

Splay Tree

Tobias Nipkow

March 19, 2025

Abstract

Splay trees are self-adjusting binary search trees which were invented by Sleator and Tarjan [3]. This entry provides executable and verified functional splay trees as well as the related splay heaps due to Okasaki [2].

The amortized complexity of splay trees and heaps is analyzed in the AFP entry [Amortized Complexity](#).

Contents

1	Splay Tree	1
1.1	Function <i>splay</i>	2
1.2	Functional Correctness Proofs I	4
1.2.1	Verification of <i>isin</i>	4
1.2.2	Verification of <i>insert</i>	4
1.2.3	Verification of <i>delete</i>	4
1.2.4	Overall Correctness	5
1.2.5	Size lemmas	5
2	Splay Tree Implementation of Maps	6
2.1	Functional Correctness Proofs	7
2.1.1	Proofs for lookup	7
2.1.2	Proofs for update	8
2.1.3	Proofs for delete	8
2.1.4	Overall Correctness	8
3	Splay Heap	8

1 Splay Tree

```
theory Splay-Tree
imports
  HOL-Library.Tree
```

HOL–Data-Structures.Set-Specs
HOL–Data-Structures.Cmp

begin

declare sorted-wrt.simps(2)[simp del]

Splay trees were invented by Sleator and Tarjan [3].

1.1 Function *splay*

```
function splay :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree where
splay x Leaf = Leaf |
splay x (Node AB x CD) = Node AB x CD |
x < b  $\Rightarrow$  splay x (Node (Node A x B) b CD) = Node A x (Node B b CD) |
x < b  $\Rightarrow$  splay x (Node Leaf b CD) = Node Leaf b CD |
x < a  $\Rightarrow$  x < b  $\Rightarrow$  splay x (Node (Node Leaf a B) b CD) = Node Leaf a (Node B b CD) |
x < a  $\Rightarrow$  x < b  $\Rightarrow$  A  $\neq$  Leaf  $\Rightarrow$ 
splay x (Node (Node A a B) b CD) =
(case splay x A of Node A1 a' A2  $\Rightarrow$  Node A1 a' (Node A2 a (Node B b CD))) |
a < x  $\Rightarrow$  x < b  $\Rightarrow$  splay x (Node (Node A a Leaf) b CD) = Node A a (Node Leaf b CD) |
a < x  $\Rightarrow$  x < b  $\Rightarrow$  B  $\neq$  Leaf  $\Rightarrow$ 
splay x (Node (Node A a B) b CD) =
(case splay x B of Node B1 b' B2  $\Rightarrow$  Node (Node A a B1) b' (Node B2 b CD)) |
b < x  $\Rightarrow$  splay x (Node AB b (Node C x D)) = Node (Node AB b C) x D |
b < x  $\Rightarrow$  splay x (Node AB b Leaf) = Node AB b Leaf |
b < x  $\Rightarrow$  x < c  $\Rightarrow$  C  $\neq$  Leaf  $\Rightarrow$ 
splay x (Node AB b (Node C c D)) =
(case splay x C of Node C1 c' C  $\Rightarrow$  Node (Node AB b C1) c' (Node C c D)) |
b < x  $\Rightarrow$  x < c  $\Rightarrow$  splay x (Node AB b (Node Leaf c D)) = Node (Node AB b Leaf) c D |
b < x  $\Rightarrow$  c < x  $\Rightarrow$  splay x (Node AB b (Node C c Leaf)) = Node (Node AB b C) c Leaf |
a < x  $\Rightarrow$  c < x  $\Rightarrow$  D  $\neq$  Leaf  $\Rightarrow$ 
splay x (Node AB a (Node C c D)) =
(case splay x D of Node D1 d' D2  $\Rightarrow$  Node (Node AB a C) c D1) d' D2)
⟨proof⟩
```

termination splay
⟨proof⟩

```
lemma splay-code: splay x (Node AB b CD) =
(case cmp x b of
EQ  $\Rightarrow$  Node AB b CD |
LT  $\Rightarrow$  (case AB of
Leaf  $\Rightarrow$  Node AB b CD |
Node A a B  $\Rightarrow$ 
(case cmp x a of EQ  $\Rightarrow$  Node A a (Node B b CD) |
LT  $\Rightarrow$  if A = Leaf then Node A a (Node B b CD)
```

```

        else case splay x A of
          Node A1 x' A2 ⇒ Node A1 x' (Node A2 a (Node B b CD)) |
          GT ⇒ if B = Leaf then Node A a (Node B b CD)
            else case splay x B of
              Node B1 x' B2 ⇒ Node (Node A a B1) x' (Node B2 b CD)) |
          GT ⇒ (case CD of
            Leaf ⇒ Node AB b CD |
            Node C c D ⇒
              (case cmp x c of EQ ⇒ Node (Node AB b C) c D |
              LT ⇒ if C = Leaf then Node (Node AB b C) c D
                else case splay x C of
                  Node C1 x' C2 ⇒ Node (Node AB b C1) x' (Node C2 c D) |
              GT ⇒ if D=Leaf then Node (Node AB b C) c D
                else case splay x D of
                  Node D1 x' D2 ⇒ Node (Node (Node AB b C) c D1) x' D2))
    ⟨proof⟩

definition is-root :: 'a ⇒ 'a tree ⇒ bool where
is-root x t = (case t of Leaf ⇒ False | Node l a r ⇒ x = a)

definition isin t x = is-root x (splay x t)

definition empty :: 'a tree where
empty = Leaf

hide-const (open) insert

fun insert :: 'a::linorder ⇒ 'a tree ⇒ 'a tree where
insert x t =
  (if t = Leaf then Node Leaf x Leaf
   else case splay x t of
     Node l a r ⇒
       case cmp x a of
         EQ ⇒ Node l a r |
         LT ⇒ Node l x (Node Leaf a r) |
         GT ⇒ Node (Node l a Leaf) x r)

fun splay-max :: 'a tree ⇒ 'a tree where
splay-max Leaf = Leaf |
splay-max (Node A a Leaf) = Node A a Leaf |
splay-max (Node A a (Node B b CD)) =
  (if CD = Leaf then Node (Node A a B) b Leaf
   else case splay-max CD of
     Node C c D ⇒ Node (Node (Node A a B) b C) c D)

lemma splay-max-code: splay-max t = (case t of
  Leaf ⇒ t |
  Node la a ra ⇒ (case ra of

```

```

Leaf  $\Rightarrow t \mid$ 
Node  $lb\ b\ rb \Rightarrow$ 
  (if  $rb = Leaf$  then Node (Node  $la\ a\ lb$ )  $b\ rb$ 
   else case splay-max  $rb$  of
     Node  $lc\ c\ rc \Rightarrow$  Node (Node (Node  $la\ a\ lb$ )  $b\ lc$ )  $c\ rc))$ 
⟨proof⟩

definition delete :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree where
delete  $x\ t =$ 
  (if  $t = Leaf$  then Leaf
   else case splay  $x\ t$  of Node  $l\ a\ r \Rightarrow$ 
     if  $x \neq a$  then Node  $l\ a\ r$ 
     else if  $l = Leaf$  then  $r$  else case splay-max  $l$  of Node  $l'\ m\ r' \Rightarrow$  Node  $l'\ m\ r)$ 

```

1.2 Functional Correctness Proofs I

This subsection follows the automated method by Nipkow [1].

```

lemma splay-Leaf-iff[simp]: (splay  $a\ t = Leaf$ )  $=$  ( $t = Leaf$ )
⟨proof⟩

```

```

lemma splay-max-Leaf-iff[simp]: (splay-max  $t = Leaf$ )  $=$  ( $t = Leaf$ )
⟨proof⟩

```

1.2.1 Verification of *isin*

```

lemma splay-elemsD:
  splay  $x\ t = Node\ l\ a\ r \implies$  sorted(inorder  $t$ )  $\implies$ 
   $x \in set\ (inorder\ t) \longleftrightarrow x = a$ 
⟨proof⟩

```

```

lemma isin-set: sorted(inorder  $t$ )  $\implies$  isin  $t\ x = (x \in set\ (inorder\ t))$ 
⟨proof⟩

```

1.2.2 Verification of *insert*

```

lemma inorder-splay: inorder(splay  $x\ t) = inorder\ t$ 
⟨proof⟩

```

```

lemma sorted-splay:
  sorted(inorder  $t$ )  $\implies$  splay  $x\ t = Node\ l\ a\ r \implies$ 
  sorted(inorder  $l @ x \# inorder\ r$ )
⟨proof⟩

```

```

lemma inorder-insert:
  sorted(inorder  $t$ )  $\implies$  inorder(insert  $x\ t) = ins-list\ x\ (inorder\ t)$ 
⟨proof⟩

```

1.2.3 Verification of *delete*

```

lemma inorder-splay-maxD:

```

$splay\text{-max } t = \text{Node } l \ a \ r \implies \text{sorted}(\text{inorder } t) \implies$
 $\text{inorder } l @ [a] = \text{inorder } t \wedge r = \text{Leaf}$
 $\langle \text{proof} \rangle$

lemma *inorder-delete*:
 $\text{sorted}(\text{inorder } t) \implies \text{inorder}(\text{delete } x \ t) = \text{del-list } x \ (\text{inorder } t)$
 $\langle \text{proof} \rangle$

1.2.4 Overall Correctness

interpretation *splay*: Set-by-Ordered
where *empty* = *empty* **and** *isin* = *isin* **and** *insert* = *insert*
and *delete* = *delete* **and** *inorder* = *inorder* **and** *inv* = $\lambda \cdot. \text{True}$
 $\langle \text{proof} \rangle$

Corollaries:

lemma *bst-splay*: $\text{bst } t \implies \text{bst}(\text{splay } x \ t)$
 $\langle \text{proof} \rangle$

lemma *bst-insert*: $\text{bst } t \implies \text{bst}(\text{insert } x \ t)$
 $\langle \text{proof} \rangle$

lemma *bst-delete*: $\text{bst } t \implies \text{bst}(\text{delete } x \ t)$
 $\langle \text{proof} \rangle$

lemma *splay-bstL*: $\text{bst } t \implies \text{splay } a \ t = \text{Node } l \ e \ r \implies x \in \text{set-tree } l \implies x < a$
 $\langle \text{proof} \rangle$

lemma *splay-bstR*: $\text{bst } t \implies \text{splay } a \ t = \text{Node } l \ e \ r \implies x \in \text{set-tree } r \implies a < x$
 $\langle \text{proof} \rangle$

1.2.5 Size lemmas

lemma *size-splay[simp]*: $\text{size}(\text{splay } a \ t) = \text{size } t$
 $\langle \text{proof} \rangle$

lemma *size-if-splay*: $\text{splay } a \ t = \text{Node } l \ u \ r \implies \text{size } t = \text{size } l + \text{size } r + 1$
 $\langle \text{proof} \rangle$

lemma *splay-not-Leaf*: $t \neq \text{Leaf} \implies \exists l \ x \ r. \text{splay } a \ t = \text{Node } l \ x \ r$
 $\langle \text{proof} \rangle$

lemma *size-splay-max*: $\text{size}(\text{splay-max } t) = \text{size } t$
 $\langle \text{proof} \rangle$

lemma *size-if-splay-max*: $\text{splay-max } t = \text{Node } l \ u \ r \implies \text{size } t = \text{size } l + \text{size } r + 1$
 $\langle \text{proof} \rangle$

end

2 Splay Tree Implementation of Maps

```

theory Splay-Map
imports
  Splay-Tree
  HOL-Data-Structures.Map-Specs
begin

function splay :: 'a::linorder  $\Rightarrow$  ('a*'b) tree  $\Rightarrow$  ('a*'b) tree where
  splay x Leaf = Leaf |
  x = fst a  $\Longrightarrow$  splay x (Node t1 a t2) = Node t1 a t2 |
  x = fst a  $\Longrightarrow$  x < fst b  $\Longrightarrow$  splay x (Node (Node t1 a t2) b t3) = Node t1 a (Node t2 b t3) |
  x < fst a  $\Longrightarrow$  splay x (Node Leaf a t) = Node Leaf a t |
  x < fst a  $\Longrightarrow$  x < fst b  $\Longrightarrow$  splay x (Node (Node Leaf a t1) b t2) = Node Leaf a (Node t1 b t2) |
  x < fst a  $\Longrightarrow$  x < fst b  $\Longrightarrow$  t1  $\neq$  Leaf  $\Longrightarrow$ 
    splay x (Node (Node t1 a t2) b t3) =
    (case splay x t1 of Node t11 y t12  $\Rightarrow$  Node t11 y (Node t12 a (Node t2 b t3))) |
    fst a < x  $\Longrightarrow$  x < fst b  $\Longrightarrow$  splay x (Node (Node t1 a Leaf) b t2) = Node t1 a (Node Leaf b t2) |
    fst a < x  $\Longrightarrow$  x < fst b  $\Longrightarrow$  t2  $\neq$  Leaf  $\Longrightarrow$ 
    splay x (Node (Node t1 a t2) b t3) =
    (case splay x t2 of Node t21 y t22  $\Rightarrow$  Node (Node t1 a t21) y (Node t22 b t3)) |
    fst a < x  $\Longrightarrow$  x = fst b  $\Longrightarrow$  splay x (Node t1 a (Node t2 b t3)) = Node (Node t1 a t2) b t3 |
    fst a < x  $\Longrightarrow$  splay x (Node t a Leaf) = Node t a Leaf |
    fst a < x  $\Longrightarrow$  x < fst b  $\Longrightarrow$  t2  $\neq$  Leaf  $\Longrightarrow$ 
    splay x (Node t1 a (Node t2 b t3)) =
    (case splay x t2 of Node t21 y t22  $\Rightarrow$  Node (Node t1 a t21) y (Node t22 b t3)) |
    fst a < x  $\Longrightarrow$  x < fst b  $\Longrightarrow$  splay x (Node t1 a (Node Leaf b t2)) = Node (Node t1 a Leaf) b t2 |
    fst a < x  $\Longrightarrow$  fst b < x  $\Longrightarrow$  splay x (Node t1 a (Node t2 b Leaf)) = Node (Node t1 a t2) b Leaf |
    fst a < x  $\Longrightarrow$  fst b < x  $\Longrightarrow$  t3  $\neq$  Leaf  $\Longrightarrow$ 
    splay x (Node t1 a (Node t2 b t3)) =
    (case splay x t3 of Node t31 y t32  $\Rightarrow$  Node (Node t1 a t2) b t31) y t32)
  ⟨proof⟩

termination splay
  ⟨proof⟩

lemma splay-code: splay (x:::linorder) t = (case t of Leaf  $\Rightarrow$  Leaf |
  Node al a ar  $\Rightarrow$  (case cmp x (fst a) of
    EQ  $\Rightarrow$  t |
    LT  $\Rightarrow$  (case al of

```

```

 $\text{Leaf} \Rightarrow t \mid$ 
 $\text{Node } bl \ b \ br \Rightarrow (\text{case } \text{cmp } x \ (\text{fst } b) \ \text{of}$ 
 $\quad EQ \Rightarrow \text{Node } bl \ b \ (\text{Node } br \ a \ ar) \mid$ 
 $\quad LT \Rightarrow \text{if } bl = \text{Leaf} \text{ then Node } bl \ b \ (\text{Node } br \ a \ ar)$ 
 $\quad \text{else case splay } x \ bl \ \text{of}$ 
 $\quad \quad \text{Node } bll \ y \ blr \Rightarrow \text{Node } bll \ y \ (\text{Node } blr \ b \ (\text{Node } br \ a \ ar)) \mid$ 
 $\quad GT \Rightarrow \text{if } br = \text{Leaf} \text{ then Node } bl \ b \ (\text{Node } br \ a \ ar)$ 
 $\quad \text{else case splay } x \ br \ \text{of}$ 
 $\quad \quad \text{Node } brl \ y \ brr \Rightarrow \text{Node } (\text{Node } bl \ b \ brl) \ y \ (\text{Node } brr \ a \ ar))) \mid$ 
 $\quad GT \Rightarrow (\text{case } ar \ \text{of}$ 
 $\quad \quad \text{Leaf} \Rightarrow t \mid$ 
 $\quad \quad \text{Node } bl \ b \ br \Rightarrow (\text{case } \text{cmp } x \ (\text{fst } b) \ \text{of}$ 
 $\quad \quad \quad EQ \Rightarrow \text{Node } (\text{Node } al \ a \ bl) \ b \ br \mid$ 
 $\quad \quad \quad LT \Rightarrow \text{if } bl = \text{Leaf} \text{ then Node } (\text{Node } al \ a \ bl) \ b \ br$ 
 $\quad \quad \quad \text{else case splay } x \ bl \ \text{of}$ 
 $\quad \quad \quad \text{Node } bll \ y \ blr \Rightarrow \text{Node } (\text{Node } al \ a \ bll) \ y \ (\text{Node } blr \ b \ br) \mid$ 
 $\quad \quad GT \Rightarrow \text{if } br = \text{Leaf} \text{ then Node } (\text{Node } al \ a \ bl) \ b \ br$ 
 $\quad \quad \text{else case splay } x \ br \ \text{of}$ 
 $\quad \quad \quad \text{Node } bll \ y \ blr \Rightarrow \text{Node } (\text{Node } (\text{Node } al \ a \ bl) \ b \ bll) \ y \ blr)))$ 
 $\langle \text{proof} \rangle$ 

```

```

definition lookup :: ('a*'b)tree  $\Rightarrow$  'a::linorder  $\Rightarrow$  'b option where lookup t x =
  (case splay x t of Leaf  $\Rightarrow$  None | Node - (a,b) -  $\Rightarrow$  if x=a then Some b else None)

```

hide-const (open) insert

```

fun update :: 'a::linorder  $\Rightarrow$  'b  $\Rightarrow$  ('a*'b) tree  $\Rightarrow$  ('a*'b) tree where
  update x y t = (if t = Leaf  $\text{then Node Leaf } (x,y) \ Leaf$ 
  else case splay x t of
    Node l a r  $\Rightarrow$  if x = fst a  $\text{then Node } l \ (x,y) \ r$ 
    else if x < fst a  $\text{then Node } l \ (x,y) \ (\text{Node Leaf } a \ r) \ \text{else Node } (\text{Node } l \ a \ \text{Leaf}) \ (x,y) \ r$ )

```

```

definition delete :: 'a::linorder  $\Rightarrow$  ('a*'b) tree  $\Rightarrow$  ('a*'b) tree where
  delete x t = (if t = Leaf  $\text{then Leaf}$ 
  else case splay x t of Node l a r  $\Rightarrow$ 
    if x = fst a
    then if l = Leaf  $\text{then r} \ \text{else case splay-max } l \ \text{of Node } l' \ m \ r' \Rightarrow \text{Node } l' \ m \ r$ 
    else Node } l \ a \ r)

```

2.1 Functional Correctness Proofs

lemma *splay-Leaf-iff*: (*splay* x t = Leaf) = (t = Leaf)
 $\langle \text{proof} \rangle$

2.1.1 Proofs for *lookup*

lemma *splay-map-of-inorder*:
splay x t = Node l a r \Rightarrow *sorted1* (inorder t) \Rightarrow
map-of (inorder t) x = (*if* x = fst a $\text{then Some}(snd a) \ \text{else None}$)

$\langle proof \rangle$

lemma *lookup-eq*:
 $sorted1(inorder t) \implies lookup t x = map\text{-}of (inorder t) x$
 $\langle proof \rangle$

2.1.2 Proofs for update

lemma *inorder-splay*: $inorder(splay x t) = inorder t$
 $\langle proof \rangle$

lemma *sorted-splay*:
 $sorted1(inorder t) \implies splay x t = Node l a r \implies$
 $sorted(map\text{-}fst (inorder l) @ x \# map\text{-}fst (inorder r))$
 $\langle proof \rangle$

lemma *inorder-update-splay*:
 $sorted1(inorder t) \implies inorder(update x y t) = upd\text{-}list x y (inorder t)$
 $\langle proof \rangle$

2.1.3 Proofs for delete

lemma *inorder-splay-maxD*:
 $splay\text{-}max t = Node l a r \implies sorted1(inorder t) \implies$
 $inorder l @ [a] = inorder t \wedge r = Leaf$
 $\langle proof \rangle$

lemma *inorder-delete-splay*:
 $sorted1(inorder t) \implies inorder(delete x t) = del\text{-}list x (inorder t)$
 $\langle proof \rangle$

2.1.4 Overall Correctness

interpretation *Map-by-Ordered*
where *empty* = *empty* and *lookup* = *lookup* and *update* = *update*
and *delete* = *delete* and *inorder* = *inorder* and *inv* = $\lambda_. True$
 $\langle proof \rangle$

end

3 Splay Heap

theory *Splay-Heap*
imports
HOL-Library.Tree-Multiset
begin

Splay heaps were invented by Okasaki [2]. They represent priority queues by splay trees, not by heaps!

```

fun get-min :: ('a::linorder) tree  $\Rightarrow$  'a where
get-min(Node l m r) = (if l = Leaf then m else get-min l)

fun partition :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree * 'a tree where
partition p Leaf = (Leaf,Leaf) |
partition p (Node al a ar) =
  (if a  $\leq$  p then
    case ar of
      Leaf  $\Rightarrow$  (Node al a ar, Leaf) |
      Node bl b br  $\Rightarrow$ 
        if b  $\leq$  p
        then let (pl,pr) = partition p br in (Node (Node al a bl) b pl, pr)
        else let (pl,pr) = partition p bl in (Node al a pl, Node pr b br)
    else case al of
      Leaf  $\Rightarrow$  (Leaf, Node al a ar) |
      Node bl b br  $\Rightarrow$ 
        if b  $\leq$  p
        then let (pl,pr) = partition p br in (Node bl b pl, Node pr a ar)
        else let (pl,pr) = partition p bl in (pl, Node pr b (Node br a ar)))

definition insert :: 'a::linorder  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree where
insert x h = (let (l,r) = partition x h in Node l x r)

fun del-min :: 'a::linorder tree  $\Rightarrow$  'a tree where
del-min Leaf = Leaf |
del-min (Node Leaf - r) = r |
del-min (Node (Node ll a lr) b r) =
  (if ll = Leaf then Node lr b r else Node (del-min ll) a (Node lr b r))

lemma get-min-in:
  h  $\neq$  Leaf  $\implies$  get-min h  $\in$  set-tree h
  {proof}

lemma get-min-min:
   $\llbracket \text{bst-wrt } (\leq) \; h; h \neq \text{Leaf} \rrbracket \implies \forall x \in \text{set-tree } h. \; \text{get-min } h \leq x$ 
  {proof}

lemma size-partition: partition p t = (l',r')  $\implies$  size t = size l' + size r'
{proof}

lemma mset-partition:  $\llbracket \text{bst-wrt } (\leq) \; t; \text{partition } p \; t = (l',r') \rrbracket$ 
 $\implies \text{mset-tree } t = \text{mset-tree } l' + \text{mset-tree } r'$ 
{proof}

lemma set-partition:  $\llbracket \text{bst-wrt } (\leq) \; t; \text{partition } p \; t = (l',r') \rrbracket$ 
 $\implies \text{set-tree } t = \text{set-tree } l' \cup \text{set-tree } r'$ 
{proof}

```

```

lemma bst-partition:
  partition p t = (l',r')  $\implies$  bst-wrt ( $\leq$ ) t  $\implies$  bst-wrt ( $\leq$ ) (Node l' p r')
   $\langle proof \rangle$ 

lemma size-del-min[simp]: size(del-min t) = size t - 1
   $\langle proof \rangle$ 

lemma mset-del-min: mset-tree (del-min h) = mset-tree h - {# get-min h #}
   $\langle proof \rangle$ 

lemma bst-del-min: bst-wrt ( $\leq$ ) t  $\implies$  bst-wrt ( $\leq$ ) (del-min t)
   $\langle proof \rangle$ 

end

```

References

- [1] T. Nipkow. Automatic functional correctness proofs for functional search trees. In J. Blanchette and S. Merz, editors, *Interactive Theorem Proving (ITP 2016)*, volume 9807 of *LNCS*, pages 307–322. Springer, 2016.
- [2] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.
- [3] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.