BinarySearchTree

Larry Paulson

June 11, 2019

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

class 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 &
   (∀l ∈ setOf t1. h l < h x) &
   (∀r ∈ 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) --- (\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) --- 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 --- 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 \( h_{ab} : h \ a = h \ b \)

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 \( s1 adef1 bdef1 h_{ab} h1 \) have \( \text{thesis} \) by simp \}

moreover \{ assume \( bdef1 : b = x \)

from \( adef1 bdef1 s \) have \( h \ a < h \ b \) by auto

from this \( h_{ab} \) 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 \( h_{ab} \) 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 this \( adef1 \) have \( h \ b < h \ a \) by auto

from \( h_{ab} \) this have \( \text{thesis} \) by simp \} — case impossible

moreover \{ assume \( bdef1 : b = x \)

from \( adef1 bdef1 \) have \( \text{thesis} \) by simp \}

moreover \{ assume \( bdef1 : b : \text{setOf} \ t2 \)

from this \( s \) have \( h \ x < h \ b \) by auto

from this \( adef1 \) have \( h \ a < h \ b \) by simp

from \( h_{ab} \) this have \( \text{thesis} \) by simp \} — case impossible

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 a1: h b < h x by auto
  from adef1 s have a2: h x < h a by auto
  from a1 a2 have h b < h a by simp
  from this habb 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

moreover { assume bdef1: b : setOf t2
  from s2 adef1 bdef1 hahb h2 have ?thesis by simp
}
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 a1: \forall x \in setOf t. h x = h y by (simp add: tlookup-none)
    from a1 yint have h y = h y by fastforce
    from this show ?thesis by simp
  next case (Some z) note res = this
    have ls: 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 ls have a1: z: setOf t & h z = h y by simp
    from s yint a1 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 \( s : \text{sortedTree} \ h \ t \)
shows \( \text{memb-spec: memb} \ h \ x \ t = (x : \text{setOf} \ t) \)
proof (cases \( \text{tlookup} \ h \ (h \ x) \))
case None note tNone = this
from tNone have res: \( \text{memb} \ h \ x \ t = \text{False} \) by (simp add: memb-def)
from s tNone tlookup-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 \( h \ x \sim = h \ x \) 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: \( \text{memb} \ h \ x \ t \) by (simp add: memb-def)
from res zin xez show \( \?\text{thesis} \) by simp
next case False note znez = this
from tSome xnez have res: \( \sim \text{memb} \ h \ x \ t \) by (simp add: memb-def)
have \( x \sim : \text{setOf} \ t \)
proof
assume xin: \( x : \text{setOf} \ t \)
from s tSome tlookup-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 znez show False by simp
qed
from this res show \( \?\text{thesis} \) by simp
qed
qed

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

4 Insertion into a Tree

primrec
\( \text{binsert} :: \tau \text{ hash} => \tau => \tau \text{ Tree} => \tau \text{ Tree} \)
where
\( \text{binsert} \ h \ c \ \text{Tip} = (T \ \text{Tip} \ c \ \text{Tip}) \)
| \( \text{binsert} \ h \ c \ (T \ t1 \ t2) = (\text{if} \ h \ c < h \ x \ \text{then} \ (T \ (\text{binsert} \ h \ c \ t1) \ t2) \ \text{else} \ (\text{if} \ h \ x < h \ c \ \text{then} \)
(T t1 x (binsert h e t2))
else (T t1 e t2))

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}

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: \( \forall el \in eqs h e. h e l < h x \) by (simp add: eqs-def eLess)
from this have eqsDisjX: \( \forall el \in eqs h e. h e l = h x \) by fastforce
from s have xLessT2: \( \forall r \in setOf t2. h x < h r \) by auto
have eqsLessT2: \( \forall el \in eqs h e. \forall r \in setOf t2. h x < h r \)
proof safe
fix el assume hel: el : eqs h e
from hel eqs-def have a1: h e l = h e by fastforce
fix r assume hr: r : setOf t2
from xLessT2 hr o1 cLess show h el < h r by auto
qed
from eqsLessT2 have eqsDisjT2: \( \forall el \in eqs h e. \forall r \in setOf t2. h el \sim h r \)
by fastforce
from eqsDisjX eqsDisjT2 show ?thesis by fastforce
qed
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: \( \forall el \in eqs h e. h x < h el \) by (simp add: eqs-def xLess)
  from this have eqsDisjX: \( \forall el \in eqs h e. h el \sim h x \) by auto
  from s have t1LessX: \( \forall l \in setOf t1. h l < h x \) by auto
  have T1lessEqs: \( \forall el \in eqs h e. \forall l \in setOf t1. h l < h el \)
  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: \( \forall el \in eqs h e. \forall l \in setOf t1. h el \sim h l \)
  by fastforce
  from eqsDisjX T1lessEqs show ?thesis by auto
  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 \((\text{setOf } t1) \cap \text{Int } (\text{eqs } h e) = \{\}\)

proof (rule disjCond)
fix \(w\)
assume \(\text{whSet}: w : \text{setOf } t1\)
assume \(\text{whEq}: w : \text{eqs } h e\)
from \(\text{whSet s have } o1: h \ w < h \ x\) by simp
from \(\text{whEq } \text{eqs-def have } o2: h \ w = h \ e\) by fastforce
from \(o2 \text{ xeqe have } o3: \sim \ h \ w < h \ x\) by simp
from \(o1 \ o3\) show False by contradiction
qed

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

proof (rule disjCond)
fix \(w\)
assume \(\text{whSet}: w : \text{setOf } t2\)
assume \(\text{whEq}: w : \text{eqs } h e\)
from \(\text{whSet s have } o1: h \ x < h \ w\) by simp
from \(\text{whEq } \text{eqs-def have } o2: h \ w = h \ e\) by fastforce
from \(o2 \text{ xeqe have } o3: \sim \ h \ x < h \ w\) by simp
from \(o1 \ o3\) show False by contradiction
qed

ultimately show \(?\text{thesis}\) by auto
qed

from \(\text{this}\) show \(?\text{thesis}\) by simp
qed

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

\textbf{lemma} \ \text{binsert-sorted}: \ \text{sortedTree} \ h \ t \longrightarrow \text{sortedTree} \ h \ (\text{binsert} \ h \ x \ t)

by (induct t) (auto simp add: binsert-set)

We summarize the specification of binsert as follows.

\textbf{corollary} \ \text{binsert-spec}: \ \text{sortedTree} \ h \ t \longrightarrow
\text{sortedTree} \ h \ (\text{binsert} \ h \ x \ t) & \text{setOf} \ (\text{binsert} \ h \ e \ t) = (\text{setOf} \ t) - (\text{eqs } h \ e) \ Un \ \{e\}

by (simp add: binsert-set binsert-sorted)

\section{Removing an element from a tree}

These proofs are influenced by those in \textit{BinaryTree-Tactic}

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

primrec
\[
\text{wrm} :: \ 'a \ \text{hash} \Rightarrow \ 'a \ \text{Tree} \\
\quad \rightarrow \ 'a \ \text{Tree} \\
\quad \text{— tree without the rightmost element}
\]

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
\[
\text{wrmrm} :: \ 'a \ \text{hash} \\
\quad \Rightarrow \ 'a \ \text{Tree} \\
\quad \Rightarrow \ 'a \ \text{Tree} \ \ast \ 'a \\
\quad \text{— computing rightmost and removal in one pass}
\]

where
\[
\text{wrmrm} \ h \ (T \ t1 \ x \ t2) = \\
(\text{if} \ t2 = \text{Tip} \ \text{then} \ (t1, x) \\
\quad \text{else} \ (T \ t1 \ x \ (\text{fst} \ (\text{wrmrm} \ h \ t2)), \\
\text{snd} \ (\text{wrmrm} \ h \ t2)))
\]

primrec
\[
\text{remove} :: \ 'a \ \text{hash} \\
\quad \Rightarrow \ 'a \\
\quad \Rightarrow \ 'a \ \text{Tree} \\
\quad \Rightarrow \ 'a \ \text{Tree} \\
\quad \text{— removal of an element from the tree}
\]

where
\[
\text{remove} \ h \ e \ \text{Tip} = \ \text{Tip} \\
\quad \text{|} \ \text{remove} \ h \ e \ (T \ t1 \ x \ t2) = \\
\quad \quad \text{(if} \ h \ e < h \ x \ \text{then} \ (T \ (\text{remove} \ h \ e \ t1) \ x \ t2) \\
\quad \quad \text{else if} \ h \ x < h \ e \ \text{then} \ (T \ t1 \ x \ (\text{remove} \ h \ e \ t2)) \\
\quad \quad \text{else} \ (\text{if} \ t1 = \text{Tip} \ \text{then} \ t2 \\
\quad \quad \quad \text{else} \ \text{let} \ (t1p, r) = \text{wrmrm} \ h \ t1 \\
\quad \quad \quad \text{in} \ (T \ t1p \ r \ t2)))
\]

theorem \text{wrmrm-decomp}: \ t \sim \Rightarrow \ \text{Tip} \longrightarrow \ \text{wrmrm} \ h \ t = (\text{wrm} \ h \ t, \ \text{rm} \ h \ t)

apply \ (\text{induct-tac} \ t)

apply \ \text{simp-all}

done

lemma \text{rm-set}: \ t \sim \Rightarrow \ \text{Tip} \ & \ \text{sortedTree} \ h \ t \longrightarrow \ \text{rm} \ h \ t : \ \text{setOf} \ t

apply \ (\text{induct-tac} \ t)

apply \ \text{simp-all}

done

lemma \text{wrm-set}: \ t \sim \Rightarrow \ \text{Tip} \ & \ \text{sortedTree} \ h \ t \longrightarrow \ \text{setOf} \ (\text{wrm} \ h \ t) = \ \text{setOf} \ t \ - \ \{\ \text{rm} \ h \ t\} \ \text{(is \ ?P} \ t)

proof \ (\text{induct} \ t)

show \ ?P \ \text{Tip} \ by \ \text{simp}

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

fix \ t2 :: \ 'a \ \text{Tree} \ \text{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 \( \text{setOf} (\text{wrm} h (T t1 x t2)) = \text{setOf} (T t1 x t2) - \{\text{rm} h (T t1 x t2)\} \)
proof (cases \( t2 = \text{Tip} \))
case True
note \( t2\texttip} = \textthis \)
from \( t2\texttip} \)
have \( \text{rm-res} \): \( \text{rm} h (T t1 x t2) = x \) by simp
from \( t2\texttip} \)
have \( \text{wrm-res} \): \( \text{wrm} h (T t1 x t2) = t1 \) by simp
from s have \( x \sim \): \( \text{setOf} t1 \) by auto
from \( \textthis \) \( \text{rm-res} \) \( \text{wrm-res} \) \( t2\texttip} \)
show \( \text{thesis} \) by simp
next case False
note \( t2\textn\texttip} = \textthis \)
from \( t2\textn\texttip} \)
have \( \text{rm-res} \): \( \text{rm} h (T t1 x t2) = \text{rm} h t2 \) by simp
from \( t2\textn\texttip} \)
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 \( \text{h2} \) \( t2\textn\texttip} \) \( s2 \)
have \( \text{o1} \): \( \text{setOf} (\text{wrm} h t2) = \text{setOf} t2 - \{\text{rm} h t2\} \) by simp
show \( \text{thesis} \) proof (simp add: \( \text{rm-res} \) \( \text{wrm-res} \) \( t2\textn\texttip} \) \( \text{h2} \) \( \text{o1} \))
show \( \text{insert} x (\text{setOf} t1 \cup (\text{setOf} t2 - \{\text{rm} h t2\})) = \text{insert} x (\text{setOf} t1 \cup \text{setOf} t2) - \{\text{rm} h t2\} \)
proof -
from s rm-set \( t2\textn\texttip} \) have \( x\textok} \): \( h x < h (\text{rm} h t2) \) by auto
have \( t1\textok} \): \( \forall l \in \text{setOf} t1. h l < h (\text{rm} h t2) \)
proof safe
  fix l :: 'a assume ldef: \( l : \text{setOf} t1 \)
  from ldef s have \( lx \): \( h l < h x \) by auto
  from \( lx \) \( x\textok} \)
  show \( h l < h (\text{rm} h t2) \) byauto
qed
from \( x\textok} \) \( t1\textok} \)
show \( \text{thesis} \) by auto
qed
qed
qed
qed
qed

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

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

11
show \textit{sortedTree} \( h \) (\textit{wrm} \( h \) (\( T \ t1 \ x \ t2 \)))

\textbf{proof (cases} \( t2 = \textit{Tip} \))

\textbf{case} True \textbf{ note} \( t2\text{tip} = \) this

\textbf{ from} \( t2\text{tip} \) \textbf{ have} res: \( \textit{wrm} \ h \ (T \ t1 \ x \ t2) = t1 \) by \textit{simp}

\textbf{ from} \( \textit{res} \) \textbf{ show} \( \textit{thesis} \) by \textit{simp}

\textbf{next case} False \textbf{ note} \( t2\text{nTip} = \) this

\textbf{ from} \( t2\text{nTip} \) \textbf{ have} res: \( \textit{wrm} \ h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\textit{wrm} \ h \ t2) \) by \textit{simp}

\textbf{ from} \( \textit{s} \) \textbf{ have} \( s1: \textit{sortedTree} \ h \ t1 \) by \textit{simp}

\textbf{ from} \( \textit{s} \) \textbf{ have} \( s2: \textit{sortedTree} \ h \ t2 \) by \textit{simp}

\textbf{ from} \( \textit{s} \) \textbf{2} \( h\text{2} \textit{t2\text{nTip}} \) \textbf{ have} \( o1: \textit{sortedTree} \ h \ (\textit{wrm} \ h \ t2) \) by \textit{simp}

\textbf{ from} \( \textit{s} \) \textbf{2} \( t2\text{nTip} \) \textbf{ wrm-set1 have} \( o2: \textit{setOf} \ (\textit{wrm} \ h \ t2) \subseteq \textit{setOf} \ t2 \) by \textit{auto}

\textbf{ from} \( \textit{s} \) \textbf{2} \( h\text{2} \) \( t2\text{nTip} \) \textbf{ have} \( o3: \forall r \in \textit{setOf} \ (\textit{wrm} \ h \ t2). \ h \ x < h \ r \) by \textit{auto}

\textbf{ from} \( \textit{s} \) \textbf{ have} \( s1 \) \( o1 \) \( \textit{res} \) \textbf{ s} \textbf{ show} \( \textit{sortedTree} \ h \ (\textit{wrm} \ h \ (T \ t1 \ x \ t2)) \) by \textit{simp}

\textbf{qed}

\textbf{qed}

\textbf{lemma} \( \textit{wrm-less-\textit{rm}}: \)

\( t \sim Tip \& \textit{sortedTree} \ h \ t \Longrightarrow \)

\( (\forall l \in \textit{setOf} \ (\textit{wrm} \ h \ t)). \ h \ l < h \ (\textit{rm} \ h \ t) \) \( (\textit{is} \ \textit{?P} \ t) \)

\textbf{proof (induct} \( t \))

\textbf{show} \( \textit{?P} \) \( \textit{Tip} \) by \textit{simp}

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

\textbf{fix} \( t2 :: 'a \textit{Tree} \) \textbf{ assume} \( h2: \textit{?P} \ t2 \)

\textbf{fix} \( x :: 'a \)

\textbf{show} \( \textit{?P} \) \( (T \ t1 \ x \ t2) \)

\textbf{proof safe}

\textbf{ fix} \( l :: 'a \) \textbf{ assume} \( l\text{def:} \ l : \textit{setOf} \ (\textit{wrm} \ h \ (T \ t1 \ x \ t2)) \)

\textbf{ assume} \( s: \textit{sortedTree} \ h \ (T \ t1 \ x \ t2) \)

\textbf{ from} \( \textit{s} \) \textbf{ have} \( s1: \textit{sortedTree} \ h \ t1 \) by \textit{simp}

\textbf{ from} \( \textit{s} \) \textbf{ have} \( s2: \textit{sortedTree} \ h \ t2 \) by \textit{simp}

\textbf{ show} \( h \ l < h \ (\textit{rm} \ h \ (T \ t1 \ x \ t2)) \)

\textbf{ proof (cases} \( t2 = \textit{Tip} \))

\textbf{ case} True \textbf{ note} \( l\text{x} = \) this

\textbf{ from} \( t2\text{tip} \) \textbf{ have} \( rm\text{-res}: \textit{rm} \ h \ (T \ t1 \ x \ t2) = x \) by \textit{simp}

\textbf{ from} \( t2\text{tip} \) \textbf{ have} \( \textit{wrm-res}: \textit{wrm} \ h \ (T \ t1 \ x \ t2) = t1 \) by \textit{simp}

\textbf{ from} \( l\text{def} \) \textbf{ wrm-res have} \( o1: l : \textit{setOf} \ t1 \) by \textit{simp}

\textbf{ from} \( \textit{rm-res} \ o1 \) \( s \) \textbf{ show} \( \textit{thesis} \) by \textit{simp}

\textbf{next case} False \textbf{ note} \( t2\text{nTip} = \) this

\textbf{ from} \( t2\text{nTip} \) \textbf{ have} \( rm\text{-res}: \textit{rm} \ h \ (T \ t1 \ x \ t2) = rm \ h \ t2 \) by \textit{simp}

\textbf{ from} \( t2\text{nTip} \) \textbf{ have} \( \textit{wrm-res}: \textit{wrm} \ h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\textit{wrm} \ h \ t2) \) by \textit{simp}

\textbf{ from} \( l\text{def} \) \textbf{ wrm-res}

\textbf{ have} \( l\text{-scope}: l : \{x\} \ Un \textit{setOf} \ t1 \ Un \textit{setOf} \ (\textit{wrm} \ h \ t2) \) by \textit{simp}

\textbf{ have} \( h\text{Less}: h \ l < h \ (\textit{rm} \ h \ t2) \)

\textbf{ proof (cases} \( l = x \))

\textbf{ case} True \textbf{ note} \( lx = \) this

\textbf{ from} \( \textit{s} \) \( t2\text{nTip} \) \textbf{ rm-set} \( s2 \) \textbf{ have} \( o1: h \ x < h \ (\textit{rm} \ h \ t2) \) by \textit{auto}

\textbf{ from} \( lx \) \( o1 \) \textbf{ show} \( \textit{thesis} \) by \textit{simp}

\textbf{next case} False \textbf{ note} \( lx = \) this
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 hlx 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
          </proof>
        </proof>
      </proof>
    </proof>
  </proof>
</proof>
assume y-in-eq: y : eqs h e
from y-in-t2 s have zly: h x < h y by auto
from y-in-eq have eey: h y = h e by (simp add: eqs-def)
from zly 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 (simp add: o1 xle nelx)
show 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 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 zly: 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 eey have nxly: \( \sim (h z < h y) \) by simp
from nxly xly show False by contradiction
qed
show ?thesis
proof (cases \( t1 = \text{Tip} \))
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 nelx nxle ex t1nTip have res: remove \( h e \) (\( \text{T \ t1 \ t2} \)) = 
\( T \text{ (wrm} \ h \text{ t1) (rm} \ h \text{ t1) t2} \) 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 \( h \text{ t1) (setOf (wrm \ h \text{ t1) Un setOf t2) =} \)
  insert \( x \) (setOf \( t1 \text{ Un setOf t2) - eqs \( h e \)
  proof (simp add: t1nTip s1 rm-set wrm-set)
    show insert (rm \( h \text{ t1) (setOf t1 - \{rm \( h \text{ t1\} Un setOf t2) =} \)
    insert \( x \) (setOf \( t1 \text{ Un setOf t2) - eqs h e \)
    proof -
      from t1nTip s1 rm-set
      have o1: insert \( (rm \( h \text{ t1) (setOf t1 - \{rm h \text{ t1\} Un setOf t2) =} \)
                      setOf t1 \text{ Un setOf t2) by auto}
      have o2: insert \( x \text{ (setOf t1 Un setOf t2) - eqs h e =} \)
                      setOf t1 \text{ 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 \( y :: \text{a} \)
          assume y-in-t1: \( y : \text{setOf t1} \)
          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: \( \sim (h y < h e) \) 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
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 (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: s s1 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
      from ...
let \( \forall C_2 \in \text{setOf } (\text{remove } h e t) \), \( h < r \)

have \( o_2 : ?C_2 \)

proof -

from \( s_2 \) have \( \text{setOf } (\text{remove } h e t) = \text{setOf } t - \text{eqs } h e \) by (simp add: remove-set)

from \( s \) this show \(?thesis by auto \)

qed

from \( o_2 s \) show \(?C_1 \& ?C_2 \) by auto

qed

next case False note \( nxle = \text{this} \)

from \( \text{nel } \text{nxle} \) have \( \text{ex } h e = h x \) by simp

show \(?thesis \)

proof (cases \( t_1 = T \text{ip} \))

case True note \( t_1 \text{tip} = \text{this} \)

from \( \text{ex } t_1 \text{tip} \) have \( \text{res } \text{remove } h e (T \text{t}_1 \text{t}_2) = t_2 \) by simp

show \(?thesis by (simp add: res \text{t}_1 \text{tip} \text{ex } s_2) \)

next case False note \( t_1 \text{nTip} = \text{this} \)

from \( \text{nel } \text{nxle } \text{ex } t_1 \text{nTip} \) have \( \text{res } \text{remove } h e (T \text{t}_1 \text{t}_2) = T (\text{wrm } h \text{t}_1) (\text{rm } h \text{t}_1) t_2 \)

by (simp add: Let-def wrm-rm-decomp)

from \( \text{res } \) show \(?thesis \)

proof simp

let \( ?C_1 = \text{sortedTree } h \text{(wrm } h \text{t}_1) \)

let \( ?C_2 = \forall l \in \text{setOf } (\text{wrm } h \text{t}_1) \), \( h l < h (\text{rm } h \text{t}_1) \)

let \( ?C_3 = \forall r \in \text{setOf } t_2 \), \( h (\text{rm } h \text{t}_1) < h r \)

let \( ?C_4 = \text{sortedTree } h t_2 \)

from \( s_1 t_1 \text{Tip} \) have \( o_1 : ?C_1 \) by (simp add: wrm-sort)

from \( s_1 t_1 \text{Tip} \) have \( o_2 : ?C_2 \) by (simp add: wrm-less-rm)

have \( o_3 : ?C_3 \)

proof

fix \( r :: \text{'}a \)

assume \( rt_2 : r : \text{setOf } t_2 \)

from \( \text{sr } \text{set } s_1 t_1 \text{Tip} \) have \( o_1 : h (\text{rm } h \text{t}_1) < h x \) by auto

from \( rt_2 s \) have \( o_2 : h x < h r \) by auto

from \( o_1 o_2 \) show \( h (\text{rm } h \text{t}_1) < h r \) by simp

qed

from \( o_1 o_2 o_3 s_2 \) show \(?C_1 \& ?C_2 \& ?C_3 \& ?C_4 \) by simp

qed

qed

qed

We summarize the specification of remove as follows.

**corollary remove-spec** sortedTree \( h t \longrightarrow \)

\( \text{sortedTree } h (\text{remove } h e t) \& \text{setOf } (\text{remove } h e t) = \text{setOf } t - \text{eqs } h e \)
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 \( \text{mapOf-lookup3: tlookup \text{fst} \text{i} \text{t} = \text{None}} \)

proof (cases tlookup \text{fst} \text{i} \text{t})

case None from this show \( \text{?thesis by assumption} \)

next case (Some \text{ia}) note \( \text{tsome = this} \)
    from this have \( \text{o1: tlookup \text{fst} \text{i} \text{t} = \text{Some} (\text{fst} \text{ia}, \text{snd} \text{ia}) by simp} \)
    have \( \text{mapOf \text{t} \text{i} = \text{Some} (\text{snd} \text{ia})} \)
    by (insert \text{mapOf-lookup2 [of \text{i} \text{t} \text{fst} \text{ia} \text{snd} \text{ia}], simp add: o1})
    from this have \( \text{mapOf \text{t} \text{i} \sim = \text{None by simp}} \)
    from this \( \text{h show ?thesis by simp — contradiction} \)

qed

lemma assumes \( \text{v: valid-tmap \text{t}} \)
    assumes \( \text{h: mapOf \text{t} \text{i} = \text{Some} \text{a}} \)
    shows \( \text{mapOf-lookup4: tlookup \text{fst} \text{i} \text{t} = \text{Some} (\text{i}, \text{a})} \)

proof (cases tlookup \text{fst} \text{i} \text{t})

case None
    from this mapOf-lookup1 have \( \text{mapOf \text{t} \text{i} = \text{None by auto}} \)
    from this \( \text{h show ?thesis by simp — contradiction} \)

next case (Some \text{ia}) note \( \text{tsome = this} \)
    have \( \text{tlookup-some-inst: sortedTree \text{fst} \text{t} \& (tlookup \text{fst} \text{i} \text{t} = \text{Some} \text{ia}) \rightarrow \text{ia : setOf \text{t} \& \text{fst} \text{ia} = \text{i by simp add: tlookup-some)}} \)
    from \( \text{tlookup-some-inst tsome v have ia : setOf \text{t by simp}} \)
    from tsome have \( \text{mapOf \text{t} \text{i} = \text{Some} (\text{snd} \text{ia}) by (simp add: mapOf-def)} \)
    from this \( \text{h have \text{o1: snd} \text{ia} = \text{a by simp}} \)
    from o1 o2 have \( \text{ia = (i, a by auto}} \)
    from this tsome show \( \text{tlookup \text{fst} \text{i} \text{t} = \text{Some} (i, a) by simp} \)

qed

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

lemma assumes \( \text{v: valid-tmap \text{t}} \)
    shows \( \text{mapset-none: (mapOf \text{t} \text{i} = \text{None}) = (\forall \text{a}. (i, a) \notin \text{setOf \text{t}})} \)

proof 
  \( \sim = \sim \).

assume \( \text{mapNone: mapOf \text{t} \text{i} = \text{None}} \)

from \( \text{v mapNone mapOf-lookup3 have lnone: tlookup \text{fst} \text{i} \text{t} = \text{None by auto}} \)
show \( \forall \text{a}. (i, a) \notin \text{setOf \text{t}} \)

proof 
  fix \text{a}
  show \( (i, a) \sim: \text{setOf \text{t}} \)

proof
  assume \( \text{iain: (i, a) : setOf \text{t}} \)
  have \( \text{tlookup-none-inst: sortedTree \text{fst} \text{t} \& (tlookup \text{fst} \text{i} \text{t} = \text{None}) \rightarrow (\forall x \in \text{setOf \text{t}. \text{fst} x \sim = i})} \)
  by (insert \text{tlookup-none [of \text{fst} \text{t} \text{i}], assumption})
  from \( \text{vlnone tlookup-none-inst have \( \forall x \in \text{setOf \text{t}. \text{fst} x \sim = i by simp}} \)
  from this \( \text{iain have \text{fst} (i, a) \sim = i by fastforce} \)
from this show False by simp
qed
qed

— i::=
next assume h: \forall a, (i,a) \notin setOf t
show mapOf t i = None
proof (cases mapOf t i)
case None then show \?thesis .
next case (Some a) note mapsome = this
  from v mapsome have o1: tlookup fst i t = Some (i,a) by (simp add: mapOf-lookup4)
from tlookup-some have tlookup-some-inst: sortedTree fst t & tlookup fst i t = Some (i,a) --->
(i,a) : setOf t & fst (i,a) = i
  by (insert tlookup-some [of fst t i (i,a)], assumption)
from v o1 this have (i,a) : setOf t by simp
  from this h show \?thesis by auto — contradiction
qed
qed

lemma assumes v: valid-tmap t
  shows mapset-some: (mapOf t i = Some a) = ((i,a) : setOf t)
proof
— ==¿
assume mapsome: mapOf t i = Some a
from v mapsome have o1: tlookup fst i t = Some (i,a) by (simp add: mapOf-lookup4)
from tlookup-some have tlookup-some-inst:
  sortedTree fst t & tlookup fst i t = Some (i,a) --->
(i,a) : setOf t & fst (i,a) = i
  by (insert tlookup-some [of fst t i (i,a)], assumption)
from v o1 this have (i,a) : setOf t by simp
— i::=
next assume iain: (i,a) : setOf t
from v iain tlookup-finds have tlookup fst (fst (i,a)) t = Some (i,a) by fastforce
from this have tlookup fst i t = Some (i,a) by simp
from this show mapOf t i = Some a by (simp add: mapOf-def)
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 mupdate :: '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 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
            have will-find: tlookup fst i ?tr = Some (i,a)
            proof
               from setSpec have kvp: (i,a) : setOf ?tr by simp
               have binsert-sorted-inst: sortedTree fst t --> sortedTree fst ?tr
                  by (insert binsert-sorted [of fst t (i,a)], assumption)
               from v binsert-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 show ?thesis by (simp add: mapOf-def)
               qed
            from rhs-res lhs-res i2ei show ?thesis by simp
         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
            proof
               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
      qed
have mapset-none-inst:
valid-tmap ?tr −−> (mapOf ?tr i2 = None) = (∀ a. (i2, a) /∈ 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 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

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 − egs 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: egs-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 if $x < e$ then memb e t2
else True)

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

## 15 Insertion operation

**primrec** binsert :: nat => tree => tree
— Insert a node into sorted tree.

**where**

\[
\begin{align*}
\text{binsert } x \text{ Tip } & = (\text{Nd Tip } x \text{ Tip}) \\
\text{binsert } x \ (\text{Nd } t1 y t2) & = (\text{if } x < y \text{ then} \\
& \quad (\text{Nd } (\text{binsert } x t1) y t2) \\
& \quad \text{else} \\
& \quad (\text{if } y < x \text{ then} \\
& \quad \quad (\text{Nd } t1 y (\text{binsert } x t2)) \\
& \quad \text{else} \ (\text{Nd } t1 y t2)))
\end{align*}
\]

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

**theorem** binsert-sorted: $\text{sorted } t \implies \text{sorted } (\text{binsert } x t)$
by (induct t) (auto simp add: set-of-binsert)

**corollary** binsert-spec:
\[
\text{sorted } t \implies \text{sorted } (\text{binsert } x t) \&
\quad \text{set-of } (\text{binsert } x t) = \text{set-of } t \cup \{x\}
\]
by (simp add: binsert-sorted)

## 16 Remove operation

**primrec**
\[
\begin{align*}
\text{rm } :: \text{ tree } => \text{ nat} & \ — \text{find the rightmost element in the tree} \\
\text{where} & \\
\text{rm}(\text{Nd } l x r) & = (\text{if } r = \text{Tip then } x \text{ else rm r})
\end{align*}
\]

**primrec**
\[
\begin{align*}
\text{rem } :: \text{ tree } => \text{ tree} & \ — \text{find the tree without the rightmost element} \\
\text{where} & \\
\text{rem}(\text{Nd } l x r) & = (\text{if } r=\text{Tip then } l \text{ else } \text{Nd } l x (\text{rem } r))
\end{align*}
\]

**primrec**
\[
\begin{align*}
\text{remove } :: \text{ nat } => \text{ tree } => \text{ tree} & \ — \text{remove a node from sorted tree} \\
\text{where} & \\
\text{remove } x \text{ Tip } & = \text{Tip} \\
\text{rem } x (\text{Nd } l y r) & = \\
& (\text{if } x < y \text{ then } \text{Nd } (\text{remove } x l) y r \text{ else}
\end{align*}
\]
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** \( \text{rm-in-set-of} \): \( t \sim \text{Tip} \implies \text{rm} \ : \text{set-of} \ t \)
by (induct \( t \)) auto

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

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

**lemma** \( \text{sorted-rem} \): \( \| t \sim \text{Tip} \| \implies x \in \text{set-of} (\text{rem} \ t) \implies x < \text{rm} \ t \)
by (induct \( t \)) (auto simp add: \( \text{set-of-rem} \) split:if-splits)

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

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

**corollary** \( \text{remove-spec} \) — summary specification of remove
\( \text{sorted} \ t = \text{set-of} (\text{remove} \ x \ t) = \text{set-of} \ t - \{x\} \)
by (simp add: \( \text{remove-sorted} \))

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

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

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

We can test this implementation by generating code.

**definition** \( \text{test} = \text{memb} \ 4 \ (\text{remove} \ (3::\text{nat}) \ (\text{binsert} \ 4 \ (\text{binsert} \ 3 \ \text{Tip}))) \)
export-code test
  in SML module-name BinaryTree-TacticStyle-Code file (BinaryTree-TacticStyle-Code.ML)
end