Splay Tree

Tobias Nipkow
May 28, 2015

Abstract

Splay trees are self-adjusting binary search trees which were invented by Sleator and Tarjan [1]. This entry provides executable and verified functional splay trees.

The amortized complexity of splay trees is analyzed in the AFP entry Amortized Complexity.

Contents

1 Splay Tree

1.1 Function splay .......................... 2
1.2 Is-in Test ............................. 11
1.3 Function insert ........................ 11
1.4 Function splay-max .................... 11
1.5 Function delete ....................... 14

theory Splay-Tree
imports ~~/src/HOL/Library/Tree
begin

1 Splay Tree

Splay trees were invented by Sleator and Tarjan [1].

This compensates for an incompleteness of the partial order prover:

setup ⟨⟨
let

fun prp t thm = Thm.prop_of thm = t;

val eq-False-if-not = @{thm eq-False} RS iffD2

fun prove-less-False ctxt ((less as Const(\_, T)) $ r $ s) =
let val prems = Simplifier.prems_of ctxt;
    val le = Const (@{const-name less-eq}, T);
    val t = HOLogic.mk_Trueprop(le $ s $ r);
in case find-first (prp t) prems of
   NONE =>
   let val t = HOLogic.mk_Trueprop(less $ s $ r)
   in case find-first (prp t) prems of
     NONE => NONE
     | SOME thm => SOME(mk-meta-eq((thm RS @{thm less-not-sym}) RS
      eq-False-if-not))
   end
   | SOME thm => NONE
   end;

fun add-simprocs procs thy =
  map-theory-simpset (fn ctxt => ctxt
  addsimprocs (map (fn (name, raw-ts, proc) =>
    Simplifier.simproc-global thy name raw-ts proc) procs)) thy;

in
  add-simprocs [
    (less False, [(x::'a::order) < y], prove-less-False) ]
end
]]>

1.1 Function splay

fun splay :: 'a::linorder ⇒ 'a tree ⇒ 'a tree where
  splay a Leaf = Leaf |
  splay a (Node cl c cr) =
    (if a=c
      then Node cl c cr
    else if a < c
      then case cl of
        Leaf ⇒ Node cl c cr |
        Node bl b br ⇒
          (if a=b then Node bl a (Node br c cr)
            else if a < b
              then if bl = Leaf then Node bl b (Node br c cr)
                else case splay a bl of
                  Node al a' ar ⇒ Node al a' (Node ar b (Node br c cr))
                else if br = Leaf then Node bl b (Node br c cr)
                  else case splay a br of
                    Node al a' ar ⇒ Node (Node bl b al) a' (Node ar c cr))
          else case cr of
            Leaf ⇒ Node cl c cr |
            Node bl b br ⇒
              (if a=b then Node (Node cl c bl) a br
                else if a < b
                  then if bl = Leaf then Node (Node cl c bl) b br
                    else case splay a bl of
                      Node al a' ar ⇒ Node (Node cl c al) a' (Node ar b br)
                    else case splay a br of
                      Node al a' ar ⇒ Node (Node cl c al) a' (Node ar b br)
                else if cr = Leaf then Node bl b c (Node cl c cr)
                  else case splay a cr of
                    Node al a' ar ⇒ Node (Node cl c al) a' (Node ar b br) |
else if $br = \text{Leaf}$ then $\text{Node} (\text{Node} c l b l) b br$
else case splay $a b r$ of
    $\text{Node} a a a' a' \Rightarrow \text{Node} (\text{Node} c l b l) b a a' a'$

define
splay (5::int) (Node (Node A 5 B) 10 C)

value splay (5::int) (Node (Node (Node (Node A 5 B) 10 C) 15 D))

value splay (7::int) (Node L 5 (Node (Node A 6 B) 7 R))

value splay (0::int) (Node (Node (Node (Node (Node (Node A 0 B) 1 C) 2 D) 3 E) 4 F) 5 G) 6 H)

value splay (74::int) (Node (Node (Node (Node A 70 (Node (Node B 71 (Node C 72 (Node D 73 (Node E 74 F)))))) 79 G)) 80 H) 90 I) 100 J

value splay (6::int) (Node (Node A 0 (Node (Node B 1 (Node (Node C 2 (Node D 6 E)) 7 F)) 8 G)) 9 H)

value splay (70::int) (Node L 50 (Node A 60 (Node (Node B 70 C) 90 D)))

lemma splay-simps(simp):
splay a (Node l a r) = Node l a r

$a < b \Rightarrow splay a (Node (Node ll a lr) b r) = Node ll a (Node lr b r)$

$a < b \Rightarrow splay a (Node Leaf b r) = Node Leaf b r$

$a < c \Rightarrow a < b \Rightarrow splay a (Node (Node Leaf b lr) c r) = Node Leaf b (Node lr c r)$

$a < c \Rightarrow a < b \Rightarrow splay a (Node l c) = Node l c$  

$b < a \Rightarrow splay a (Node ll b lr) c r) = Node ll b (Node lr c r)$

$a < c \Rightarrow b < a \Rightarrow splay a (Node (Node ll b Leaf) c r) = Node ll b (Node Leaf c r)$

$a < c \Rightarrow b < a \Rightarrow splay a (Node l c) = Node l c$

$b < a \Rightarrow splay a (Node ll b lr) c r) = Node ll b (Node lr c r)$

$c < a \Rightarrow a < b \Rightarrow splay a (Node l c (Node Leaf b rr)) = Node (Node l c Leaf) b rrr$

$c < a \Rightarrow b < a \Rightarrow splay a (Node l c (Node rl b rr)) = Node (Node l c rl) b Leaf$

by auto

declare splay.simps(2)[simp del]

lemma splay-Leaf-iff[simp]: (splay a t = Leaf) = (t = Leaf)

apply(induction a t rule: splay.induct)
apply(simp)
apply(subst splay.simps(2))
apply(auto)
apply(auto split: tree.splits if-splits)
done

lemma size-splay[simp]: size (splay a t) = size t
apply(induction a t rule: splay.induct)
apply(simp)
apply(subst splay.simps)
apply(auto)
apply(force split: tree.split)+
done

lemma size-if-splay: splay a t = Node l u r \Rightarrow size t = size l + size r + 1
by (metis One-nat-def size-splay tree.size(4))

lemma set-splay: set-tree(splay a t) = set-tree t
proof(induction a t rule: splay.induct)
case 1 thus ?case by simp
next
case (2 a l b r)
show ?case
proof cases
  assume a=b thus ?thesis by simp
next
  assume a\ne b
  hence a\lt\lt b \lor b\lt\lt a by (metis neqE)
  thus ?thesis
proof
  assume a\lt b
  show ?thesis
  proof(cases l)
    case Leaf thus ?thesis using (a\lt b) by simp
  next
    case (Node ll c lr)[simp]
    show ?thesis
    proof cases
      assume a=c thus ?thesis using (a\lt b) by auto
  next
    assume a\ne c
    hence a\lt c \lor c\lt a by (metis neqE)
    thus ?thesis
    proof
      assume a\lt c
      show ?thesis
      proof cases
        assume ll = Leaf thus ?thesis using (a\lt b) (a\lt c) by auto
  next

assume \( ll \neq Leaf \)

hence \( splay a \) \( ll \neq Leaf \) by simp

then obtain \( ll u ll r \) where [simp]: \( splay a \) \( ll = Node ll u ll r \)
    by (metis tree.exhaust)

from 2.IH(1)[OF \( \langle a\neq b \rangle \langle a\neq c \rangle \langle a < c \rangle \langle ll \neq Leaf \rangle \)]

show \( ?thesis \) using \( \langle a < b \rangle \langle a < c \rangle \) by auto

qed

next

assume \( c < a \) hence \( \neg a < c \) by simp

show \( ?thesis \)

proof cases

  assume \( br = Leaf \) thus \( ?thesis \) using \( \langle a < b \rangle \langle c < a \rangle \) by(auto)

next

assume \( br \neq Leaf \)

hence \( splay a \) \( br \neq Leaf \) by simp

then obtain \( lrr u lrr \) where [simp]: \( splay a \) \( br = Node lrr u lrr \)
    by (metis tree.exhaust)

from 2.IH(2)[OF \( \langle a\neq b \rangle \langle a < b \rangle \langle a < c \rangle \langle br \neq Leaf \rangle \)]

show \( ?thesis \) using \( \langle a < b \rangle \langle c < a \rangle \) by auto

qed

qed

qed

next

assume \( b < a \) hence \( \neg a < b \) by simp

show \( ?thesis \)

proof(cases \( r \))

  case Leaf thus \( ?thesis \) using \( \langle b < a \rangle \) by(auto)

next

  case (Node rl c rr)[simp]

  show \( ?thesis \)

  proof cases

    assume \( a = c \) thus \( ?thesis \) using \( \langle b < a \rangle \) by auto

next

    assume \( a \neq c \)

    hence \( a < c \lor c < a \) by (metis neqE)

    thus \( ?thesis \)

    proof

      assume \( a < c \) hence \( \neg c < a \) by simp

      show \( ?thesis \)

      proof cases

        assume \( rl = Leaf \) thus \( ?thesis \) using \( \langle b < a \rangle \langle a < c \rangle \) by(auto)

next

        assume \( rl \neq Leaf \)

        hence \( splay a \) \( rl \neq Leaf \) by simp

        then obtain \( lrr u lrr \) where [simp]: \( splay a \) \( rl = Node lrr u lrr \)
            by (metis tree.exhaust)

        from 2.IH(3)[OF \( \langle a\neq b \rangle \langle \neg a < b \rangle \langle a < c \rangle \langle rl \neq Leaf \rangle \)]

        show \( ?thesis \) using \( \langle b < a \rangle \langle a < c \rangle \) by auto
qed
next
assume c < a hence ¬ a < c by simp
show ?thesis
proof cases
  assume rr = Leaf thus ?thesis using (b < a) (c < a) by (auto)
next
assume rr ≠ Leaf
hence splay a rr ≠ Leaf by simp
then obtain rrl u rrr where [simp]: splay a rr = Node rrl u rrr
by (metis tree.exhaust)
from 2.IH(4)[OF (a ≠ b) (¬ a < b) Node (a ≠ c) (¬ a < c) (rr ≠ Leaf)]
show ?thesis using (b < a) (c < a) by auto
qed
qed
qed
qed
qed
qed
qed
qed

case

lemma splay-bstL: bst t ⇒ splay a t = Node l e r ⇒ x ∈ set-tree l ⇒ x < a
apply (induction a t arbitrary: l x r rule: splay.induct)
  apply simp
apply (subst (asm) splay.simps)
apply (auto split: if-splits tree.splits)
apply auto
done

lemma splay-bstR: bst t ⇒ splay a t = Node l e r ⇒ x ∈ set-tree r ⇒ a < x
apply (induction a t arbitrary: l e x r rule: splay.induct)
  apply simp
apply (subst (asm) splay.simps)
using [[simp-depth-limit = 3]] apply (fastforce split: if-splits tree.splits)
done

lemma bst-splay: bst t ⇒ bst(splay a t)
proof (induction a t rule: splay.induct)
  case 1 show ?case by simp
next
  case (2 a l b r)
  show ?case
  proof cases
    assume a = b thus ?thesis using 2.prems by auto
  next
    assume a ≠ b
    hence a < b ∨ b < a by (metis neqE)
    thus ?thesis
    proof
assume \( a \lt b \)

show \(?thesis\)

proof (cases \( l \))
  
  case Leaf thus \(?thesis\) using 2.prems \( a \lt b \) by (auto)

next
  
  case (Node ll c lr)[simp]
  
  have \( c \lt b \) using 2.prems by (auto)
  
  show \(?thesis\)
  
  proof cases
    
    assume \( a = c \) thus \(?thesis\) using 2.prems \( a \lt b \) by auto

next
  
  assume \( a \neq c \)
  
  hence \( a \lt c \lor c \lt a \) by (metis neqE)
  
  thus \(?thesis\)
  
  proof
    
    assume \( a \lt c \)
    
    show \(?thesis\)
    
    proof cases
      
      assume \( ll = \) Leaf thus \(?thesis\) using \( a \lt b \) \( a \lt c \) \( c \lt b \) 2.prems by (auto)

next
  
  assume \( ll \neq \) Leaf
  
  hence splay a \( ll \neq \) Leaf by simp
  
  then obtain \( ll l u llr \) where [simp]: \( \text{splay a ll = Node ll l u llr} \)
  
  by (metis tree.exhaust)
  
  have \( \text{bst ll} \) using 2.prems by simp
  
  from 2.IH(1)(OF \( a \neq b \) \( a \lt b \) Node \( a \neq c \) \( a \lt c \) \( ll \neq \) Leaf) \( \text{bst ll} \)

  show \(?thesis\) using \( a \lt b \) \( a \lt c \) \( c \lt b \) 2.prems set-splay[of a ll]
  
  by auto (metis insertI1 less-trans)+

qed

next
  
  assume \( c \lt a \) hence \( \neg a \lt c \) by simp
  
  show \(?thesis\)
  
  proof cases
    
    assume \( lr = \) Leaf thus \(?thesis\) using \( a \lt b \) \( c \lt a \) \( c \lt b \) 2.prems by (auto)

next
  
  assume \( lr \neq \) Leaf
  
  hence splay a \( lr \neq \) Leaf by simp
  
  then obtain \( lr l u lrr \) where [simp]: \( \text{splay a lr = Node lr l u lrr} \)
  
  by (metis tree.exhaust)
  
  have \( \text{bst lr} \) using 2.prems by simp
  
  from 2.IH(2)(OF \( a \neq b \) \( a \lt b \) Node \( a \neq c \) \( \neg a \lt c \) \( lr \neq \) Leaf)

  show \(?thesis\) using \( a \lt b \) \( c \lt a \) \( c \lt b \) 2.prems set-splay[of a lr]
  
  by auto (metis Un iff insertI1 less-trans)+

qed

qed

qed

next
assume \( a > b \) hence \( \neg a < b \) by simp
show \(?thesis\)
proof (cases \( r \))
  case Leaf thus \(?thesis\) using \( 2 \).prems \( \langle a > b \rangle \) by (auto)
next
  case \( \text{Node } rl \ c \ rr \) [simp]
  have \( c > b \) using \( 2 \).prems by (auto)
  show \(?thesis\)
  proof cases
    assume \( a = c \) thus \(?thesis\) using \( 2 \).prems \( \langle a > b \rangle \) [simp]
  qed
next
  assume \( a \neq c \)
  hence \( a < c \lor c < a \) by (metis neqE)
  thus \(?thesis\)
  proof cases
    assume \( a < c \)
    show \(?thesis\)
    proof cases
      case Leaf thus \(?thesis\) using \( 2 \).prems \( \langle a > b \rangle \) \( \langle a < c \rangle \) \( \langle c > b \rangle \) by (auto)
      qed
      next
        assume \( rl = \text{Leaf} \) thus \(?thesis\) using \( 2 \).prems \( \langle a > b \rangle \) \( \langle a < c \rangle \) \( \langle c > b \rangle \) \( 2 \).prems
        by (auto)
      qed
      qed
      qed
      qed
      qed
      qed
next
  assume \( rl \neq \text{Leaf} \)
  hence \( \text{splay } a \ rl \neq \text{Leaf} \) by simp
  then obtain \( rll \ u \ rlr \) where [simp]: \( \text{splay } a \ rl = \text{Node } rll \ u \ rlr \)
  by (metis tree.exhaust)
  have \( \text{bst } rl \) using \( 2 \).prems by simp
  from \( 2 \).IH \( (3) \)[OF \( \langle a \neq b \rangle \ \langle \neg a < b \rangle \ \text{Node } \langle a \neq c \rangle \ \langle a < c \rangle \ \langle rl \neq \text{Leaf} \rangle \ \langle \text{bst } rl \rangle \] show \(?thesis\) using \( a > b \) \( a < c \) \( c > b \) \( 2 \).prems \( \text{set-splay[of } a \ rl \] by auto (metis Un_iff insertI1 less_trans)+
  qed
next
  assume \( c < a \) hence \( \neg a < c \) by simp
  show \(?thesis\)
  proof cases
    assume \( rr = \text{Leaf} \) thus \(?thesis\) using \( a > b \) \( a < c \) \( 2 \).prems by (auto)
    qed
    next
      assume \( rr \neq \text{Leaf} \)
      hence \( \text{splay } a \ rr \neq \text{Leaf} \) by simp
      then obtain \( rrl \ u \ rrr \) where [simp]: \( \text{splay } a \ rr = \text{Node } rrl \ u \ rrr \)
      by (metis tree.exhaust)
      have \( \text{bst } rr \) using \( 2 \).prems by simp
      from \( 2 \).IH \( (4) \)[OF \( \langle a \neq b \rangle \ \langle \neg a < b \rangle \ \text{Node } \langle a \neq c \rangle \ \langle \neg a < c \rangle \ \langle rr \neq \text{Leaf} \rangle \] show \(?thesis\) using \( a > b \) \( c < a \) \( c > b \) \( 2 \).prems \( \text{set-splay[of } a \ rr \] by auto (metis insertI1 less_trans)+
      qed
      qed
      qed
      qed
    qed
  qed
qed
```text
lemma splay-to-root: \( \text{bst } t; \ splay \ a \ t = t' \) \implies \( a \in \text{set-tree } t \iff (\exists \ l \ r. \ t' = \text{Node } l \ a \ r) \)
proof(induction a t arbitrary; t' rule: splay.induct)
case 1 thus \(?case by simp
next
case \( 2 \ a \ l \ b \ r \)
show \(?case
proof\ cases
  assume \( a = b \) thus \(?thesis using 2.prems by auto
next
  assume \( a \neq b \)
hence \( a < b \vee b < a \) by (metis neqE)
thus \(?thesis
proof
  assume \( a < b \)
show \(?thesis
proof\(\) cases \( l \)
case Leaf thus \(?thesis using \( a < b \) 2.prems by fastforce
next
case \( \text{Node } ll \ c \ lr \)[simp]
show \(?thesis
proof\ cases
  assume \( a = c \) thus \(?thesis using \( a < b \) 2.prems by auto
next
  assume \( a \neq c \)
hence \( a < c \vee c < a \) by (metis neqE)
thus \(?thesis
proof
  assume \( c < a \)
show \(?thesis
proof\ cases
    assume \( ll = \text{Leaf} \) thus \(?thesis using \( a < b \) \( a < c \) 2.prems by auto
next
    assume \( ll \neq \text{Leaf} \)
hence \( \text{splay } a \ l l \neq \text{Leaf} \) by simp
then obtain \( l l l \ a \ ll r \) where [simp]: \( \text{splay } a \ ll = \text{Node } l l l \ a \ ll r \)
by (metis tree.exhaust)
from 2.IH(1)[OF \( a \neq b \) \( a < b \) Node \( a \neq c \) \( a < c \) \( ll \neq \text{Leaf} \)]
show \(?thesis using \( a < b \) \( a < c \) 2.prems by auto
qed
next
  assume \( c < a \) hence \( \neg a < c \) by simp
show \(?thesis
proof\ cases
    assume \( lr = \text{Leaf} \) thus \(?thesis using \( a < b \) \( c < a \) 2.prems by(_auto
next
```

9
assume \( lr \neq \text{Leaf} \)

then obtain \( lr l u lrr \) where [simp]: \( \text{splay } lr = \text{Node } lr l u lrr \)

by \((\text{metis tree.exhaust})\)

from \(2.IH \(2\)[OF \( a\neq b \) \( a\lt b \) \( \text{Node } (a\neq c) \) \( \neg a\lt c \) \( (lr \neq \text{Leaf}) \)]\)

show \?thesis using \( a\lt b \) \( c\lt a \) \( 2\).prems by auto

qed

next

assume \( b\lt a \) hence \( \neg a\lt b \) by simp

show \?thesis

proof (cases \( r \))

  case \( \text{Leaf} \)

  thus \?thesis using \( b\lt a \) \( \neg a\lt b \) \( 2\).prems by fastforce

next

  case \( \text{Node } rl c rrr \)[simp]

  show \?thesis

  proof cases

    assume \( a\neq c \)

    hence \( a\lt c \lor c\lt a \) by \((\text{metis neqE})\)

    thus \?thesis

    proof

      assume \( a\lt c \) hence \( \neg c\lt a \) by simp

      show \?thesis

      proof cases

        assume \( rl = \text{Leaf} \)

        thus \?thesis using \( b\lt a \) \( a\lt c \) \( 2\).prems by(auto)

      next

        assume \( rl \neq \text{Leaf} \)

        hence \( \text{splay } rl \neq \text{Leaf} \) by simp

        then obtain \( rll u rl r \) where [simp]: \( \text{splay } rl = \text{Node } rll u rl r \)

        by \((\text{metis tree.exhaust})\)

        from \(2.IH \(3\)[OF \( a\neq b \) \( \neg a\lt b \) \( \text{Node } (a\neq c) \) \( c\lt a \) \( rl \neq \text{Leaf}) \)]\)

        show \?thesis using \( b\lt a \) \( c\lt a \) \( 2\).prems by auto

    qed

next

assume \( c\lt a \) hence \( \neg a\lt c \) by simp

show \?thesis

proof cases

  assume \( rr = \text{Leaf} \)

  thus \?thesis using \( b\lt a \) \( c\lt a \) \( 2\).prems by(auto)

next

  assume \( rr \neq \text{Leaf} \)

  hence \( \text{splay } rr \neq \text{Leaf} \) by simp

  then obtain \( rrl u rrr r \) where [simp]: \( \text{splay } rr = \text{Node } rrl u rrr r \)

  by \((\text{metis tree.exhaust})\)

  from \(2.IH \(4\)[OF \( a\neq b \) \( \neg a\lt b \) \( \text{Node } (a\neq c) \) \( c\lt a \) \( (rr \neq \text{Leaf}) \)]\)

  show \?thesis using \( b\lt a \) \( c\lt a \) \( 2\).prems by auto
1.2 Is-in Test

To test if an element \( a \) is in \( t \), first perform \( \text{splay} \ a \ t \), then check if the root is \( a \). One could put this into one function that returns both a new tree and the test result.

**Definition** is-root :: \( 'a \Rightarrow \text{tree} \Rightarrow \text{bool} \)

\[
\text{is-root} \ a \ t = (\text{case } t \text{ of Leaf } \Rightarrow \text{False} \mid \text{Node } x \cdot x \Rightarrow x = a)
\]

**Lemma** is-root-splay: \( \text{bst} \ t \implies \text{is-root} \ a \ (\text{splay} \ a \ t) \iff a \in \text{set-tree} \ t \)

\[
\text{by}(\text{auto simp add: is-root-def splay-to-root split: tree.split})
\]

1.3 Function insert

**Function** insert :: \( 'a::\text{linorder} \Rightarrow \text{tree} \Rightarrow \text{tree} \)

\[
\text{insert} \ a \ t = \begin{cases} \text{Leaf} & \text{if } t = \text{Leaf} \text{ then Node Leaf } a \text{ Leaf} \\ \text{case } \text{splay} \ a \ t \text{ of} \\ \text{Node } l \ a' \ r & \text{if } a = a' \text{ then Node } l \ a \ r \\ \text{else if } a < a' \text{ then Node } l \ a \ (\text{Node Leaf } a' \ r) \text{ else Node } (\text{Node } l \ a' \ \text{Leaf}) \ a \ r \\ \end{cases}
\]

\[
\text{hide-const (open) Splay-Tree.insert}
\]

**Lemma** set-insert: \( \text{set-tree}(\text{Splay-Tree.insert} \ a \ t) = \text{insert} \ a \ (\text{set-tree} \ t) \)

\[
\text{apply(cases } t) \\ \text{apply simp} \\ \text{using set-splay[of } a \ t] \\ \text{by(simp split: tree.split) fastforce}
\]

**Lemma** bst-insert: \( \text{bst} \ t \implies \text{bst}(\text{Splay-Tree.insert} \ a \ t) \)

\[
\text{apply(cases } t) \\ \text{apply simp} \\ \text{using bst-splay[of } t \ a] \text{ splay-bstL[of } t \ a] \text{ splay-bstR[of } t \ a]} \\ \text{by(auto simp: ball-Un split: tree.split})
\]

1.4 Function splay-max

**Function** splay-max :: \( 'a::\text{linorder} \Rightarrow \text{tree} \Rightarrow \text{tree} \)

\[
\text{splay-max} \ \text{Leaf} = \text{Leaf} \\ \text{splay-max} \ (\text{Node } l \ b \ \text{Leaf}) = \text{Node } l \ b \ \text{Leaf} \\ \text{splay-max} \ (\text{Node } l \ b \ (\text{Node } r l \ c \ rr)) = \\ \begin{cases} \text{Leaf} & \text{if } rr = \text{Leaf} \text{ then Node } (\text{Node } l \ b \ r l) \ c \ \text{Leaf} \\ \text{else case } \text{splay-max} \ rr \text{ of} \\ \end{cases}
\]

\[
\text{by(auto simp: ball-Un split: tree.split})
\]

11
Node rrl m rrr ⇒ Node (Node (Node l b rl) c rrl) m rrr

lemma splay-max-Leaf-iff [simp]; (splay-max t = Leaf) = (t = Leaf)
apply (induction t rule: splay-max.induct)
apply (auto split: tree.splits)
done

lemma splay-max-code: splay-max t = (case t of
  Leaf ⇒ t |
  Node l b r ⇒ (case r of
               Leaf ⇒ t |
               Node rl c rr ⇒ (if rr = Leaf then Node (Node l b rl) c rr
e else case splay-max rr of
               Node rrl u rrr ⇒ Node (Node (Node l b rl) c rrl) u rrr)))))
by (auto simp: neq-Leaf-iff split: tree.splits)

lemma size-splay-max: size (splay-max t) = size t
apply (induction t rule: splay-max.induct)
apply (simp)
apply (simp)
apply (clarsimp split: tree.split)
done

lemma size-if-splay-max: splay-max t = Node l u r ⇒ size t = size l + size r + 1
by (metis One-nat-def size-splay-max tree.size 4)

lemma set-splay-max: set-tree (splay-max t) = set-tree t
apply (induction t rule: splay-max.induct)
apply (simp)
apply (simp)
apply (clarsimp split: tree.split)
done

lemma bst-splay-max: bst t ⇒ bst (splay-max t)
proof (induction t rule: splay-max.induct)
case (Node l b rl c rr)
{ fix rrl' d' rrr'
  have splay-max rr = Node rrl' d' rrr'
    ⇒ ∀x : set-tree (Node rrl' d' rrr'). c < x
    using 3.prems set-splay-max[of rr]
    by (clarsimp split: tree.split simp: ball-Un)
  }
with 3 show ?case by (fastforce split: tree.split simp: ball-Un)
qed auto

lemma splay-max-Leaf: splay-max t = Node l a r ⇒ r = Leaf
by (induction t arbitrary: l rule: splay-max.induct)
For sanity purposes only:

**lemma** splay-max-eq-splay:

\[ \text{bst } t \implies \forall x \in \text{set-tree } t. \ x \leq a \implies \text{splay-max } t = \text{splay } a \ t \]

**proof** (induction \( a \ t \) rule: splay.induct)

**case** \( 1 \) **thus** ?case by simp

**next**

**case** \( (2 \ a \ l \ b \ r) \)

**have** \( b = a \lor b < a \) **using** 2.prems(2) by auto

**thus** ?case

**proof**

**assume** [simp]: \( b = a \)

**have** \( r = \text{Leaf} \)

**apply** (rule ccontr)

**using** 2.prems by (auto simp: ball-Un neq-Leaf-iff)

**thus** ?thesis by simp

**next**

**assume** \( b < a \)

**hence** \( a \neq b \) by simp

**show** ?thesis

**proof** (cases \( r \))

**case** \( \text{Leaf} \) **thus** ?thesis using \( (b < a) \) by(auto)

**next**

**case** \( (\text{Node } rl \ c \ rr)[simp] \)

**have** \( c = a \lor c < a \) **using** 2.prems by auto

**thus** ?thesis

**proof**

**assume** [simp]: \( c = a \)

**have** \( rr = \text{Leaf} \)

**apply** (rule ccontr)

**using** 2.prems by (auto simp: ball-Un neq-Leaf-iff)

**thus** ?thesis using \( (b < a) \) by simp

**next**

**assume** \( c < a \)

**hence** \( a \neq c \) by simp

**show** ?thesis

**proof** cases

**assume** \( rr = \text{Leaf} \) **thus** ?thesis using \( (b < a) \) \( (c < a) \) by(auto)

**next**

**assume** \( rr \neq \text{Leaf} \)

**then** obtain \( rrl \ a \ rrr \) where [simp]: \( rr = \text{Node } rrl \ a \ rrr \)

**by** (auto simp: neq-Leaf-iff)

**hence** \( \text{splay } a \ rr \neq \text{Leaf} \) by simp

**then** obtain \( rrl' \ a' rrr' \) **where** [simp]: \( \text{splay } a \ rr = \text{Node } rrl' \ a' rrr' \)

**by** (metis tree.exhaust)

**from** 2.IH(4)[OF \( (a \neq b) \ - \text{Node } (a \neq c) \ - \rr \neq \text{Leaf} \)] 2.prems

**show** ?thesis using \( (b < a) \) \( (c < a) \) by (auto split: tree.split)

**qed**
lemma splay-max-eq-splay-ex: assumes bst t shows ∃a. splay-max t = splay a t
proof (cases t)
  case Leaf thus ?thesis by simp
next
  case Node
  hence splay-max t = splay (Max (set-tree t)) t
    using assms (auto simp: splay-max-eq-splay)
  thus ?thesis by auto
qed

1.5 Function delete

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

hide-const (open) delete

lemma set-delete: assumes bst t
  shows set-tree (Splay-Tree.delete a t) = set-tree t − {a}
proof (cases t)
  case Leaf thus ?thesis by (simp add: delete-def)
next
  case (Node l x r)
  obtain l' x' r' where sp[simp]: splay a (Node l x r) = Node l' x' r'
    by (metis neq-Leaf-iff splay-Leaf-iff)
  show ?thesis
proof cases
  assume [simp]: x' = a
  show ?thesis
proof cases
  assume l' = Leaf
  thus ?thesis
  using Node assms set-splay[of a Node l x r] bst-splay[of Node l x r a]
    by (simp add: delete-def split: tree.split prod.split) (fastforce)
next
  assume l' ≠ Leaf
  moreover then obtain l'' m r'' where splay-max l' = Node l'' m r''
    using splay-max-Leaf-iff tree.exhaust by blast
  moreover have a ∉ set-tree l'
    by (metis (no-types) Node assms less_irrefl sp splay-bstL)
ultimately show \( \text{thesis} \)
using Node assms set-splay[of a Node l x r] bst-splay[of Node l x r a]
\( \text{splay-max-Leaf[of l' l'' m r']} \text{ set-splay-max[of l']} \)
by(clarsimp simp: delete-def split: tree.split) auto
qed
next
assume x' \( \neq \) a
thus \( \text{thesis} \)
using Node assms splay[of a Node l x r]
\( \text{splay-to-root[OF - sp]} \)
by (simp add: delete-def)
qed
qed

lemma bst-delete: assumes bst t shows bst (Splay-Tree.delete a t)
proof(cases t)
case Leaf thus \( \text{thesis} \)
by(simp add: delete-def)
next
case (Node l x r)
obtain l' x' r' where sp[simp]: splay a (Node l x r) = Node l' x' r'
by (metis neq-Leaf-iff splay-Leaf-iff)
show \( \text{thesis} \)
proof cases
assume x' = a
show \( \text{thesis} \)
proof cases
assume l' = Leaf
thus \( \text{thesis} \)
using Node assms bst-splay[of Node l x r a]
by(simp add: delete-def split: tree.split prod.split)
next
assume l' \( \neq \) Leaf
thus \( \text{thesis} \)
using Node assms bst-splay[of a Node l x r] bst-splay[of Node l x r a]
\( \text{bst-splay-max[of l'] set-splay-max[of l']} \)
by(clarsimp simp: delete-def split: tree.split)
\( \text{metis (no-types) insertI1 less-trans} \)
qed
next
assume x' \( \neq \) a
thus \( \text{thesis} \)
using Node assms bst-splay[of a Node l x r]
by(auto simp: delete-def split: tree.split prod.split)
qed
qed

end

References