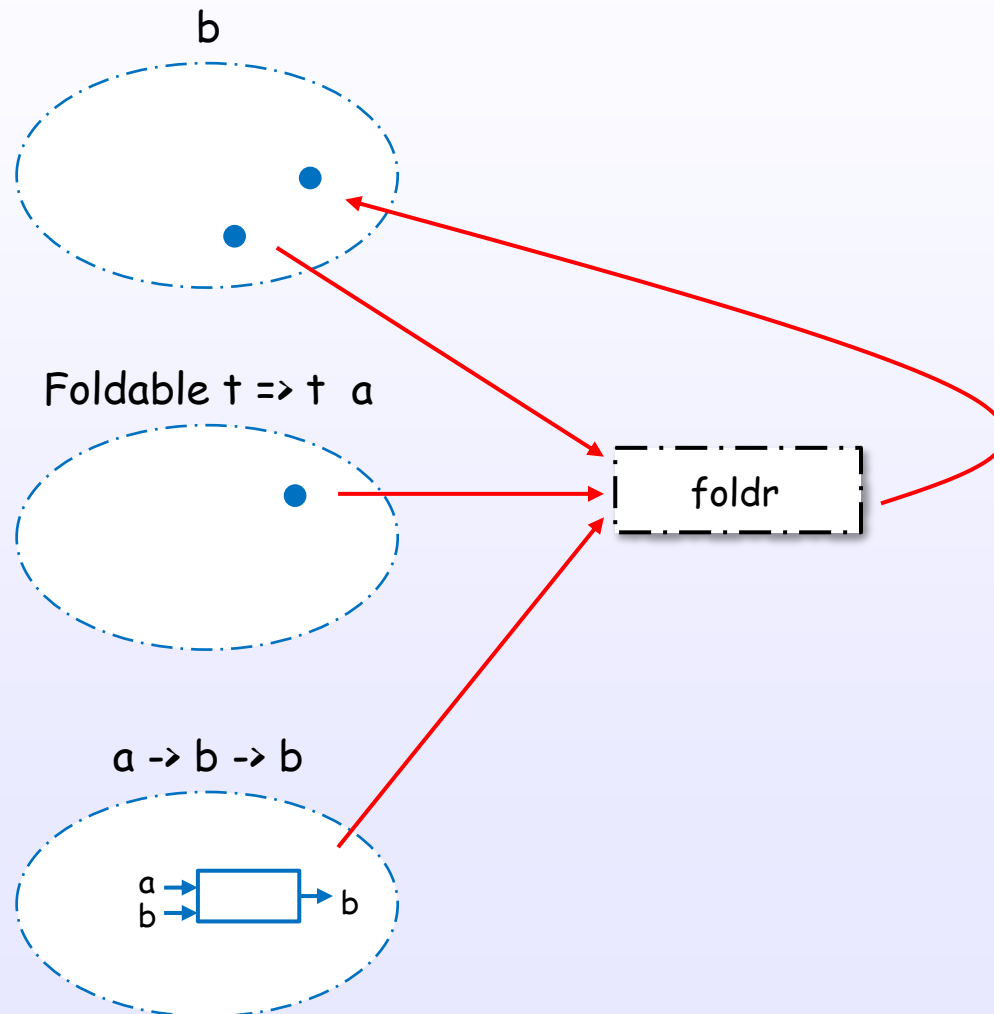
What is this ?!

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



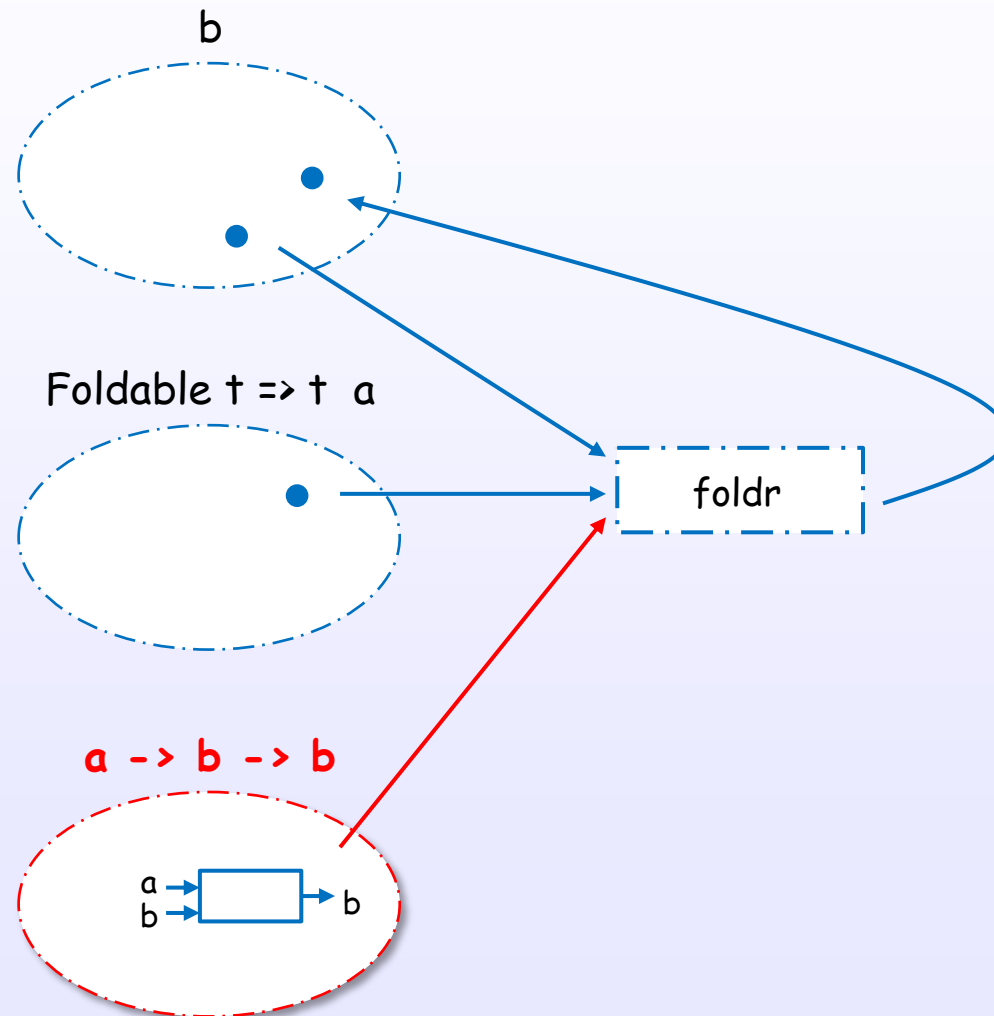
What is this ?!

$\text{foldr} :: \text{Foldable } t \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow b$



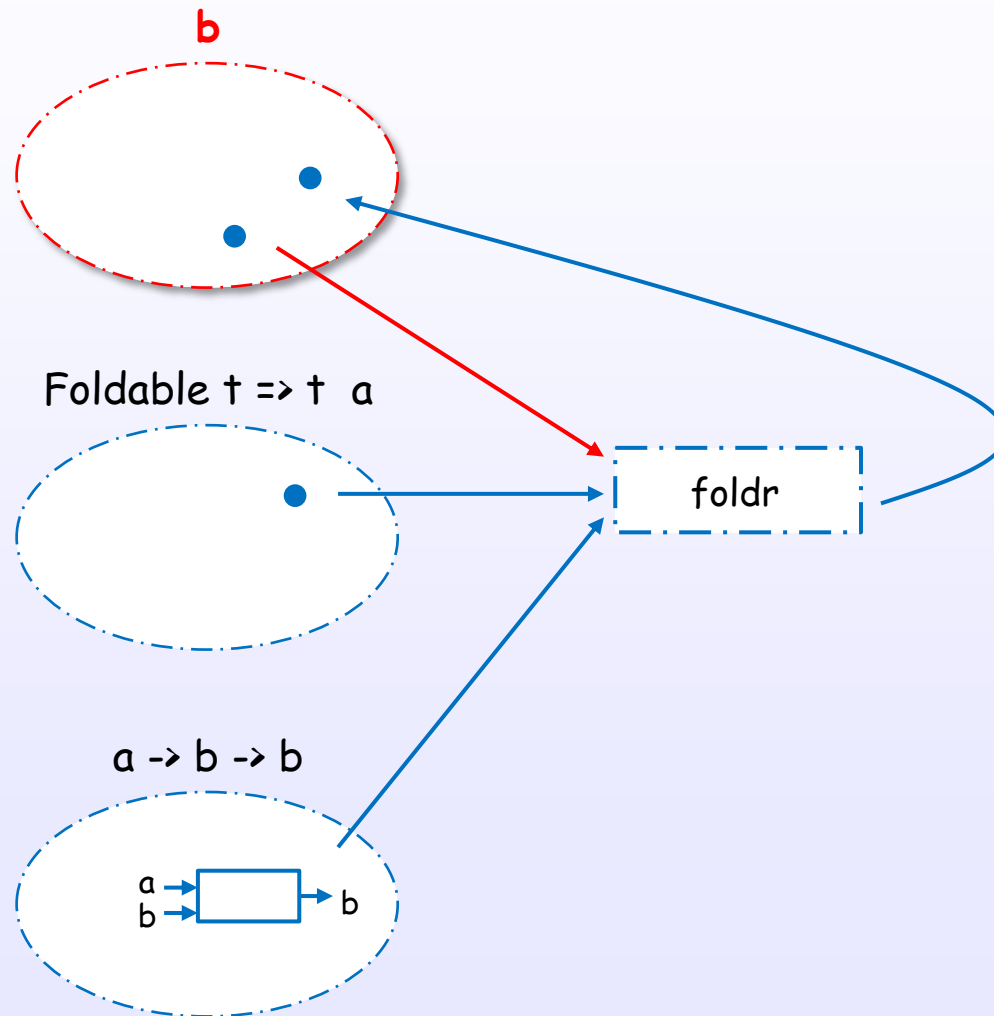
What is this ?!

$\text{foldr} :: \text{Foldable } t \Rightarrow \underline{(a \rightarrow b \rightarrow b)} \rightarrow b \rightarrow t\ a \rightarrow b$



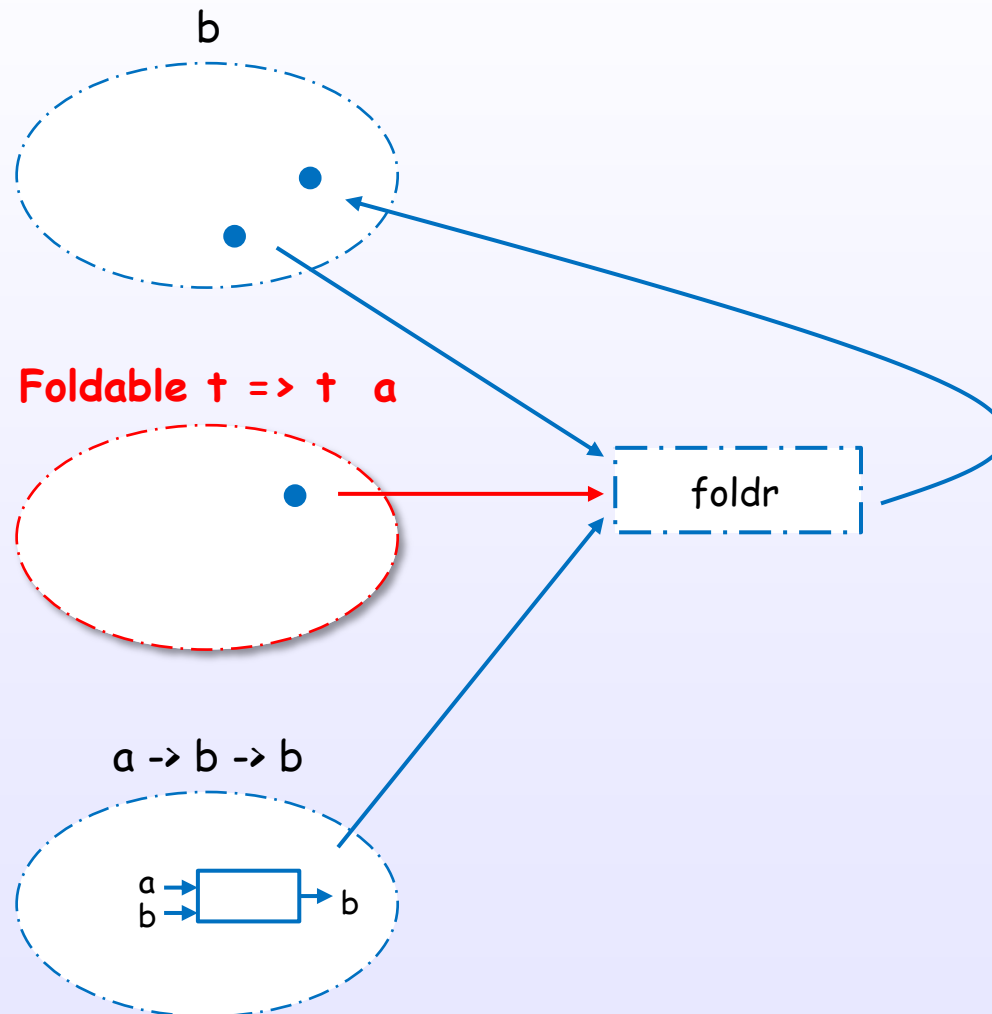
What is this ?!

$\text{foldr} :: \text{Foldable } t \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow \underline{b} \rightarrow t\ a \rightarrow b$



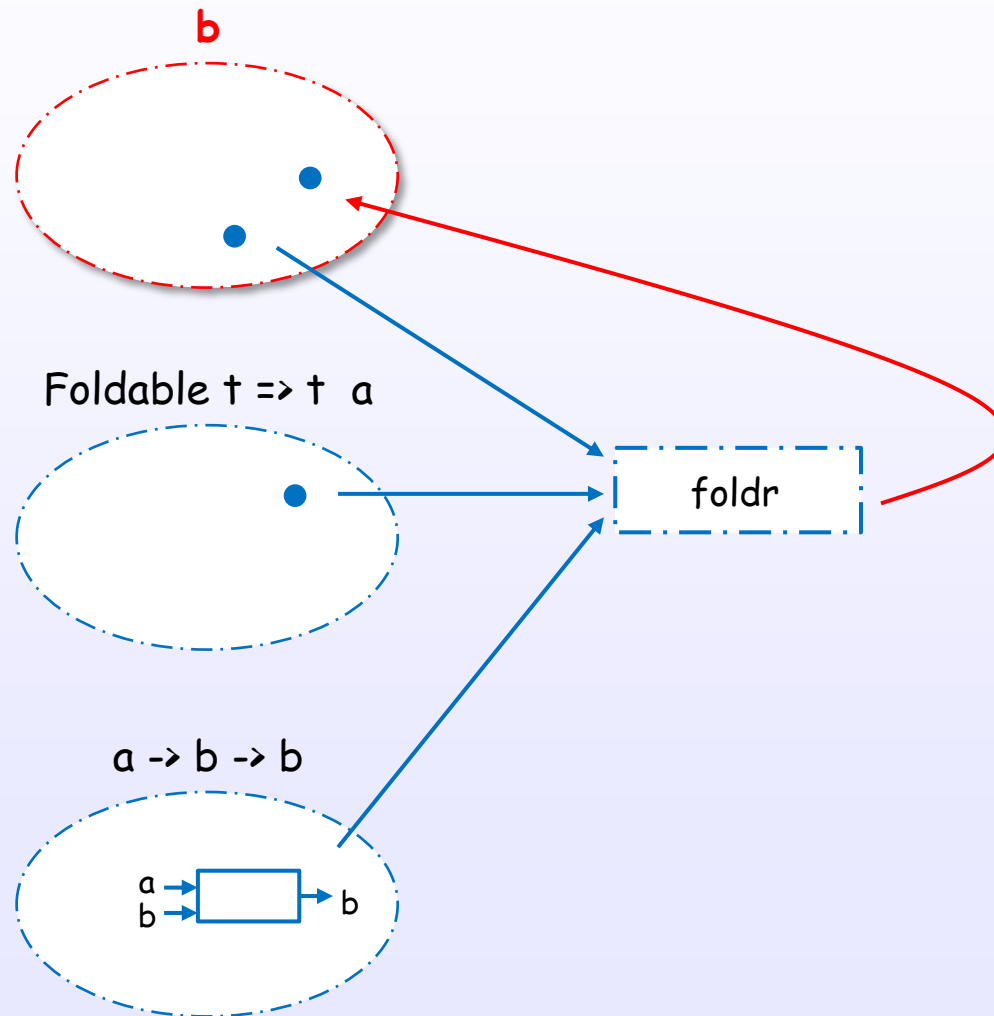
What is this ?!

$\text{foldr} :: \text{Foldable } \underline{t} \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \underline{t} \, a \rightarrow b$

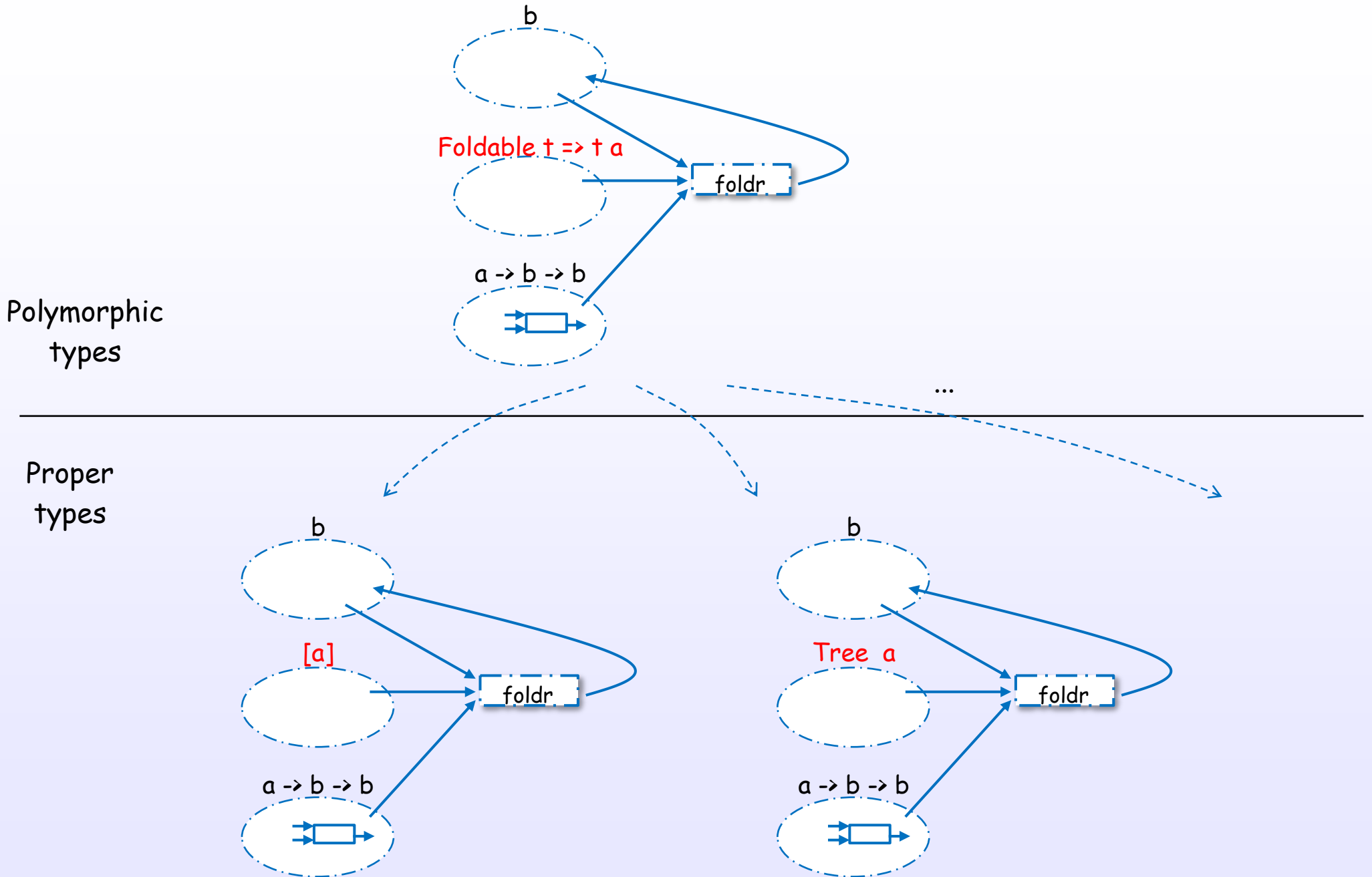


What is this ?!

$\text{foldr} :: \text{Foldable } t \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow \underline{b}$



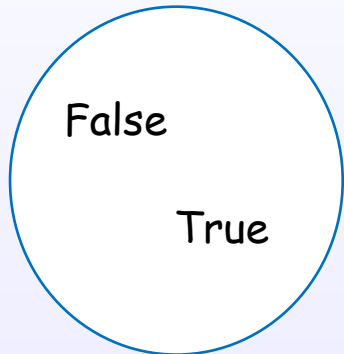
Example of polymorphism on foldr



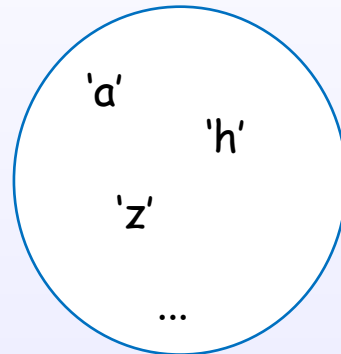
Appendix I - Various types

Bool, Char, Int, Float types

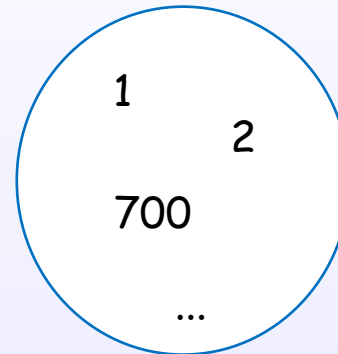
Bool



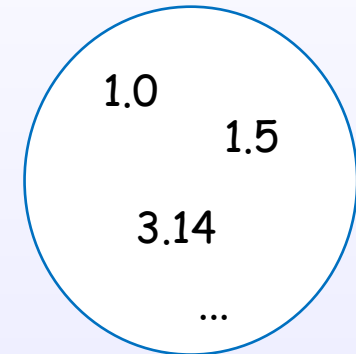
Char



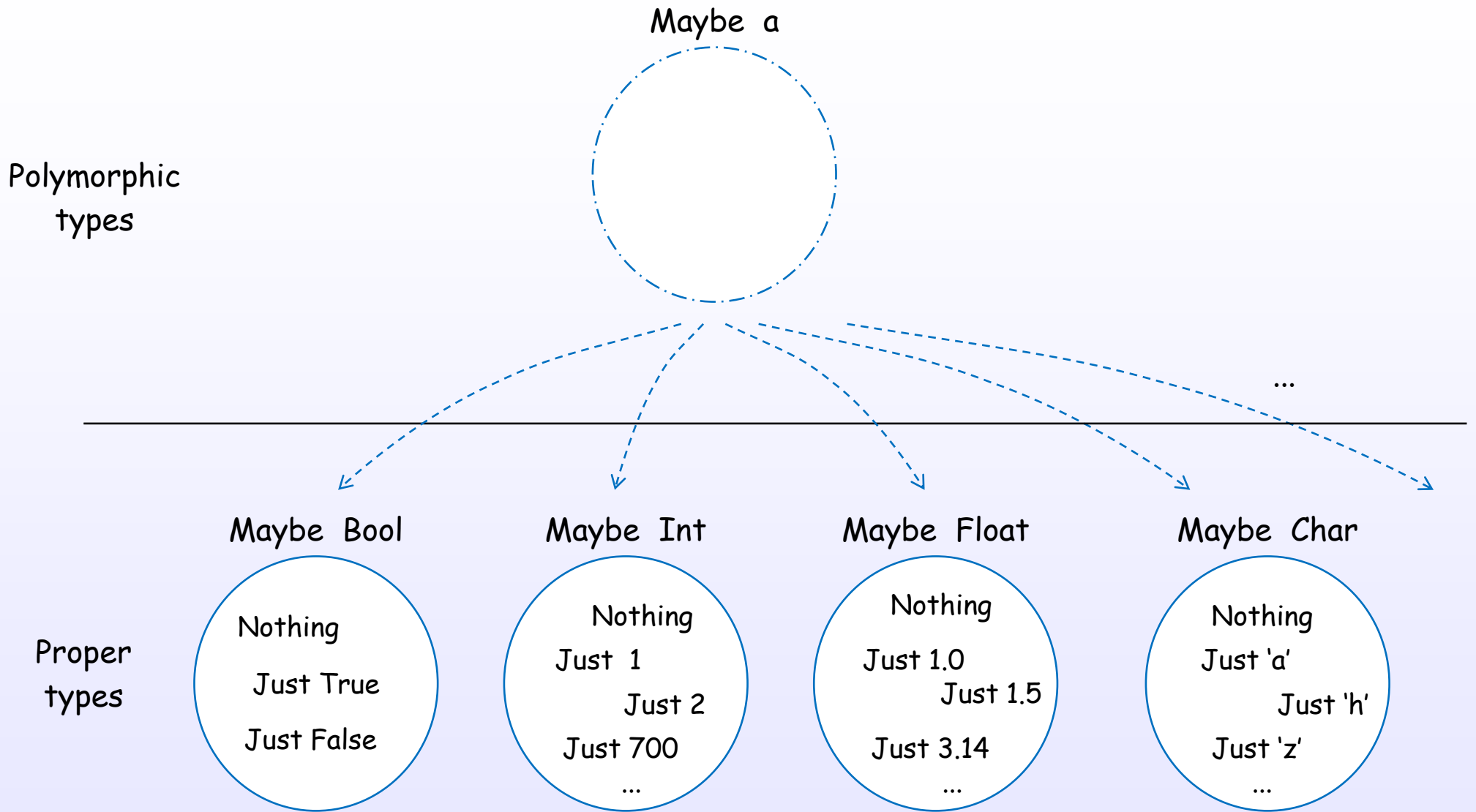
Int



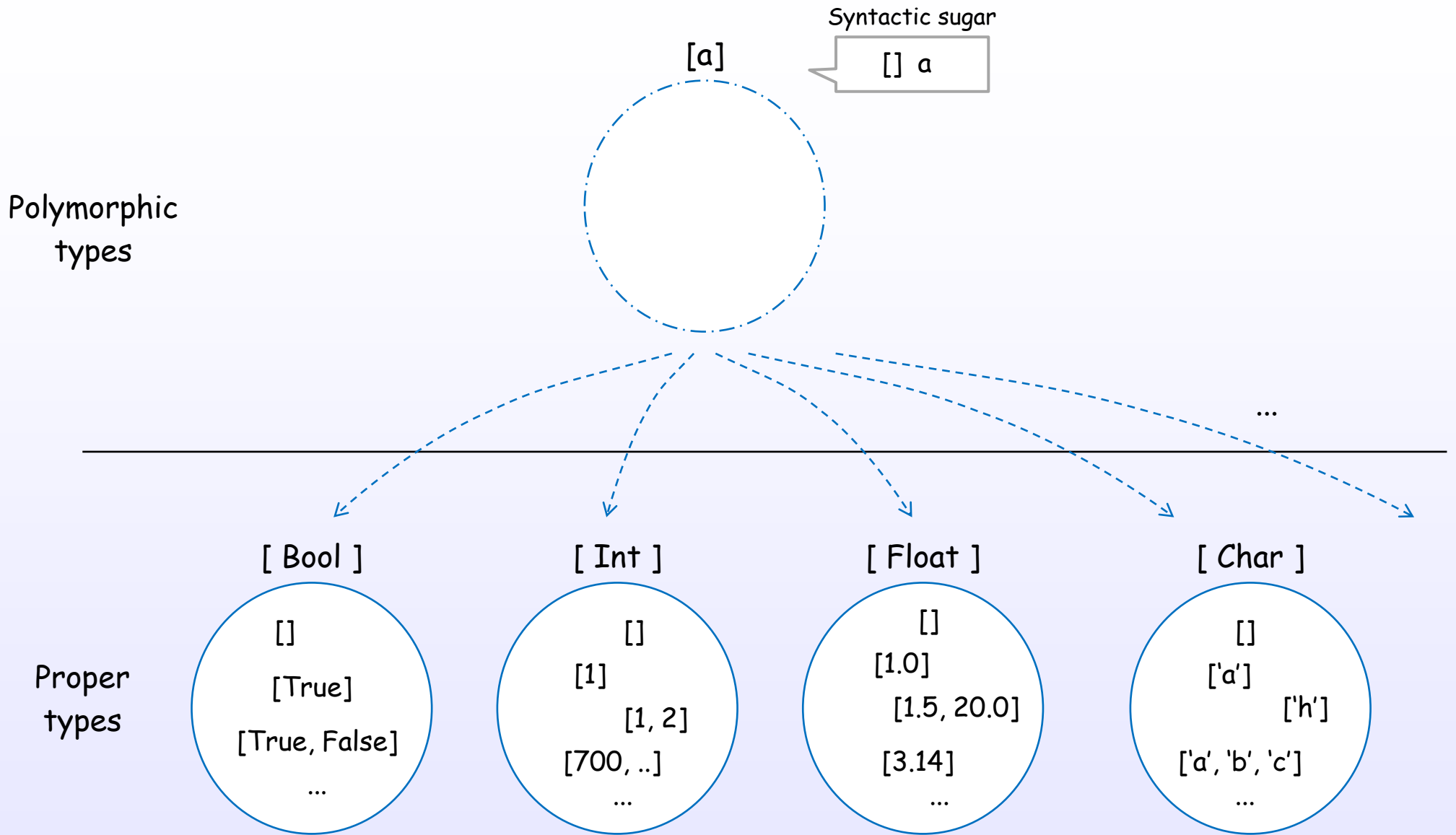
Float



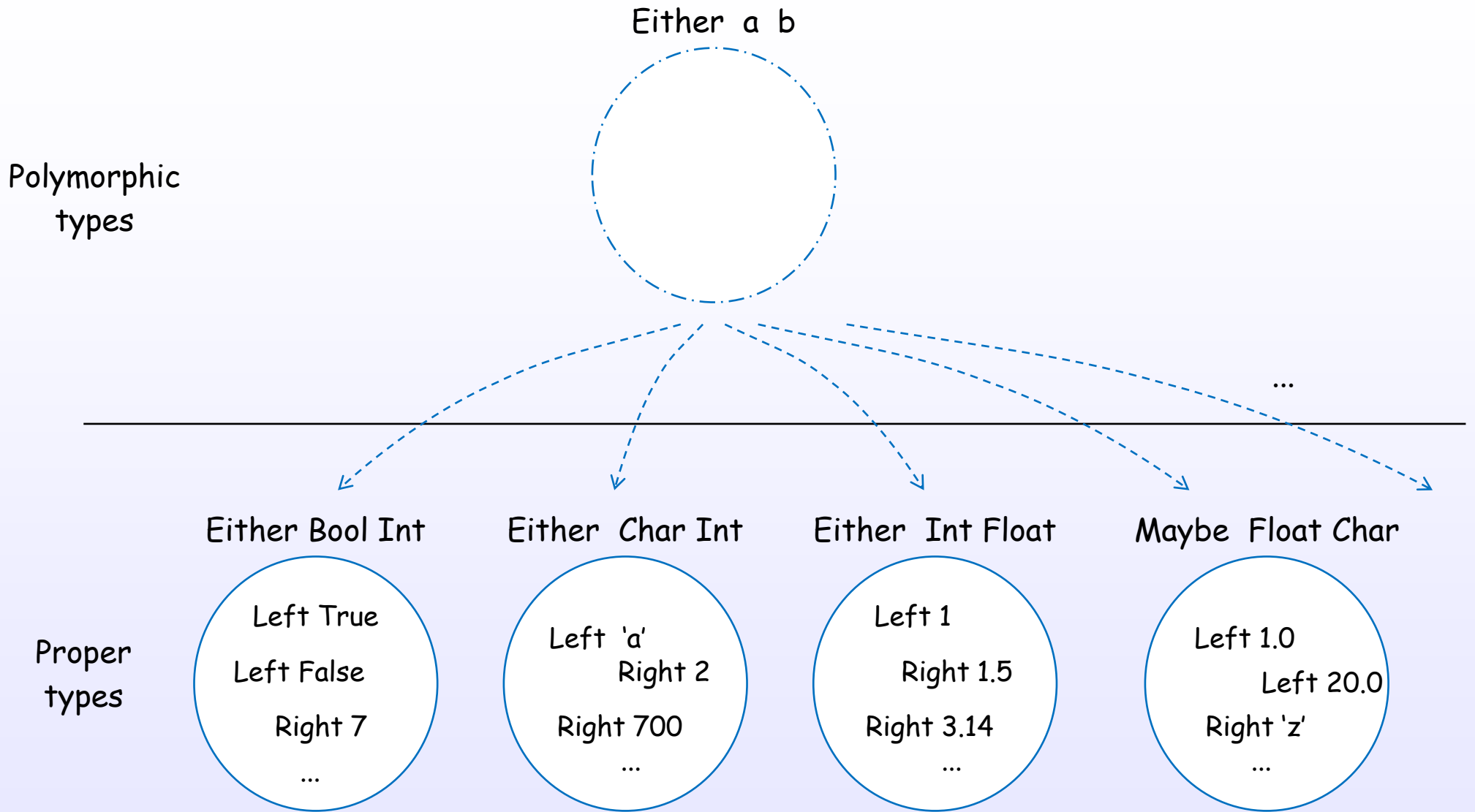
Maybe type



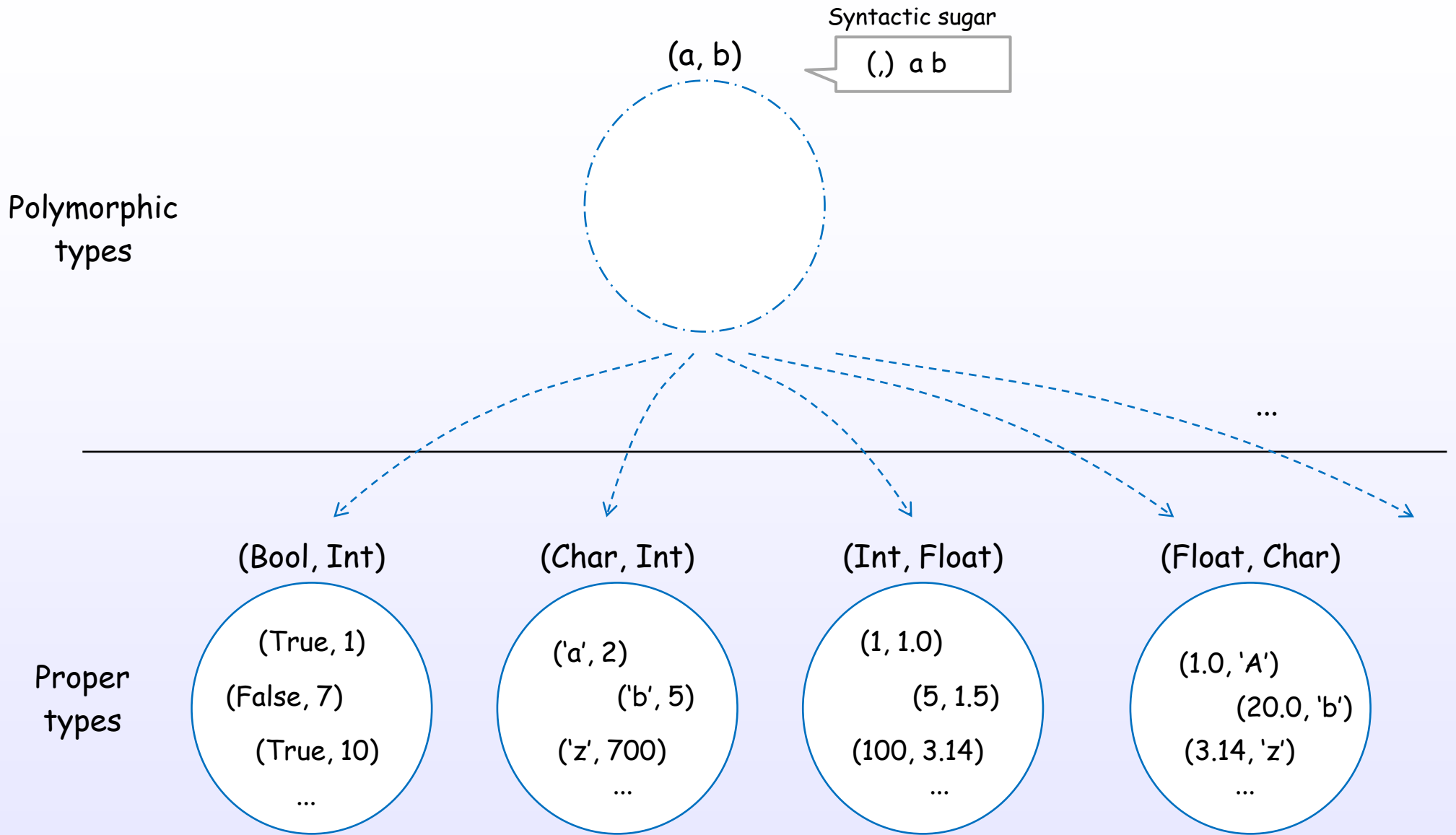
List type



Either type

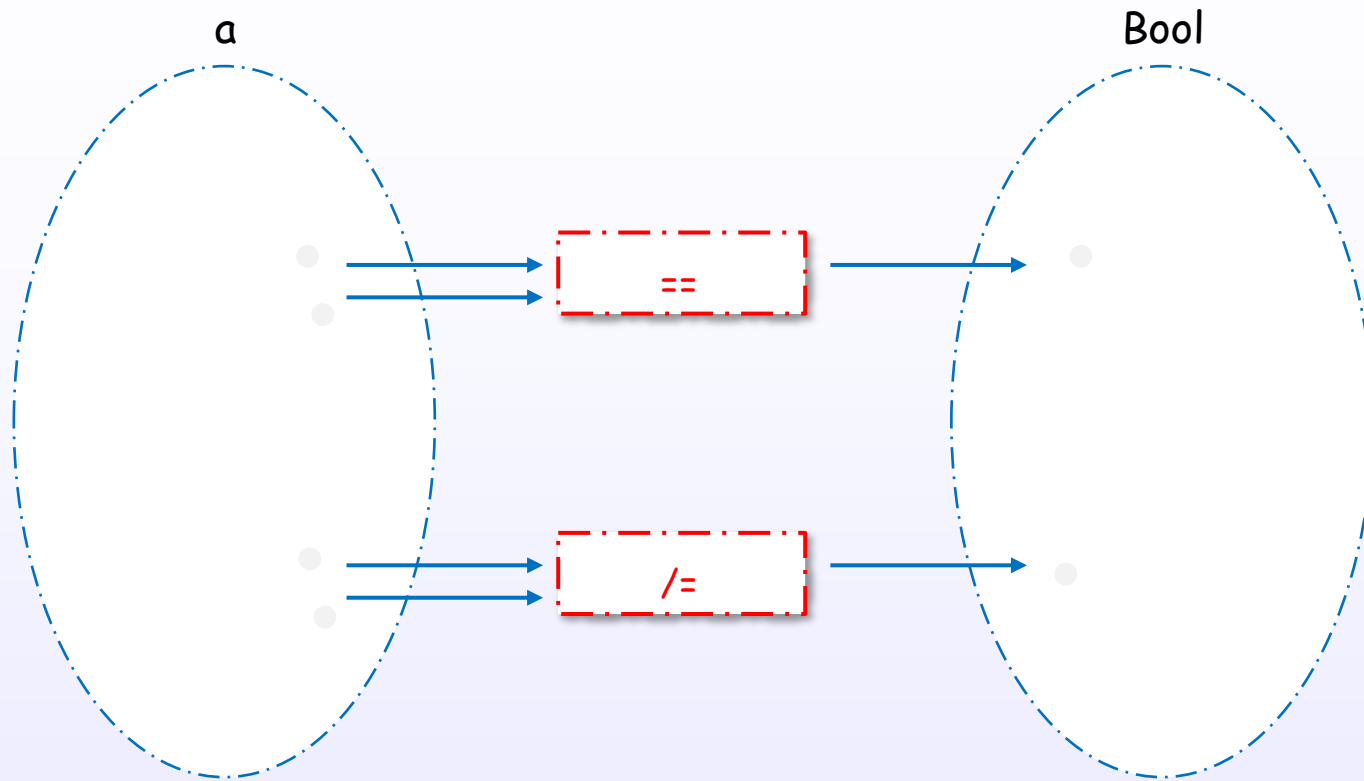


Tuple (pair) type



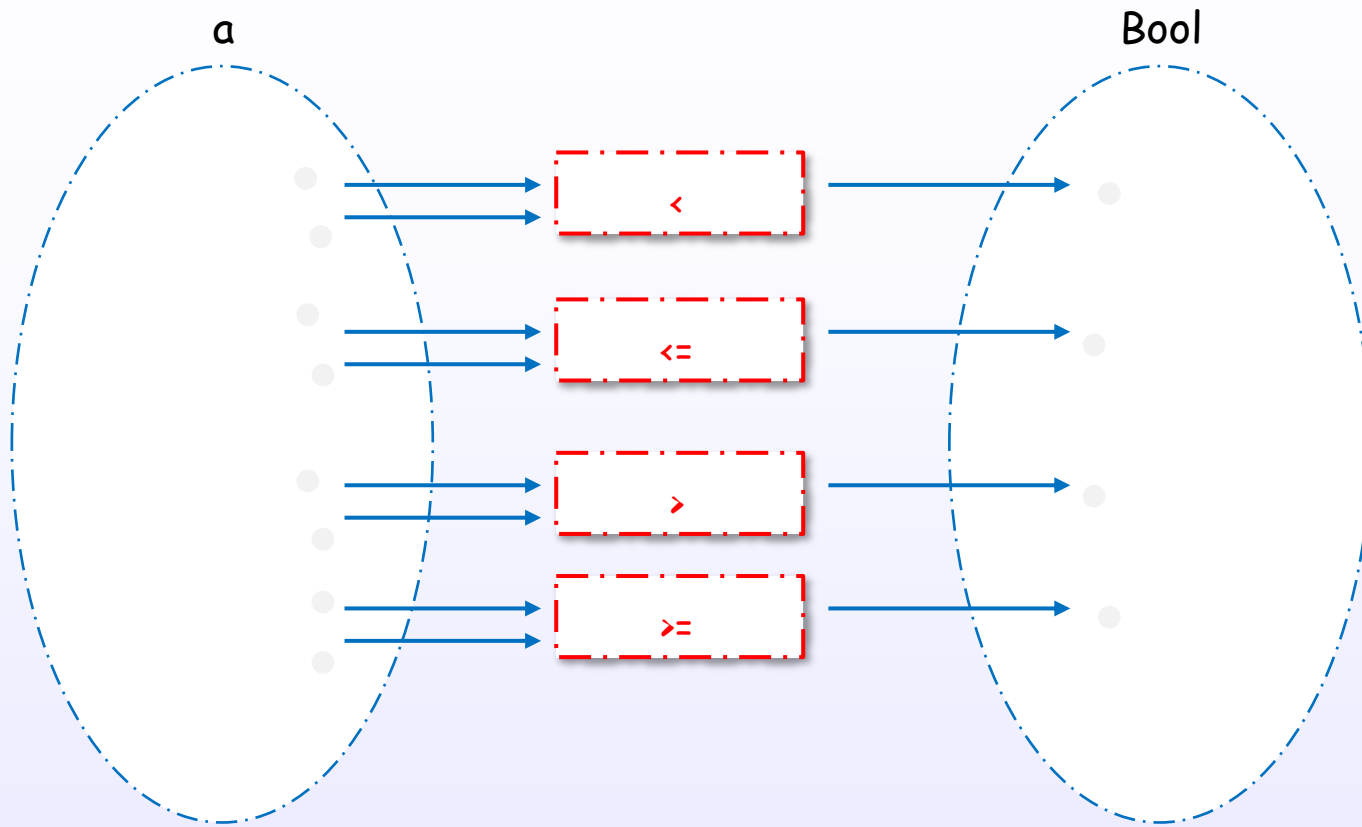
Appendix II - Various type classes

Eq class's characteristic operations



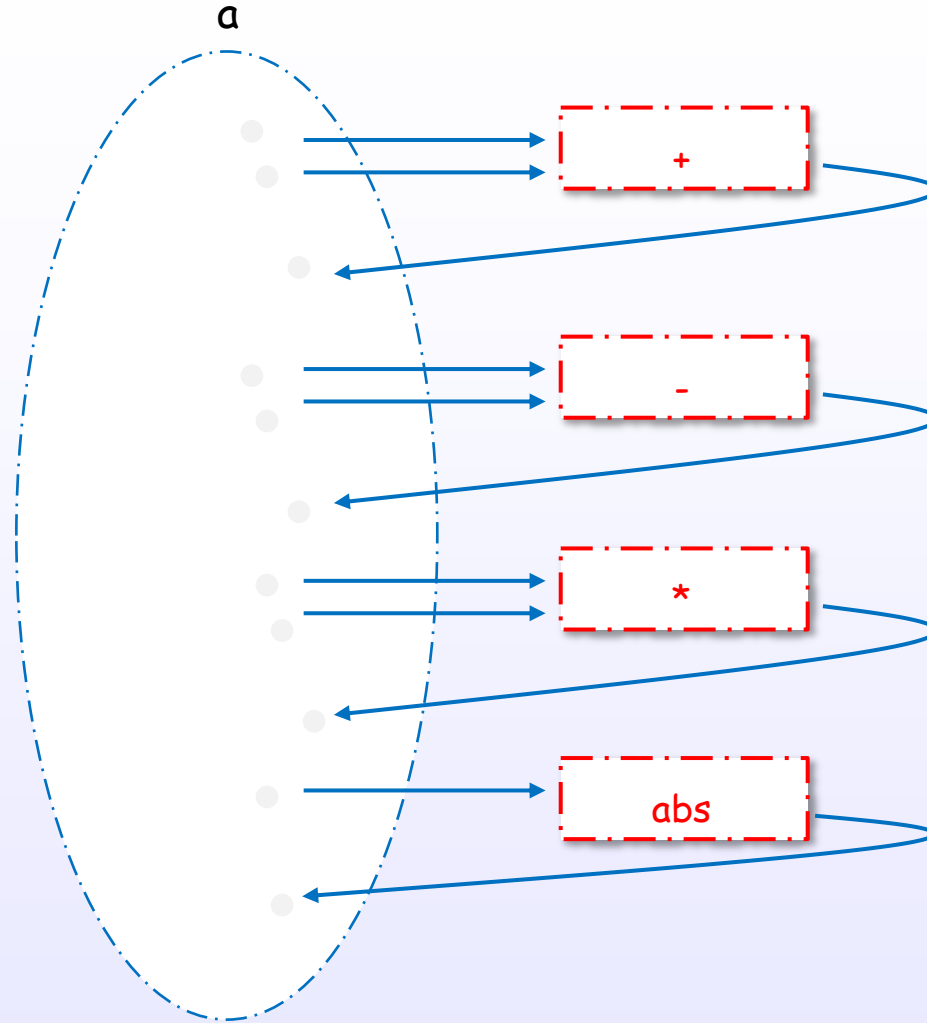
The Eq class has equality operations.

Ord class's characteristic operations



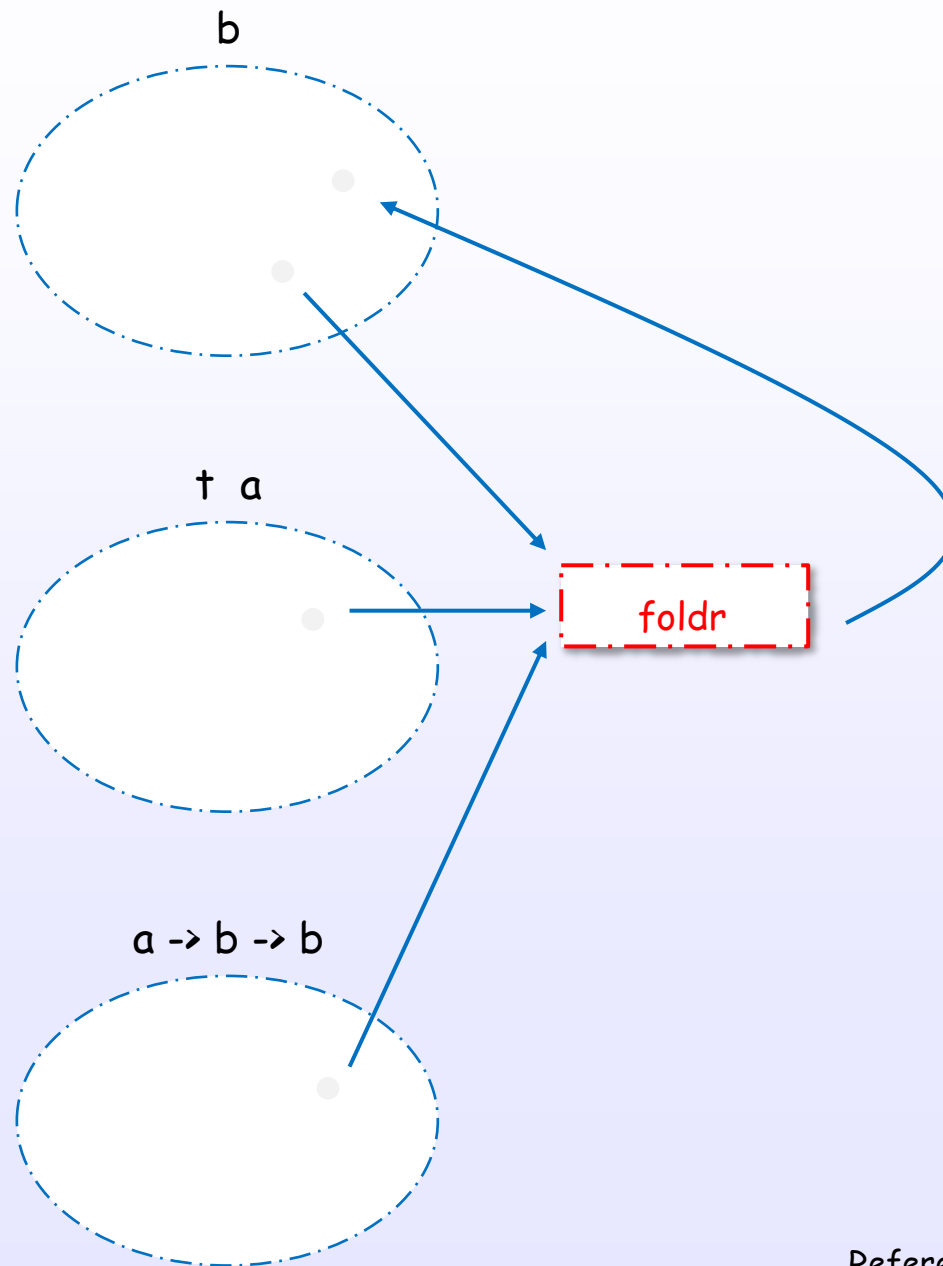
The Ord class has comparability operations.

Num class's characteristic operations

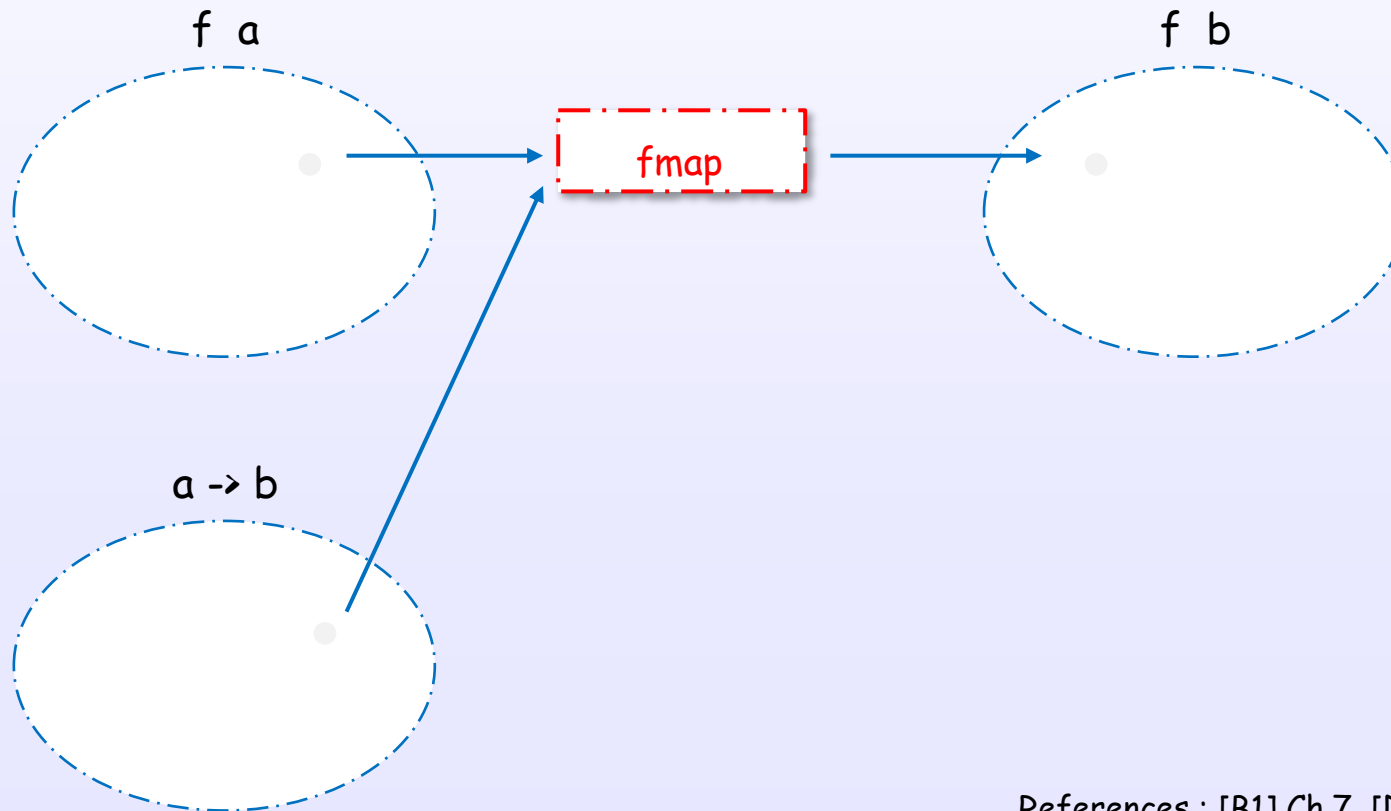


The Num class has arithmetic operations.

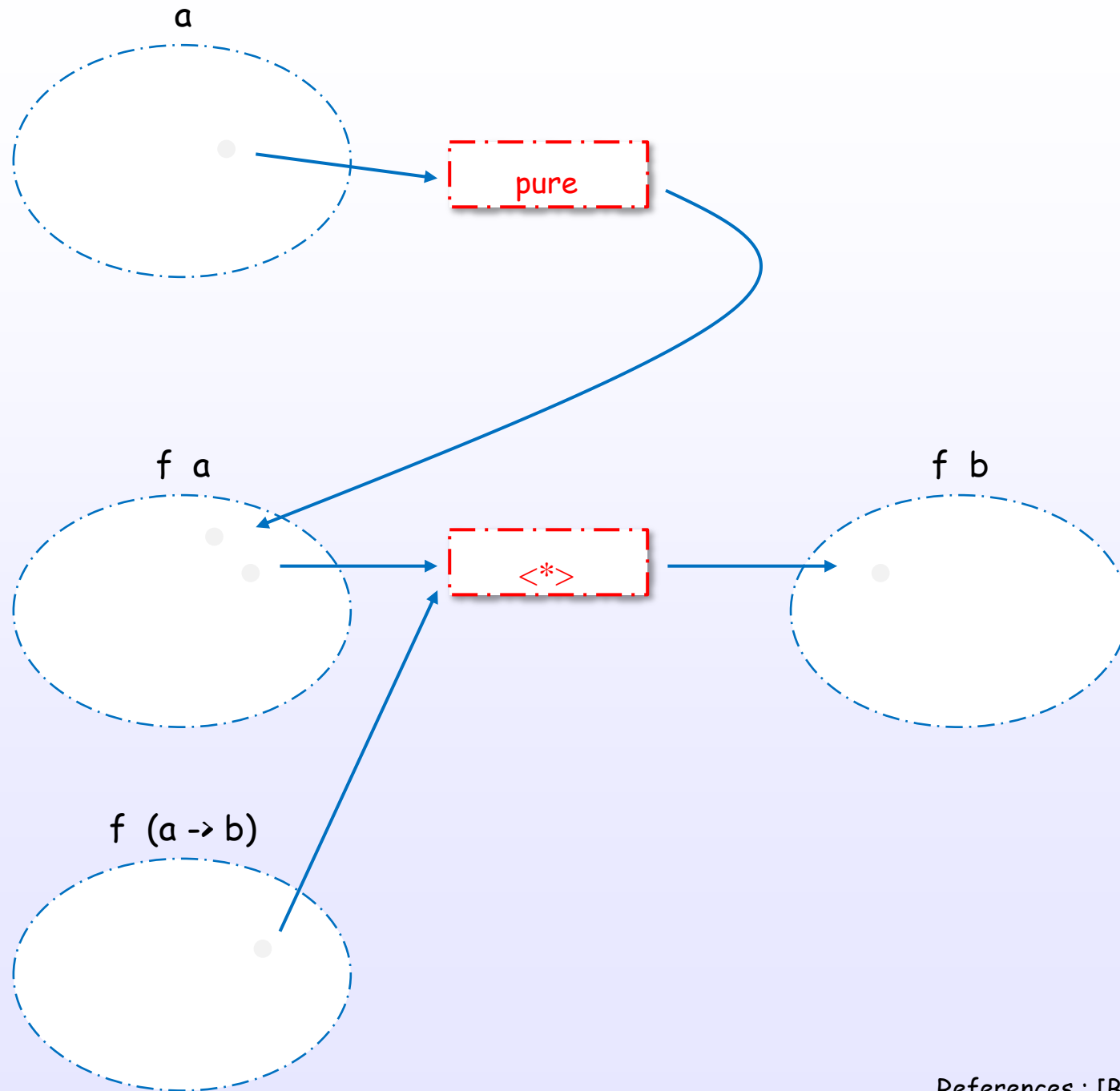
Foldable class's characteristic operations



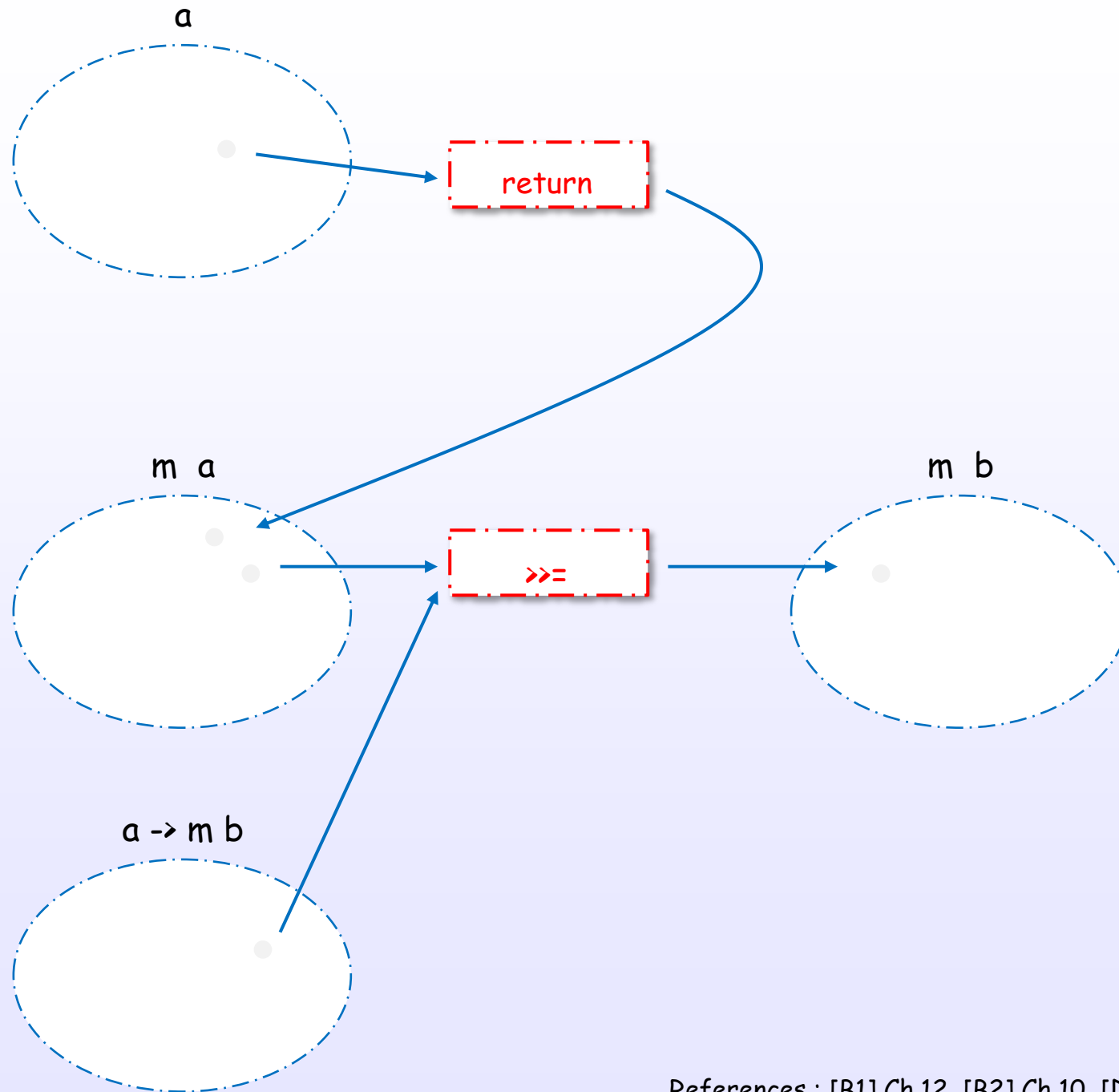
Functor class's characteristic operations



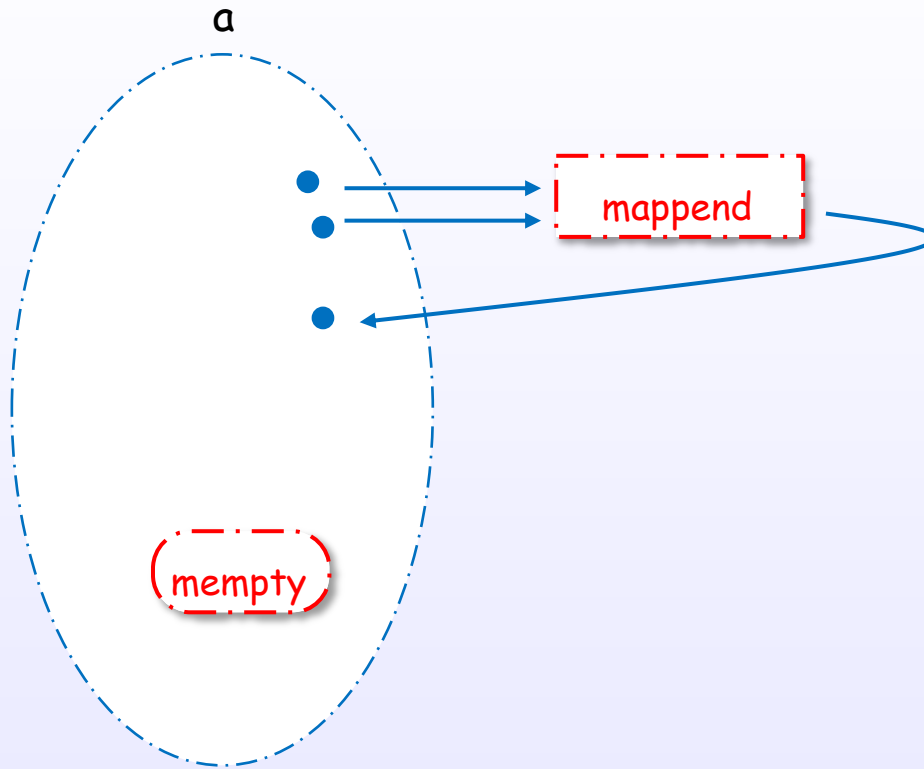
Applicative class's characteristic operations



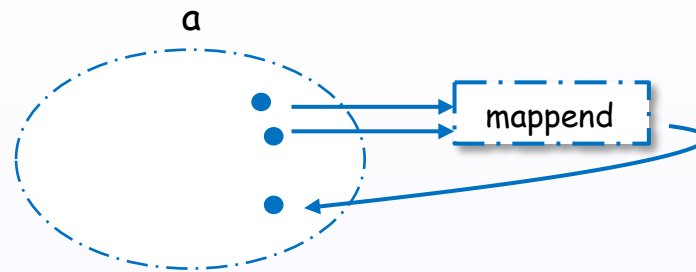
Monad class's characteristic operations



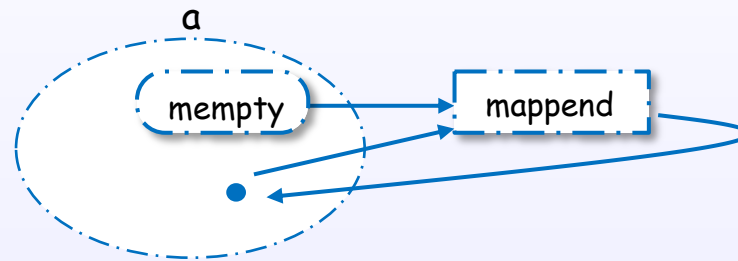
Monoid class's characteristic operations



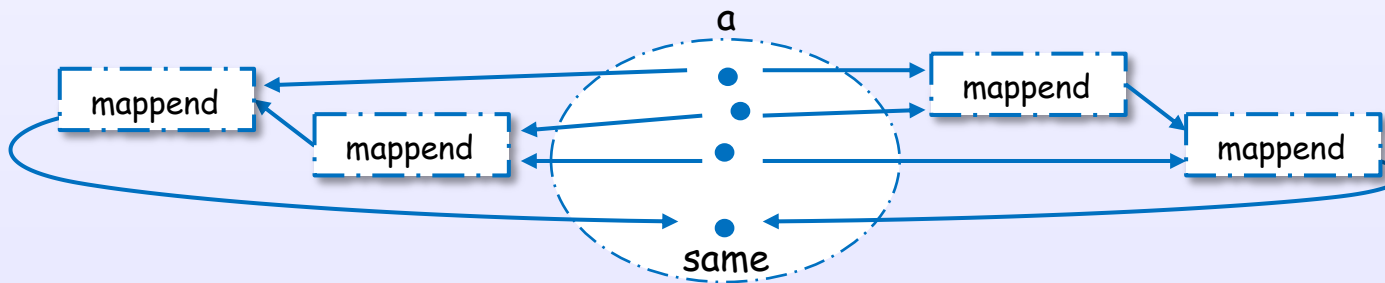
Related topics: monoid laws



binary operation



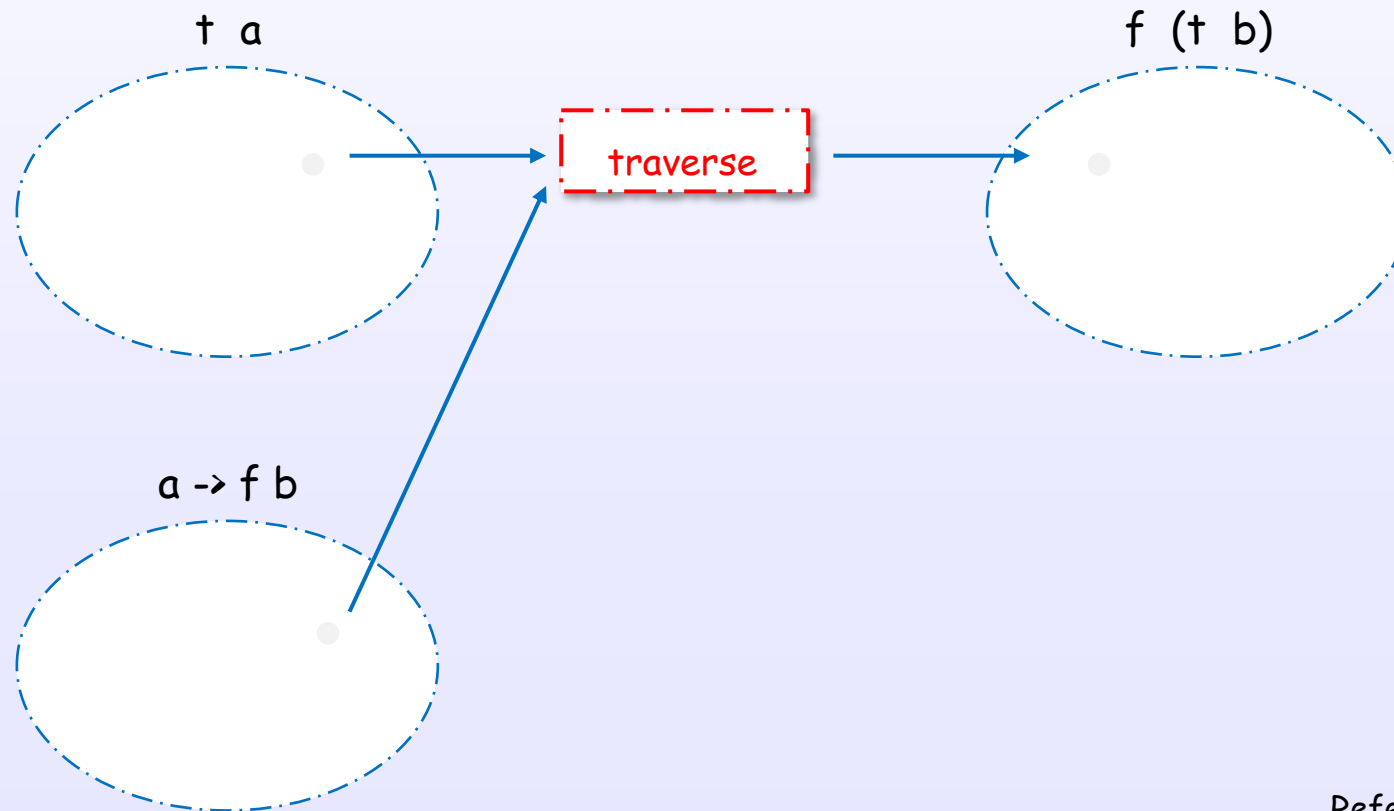
unit law



associative law

Programmers should satisfy the monoid laws.

Traversable class's characteristic operations



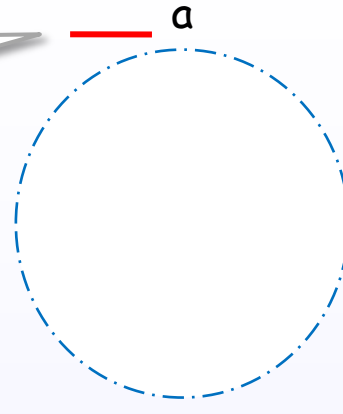
Appendix III - Advanced topics

Universally quantified types

forall a \Rightarrow a

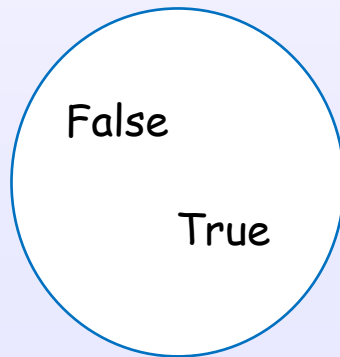
implicitly specified with
universal quantification ($\forall a$)

Polymorphic
types

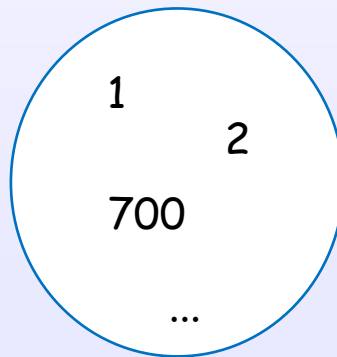


Proper
types

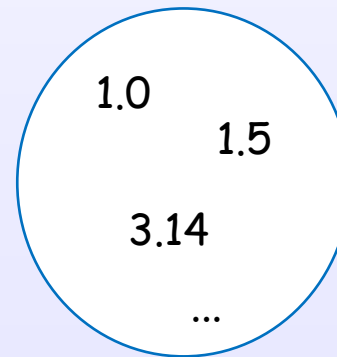
Bool



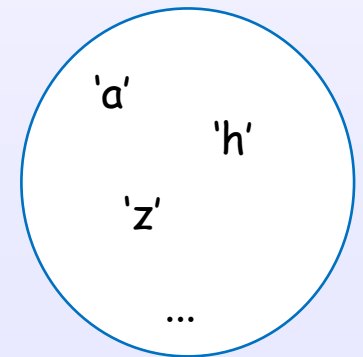
Int



Float

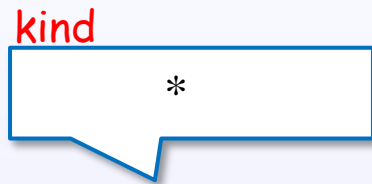


Char



Kind and type constructors

nullary



Int



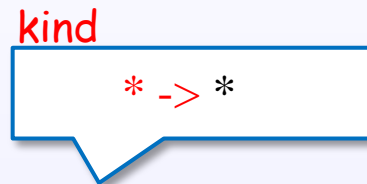
Float



Int -> Bool



unary



Maybe Int



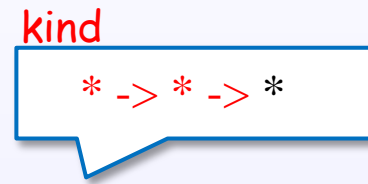
IO Int



[] Int



binary



Either Int Char



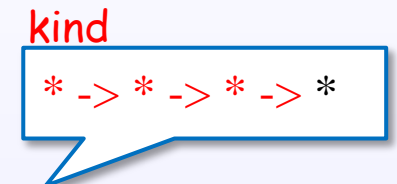
State Int Char



(,) Int Char



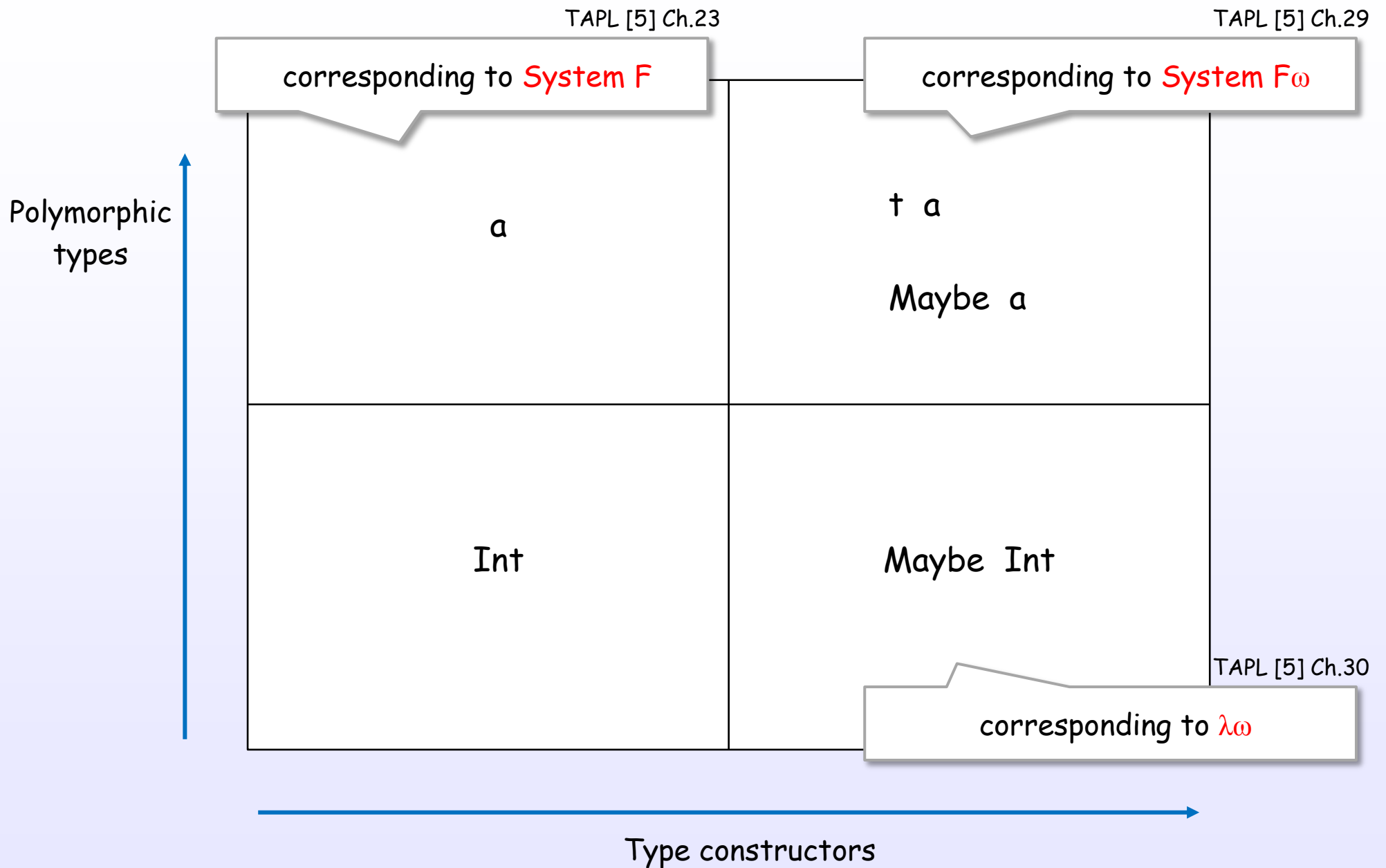
3-ary ...



(,) Int Int Float



Type systems



References

References

Books

- [B1] Learn You a Haskell for Great Good!
<http://learnyouahaskell.com/>

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

- [B3] Programming in Haskell
<https://www.cs.nott.ac.uk/~gmh/book.html>

- [B4] Real World Haskell
<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/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] A Haskell Implementation Reading List
http://www.stephendiehl.com/posts/essential_compilers.htm
- [A4] The GHC reading list
<https://ghc.haskell.org/trac/ghc/wiki/ReadingList>