# BinarySearchTree

Larry Paulson  
February 23, 2021

## Contents

1. Isar-style Reasoning for Binary Tree Operations  
2. Tree Definition  
3. Tree Lookup  
   3.1. Tree membership as a special case of lookup  
4. Insertion into a Tree  
5. Removing an element from a tree  
6. Mostly Isar-style Reasoning for Binary Tree Operations  
7. Map implementation and an abstraction function  
8. Auxiliary Properties of our Implementation  
   8.1. Lemmas mapset-none and mapset-some establish a relation between the set and map abstraction of the tree  
9. Empty Map  
10. Map Update Operation  
11. Map Remove Operation  
12. Tactic-Style Reasoning for Binary Tree Operations  
13. Definition of a sorted binary tree  
14. Tree Membership  
15. Insertion operation  
16. Remove operation
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 <
— 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 where
— 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) ---\to (\forall x \in setOf t. h x \sim k)
by (induct t, auto)

lemma tlookup-some:
   sortedTree h t & (tlookup h k t = Some x) ---\to 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
— No two elements have the same hash code
   sorted-distinct-pred h a b t == sortedTree h t &
   a: setOf t & b: setOf t & h a = h b ---\to
   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 [] \to 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)

3
assume \( h a h b ; h a = h b \)
from \( s \) have \( s1: \text{sortedTree} \ h \ t1 \) by auto
from \( s \) have \( s2: \text{sortedTree} \ h \ t2 \) by auto
show \( a = b \)

\[ \text{— We consider 9 cases for the position of } a \text{ and } b \text{ are in the tree} \]
proof
\[ \text{— three cases for } a \]
from \( a \text{def} \) have \( a : \text{setOf} \ t1 | a = x | a : \text{setOf} \ t2 \) by auto
moreover \{ assume \( \text{adef1: } a : \text{setOf} \ t1 \) have \( ?\text{thesis} \) proof
\[ \text{— three cases for } b \]
from \( b \text{def} \) have \( b : \text{setOf} \ t1 | b = x | b : \text{setOf} \ t2 \) by auto
moreover \{ assume \( \text{bdef1: } b : \text{setOf} \ t1 \) from \( s1 \) \( \text{adef1 bdef1 hahb h1 have } ?\text{thesis by simp} \) \}
moreover \{ assume \( \text{bdef1: } b = x \) from \( \text{adef1 bdef1 s have } h a < h b \) by auto
from this \( \text{hahb have } ?\text{thesis by simp} \) \}
moreover \{ assume \( \text{bdef1: } b : \text{setOf} \ t2 \) from \( \text{adef1 s have } o1: h a < h x \) by auto
from \( \text{bdef1 s have } o2: h x < h b \) by auto
from \( o1 \ o2 \text{ have } h a < h b \) by simp
from this \( \text{hahb have } ?\text{thesis by simp} \) — case impossible
ultimately show \( ?\text{thesis by blast} \)
qed
\}
moreover \{ assume \( \text{adef1: } a = x \) have \( ?\text{thesis} \) proof
\[ \text{— three cases for } b \]
from \( \text{bdef} \) have \( b : \text{setOf} \ t1 | b = x | b : \text{setOf} \ t2 \) by auto
moreover \{ assume \( \text{bdef1: } b : \text{setOf} \ t1 \) from this \( \text{s have } o1: h a < h x \) by auto
from this \( \text{adef1 have } h b < h a \) by auto
from \( \text{hahb this have } ?\text{thesis by simp} \) — case impossible
moreover \{ assume \( \text{bdef1: } b = x \) from \( \text{adef1 bdef1 have } ?\text{thesis by simp} \) \}
moreover \{ assume \( \text{bdef1: } b : \text{setOf} \ t2 \) from this \( \text{s have } o2: h x < h b \) by auto
from this \( \text{adef1 have } h a < h b \) by simp
from \( \text{hahb this have } ?\text{thesis by simp} \) — case impossible
ultimately show \( ?\text{thesis by blast} \)
qed
\}
moreover \{ assume \( \text{adef1: } a : \text{setOf} \ t2 \) have \( ?\text{thesis} \) proof
\[ \text{— three cases for } b \]
from \( \text{bdef} \) 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 adef1 s have o2: h x < h a by auto
  from o1 o2 have h b < h a by simp
  from this hahb 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 } — case impossible
moreover { assume bdef1: b : setOf t2
  from s2 adef1 bdef1 hahb h2 have ?thesis by simp
}
ultimately show ?thesis by blast
qed

ultimately show ?thesis by blast
qed

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: \forall 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 t x =
    (case (tlookup h (h x) t) of
      None => False
      Some y => y = x)
Some \( z \geq (x = z) \)

**Lemma** assumes \( s \): sortedTree \( h \; t \)

**Shows** memb-spec: \( \text{memb} \; h \; x \; t = (x : \text{setOf} \; t) \)

**Proof** (cases \( \text{lookup} \; h \; (h \; x) \; t \))

**Case** None note tNone = this

- from tNone have res: \( \text{memb} \; h \; x \; t = \text{False} \) by (simp add: memb-def)

- from s tNone lookup-none have o1: \( \forall \; y \in \text{setOf} \; t. \; h \; y \sim h \; x \) by fastforce

- have notIn: \( x \sim : \text{setOf} \; t \)

  **Proof**

  - assume \( h: \; x : \text{setOf} \; t \)

  - from h o1 have \( h \; x = h \; x \) 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 lookup-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: \( \text{memb} \; h \; x \; t \) by (simp add: memb-def)

  - from res zin xez show \( \text{thesis} \) by simp

  **Next case** False note xnez = this

  - from tSome xnez have res: \( \neg \; \text{memb} \; h \; x \; t \) by (simp add: memb-def)

  - have \( x \sim : \text{setOf} \; t \)

    **Proof**

    - assume \( xin: \; x : \text{setOf} \; t \)

    - from s tSome lookup-some have hzhx: \( h \; x = h \; z \) by fastforce

    - have o1: \( \text{sorted-distinct-pred} \; h \; x \; z \; t \)

      by (insert sorted-distinct [of h x z t], simp)

    - from s xin zin hzhx o1 have \( x = z \) by fastforce

    - from this xnez show False by simp

  - qed

  - from this res show \( \text{thesis} \) by simp

- qed

- declare sorted-distinct-pred-def [simp del]

**4 Insertion into a Tree**

primrec

\( \text{binsert} \; :: \; 'a \; \text{hash} \Rightarrow \; 'a \Rightarrow \; 'a \; \text{Tree} \Rightarrow \; 'a \; \text{Tree} \)

**Where**

\[
\text{binsert} \; h \; e \; \text{Tip} = (T \; \text{Tip} \; e \; \text{Tip})
\]

| \( \text{binsert} \; h \; e \; (T \; t1 \; x \; t2) = \) (if \( h \; e < h \; x \) then \\
(\( T \; (\text{binsert} \; h \; e \; t1 \) \) \; t2) \\
else \\
(\( h \; x < h \; e \) then

6
A technique for proving disjointness of sets.

**lemma** disjCond: \[ | x. x:A; x:B | \Rightarrow \] False \[ ] \Rightarrow 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: \( \text{sortedTree } h \: t \: \Rightarrow \) 

\[
\text{setOf } (\text{binsert } h \: e \: t) = (\text{setOf } t) - (\text{eqs } h \: e) \: \text{Un} \: \{ e \}
\]

**proof** (induct t)

— base case

show \( ?P \: \text{Tip} \) by (simp add: eqs-def)

— induction step

fix \( t1 \:: 'a \: \text{Tree} \) assume \( h1 \:: ?P \: t1 \)

fix \( t2 :: 'a \: \text{Tree} \) 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 (rule sortLemmaL)

from \( s1 \) and \( h1 \) have \( c1 :: \text{setOf } (\text{binsert } h \: e \: t1) = \text{setOf } t1 - (\text{eqs } h \: e) \: \text{Un} \: \{ e \} \)

by simp

from \( s \) have \( s2 :: \text{sortedTree } h \: t2 \) by (rule sortLemmaR)

from \( s2 \) and \( h2 \) have \( c2 :: \text{setOf } (\text{binsert } h \: e \: t2) = \text{setOf } t2 - (\text{eqs } h \: e) \: \text{Un} \: \{ e \} \)

by simp

show \( \text{setOf } (\text{binsert } h \: e \: (T \: t1 \: x \: t2)) = \text{setOf } (T \: t1 \: x \: t2) - (\text{eqs } h \: e) \: \text{Un} \: \{ e \} \)

proof (cases \( h \: e \: < \: h \: x \))

case True note eLess = this

from \( eLess \) have \( \text{res} :: \text{binsert } h \: e \: (T \: t1 \: x \: t2) = (T \: \text{binsert } h \: e \: t1) \: x \: t2) \) by simp

show \( \text{setOf } (\text{binsert } h \: e \: (T \: t1 \: x \: t2)) = \text{setOf } (T \: t1 \: x \: t2) - (\text{eqs } h \: e) \: \text{Un} \: \{ e \} \)

proof (simp add: res eLess c1)

show \( \text{insert } x \: (\text{insert } e \: (\text{setOf } t1 - (\text{eqs } h \: e \: \text{Un} \: \text{setOf } t2)) = \text{insert } e \: (\text{insert } x \: (\text{setOf } t1 \: \text{Un} \: \text{setOf } t2) - (\text{eqs } h \: e) \)

proof –

have \( \text{eqsLessX} :: \forall \: el \: \in \: \text{eqs } h \: e. \: h \: el \: < \: h \: x \) by (simp add: eqs-def eLess)

from \( \text{this} \) have \( \text{eqsDisjX} :: \forall \: el \: \in \: \text{eqs } h \: e. \: h \: el \: \not\leq \: h \: x \) by fastforce

from \( s \) have \( \text{xLessT2} :: \forall \: r \: \in \: \text{setOf } t2. \: h \: x \: < \: h \: r \) by auto

have \( \text{eqsLessT2} :: \forall \: el \: \in \: \text{eqs } h \: e. \: \forall \: r \: \in \: \text{setOf } t2. \: h \: el \: < \: h \: r \)

proof safe

fix \( el \) assume \( hel :: el \: \in \: \text{eqs } h \: e \)

from \( hel \) eqs-def have \( @f :: h \: el \: = \: h \: e \) by fastforce

fix \( r \) assume \( hr :: r \: \in \: \text{setOf } t2 \)

7
from \(x\text{LessT2}\) \(hr\) \(o1\) \(c\text{Less}\) show \(h \, el \; < \; h \, r\) by \(auto\)
qed
from \(eqs\text{LessT2}\) have \(eqs\text{DisjT2}\) \(\forall\) \(el \in eqs\, h\, e\). \(\forall\) \(r \in setOf\, t2\). \(h \, el \; \sim\; =\) \(h\, r\)
by \(fastforce\)
from \(eqs\text{DisjX}\) \(eqs\text{DisjT2}\) show \(\negthesis\) by \(fastforce\)
qed
next case \(False\) note \(e\text{NotLess} = this\)
show \(setOf\ (bin\, sert\ h\, e\; (T\; t1\; x\; t2)) = setOf\; (T\; t1\; x\; t2) = eqs\, h\, e\; Un\; \{e\}\)
proof (cases \(h\, x\; <\; h\, e\))
\begin{itemize}
  \item case \(True\) note \(x\text{Less} = this\)
  from \(x\text{Less}\) have \(res:\; bin\, sert\ h\, e\; (T\; t1\; x\; t2) = (T\; t1\; x\; (bin\, sert\ h\, e\; t2))\) by \(simp\)
  show \(setOf\; (bin\, sert\ h\, e\; (T\; t1\; x\; t2)) = setOf\; (T\; t1\; x\; t2) = eqs\, h\, e\; Un\; \{e\}\)
proof (simp add: \(x\text{Less} e\text{NotLess} c2\))
\begin{itemize}
  \item have \(X\text{LessEqs}\) \(\forall\) \(el \in eqs\, h\, e\). \(h\, x\; <\; h\, el\) by (simp add: \(eqs\text{-def xLess}\))
  from this have \(eqs\text{DisjX}\) \(\forall\) \(el \in eqs\, h\, e\). \(h\, el\) \(\sim\; =\) \(h\, x\) by \(auto\)
  from s have \(t1\text{LessX}\) \(\forall\) \(l \in setOf\, t1\). \(h\, l\; <\; h\, x\) by \(auto\)
  have \(T1\text{lessEqs}\) \(\forall\) \(el \in eqs\, h\, e\). \(\forall\) \(l \in setOf\, t1\). \(h\, l\; <\; h\, el\)
proof safe
  fix \(el\) assume \(hel\): \(el\) \(\in eqs\, h\, e\)
  fix \(l\) assume \(hl\): \(l\) \(\in setOf\, t1\)
  from \(hel\) \(eqs\text{-def}\) have \(o1\): \(h\, el\) \(\sim\) \(h\, e\) by \(fastforce\)
  from \(t1\text{lessX hl o1 xLess show h l < h e}\) by \(auto\)
  qed
  from \(T1lessEqs\) have \(T1\text{disjEqs}\) \(\forall\) \(el \in eqs\, h\, e\). \(\forall\) \(l \in setOf\, t1\). \(h\, el\) \(\sim\; =\) \(h\, l\)
  by \(fastforce\)
  from \(eqs\text{DisjX}\) \(T1lessEqs\) show \(\negthesis\) by \(auto\)
  qed
  qed
next case \(False\) note \(x\text{NotLess} = this\)
from \(x\text{NotLess eNotLess}\) have \(xe\text{qe}\): \(h\, x\; =\; h\, e\) by \(simp\)
from \(xe\text{qe}\) have \(res:\; bin\, sert\ h\, e\; (T\; t1\; x\; t2) = (T\; t1\; e\; t2)\) by \(simp\)
show \(setOf\; (bin\, sert\ h\, e\; (T\; t1\; x\; t2)) = setOf\; (T\; t1\; x\; t2) = eqs\, h\, e\; Un\; \{e\}\)
proof (simp add: \(res\, e\text{NotLess}\, xe\text{qe}\))
show \(insert\; e\; (setOf\, t1\; Un\, setOf\, t2) = insert\; e\; (insert\; x\; (setOf\, t1\; Un\, setOf\, t2) = eqs\, h\, e)\)
proof –
\begin{itemize}
  \item have \(insert\; x\; (setOf\, t1\; Un\, setOf\, t2) = eqs\, h\, e\)
  = \(setOf\, t1\; Un\, setOf\, t2\)
  proof –
  \begin{itemize}
    \item have \(x\): \(eqs\, h\, e\) by (simp add: \(eqs\text{-def xe}\))
moreover have \((\text{setOf } t1) \cap \text{Int} (\text{eqs } h \ e) = \{\}\)

proof (rule disjCond)
  fix \(w\)
  assume \(\text{whSet}: \ w : \text{setOf } t1\)
  assume \(\text{whEq}: \ w : \text{eqs } h \ e\)
  from \(\text{whSet} \ s\) have \(\alpha1: \ h \ w < h \ x\) by simp
  from \(\text{whEq} \ \text{eqs-def}\) have \(\alpha2: \ h \ w = h \ e\) by fastforce
  from \(\alpha2 \ \text{xeq}\) have \(\alpha3: \sim h \ w < h \ x\) by simp
  from \(\alpha1 \ \alpha3\) show False by contradiction
  qed

moreover have \((\text{setOf } t2) \cap \text{Int} (\text{eqs } h \ e) = \{\}\)

proof (rule disjCond)
  fix \(w\)
  assume \(\text{whSet}: \ w : \text{setOf } t2\)
  assume \(\text{whEq}: \ w : \text{eqs } h \ e\)
  from \(\text{whSet} \ s\) have \(\alpha1: \ h \ x < h \ w\) by simp
  from \(\text{whEq} \ \text{eqs-def}\) have \(\alpha2: \ h \ w = h \ e\) by fastforce
  from \(\alpha2 \ \text{xeq}\) have \(\alpha3: \sim h \ x < h \ w\) by simp
  from \(\alpha1 \ \alpha3\) show False by contradiction
  qed

ultimately show \(?thesis\) by auto
  qed

from this show \(?thesis\) by simp
  qed
  qed
  qed
  qed
  qed
  qed

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

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

We summarize the specification of binsert as follows.

corollary \(\text{binsert-spec}: \text{sortedTree } h \ t \rightarrow \rightarrow \)
  \(\text{sortedTree } h \ (\text{binsert } h \ x \ t)\) \&
  \(\text{setOf } (\text{binsert } h \ e \ t) = (\text{setOf } t) \ - \ (\text{eqs } h \ e) \ \text{Un} \ \{e\}\)
  by (simp add: \(\text{binsert-set} \ \text{binsert-sorted}\))

5 Removing an element from a tree

These proofs are influenced by those in \(\text{BinaryTree-Tactic}\)

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

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

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

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

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

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

\text{lemma wrm-set: } t \sim = \text{Tip} & \text{sortedTree } h \ t \longrightarrow \text{setOf (wrm } h \ t) = \text{setOf } t - \{ \text{rm } h \ t\} \ \text{(is } \ ?P \ t)
\text{proof (induct } t) \]
\text{show } ?P \ Tip \ \text{by simp}
\text{fix } t1 :: 'a Tree \ \text{assume } h1: ?P \ t1
\text{fix } t2 :: 'a Tree \ \text{assume } h2: ?P \ t2
\text{fix } x :: 'a
show \(\varphi (T \ t1 \ x2)\)
proof (rule \text{impI}, crule \text{conjE})
assume \(s\): \text{sortedTree} \(h\) (\(T \ t1 \ x2\))
show \(\text{setOf} \ (\text{wrm} \ h \ (T \ t1 \ x2)) = \text{setOf} \ (T \ t1 \ x2) - \{\text{rm} \ h \ (T \ t1 \ x2)\}\)
proof (cases \(t2 = \text{Tip}\))
case True
note \(t2tip = \text{this}\)
from \(t2tip\) have \(\text{rm-res}\): \(\text{rm} \ h \ (T \ t1 \ x2) = x\) by simp
from \(t2tip\) have \(\text{wrm-res}\): \(\text{wrm} \ h \ (T \ t1 \ x2) = t1\) by simp
from \(s\) have \(x \sim\): \(\text{setOf} \ t1\) by auto
from \(t2tip\) \(\text{rm-res}\) \(\text{wrm-res}\) \(t2tip\) show \(\text{thesis}\) by simp
next case False
note \(t2nTip = \text{this}\)
from \(t2nTip\) have \(\text{rm-res}\): \(\text{rm} \ h \ (T \ t1 \ x2) = \text{rm} \ h \ t2\) by simp
from \(t2nTip\) have \(\text{wrm-res}\): \(\text{wrm} \ h \ (T \ t1 \ x2) = T \ t1 \ x (\text{wrm} \ h \ t2)\) by simp
from \(s\) have \(s2\): \(\text{sortedTree} \ h \ t2\) by simp
have \(\text{o1}\): \(\text{setOf} \ (\text{wrm} \ h \ t2) = \text{setOf} \ t2 - \{\text{rm} \ h \ t2\}\) by simp
show \(\text{thesis}\)
proof (simp add: \(\text{rm-res} \ \text{wrm-res} \ t2nTip \ h2 \ o1\))
show \(\text{insert} \ x \ (\text{setOf} \ t1 \ Un \ (\text{setOf} \ t2 - \{\text{rm} \ h \ t2\})) = \text{insert} \ x \ (\text{setOf} \ t1 \ Un \ \text{setOf} \ t2) - \{\text{rm} \ h \ t2\}\)
proof
from \(s\) \(\text{rm-set} \ t2nTip\) have \(\text{xoK}\): \(h \ x < h \ (\text{rm} \ h \ t2)\) by auto
have \(\text{t1Ok}\): \(\forall l \in \text{setOf} \ t1. \ h \ l < h \ (\text{rm} \ h \ t2)\)
proof safe
fix l :: 'a assume \(ldef\): \(l \in \text{setOf} \ t1\)
from \(ldef\) \(s\) have \(\text{lx}\): \(h \ l < h \ x\) by auto
from \(lx\) \(\text{xoK}\) show \(h \ l < h \ (\text{rm} \ h \ t2)\) by auto
qed
from \(\text{xoK} \ \text{t1Ok}\) show \(\text{thesis}\) by auto
qed
qed
qed
qed

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

lemma \(\text{wrm-sort}\): \(t \sim = \text{Tip} \ & \ \text{sortedTree} \ h \ t \implies \text{sortedTree} \ h \ (\text{wrm} \ h \ t)\) (is \(\varphi \ t\))
proof (induct \(t\))
show \(\varphi \ \text{Tip}\) by simp
fix \(t1 :: 'a \ Tree\) assume \(h1:: \?P \ t1\)
fix \(t2 :: 'a \ Tree\) assume \(h2:: \?P \ t2\)
fix \(x :: 'a\)
show \(\varphi \ (T \ t1 \ x2)\)
proof safe
assume \(s:: \text{sortedTree} \ h \ (T \ t1 \ x2)\)

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 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 o3: ∀ r ∈ setOf (wrm h t2). h x < h r by auto
  from s1 o1 o3 res s show sortedTree h (wrm h (T t1 x t2)) by simp
qed
qed

lemma wrm-less-rm:
  t ~= Tip & sortedTree h t --->
  (∀ l ∈ 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 t2nTip have wrm-res: wrm h (T t1 x t2) = T t1 x (wrm 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 lnx = this
  qed
qed
show \(?thesis\)
proof (cases l : setOf t1)
case True note l-in-t1 = this
  from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
  from l-in-t1 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 l-scope hwz l-notin-t1
  have l-in-res: l : setOf (wrm h t2) by auto
  from l-in-res h2 t2nTip s2 show \(?thesis\) by auto
qed
qed from rm-res hLess show \(?thesis\) by simp qed qed

lemma remove-set: sortedTree h t --->
  setOf (remove h e t) = setOf t - eqs h e (is \(?P\) t)
proof (induct t)
  show \(?P\) 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: sortedTree h (T t1 x t2)
    show setOf (remove h e (T t1 x t2)) = setOf (T t1 x t2) - eqs h e
      proof (cases h e < h x)
      case True note elx = this
      from elx have res: remove h e (T t1 x t2) = T (remove h e t1) x t2
        by simp
      from s have s1: sortedTree h t1 by simp
      from s1 h1 have o1: setOf (remove h e t1) = setOf t1 - eqs h e by simp
      show \(?thesis\)
      proof (simp add: o1 elx)
        show insert x (setOf t1 - eqs h e Un setOf t2) =
          insert x (setOf t1 Un setOf t2) - eqs h e
          proof
          have xOk: x : eqs h e
          proof
            assume h: x : eqs h e
            from h have o1: ~ (h e < h x) by (simp add: eqs-def)
            from elx o1 show False by contradiction
          qed
          have t2Ok: (setOf t2) Int (eqs h e) = {}
          proof (rule disjCond)
            fix y :: 'a
            assume y-in-t2: y : setOf t2
          qed
        qed
    qed
  qed
qed
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 elx 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 (simp add: o1 xle nelx)
show insert x (setOf t1 Un setOf t2 - eqs h e)) =
insert x (setOf t1 Un setOf t2) - eqs h e
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 nxle: ~ (h x < h e) by simp
  from nxle xle show False by contradiction
qed
from xOk t1Ok show ?thesis by auto
qed
qed
next case False note nxle = this
from nxle 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 y = h e by (simp add: eqs-def)
from \( y \)-in-eq \( ex \) \( eeq \) have \( neeqy \colon \sim (h \ x < h \ y) \) by simp
from \( neeqx \) \( xly \) show \( False \) by contradiction
qed
show \( ?thesis \)
proof (cases \( t1 = \text{Tip} \))
case \( True \) note \( t1t = \text{this} \)
from \( ex \) \( t1t \) have \( \text{res: remove h e (T t1 x t2) = t2 by simp} \)
show \( ?thesis \)
proof (simp add: \( \text{res t1t ex} \))
show \( \text{setOf t2 = insert x (setOf t2) - eqs h e} \)
proof -
from \( ex \) have \( x\text{-in-eqs: x : eqs h e by (simp add: eqs-def)} \)
from \( x\text{-in-eqs t2Ok show \( ?thesis \) by auto} \)
qed
qed
next case \( False \) note \( t1nT = \text{this} \)
from \( nle xle ex \) \( t1nT \)
have \( \text{res: remove h e (T t1 x t2) = T (wrm h t1) (rm h t1) t2} \)
by (simp add: \( \text{Let-def wrm-rm-decomp} \))
from \( \text{res show \( ?thesis \)} \)
proof simp
from \( s \) have \( s1: \text{sortedTree h tI by simp} \)
show \( \text{insert (rm h tI) (setOf (wrm h tI) Un setOf t2) = insert x (setOf tI Un setOf t2) - eqs h e} \)
proof (simp add: \( t1nT s1 \) \( \text{rm-set wrm-set} \))
show \( \text{insert (rm h tI) (setOf tI - \{ rm h tI \} Un setOf t2) = insert x (setOf tI Un setOf t2) - eqs h e} \)
proof -
from \( t1nT s1 \) \( \text{rm-set} \)
have \( o1: \text{insert (rm h tI) (setOf tI - \{ rm h tI \} Un setOf t2) = setOf tI Un setOf t2 by auto} \)
have \( o2: \text{insert x (setOf tI Un setOf t2) - eqs h e = setOf tI Un setOf t2} \)
proof -
from \( ex \) have \( xOk: x : eqs h e by (simp add: eqs-def) \)
have \( t1Ok: (setOf tI) \text{ Int (eqs h e) = \{}} \)
proof (rule disjCond)
fix \( y :: 'a \)
assume \( y\text{-in-tI: y : setOf tI} \)
assume \( y\text{-in-eq: y : eqs h e} \)
from \( y\text{-in-tI s ex have o1: h y < h e by auto} \)
from \( y\text{-in-eq have o2: \sim (h y < h e) by (simp add: eqs-def)} \)
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
lemma remove-sort: sortedTree h t -->
  sortedTree h (remove h e t) (is ?P t)
proof (induct t)
  show ?P 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: sortedTree h (T t1 x t2)
    from s have s1: sortedTree h t1 by simp
    from s have s2: sortedTree h t2 by simp
    from h1 s1 have sr1: sortedTree h (remove h e t1) by simp
    from h2 s2 have sr2: sortedTree h (remove h e t2) by simp
    show sortedTree h (remove h e (T t1 x t2))
    proof (cases h e < h x)
      case True note elx = this
      from elx have res: remove h e (T t1 x t2) = T (remove h e t1) x t2 by simp
      show ?thesis
      proof (simp add: sr1 s2 elx res)
        let ?C1 = \forall l \in setOf (remove h e t1). h l < h x
        let ?C2 = \forall r \in setOf t2. h x < h r
        have o1: ?C1
        proof
          from s1 have setOf (remove h e t1) = setOf t1 - eqs h e by (simp add: remove-set)
          from s this show ?thesis by auto
          qed
          from o1 s show ?C1 & ?C2 by auto
          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
        show ?thesis
        proof (simp add: s s1 sr2 xle nelx res)
          let ?C1 = \forall l \in setOf t1. h l < h x
          let ?C2 = \forall r \in setOf (remove h e t2). h x < h r
          have o2: ?C2
proof  
  from s2 have setOf (remove h e t2) = setOf t2 - eqs h e by (simp add: remove-set) 
  from s this show ?thesis by auto 
qed

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

We summarize the specification of remove as follows.

**corollary** remove-spec: sortedTree h t ---> 
  sortedTree h (remove h e t) & 
setOf (remove h e t) = setOf t - eqs h e 
by (simp add: remove-sort remove-set)
definition test = tlookup id 4 (remove id 3 (bininsert id 4 (bininsert 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)

18
case None from this show \textit{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 \sim\ = None by simp
  from this h show \textit{thesis} by simp — contradiction
qed

lemma assumes v: valid-tmap t
  assumes h: mapOf t i = Some a
  shows mapset-some: 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 \textit{thesis} by simp — contradiction
  next case (Some ia)
    note tsome = this
    have tlookup-some-inst: sortedTree fst t & (tlookup fst i t = Some ia) \implies ia : setOf t & fst ia = i by (simp add: tlookup-some)
    from tlookup-some-inst tsome v have ia : setOf t by simp
    from tsome 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 isome 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) = (\forall 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 \forall a. (i,a) \notin setOf t
  proof
    fix a
    show (i,a) \sim\ : setOf t
    proof
      assume iain: (i,a) : setOf t
      have tlookup-none-inst:
        sortedTree fst t & (tlookup fst i t = None) \implies (\forall x \in setOf t. fst x \sim\ = i)
        by (insert tlookup-none [of fst t i], assumption)
      from v lnone tlookup-none-inst have \forall x \in setOf t. fst x \sim\ = i by simp
      from this iain have fst (i,a) \sim\ = i by fastforce
      from this show False by simp
    qed
qed
**9 Empty Map**

**Lemma mnew-spec-valid:** valid-tmap Tip

by (simp add: mapOf-def)

**Lemma mtip-spec-empty:** mapOf Tip k = None

by (simp add: mapOf-def)

**10 Map Update Operation**

**Definition mapupdate ::** index => 'a => 'a tarray => 'a tarray where
mupdate i a t == binset 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 = binset fst (i,a) t
  have upres: mupdate i a t = ?tr by (simp add: mupdate-def)
  from v binset-set
  have setSpec: setOf ?tr = setOf t - eqs fst (i,a) Un {i,a} by fastforce
  from v binset-sorted have vr: valid-tmap ?tr by fastforce
  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
        from setSpec have kvp: (i,a) : setOf ?tr by simp
        have binset-sorted-inst: sortedTree fst t --> sortedTree fst ?tr by (insert binset-sorted [of fst t (i,a)], assumption)
        from v binset-sorted-inst have rs: sortedTree fst ?tr by simp
        have tlookup-finds-inst: sortedTree fst ?tr & (i,a) : setOf ?tr --> tlookup fst i ?tr = Some (i,a)
          by (insert tlookup-finds [of fst ?tr (i,a)], simp)
        from rs kvp tlookup-finds-inst show ?thesis by simp
      qed
      from upres will-find: tlookup fst i ?tr = Some (i,a)
      proof
        from setSpec have kvp: (i,a) : setOf ?tr by simp
        have binset-sorted-inst: sortedTree fst t --> sortedTree fst ?tr
          by (insert binset-sorted [of fst t (i,a)], assumption)
        from v binset-sorted-inst have rs: sortedTree fst ?tr by simp
        have tlookup-finds-inst: sortedTree fst ?tr & (i,a) : setOf ?tr --> tlookup fst i ?tr = Some (i,a)
          by (insert tlookup-finds [of fst ?tr (i,a)], simp)
        from rs kvp tlookup-finds-inst show ?thesis by simp
      qed
    from lhs-res rhs-res i2ei show ?thesis by simp
  qed
  from v binset-set
  have setSpec: setOf ?tr = setOf t - eqs fst (i,a) Un {i,a} by fastforce
  from this i2nei i2nin show (i2,b) ~: setOf ?tr
  proof
    fix b
    from v binset-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 i by fastforce
from this setSpec i2nei have (i2, z) : setOf ?tr by (simp add: eqs-def)
from this vr mapset-some have mapOf ?tr i2 = Some z by fastforce
from this upres mapSome show ?thesis by simp
qed
from lhs-res rhs-res show ?thesis by simp
qed
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: ∀ a. (i, a) ∉ setOf ?tr
proof
  fix a
  from remSet show (i, a) ~: setOf ?tr by (simp add: eqs-def)
qed
  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
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 if x < e then memb e t2
   else True)

23
lemma member-set: sorted t ---\rightarrow memb e t = (e : set-of t)
by (induct t) auto

15 Insertion operation

primrec binsert :: nat => tree => tree
— Insert a node into sorted tree.
where
  binsert x Tip = (Nd Tip x Tip)
| binsert x (Nd t1 y t2) = (if x < y then
   (Nd (binsert x t1) y t2)
   else
   (if y < x then
    (Nd t1 y (binsert x t2))
   else (Nd t1 y t2))

theorem set-of-binsert [simp]: set-of (binsert x t) = set-of t Un \{x\}
by (induct t) auto

theorem binsert-sorted: sorted t ---\rightarrow sorted (binsert x t)
by (induct t) (auto simp add: set-of-binsert)

corollary binsert-spec:
  sorted t ---\rightarrow
  sorted (binsert x t) &
  set-of (binsert x t) = set-of t Un \{x\}
by (simp add: binsert-sorted)

16 Remove operation

primrec
  rm :: tree => nat — find the rightmost element in the tree
where
  rm(Nd l x r) = (if r = Tip then x else rm r)

primrec
  rem :: tree => tree — find the tree without the rightmost element
where
  rem(Nd l x r) = (if r=Tip then l else Nd l x (rem r))

primrec
  remove :: nat => tree => tree — remove a node from sorted tree
where
  remove x Tip = Tip
| remove x (Nd l y r) =
  (if x < y then Nd (remove x l) y r else
   if y < x then Nd l y (remove x r) else
   if l = Tip then r
   else Nd (rem l) (rm l) r)
lemma rm-in-set-of: $t \sim \text{Tip} \Longrightarrow \text{rm } t : \text{set-of } t$
by (induct t) auto

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

lemma [simp]: [| $t \sim \text{Tip}; \text{sorted } t \|] \Longrightarrow \text{sorted}(\text{rem } t)$
by (induct t) (auto simp add:set-of-rem)

lemma sorted-rem: [| $t \sim \text{Tip}; \; x \in \text{set-of}(\text{rem } t); \; \text{sorted } t \|] \Longrightarrow x < \text{rm } t$
by (induct t) (auto simp add: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(induct t)
apply simp
apply simp
apply (rule conjI)
apply fastforce
apply (rule impl)
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 (induct t) (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 (simp add:remove-sorted)

Finally, note that rem and rm can be computed using a single tree traversal
given by remrm.

primrec remrm :: tree => tree * nat
where
remrm(Nd l x r) = (if r=\text{Tip} then (l,x) else
let (r',y) = remrm r in (Nd l x r',y))

lemma $t \sim \text{Tip} \Longrightarrow \text{remrm } t = (\text{rem } t, \text{rm } t)$
by (induct t) (auto simp:Let-def)

We can test this implementation by generating code.

definition test = memb 4 (remove (3::nat) (binsert 4 (binsert 3 \text{Tip})))

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