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 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

2
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 = k)
by (induct t, auto)

lemma tlookup-some:
  sortedTree h t & (tlookup h k t = Some x) ---> x \in 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 \in setOf t & b \in 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 \( h a b b : h \ a = h \ b \)
from \( s \) have \( s1 : \text{sortedTree} \ h \ t1 \) by auto
from \( s \) have \( s2 : \text{sortedTree} \ h \ t2 \) by auto

show \( a = b \)
— We consider 9 cases for the position of \( a \) and \( b \) are in the tree
proof —
— three cases for \( a \)
from \( a \text{def} \) have \( a : \text{setOf} \ t1 \ | \ a = x \ | \ a : \text{setOf} \ t2 \) by auto
moreover \{ assume \( a \text{def}1 : a : \text{setOf} \ t1 \n  \n  have \ ?\text{thesis} \n  \n  proof —
  — three cases for \( b \)
from \( b \text{def} \) have \( b : \text{setOf} \ t1 \ | \ b = x \ | \ b : \text{setOf} \ t2 \) by auto
moreover \{ assume \( b \text{def}1 : b : \text{setOf} \ t1 \n  \n  from \( s1 \) \( a \text{def}1 \) \( b \text{def} \ hahb h1 \) have \ ?\text{thesis} by simp \}
moreover \{ assume \( b \text{def}1 : b = x \n  \n  from \( a \text{def}1 \) \( b \text{def}1 \) \( s \) have \( h a < h b \) by auto
  from \( \text{this} \) \( hahb \) have \ ?\text{thesis} by simp \}
moreover \{ assume \( b \text{def}1 : b : \text{setOf} \ t2 \n  \n  from \( a \text{def}1 \) \( s \) have \( o1 : h a < h x \) by auto
  from \( b \text{def}1 \) \( s \) have \( o2 : h x < h b \) by auto
  from \( o1 \) \( o2 \) have \( h a < h b \) by simp
  from \( \text{this} \) \( hahb \) have \ ?\text{thesis} by simp \} — case impossible
ultimately show \ ?\text{thesis} by blast
qed

moreover \{ assume \( a \text{def}1 : a = x \n  \n  have \ ?\text{thesis} \n  \n  proof —
  — three cases for \( b \)
from \( b \text{def} \) have \( b : \text{setOf} \ t1 \ | \ b = x \ | \ b : \text{setOf} \ t2 \) by auto
moreover \{ assume \( b \text{def}1 : b : \text{setOf} \ t1 \n  \n  from \( \text{this} \) \( hahb \) have \ ?\text{thesis} by simp \}
moreover \{ assume \( b \text{def}1 : b = x \n  \n  from \( a \text{def}1 \) \( b \text{def}1 \) have \ ?\text{thesis} by simp \}
moreover \{ assume \( b \text{def}1 : b : \text{setOf} \ t2 \n  \n  from \( \text{this} \) \( hahb \) have \ ?\text{thesis} by simp \} — case impossible
ultimately show \ ?\text{thesis} by blast
qed

moreover \{ assume \( a \text{def}1 : a : \text{setOf} \ t2 \n  \n  have \ ?\text{thesis} \n  \n  proof —
  — three cases for \( b \)
from \( b \text{def} \) have \( b : \text{setOf} \ t1 \ | \ b = x \ | \ b : \text{setOf} \ t2 \) by auto

moreover \{ \text{assume } bdef1: \text{ } b : \text{setOf } t1 \}
from bdef1 \text{ s have } a1: \text{ } h \ b < h \ x \text{ by } \text{auto} 
from adef1 \text{ s have } a2: \text{ } h \ x < h \ a \text{ by } \text{auto} 
from a1 a2 \text{ have } h \ b < h \ a \text{ by } \text{simp} 
from this \text{ habb have } \text{thesis by } \text{simp} \} — \text{case impossible} 
moreover \{ \text{assume } bdef1: \text{ } b = x \}
from adef1 bdef1 \text{ s have } h \ b < h \ a \text{ by } \text{auto} 
from this hahb \text{ have } \text{thesis by } \text{simp} \} — \text{case impossible} 
morerover \{ \text{assume } bdef1: \text{ } b : \text{setOf } t2 \}
from s2 adef1 bdef1 hahb h2 \text{ have } \text{thesis by } \text{simp} \} 
ultimately show \text{thesis by } \text{blast} 
qed 

ultimately show \text{thesis by } \text{blast} 
qed 

ultimately show \text{thesis by } \text{blast} 
qed 

\text{lemma } \text{tlookup-finds: } — \text{if a node is in the tree, lookup finds it} 
\text{sortedTree } h \ t \ & \ y : \text{setOf } t \implies \text{ } 
\text{tlookup } h \ (h \ y) \ t = \text{Some } y 
\text{proof safe} 
\text{assume } s: \text{sortedTree } h \ t 
\text{assume } yint: \text{ } y : \text{setOf } t 
\text{show } \text{tlookup } h \ (h \ y) \ t = \text{Some } y 
\text{proof } (\text{cases } \text{tlookup } h \ (h \ y) \ t) 
\text{case } \text{None } \text{note } \text{res } = \text{this} 
\text{from s res have } \text{sortedTree } h \ t \ & \ (\text{tlookup } h \ (h \ y) \ t = \text{None}) \text{ by } \text{simp} 
\text{from this have } a1: \forall x \in \text{setOf } t. \text{ } h \ x \sim h \ y \text{ by } (\text{simp add: tlookup-none}) 
\text{from a1 yint have } h \ y \sim h \ y \text{ by } \text{fastforce} 
\text{from this show } \text{thesis by } \text{simp} 
\text{next case } (\text{Some } z) \text{ note } \text{res } = \text{this} 
\text{have ls: } \text{sortedTree } h \ t \ & \ (\text{tlookup } h \ (h \ y) \ t = \text{Some } z) \implies \text{ } 
\text{z:setOf } t \ & \ h \ z = h \ y \text{ by } (\text{simp add: tlookup-some}) 
\text{have sd: } \text{sorted-distinct-pred } h \ y \ z \ t 
\text{by } (\text{insert sorted-distinct } [of \ h \ y \ z \ t], \text{ simp}) 
\text{from s res ls have } a1: \text{ } z: \text{setOf } t \ & \ h \ z = h \ y \text{ by } \text{simp} 
\text{from s yint a1 sd have } y = z \text{ by } \text{auto} 
\text{from this res show } \text{tlookup } h \ (h \ y) \ t = \text{Some } y \text{ by } \text{simp} 
\text{qed} 
\text{qed} 

3.1 Tree membership as a special case of lookup 

\text{definition } \text{memb :: } \text{'}a \text{ hash } => \text{'}a => \text{'}a \text{ Tree } => \text{bool where} 
\text{memb } h \ t \ == 
(\text{case } (\text{tlookup } h \ (h \ x) \ t) \ of 
\text{None } => \text{False} 

5
lemma assumes s: sortedTree h t
  shows memb-spec: memb h x t = (x : setOf t)
proof (cases tlookup h (h x) t)
case None note tNone = this
  from tNone have res: memb h x t = False by (simp add: memb-def)
  from s tNone lookup-none have o1: \forall y \in setOf t. h y \sim h x by fastforce
  have notIn: x \sim: setOf t
    proof
      assume h: x \in setOf t
      from h o1 have h x \sim h x by fastforce
      from this show False by simp
    qed
  from res notIn show ?thesis by simp
next case (Some z) note tSome = this
  from s tSome lookup-some have zin: z \in setOf t by fastforce
  show ?thesis
    proof (cases x=z)
      case True note xez = this
      from tSome xez have res: memb h x t by (simp add: memb-def)
      from res zin xez show ?thesis by simp
    next case False note xnez = this
      from tSome xnez have res: \sim memb h x t by (simp add: memb-def)
      have x \sim: setOf t
        proof
          assume xin: x \in setOf t
          from s tSome lookup-some have hzhx: h x = h z by fastforce
          have o1: 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 ?thesis by simp
      qed
      qed
  qed

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

4 Insertion into a Tree

primrec
  binsert :: 'a hash => 'a => 'a Tree => 'a Tree
where
  binsert h e Tip = (T Tip e Tip)
| binsert h e (T t1 x t2) = (if h e < h x then
    (T (binsert h e t1) x t2)
  else
    (if h x < h e then...
A technique for proving disjointness of sets.

**lemma** disjCond: \[ \text{!! } x. \text{!! } x:A; x:B \implies \text{False} \implies A \cap B = \{\}\]
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: \[\text{sortedTree } h t \implies \text{setOf } (\text{binsert } h e t) = (\text{setOf } t) - (\text{eqs } h e) \cup \{e\}\]

**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: \(\text{setOf } (\text{binsert } h e t1) = \text{setOf } t1 - (\text{eqs } h e) \cup \{e\}\)
by simp
**from** s have s2: sortedTree h t2 by (rule sortLemmaR)
**from** s2 and h2 have c2: \(\text{setOf } (\text{binsert } h e t2) = \text{setOf } t2 - (\text{eqs } h e) \cup \{e\}\)
by simp
**show** setOf (binsert h e (T t1 x t2)) = setOf (T t1 x t2) - (eqs h e) \cup \{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) \cup \{e\}
**proof** (simp add: res eLess c1)

**show** insert x (insert e (setOf t1 - (eqs h e) \cup \{e\})) = insert e (insert x (setOf t1 \cup \{e\} - (eqs h e))
**proof**

**have** eqsLessX: \(\forall e.l \in \text{eqs } h e. \ h \ e l < h x\) by (simp add: eqs-def eLess)
**from** this have eqsDisjX: \(\forall e.l \in \text{eqs } h e. \ h \ e l \sim h x\) by fastforce
**from** s have xLessT2: \(\forall r \in \text{setOf } t2. \ h \ x < h r\) by auto
**have** eqsLessT2: \(\forall e.l \in \text{eqs } h e. \ \forall r \in \text{setOf } t2. \ h \ e l < h r\)
**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

7
from xLessT2 hr o1 eLess show h el < h r by auto
qed
from eqsLessT2 have eqsDisjT2: \( \forall \, e \in \text{eqs} \ h \ e \ \forall \, r \in \text{setOf} \ t2 \, \ h \ el \sim h \ r \)
by fastforce
from eqsDisjX eqsDisjT2 show ?thesis by fastforce
qed
qed
next case False note eNotLess = this
show setOf \((\text{binset} \ h \ e \ (T \ t1 \ x \ t2)) = \text{setOf} \ (T \ t1 \ x \ t2) - \text{eqs} \ h \ e \ \cup \ {e}\) proof (cases h x < h e)
case True note xLess = this
from xLess have res: \(\text{binset} \ h \ e \ (T \ t1 \ x \ t2) = (T \ t1 \ x \ (\text{binset} \ h \ e \ t2))\) by simp
show setOf \((\text{binset} \ h \ e \ (T \ t1 \ x \ t2)) = \text{setOf} \ (T \ t1 \ x \ t2) - \text{eqs} \ h \ e \ \cup \ {e}\)
proof (simp add: res xLess eNotLess c2)
show insert x (insert e (setOf t1 \ \cup \ \text{setOf} \ t2 - eqs h e)) = insert e (insert x (setOf t1 \ \cup \ \text{setOf} \ t2 - eqs h e))
proof
have XLessEqs: \(\forall \, e \in \text{eqs} \ h \ e \, \ h \ x < h \ el \) by (simp add: eqs-def xLess)
from this have eqsDisjX: \(\forall \, e \in \text{eqs} \ h \ e \, \ h \ el \sim h \ x \) by auto
from s have t1LessX: \(\forall \, l \in \text{setOf} \ t1 \, \ h \ l < h \ x \) by auto
have T1lessEqs: \(\forall \, e \in \text{eqs} \ h \ e \, \forall \, l \in \text{setOf} \ t1 \, \ h \ l < h \ el \)
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: \(\forall \, e \in \text{eqs} \ h \ e \, \forall \, l \in \text{setOf} \ t1 \, \ h \ el \sim h \ l \)
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: \(\text{binset} \ h \ e \ (T \ t1 \ x \ t2) = (T \ t1 \ e \ t2)\) by simp
show setOf \((\text{binset} \ h \ e \ (T \ t1 \ x \ t2)) = \text{setOf} \ (T \ t1 \ x \ t2) - eqs h e \ \cup \ {e}\)
proof (simp add: res eNotLess xege)
show insert e (setOf t1 \ \cup \ \text{setOf} \ t2) = insert e (insert x (setOf t1 \ \cup \ \text{setOf} \ t2 - eqs h e))
proof
  have insert x (setOf t1 \ \cup \ \text{setOf} \ t2) - eqs h e = setOf t1 \ \cup \ \text{setOf} \ t2
  proof
    have x : eqs h e by (simp add: eqs-def xege)
moreover have \((setOf \ t1) \cap (eqs \ h \ e) = \{\}\) 
proof (rule disjCond) 
  fix \(w\) 
  assume \(whSet: \ w \in (setOf \ t1)\) 
  assume \(whEq: \ w \in (eqs \ h \ e)\) 
  from \(whSet s \Rightarrow o1: h \ w < h \ x\) by simp 
  from \(whEq eqs-def \Rightarrow o2: h \ w = h \ e\) by fastforce 
  from \(o2 xeqe \Rightarrow o3: \sim h \ w < h \ x\) by simp 
  from \(o1 o3 \Rightarrow show False\) by contradiction 
qed 
moreover have \((setOf \ t2) \cap (eqs \ h \ e) = \{\}\) 
proof (rule disjCond) 
  fix \(w\) 
  assume \(whSet: \ w \in (setOf \ t2)\) 
  assume \(whEq: \ w \in (eqs \ h \ e)\) 
  from \(whSet s \Rightarrow o1: h \ x < h \ w\) by simp 
  from \(whEq eqs-def \Rightarrow o2: h \ w = h \ e\) by fastforce 
  from \(o2 xeqe \Rightarrow o3: \sim h \ x < h \ w\) by simp 
  from \(o1 o3 \Rightarrow show False\) 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: \(sortedTree \ h \ t \Rightarrow sortedTree \ h \ (binsert \ h \ x \ t)\)
by (induct \(t\)) (auto simp add: binsert-set)

We summarize the specification of binsert as follows.

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

5 Removing an element from a tree

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

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

primrec
\[\text{wrm} :: \text{'a hash} => \text{'a Tree} => \text{'a Tree}\]
— tree without the rightmost element

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

primrec
\[\text{wrmrm} :: \text{'a hash} => \text{'a Tree} => \text{'a Tree} * \text{'a}\]
— computing rightmost and removal in one pass

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

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

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

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

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

lemma \[\text{wrm-set}: t \sim = \text{Tip} & \text{sortedTree } h \ t \longrightarrow \text{setOf } (\text{wrm } h \ t) = \text{setOf } t - \{\text{rm } h \ t\} \ \text{(is } ?P \ t)\]
proof (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 \( \forall \)P (T t1 x t2)

proof (rule impI, erule conjE)
  assume s: sortedTree h (T t1 x t2)
  show setOf (wrm h (T t1 x t2)) =
    setOf (T t1 x t2) - \{ 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 s have x ∼: setOf t1 by auto
      from this rm-res wrm-res t2tip 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 s have s2: sortedTree h t2 by simp
      from h2 t2nTip s2 have o1: setOf (wrm h t2) = setOf t2 - \{ rm h t2 \} by simp
      show ?thesis
        proof (simp add: rm-set t2nTip h2 o1)
          show insert x (setOf t1 Un (setOf t2 - \{ rm h t2 \})) =
            insert x (setOf t1 Un setOf t2) - \{ rm h t2 \}
            proof -
              from s rm-set t2nTip have xOk: h x < h (rm h t2) by auto
              have t1Ok: \forall l ∈ setOf t1. h l < h (rm h t2)
                proof safe
                  fix l :: 'a assume ldef: l : setOf t1
                  from ldef s have lx: h l < h x by auto
                  from lx xOk show h l < h (rm h t2) by auto
                qed
              from xOk t1Ok show ?thesis by auto
            qed
        qed
    qed
  qed

lemma wrm-set1: t ∼ = Tip & sortedTree h t ---> setOf (wrm h t) <= setOf t
  by (auto simp add: wrm-set)

lemma wrm-sort: t ∼ = Tip & sortedTree h t ---> sortedTree h (wrm h t) (is \( \forall \)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
        assume s: sortedTree h (T t1 x t2)
show \texttt{sortedTree\ h (\texttt{wrm\ h (T\ t1\ x\ t2))}}

proof (cases \texttt{t2 = Tip})

case \texttt{True} note \texttt{t2tip = this}
    from \texttt{t2tip} have \texttt{res: wrm\ h (T\ t1\ x\ t2) = t1 by simp}
    from \texttt{res\ s\ show \ ?thesis by simp}

next case \texttt{False} note \texttt{t2nTip = this}
    from \texttt{t2nTip} have \texttt{res: wrm\ h (T\ t1\ x\ t2) = T\ t1\ x\ (wrm\ h\ t2) by simp}
    from \texttt{s\ have s1: sortedTree\ h\ t1 by simp}
    from \texttt{s\ have s2: sortedTree\ h\ t2 by simp}
    from \texttt{s\ o2\ have o3: \forall\ r\ \in\ setOf\ (wrm\ h\ t2).\ h\ x < h\ r\ by\ auto}
    from \texttt{s\ o2\ have o3: setOf\ (wrm\ h\ t2) < setOf\ t2\ by\ auto}
    from \texttt{s\ o2\ have o3: \forall\ r\ \in\ setOf\ (wrm\ h\ t2).\ h\ x < h\ r\ by\ auto}
    from \texttt{s\ o2\ have o3: setOf\ (wrm\ h\ t2) < setOf\ t2\ by\ auto}

qed

qed

lemma \texttt{wrm-less-rm}: 
    \texttt{t \sim Tip & sortedTree\ h\ t \longrightarrow} 
    \((\forall\ l\ \in\ setOf\ (wrm\ h\ t).\ h\ x < h\ (rm\ h\ t))\) (is \texttt{?P\ t})

proof (induct \texttt{t})
    show \texttt{?P\ Tip by simp}
    fix \texttt{t1 :: 'a\ Tree\ assume h1: \?P\ t1}
    fix \texttt{t2 :: 'a\ Tree\ assume h2: \?P\ t2}
    fix \texttt{x :: 'a}
    show \texttt{?P\ (T\ t1\ x\ t2)}
        proof safe
            fix \texttt{l :: 'a\ assume ldef: l : setOf\ (wrm\ h\ (T\ t1\ x\ t2))}
            assume \texttt{s: sortedTree\ h\ (T\ t1\ x\ t2)}
            from \texttt{s\ have s1: sortedTree\ h\ t1 by simp}
            from \texttt{s\ have s2: sortedTree\ h\ t2 by simp}
            show \texttt{h\ l < h\ (rm\ h\ (T\ t1\ x\ t2))}
                proof (cases \texttt{t2 = Tip})
                    case \texttt{True} note \texttt{t2tip = this}
                        from \texttt{t2tip} have \texttt{rm-res: rm\ h\ (T\ t1\ x\ t2) = x by simp}
                        from \texttt{t2tip} have \texttt{wrm-res: wrm\ h\ (T\ t1\ x\ t2) = t1 by simp}
                        from \texttt{ldef\ wrm-res\ have o1: l : setOf\ t1 by simp}
                        from \texttt{rm-res\ o1\ s\ show \ ?thesis by simp}
                    next case \texttt{False} note \texttt{t2nTip = this}
                        from \texttt{t2nTip} have \texttt{rm-res: rm\ h\ (T\ t1\ x\ t2) = rm\ h\ t2 by simp}
                        from \texttt{t2nTip} have \texttt{wrm-res: wrm\ h\ (T\ t1\ x\ t2) = T\ t1\ x\ (wrm\ h\ t2) by simp}
                        from \texttt{ldef\ wrm-res}
                            have \texttt{l-scope: l : \{x\} Un\ setOf\ t1\ Un\ setOf\ (wrm\ h\ t2) by simp}
                            have \texttt{hLess: h\ l < h\ (rm\ h\ t2)}
                            proof (cases \texttt{l = x})
                                case \texttt{True} note \texttt{lx = this}
                                    from \texttt{s\ t2nTip\ rm-set\ s2\ have o1: h\ x < h\ (rm\ h\ t2) by auto}
                                    from \texttt{lx\ o1\ show \ ?thesis by simp}
                                next case \texttt{False} note \texttt{lx = this}
show \( ?\text{thesis} \)

proof (cases \( l : \text{setOf} \ t1 \))

- case True
  note \( l\text{-in-t1} = \text{this} \)
  from \( s \ t2n\text{Tip} \) \( rm\text{-set} \ s2 \) have \( o1 : h \ x < h \ (rm \ h \ t2) \) by auto
  from \( l\text{-in-t1} s \) have \( o2 : h \ l < h \ x \) by auto
  from \( o1 o2 \) show \( ?\text{thesis} \) by simp

- next case False
  note \( l\text{-notin-t1} = \text{this} \)
  from \( l\text{-scope} l x \) \( l\text{-notin-t1} \) have \( l\text{-in-res} : l : \text{setOf} \ (wrm \ h \ t2) \) by auto
  from \( l\text{-in-res} \) \( h2 \ t2n\text{Tip} \) \( s2 \) show \( ?\text{thesis} \) by auto

qed

qed

lemma remove-set: \( \text{sortedTree} \ h \ t \quad \longrightarrow \quad \text{setOf} \ \text{remove} \ h \ e \ t = \text{setOf} \ t - \text{eqs} \ h \ e \) (is \( ?P \ t \))

proof (induct \( t \))

- show \( ?P \ \text{Tip} \) by auto
- 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 \( s : \text{sortedTree} \ h \ (T \ t1 \ x \ t2) \)
    show \( \text{setOf} \ \text{remove} \ h \ e \ (T \ t1 \ x \ t2) = \text{setOf} \ (T \ t1 \ x \ t2) - \text{eqs} \ h \ e \)
    proof (cases \( h \ e < h \ x \))
      case True
      note \( elx = \text{this} \)
      from \( elx \) have \( \text{res: remove} \ h \ e \ (T \ t1 \ x \ t2) = T \ (\text{remove} \ h \ e \ t1) \ x \ t2 \)
      by simp
      from \( s \) have \( s1 : \text{sortedTree} \ h \ t1 \) by simp
      from \( s1 h1 \) have \( o1 : \text{setOf} \ \text{remove} \ h \ e \ t1 = \text{setOf} \ t1 - \text{eqs} \ h \ e \) by simp
      show \( ?\text{thesis} \)
      proof (simp add: \( o1 \) \( elx \))
        show \( \text{insert} \ x \ \text{setOf} \ t1 - \text{eqs} \ h \ e \ \text{Un} \ \text{setOf} \ t2 = \)
        \( \text{insert} \ x \ (\text{setOf} \ t1 \ \text{Un} \ \text{setOf} \ t2) - \text{eqs} \ h \ e \)
        proof
          have \( xOk : x \sim: \text{eqs} \ h \ e \)
          proof
            assume \( h : x : \text{eqs} \ h \ e \)
            from \( h \) have \( o1 : \sim \ (h \ e < h \ x) \) by (simp add: \( \text{eqs-def} \))
            from \( elx o1 \) show \( False \) by contradiction
          qed
          have \( t2Ok : (\text{setOf} \ t2) \ \text{Int} \ (\text{eqs} \ h \ e) = \{\} \)
          proof (rule \( \text{disjCond} \))
            fix \( y :: ’a \)
            assume \( y\text{-in-t2} : y : \text{setOf} \ t2 \)
          qed
        qed
      qed
    qed
  qed

qed
assume \( y \in \text{eqs} \) : \( y = \text{h e} \)
from \( y \in \text{t2} \) s have \( \text{xly: } h x < h y \) by auto
from \( y \in \text{eqs} \) have \( \text{eey: } h y = h e \) by (simp add: eqs-def)
from \( \text{xly eey} \) have \( \text{nelx: } \sim (h e < h x) \) by simp
from \( \text{nelx xle show False by contradiction} \)
qed
from \( \text{xOk t2Ok show ?thesis by auto} \)
qed
next case False
note \( \text{nelx} = \text{this} \)
show ?thesis
proof (cases \( h x < h e \))
  case True
  note \( \text{xle} = \text{this} \)
  from \( \text{xle} \) have \( \text{res: } \text{remove h e } (T \text{ t1 x t2}) = T \text{ t1 x (remove h e t2)} \) by simp
from \( s \) have \( s2: \text{sortedTree h t2 by simp} \)
from \( s2 \) \( h2 \) have \( \text{o1: } \text{setOf } (\text{remove h e t2}) = \text{setOf t2} - \text{eqs h e by simp} \)
show ?thesis
proof (simp add: \( \text{o1 xle nelx} \))
  show \( \text{insert x } (\text{setOf t1 Un setOf t2}) - \text{eqs h e} \)
  proof
  have \( \text{t1Ok: } (\text{setOf t1}) \text{ Int (eqs h e}) = \{ \}
  proof (rule disjCond)
    fix \( y \) :: \('a\)
    assume \( \text{y-in-t1: } y : \text{setOf t1} \)
    assume \( \text{y-in-eq: } y : \text{eqs h e} \)
    from \( \text{y-in-t1 s have } \text{gylx: } h y < h x \) by auto
    from \( \text{y-in-eq have } \text{eey: } h y = h e \) by (simp add: eqs-def)
    from \( \text{gylx eey have } \text{nxle: } \sim (h x < h e) \) by simp
    from \( \text{nxle xle show False by contradiction} \)
    qed
  from \( \text{xOk t1Ok show ?thesis by auto} \)
  qed
  qed
next case False
note \( \text{nxle} = \text{this} \)
from \( \text{nelx nxle} \) have \( \text{ex: } h e = h x \) by simp
have \( \text{t2Ok: } (\text{setOf t2}) \text{ Int (eqs h e}) = \{ \}
proof (rule disjCond)
  fix \( y \) :: \('a\)
  assume \( \text{y-in-t2: } y : \text{setOf t2} \)
  assume \( \text{y-in-eq: } y : \text{eqs h e} \)
  from \( \text{y-in-t2 s have } \text{gylx: } h x < h y \) 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: ~ (h x < h y) by simp
from nxly xly show \textit{False} by contradiction
qed

show ?thesis
proof (cases t1 = \textit{Tip})
case True
note \textit{t1tip = this}
from ex \textit{t1tip} have \textit{res: remove h e (T t1 x t2) = t2 by simp}
show ?thesis
proof (simp add: \textit{t1tip ex})
  show \textit{setOf t2 = insert x (setOf t2) = eqs h e}
    proof
      from \textit{ex} have \textit{x-in-eqs: x : eqs h e by (simp add: eqs-def)}
      from \textit{x-in-eqs t2Ok} show ?thesis by auto
    qed
qed

case False
note \textit{t1nTip = this}
from \textit{nelx nxle ex \textit{t1nTip}} have \textit{res: remove h e (T t1 x t2) = T (wrm h t1) (rm h t1) t2}
by (simp add: \textit{Let-def wrmrm-decomp})
from \textit{res} show ?thesis
proof simp
  from \textit{s have s1: \textit{sortedTree h t1} by simp}
  show \textit{insert (rm h t1) (setOf (wrm h t1) Un setOf t2) = insert x (setOf t1 Un setOf t2) = eqs h e}
    proof (simp add: \textit{t1nTip s1 rm-set wrm-set})
      show \textit{insert (rm h t1) (setOf t1 - \{rm h t1\} Un setOf t2) = insert x (setOf t1 Un setOf t2) = eqs h e}
        proof
          from \textit{t1nTip s1 rm-set} have \textit{o1: insert (rm h t1) (setOf t1 - \{rm h t1\} Un setOf t2) = setOf t1 Un setOf t2}
            by auto
          have \textit{o2: insert x (setOf t1 Un setOf t2) = eqs h e = setOf t1 Un setOf t2}
            proof
              from \textit{ex} have \textit{xOk: x : eqs h e by (simp add: eqs-def)}
              have \textit{t1Ok: (setOf t1) Int (eqs h e) = \{\}}
                proof (rule \textit{disjCond})
                  fix y :: 'a
                  assume \textit{y-in-t1: y : setOf t1}
                  assume \textit{y-in-eq: y : eqs h e}
                  from \textit{y-in-t1 s ex} have \textit{o1: h y < h e by auto}
                  from \textit{y-in-eq} have \textit{o2: \sim (h y < h e) by (simp add: eqs-def)}
                  from \textit{o1 o2} show \textit{False} by contradiction
                qed
                from \textit{xOk t1Ok t2Ok} show ?thesis by auto
            qed
            from \textit{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
    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: s 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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
            show \( \?thesis\)
            proof (cases h e < h l)
              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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
                show \( \?thesis\)
                proof (cases h e < h l)
                  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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
                    show \( \?thesis\)
                    proof (cases h e < h l)
                      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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
                        show \( \?thesis\)
                        proof (cases h e < h l)
                          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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
                            show \( \?thesis\)
                            proof (cases h e < h l)
                              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 \( \?C1\) = \( \forall l \in setOf t1. h l < h x\)
                                show \( \?thesis\)
                                proof (cases h e < h l)
                                  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 \( ?C2 = \forall r \in \text{setOf} \ (\text{remove} \ h \ e \ t2) \cdot h < h r \)

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 \( \text{thesis} \) by auto
qed

next case False note nxle = this
from nle nxle have ex: h e = h x by simp
show \( \text{thesis} \)
proof (cases t1 = T\(\text{ip}\))
  case True note t1tip = this
  from ex t1tip have 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 = this
from nle nxle ex t1nTip
have res: 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 \( \text{thesis} \)
proof simp
  let \( ?C1 = \text{sortedTree} \ h \ (\text{wrm} \ h \ t1) \)
  let \( ?C2 = \forall l \in \text{setOf} \ (\text{wrm} \ h \ t1) \cdot h l < h (\text{rm} \ h \ t1) \)
  let \( ?C3 = \forall r \in \text{setOf} \ t2 \cdot h (\text{rm} \ h \ t1) < h r \)
  let \( ?C4 = \text{sortedTree} \ h \ t2 \)
  from s1 t1nTip have a1: \( ?C1 \) by (simp add: wrm-sort)
  from s1 t1nTip have a2: \( ?C2 \) by (simp add: wrm-less-rm)
  have o3: \( ?C3 \)
proof
  fix r :: 'a
  assume rt2: \( r : \text{setOf} \ t2 \)
  from s rm-set s1 t1nTip have a1: \( h (\text{rm} \ h \ t1) < h x \) by auto
  from rt2 s have a2: \( h x < h r \) by auto
  from a1 a2 show \( h (\text{rm} \ h \ t1) < h r \) by simp
qed
from a1 a2 o3 s2 show \( \text{thesis} \) by simp
qed
qed
qed

We summarize the specification of remove as follows.

corollary remove-spec: \( \text{sortedTree} \ h \ t \longrightarrow \text{sortedTree} \ h \ (\text{remove} \ h \ e \ t) \) &
  \( \text{setOf} \ (\text{remove} \ h \ e \ t) = \text{setOf} \ t - \text{eqs} \ h \ e \)
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 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 tsome have mapOf t i = Some (snd ia) by (simp add: mapOf-def)
  from this h have o1: snd ia = a by simp
  from tlookup-some-inst tsome v have o2: fst ia = i 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) ∉ setOf t)
proof
  ==⇒ i.
  assume mapNone: mapOf t i = None
  from v mapNone mapOf-lookup3 have lnone: tlookup fst i t = None by auto
  show ∀ a. (i, a) ∉ 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 ∈ setOf t. fst x ~ = i)
      by (insert tlookup-none [of fst t i], assumption)
      from v lnone tlookup-none-inst have ∃ x ∈ setOf t. fst x ~ = i by simp
      from this iain have fst (i, a) ~ = i by fastforce
from this show False by simp
qed
qed

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

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
  — j==
  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 :: 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 by (simp add: mupdate-def)
from v binsert-set
have setSpec: setOf ?tr = setOf t - eqs fst (i,a) Un {(i,a)} by fastforce
from v binsert-sorted
have valid-tmap ?tr: valid-tmap ?tr by fastforce
show mapOf (mupdate i a t) i2 = ((mapOf t)(i |-> a)) i2 by (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 kvin: (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 kvin tlookup-finds-inst
show ?thesis by simp
qed
from upres will-find show ?thesis by (simp add: mapOf-def)
qed
from lhs-res rhs-res i2ei
show ?thesis by simp
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
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 apres 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 - 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
  qed

qed
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 t x r) = set-of t 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 & (\forall x \in set-of l. x < y) & (\forall z \in 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: sorted t −→ memb e t = (e : set-of t)
by (induct t) auto

15 Insertion operation

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

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

theorem binset-sorted: sorted t −→ sorted (binset x t)
by (induct t) (auto simp add: set-of-binset)

corollary binset-spec:
sorted t −→
  sorted (binset x t) &
  set-of (binset x t) = set-of t Un {x}
by (simp add: binset-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 \( \text{Nd } l \ y \) (remove \( x \) \( r \)) else
if \( l = \text{Tip} \) then \( r \)
else \( \text{Nd } (\text{rem } l) \ (\text{rem } l) \ r \)

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

**Lemma 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: set-of-rem)

**Lemma sorted-rem**: \( \| t \sim \text{Tip} \| \implies \text{sorted}(\text{rem } t) \in \{\text{rm } t\} \)
by (induct \( t \)) (auto simp add: set-of-rem split: if_splits)

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

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

**Corollary remove-spec**: summary specification of remove
\( \text{sorted } t \implies \text{sorted}(\text{remove } x \ t) \)
by (simp add: remove-sorted)

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

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

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

We can test this implementation by generating code.

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