

# AVL Trees

D Oisín Kidney

July 28, 2018

```

open import Relation.Binary.PropositionalEquality
open import Level using (Lift; lift;  $\sqsubseteq$ _; lower)
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  $\_ \sqsubseteq \_$ 
 $\_ \sqsubseteq \_$  :  $\top \rightarrow \top \rightarrow$  Set r
 $\perp \sqsubseteq \perp$  = Lift r 0
 $\perp \sqsubseteq \top$  = Lift r 1
 $\perp \sqsubseteq [\_]$  = Lift r 1
 $\top \sqsubseteq \perp$  = Lift r 0
 $[\_] \sqsubseteq \perp$  = Lift r 0
 $[\_] \sqsubseteq \top$  = Lift r 1
 $[x] \sqsubseteq [y]$  =  $x < y$ 

```

```

x<⊥ : ∀ {x} → x ⊥ < ⊥ → Lift r 0
x<⊥ {⊥} = lift ∘ lower
x<⊥ {⊤} = lift ∘ lower
x<⊥ {[ _ ]} = lift ∘ lower

⊤<-trans : ∀ {x y z} → x ⊤ < y → y ⊤ < z → x ⊤ < z
⊤<-trans {⊥} {y} {⊥} _ y<z = x<⊥ {x = y} y<z
⊤<-trans {⊥} {} {⊤} _ _ = _
⊤<-trans {⊥} {} {[ _ ]} _ _ = _
⊤<-trans {⊤} {} {} (lift ()) _ = _
⊤<-trans {[ _ ]} {y} {⊥} _ y<z = x<⊥ {x = y} y<z
⊤<-trans {[ _ ]} {} {⊤} _ _ = _
⊤<-trans {[ _ ]} {⊥} {[ _ ]} (lift ()) _ = _
⊤<-trans {[ _ ]} {⊤} {[ _ ]} (lift ()) _ = _
⊤<-trans {[ x ]} {[ y ]} {[ z ]} x<y y<z =
  IsStrictTotalOrder.trans isStrictTotalOrder x<y y<z

```

```

infix 4 _<_<_

```

```

_<_<_ : ⊤ → Key → ⊤ → Set r
l < x < u = l ⊤ < [ x ] × [ x ] ⊤ < u

```

```

module Bounded where

```

```

data { _ ⊔ _ } ≡ _ : ℕ → ℕ → ℕ → Set where
  ⋈ : ∀ {n} → { suc n ⊔ n } ≡ suc n
  ⊔ : ∀ {n} → { n ⊔ n } ≡ n
  ⋇ : ∀ {n} → { n ⊔ suc n } ≡ suc n

⋈ ⇒ ⋈ : ∀ {x y z} → { x ⊔ y } ≡ z → { z ⊔ x } ≡ z
⋈ ⇒ ⋈ ⋈ = ⊔
⋈ ⇒ ⋈ ⊔ = ⊔
⋈ ⇒ ⋈ ⋇ = ⋈

⋇ ⇒ ⋇ : ∀ {x y z} → { x ⊔ y } ≡ z → { y ⊔ z } ≡ z
⋇ ⇒ ⋇ ⋈ = ⋇
⋇ ⇒ ⋇ ⊔ = ⊔
⋇ ⇒ ⋇ ⋇ = ⊔

```

```

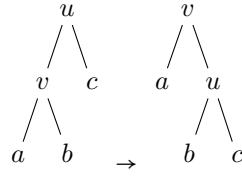
data Tree {v} (V : Key → Set v) (l u : ⊤) : ℕ → Set (k ⊔ v ⊔ r) where
  leaf : (l < u : l ⊤ < u) → Tree V l u 0
  node : ∀ {h lh rh}
    (k : Key)
    (v : V k)
    (bl : { lh ⊔ rh } ≡ h)

```

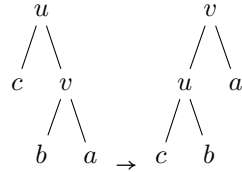
$(lk : \text{Tree } V l [k] lh)$   
 $(ku : \text{Tree } V [k] u rh) \rightarrow$   
 $\text{Tree } V l u (\text{suc } h)$

$\text{Altered} : \forall \{v\} (V : \text{Key} \rightarrow \text{Set } v) (l u : \mathbb{I}) (n : \mathbb{N}) \rightarrow \text{Set } (k \sqcup v \sqcup r)$   
 $\text{Altered } V l u n = \exists [inc] (\text{Tree } V l u (\text{if } inc \text{ then } \text{suc } n \text{ else } n))$

$\text{pattern } 0+ \text{ } tr = \text{false} , tr$   
 $\text{pattern } 1+ \text{ } tr = \text{true} , tr$



$\text{rot}^r : \forall \{lb ub rh v\} \{V : \text{Key} \rightarrow \text{Set } v\}$   
 $\rightarrow (k : \text{Key})$   
 $\rightarrow V k$   
 $\rightarrow \text{Tree } V lb [k] (\text{suc } (\text{suc } rh))$   
 $\rightarrow \text{Tree } V [k] ub rh$   
 $\rightarrow \text{Altered } V lb ub (\text{suc } (\text{suc } rh))$   
 $\text{rot}^r u uc (\text{node } v vc \nearrow ta tb) tc = 0+ (\text{node } v vc \div ta (\text{node } u uc \div tb tc))$   
 $\text{rot}^r u uc (\text{node } v vc \div ta tb) tc = 1+ (\text{node } v vc \searrow ta (\text{node } u uc \nearrow tb tc))$   
 $\text{rot}^r u uc (\text{node } v vc \searrow ta (\text{node } w wc bw tb tc)) td =$   
 $0+ (\text{node } w wc \div (\text{node } v vc (\searrow \Rightarrow \nearrow bw) ta tb) (\text{node } u uc (\nearrow \Rightarrow \searrow bw) tc td))$



$\text{rot}^l : \forall \{lb ub lh v\} \{V : \text{Key} \rightarrow \text{Set } v\}$   
 $\rightarrow (k : \text{Key})$   
 $\rightarrow V k$   
 $\rightarrow \text{Tree } V lb [k] lh$   
 $\rightarrow \text{Tree } V [k] ub (\text{suc } (\text{suc } lh))$   
 $\rightarrow \text{Altered } V lb ub (\text{suc } (\text{suc } lh))$   
 $\text{rot}^l u uc tc (\text{node } v vc \searrow tb ta) = 0+ (\text{node } v vc \div (\text{node } u uc \div tc tb) ta)$   
 $\text{rot}^l u uc tc (\text{node } v vc \div tb ta) = 1+ (\text{node } v vc \nearrow (\text{node } u uc \searrow tc tb) ta)$   
 $\text{rot}^l u uc td (\text{node } v vc \nearrow (\text{node } w wc bw tc tb) ta) =$   
 $0+ (\text{node } w wc \div (\text{node } u uc (\searrow \Rightarrow \nearrow bw) td tc) (\text{node } v vc (\nearrow \Rightarrow \searrow bw) tb ta))$

$\text{insert} : \forall \{l u h v\} \{V : \text{Key} \rightarrow \text{Set } v\} (k : \text{Key})$

```

→ V k
→ (V k → V k → V k)
→ Tree V l u h
→ l < k < u
→ Altered V l u h
insert v vc f (leaf l < u) (l, u) = 1+ (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)
... | 0+ tl' = 0+ (node k kc bl tl' tr)
... | 1+ tl' with bl
... | ⋈ = rotr k kc tl' tr
... | ▸ = 1+ (node k kc ⋈ tl' tr)
... | ⋋ = 0+ (node k kc ▸ tl' tr)
insert v vc f (node k kc bl tl tr) _
| tri ≈ _ refl _ = 0+ (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)
... | 0+ tr' = 0+ (node k kc bl tl tr')
... | 1+ tr' with bl
... | ⋈ = 0+ (node k kc ▸ tl tr')
... | ▸ = 1+ (node k kc ⋋ tl tr')
... | ⋋ = rotl 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

```

```

record Uncons {v} (V : Key → Set v) (lb : ℤ) (ub : ℤ) (h : ℕ) : Set (k ⊔ v ⊔ r) where
  constructor uncons
  field
    head : Key
    val : V head
    l < u : lb ℤ < [ head ]
    tail : Altered V [ head ] ub h

```

```

uncons' : ∀ {lb ub h lh rh v} {V : Key → Set v}
→ (k : Key)
→ V k

```

```

→ ⟨ lh ⊔ rh ⟩≡ h
→ Tree V lb [ k ] lh
→ Tree V [ k ] ub rh
→ Uncons V lb ub h
uncons' k v bl tl tr = go k v bl tl tr id
where
go : ∀ {lb ub h lh rh v ub' h'} {V : Key → Set v}
→ (k : Key)
→ V k
→ ⟨ lh ⊔ rh ⟩≡ h
→ Tree V lb [ k ] lh
→ Tree V [ k ] ub rh
→ (∀ {lb'} → Altered V [ lb' ] ub h → Altered V [ lb' ] ub' h')
→ Uncons V lb ub' h'
go k v ⊢ (leaf l<u) tr c = uncons k v l<u (c (0+ tr))
go k v ⊢ (node kl vl bll tll trl) tr c = go kl vl bll tll trl
λ { (0+ tl') → c (1+ (node k v ⊎ tl' tr))
; (1+ tl') → c (1+ (node k v ⊢ tl' tr)) }
go k v ⊎ (leaf l<u) tr c = uncons k v l<u (c (0+ tr))
go k v ⊎ (node kl vl bll tll trl) tr c = go kl vl bll tll trl
λ { (0+ tl') → c (rotl k v tl' tr)
; (1+ tl') → c (1+ (node k v ⊎ tl' tr)) }
go k v ⊘ (node kl vl bll tll trl) tr c = go kl vl bll tll trl
λ { (0+ tl') → c (0+ (node k v ⊢ tl' tr))
; (1+ tl') → c (1+ (node k v ⊘ tl' tr)) }

widen : ∀ {lb ub ub' h v} {V : Key → Set v}
→ ub ⊑< ub'
→ Tree V lb ub h
→ Tree V lb ub' h
widen {lb} ub<ub' (leaf l<u) = leaf (⊑<-trans {lb} l<u ub<ub')
widen ub<ub' (node k v bl tl tr) = node k v bl tl (widen ub<ub' tr)

delete : ∀ {lb ub h v} {V : Key → Set v}
→ (k : Key)
→ Tree V lb ub (suc h)
→ Altered V lb ub h
delete k (node k1 v bl tl tr) with compare k k1
delete k (node {lh = zero} k1 v bl tl tr) | tri< a ⊢ b ⊢ c = 1+ (node k1 v bl tl tr)
delete k (node {lh = suc lh} k1 v bl tl tr) | tri< a ⊢ b ⊢ c with delete k tl | bl
... | 0+ tl' | ⊘ = 0+ (node k1 v ⊢ tl' tr)
... | 0+ tl' | ⊢ = 1+ (node k1 v ⊎ tl' tr)
... | 0+ tl' | ⊎ = rotl k1 v tl' tr
... | 1+ tl' | _ = 1+ (node k1 v bl tl' tr)
delete {lb} k1 (node {rh = zero} k1 v bl tl (leaf k1<ub)) | tri≈ ⊢ a refl ⊢ c with bl | tl
... | ⊘ | _ = 0+ (widen k1<ub tl)

```

```

... |  $\neg$  | leaf  $lb < k_1 = 0+$  (leaf ( $\top < \text{trans } \{lb\} \text{ } lb < k_1 \text{ } k_1 < ub$ ))
delete  $k_1$  (node { $rh = \text{succ } rh$ }  $k_1 \text{ } v \text{ } bl \text{ } tl$  (node  $k \text{ } v_1 \text{ } bl_1 \text{ } tr \text{ } tr_1$ )) |  $\text{tri} \approx \neg a \text{ refl } \neg c \text{ with } bl$  |  $\text{uncons}' k \text{ } v_1 \text{ } bl_1$ 
... |  $\swarrow$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (0+ \text{ } tr') = \text{rot}^r k' \text{ } v' \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr'$ 
... |  $\swarrow$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (1+ \text{ } tr') = 1+ \text{ } (\text{node } k' \text{ } v' \text{ } \swarrow \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr')$ 
... |  $\neg$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (0+ \text{ } tr') = 1+ \text{ } (\text{node } k' \text{ } v' \text{ } \neg \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr')$ 
... |  $\neg$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (1+ \text{ } tr') = 1+ \text{ } (\text{node } k' \text{ } v' \text{ } \neg \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr')$ 
... |  $\searrow$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (0+ \text{ } tr') = 0+ \text{ } (\text{node } k' \text{ } v' \text{ } \searrow \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr')$ 
... |  $\searrow$  |  $\text{uncons } k' \text{ } v' \text{ } l < u \text{ } (1+ \text{ } tr') = 1+ \text{ } (\text{node } k' \text{ } v' \text{ } \searrow \text{ } (\text{widen } l < u \text{ } tl) \text{ } tr')$ 
delete  $k$  (node { $rh = \text{zero}$ }  $k_1 \text{ } v \text{ } bl \text{ } tl \text{ } tr$ ) |  $\text{tri} > \neg a \neg b \text{ } c = 1+ \text{ } (\text{node } k_1 \text{ } v \text{ } bl \text{ } tl \text{ } tr)$ 
delete  $k$  (node { $rh = \text{succ } rh$ }  $k_1 \text{ } v \text{ } bl \text{ } tl \text{ } tr$ ) |  $\text{tri} > \neg a \neg b \text{ } c \text{ with delete } k \text{ } tr$  |  $bl$ 
... |  $0+ \text{ } tr'$  |  $\swarrow = \text{rot}^r k_1 \text{ } v \text{ } tl \text{ } tr'$ 
... |  $0+ \text{ } tr'$  |  $\neg = 1+ \text{ } (\text{node } k_1 \text{ } v \text{ } \swarrow \text{ } tl \text{ } tr')$ 
... |  $0+ \text{ } tr'$  |  $\searrow = 0+ \text{ } (\text{node } k_1 \text{ } v \text{ } \neg \text{ } tl \text{ } tr')$ 
... |  $1+ \text{ } tr'$  |  $\_ = 1+ \text{ } (\text{node } k_1 \text{ } v \text{ } bl \text{ } tl \text{ } tr')$ 

```

module DependantMap where

```

data Map {v} (V : Key → Set v) : Set (k  $\sqcup$  v  $\sqcup$  r) where
  tree :  $\forall \{h\} \rightarrow \text{Bounded.Tree } V \perp \top h \rightarrow \text{Map } V$ 

```

```

insertWith :  $\forall \{v\} \{V : \text{Key} \rightarrow \text{Set } v\} (k : \text{Key})$ 
  → 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 :  $\forall \{v\} \{V : \text{Key} \rightarrow \text{Set } v\} (k : \text{Key}) \rightarrow V k \rightarrow \text{Map } V \rightarrow \text{Map } V$ 
insert k v = insertWith k v const

```

```

lookup : (k : Key) →  $\forall \{v\} \{V : \text{Key} \rightarrow \text{Set } v\} \rightarrow \text{Map } V \rightarrow \text{Maybe } (V k)$ 
lookup k (tree tr) = Bounded.lookup k tr

```

module Map where

```

data Map {v} (V : Set v) : Set (k  $\sqcup$  v  $\sqcup$  r) where
  tree :  $\forall \{h\} \rightarrow \text{Bounded.Tree } (\text{const } V) \perp \top h \rightarrow \text{Map } V$ 

```

```

insertWith :  $\forall \{v\} \{V : \text{Set } v\} (k : \text{Key}) \rightarrow V \rightarrow (V \rightarrow V \rightarrow V) \rightarrow \text{Map } V \rightarrow \text{Map } V$ 
insertWith k v f (tree tr) =
  tree (proj2 (Bounded.insert k v f tr (lift tt , lift tt)))

```

```

insert :  $\forall \{v\} \{V : \text{Set } v\} (k : \text{Key}) \rightarrow V \rightarrow \text{Map } V \rightarrow \text{Map } V$ 
insert k v = insertWith k v const

```

```

lookup : (k : Key) →  $\forall \{v\} \{V : \text{Set } v\} \rightarrow \text{Map } V \rightarrow \text{Maybe } V$ 
lookup k (tree tr) = Bounded.lookup k tr

```

```

module Sets where
data ⟨Set⟩ : Set (k ⊔ r) where
  tree : ∀ {h} → Bounded.Tree (const 1) ⊥ ⊤ h → ⟨Set⟩

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

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

```