Numerical representations à la ornamentation

Josh Ko

Department of Computer Science
University of Oxford

Fun in the Afternoon
28 Feb 2012, Oxford, UK

Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	0	0	0	0

Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	1	0	0	0

Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	1 1	0	0	0

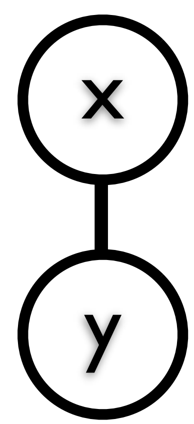
Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	0	1	0	0

Numerical representations

Operations on data structures ≈ numerical operations

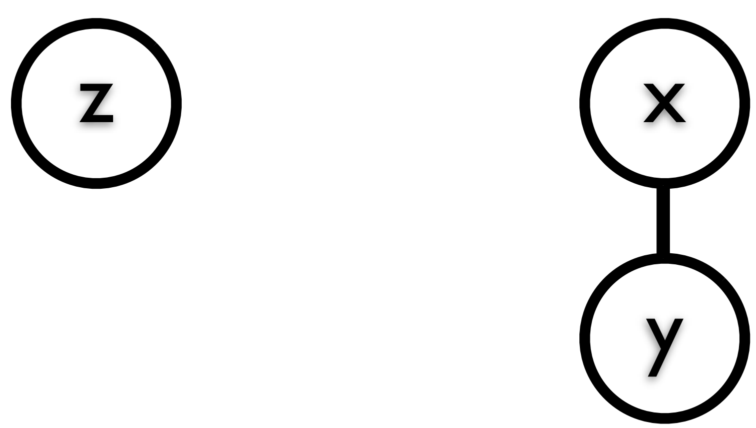
weight	2^0	2^1	2^2	2^3
number	0	1	0	0
data structure				

Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	1	1	0	0

data structure

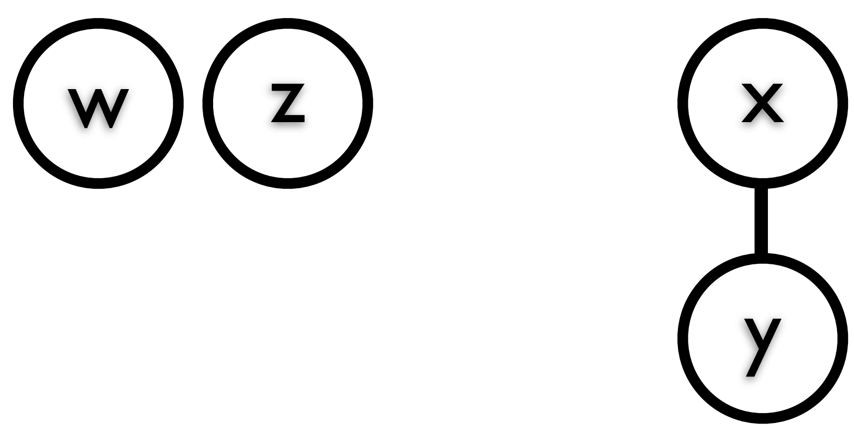


Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	1 1	1	0	0

data structure

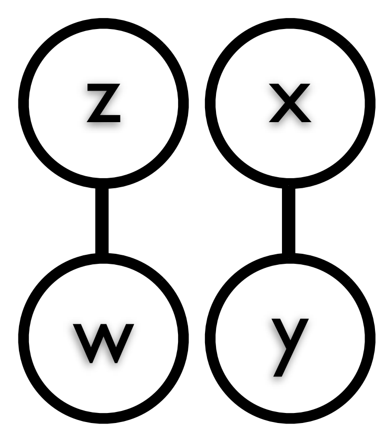


Numerical representations

Operations on data structures ≈ numerical operations

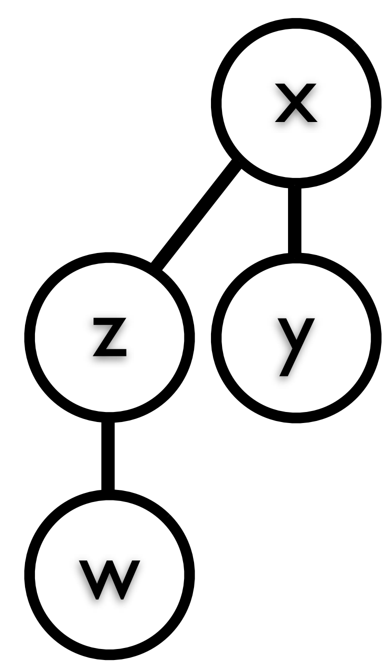
weight	2^0	2^1	2^2	2^3
number	0	1 1	0	0

data structure



Numerical representations

Operations on data structures ≈ numerical operations

weight	2^0	2^1	2^2	2^3
number	0	0	1	0
data structure				

Ornamenting a datatype

```
data Bin : Set where
  nul  : Bin
  zero : Bin → Bin
  one  : Bin → Bin
```

Ornamenting a datatype

```
data Bin : Set where
  nul  : Bin
  zero : Bin → Bin
  one  : BTree → Bin → Bin
```

Ornamenting a datatype

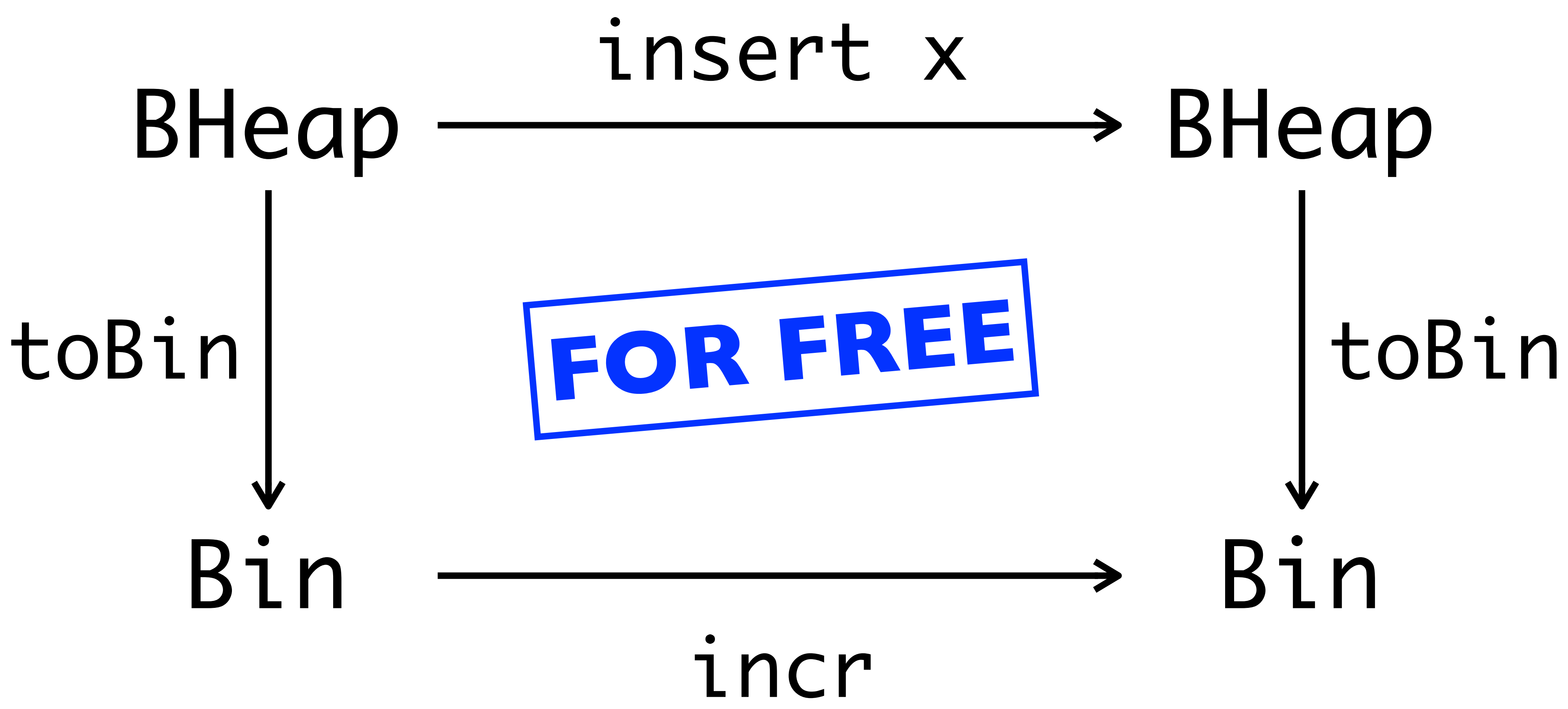
```
data BHeap : Set where
  nul  : BHeap
  zero : BHeap → BHeap
  one  : BTree → BHeap → BHeap

toBin : BHeap → Bin
toBin nul          = nul
toBin (zero   h) = zero (toBin h)
toBin (one t h) = one  (toBin h)
```

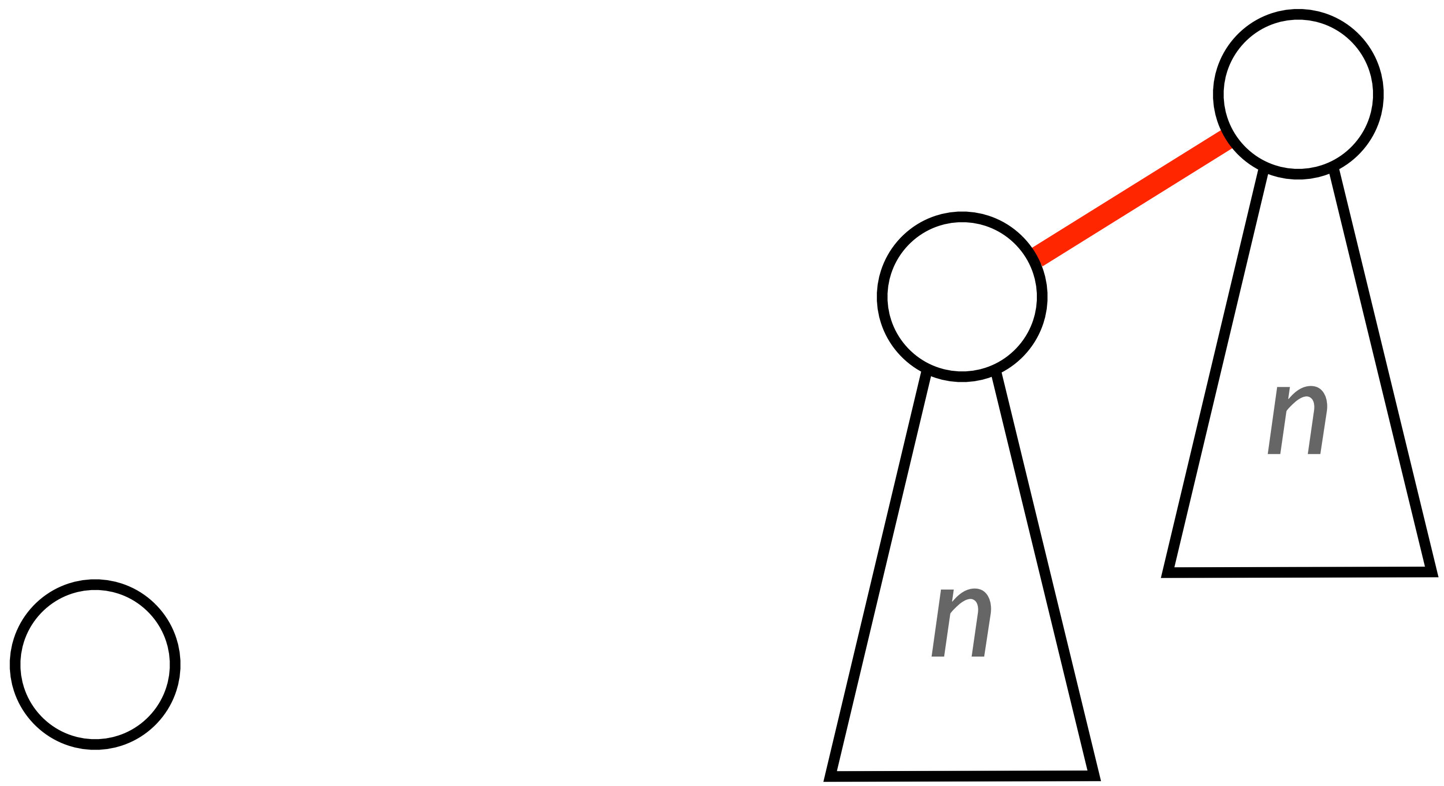
Coherence property

"Strong resemblance" made precise

```
incr   : Bin → Bin
insert : V → BHeap → BHeap
```



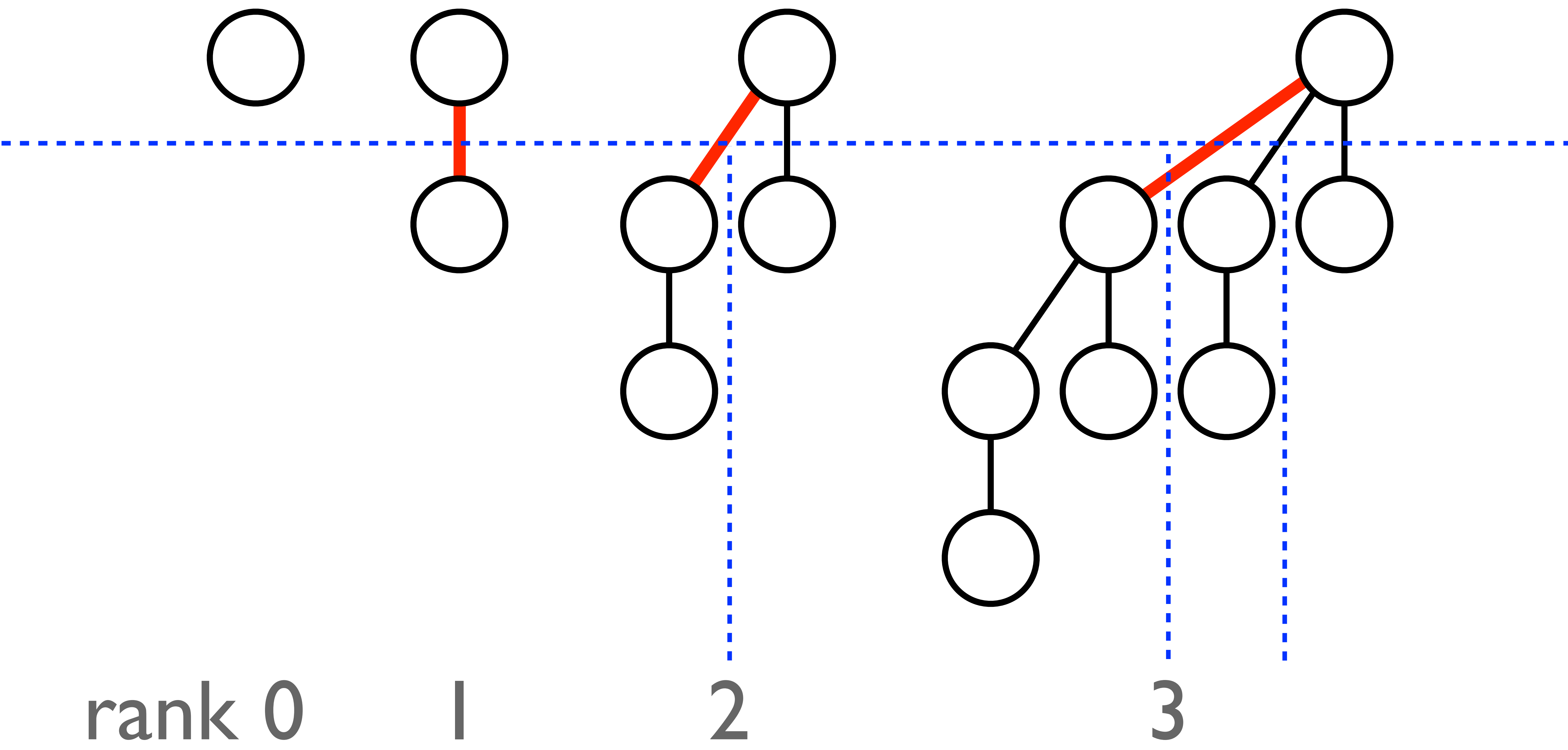
Binomial trees



rank 0 1 + *n*

$$\text{size} = 2^{\text{rank}}$$

Binomial trees



Binomial trees

using dependent types

```
data BTree : Nat → Set where
  node : V → BTree ^ r → BTree r

  -- BTree ^ 3 =
  --   BTree 2 × (BTree 1 × (BTree 0 × ⊤))

_^_ : (Nat → Set) → Nat → Set
X ^ zero    = ⊤
X ^ (suc r) = X r  ×  X ^ r
```

Binomial trees

using dependent types

```
attach : BTree r → BTree r → BTree (suc r)
attach t (node x ts) = node x (t , ts)

link : BTree r → BTree r → BTree (suc r)
link t u = if root t ≤ root u
             then attach t u else attach u t
```

```
data BTree : Nat → Set where
  node : V → BTree ^ r → BTree r

_^_ : (Nat → Set) → Nat → Set
X ^ zero    = ⊤
X ^ (suc n) = X n  ×  X ^ n
```

Binomial heaps

as an ornamentation of the binary number datatype

```
data BHeap : Nat → Set where
  nul  : BHeap r
  zero : BHeap (suc r) → BHeap r
  one  : BTree r → BHeap (suc r) → BHeap r
```

Binomial heaps

as an ornamentation of the binary number datatype

```
data BHeap : Nat → Set where
  nul  : BHeap r
  zero : BHeap (suc r) → BHeap r
  one  : BTree r → BHeap (suc r) → BHeap r
```

Binomial heaps

as an ornamentation of the binary number datatype

```
BinaryD : IDesc ⊤
BinaryD tt = σ C λ { nul  → ■
                   ; zero → v tt
                   ; one  → v tt }

⟦_⟧ : IDesc I → (I → Set) → (I → Set)

data μ (D : IDesc I) : I → Set where
  con : ⟦ D ⟧ (μ D) ⇒ μ D
```

Binomial heaps

as an ornamentation of the binary number datatype

```
BHeapOD : IOrnDesc Nat ! BinaryD
BHeapOD r =
  σ C λ { nul  → ■
        ; zero → v (ok (suc r))
        ; one  → Δ (BTree r)
                   λ _ → v (ok (suc r)) }
```

```
BinaryD : IDesc ⊤
BinaryD tt = σ C λ { nul  → ■
                   ; zero → v tt
                   ; one  → v tt }
```

Binomial heaps

as an ornamentation of the binary number datatype

```
BHeapOD : IOrnDesc Nat ! BinaryD
BHeapOD r =
  σ C λ { nul  → ■
        ; zero → v (ok (suc r))
        ; one  → Δ (BTree r)
                   λ _ → v (ok (suc r)) }

⌞_⌟ : IOrnDesc J e D → Desc J
```

Binomial heaps

as an ornamentation of the binary number datatype

```
BHeapOD : IOrnDesc Nat ! BinaryD
BHeapOD r =
  σ C λ { nul  → ■
        ; zero → v (ok (suc r))
        ; one  → Δ (BTree r)
                   λ _ → v (ok (suc r)) }

forget : (O : IOrnDesc J e D) →
         μ ⌞ O ⌟ ⇒ μ D ∘ e
```

Binomial heaps

as an ornamentation of the binary number datatype

```
data BHeap : Nat → Set where
  nul  : BHeap r
  zero : BHeap (suc r) → BHeap r
  one  : BTree r → BHeap (suc r) → BHeap r

toBin : BHeap r → Bin
toBin nul         = nul
toBin (zero  h) = zero (toBin h)
toBin (one t h) = one  (toBin h)
```

Increment & insertion

```
incr : Bin → Bin
incr nul        = one nul
incr (zero b) = one b
incr (one  b) = zero (incr b)

insT : BTree r → BHeap r → BHeap r
insT t nul          = one t nul
insT t (zero  h) = one t h
insT t (one u h) = zero (insT (link t u) h)

insert : V → BHeap 0 → BHeap 0
insert x = insT (node x tt)
```

```
incr : Bin → Bin
incr nul      = one nul
incr (zero b) = one b
incr (one  b) = zero (incr b)
```

We do not get the coherence property for free!

```
insT : BTree r → BHeap r → BHeap r
insT t nul        = one t nul
insT t (zero  h) = one t h
insT t (one u h) = zero (insT (link t u) h)
```

Realisability predicate

Indexing the type of a heap with its underlying number

```
data BHeap' : Nat → Bin → Set where
 nul  : BHeap' r nul
 zero : BHeap' (suc r) b → BHeap' r (zero b)
 one  : BTree r →
        BHeap' (suc r) b → BHeap' r (one b)
```

```
BHeap r  ≅  (b : Bin) × BHeap' r b
```

Realisability predicate

Indexing the type of a heap with its underlying number

```
BHeap r  ≅  (b : Bin) × BHeap' r b
```

```
toBin      : BHeap r → Bin
fromBHeap : (h : BHeap r) → BHeap' r (toBin h)
fromBHeap nul        = nul
fromBHeap (zero  h) = zero  (fromBHeap h)
fromBHeap (one t h) = one t (fromBHeap h)
```

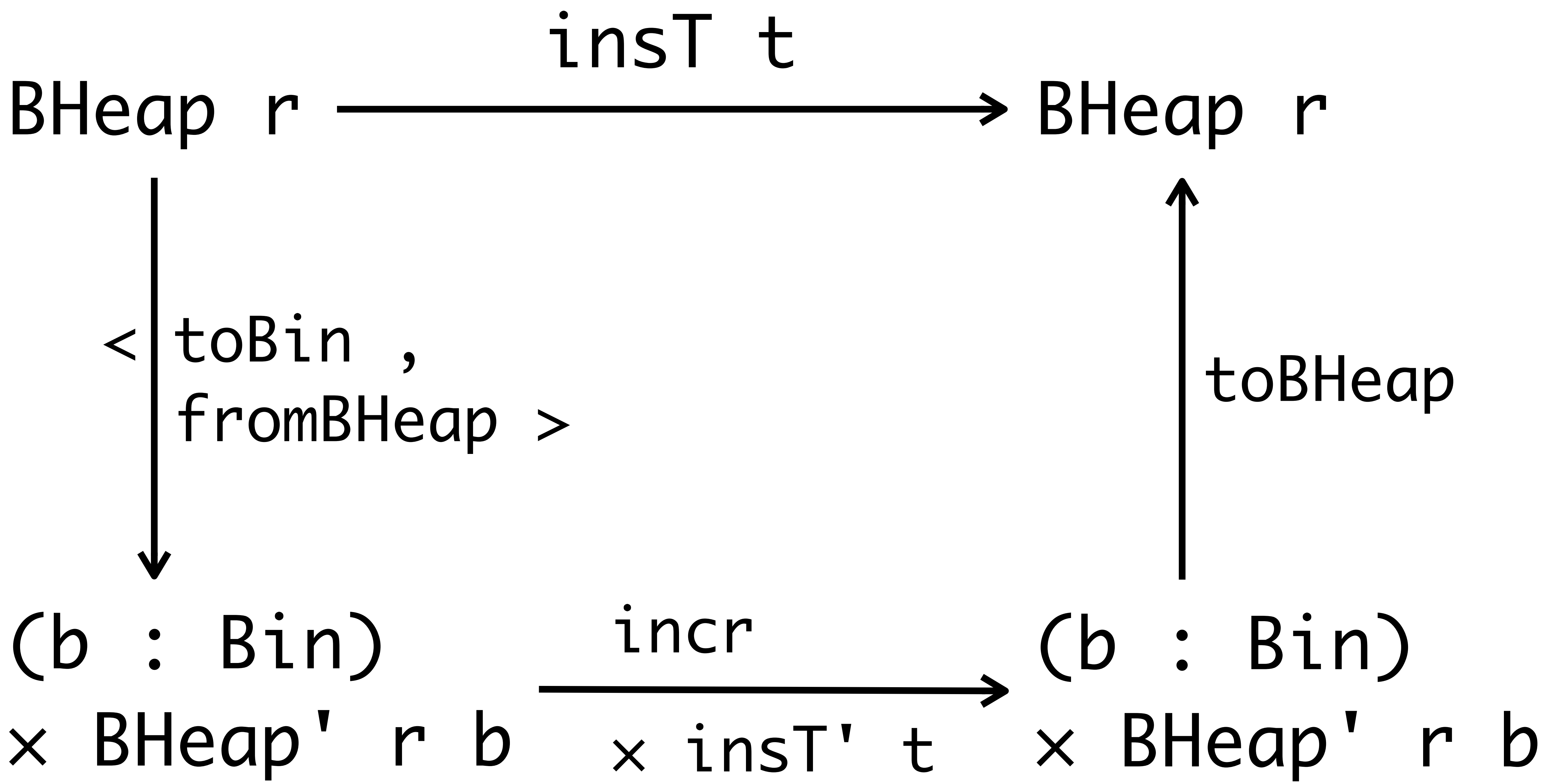
```
toBHeap : (b : Bin) × BHeap' r b → BHeap r
toBHeap (._ , nul     ) = nul
toBHeap (._ , zero  h) = zero  (toBHeap h)
toBHeap (._ , one t h) = one t (toBHeap h)
```

Insertion revisited

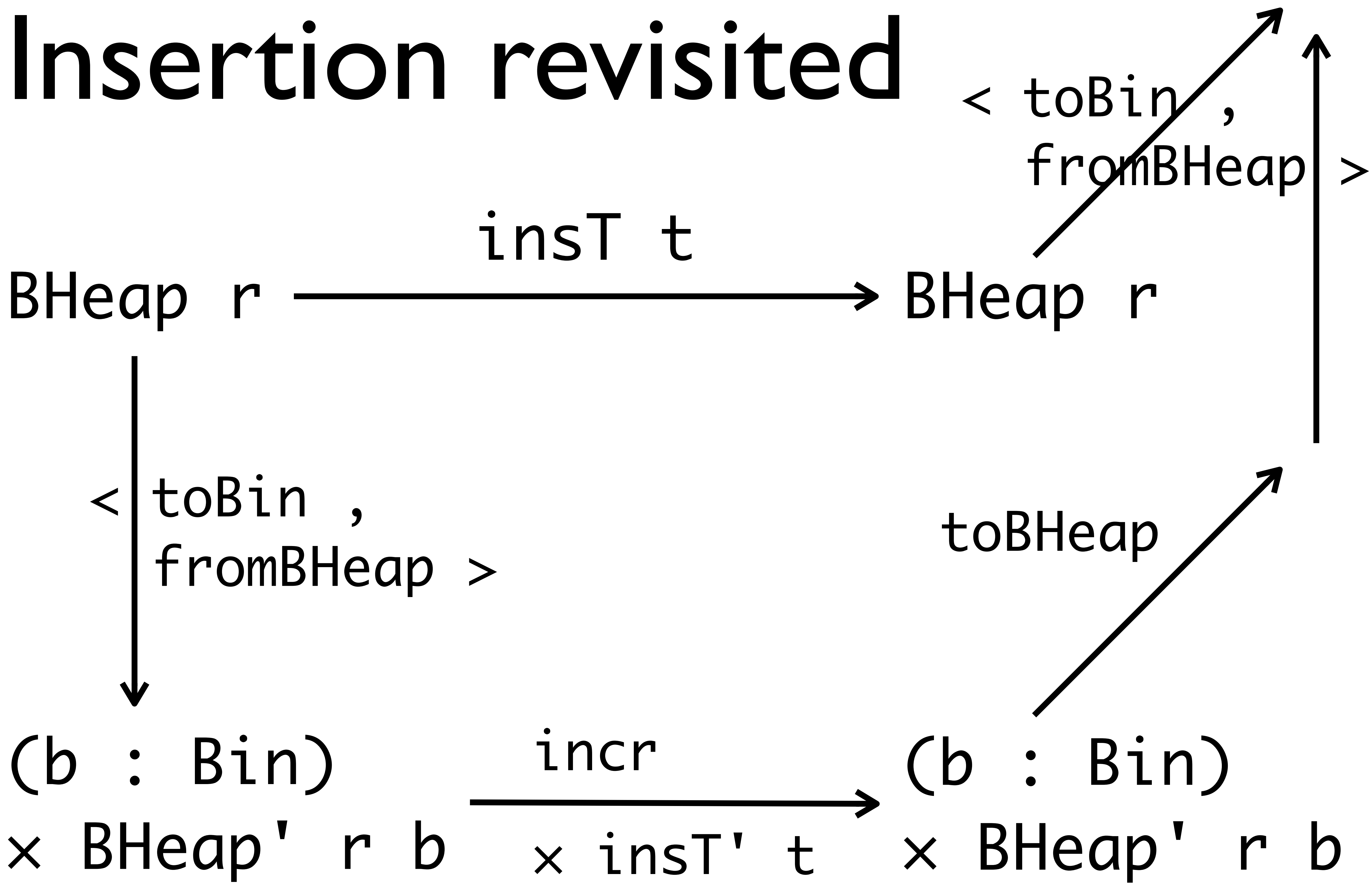
```
incr : Bin → Bin
incr nul        = one nul
incr (zero b) = one b
incr (one  b) = zero (incr b)

insT' : BTree r →
        BHeap' r b → BHeap' r (incr b)
insT' t nul         = one t nul
insT' t (zero  h) = one t h
insT' t (one u h) = zero (insT' (link t u) h)
```

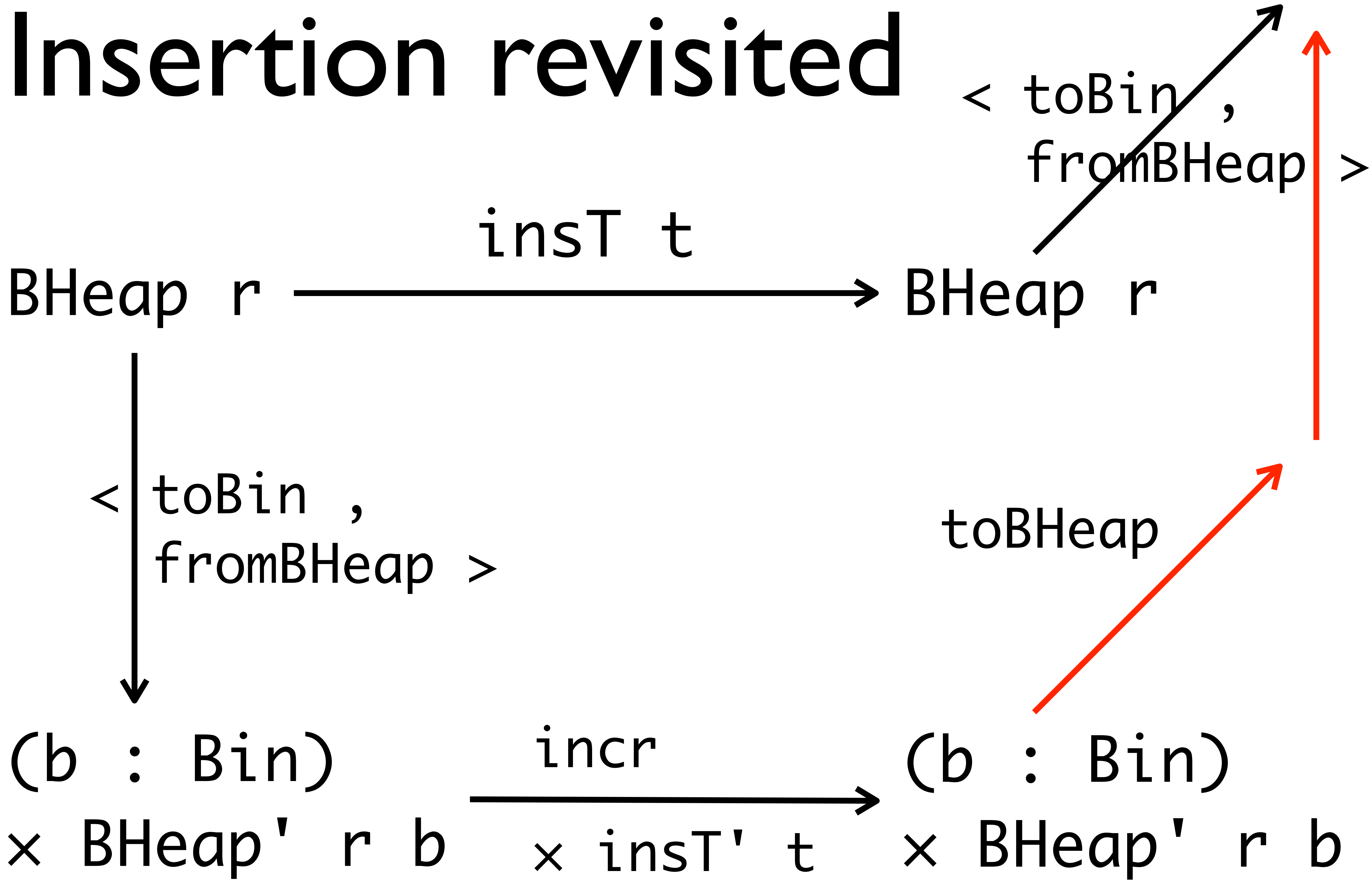
Insertion revisited



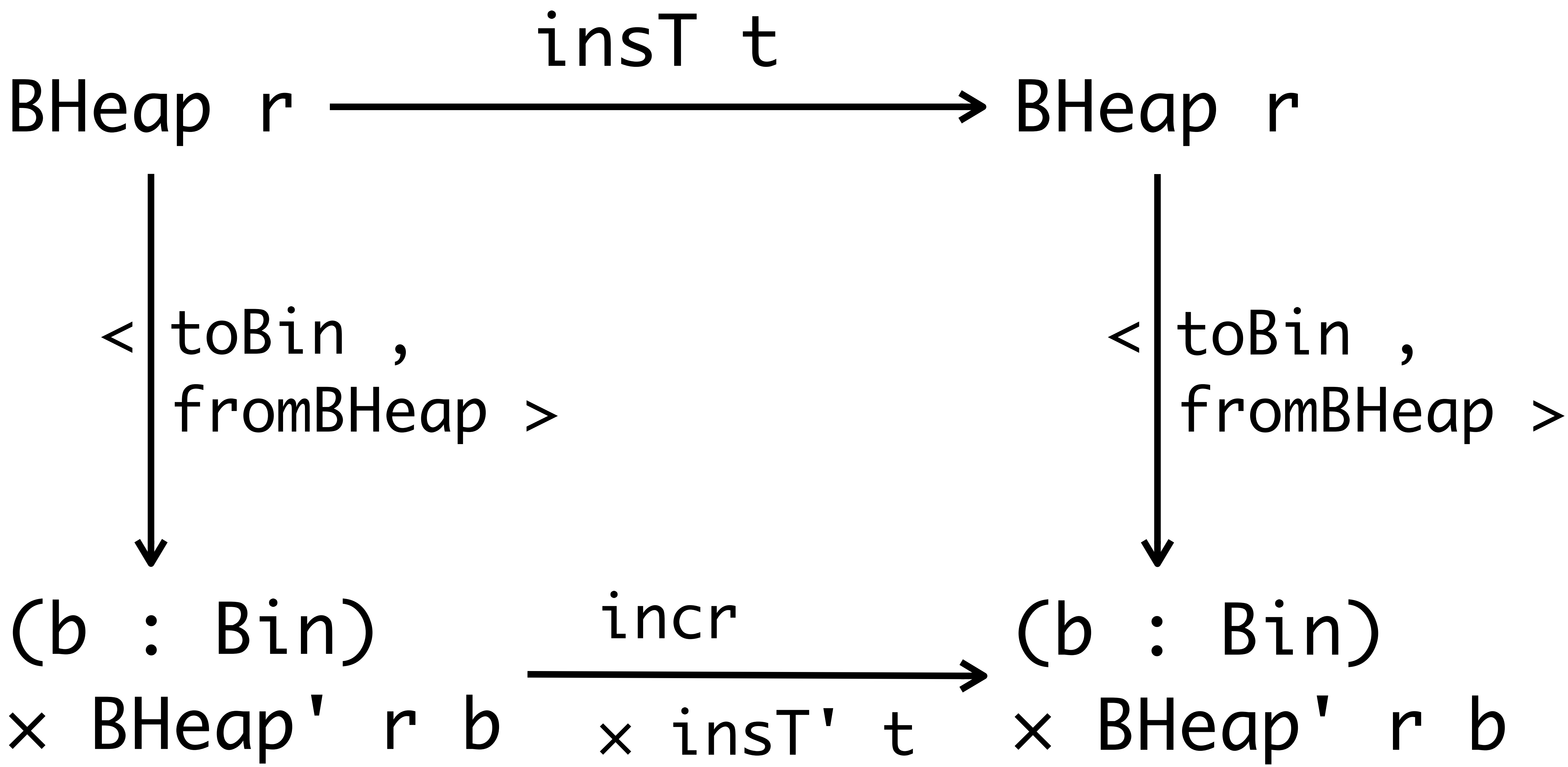
Insertion revisited



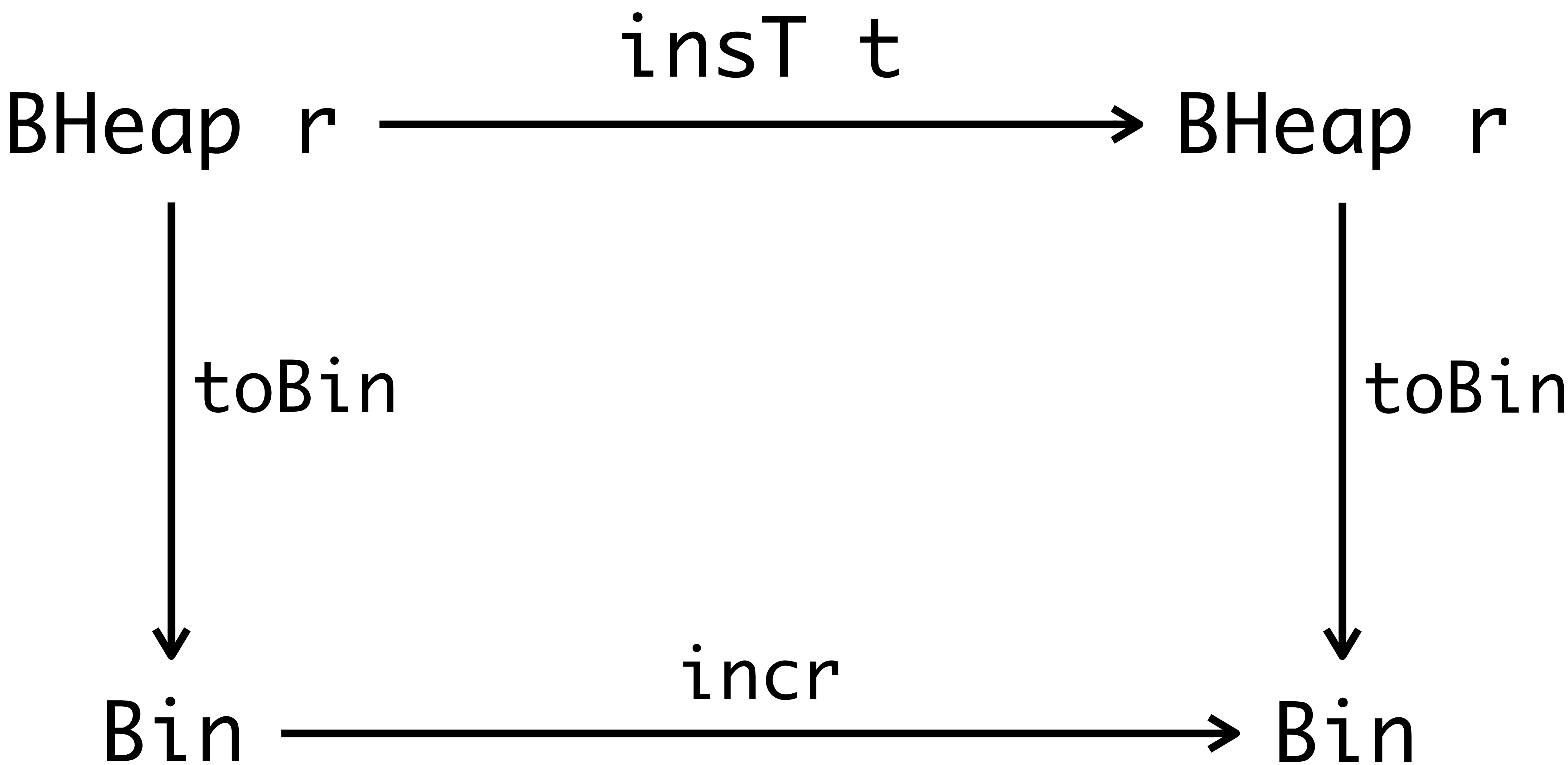
Insertion revisited



Insertion revisited



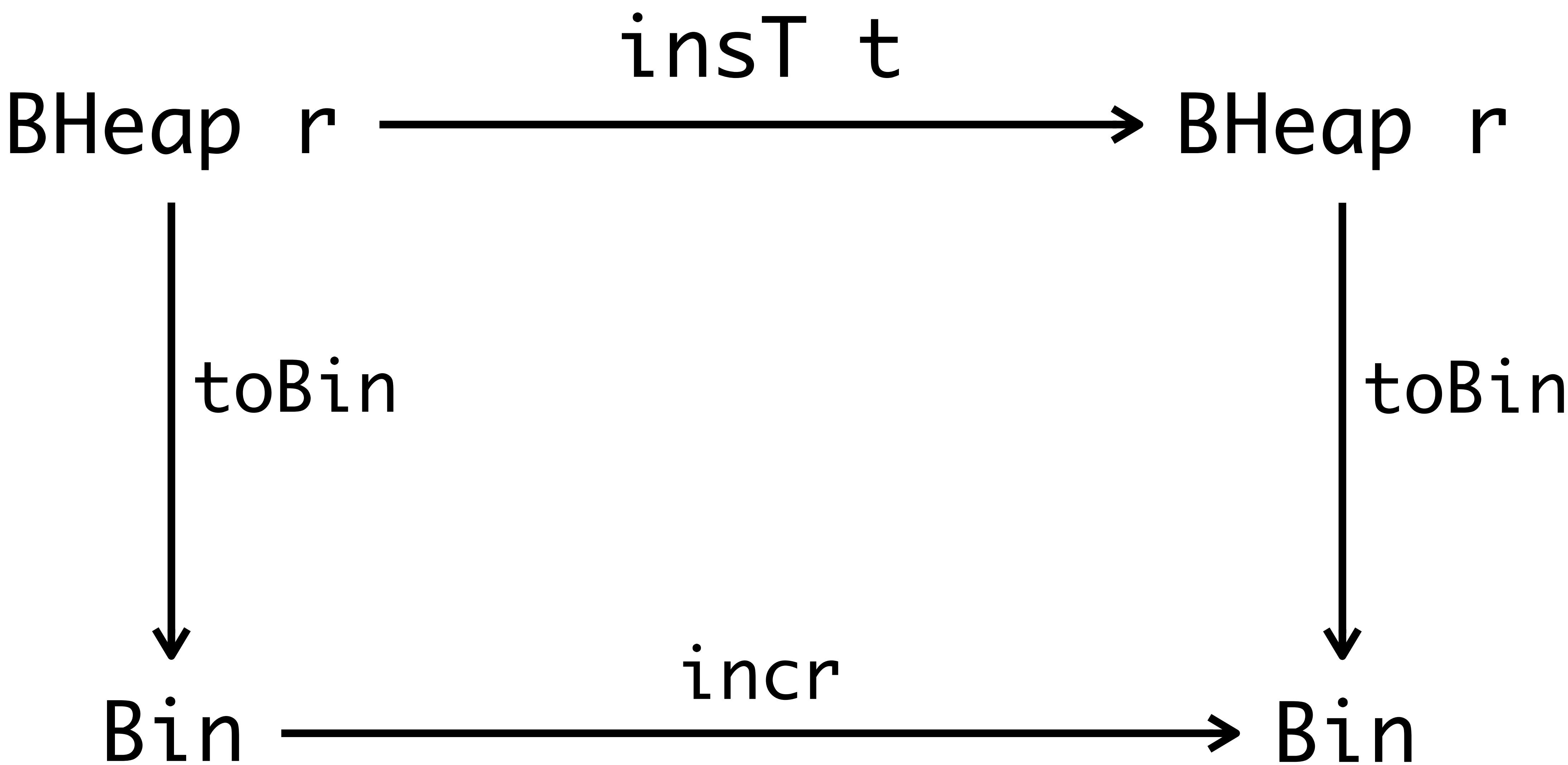
Coherence property



Coherence property

A calculational proof

```
toBin ∘ insT t = incr ∘ toBin
```



Coherence property

A calculational proof

```
  toBin ∘ insT t
=   { definition of insT }
  toBin ∘ toBHeap ∘
    (incr × insT' t) ∘ < toBin , fromBHeap >
=   { cancellation; absorption }
  fst ∘ < toBin , fromBHeap > ∘ toBHeap ∘
    < incr ∘ toBin , insT' t ∘ fromBHeap >
=   { isomorphism }
  fst ∘ < incr ∘ toBin , insT' t ∘ fromBHeap >
=   { cancellation }
  incr ∘ toBin
```

Cost and gain

Write:

`Bin`, and `BHeap` as an ornamentation of `Bin`

`incr` on `Bin` and `insT'` on `BHeap'`

Get (via generic programming):

realisability predicate `BHeap'` and corresponding isomorphism

`insT` on `BHeap` and the coherence property w.r.t. `incr`

Where the ideas come from

and also where to find more

1. Conor McBride. Ornamental algebras, algebraic ornaments. To appear in *Journal of Functional Programming*.
2. Hsiang-Shang Ko and Jeremy Gibbons. Modularising inductive families. *Workshop on Generic Programming* 2011.
3. Pierre-Evariste Dagand and Conor McBride. Transporting functions across ornaments. Technical report, January 2012.

Thanks!

Towards extraction

Total functions only in dependently typed programs

```
data Bin : Bool → Set where
  nul  : Bin false
  zero : Bin nz → Bin nz
  one  : Bin nz → Bin true

decr : Bin true → (nz : Bool) × Bin nz
decr (zero b) = _ , one (snd (decr b))
decr (one  b) = _ , zero b
```

Towards extraction

Total functions only in dependently typed programs