Generic Deriving of Generic Traversals

Csongor Kiss
Imperial College London
University of Bristol

Matthew Pickering
University of Bristol

Nicolas Wu
University of Bristol

Imperial College London

University of BRISTOL

International Conference on Functional Programming 2018
St. Louis, MO

Fighting boilerplate

- Template Haskell (2002)
- Scrap Your Boilerplate (2003)
- GHC.Generics (2011)

Fighting boilerplate

	High-level	**Efficient**
• Template Haskell (2002)		✓
• Scrap Your Boilerplate (2003)		
• GHC.Generics (2011)		

Fighting boilerplate

	High-level	Efficient
• Template Haskell (2002)		✓
• Scrap Your Boilerplate (2003)	✓	
• GHC.Generics (2011)		

Fighting boilerplate

	High-level	Efficient
• Template Haskell (2002)		✓
• Scrap Your Boilerplate (2003)	✓	
• GHC.Generics (2011)		✓

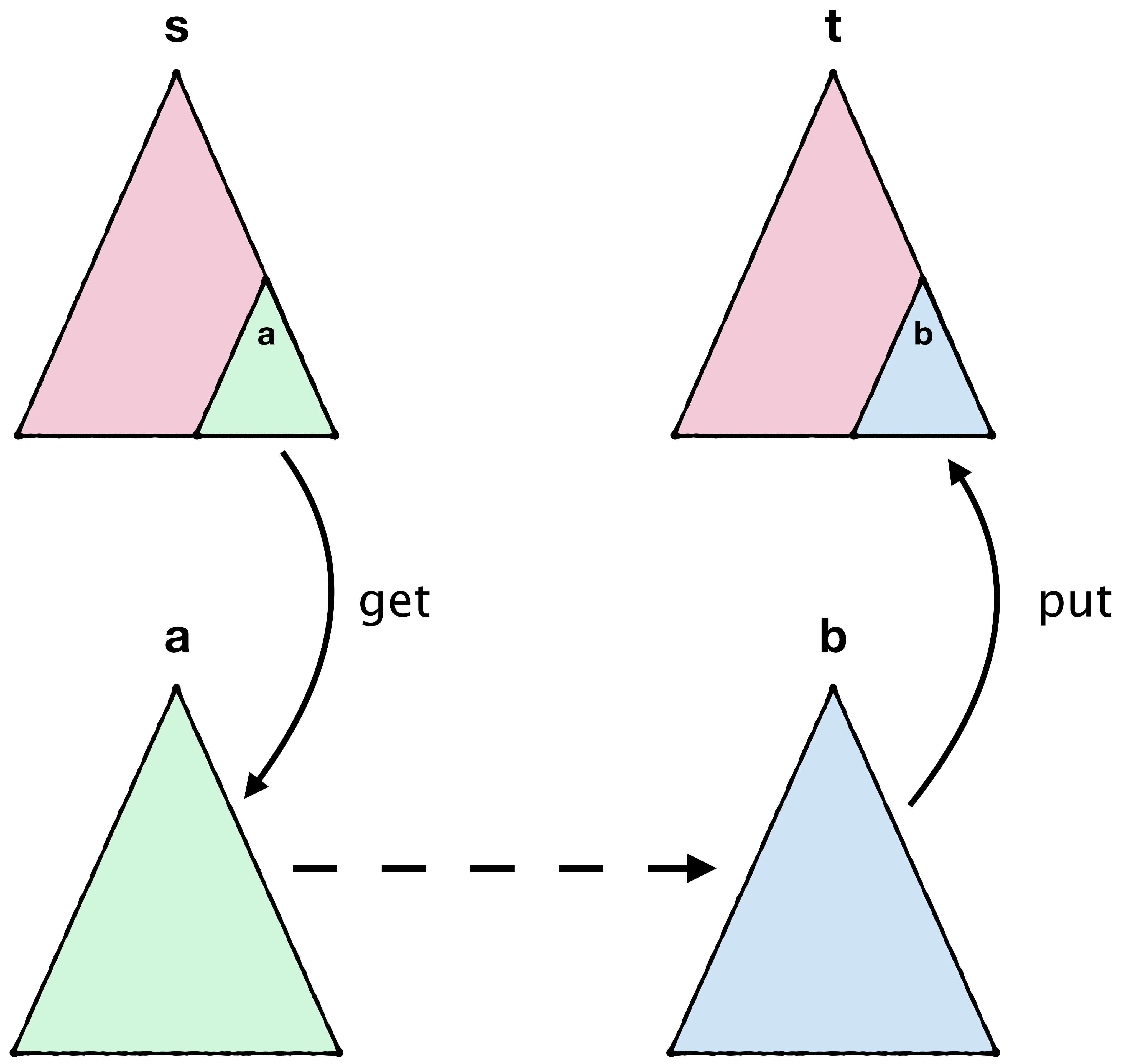
Fighting boilerplate

	High-level	Efficient
• Template Haskell (2002)		✓
• Scrap Your Boilerplate (2003)	✓	
• GHC.Generics (2011)		✓
• generic-lens (2018)	✓	✓

Lenses

Lenses

```
type Lens s t a b
```

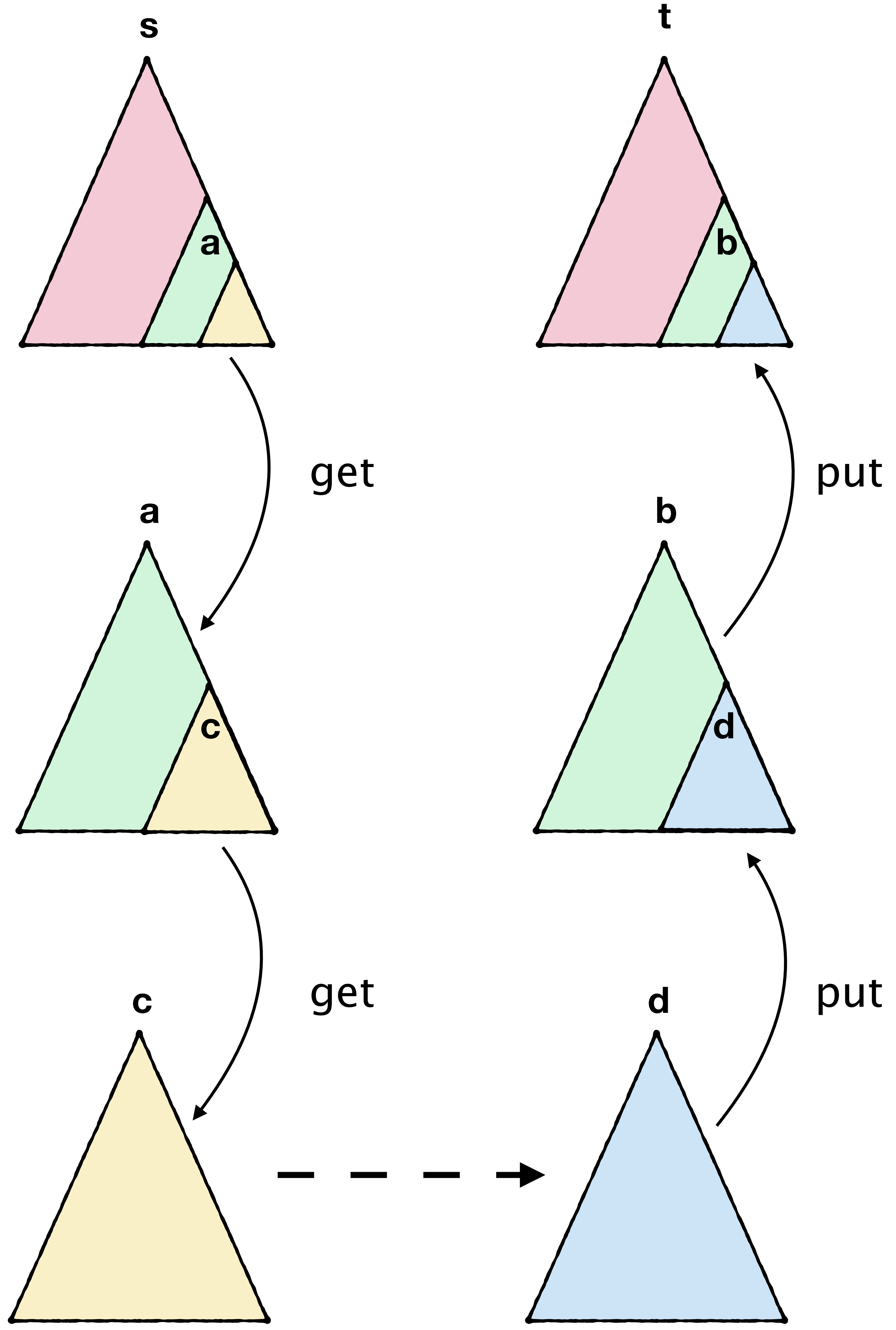


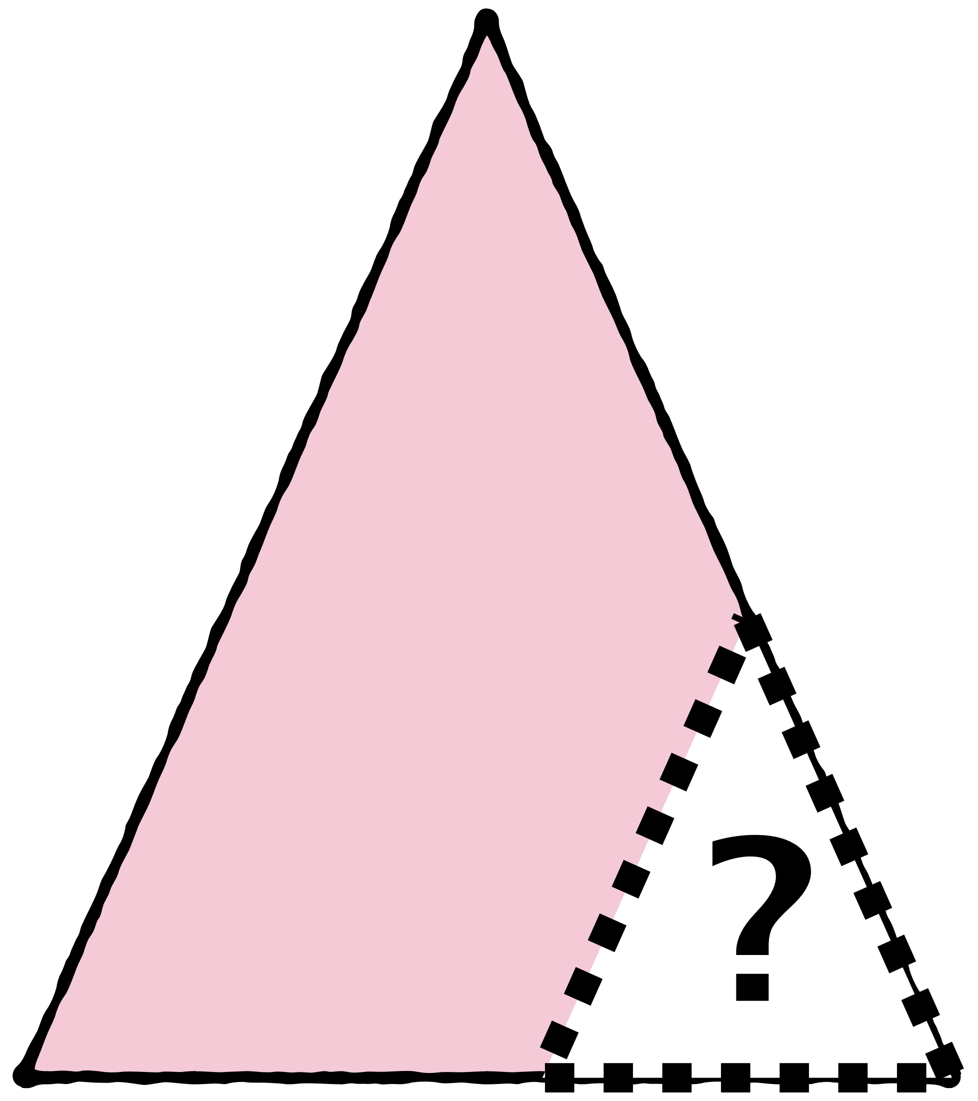
Composition

```
type Lens s t c d
```

```
Lens s t
     a b
```

```
Lens a b
     c d
```





```haskell
data Person = Person
 { name   :: String
 , age    :: Int
 , height :: Int
 }
```

```haskell
data Person = Person
  { name   :: String
  , age    :: Int
  , height :: Int
  }

  view (typed @String)
```

```haskell
data Person = Person
  { name   :: String
  , age    :: Int
  , height :: Int
  }

   view (field @"age")
```

```haskell
data Person = Person
 { name   :: String
 , age    :: Int
 , height :: Int
 }
```

```haskell
view (position @3)
```

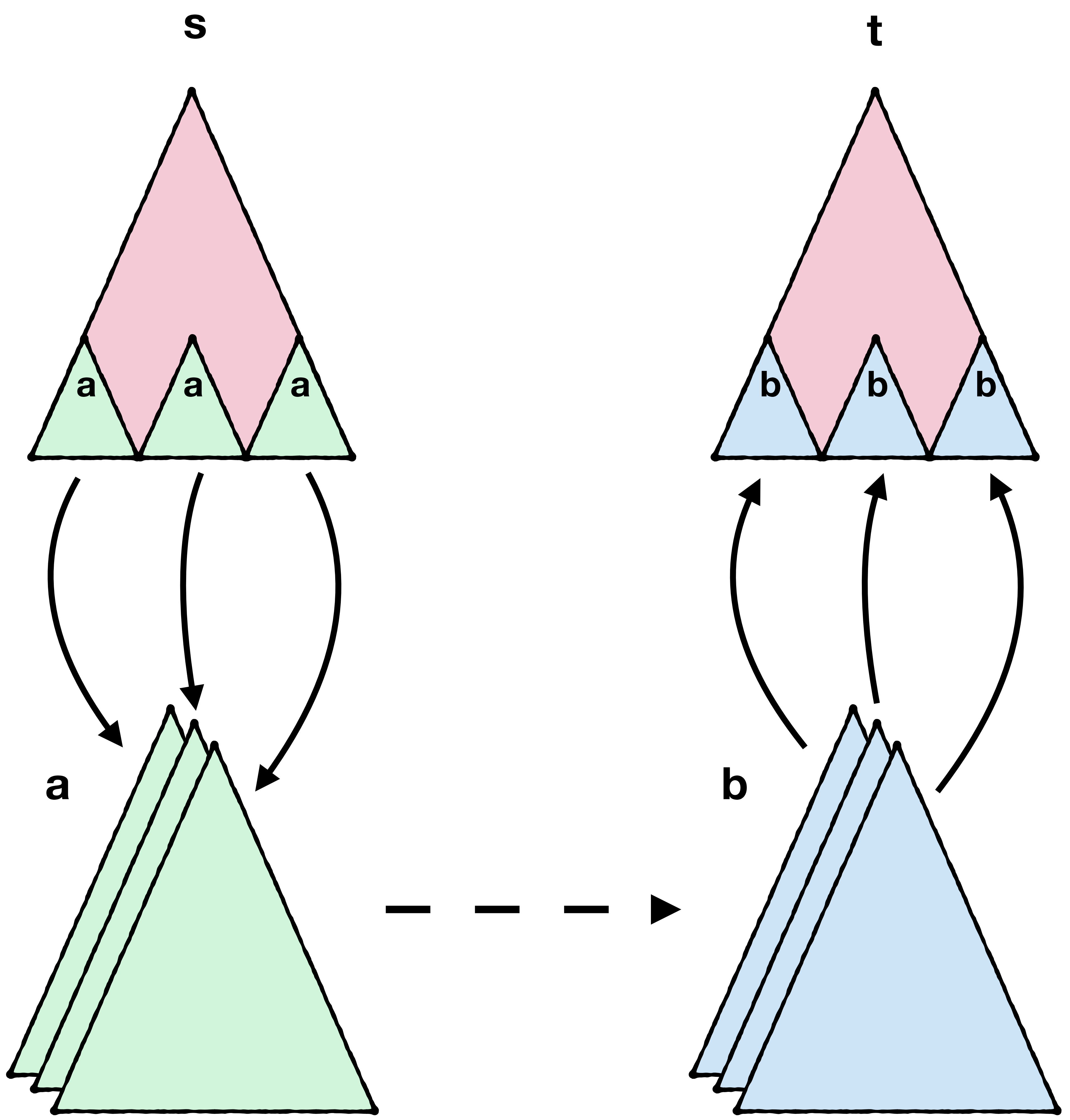
```haskell
data Person = Person
  { name   :: String
  , age    :: Int
  , height :: Int
  }
```

```haskell
view (typed @Int)
```

Traversals

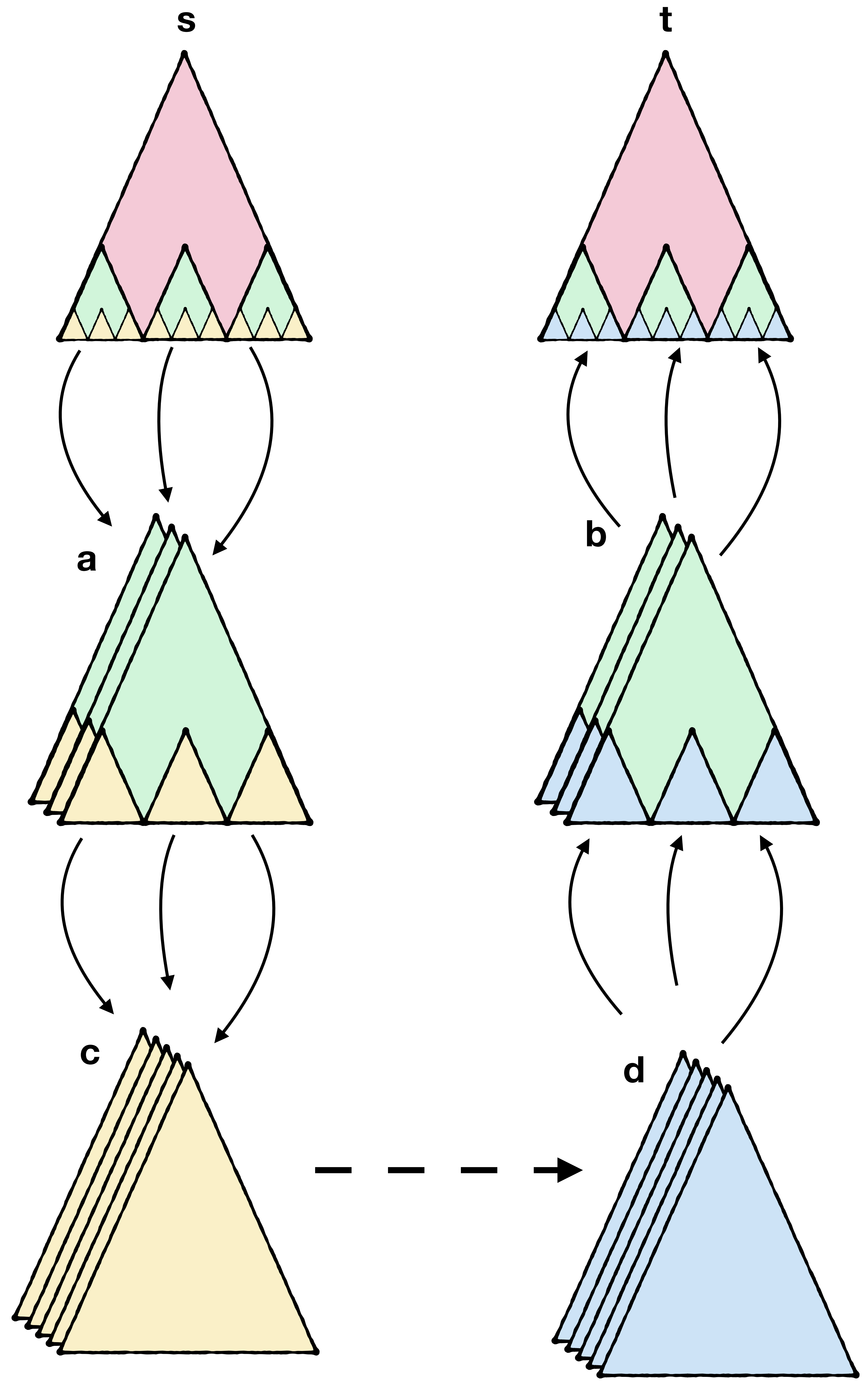
Traversals

```
type Traversal s t a b
```



Composition

```
type Traversal s t c d
```



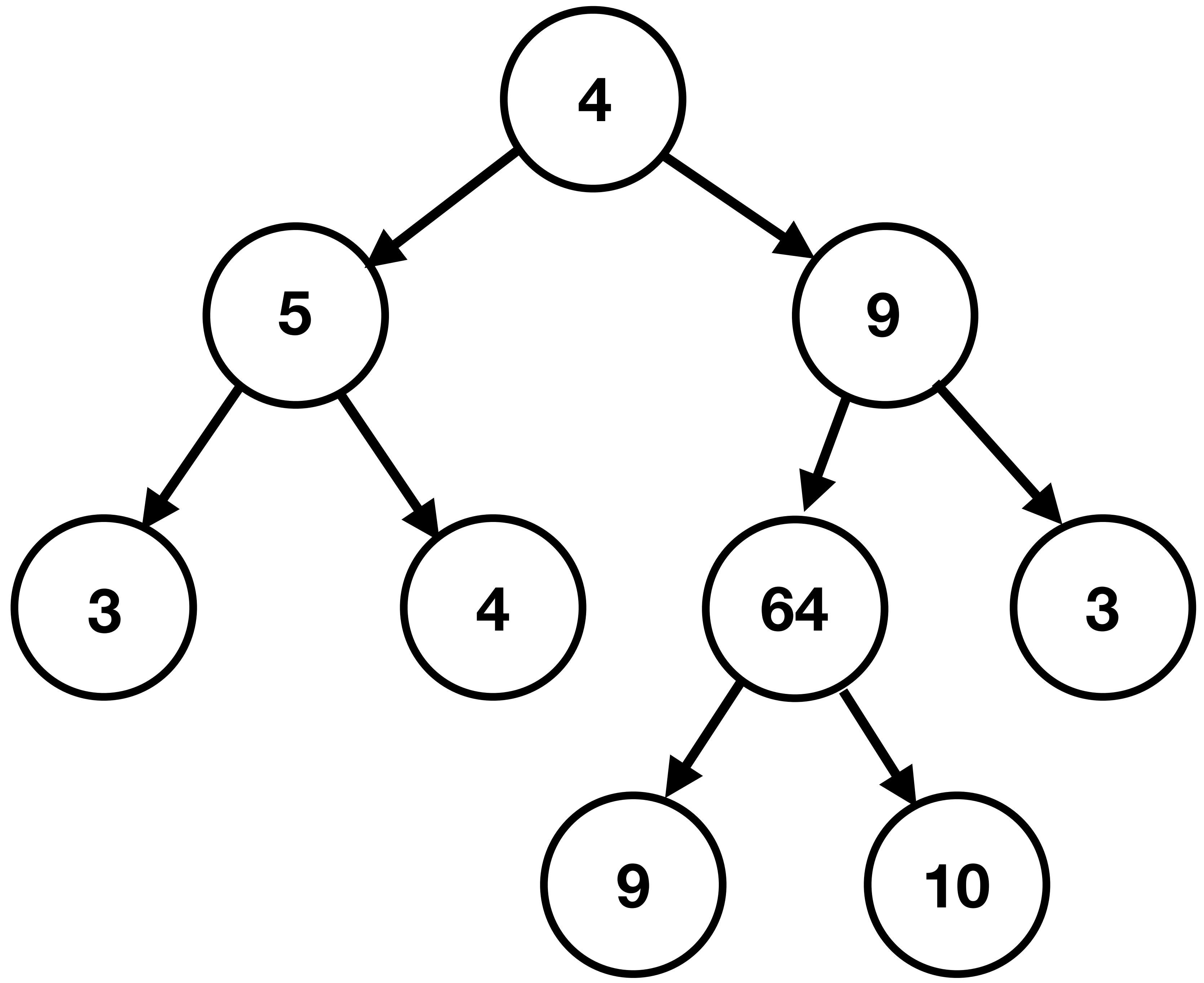
```
Traversal s t
          a b
```

```
Traversal a b
          c d
```

Example

Type-directed traversal

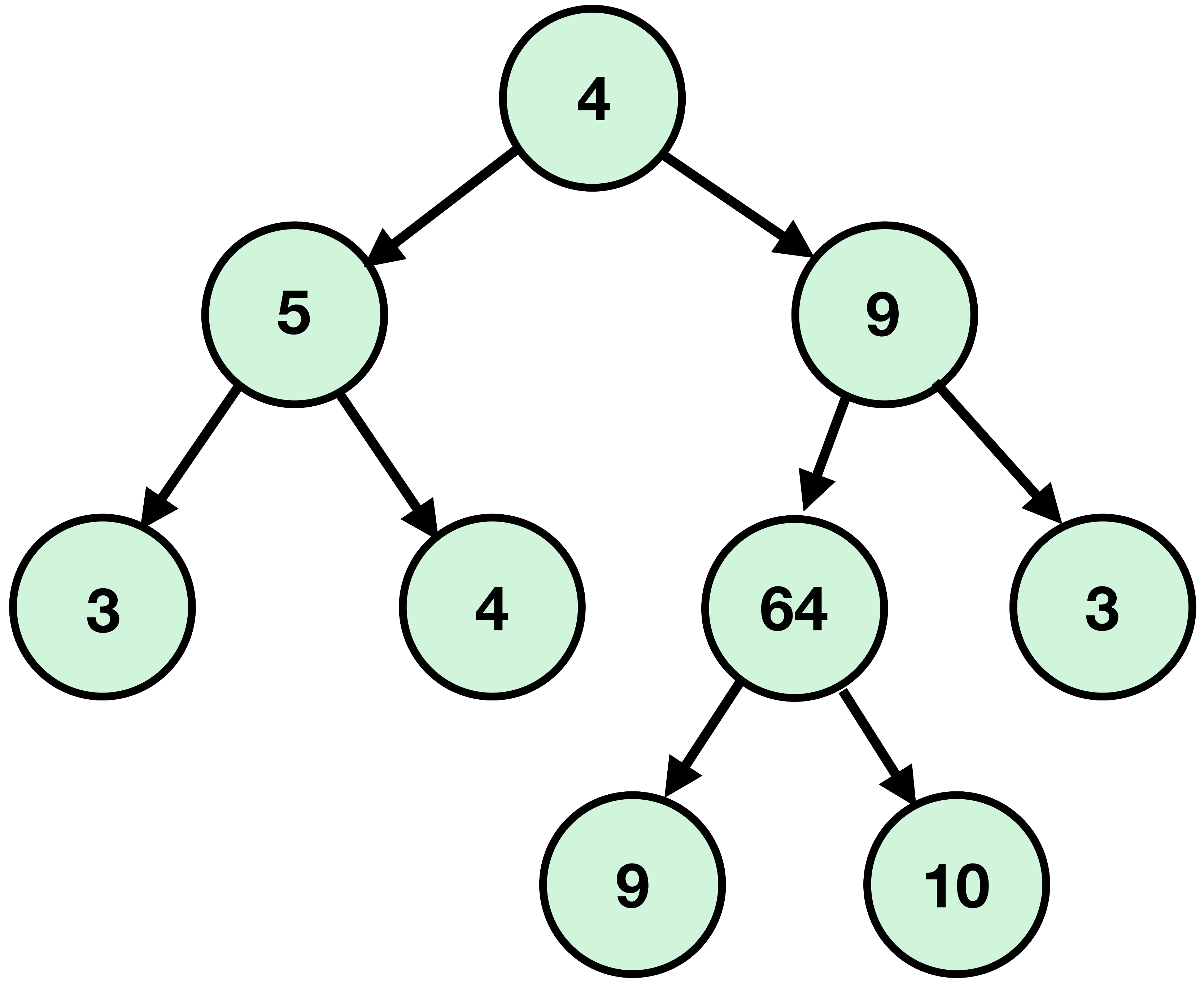
```
data Tree a w
  = Leaf a
  | Node w (Tree a w) (Tree a w)
```



`Tree Int Int`

Type-directed traversal

```
data Tree a w
  = Leaf a
  | Node w (Tree a w) (Tree a w)
```



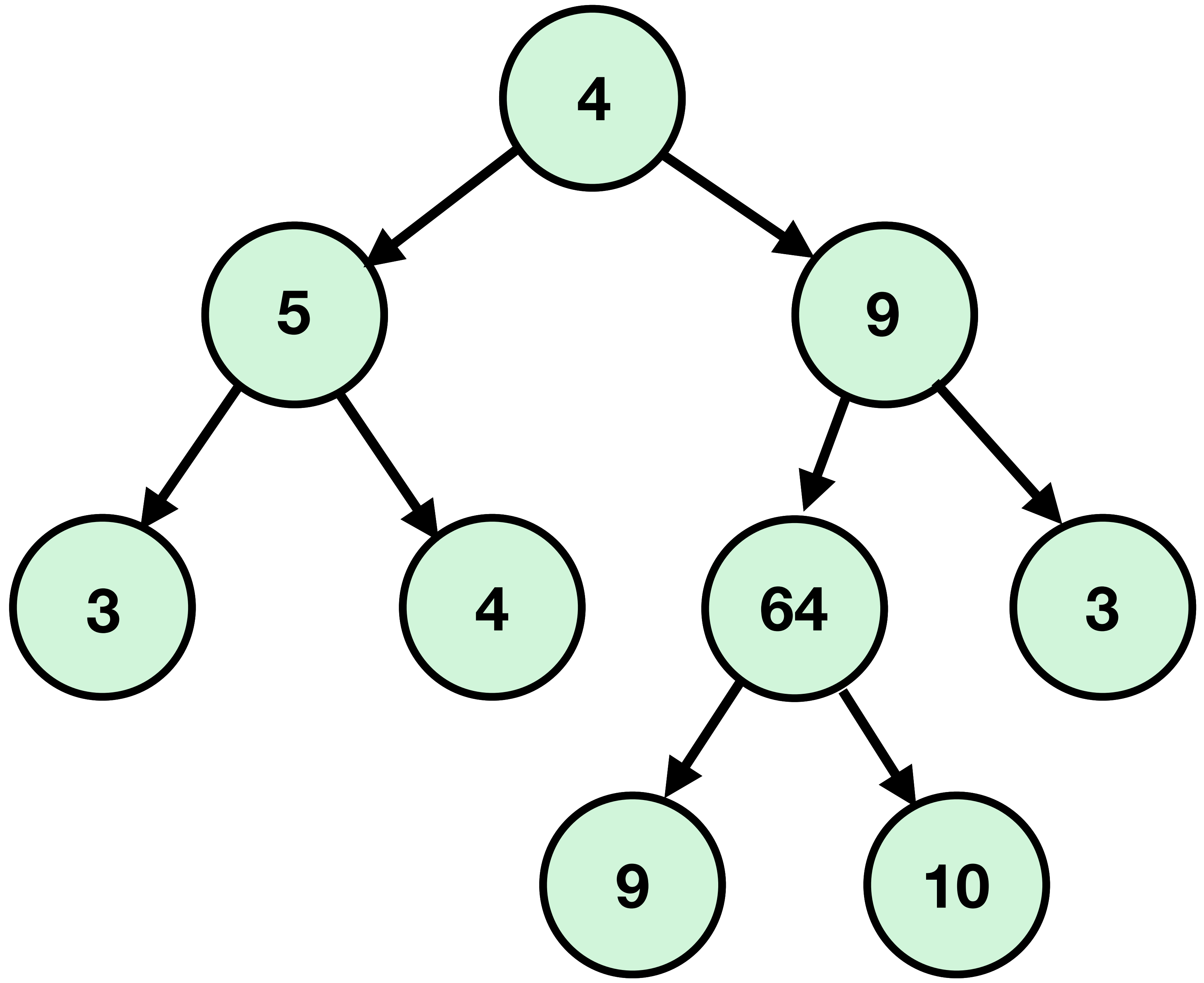
```
types @Int
```

```
Tree Int Int
```

Type-directed traversal

```haskell
data Tree a w
  = Leaf a
  | Node w (Tree a w) (Tree a w)
```

```
[4,5,3,4,9,64,9,10,3]
```

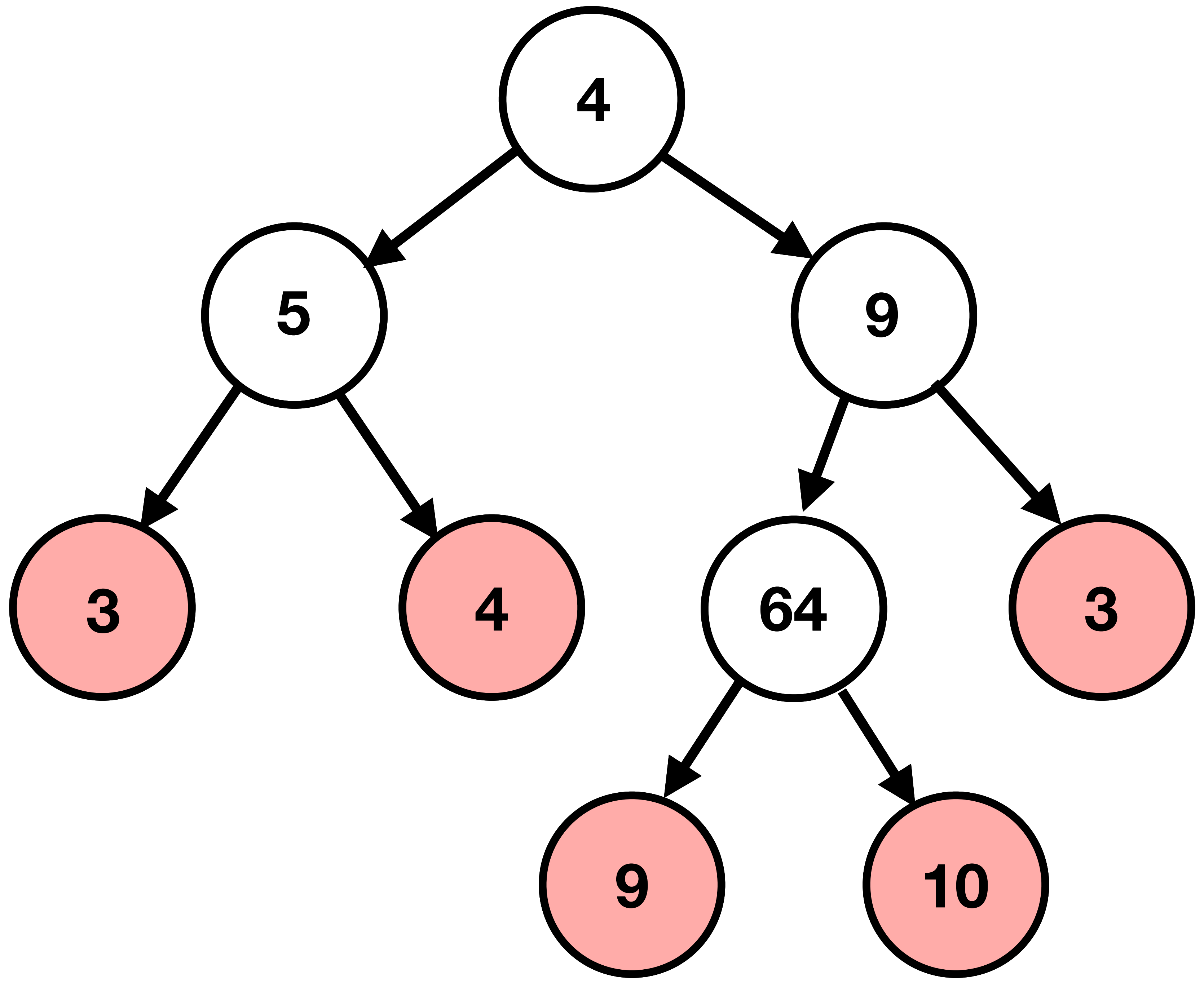


```haskell
toListOf (types @Int )
```

```haskell
Tree Int Int
```

Polymorphic traversals

```
data Tree a w
  = Leaf a
  | Node w (Tree a w) (Tree a w)
```

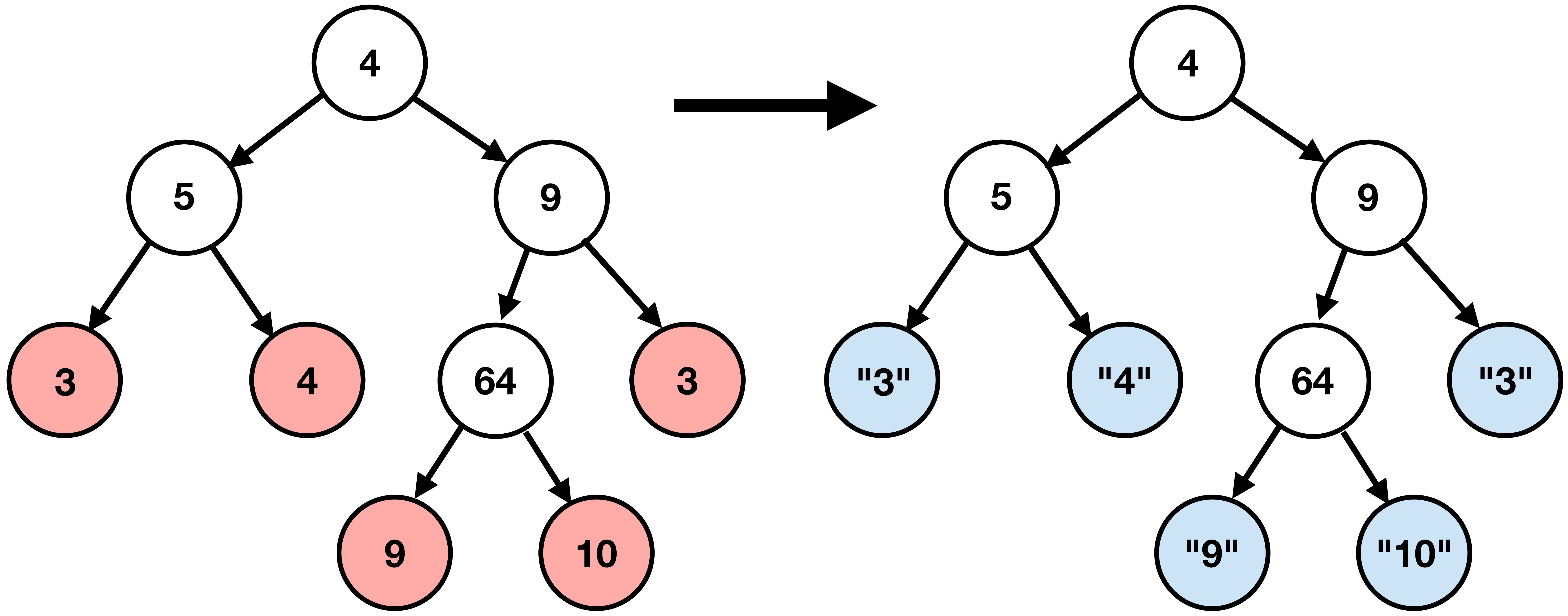


`param @1`

`Tree Int Int`

Polymorphic traversals

```
Traversal (Tree Int Int) (Tree String Int) Int String
```



```
over ( param @1 ) show
```

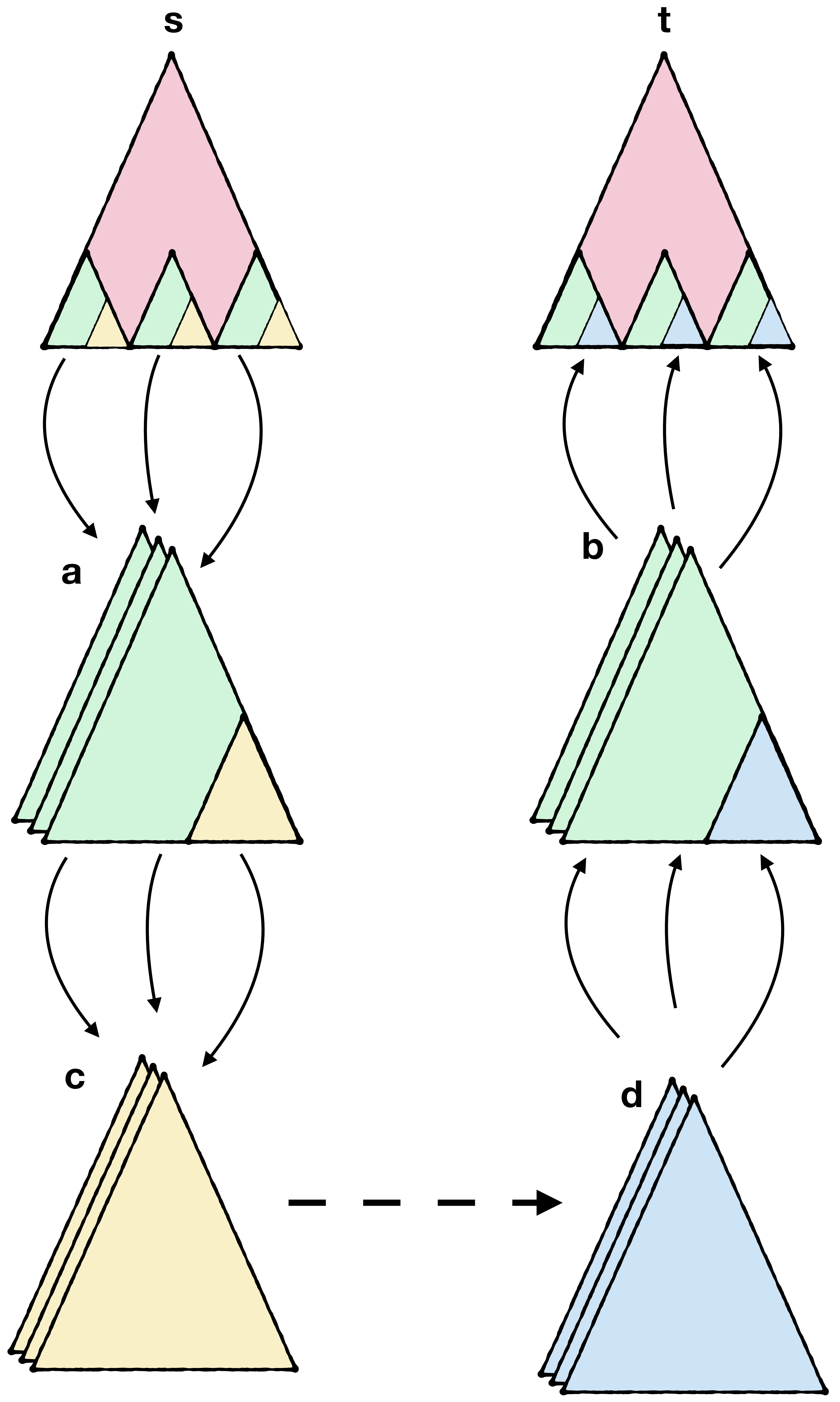
```
Tree String Int
```

Lenses are Traversals

Composition

```
Traversal s t
          a b
```

```
     Lens a b
          c d
```



Example

```haskell
data Class = Class
  { teachers :: [Person]
  , students :: [Person]
  }
```

```haskell
toListOf ( field @"teachers"
         . types @Person
         . field @"name" )
```

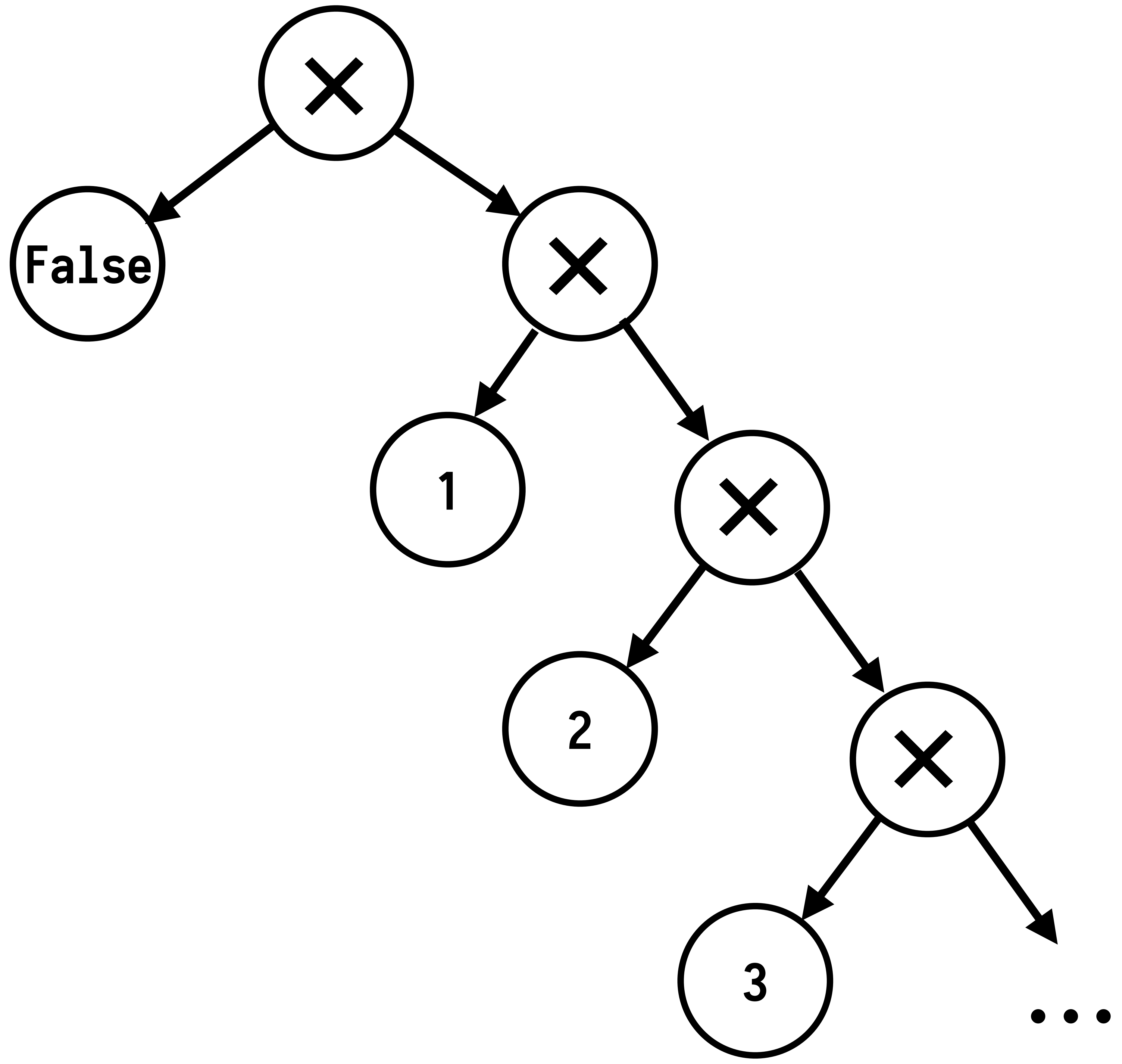
```
Class [john, peter] [gleb, ash] ----► ["John", "Peter"]
```

Locating values

Generic universe

Sums	`data f :+: g = L f | R g`
Products	`data f :×: g = f :×: g`
Constants	`newtype K a = K a`
Unit	`data U = U`
Void	`data V`

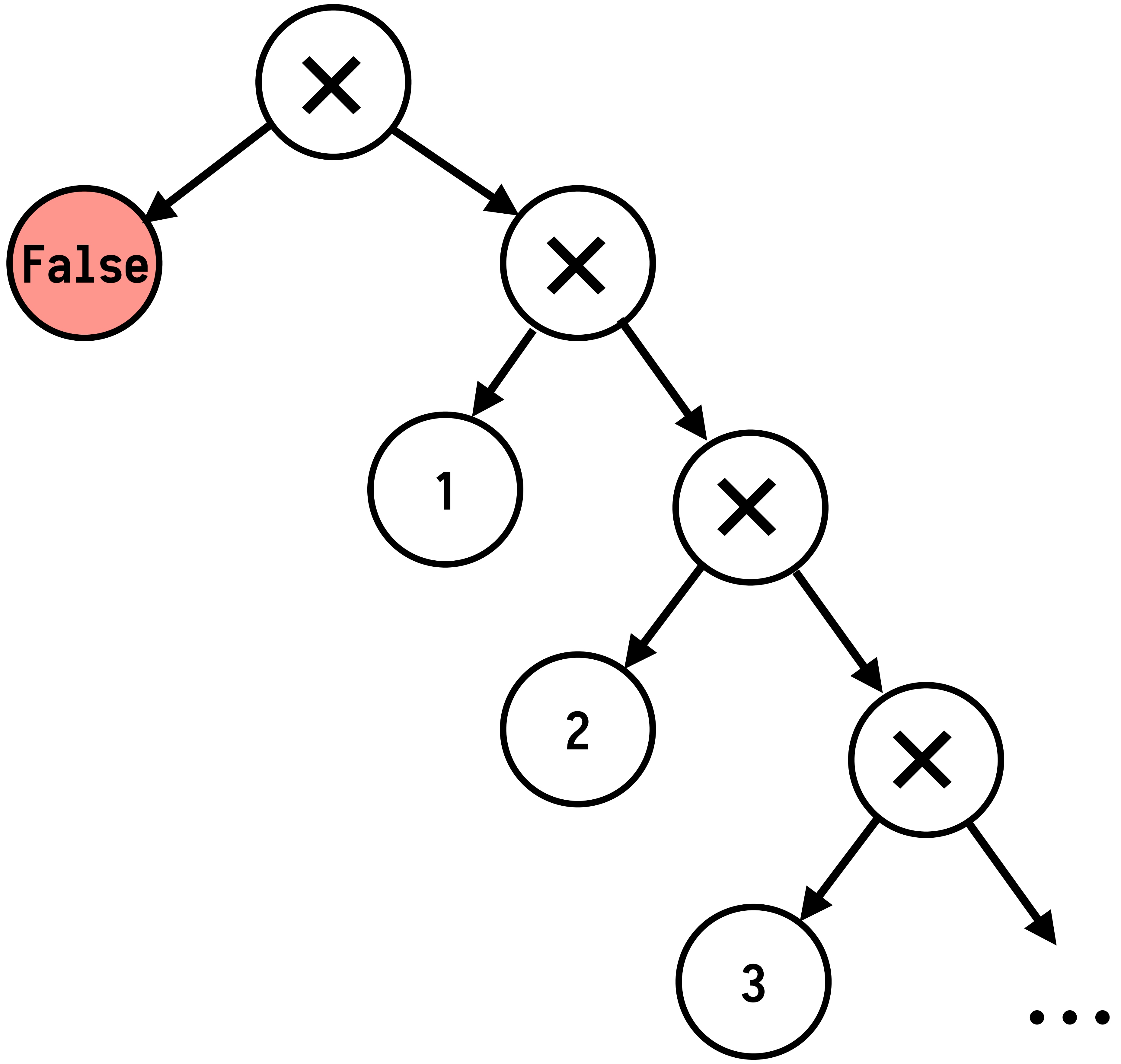
```
(False, [1, 2, 3, 4, 5, 6, 7]) :: (Bool, [Int])
```



Value structure

```
(False, [1, 2, 3, 4, 5, 6, 7]) :: (Bool, [Int])
```

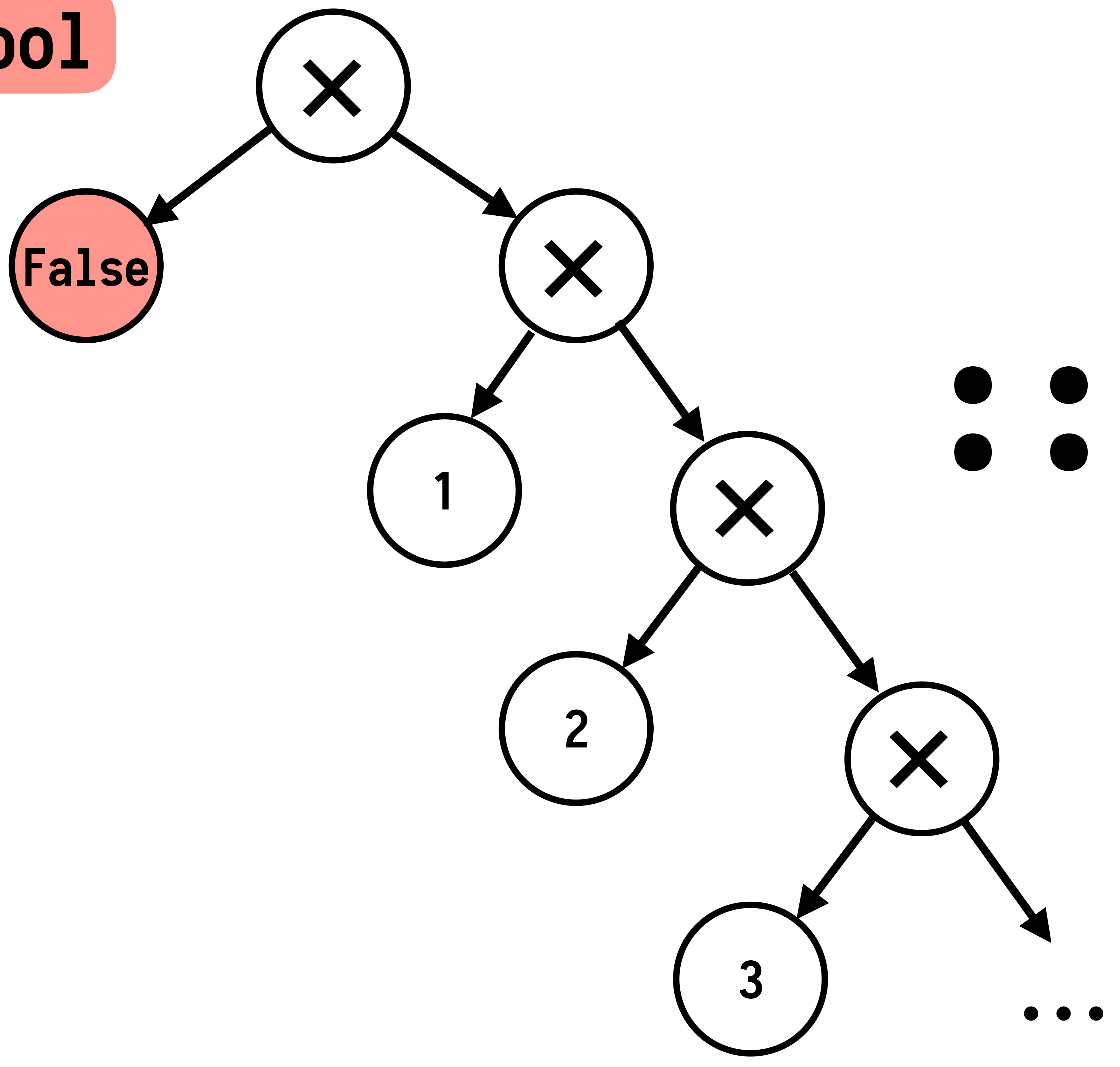
```
types @Bool
```



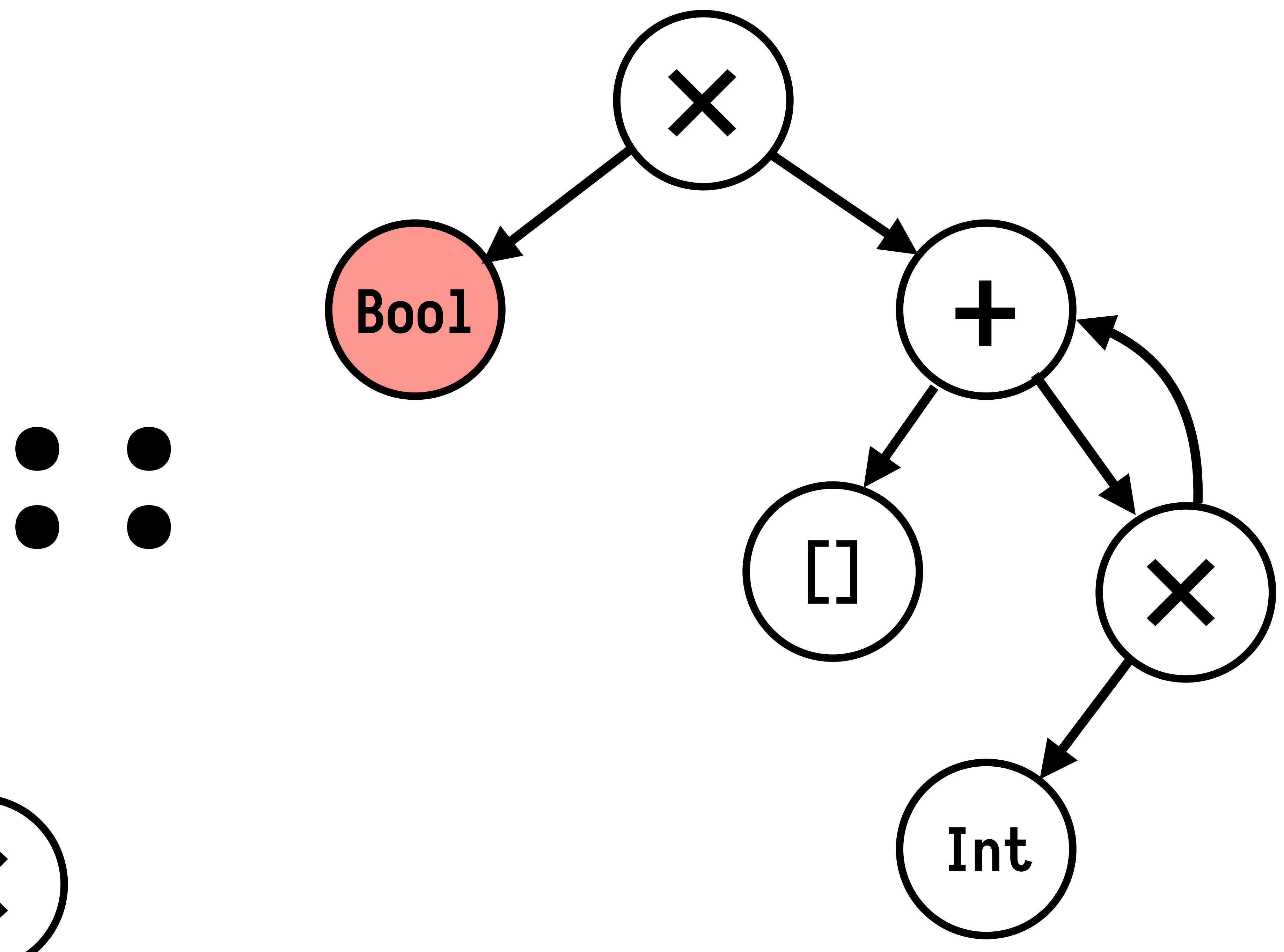
Value structure

`(False, [1, 2, 3, 4, 5, 6, 7]) :: (Bool, [Int])`

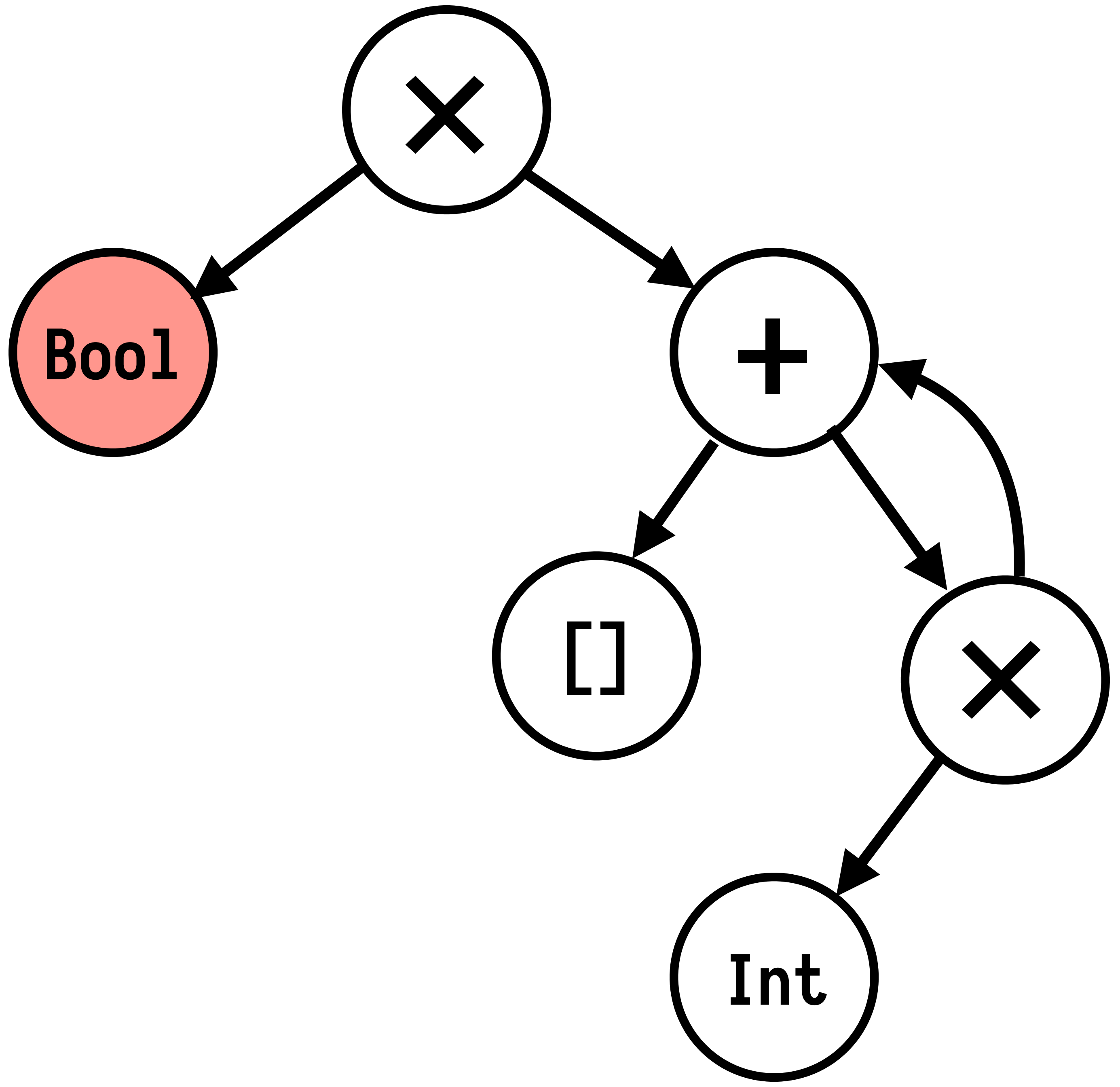
`types @Bool`



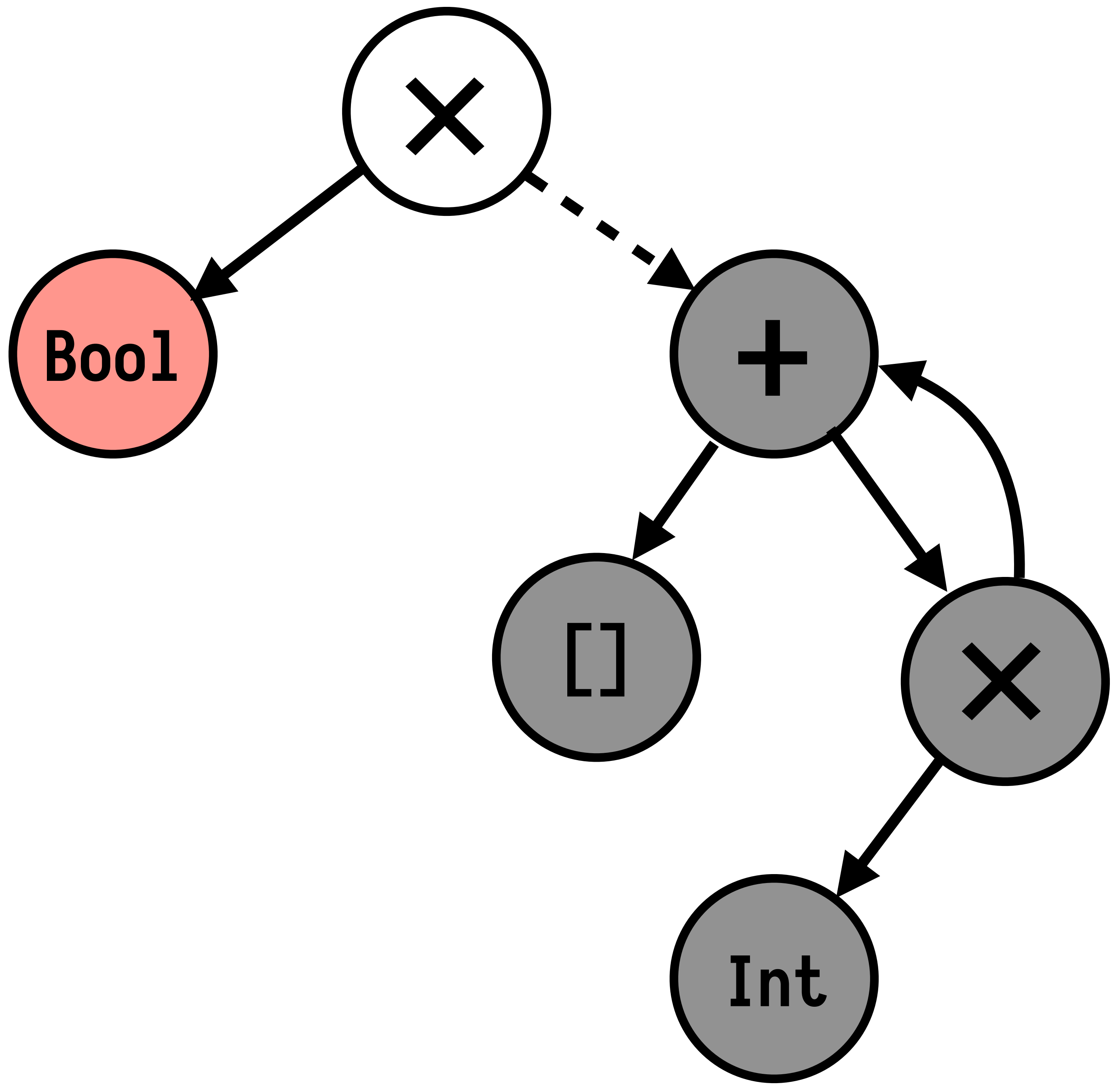
Value structure



Type structure



Type structure

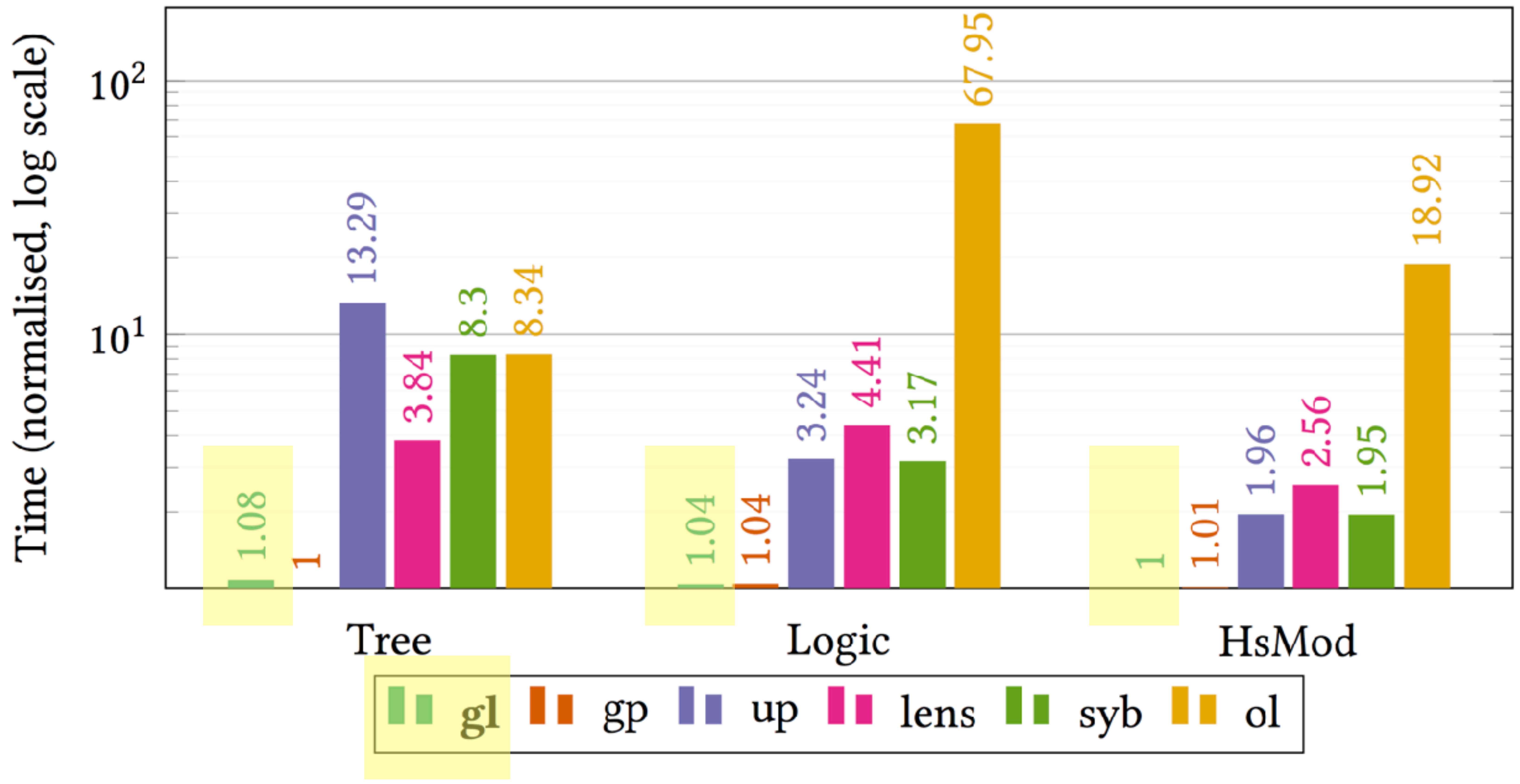


Type structure

```
type family Interesting (rep :: ⋆) (a :: ⋆) (seen :: [⋆]) :: Bool where
 Interesting (l :+: r) t seen = Interesting l t seen ∨ Interesting r t seen
 Interesting (l :×: r) t seen = Interesting l t seen ∨ Interesting r t seen
 Interesting (K t)     t seen = 'True
 Interesting (K r)     t seen = If (Elem r seen)
                                     'False
                                    (Interesting (Rep r) t (r ': seen))
 Interesting U t seen = 'False
 Interesting V t seen = 'False
```

Performance

Benchmarks



Execution time normalised as a factor of hand-written code

More!

- Lenses (by position, type, or field name)
- Prisms (by constructor name, or type)
- Type-inference of polymorphic traversals
- Custom type-errors
- Case study of inlining and specialisation (+ inspection testing)

Hackage: generic-lens

https://github.com/kcsongor/generic-lens

Thank you!