BinarySearchTree
Larry Paulson
September 19, 2015

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 &
   (ALL l: setOf t1. h l < h x) &
   (ALL 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) --->(ALL x:setOf t. h x ~ k)
by (induct t, auto)

lemma tlookup-some:
  sortedTree h t & (tlookup h k t = Some x) ---> x:setOf t & h x = k
apply (induct t)
  — Just auto will do it, but very slowly
apply (simp)
apply (clarify, auto)
apply (simp-all split: split-if-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 \( hab : h \ a = h \ b \)
from \( s \) have \( s1 : \text{sortedTree} \ h \ t1 \) by auto
from \( s \) have \( s2 : \text{sortedTree} \ h \ t2 \) by auto
show \( a = b \)
— We consider 9 cases for the position of \( a \) and \( b \) are in the tree
proof
— three cases for \( a \)
from \( adef \) have \( a : \text{setOf} \ t1 \ | \ a = x \ | \ a : \text{setOf} \ t2 \) by auto
moreover \{ assume \( adef1 : a : \text{setOf} \ t1 \)
  have \( \text{thesis} \)
  proof
  — three cases for \( b \)
  from \( bdef \) have \( b : \text{setOf} \ t1 \ | \ b = x \ | \ b : \text{setOf} \ t2 \) by auto
  moreover \{ assume \( bdef1 : b : \text{setOf} \ t1 \)
    from \( s1 \) \( adef1 \) \( bdef \) \( habh \) \( h1 \) have \( \text{thesis} \) by simp \}
  moreover \{ assume \( bdef1 : b = x \)
    from adef1 bdef1 s have \( h \ a < h \ b \) by auto
    from this habh 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 habh have \( \text{thesis} \) by simp \}
  — case impossible
moreover \{ assume \( bdef1 : b = x \)
  from this have \( h \ b < h \ x \) by auto
  from this habh have \( \text{thesis} \) by simp \}
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 have \( h \ a < h \ b \) by auto
    from this habh have \( \text{thesis} \) by simp \}
moreover \{ assume \( bdef1 : b = x \)
  from this have \( h \ a < h \ b \) by simp
  from this habh have \( \text{thesis} \) by simp \}
  — case impossible
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 \( bdef1 : b : \text{setOf} \ t2 \)
  from this have \( h \ b < h \ a \) by auto
  from this habh have \( \text{thesis} \) by simp \}
moreover \{ assume \( bdef1 : b = x \)
  from this have \( h \ a < h \ b \) by simp
  from this habh have \( \text{thesis} \) by simp \}
  — case impossible
ultimately show \( \text{thesis} \) by blast
qed

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

lemma tlookup-finds: — if a node is in the tree, lookup finds it 
sortedTree h t & y : setOf t --> 
tlookup h (h y) t = Some y 
proof safe 
assume s: sortedTree h t assume yint: y : setOf t show tlookup h (h y) t = Some y proof (cases tlookup h (h y) t) case None note res = this from s res have sortedTree h t & (tlookup h (h y) t = None) by simp from this have o1: ALL x : setOf t. h x ~= h y by (simp add: tlookup-none) from o1 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 o1: z : setOf t & h z = h y by simp from s yint o1 sd have y = z by auto from this res show tlookup h (h y) t = Some y by simp qed qed 

3.1 Tree membership as a special case of lookup 
definition memb :: 'a hash => 'a => 'a Tree => bool where 
memb h x t == 
(case (tlookup h (h x) t) of 
None => False 

lemma assumes s: sortedTree h t
  shows memb-spec: 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)
next case (Some z) note tSome = this
  from s tNone tlookup-none have o1: ALL y: setOf t. h y ~ = h x by fastforce
  have notIn: x ~: setOf t
    proof (cases tlookup h (h x) t)
      case None note tNone = this
      have res: memb h x t = False by (simp add: memb-def)
next case (Some z) note tSome = this
  from s tSome tlookup-some have zin: z : 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)
        have zin: z : setOf t proof assume xin: x : setOf t
          from s tSome tlookup-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
          have notIn: x ~: setOf t by simp
          qed
          from this res show ?thesis by simp
        qed
      next case False
        note xnez = this
        from tSome xnez have res: ~ memb h x t by (simp add: memb-def)
        have x ~: setOf t proof (cases x = z)
          case True
            note xez = this
            from tSome xez have res: memb h x t by (simp add: memb-def)
            have x ~: setOf t
          next case False
            note xnez = this
            from tSome xnez have res: ~ memb h x t by (simp add: memb-def)
            have x ~: setOf t proof
              assume xin: x : setOf t
              from s tSome tlookup-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
              have notIn: x ~: setOf t by simp
              qed
              from this res show ?thesis by simp
            qed
          qed
        qed
    qed
  qed
next case (Some z)
  note tNone = this
  from tNone have res: memb h x t = False by (simp add: memb-def)
next case (Some z)
  note tSome = this
  from s tNone tlookup-none have o1: ALL y: setOf t. h y ~ = h x by fastforce
  have notIn: x ~: setOf t
  proof (cases tlookup h (h x) t)
    case None note tNone = this
    have res: memb h x t = False by (simp add: memb-def)
  qed
  from res notIn show ?thesis by simp
qed
from this res show ?thesis by simp
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** \textit{disjCond}: \[ \forall x. \forall x:A; x:B. x = \rightarrow \text{False} \implies A \cap B = \{\} \]
by \texttt{fastforce}

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

**Lemma** \textit{binsert-set}$\colon$ \textbf{sortedTree} \textit{h} \textit{t} $\implies$ \textit{setOf} (\textit{binsert} \textit{h} \textit{e} \textit{t}) = (\textit{setOf} \textit{t}) - (\textit{eqs} \textit{h} \textit{e}) \cup \{\textit{e}\}

\begin{proof}
\begin{itemize}
  \item \textbf{in} \textit{duct} \textit{t} \hfill \\
    \begin{itemize}
      \item \textbf{base} \textit{case} \\
        \textbf{show} \ ?P \ Tip \ \textbf{by} \ (\textit{simp} \ \textit{add}: \textit{eqs-def}) \\
      \begin{itemize}
        \item \textbf{induction step} \\
          \textbf{fix} \textit{t1} :: \textit{Tree} \ \ass \ ?P \ \textit{t1} \\
          \textbf{fix} \textit{t2} :: \textit{Tree} \ \ass \ ?P \ \textit{t2} \\
          \textbf{fix} \textit{x} :: \textit{a} \ \ass \ ?P \ \textit{(T t1 x t2)} \\
          \begin{proof}
            \begin{itemize}
              \item \textbf{assume} \textit{s} \colon \textbf{sortedTree} \textit{h} \ (\textit{T t1 x t2}) \\
              \textbf{from} \textit{s} \ \textbf{have} \textit{s1} :: \textbf{sortedTree} \textit{h} \ \textit{t1} \ \textbf{by} \ (\textit{rule} \textit{sortLemmaL}) \\
              \textbf{from} \textit{s1} \ \textbf{and} \textit{h1} \ \textbf{have} \textit{c1} :: \textbf{setOf} \ (\textit{binsert} \textit{h} \textit{e} \textit{t1}) = \textbf{setOf} \textit{t1} - (\textit{eqs} \textit{h} \textit{e}) \cup \{\textit{e}\} \\
              \textbf{by} \ \texttt{simp} \\
              \textbf{from} \textit{s1} \ \textbf{have} \textit{s2} :: \textbf{sortedTree} \textit{h} \ \textit{t2} \ \textbf{by} \ (\textit{rule} \textit{sortLemmaR}) \\
              \textbf{from} \textit{s2} \ \textbf{and} \textit{h2} \ \textbf{have} \textit{c2} :: \textbf{setOf} \ (\textit{binsert} \textit{h} \textit{e} \textit{t2}) = \textbf{setOf} \textit{t2} - (\textit{eqs} \textit{h} \textit{e}) \cup \{\textit{e}\} \\
              \textbf{by} \ \texttt{simp} \\
              \textbf{show} \ \textit{setOf} \ (\textit{binsert} \textit{h} \textit{e} \ (\textit{T t1 x t2})) = \textbf{setOf} \ (\textit{T t1 x t2}) - (\textit{eqs} \textit{h} \textit{e}) \cup \{\textit{e}\} \\
              \textbf{proof} \ (\textit{cases} \textit{h} \textit{e} < \textit{h} \textit{x}) \\
              \begin{itemize}
                \item \textbf{case} \ \textit{True} \ \textbf{note} \ \textit{eLess} = \textit{this} \\
                  \textbf{from} \ \textit{eLess} \ \textbf{have} \ \textit{res} :: \textit{binsert} \textit{h} \textit{e} \ (\textit{T t1 x t2}) = (\textit{T (binsert} \textit{h} \textit{e} \textit{t1}) \ \textit{x} \textit{t2}) \\
              \end{itemize}
            \end{itemize}
          \end{proof}
        \end{itemize}
      \end{itemize}
    \end{itemize}
  \end{itemize}
\end{proof}
from xLessT2 have eqsDisjT2: ALL el: eqs h e. ALL r: setOf t2. h el
\sim = h r
  by fastforce
from eqsDisjX eqsDisjT2 show ?thesis by fastforce
qed
next case False note eNotLess = this
show setOf (binsert h e (T t1 x t2)) = setOf (T t1 x t2) \sim 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) \sim eqs h e Un \{ e \}
proof (simp add: res xLess eNotLess c2)
  show insert e (insert x (setOf t1 Un setOf t2) \sim eqs h e) = insert e (insert x (setOf t1 Un setOf t2)) \sim eqs h e
  proof
    have XLessEqs: ALL el: eqs h e. h x < h el by (simp add: eqs-def xLess)
    from this have T1lessX: ALL l: setOf t1. h l < h x by auto
    have T1lessEqs: ALL el: eqs h e. ALL l: setOf t1. h l < h el
    proof safe
      fix el assume hel: el : eqs h e
      fix l assume hl: l : setOf t1
      from hel eqs-def have o1: h el = h e by fastforce
      from t1LessX hl o1 xLess show h l < h el by auto
    qed
    from T1lessEqs have T1disjEqs: ALL el: eqs h e. ALL l: 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 xeqe: h x = h e by simp
from xeqe 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) \sim eqs h e Un \{ e \}
proof (simp add: res eNotLess xeqe)
  show insert e (setOf t1 Un setOf t2) = insert e (insert x (setOf t1 Un setOf t2)) \sim 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 xeqe)
moreover have \((\text{setOf } t_1) \cap \text{Int } (\text{eqs } h\ e) = \{\}\)

proof (rule disjCond)
  fix \(w\)
  assume \(\text{whSet}: w : \text{setOf } t_1\)
  assume \(\text{whEq}; w : \text{eqs } h\ e\)
  from \(\text{whSet}\) s have \(o1: h\ w < h\ x\) by simp
  from \(\text{whEq}\) eqs-def have \(o2: h\ w = h\ e\) by fastforce
  from \(o2\) xeqe have \(o3: \neg h\ w < h\ x\) by simp
  from \(o1\ o3\) show False by contradiction
qed

moreover have \((\text{setOf } t_2) \cap \text{Int } (\text{eqs } h\ e) = \{\}\)

proof (rule disjCond)
  fix \(w\)
  assume \(\text{whSet}: w : \text{setOf } t_2\)
  assume \(\text{whEq}; w : \text{eqs } h\ e\)
  from \(\text{whSet}\) s have \(o1: h\ x < h\ w\) by simp
  from \(\text{whEq}\) eqs-def have \(o2: h\ w = h\ e\) by fastforce
  from \(o2\) xeqe have \(o3: \neg h\ x < h\ w\) by simp
  from \(o1\ o3\) show False by contradiction
qed

ultimately show ?thesis by auto
qed

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

lemma 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.

corollary 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)

5 Removing an element from a tree

These proofs are influenced by those in BinaryTree-Tactic

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

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

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

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

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

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

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

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

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

lemma \text{wrm-set}: \ t \sim \text{ Tip} \& \text{sortedTree } h \ t \dashv \vdash \text{setOf } (\text{wrm } h \ t) = \text{setOf } t - \{\text{rm } h \ t\} \ (is \ ?P \ t) \]
proof \text{(induct } t) \]
show \ ?P \ Tip \ by \ simp \]
fix \ t1 :: \text{'a Tree} \ assume \ h1: \ ?P \ t1 \]
fix \ t2 :: \text{'a Tree} \ assume \ h2: \ ?P \ t2 \]
fix \ x :: \text{'a} \]
show \( ?P (T \times x \times t2) \)
proof (rule impI, erule conjE)
  assume s: sortedTree h (T \times x \times t2)
  show setOf (wrm h (T \times x \times t2)) =
    setOf (T \times x \times t2) - {rm h (T \times x \times t2)}
  proof (cases t2 = Tip)
    case True note t2Tip = this
    from t2Tip have rm-res: rm h (T \times x \times t2) = x by simp
    from s have x \sim: 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 \times x \times t2) = rm 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-res wrm-res 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: ALL 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
        qed
  qed

lemma wrm-set1: \( t \sim= \) Tip & sortedTree h t \Longrightarrow setOf (wrm h t) \subseteq setOf t
by (auto simp add: wrm-set)

lemma wrm-sort: \( t \sim= \) Tip & sortedTree h t \Longrightarrow sortedTree h (wrm h t) (is \(?P t)
proof (induct t)
  show \(?P Tip by simp
  fix t1 :: 'a Tree assume h1: \(?P t1
  fix t2 :: 'a Tree assume h2: \(?P t2
  fix x :: 'a
  show \(?P (T \times x \times t2)
  proof safe
    assume s: sortedTree h (T \times x \times t2)
show sortedTree h (wrm h (T t1 x t2))
proof (cases t2 = Tip)
case True note t2tip = this
  from t2tip have res: wrm h (T t1 x t2) = t1 by simp
  from res s show ?thesis by simp
next case False note t2nTip = this
  from t2nTip have res: wrm h (T t1 x t2) = T t1 x (wrm h t2) by simp
  from s have s1: sortedTree h t1 by simp
  from s have s2: sortedTree h t2 by simp
  from s2 t2nTip have o1: sortedTree h (wrm h t2) by simp
  from s o2 have o3: ALL r: setOf (wrm h t2). h x < h r by auto
  from s1 o1 o3 res s show sortedTree h (wrm h (T t1 x t2)) by simp
qed
qed

lemma wrm-less-rm:
  t ∼ Tip & sortedTree h t ---->
  (ALL l:setOf (wrm h t). h l < h (rm h t)) (is ?P t)
proof (induct t)
show ?P Tip by simp
fix t1 :: 'a Tree assume h1: ?P t1
fix t2 :: 'a Tree assume h2: ?P t2
fix x :: 'a
show ?P (T t1 x t2)
proof safe
  fix l :: 'a assume ldef: l : setOf (wrm h (T t1 x t2))
  assume s: sortedTree h (T t1 x t2)
  from s have s1: sortedTree h t1 by simp
  from s have s2: sortedTree h t2 by simp
  show h l < h (rm h (T t1 x t2))
  proof (cases l = x)
    case True note lx = this
    from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
    from lx o1 show ?thesis by simp
  next case False note lx = this
  have l-scope: l : {x} Un setOf t1 Un setOf (wrm h t2) by simp
  have hLess: h l < h (rm h t2)
  proof (cases l = x)
    case True note lx = this
  from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
  from lx o1 show ?thesis by simp
  next case False note lx = this
  have l-scope: l : {x} Un setOf t1 Un setOf (wrm h t2) by simp
  have hLess: h l < h (rm h t2)
  proof (cases l = x)
    case True note lx = this
  from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
  from lx o1 show ?thesis by simp
next case False note lx = this
qed
show ?thesis
proof (cases l : setOf t1)
  case True note l-in-t1 = this
    from s t2n Tip 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 hx 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 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
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 ele show False by contradiction 
qed 
from xOk t2Ok show ?thesis 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 
from s have s2: sortedTree h t2 by simp 
from s2 h2 have o1: setOf (remove h e t2) = setOf t2 - eqs h e by simp 
show ?thesis 
proof (simp add: o1 xle nelx) 
show insert x (setOf t1 Un (setOf t2 - eqs h e)) = 
insert x (setOf t1 Un setOf t2) - eqs h e 
proof - 
have xOk: x ~: eqs h e 
proof 
assume h: x : eqs h e 
from h have o1: ~ (h x < h e) by (simp add: eqs-def) 
from xle o1 show False by contradiction 
qed 
have t1Ok: (setOf t1) Int (eqs h e) = {} 
proof (rule disjCond) 
fix y :: 'a 
assume y-in-t1: y : setOf t1 
assume y-in-eq: y : eqs h e 
from y-in-t1 s have ylx: h y < h x by auto 
from y-in-eq have eey: h y = h e by (simp add: eqs-def) 
from ylx eey have nxle: ~ (h x < h e) by simp 
from nxle ele show False by contradiction 
qed 
from xOk t1Ok show ?thesis by auto 
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 

14
from \( y \in \text{eq} \) have \( \text{cey} : h \ y = h \ e \) by (simp add: eqs-def)
from \( y \in \text{eq} \) \( \text{ex cey} \) have \( \text{nxly} : \sim (h \ x < h \ y) \) by simp
from \( \text{nxly xly} \) show False by contradiction
qed
show \(?\text{thesis}\)
proof (cases \( t1 = \text{Tip} \))
case True note \( t1\text{tip} = \) this
from \( \text{ex t1\text{tip}} \) have \( \text{res} : \) remove \( h \ e \) (\( T \ t1 \ x \ t2 \)) = \( t2 \) by simp
show \(?\text{thesis}\)
proof (simp add: \( t1\text{tip} \) \( \text{ex} \))
show \( \text{setOf t2} = \) insert \( x \) (\( \text{setOf t2} \)) − eqs \( h \ e \)
proof −
from \( \text{ex} \) have \( \text{x-in-eqs} : x : \) eqs \( h \ e \) by (simp add: eqs-def)
from \( \text{x-in-eqs t2Ok} \) show \(?\text{thesis}\) by auto
qed
qed
next case False note \( t1\text{nTip} = \) this
from \( \text{nelx nxle ex t1\text{nTip}} \) have \( \text{res} : \) remove \( h \ e \) (\( T \ (\text{wrm h t1}) \ (\text{rm h t1}) \ t2 \)) by (simp add: Let-def wrmrm-decomp)
from \( \text{res} \) show \(?\text{thesis}\)
proof simp
from \( \text{s} \) have \( s1 : \) sortedTree \( h \ t1 \) by simp
show \( \text{insert} (\text{rm h t1}) (\text{setOf (wrm h t1)}) \text{ Un setOf t2}) = \) insert \( x \) (\( \text{setOf t1 Un setOf t2} \)) − eqs \( h \ e \)
proof (simp add: \( t1\text{nTip} s1 \) \( \text{rm-set wrm-set} \))
show \( \text{insert} (\text{rm h t1}) (\text{setOf t1 \{\text{rm h t1}\}} \text{ Un setOf t2}) = \) insert \( x \) (\( \text{setOf t1 Un setOf t2} \)) − eqs \( h \ e \)
proof −
from \( \text{t1\text{nTip}} s1 \) \( \text{rm-set} \)
have \( o1 : \text{insert} (\text{rm h t1}) (\text{setOf t1 \{\text{rm h t1}\}} \text{ Un setOf t2}) = \) setOf \( t1 \text{ Un setOf t2} \) by auto
have \( o2 : \text{insert x (setOf t1 Un setOf t2}) = \) eqs \( h \ e \) = setOf \( t1 \text{ Un setOf t2} \)
proof −
from \( \text{ex} \) have \( \text{xOk} : x : \) eqs \( h \ e \) by (simp add: eqs-def)
have \( t1\text{Ok} : (\text{setOf t1}) \text{ Int (eqs h e)} = \) \( \)\( \emptyset \)
proof (rule disjCond)
fix \( y \) :: \( 'a \)
assume \( \text{y-in-t1} : y : \text{setOf t1} \)
assume \( \text{y-in-eq} : y : \) eqs \( h \ e \)
from \( \text{y-in-t1 s ex} \) have \( o1 : h \ y < h \ e \) by auto
from \( \text{y-in-eq} \) have \( o2 : \sim (h \ y < h \ e) \) by (simp add: eqs-def)
from \( o1 \) \( o2 \) show False by contradiction
qed
from \( \text{xOk t1\text{Ok}} t2\text{Ok} \) show \(?\text{thesis}\) by auto
qed
from \( o1 \) \( o2 \) show \(?\text{thesis}\) 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 = ALL l: setOf (remove h e t1). h l < h x
        let ?C2 = ALL r: 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 = ALL l: setOf t1. h l < h x
let \(?C2 = \text{ALL } r : \text{setOf } (\text{remove } h e \ t2), \ h x < h r\)

have o2: \(?C2\)

proof –

from s2 have \(\text{setOf } (\text{remove } h e \ t2) = \text{setOf } t2 - \text{eqs } h e\) by (simp add: remove-set)

from s this show \(?\text{thesis}\) by auto

qed

from o2 s show \(?C1 \& \?C2\) by auto

qed

next case False

note nxle = this

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

show \(?\text{thesis}\)

proof (cases \(t1 = \text{Tip}\))

case True

note t1tip = this

from ex t1tip have res: remove \(h e \ (T \ t1 \ x \ t2) = t2\) by simp

show \(?\text{thesis}\)

next case False

note t1nTip = this

from \(\text{neq t1nTip}\) ex t1nTip have res: remove \(h e \ (T \ t1 \ x \ t2) = T \ (\text{wrm } h \ t1) \ (\text{rm } h \ t1) \ t2\)

by (simp add: Let-def wrmrm-decomp)

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

proof simp

let \(?C1 = \text{sortedTree } h \ (\text{wrm } h \ t1)\)

let \(?C2 = \text{ALL } l : \text{setOf } (\text{wrm } h \ t1), \ h l < h (\text{rm } h \ t1)\)

let \(?C3 = \text{ALL } r : \text{setOf } t2, \ h (\text{rm } h \ t1) < h r\)

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 a3: \(?C3\)

proof

fix r :: 'a

assume rt2: \(r : \text{setOf } t2\)

from s \(\text{rm-set } s1\) 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 a3 s2 show \(?C1 \& \?C2 \& \?C3 \& \?C4\) 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 (bininsert id 4 (bininsert id 3 Tip)))

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

6 Mostly Isar-style Reasoning for Binary Tree Operations

theory BinaryTree-Map imports BinaryTree begin

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

7 Map implementation and an abstraction function

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

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

declare valid-tmap-def [simp]

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

8 Auxiliary Properties of our Implementation

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

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

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

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

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

\text{lemma assumes} \text{v: valid-tmap t}
\text{shows mapset-none: \text{(mapOf \text{t i} = None)} = (\text{ALL a. (i, a) \neg\neg: setOf t)}}
proof
\text{— ==\neg\neg i.}
\text{assume mapNone: \text{mapOf \text{t i} = None}}
\text{from v mapNone \text{mapOf-lookup3 have lnone: lookup \text{fst} \text{i t} = None by auto}}
\text{show \text{ALL a. (i, a) \neg\neg: setOf t}}
proof
\text{fix a}
\text{show (i, a) \neg\neg: setOf t}
proof
\text{assume iain: (i, a) : setOf t}
\text{have lookup-none-inst:}
\text{sortedTree \text{fst t \& (lookup \text{fst} \text{i t} = None) \longrightarrow (\text{ALL x: setOf t. \text{fst x} \neg\neg = i)}}
\text{by (insert lookup-none [of \text{fst \text{i t}], assumption}}
\text{from v lnone lookup-none-inst have \text{ALL x : setOf t. \text{fst x} \neg\neg = i by simp}}
\text{from this iain have \text{fst} (i,a) \neg\neg = i by fastforce}
from this show False by simp
qed
qed

next assume h: ALL a. (i,a) ~: 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 show (i,a) : setOf t by simp
— ==¿
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 array => 'a array 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 vr: valid-tmap ?tr by fastforce
show mapOf (mupdate i a t) i2 = ((mapOf t)(i |-> a)) i2
proof (cases i = i2)
case True
note i2ei = this
from i2ei have rhs-res: ((mapOf t)(i |-> a)) i2 = Some a by simp
have lhs-res: mapOf (mupdate i a t) i = Some a
proof
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
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: ALL a. (i2,a) ~: setOf t by fastforce
have noneIn: ALL b. (i2,b) ~: 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
have mapset-none-inst:
valid-tmap ?tr \rightarrow (mapOf ?tr i2 = None) = (\forall a. (i2, a) \sim\subseteq 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

from lhs-res rhs-res 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 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

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 \simsetOf {fst (i, undefined)}
  by fastforce
  have noneIn: \forall a. (i, a) \sim\subseteq setOf ?tr
  proof
fix a
from remSet show (i, a) ~: setOf ?tr by (simp add: eqs-def)
qed
from v remove-sort have vr: valid-tmap ?tr by fastforce
have mapset-none-inst: valid-tmap ?tr ==> (mapOf ?tr i = None) = (ALL a. (i, a) ~: setOf ?tr)
by (insert mapset-none [of ?tr i], simp)
from vr this have (mapOf ?tr i = None) = (ALL 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 & (ALL x:set-of l. x < y) & (ALL z:set-of r. y < z))

14 Tree Membership

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

**Lemma** member-set: sorted \( t \rightarrow \text{memb } e \text{ t } = (e : \text{set-of } t) \)

by (induct \( t \)) auto

### 15 Insertion operation

**Primrec** \( \text{binsert :: nat } => \text{ tree } => \text{ tree} \)
— Insert a node into sorted tree.

**Where**
\[ \text{binsert } x \text{ Tip } = (\text{Nd } \text{ Tip } x \text{ Tip}) \]
| \[ \text{binsert } x \text{ (Nd } t1 \text{ y } t2) = (\text{if } x < y \text{ then}
| (\text{Nd } \text{ (binsert } x \text{ t1}) \text{ y } t2)
| \text{else}
| (\text{if } y < x \text{ then}
| (\text{Nd } t1 \text{ y } (\text{binsert } x \text{ t2}))
| \text{else } (\text{Nd } t1 \text{ y } t2)) \]

**Theorem** set-of-binsert [simp]: set-of (binsert \( x \text{ t } ) = \text{set-of } t \text{ Un } \{x\}

by (induct \( t \)) auto

**Theorem** binsert-sorted: sorted \( t \rightarrow \text{sorted } (\text{binsert } x \text{ t }) \)

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

**Corollary** binsert-spec:
sorted \( t \rightarrow \text{spec:}
| \text{set-of } (\text{binsert } x \text{ t }) \land
| \text{set-of } (\text{binsert } x \text{ t }) = \text{set-of } t \text{ Un } \{x\}

by (simp add: binsert-sorted)

### 16 Remove operation

**Primrec**
\( \text{rm :: tree } => \text{ nat } \rightarrow \text{nat} \)
— find the rightmost element in the tree.

**Where**
\[ \text{rm}(\text{Nd } l \text{ x } r) = (\text{if } r = \text{Tip then } x \text{ else } \text{rm } r) \]

**Primrec**
\( \text{rem :: tree } => \text{ tree } \rightarrow \text{tree} \)
— find the tree without the rightmost element

**Where**
\[ \text{rem}(\text{Nd } l \text{ x } r) = (\text{if } r=\text{Tip then } l \text{ else } \text{Nd } l \text{ x } (\text{rem } r)) \]

**Primrec**
\( \text{remove: nat } => \text{ tree } => \text{ tree} \rightarrow \text{remove a node from sorted tree} \)

**Where**
remove \( x \) Tip = Tip

\[
\text{remove } x \ (\text{Nd } l \ y \ r) = \\
\begin{cases} 
  (\text{if } x < y \text{ then Nd (remove } x \ l) \ y \ r \text{ else}) \\
  (\text{if } y < x \text{ then Nd } l \ y \ (\text{remove } x \ r) \text{ else}) \\
  (\text{if } l = \text{Tip} \text{ then } x) \\
  \text{else Nd (rem } l \) (rm } l \ r)
\end{cases}
\]

**lemma** rm-in-set-of: \( t \sim= \text{Tip} \implies \text{set-of } t \)

by (induct \( t \)) auto

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

by (induct \( t \)) auto

**lemma** [simp]: \( \begin{array}{l} \mid t \sim= \text{Tip}; \text{sorted } t \mid \implies \text{sorted}(\text{rem } t) \end{array} \)

by (induct \( t \)) (auto simp add: set-of-rem)

**lemma** sorted-rem: \( \begin{array}{l} \mid t \sim= \text{Tip}; x \in \text{set-of } (\text{rem } t); \text{sorted } t \mid \implies x < \text{rm } t \end{array} \)

by (induct \( t \)) (auto simp add: set-of-rem split: if-splits)

**theorem** set-of-remove [simp]: \( \begin{array}{l} \text{sorted } t \implies \text{set-of } (\text{rem } x \ t) = \text{set-of } t - \{ x \} \end{array} \)

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: \( \begin{array}{l} \text{sorted } t \implies \text{sorted} (\text{remove } x \ t) \end{array} \)

by (induct \( t \)) (auto intro: less-trans rm-in-set-of sorted-rem)

**corollary** remove-spec: — summary specification of remove

\( \begin{array}{l} \text{set-of } \text{remove } x \ t \end{array} \)

by (simp add: remove-sorted)

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

**primrec** remrm :: tree \Rightarrow tree \star nat

where \(\text{remrm}(\text{Nd } l \ x \ r) = (\text{if } r = \text{Tip} \text{ then } (l, x) \text{ else}) \)

\(\begin{array}{l} \text{let } (r', y) = \text{remrm } r \text{ in } (\text{Nd } l \ x \ r', y) \end{array} \)

**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.

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

\texttt{export-code test in SML module-name BinaryTree-TacticStyle-Code file BinaryTree-TacticStyle-Code.ML}

\texttt{end}