

# AVL Trees

D Oisín Kidney

July 28, 2018

```

open import Relation.Binary.PropositionalEquality
open import Level using (Lift; lift;  $\_ \sqcup \_$ )
open import Data.Nat as  $\mathbb{N}$  using ( $\overline{\mathbb{N}}$ ; suc; zero; pred)
open import Data.Product
open import Data.Unit renaming ( $\top$  to 1)
open import Data.Maybe
open import Function
open import Data.Bool
open import Data.Empty renaming ( $\perp$  to 0)

module AVL
  {k r} (Key : Set k)
  { $\_ < \_$  : Rel Key r}
  (isStrictTotalOrder : IsStrictTotalOrder  $\_ \equiv \_ < \_$ )
  where

    open IsStrictTotalOrder isStrictTotalOrder

infix 5 [ $\_$ ]

data  $\top$  : Set k where
   $\perp$   $\top$  :  $\top$ 
  [ $\_$ ] : (k : Key)  $\rightarrow$   $\top$ 

infix 4  $\top$   $\_ < \_$ 
 $\_ \top < \_$  :  $\top \rightarrow \top \rightarrow$  Set r
 $\perp \top < \perp$  = Lift r 0
 $\perp \top < \top$  = Lift r 1
 $\perp \top < [\_]$  = Lift r 1
 $\top \top < \perp$  = Lift r 0
 $[\_] \top < \perp$  = Lift r 0
 $[\_] \top < \top$  = Lift r 1
 $[x] \top < [y]$  =  $x < y$ 

```

infix 4  $\_<\_<\_$

$\_<\_<\_ : \mathbb{I} \rightarrow \text{Key} \rightarrow \mathbb{I} \rightarrow \text{Set } r$   
 $l < x < u = l \mathbb{I} < [x] \times [x] \mathbb{I} < u$

module Bounded where

data  $\langle \_ \sqcup \_ \rangle \equiv \_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Set}$  where

$\swarrow : \forall \{n\} \rightarrow \langle \text{suc } n \sqcup \quad n \rangle \equiv \text{suc } n$   
 $\dashv : \forall \{n\} \rightarrow \langle \quad n \sqcup \quad n \rangle \equiv n$   
 $\searrow : \forall \{n\} \rightarrow \langle \quad n \sqcup \text{suc } n \rangle \equiv \text{suc } n$

$\swarrow \Rightarrow \swarrow : \forall \{x \ y \ z\} \rightarrow \langle x \sqcup y \rangle \equiv z \rightarrow \langle z \sqcup x \rangle \equiv z$

$\swarrow \Rightarrow \swarrow \swarrow = \dashv$

$\swarrow \Rightarrow \swarrow \dashv = \dashv$

$\swarrow \Rightarrow \swarrow \searrow = \swarrow$

$\swarrow \Rightarrow \searrow : \forall \{x \ y \ z\} \rightarrow \langle x \sqcup y \rangle \equiv z \rightarrow \langle y \sqcup z \rangle \equiv z$

$\swarrow \Rightarrow \searrow \swarrow = \searrow$

$\swarrow \Rightarrow \searrow \dashv = \dashv$

$\swarrow \Rightarrow \searrow \searrow = \dashv$

data Tree  $\{v\} (V : \text{Key} \rightarrow \text{Set } v) (l \ u : \mathbb{I}) : \mathbb{N} \rightarrow \text{Set} (k \sqcup v \sqcup r)$  where

leaf :  $(l < u : l \mathbb{I} < u) \rightarrow \text{Tree } V \ l \ u \ 0$

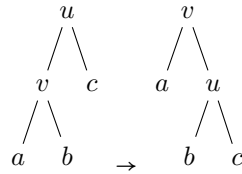
node :  $\forall \{h \ lh \ rh\}$   
 $(k : \text{Key})$   
 $(v : V \ k)$   
 $(bl : \langle lh \sqcup rh \rangle \equiv h)$   
 $(lk : \text{Tree } V \ l \ [k] \ lh)$   
 $(ku : \text{Tree } V \ [k] \ u \ rh) \rightarrow$   
 $\text{Tree } V \ l \ u \ (\text{suc } h)$

Inserted :  $\forall \{v\} (V : \text{Key} \rightarrow \text{Set } v) (l \ u : \mathbb{I}) (n : \mathbb{N}) \rightarrow \text{Set} (k \sqcup v \sqcup r)$

Inserted  $V \ l \ u \ n = \exists [inc] (\text{Tree } V \ l \ u \ (\text{if } inc \text{ then suc } n \text{ else } n))$

pattern same  $tr = \text{false}$  ,  $tr$

pattern chng  $tr = \text{true}$  ,  $tr$

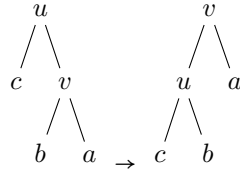


rot-r :  $\forall \{lb \ ub \ rh \ v\} \{V : \text{Key} \rightarrow \text{Set } v\}$

```

→ (k : Key)
→ V k
→ Tree V lb [ k ] (suc (suc rh))
→ Tree V [ k ] ub rh
→ Inserted V lb ub (suc (suc rh))
rot-r u uc (node v vc ⋈ ta tb) tc = same (node v vc ⋈ ta (node u uc ⋈ tb tc))
rot-r u uc (node v vc ⋈ ta tb) tc = chng (node v vc ⋈ ta (node u uc ⋈ tb tc))
rot-r u uc (node v vc ⋈ ta (node w wc bw tb tc)) td =
  same (node w wc ⋈ (node v vc (⋈ ⇒ ⋈ bw) ta tb) (node u uc (⋈ ⇒ ⋈ bw) tc td))

```



```

rot-l : ∀ {lb ub lh v} {V : Key → Set v}
→ (k : Key)
→ V k
→ Tree V lb [ k ] lh
→ Tree V [ k ] ub (suc (suc lh))
→ Inserted V lb ub (suc (suc lh))
rot-l u uc tc (node v vc ⋈ tb ta) = same (node v vc ⋈ (node u uc ⋈ tc tb) ta)
rot-l u uc tc (node v vc ⋈ tb ta) = chng (node v vc ⋈ (node u uc ⋈ tc tb) ta)
rot-l u uc td (node v vc ⋈ (node w wc bw tc tb) ta) =
  same (node w wc ⋈ (node u uc (⋈ ⇒ ⋈ bw) td tc) (node v vc (⋈ ⇒ ⋈ bw) tb ta))

```

```

insert : ∀ {l u h v} {V : Key → Set v} (k : Key)
→ V k
→ (V k → V k → V k)
→ Tree V l u h
→ l < k < u
→ Inserted V l u h

```

```

insert v vc f (leaf l < u) (l , u) = chng (node v vc ⋈ (leaf l) (leaf u))

```

```

insert v vc f (node k kc bl tl tr) prf with compare v k

```

```

insert v vc f (node k kc bl tl tr) (l , _)
| tri < a _ _ with insert v vc f tl (l , a)
... | same tl' = same (node k kc bl tl' tr)

```

```

... | chng tl' with bl
... | ⋈ = rot-r k kc tl' tr

```

```

... | ⋈ = chng (node k kc ⋈ tl' tr)
... | ⋈ = same (node k kc ⋈ tl' tr)

```

```

insert v vc f (node k kc bl tl tr) _
| tri ≈ _ refl _ = same (node k (f vc kc) bl tl tr)
insert v vc f (node k kc bl tl tr) (_, u)

```

```

    | tri> _ _ c with insert v vc f tr (c , u)
... | same tr' = same (node k kc bl tl tr')
... | chng tr' with bl
... |  / = same (node k kc  tl tr')
... |  = chng (node k kc  tl tr')
... |  = rot-l k kc tl tr'

```

```

lookup : (k : Key)
  → ∀ {l u s v} {V : Key → Set v}
  → Tree V l u s
  → Maybe (V k)
lookup k (leaf l<u) = nothing
lookup k (node v vc _ tl tr) with compare k v
... | tri< _ _ _ = lookup k tl
... | tri≈ _ refl _ = just vc
... | tri> _ _ _ = lookup k tr

```

```

module DependantMap where
data Map {v} (V : Key → Set v) : Set (k ⊔ v ⊔ r) where
  tree : ∀ {h} → Bounded.Tree V ⊥ ⊤ h → Map V

```

```

insertWith : ∀ {v} {V : Key → Set v} (k : Key)
  → V k
  → (V k → V k → V k)
  → Map V
  → Map V
insertWith k v f (tree tr) =
  tree (proj₂ (Bounded.insert k v f tr (lift tt , lift tt)))

```

```

insert : ∀ {v} {V : Key → Set v} (k : Key) → V k → Map V → Map V
insert k v = insertWith k v const

```

```

lookup : (k : Key) → ∀ {v} {V : Key → Set v} → Map V → Maybe (V k)
lookup k (tree tr) = Bounded.lookup k tr

```

```

module Map where
data Map {v} (V : Set v) : Set (k ⊔ v ⊔ r) where
  tree : ∀ {h} → Bounded.Tree (const V) ⊥ ⊤ h → Map V

```

```

insertWith : ∀ {v} {V : Set v} (k : Key) → V → (V → V → V) → Map V → Map V
insertWith k v f (tree tr) =
  tree (proj₂ (Bounded.insert k v f tr (lift tt , lift tt)))

```

```

insert : ∀ {v} {V : Set v} (k : Key) → V → Map V → Map V
insert k v = insertWith k v const

```

```

lookup : (k : Key) → ∀ {v} {V : Set v} → Map V → Maybe V

```

```

lookup  $k$  (tree  $tr$ ) = Bounded.lookup  $k$   $tr$ 

module Sets where
data ⟨Set⟩ : Set ( $k \sqcup r$ ) where
  tree :  $\forall \{h\} \rightarrow$  Bounded.Tree (const 1)  $\perp \top$   $h \rightarrow$  ⟨Set⟩

insert : Key  $\rightarrow$  ⟨Set⟩  $\rightarrow$  ⟨Set⟩
insert  $k$  (tree  $tr$ ) =
  tree (proj2 (Bounded.insert  $k$  tt const  $tr$  (lift tt , lift tt)))

member : Key  $\rightarrow$  ⟨Set⟩  $\rightarrow$  Bool
member  $k$  (tree  $tr$ ) = is-just (Bounded.lookup  $k$   $tr$ )

```