BinarySearchTree
Larry Paulson
August 16, 2018

Contents

1 Isar-style Reasoning for Binary Tree Operations 2
2 Tree Definition 2
3 Tree Lookup 3
   3.1 Tree membership as a special case of lookup . . . . . . . 5
4 Insertion into a Tree 6
5 Removing an element from a tree 9
6 Mostly Isar-style Reasoning for Binary Tree Operations 18
7 Map implementation and an abstraction function 18
8 Auxiliary Properties of our Implementation 18
   8.1 Lemmas mapset-none and mapset-some establish a relation
       between the set and map abstraction of the tree . . . . . . . 19
9 Empty Map 20
10 Map Update Operation 21
11 Map Remove Operation 22
12 Tactic-Style Reasoning for Binary Tree Operations 23
13 Definition of a sorted binary tree 23
14 Tree Membership 23
15 Insertion operation 24
16 Remove operation 24
1 Isar-style Reasoning for Binary Tree Operations

theory BinaryTree imports Main begin

We prove correctness of operations on binary search tree implementing a set.
This document is LGPL.
Author: Viktor Kuncak, MIT CSAIL, November 2003

2 Tree Definition

datatype 'a Tree = Tip | T 'a Tree 'a 'a Tree

primrec 
  setOf :: 'a Tree => 'a set 
  — set abstraction of a tree
where 
  setOf Tip = {}
  | setOf (T t1 x t2) = (setOf t1) Un (setOf t2) Un {x}

type-synonym 
  — we require index to have an irreflexive total order \( i \)
  — apart from that, we do not rely on index being int
  index = int

type-synonym — hash function type
  'a hash = 'a => index

definition eqs :: 'a hash => 'a => 'a set 
  — equivalence class of elements with the same hash code
  eqs h x == { y. h y = h x}

primrec 
  sortedTree :: 'a hash => 'a Tree => bool 
  — check if a tree is sorted
where 
  sortedTree h Tip = True
  | sortedTree h (T t1 x t2) = 
    (sortedTree h t1 &
     (\forall l \in setOf t1. h l < h x) &
     (\forall r \in setOf t2. h x < h r) &
     sortedTree h t2)

lemma sortLemmaL: 
  sortedTree h (T t1 x t2) ==> sortedTree h t1 by simp
lemma sortLemmaR: 
  sortedTree h (T t1 x t2) ==> sortedTree h t2 by simp
3 Tree Lookup

primrec
tlookup :: 'a hash => index => 'a Tree => 'a option

where
tlookup h k Tip = None
| tlookup h k (T t1 x t2) =
  (if k < h x then tlookup h k t1
   else if h x < k then tlookup h k t2
   else Some x)

lemma tlookup-none:
  sortedTree h t & (tlookup h k t = None) --> (∀ x∈setOf t. h x ~= k)
by (induct t, auto)

lemma tlookup-some:
  sortedTree h t & (tlookup h k t = Some x) --> x∈setOf t & h x = k
apply (induct t)
  — Just auto will do it, but very slowly
apply (simp)
apply (clarify, auto)
apply (simp-all split: if-split-asm)
done

definition sorted-distinct-pred :: 'a hash => 'a => 'a => 'a Tree => bool

where
  sorted-distinct-pred h a b t == sortedTree h t &:
  a∈setOf t & b∈setOf t & h a = h b -->
  a = b

declare sorted-distinct-pred-def [simp]

— for case analysis on three cases
lemma cases3: [ | C1 ==> G; C2 ==> G; C3 ==> G; C1 | C2 | C3 |] ==> G
by auto

sorted-distinct-pred holds for out trees:

lemma sorted-distinct: sorted-distinct-pred h a b t (is ?P t)
proof (induct t)
  show ?P Tip by simp
  fix t1 :: 'a Tree assume h1: ?P t1
  fix t2 :: 'a Tree assume h2: ?P t2
  fix x :: 'a
  show ?P (T t1 x t2)
  proof (unfold sorted-distinct-pred-def, safe)
    assume s: sortedTree h (T t1 x t2)
    assume adef: a : setOf (T t1 x t2)
    assume bdef: b : setOf (T t1 x t2)
assume \( hab : h \ a = h \ b \)
from \( s \) have \( s1 : \text{sortedTree} \ h \ t1 \) by auto
from \( s \) have \( s2 : \text{sortedTree} \ h \ t2 \) by auto
show \( a = b \)
— We consider 9 cases for the position of \( a \) and \( b \) are in the tree
proof
— three cases for \( a \)
from \( adef \) have \( a : \text{setOf} \ t1 \ | \ a = x | \ a : \text{setOf} \ t2 \) by auto
moreover \{ assume \( adef1 : a : \text{setOf} \ t1 \)
    have \( ?\text{thesis} \)
    proof
    — three cases for \( b \)
    from \( bdef \) have \( b : \text{setOf} \ t1 \ | \ b = x | \ b : \text{setOf} \ t2 \) by auto
    moreover \{ assume \( bdef1 : b : \text{setOf} \ t1 \)
        from \( s1 \) \( adef1 \) \( bdef1 \) \( hab \) \( h1 \) have \( ?\text{thesis} \) by simp \}
    moreover \{ assume \( bdef1 : b = x \)
        from \( adef1 \) \( bdef1 \) \( s \) have \( h \ a < h \ b \) by auto
        from this \( hab \) have \( ?\text{thesis} \) by simp \}
    moreover \{ assume \( bdef1 : b : \text{setOf} \ t2 \)
        from \( adef1 \) \( s \) have \( o1 : h \ a < h \ x \) by auto
        from \( bdef1 \) \( s \) have \( o2 : h \ x < h \ b \) by auto
        from \( o1 \) \( o2 \) have \( h \ a < h \ b \) by simp
        from this \( hab \) have \( ?\text{thesis} \) by simp \} — case impossible
ultimately show \( ?\text{thesis} \) by blast
qed

moreover \{ assume \( adef1 : a = x \)
    have \( ?\text{thesis} \)
    proof
    — three cases for \( b \)
    from \( bdef \) have \( b : \text{setOf} \ t1 \ | \ b = x | \ b : \text{setOf} \ t2 \) by auto
    moreover \{ assume \( bdef1 : b : \text{setOf} \ t1 \)
        from this \( s \) have \( h \ b < h \ x \) by auto
        from \( this1 \) \( adef1 \) \( h a h b \) have \( ?\text{thesis} \) by simp \}
    moreover \{ assume \( bdef1 : b = x \)
        from this \( hab \) this have \( ?\text{thesis} \) by simp \}
    moreover \{ assume \( bdef1 : b : \text{setOf} \ t2 \)
        from \( adef1 \) \( s \) have \( h \ x < h \ b \) by auto
        from this \( h a h b \) this have \( ?\text{thesis} \) by simp \}
    moreover \{ assume \( bdef1 : b = x \)
        from this \( s \) have \( h \ a < h \ b \) by auto
        from \( this \) \( adef1 \) \( h a h b \) have \( ?\text{thesis} \) by simp \}
    ultimately show \( ?\text{thesis} \) by blast
qed

moreover \{ assume \( adef1 : a : \text{setOf} \ t2 \)
    have \( ?\text{thesis} \)
    proof
    — three cases for \( b \)
    from \( bdef \) have \( b : \text{setOf} \ t1 \ | \ b = x | \ b : \text{setOf} \ t2 \) by auto
moreover { assume bdef1: b : setOf t1
  from bdef1 s have o1: h b < h x by auto
  from o1 o2 have h b < h a by simp
  from this hab have ?thesis by simp } — case impossible
moreover { assume bdef1: b = x
  from adef1 bdef1 s have h b < h a by auto
  from this hahb have ?thesis by simp
  ultimately show ?thesis by blast
qed }
ultimately show ?thesis by blast
qed

lemma tlookup-finds: — if a node is in the tree, lookup finds it
sortedTree h t & y : setOf t -->
  tlookup h (h y) t = Some y
proof safe
  assume s: sortedTree h t
  assume yint: y : setOf t
  show tlookup h (h y) t = Some y
  proof (cases tlookup h (h y) t)
    case None note res = this
    from s res have sortedTree h t & (tlookup h (h y) t = None) by simp
    from this have o1: \x\in setOf t. h x \sim= h y by (simp add: tlookup-none)
    from o1 yint have h y \sim= h y by fastforce
    from this show ?thesis by simp
  next case (Some z) note res = this
    have is: sortedTree h t & (tlookup h (h y) t = Some z) -->
      z : setOf t & h z = h y by (simp add: tlookup-some)
    have sd: sorted-distinct-pred h y z t
    by (insert sorted-distinct [of h y z t], simp)
    from s res is have o1: z : setOf t & h z = h y by simp
    from s yint o1 sd have y = z by auto
    from this res show tlookup h (h y) t = Some y by simp
qed
qed

3.1 Tree membership as a special case of lookup
definition memb :: 'a hash => 'a => 'a Tree => bool where
  memb h x t ==
  (case (tlookup h (h x) t) of
    None => False
lemma assumes \( s \): sortedTree \( h \ t \)
shows memb-spec: \( \text{memb} \( h \ x \ t \) = (x : \text{setOf} \ t) \)
proof (cases tlookup \( h \ (h \ x) \) )
case None note tNone = this 
from tNone have res: \( \text{memb} \( h \ x \ t \) = \text{False} \) by (simp add: memb-def)
have notIn: \( x \sim : \text{setOf} \ t \)
proof 
  assume h: \( x : \text{setOf} \ t \)
  from h have h x \( \sim = h \ x \) by fastforce
  from this show False by simp
qed
from res notIn show \( \text{thesis} \) by simp
next case (Some \( z \)) note tSome = this
from s tSome tlookup-some have zin: \( z : \text{setOf} \ t \) by fastforce
show \( \text{thesis} \)
proof (cases \( x = z \) )
case True note xez = this
from tSome xez have res: \( \text{memb} \( h \ x \ t \) \)
from res zin xez show \( \text{thesis} \) by simp
next case False note xnez = this
from tSome xnez have res: \( \sim \text{memb} \( h \ x \ t \) \)
have x \( \sim : \text{setOf} \ t \)
proof 
  assume xin: \( x : \text{setOf} \ t \)
  from s tSome tlookup-some have hhzx: \( h \ x = h \ z \) by fastforce
  have o1: \( \text{sorted-distinct-pred} \( h \ x \ z \ t \) \)
  by (insert sorted-distinct \( [h \ x \ z \ t] \), simp)
  from s xin zin hhzx o1 have x = z by fastforce
  from this xnez show False by simp
qed
from this res show \( \text{thesis} \) by simp
qed
qed

declare sorted-distinct-pred-def \( [\text{simp del}] \)

4 Insertion into a Tree

primrec 
bin\( \text{insert} \) :: \( 'a \text{ hash} \Rightarrow 'a \Rightarrow 'a \text{ Tree} \Rightarrow 'a \text{ Tree} \)
where
\( \text{bin} \text{insert} \( h \ e \ T\text{ip} \) = (T \ T\text{ip} e \ T\text{ip}) \)
| \( \text{bin} \text{insert} \( h \ e \ (T \ t1 \ t2) \) = (\text{if} \ h \ e < h \ x \ \text{then} \ (T \ (\text{bin} \text{insert} \( h \ e \ t1 \)) \ t2) \) \)
else 
(\text{if} h x < h e \ \text{then} \n
6
A technique for proving disjointness of sets.

**lemma** disjCond: [!! x. [!! x:A; x:B] ==> False] ==> A Int B = {}

by fastforce

The following is a proof that insertion correctly implements the set interface. Compared to *BinaryTree-TacticStyle*, the claim is more difficult, and this time we need to assume as a hypothesis that the tree is sorted.

**lemma** binsert-set: sortedTree h t --->
setOf (binsert h e t) = (setOf t) − (eqs h e) Un {e}

(is ?P t)

**proof** (induct t)
— base case
show ?P Tip by (simp add: eqs-def)
— induction step
fix t1 :: 'a Tree assume h1: ?P t1
fix t2 :: 'a Tree assume h2: ?P t2
fix x :: 'a
show ?P (T t1 x t2)
proof
assume s: sortedTree h (T t1 x t2)
from s have s1: sortedTree h t1 by (rule sortLemmaL)
from s1 and h1 have c1: setOf (binsert h e t1) = setOf t1 − (eqs h e) Un {e}
by simp
from s have s2: sortedTree h t2 by (rule sortLemmaR)
from s2 and h2 have c2: setOf (binsert h e t2) = setOf t2 − (eqs h e) Un {e}
by simp
show setOf (binsert h e (T t1 x t2)) =
setOf (T t1 x t2) − (eqs h e) Un {e}
proof (cases h e < h x)
case True note eLess = this
from eLess have res: binsert h e (T t1 x t2) = (T (binsert h e t1) x t2)
by simp
show setOf (binsert h e (T t1 x t2)) =
setOf (T t1 x t2) − (eqs h e) Un {e}
proof (simp add: res eLess c1)
show insert x (insert e (setOf t1 − (eqs h e Un setOf t2))) =
insert e (insert x (setOf t1 Un setOf t2) − (eqs h e))
proof —
have eqsLessX: ∀el ∈ eqs h e. h el < h x by (simp add: eqs-def eLess)
from this have eqsDisjX: ∀el ∈ eqs h e. h el ~= h x by fastforce
from s have xLessT2: ∀r ∈ setOf t2. h x < h r by auto
have eqsLessT2: ∀el ∈ eqs h e. ∀r ∈ setOf t2. h el < h r
proof safe
fix el assume hel: el : eqs h e
from hel eqs-def have a1: h el = h e by fastforce
fix r assume hr: r : setOf t2
from xLessT2 hr o1 eLess show h el < h r by auto
qed
from eqsLessT2 have eqsDisjT2: ∀ el ∈ eqs h e. ∀ r ∈ setOf t2. h el ∼ = h r
by fastforce
from eqsDisjX eqsDisjT2 show ?thesis by fastforce
qed
next case False note eNotLess = this
show setOf (binsert h e (T t1 x t2)) = setOf (T t1 x t2) − eqs h e Un {e}
proof (cases h x < h e)
  case True note xLess = this
  from xLess have res: binsert h e (T t1 x t2) = (T t1 x (binsert h e t2)) by simp
  show setOf (binsert h e (T t1 x t2)) = setOf (T t1 x t2) − eqs h e Un {e}
  proof (simp add: res xLess eNotLess c2)
    show insert x (insert e (setOf t1 Un (setOf t2 − eqs h e))) =
      insert e (insert x (setOf t1 Un setOf t2) − eqs h e)
    proof
      have XLessEqs: ∀ el ∈ eqs h e. h x < h el by (simp add: eqs-def xLess)
      from this have eqsDisjX: ∀ el ∈ eqs h e. h el ∼ = h x by auto
      from s have t1LessX: ∀ l ∈ setOf t1. h l < h x by auto
      have T1lessEqs: ∀ el ∈ eqs h e. ∀ l ∈ setOf t1. h l < h el
      proof safe
        fix el assume hel: el : eqs h e
        fix l assume hl: l : setOf t1
        from hel eqs-def have o1: h el = h e by fastforce
        from t1LessX hl o1 xLess show h l < h el by auto
        qed
      from T1lessEqs have T1disjEqs: ∀ el ∈ eqs h e. ∀ l ∈ setOf t1. h el ∼ = h l
        by fastforce
      from eqsDisjX T1lessEqs show ?thesis by auto
      qed
    qed
  qed
next case False note xNotLess = this
from xNotLess eNotLess have xege: h x = h e by simp
from xege have res: binsert h e (T t1 x t2) = (T t1 e t2) by simp
show setOf (binsert h e (T t1 x t2)) =
  setOf (T t1 x t2) − eqs h e Un {e}
proof (simp add: res eNotLess xege)
  show insert e (setOf t1 Un setOf t2) =
    insert e (insert x (setOf t1 Un setOf t2) − eqs h e)
  proof
    have insert x (setOf t1 Un setOf t2) =
      setOf t1 Un setOf t2
    proof
      have x : eqs h e by (simp add: eqs-def xege)
moreover have \((\text{setOf } t1) \cap \text{eqs } h \ e) = \emptyset\)
proof (rule disjCond)
fix \(w\)
assume \(whSet: w : \text{setOf } t1\)
assume \(whEq: w : \text{eqs } h \ e\)
from \(whSet\) have \(o1: h \ w < h \ x\) by simp
from \(whEq\) have \(o2: h \ w = h \ e\) by fastforce
from \(o2 \ xeqe\) have \(o3: \sim h \ w < h \ x\) by simp
from \(o1 \ o3\) show False by contradiction
qed
moreover have \((\text{setOf } t2) \cap \text{eqs } h \ e) = \emptyset\)
proof (rule disjCond)
fix \(w\)
assume \(whSet: w : \text{setOf } t2\)
assume \(whEq: w : \text{eqs } h \ e\)
from \(whSet\) have \(o1: h \ x < h \ w\) by simp
from \(whEq\) have \(o2: h \ w = h \ e\) by fastforce
from \(o2 \ xeqe\) have \(o3: \sim h \ x < h \ w\) by simp
from \(o1 \ o3\) show False by contradiction
qed
ultimately show \(?thesis\) by auto
qed
from this show \(?thesis\) by simp
qed
qed
qed
qed

Using the correctness of set implementation, preserving sortedness is still simple.

**Lemma** \(binsert\)-sorted: \(\text{sortedTree } h \ t \longrightarrow \text{sortedTree } h \ (binsert \ h \ x \ t)\)
by (induct \(t\)) (auto simp add: \(binsert\)-set)

We summarize the specification of \(binsert\) as follows.

**Corollary** \(binsert\)-spec: \(\text{sortedTree } h \ t \longrightarrow \)
\(\text{sortedTree } h \ (binsert \ h \ x \ t)\ & \)
\(\text{setOf } (binsert \ h \ e \ t) = (\text{setOf } t) - (\text{eqs } h \ e) \cup \{e\}\)
by (simp add: \(binsert\)-set \(binsert\)-sorted)

5 Removing an element from a tree

These proofs are influenced by those in *BinaryTree-Tactic*

**primrec**
\(rm :: \ ('a \ \text{hash} \Rightarrow 'a \ \text{Tree} \Rightarrow 'a)\)
— rightmost element of a tree
where
\[
\text{rm } h \ (T \ t1 \ x \ t2) = \\
(\text{if } t2 = \text{Tip} \ \text{then } x \ \text{else rm } h \ t2)
\]

primrec
\[
\text{wrm} :: 'a \ 	ext{hash} => 'a \ \text{hash} => 'a \ 	ext{Tree} \\
\text{— tree without the rightmost element}
\]
where
\[
\text{wrm } h \ (T \ t1 \ x \ t2) = \\
(\text{if } t2 = \text{Tip} \ \text{then } t1 \ \text{else (T } t1 \ x \ (\text{wrm } h \ t2)))
\]

primrec
\[
\text{wrmrm} :: 'a \ 	ext{hash} => 'a \ \text{hash} => 'a \ 	ext{Tree} \ 	ext{∗ 'a} \\
\text{— computing rightmost and removal in one pass}
\]
where
\[
\text{wrmrm } h \ (T \ t1 \ x \ t2) = \\
(\text{if } t2 = \text{Tip} \ \text{then (t1, x) else (T } t1 \ x \ (\text{fst (wrmrm } h \ t2)), \\
\text{snd (wrmrm } h \ t2)))
\]

primrec
\[
\text{remove} :: 'a \ 	ext{hash} => 'a => 'a \ 	ext{hash} => 'a \ 	ext{Tree} \\
\text{— removal of an element from the tree}
\]
where
\[
\text{remove } h \ e \ \text{Tip} = \text{Tip} \\
\mid \text{remove } h \ e \ (T \ t1 \ x \ t2) = \\
(\text{if } h < h \ e \ \text{then (T (remove } h \ e \ t1) \ x \ t2) \\
\text{else if } h \ e < h \ \text{then (T } t1 \ x \ (\text{remove } h \ e \ t2))) \\
\text{else (if } t1 = \text{Tip} \ \text{then t2 else let } (t1p, r) = \text{wrmrm } h \ t1 \\
\text{in (T } t1p \ r \ t2)))
\]

theorem \text{wrmrm-decomp: } t \sim \text{ Tip} \rightarrow \text{ wrmrm } h \ t = (\text{wrm } h \ t, \text{ rm } h \ t)
apply (induct-tac t)
apply simp-all
done

lemma \text{rm-set: } t \sim \text{ Tip & sortedTree } h \ t \rightarrow \text{ rm } h \ t : \text{setOf } t
apply (induct-tac t)
apply simp-all
done

lemma \text{wrm-set: } t \sim \text{ Tip & sortedTree } h \ t \rightarrow \\
\text{setOf (wrm } h \ t) = \text{setOf } t \setminus \{\text{rm } h \ t\} \ (\text{is } ?P \ t)
proof (induct t)
show ?P \ Tip by simp
fix t1 :: 'a \ 	ext{Tree} \ assume \ h1: ?P \ t1
fix t2 :: 'a \ 	ext{Tree} \ assume \ h2: ?P \ t2
fix x :: 'a
show \(?P\ (T \ t1 \ x \ t2)\)

proof (rule impI, erule conjE)
assume s: sortedTree h \((T \ t1 \ x \ t2)\)
show setOf \((\text{wrm} \ h \ (T \ t1 \ x \ t2))\) =
setOf \((T \ t1 \ x \ t2) - \{\text{rm} \ h \ (T \ t1 \ x \ t2)\}\)
proof (cases \(t2 = \text{Tip}\))
case True
note \(t2\text{-tip} = \text{this}\)
from t2-tip have \(\text{rm-res}: \text{rm} \ h \ (T \ t1 \ x \ t2) = x\) by simp
from s have \(x \sim \text{setOf} \ t1\) by auto
from this rm-res wrm-res t2-tip show \(\text{thesis}\) by simp
next case False
note \(t2\text{-nTip} = \text{this}\)
from t2n-tip have \(\text{rm-res}: \text{rm} \ h \ (T \ t1 \ x \ t2) = \text{rm} \ h \ t2\) by simp
from t2n-tip have \(\text{wrm-res}: \text{wrm} \ h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\text{wrm} \ h \ t2)\) by simp
from s have \(s2: \text{sortedTree} \ h \ t2\) by simp
from \(h2 \ t2\text{-nTip} \ s2\) have \(o1: \text{setOf} \ (\text{wrm} \ h \ t2) = \text{setOf} \ t2 - \{\text{rm} \ h \ t2\}\) by simp
show \(\text{thesis}\) proof (simp add: rm-res wrm-res t2n-tip h2 o1)

show \(\text{thesis}\)

qed
qed
qed

lemma \(\text{wrm-set1}: t \sim = \text{Tip} \ & \ \text{sortedTree} \ h \ t \ -\rightarrow \ \text{setOf} \ (\text{wrm} \ h \ t) \ leq \ \text{setOf} \ t\)
by (auto simp add: wrm-set)

lemma \(\text{wrm-sort}: t \sim = \text{Tip} \ & \ \text{sortedTree} \ h \ t \ -\rightarrow \ \text{sortedTree} \ h \ (\text{wrm} \ h \ t) \ (\text{is} \ \ ?P \ t)\)
proof (induct t)
show \(?P\ \text{Tip}\) by simp
fix \(t1 :: \text{Tree} \) assume h1: \(?P\ t1\)
fix \(t2 :: \text{Tree} \) assume h2: \(?P\ t2\)
fix \(x :: \text{a}\)
show \(?P\ (T \ t1 \ x \ t2)\)
proof safe
assume s: sortedTree h \((T \ t1 \ x \ t2)\)

11
show sortedTree h (wrm h (T t1 x t2))
proof (cases t2 = Tip)
  case True note t2tip = this
    from t2tip have res: wrm h (T t1 x t2) = t1 by simp
    from res s show ?thesis by simp
next case False note t2nTip = this
  from t2nTip have res: wrm h (T t1 x t2) = T t1 x (wrm h t2) by simp
  from s have s1: sortedTree h t1 by simp
  from s have s2: sortedTree h t2 by simp
  from s2 h2 t2nTip have o1: sortedTree h (wrm h t2) by simp
  from s2 t2nTip wrm-set1 have o2: setOf (wrm h t2) <= setOf t2 by auto
  from s o2 have s1 o1 o3 res s show sortedTree h (wrm h (T t1 x t2)) by simp
qed

lemma wrm-less-rm:
t ~ =~ Tip & sortedTree h t --->
(\forall l \in setOf (wrm h t). h l < h (rm h t)) (is ?P t)
proof (induct t)
  show ?P Tip by simp
  fix t1 :: 'a Tree assume h1: ?P t1
  fix t2 :: 'a Tree assume h2: ?P t2
  fix x :: 'a
  show ?P (T t1 x t2)
  proof safe
    fix l :: 'a assume ldef: l : setOf (wrm h (T t1 x t2))
    assume s: sortedTree h (T t1 x t2)
    from s have s1: sortedTree h t1 by simp
    from s have s2: sortedTree h t2 by simp
    show h l < h (rm h (T t1 x t2))
    proof (cases t2 = Tip)
      case True note t2tip = this
      from t2tip have rm-res: rm h (T t1 x t2) = x by simp
      from t2tip have wrm-res: wrm h (T t1 x t2) = t1 by simp
      from ldef wrm-res have o1: l : setOf t1 by simp
      from rm-res o1 s show ?thesis by simp
    next case False note t2nTip = this
      from t2nTip have rm-res: rm h (T t1 x t2) = rm h t2 by simp
      from ldef wrm-res
      have l-scope: l : {x} Un setOf t1 Un setOf (wrm h t2) by simp
      have hLess: h l < h (rm h t2)
      proof (cases l = x)
        case True note lx = this
        from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
        from lx o1 show ?thesis by simp
      next case False note lx = this
  qed
  qed
show \(?thesis\)
proof (cases \(l\) : setOf \(t1\))
case True note \(l\)-in-
\(t1\) = this
  from \(s\) \texttt{2nTip rm-set \(s2\)} have \(o1\)\(: h\ x < h\ (rm\ h\ t2)\) by auto
  from \(l\)-in-
\(t1\) \texttt{s have} \(o2\)\(: h\ l < h\ x\) by auto
  from \(o1\) \(o2\) show \(?thesis\) by simp
next case False note \(l\)-notin-
\(t1\) = this
  from \texttt{l-scope} \(h\) \texttt{l-notin-
\(t1\)} have \(l\)-in-
\(\texttt{res}\) : \(l\) : setOf \((\texttt{wrm}\ h\ t2)\) by auto
  from \(l\)-in-
\(\texttt{res}\) \texttt{h2 \texttt{2nTip \(s2\)}} show \(?thesis\) by auto
qed
qed
from \texttt{rm-res} \(h\) \texttt{Less} show \(?thesis\) by simp
qed
qed

lemma remove-set : sortedTree \(h\ \(t\) \longrightarrow \)
setOf \((\texttt{remove}\ h\ \texttt{e}\ \(t\)) = \text{setOf}\ \(t\) \{-\text{eqs}\ h\ \texttt{e}\ \} \{(\text{is} \ ?P \ \(t\) \})
proof \(\text{induct} \(t\)\)
  show \(?P \ \texttt{Tip}\) by auto
  fix \(t1\) :: \(\texttt{Tree}\) assume \(h1\)\(: \ ?P \ \(t1\)\)
  fix \(t2\) :: \(\texttt{Tree}\) assume \(h2\)\(: \ ?P \ \(t2\)\)
  fix \(x\) :: \(\texttt{a}\)
  show \(?P\ \(\texttt{T} \ \(t1\) \ \(x\) \ \(t2\)\)
proof
  assume \(s\) : sortedTree \(h\ \(\texttt{T} \ \(t1\) \ \(x\) \ \(t2\)\)
  show setOf \((\texttt{remove}\ h\ \texttt{e}\ \(t\)) = \text{setOf}\ \(t\) \{-\text{eqs}\ h\ \texttt{e}\ \} \{(\text{is} \ ?P \ \(t\) \}) by simp
proof \(\text{cases} \(h\ \texttt{e} < h\ \texttt{x}\)\)
  case True note \(el\) = \texttt{this}
    from \(el\) have res : remove \(h\ \texttt{e}\ \(\texttt{T} \ \(t1\) \ \(x\) \ \(t2\)\) = \(T\) \{(remove\ \texttt{h} \ \texttt{e}\ \(\texttt{t1}\) \ \(x\) \ \(t2\)\) by simp
from \texttt{s have} \(s1\) : sortedTree \(h\ \texttt{t1}\) by simp
from \(s1\ \texttt{h1}\) have \(o1\) : setOf \((\texttt{remove}\ h\ \texttt{e}\ \(\texttt{t1}\)\) = setOf \(\texttt{t1}\) \{-\text{eqs}\ h\ \texttt{e}\ \} \{(\text{by simp}\}\)
show \(?thesis\)
proof \(\texttt{(simp add} : \(o1\) \(el\)\)
  show insert \(x\) \((\text{setOf}\ \texttt{t1} \ -\ \text{eqs}\ h\ \texttt{e}\ \text{Un}\ \text{setOf}\ \texttt{t2})\) =
    insert \(x\) \((\text{setOf}\ \texttt{t1} \ \text{Un}\ \text{setOf}\ \texttt{t2})\) \{-eqs\ h\ \texttt{e}\ \}
proof
  have \(x\)\(Ok\) : \(x\) \texttt{\texttt{\sim}: eqs\ h\ \texttt{e}\)
  proof
    assume \(h\) : \(x\) \texttt{\texttt{\sim}: eqs\ h\ \texttt{e}\)
      from \(h\) have \(o1\)\(: \sim\ (h\ \texttt{e} < h\ \texttt{x})\) by \(\texttt{(simp add} : \texttt{eqs-def}\}\)
    from \(el\) \(o1\) show \(False\) by contradiction
  qed
  have \(t\)\(2Ok\) : \((\text{setOf}\ \texttt{t2})\) \texttt{Int} \{(eqs\ h\ \texttt{e})\) = \{
  proof \(\texttt{(rule} \texttt{disjCond)}\)
    fix \(y\) :: \(\texttt{a}\)
    assume \(y\)-in-
\(t2\) : \(y\) : setOf \(t2\)
assume y-in-eq: y : eqs h e
from y-in-t2 s have xly: h x < h y by auto
from y-in-eq have eey: h y = h e by (simp add: eqs-def)
from xly eey have nelx: ~ (h e < h x) by simp
from nelx xle show False by contradiction
qed
from xOk t2Ok show ?thesis by auto
qed
qed
next case False note nelx = this
show ?thesis
proof (cases h x < h e)
case True
note xle = this
from xle have res: remove h e (T t1 x t2) = T t1 x (remove h e t2) by simp
from s have s2: sortedTree h t2 by simp
from s2 h2 have o1: setOf (remove h e t2) = setOf t2 - eqs h e by simp
show ?thesis
proof
have xOk: x ∼: eqs h e
proof
  assume h: x : eqs h e
  from h have o1: ~ (h x < h e) by (simp add: eqs-def)
  from xle o1 show False by contradiction
qed
have t1Ok: (setOf t1) Int (eqs h e) = {}
proof (rule disjCond)
  fix y :: 'a
  assume y-in-t1: y : setOf t1
  assume y-in-eq: y : eqs h e
  from y-in-t1 s have ylx: h y < h x by auto
  from y-in-eq have eey: h y = h e by (simp add: eqs-def)
  from ylx eey have nelx: ~ (h x < h e) by simp
  from nelx xle show False by contradiction
qed
from xOk t1Ok show ?thesis by auto
qed
qed
next case False note nxle = this
from nelx nxle have ex: h e = h x by simp
have t2Ok: (setOf t2) Int (eqs h e) = {}
proof (rule disjCond)
  fix y :: 'a
  assume y-in-t2: y : setOf t2
  assume y-in-eq: y : eqs h e
  from y-in-t2 s have xly: h x < h y by auto
from y-in-eq have eey: \( h \cdot y = h \cdot e \) by (simp add: eqs-def)
from y-in-eq ex eey have nxly: \(~ (h \cdot z < h \cdot y)\) by simp
from nxly xly show False by contradiction

qed

show ?thesis

proof (cases t1 = Tip)
case True note t1tip = this
from ex t1tip have res: remove h e (T t1 t2) = t2 by simp
show ?thesis

proof (simp add: res t1tip ex)
  show (setOf t2 = insert x (setOf t2) - eqs h e)
    proof
      from ex have x-in-eqs: x : eqs h e by (simp add: eqs-def)
      from x-in-eqs t2Ok show ?thesis by auto
    qed

  qed

next case False note t1nTip = this
from nxle ex t1nTip have res: remove h e (T t1 t2) = T (wrm h t1) (rm h t1) t2
by (simp add: Let-def wrmrm-decomp)
from res show ?thesis

proof simp
from s have s1: sortedTree h t1 by simp
  show (insert (rm h t1) (setOf (wrm h t1) \ Un \ setOf t2) =
    insert x (setOf t1 \ Un \ setOf t2) - eqs h e)
    proof (simp add: t1nTip s1 rm-set wrm-set)
      show (insert (rm h t1) (setOf t1 - {rm h t1} \ Un \ setOf t2) =
        insert x (setOf t1 \ Un \ setOf t2) - eqs h e)
        proof
          from t1nTip s1 rm-set
          have o1: insert (rm h t1) (setOf t1 - {rm h t1} \ Un \ setOf t2) =
            setOf t1 \ Un \ setOf t2 by auto
          have o2: insert x (setOf t1 \ Un \ setOf t2) - eqs h e =
            setOf t1 \ Un \ setOf t2
            proof
              from ex have xOk: x : eqs h e by (simp add: eqs-def)
              have t1Ok: (setOf t1) Int (eqs h e) = {}
              proof (rule disjCond)
                fix y :: 'a
                assume y-in-t1: y : setOf t1
                assume y-in-eq: y : eqs h e
                from y-in-t1 s ex have o1: h y < h e by auto
                from y-in-eq have o2: ~ (h y < h e) by (simp add: eqs-def)
                from o1 o2 show False by contradiction
              qed
              from xOk t1Ok t2Ok show ?thesis by auto
              qed
            from o1 o2 show ?thesis by simp
          qed
        qed
    qed

  qed
lemma remove-sort: sortedTree h t -->
    sortedTree h (remove h e t) (is \( ?P \, t \))
proof (induct t)
  show \( ?P \, \text{Tip} \) by auto
  fix t1 :: 'a Tree assume h1: \( ?P \, t1 \)
  fix t2 :: 'a Tree assume h2: \( ?P \, t2 \)
  fix x :: 'a
  show \( ?P \, (T \, t1 \, x \, t2) \)
  proof
    assume \( s \, : \, \text{sortedTree} \, h \, (T \, t1 \, x \, t2) \)
    from \( s \) have \( s1 \, : \, \text{sortedTree} \, h \, t1 \) by simp
    from \( s \) have \( s2 \, : \, \text{sortedTree} \, h \, t2 \) by simp
    from \( h1 \, s1 \) have \( sr1 \, : \, \text{sortedTree} \, h \, (\text{remove} \, h \, e \, t1) \) by simp
    from \( h2 \, s2 \) have \( sr2 \, : \, \text{sortedTree} \, h \, (\text{remove} \, h \, e \, t2) \) by simp
    show \( \text{sortedTree} \, h \, (\text{remove} \, h \, e \, (T \, t1 \, x \, t2)) \)
    proof (cases h e < h x)
      case True note elx = this
      from elx have res: \( \text{remove} \, h \, e \, (T \, t1 \, x \, t2) = T \, (\text{remove} \, h \, e \, t1) \, x \, t2 \)
        by simp
      show \( \text{thesis} \)
        proof (simp add: \( s \, s1 \, sr1 \, s2 \, elx \, res \))
          let \( C1 \, = \forall l \, \in \, \text{setOf} \, (\text{remove} \, h \, e \, t1), \, h \, l < h \, x \)
          let \( C2 \, = \forall r \, \in \, \text{setOf} \, t2, \, h \, x < h \, r \)
          have o1: \( C1 \)
            proof
              from \( s1 \) have \( \text{setOf} \, (\text{remove} \, h \, e \, t1) = \text{setOf} \, t1 \, - \, \text{eqs} \, h \, e \) by (simp add: \( \text{remove-set} \))
                from \( s \, \text{this} \) show \( \text{thesis} \) by auto
            qed
          from \( o1 \) show \( C1 \, \& \, C2 \) by auto
          qed
        qed
      next case False note nelx = this
      show \( \text{thesis} \)
        proof (cases h x < h e)
          case True note xle = this
          from xle have res: \( \text{remove} \, h \, e \, (T \, t1 \, x \, t2) = T \, t1 \, x \, (\text{remove} \, h \, e \, t2) \) by simp
            show \( \text{thesis} \)
              proof (simp add: \( s \, s1 \, sr2 \, xle \, nelx \, res \))
                let \( C1 \, = \forall l \, \in \, \text{setOf} \, t1, \, h \, l < h \, x \)
              qed
            qed
          qed
        qed
    qed
  qed
qed
let $C2 = \forall r \in \text{setOf} \ (\text{remove} \ h \ e \ t2). \ h \ x < h \ r$

have $o2$: $C2$

proof -
    from $s2$ have $\text{setOf} \ (\text{remove} \ h \ e \ t2) = \text{setOf} \ t2 - \text{eqs} \ h \ e$ by (simp add: remove-set)
    from $s$ this show $\text{?thesis by auto}$
    qed

next case False note $nxle = \text{this}$
    from $\text{nelx \ nxle}$ have $\text{ex: \ h} \ e = h \ x$ by simp
    show $\text{?thesis}$
    proof (cases $t1 = T\text{ip}$)
    case True
    note $t1tip = \text{this}$
    from $\text{ex \ t1tip}$ have $\text{res: \ remove} \ h \ e \ (T \ t1 \ x \ t2) = t2$ by simp
    show $\text{?thesis}$ by (simp add: res $t1tip$ $ex \ s2$)
    next case False
    note $t1nTip = \text{this}$
    from $\text{nelx \ nxle \ ex \ t1nTip}$ have $\text{res: \ remove} \ h \ e \ (T \ t1 \ x \ t2) = T \ (\text{wrm} \ h \ t1) \ (\text{rm} \ h \ t1) \ t2$
    by (simp add: Let-def wrmrm-decomp)
    from $\text{res}$ show $\text{?thesis}$
    proof simp
    let $C1 = \text{sortedTree} \ h \ (\text{wrm} \ h \ t1)$$C2 = \forall l \in \text{setOf} \ (\text{wrm} \ h \ t1). \ h \ l < h \ (\text{rm} \ h \ t1)$$C3 = \forall r \in \text{setOf} \ t2. \ h \ (\text{rm} \ h \ t1) < h \ r$$C4 = \text{sortedTree} \ h \ t2$
    from $s1 \ \text{t1nTip}$ have $o1$: $C1$ by (simp add: wrm-sort)
    from $s1 \ \text{t1nTip}$ have $o2$: $C2$ by (simp add: wrm-less-rm)
    have $o3$: $C3$
    proof
    fix $r :: 'a$
    assume $rt2$: $r : \text{setOf} \ t2$
    from $\text{sr} \ t1nTip$ have $o1$: $h \ (\text{rm} \ h \ t1) < h \ x$ by auto
    from $rt2 \ s$ have $o2$: $h \ x < h \ r$ by auto
    from $o1 \ o2$ show $h \ (\text{rm} \ h \ t1) < h \ r$ by simp
    qed
    from $o1 \ o2 \ o3 \ s2$ show $C1 \ & \ C2 \ & \ C3 \ & \ C4$ by simp
    qed
    qed
    qed
    qed
    qed
    qed

We summarize the specification of remove as follows.

corollary remove-spec: \text{sortedTree} \ h \ t --\rightarrow
\text{sortedTree} \ h \ (\text{remove} \ h \ e \ t) \ & $\text{setOf} \ (\text{remove} \ h \ e \ t) = \text{setOf} \ t - \text{eqs} \ h \ e$
by (simp add: remove-sort remove-set)

definition test = tlookup id 4 (remove id 3 (binsert id 4 (binsert id 3 Tip)))

export-code test
  in SML module-name BinaryTree-Code file BinaryTree-Code.ML
end

6 Mostly Isar-style Reasoning for Binary Tree Operations

theory BinaryTree-Map imports BinaryTree begin

We prove correctness of map operations implemented using binary search trees from BinaryTree.
This document is LGPL.
Author: Viktor Kuncak, MIT CSAIL, November 2003

7 Map implementation and an abstraction function

type-synonym 'a tarray = (index * 'a) Tree

definition valid-tmap :: 'a tarray => bool where
  valid-tmap t == sortedTree fst t

declare valid-tmap-def [simp]

definition mapOf :: 'a tarray => index => 'a option where
  — the abstraction function from trees to maps
  mapOf t i ==
    (case (tlookup fst i t) of
      None => None
    | Some ia => Some (snd ia))

8 Auxiliary Properties of our Implementation

lemma mapOf-lookup1: tlookup fst i t = None ==> mapOf t i = None
  by (simp add: mapOf-def)

lemma mapOf-lookup2: tlookup fst i t = Some (j,a) ==> mapOf t i = Some a
  by (simp add: mapOf-def)

lemma assumes h: mapOf t i = None
shows mapOf-lookup3: tlookup fst i t = None
proof (cases tlookup fst i t)
case None from this show ?thesis by assumption
next case (Some ia) note tsome = this
  from this have o1: tlookup fst i t = Some (fst ia, snd ia) by simp
  have mapOf t i = Some (snd ia)
    by (insert mapOf-lookup2 [of i t fst ia snd ia], simp add: o1)
  from this have mapOf t i = None by simp
  from this h show ?thesis by simp — contradiction
qed

lemma assumes v: valid-tmap t
  assumes h: mapOf t i = Some a
  shows mapOf-lookup4: tlookup fst i t = Some (i, a)
proof (cases tlookup fst i t)
case None from this mapOf-lookup1 have mapOf t i = None by auto
  from this h show ?thesis by simp — contradiction
next case (Some ia) note tsome = this
  have tlookup-some-inst: sortedTree fst t & (tlookup fst i t = Some ia) -->
    ia : setOf t & fst ia = i by (simp add: tlookup-some)
  from tlookup-some-inst tsome v have ia : setOf t by simp
  from tlookup-some-inst tsome v have mapOf t i = Some (snd ia) by (simp add: mapOf-def)
  from this h have o1: snd ia = a by simp
  from o1 o2 have ia = (i, a) by auto
  from this tsome show tlookup fst i t = Some (i, a) by simp
qed

8.1 Lemmas mapset-none and mapset-some establish a relation
between the set and map abstraction of the tree

lemma assumes v: valid-tmap t
  shows mapset-none: (mapOf t i = None) = (∀ a. (i, a) \notin setOf t)
proof
  -- ==¿
  assume mapNone: mapOf t i = None
  from v mapNone mapOf-lookup3 have lnone: tlookup fst i t = None by auto
  show ∀ a. (i, a) \notin setOf t
    proof
      fix a
      show (i, a) : setOf t
        proof
          assume iain: (i, a) : setOf t
          have tlookup-none-inst:
            sortedTree fst t & (tlookup fst i t = None) -->
            (∀ x \in setOf t. fst x = i)
          by (insert tlookup-none [of fst t i], assumption)
          from v lnone tlookup-none-inst have ∀ x \in setOf t. fst x = i by simp
          from this iain have fst (i, a) = i by fastforce
        qed
    qed
  qed
lemma assumes \( v :: \text{valid-tmap } t \)
shows \( \text{mapset-some: } (\text{mapOf } t \ i = \text{Some } (i,a)) = ((i,a) : \text{setOf } t) \)
proof
  — ==¿
  assume \( \text{mapsome: } \text{mapOf } t \ i = \text{Some } a \)
  from \( v \text{ mapsome have o1: } \text{lookup } \text{fst } i \ t = \text{Some } (i,a) \) by (simp add: \text{mapOf-lookup4})
  from \( t \text{lookup-some have tlookup-some-inst: } \text{sortedTree } \text{fst } t \ & \ \text{lookup } \text{fst } i \ t = \text{Some } (i,a) \longrightarrow \)
  \( (i,a) : \text{setOf } t \ & \ \text{fst } (i,a) = i \)
  by (insert \( t \text{lookup-some [of } \text{fst } t \ i \ (i,a)] \), assumption)
  from \( v \ o1 \text{ this have } (i,a) : \text{setOf } t \) by simp
  from \( t \text{ this show } \text{thesis by auto} — \text{contradiction} \)
qed

9 Empty Map

lemma \text{mnew-spec-valid: } \text{valid-tmap Tip}
by (simp add: \text{mapOf-def})

lemma \text{mtip-spec-empty: } \text{mapOf Tip } k = \text{None}
by (simp add: \text{mapOf-def})
10 Map Update Operation

**definition** `mupdate :: index => 'a => 'a tarray => 'a tarray where` `mupdate i a t == binsert fst (i,a) t`

**lemma assumes** `v :: valid-tmap t`
**shows** `mupdate-map: mapOf (mupdate i a t) = (mapOf t)(i |-> a)`
**proof**
- `fix i2`  
  - `let ?tr = binsert fst (i,a) t`  
  - `have upres: mupdate i a t = ?tr`  
  - `from v binsert-set have setSpec: setOf ?tr = setOf t - eqs fst (i,a) Un {(i,a)} by fastforce`
  - `from v binsert-sorted have vr: valid-tmap ?tr by simp`
  - `show mapOf (mupdate i a t) i2 = ((mapOf t)(i |-> a)) i2`  
  - `proof (cases i = i2)`
  - `case True`  
    - `note i2ei = this`  
    - `from i2ei have rhs-res: ((mapOf t)(i |-> a)) i2 = Some a by simp`
    - `have lhs-res: mapOf (mupdate i a t) i = Some a`  
      - `proof (cases mapOf t i2)`
      - `case None`  
        - `from this have mapNone: mapOf t i2 = None by simp`
        - `from v mapNone mapset-none have i2nin: \forall a. (i2,a) \notin setOf t by fastforce`  
        - `have noneIn: \forall b. (i2,b) \notin setOf t`  
          - `from v binsert-set have setOf ?tr = setOf t - eqs fst (i,a) Un {(i,a)} by fastforce`  
          - `from this i2ei i2nin show (i2,b) ~: setOf ?tr by fastforce`  
          - `qed`
    - `next case False note i2nei = this`  
      - `from i2nei have rhs-res: ((mapOf t)(i |-> a)) i2 = mapOf t i2 by auto`
      - `have lhs-res: mapOf (mupdate i a t) i2 = mapOf t i2`  
        - `proof (cases mapOf t i2)`
      - `case None`  
        - `from this have mapNone: mapOf t i2 = None by simp`
        - `from v mapNone mapset-none have i2nin: \forall a. (i2,a) \notin setOf t by fastforce`  
        - `have noneIn: \forall b. (i2,b) \notin setOf ?tr`  
          - `fix b`  
          - `from v binsert-set have setOf ?tr = setOf t - eqs fst (i,a) Un {(i,a)} by fastforce`  
          - `from this i2nei i2nin show (i2,b) ~: setOf ?tr by fastforce`  
          - `qed`
have mapset-none-inst:
valid-tmap ?tr --> (mapOf ?tr i2 = None) = (forall a. (i2, a) \notin setOf ?tr)
by (insert mapset-none [of ?tr i2], simp)
from vr noneIn mapset-none-inst have mapOf ?tr i2 = None by fastforce
from this upres mapNone show ?thesis by simp

next case (Some z) from this have mapSome: mapOf t i2 = Some z by simp
from v mapSome mapset-some have (i2, z) : setOf t by fastforce
from this setSpec i2nei have (i2, z) : setOf ?tr by (simp add: eqs-def)
from this vr mapset-some show ?thesis by simp
qed
from lhs-res rhs-res show ?thesis by simp
qed

lemma assumes v: valid-tmap t
shows mupdate-valid: valid-tmap (mupdate i a t)
proof -
let ?tr = binsert fst (i, a) t
have upres: mupdate i a t = ?tr by (simp add: mupdate-def)
from v binsert-sorted have vr: valid-tmap ?tr by fastforce
from vr upres show ?thesis by simp
qed

11 Map Remove Operation

definition mremove :: index => 'a tarray => 'a tarray where
mremove i t == remove fst (i, undefined) t

lemma assumes v: valid-tmap t
shows mremove-valid: valid-tmap (mremove i t)
proof (simp add: mremove-def)
from v remove-sort show sortedTree fst (remove fst (i, undefined) t) by fastforce
qed

lemma assumes v: valid-tmap t
shows mremove-map: mapOf (mremove i t) i = None
proof (simp add: mremove-def)
let ?tr = remove fst (i, undefined) t
show mapOf ?tr i = None
proof -
from v remove-spec have remSet: setOf ?tr = setOf t - eqs fst (i, undefined)
by fastforce
have noneIn: forall a. (i,a) \notin setOf ?tr
proof
fix a
from remSet show (i,a) ~: setOf ?tr by (simp add: eqs-def)
from v remove-sort have vr: valid-tmap ?tr by fastforce
have mapset-none-inst: valid-tmap ?tr ==>
(mapOf ?tr i = None) = (∀a. (i,a) ∉ setOf ?tr)
by (insert mapset-none [of ?tr i], simp)
from vr this have (mapOf ?tr i = None) = (∀a. (i,a) ∉ setOf ?tr) by fastforce
from this noneIn show mapOf ?tr i = None by simp
qed
qed
end

12 Tactic-Style Reasoning for Binary Tree Operations

theory BinaryTree-TacticStyle imports Main begin
This example theory illustrates automated proofs of correctness for binary tree operations using tactic-style reasoning. The current proofs for remove operation are by Tobias Nipkow, some modifications and the remaining tree operations are by Viktor Kuncak.

13 Definition of a sorted binary tree

datatype tree = Tip | Nd tree nat tree
primrec set-of :: tree => nat set
— The set of nodes stored in a tree.
where
  set-of Tip = {}
| set-of(Nd l x r) = set-of l Un set-of r Un {x}
primrec sorted :: tree => bool
— Tree is sorted
where
  sorted Tip = True
| sorted(Nd l y r) =
  (sorted l & sorted r & (∀x ∈ set-of l. x < y) & (∀z ∈ set-of r. y < z))

14 Tree Membership

primrec
  memb :: nat => tree => bool
where
  memb e Tip = False
| memb e (Nd t1 x t2) =
  (if e < x then memb e t1
     else memb e t2)
else if \( x < e \) then \( \text{memb} \ e \ t2 \) 
else \( \text{True} \)

\textbf{Lemma} \ member-set: \( \text{sorted} \ t \implies \text{memb} \ e \ t = (e : \text{set-of} \ t) \) 
by (induct \( t \)) auto

\section{15 Insertion operation}

\textbf{primrec} \ \textbf{bininsert} :: \( \text{nat} \Rightarrow \text{tree} \Rightarrow \text{tree} \) — Insert a node into sorted tree.
\textbf{where} 
\[ \text{bininsert} \ x \ \text{Tip} = (\text{Nd} \ \text{Tip} \ x \ \text{Tip}) \] 
| \( \text{bininsert} \ x \ (\text{Nd} \ t1 \ y \ t2) = (\text{if} \ x < y \ \text{then} \) 
& \( (\text{Nd} \ (\text{bininsert} \ x \ t1) \ y \ t2) \) 
| \( \text{else} \) 
& \( (\text{if} \ y < x \ \text{then} \) 
& \( (\text{Nd} \ t1 \ y \ (\text{bininsert} \ x \ t2)) \) 
| \( \text{else} \) 
& \( (\text{Nd} \ t1 \ y \ t2) \))

\textbf{Theorem} \ \text{set-of-bininsert} \ [\text{simp}]: \ \text{set-of} \ (\text{bininsert} \ x \ t) = \text{set-of} \ t \cup \{x\} 
by (induct \( t \)) auto

\textbf{Theorem} \ \text{bininsert-sorted}: \ \text{sorted} \ t \implies \text{sorted} \ (\text{bininsert} \ x \ t) 
by (induct \( t \)) (auto simp add: \text{set-of-bininsert})

\textbf{Corollary} \ \text{bininsert-spec}: 
\text{sorted} \ t \implies \text{sorted} \ (\text{bininsert} \ x \ t) \& 
\text{set-of} \ (\text{bininsert} \ x \ t) = \text{set-of} \ t \cup \{x\} 
by (simp add: \text{bininsert-sorted})

\section{16 Remove operation}

\textbf{primrec} \ \textbf{rm} :: \( \text{tree} \Rightarrow \text{nat} \) — find the rightmost element in the tree
\textbf{where} 
\[ \text{rm} (\text{Nd} \ l \ x \ r) = (\text{if} \ r = \text{Tip} \ \text{then} \ x \ \text{else} \ \text{rm} \ r) \]

\textbf{primrec} \ \textbf{rem} :: \( \text{tree} \Rightarrow \text{tree} \) — find the tree without the rightmost element
\textbf{where} 
\[ \text{rem} (\text{Nd} \ l \ x \ r) = (\text{if} \ r = \text{Tip} \ \text{then} \ l \ \text{else} \ \text{Nd} \ l \ x \ (\text{rem} \ r)) \]

\textbf{primrec} \ \textbf{remove} :: \( \text{nat} \Rightarrow \text{tree} \Rightarrow \text{tree} \) — remove a node from sorted tree
\textbf{where} 
\[ \text{remove} \ x \ \text{Tip} = \text{Tip} \] 
| \( \text{remove} \ x \ (\text{Nd} \ l \ y \ r) = \) 
& \( (\text{if} \ x < y \ \text{then} \ \text{Nd} \ (\text{remove} \ x \ l) \ y \ r \ \text{else} \)
if \( y < x \) then \( \text{Nd} \ l \ y \) \((\text{remove} \ x \ r)\) else
if \( l = \text{Tip} \) then \( r \)
else \( \text{Nd} \ (\text{rem} \ l) \ (\text{rm} \ l) \ r \)

**Lemma rm-in-set-of:** \( t \sim \text{Tip} \Longrightarrow \text{rm} \ : \text{set-of} \ t \)
by \((\text{induct} \ t)\) \text{auto}

**Lemma set-of:** \( t \sim \text{Tip} \Longrightarrow \text{set-of} \ t = \text{set-of} \ (\text{rem} \ t) \ \cup \ \{\text{rm} \ t\} \)
by \((\text{induct} \ t)\) \text{auto}

**Lemma [simp]:** \[ t \sim \text{Tip} \Longrightarrow \text{sorted} \ t \]
by \((\text{induct} \ t)\) \text{auto simp add: set-of-rem}

**Lemma sorted-rem:** \[ t \sim \text{Tip} \Longrightarrow \text{x} \in \text{set-of} \ (\text{rem} \ t); \text{sorted} \ t \Longrightarrow x < \text{rm} \ t \]
by \((\text{induct} \ t)\) \text{auto simp add: set-of-rem split: if-splits}

**Theorem set-of-remove [simp]:** \( \text{sorted} \ t \Longrightarrow \text{set-of} \ (\text{remove} \ x \ t) = \text{set-of} \ t - \{x\} \)
apply \((\text{induct} \ t)\)
apply simp
apply simp
apply (rule conjI)
apply fastforce
apply (rule impI)
apply (rule conjI)
apply fastforce
apply (fastforce simp: set-of-rem)
done

**Theorem remove-sorted:** \( \text{sorted} \ t \Longrightarrow \text{sorted} \ (\text{remove} \ x \ t) \)
by \((\text{induct} \ t)\) \text{auto intro: less-trans rm-in-set-of sorted-rem}

**Corollary remove-spec:** — summary specification of remove
\( \text{sorted} \ t \Longrightarrow \)
\[ \text{sorted} \ (\text{remove} \ x \ t) \& \]
\[ \text{set-of} \ (\text{remove} \ x \ t) = \text{set-of} \ t - \{x\} \]
by \((\text{simp add: remove-sorted})\)

Finally, note that rem and rm can be computed using a single tree traversal given by \( \text{rmrm} \).

**primrec** \( \text{rmrm} :: \text{tree} \Rightarrow \text{tree} \times \text{nat} \)
where
\[ \text{rmrm} \ (\text{Nd} \ l \ x \ r) = (\text{if} \ r = \text{Tip} \ \text{then} \ (l, x) \ \text{else} \)
\[ \text{let} \ (r', y) = \text{rmrm} \ r \ \text{in} \ (\text{Nd} \ l \ x \ r', y)) \]

**Lemma t \sim \text{Tip} \Longrightarrow \text{rmrm} \ t = (\text{rem} \ t, \text{rm} \ t) \)
by \((\text{induct} \ t)\) \text{auto simp: Let-def}

We can test this implementation by generating code.

**Definition** \( \text{test} = \text{memb} \\ 4 \ (\text{remove} \ (3::\text{nat}) \ (\text{binset} \\ 4 \ (\text{binset} \ 3 \text{Tip}))) \)

25
export-code test
  in SML module-name BinaryTree-TacticStyle-Code file BinaryTree-TacticStyle-Code.ML
end