BinarySearchTree

Larry Paulson

May 27, 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 \( \triangleright \)
  — apart from that, we do not rely on index being int
  index = int

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

definition eqs :: 'a hash => 'a => 'a set
  — equivalence class of elements with the same hash code
where
  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 => 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 $hahb \colon h \ a = h \ b$
from $s$ have $s1 \colon \text{sortedTree} \ h \ t1$ by auto
from $s$ have $s2 \colon \text{sortedTree} \ h \ t2$ 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 $adef1 \colon a \ : \ \text{setOf} \ t1$
  have $?thesis$
  proof
    — three cases for $b$
from $bdef$ have $b \ : \ \text{setOf} \ t1 \ | \ b = x \ | \ b \ : \ \text{setOf} \ t2$ by auto
moreover { assume $bdef1 \colon b \ : \ \text{setOf} \ t1$
  from $s1$ $adef1$ $bdef1$ $hahb$ $h1$ have $?thesis$ by simp }
moreover { assume $bdef1 \colon b = x$
  from $adef1$ $bdef1$ $s$ have $h \ a < h \ b$ by auto
  from this $hahb$ have $?thesis$ by simp }
moreover { assume $bdef1 \colon b \ : \ \text{setOf} \ t2$
  from $adef1$ $s$ have $o1 \colon h \ a < h \ x$ by auto
  from $bdef1$ $s$ have $o2 \colon h \ x < h \ b$ by auto
  from $o1$ $o2$ have $h \ a < h \ b$ by simp
  from this $hahb$ have $?thesis$ by simp } — case impossible
ultimately show $?thesis$ by blast
qed
}
moreover { assume $adef1 \colon a = x$
  have $?thesis$
  proof
    — three cases for $b$
from $bdef$ have $b \ : \ \text{setOf} \ t1 \ | \ b = x \ | \ b \ : \ \text{setOf} \ t2$ by auto
moreover { assume $bdef1 \colon b \ : \ \text{setOf} \ t1$
  from this $s$ have $h \ b < h \ x$ by auto
  from $this$ $hahb$ $adef1$ have $h \ b < h \ a$ by auto
  from $this$ $hahb$ this have $?thesis$ by simp }
moreover { assume $bdef1 \colon b = x$
  from $adef1$ $bdef1$ have $?thesis$ by simp }
moreover { assume $bdef1 \colon b \ : \ \text{setOf} \ t2$
  from this $s$ have $h \ x < h \ b$ by auto
  from $this$ $adef1$ have $h \ a < h \ b$ by simp
  from $this$ $hahb$ this have $?thesis$ by simp } — case impossible
ultimately show $?thesis$ by blast
qed
}
moreover { assume $adef1 \colon a \ : \ \text{setOf} \ t2$
  have $?thesis$
  proof
    — three cases for $b$
from $bdef$ have $b \ : \ \text{setOf} \ t1 \ | \ b = x \ | \ b \ : \ \text{setOf} \ t2$ by auto

moreover \{ assume \ bdef1: \ b : \ setOf \ t1 \\
from \ bdef1 \ s \ have \ o1: \ h \ b < h \ x \ by \ auto \\
from \ adef1 \ s \ have \ o2: \ h \ x < h \ a \ by \ auto \\
from \ o1 \ o2 \ have \ h \ b < h \ a \ by \ simp \\
from \ this \ hahb \ have \ ?thesis \ by \ simp \} — case impossible 
moreover \{ assume \ bdef1: \ b = x \\
from \ adef1 \ bdef1 \ s \ have \ h \ b < h \ a \ by \ auto \\
from \ this \ hahb \ have \ ?thesis \ by \ simp \} — case impossible 
moreover \{ assume \ bdef1: \ b : \ setOf \ t2 \\
from \ s2 \ adef1 \ bdef1 \ hahb \ h2 \ have \ ?thesis \ by \ simp \} 
ultimately show \ ?thesis \ by \ blast 
qed 

ultimately show \ ?thesis \ by \ blast 
qed 

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 

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 $\text{memb-spec}$: $\text{memb} \; h \; x \; t = (x : \text{setOf} \; t)$
proof (cases $\text{tlookup} \; h \; (h \; x)$ $t$)
case $\text{None}$
  note $\text{tNone} = \text{this}$
  from $\text{tNone}$ have $\text{res}$: $\text{memb} \; h \; x \; t = \text{False} \; \text{by} \; (\text{simp add}: \text{memb-def})$
  have $\text{notIn}$: $x \sim: \text{setOf} \; t$
    proof
      assume $h \: x : \text{setOf} \; t$
      from $h \: \text{a1}$ have $h \: x \sim = h \: x \; \text{by fastforce}$
      from this show $\text{False} \; \text{by simp}$
    qed
  from $\text{res notIn}$ show $?\text{thesis}$ by simp
next case ($\text{Some}$ $z$)
  note $\text{tSome} = \text{this}$
  from $\text{s}$ $\text{tSome lookup-some}$ have $\text{zin}$: $z : \text{setOf} \; t \; \text{by fastforce}$
  show $?\text{thesis}$
  proof (cases $x=$$z$)
    case $\text{True}$
      note $\text{xez} = \text{this}$
      from $\text{tSome xez}$ have $\text{res}$: $\text{memb} \; h \; x \; t$ by ($\text{simp add}: \text{memb-def}$)
      from $\text{res zin xez}$ show $?\text{thesis}$ by simp
    next case $\text{False}$
      note $\text{xnez} = \text{this}$
      from $\text{tSome xnez}$ have $\sim \; \text{memb} \; h \; x \; t$ by ($\text{simp add}: \text{memb-def}$)
      have $x \sim: \text{setOf} \; t$
        proof
          assume $\text{xin}$: $x : \text{setOf} \; t$
          from $\text{s}$ $\text{tSome lookup-some}$ have $\text{hzx}$: $h \: x = h \: z$ \text{by fastforce}
          have $\text{a1}$: $\text{sorted-distinct-pred} \; h \; x \; z \; t$
            by ($\text{insert}$ $\text{sorted-distinct}$ \text{of} $h \; x \; z \; t$, simp)
          from $\text{s}$ $\text{ixin}$ $\text{zin}$ $\text{hzx}$ $\text{a1}$ have $x = z$ \text{by fastforce}
          from this $\text{xnez}$ show $\text{False}$ \text{by simp}
        qed
      from this $\text{res}$ show $?\text{thesis}$ by simp
    qed
  qed

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

4 Insertion into a Tree

primrec
  $\text{binsert}$ :: $'a$ hash $=>$ $'a$ $=>$ $'a$ Tree $=>$ $'a$ Tree
where
  $\text{binsert}$ $h$ $e$ $\text{Tip}$ $= (T \; \text{Tip} \; e \; \text{Tip})$
| $\text{binsert}$ $h$ $e$ ($T \; t1 \; x \; t2$) $=$ ($\text{if}$ $h \; e < h \; x$ \text{then}
    ($T \; (\text{binsert} \; h \; e \; t1) \; x \; t2$)
  else
    ($\text{if}$ $h \; x < h \; e$ \text{then}


A technique for proving disjointness of sets.

**lemma** disjCond: \[ | x. | x:A; x:B | ==> False \] \[ ==> A \cap B = \{\} \] 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: `sortedTree h \ t --\> setOf (binsert h e \ t) = (setOf \ t) - (eqs h e Un \{e\})`

(\textbf{is} ?P \ t)

**proof** (induct \ t)

— base case

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

— induction step

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

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

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

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

**proof**

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

\textbf{from} \ s \ \textbf{have} \ s1: \ sortedTree \ h \ t1 \ by \ (rule \ sortLemmaL)

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

by simp

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

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

by simp

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

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

\textbf{case} \ True \ \textbf{note} \ eLess = \ this

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

by simp

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

**proof** (simp add: res eLess c1)

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

**proof** —

\textbf{have} \ eqsLessX: \ \textit{ALL} el: \ eqs \ h \ e, \ h \ el < \ h \ x \ by \ (simp \ add: \ eqs-def \ eLess)

\textbf{from} \ this \ \textbf{have} \ eqsDisjX: \ \textit{ALL} el: \ eqs \ h \ e, \ h \ el \ =\ h \ x \ by \ fastforce

\textbf{from} \ s \ \textbf{have} \ xLessT2: \ \textit{ALL} r: \ setOf \ t2, \ h \ x < \ h \ r \ by \ auto

\textbf{have} \ eqsLessT2: \ \textit{ALL} el: \ eqs \ h \ e, \ \textit{ALL} r: \ setOf \ t2, \ h \ el < \ h \ r

**proof** safe

\textbf{fix} \ el \ \textbf{assume} \ hel: \ el : \ eqs \ h \ e

\textbf{from} \ hel \ eqs-def \ \textbf{have} \ a1: \ h \ el = \ h \ e \ by \ fastforce

\textbf{fix} \ r \ \textbf{assume} \ hr: \ r : \ setOf \ t2
from xLessT2 hr o1 eLess show \( h \ el < h r \) by auto

qed

from eqsLessT2 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 (\binset h e (T t1 x t2)) = setOf (T t1 x t2) - eqs h e Un \{ e \}

proof (cases h x < h e)

- case True note xLess = this

  from xLess have res: \binset h e (T t1 x t2) = (T t1 x (\binset h e t2)) by simp

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

  proof (simp add: res xLess eNotLess c2)

- have XLessEqs: ALL el: eqs h e. h x < h el by (simp add: eqs-def xLess)

  from this have eqsDisjX: ALL el: eqs h e. h el \sim = h x by auto

  from s 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 xege: h x = h e by simp

from xege have res: \binset h e (T t1 x t2) = (T t1 e t2) by simp

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

proof (simp add: res eNotLess xege)

  show insert e (setOf t1 Un setOf t2) =
   insert e (insert x (setOf t1 Un setOf t2) - eqs h e)

  proof -

  - have insert x (setOf t1 Un setOf t2) - eqs h e =
   setOf t1 Un setOf t2

  proof -

  - have x : eqs h e by (simp add: eqs-def xege)
moreover have \((\text{setOf } t1) \cap \text{Int } (\text{eqs } h \ e) = \{\}\)

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

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

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

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

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

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

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

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

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

5 Removing an element from a tree

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

primrec \(\text{rm} :: \text{a hash} => \text{a Tree} => \text{a}\)
  — rightmost element of a tree
where
rm h (T t1 x t2) =
  (if t2 = Tip then x else rm h t2)

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

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

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

theorem wrmrm-decomp: t ~ Tip --> wrmrm h t = (wrm h t, rm h t)
apply (induct-tac t)
apply simp-all
done

lemma rm-set: t ~ Tip & sortedTree h t --> rm h t : setOf t
apply (induct-tac t)
apply simp-all
done

lemma wrm-set: t ~ Tip & sortedTree h t -->
  setOf (wrm h t) = setOf t - {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 $\exists P \ (T \ t1 \times t2)$

proof (rule impI, erule conjE)
assume $s \colon \text{sortedTree} \ h \ (T \ t1 \times t2)$
show $\setOf (\text{wrm} \ h \ (T \ t1 \times t2)) = \setOf (T \ t1 \times t2) - \{\text{rm} \ h \ (T \ t1 \times t2)\}$
proof (cases t2 = \text{Tip})
case True
note \text{t2\_tip} = this
from \text{t2\_tip} have \text{rm\_res}: \text{rm} \ h \ (T \ t1 \times t2) = x \ by \ simp
from \text{s} have x \sim: \setOf t1 \ by \ auto
from this \text{rm\_res} \text{ wrm\_res} \text{t2\_tip} show \text{?thesis} by simp
next case False
note \text{t2\_n\_tip} = this
from \text{t2\_n\_tip} have \text{rm\_res}: \text{rm} \ h \ (T \ t1 \times t2) = \text{rm} \ h \ t2 \ by \ simp
from \text{s} have s2: \text{sortedTree} \ h \ t2 \ by \ simp
from \text{h2} \text{t2\_n\_tip} \text{ s2} have o1: \setOf (\text{wrm} \ h \ t2) = \setOf t2 - \{\text{rm} \ h \ t2\} \ by \ simp
show \text{?thesis}
proof (simp add: \text{rm\_res} \text{ wrm\_res} \text{t2\_n\_tip} \text{ h2} \text{ o1})
show \text{insert} \ x \ (\setOf t1 \ Un \ (\setOf t2 - \{\text{rm} \ h \ t2\})) = \text{insert} \ x \ (\setOf t1 \ Un \ \setOf t2) - \{\text{rm} \ h \ t2\}
proof
from \text{s \ rm\_set} \text{t2\_n\_tip} have \text{x\_ok}: h \ x < h \ (\text{rm} \ h \ t2) \ by \ auto
have t1\_ok: ALL \ l : \setOf t1. h \ l < h \ (\text{rm} \ h \ t2)
proof safe
fix l :: 'a
assume \text{l\_def}: \text{l} : \setOf t1
from \text{l\_def} \text{s} have \text{l\_x}: \text{h \ l < h \ x} \ by \ auto
from \text{l\_x} \text{x\_ok} show \text{h \ l < h \ (rm \ h \ t2)} \ by \ auto
qed
from \text{x\_ok} \text{t1\_ok} show \text{?thesis} \ by \ auto
qed
qed
qed
qed

lemma \text{wrm\_set1}: t \sim \text{ Tip} \& \text{sortedTree} \ h \ t \rightarrow \setOf (\text{wrm} \ h \ t) <= \setOf t
by (auto simp add: \text{wrm\_set})

lemma \text{wrm\_sort}: t \sim \text{ Tip} \& \text{sortedTree} \ h \ t \rightarrow \text{sortedTree} \ h \ (\text{wrm} \ h \ t) \ (\text{is} \ ?P \ t)
proof (induct t)
show ?P \text{ Tip} \ by \ simp
fix \text{t1} :: 'a \text{ Tree} \ assume \text{h1}: ?P \text{ t1}
fix \text{t2} :: 'a \text{ Tree} \ assume \text{h2}: ?P \text{ t2}
fix x :: 'a
show ?P \ (T \ t1 \times t2)
proof safe
assume s: \text{sortedTree} \ h \ (T \ t1 \times t2)
show $\text{sortedTree } h \ (\text{wrm } h \ (T \ t1 \ x \ t2))$
proof (cases $t2 = \text{Tip}$)
  case True note $t2tip = \text{this}$
    from $t2tip$ have res: $\text{wrm } h \ (T \ t1 \ x \ t2) = t1$ by simp
    from res s show ?thesis by simp
  next case False note $t2\neg\text{Tip} = \text{this}$
    from $t2\neg\text{Tip}$ have res: $\text{wrm } h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\text{wrm } h \ t2)$ by simp
    from s have s1: $\text{sortedTree } h \ t1$ by simp
    from s have s2: $\text{sortedTree } h \ t2$ by simp
    from s2 $h2 \ t2\neg\text{Tip}$ have o1: $\text{sortedTree } h \ (\text{wrm } h \ t2)$ by simp
    from s o2 have o3: $\text{ALL } r: \text{setOf } (\text{wrm } h \ t2), h \ l < h (\text{rm } h \ t2)$ by auto
    from o2 have o4: $\text{setOf } \ (\text{wrm } h \ t2) < = \text{setOf } t2$ by auto
    from s1 o1 o3 res s show $\text{sortedTree } h \ (\text{wrm } h \ (T \ t1 \ x \ t2))$ by simp
  qed
qed

lemma wrm-less-rm:
  $t \sim \text{ Tip } \& \ \text{sortedTree } h \ t \longrightarrow$
  ($\text{ALL } l: \text{setOf } (\text{wrm } h \ t), h \ l < h (\text{rm } h \ t))$ (is $?P \ t$)
proof (induct t)
  show $?P \ \text{Tip}$ by simp
  fix $t1 :: 'a \ \text{Tree}$ assume h1: $?P \ t1$
  fix $t2 :: 'a \ \text{Tree}$ assume h2: $?P \ t2$
  fix x :: 'a
  show $?P \ (T \ t1 \ x \ t2)$
  proof safe
    fix l :: 'a assume ldef: $l: \text{setOf } (\text{wrm } h \ (T \ t1 \ x \ t2))$
    assume s: $\text{sortedTree } h \ (T \ t1 \ x \ t2)$
    from s have s1: $\text{sortedTree } h \ t1$ by simp
    from s have s2: $\text{sortedTree } h \ t2$ by simp
    show $h \ l < h (\text{rm } h \ (T \ t1 \ x \ t2))$
    proof (cases $t2 = \text{Tip}$)
      case True note $t2\text{tip} = \text{this}$
      from $t2\text{tip}$ have rm-res: $\text{rm } h \ (T \ t1 \ x \ t2) = x$ by simp
      from $t2\text{tip}$ have wrm-res: $\text{wrm } h \ (T \ t1 \ x \ t2) = t1$ by simp
      from ldef wrm-res have o1: $l: \text{setOf } t1$ by simp
      from rm-res o1 s show ?thesis by simp
    next case False note $t2\neg\text{tip} = \text{this}$
      from $t2\neg\text{tip}$ have rm-res: $\text{rm } h \ (T \ t1 \ x \ t2) = \text{rm } h \ t2$ by simp
      from $t2\neg\text{tip}$ have wrm-res: $\text{wrm } h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\text{wrm } h \ t2)$ by simp
      from ldef wrm-res have l-scope: $l: \{x\}$ $\text{Un } \text{setOf } t1 \ \text{Un } \text{setOf } (\text{wrm } h \ t2)$ by simp
      have hLess: $h \ l < h (\text{rm } h \ t2)$
      proof (cases $l = x$)
        case True note $lx = \text{this}$
        from s $t2\neg\text{tip}$ rm-set s2 have o1: $h \ x < h (\text{rm } h \ t2)$ by auto
        from lx o1 show ?thesis by simp
      next case False note $lx = \text{this}$
    qed
  qed
show ?thesis
proof (cases l : setOf t1)
  case True note l-in-t1 = this
    from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
    from l-in-t1 s have o2: h l < h x by auto
    from o1 o2 show ?thesis by simp
next case False note l-notin-t1 = this
  from l-scope lnx 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
qed
qed
qed

lemma remove-set: sortedTree h t \longrightarrow
setOf (remove h e t) = setOf t \setminus 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) \setminus eqs h e
    proof (cases h e < h x)
      case True note elx = this
      from elx have res: remove h e (T t1 x t2) = T (remove h e t1) x t2
      by simp
      from s have s1: sortedTree h t1 by simp
      from s1 h1 have o1: setOf (remove h e t1) = setOf t1 \setminus eqs h e by simp
      show ?thesis
      proof (simp add: o1 elx)
        show insert x (setOf t1 \setminus eqs h e Un setOf t2) =
          insert x (setOf t1 Un setOf t2) \setminus eqs h e
        proof
          have xOk: x \sim\: eqs h e
          proof
            assume h: x : eqs h e
            from h have o1: \sim\ (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
            
  13
assume y-in-eq; y : eqs h e
from y-in-t2 s have zly: h x < h y by auto
from y-in-eq have eey: h y = h e by (simp add: eqs-def)
from zly eey have nelx: ~ (h e < h x) by simp
from nelx xle show False by contradiction
qed
from xOk t2Ok show ?thesis by auto
qed

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)
have xOk: x ~: eqs h e
proof
assume h: x : eqs h e
from h have o1; ~ (h x < h e) by (simp add: eqs-def)
from xle o1 show False by contradiction
qed
have t1Ok: (setOf t1) Int (eqs h e) = {}
proof (rule disjCond)
fix y :: 'a
assume y-in-t1: y : setOf t1
assume y-in-eq: y : eqs h e
from y-in-t1 s have ylx: h y < h x by auto
from y-in-eq have eey: h y = h e by (simp add: eqs-def)
from ylx eey have nelx: ~ (h x < h e) by simp
from nelx xle show False by contradiction
qed
from xOk t1Ok show ?thesis by auto
qed

next case False note nxle = this
from nelx nxle have ex: h e = h x by simp
have t2Ok: (setOf t2) Int (eqs h e) = {}
proof (rule disjCond)
fix y :: 'a
assume y-in-t2: y : setOf t2
assume y-in-eq: y : eqs h e
from y-in-t2 s have zly: h x < h y by auto
from y-in-eq have eey: h y = h e by (simp add: eqs-def)
from y-in-eq ex eey have nxly: ~ (h x < h y) by simp
from nxly xly show False by contradiction
qed
show ?thesis
proof (cases t1 = Tip)
case True note t1tip = this
from ex t1tip have res: remove h e (T t1 x t2) = t2 by simp
show ?thesis
proof (simp add: res t1tip ex)
  show setOf t2 = insert x (setOf t2) − eqs h e
  proof
    from ex have x-in-eqs: x : eqs h e by (simp add: eqs-def)
    from x-in-eqs t2Ok show ?thesis by auto
  qed
qed
next case False note t1nTip = this
from nelx nxle ex t1nTip have res: remove h e (T t1 x t2) = T (wrm h t1) (rm h t1) t2
by (simp add: Let-def wrmrm-decomp)
from res show ?thesis
proof simp
  from s have s1: sortedTree h t1 by simp
  show insert (rm h t1) (setOf (wrm h t1) ∩ setOf t2) = insert x (setOf t1 ∩ setOf t2) − eqs h e
  proof (simp add: t1nTip s1 rm-set wrm-set)
    show insert (rm h t1) (setOf t1 − {rm h t1}) ∩ setOf t2 = insert x (setOf t1 ∩ setOf t2) − eqs h e
    proof
      from t1nTip s1 rm-set
      have o1: insert (rm h t1) (setOf t1 − {rm h t1}) ∩ setOf t2 = setOf t1 ∩ setOf t2 by auto
      have o2: insert x (setOf t1 ∩ setOf t2) − eqs h e = setOf t1 ∩ setOf t2
      proof
        from ex have xOk: x : eqs h e by (simp add: eqs-def)
        have t1Ok: (setOf t1) Int (eqs h e) = {}
        proof (rule disjCond)
          fix y :: 'a
          assume y-in-t1: y : setOf t1
          assume y-in-eq: y : eqs h e
          from y-in-t1 s ex have o1: h y < h e by auto
          from y-in-eq have o2: ~ (h y < h e) by (simp add: eqs-def)
          from o1 o2 show False by contradiction
        qed
        from xOk t1Ok t2Ok show ?thesis by auto
      qed
    qed
  qed
qed
from o1 o2 show ?thesis by simp
lemma remove-sort: sortedTree h t -->
   sortedTree h (remove h e t) (is ?P t)
proof (induct t)
show ?P Tip by auto
fix t1 :: 'a Tree assume h1: ?P t1
fix t2 :: 'a Tree assume h2: ?P t2
fix x :: 'a
show ?P (T t1 x t2)
proof
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
We summarize the specification of remove as follows.

**Corollary** remove-spec: sortedTree h t --->
  sortedTree h (remove h e t) &
  setOf (remove h e t) = setOf t - eqs h e
by (simp add: remove-sort remove-set)

definition test = tlookup id 4 (remove id 3 (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)
  have 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 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 isome show tlookup fst i t = Some (i, a) by simp
qed

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

lemma assumes v: valid-tmap t
  shows mapset-none: (mapOf t i = None) = (ALL a. (i, a) ~: setOf t)
proof
  show (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) --> (ALL x: setOf t. fst x ~ = i)
    by (insert tlookup-none [of fst t i], assumption)
  from v lnone tlookup-none-inst have ALL 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

-- j==

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

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**  
\[ \text{mupdate} :: \text{index} \Rightarrow 'a \Rightarrow 'a \text{tarray} \Rightarrow 'a \text{tarray} \]
\[ \text{mupdate} \ i \ a \ t \equiv \text{binsert} \ \text{fst} \ (i,a) \ t \]

**lemma assumes**  
\[ v : \text{valid-tmap} \ t \]
\[ \text{shows} \ \text{mupdate-map}: \text{mapOf} \ (\text{mupdate} \ i \ a \ t) = (\text{mapOf} \ t)(i \mid-> a) \]

**proof**  
\[ \text{fix } i \]
\[ \text{let } \ ?t = \text{binsert} \ \text{fst} \ (i,a) \ t \]
\[ \text{have } \text{upres: } \text{mupdate} \ i \ a \ t = ?t \text{ by (simp add: mupdate-def)} \]
\[ \text{from } v \ \text{binsert-set} \]
\[ \text{have } \text{setSpec: } \text{setOf} \ ?t = \text{setOf} \ t - \text{eqs} \ \text{fst} \ (i,a) \ \text{Un} \ \{(i,a)\} \text{ by fastforce} \]
\[ \text{from } v \ \text{binsert-sorted} \ \text{have } \text{vr: } \text{valid-tmap} \ ?t \text{ by fastforce} \]
\[ \text{show } \text{mapOf} \ (\text{mupdate} \ i \ a \ t) \ i2 = ((\text{mapOf} \ t)(i \mid-> a)) \ i2 \]
\[ \text{proof} \ (\text{cases } i = i2) \]
\[ \text{case } True \ \text{note } i2ei = \text{this} \]
\[ \text{from } i2ei \ \text{have } \text{rhs-res: } ((\text{mapOf} \ t)(i \mid-> a)) \ i2 = \text{Some } a \text{ by simp} \]
\[ \text{have } \text{lhs-res: } \text{mapOf} \ (\text{mupdate} \ i \ a \ t) \ i = \text{Some } a \]
\[ \text{proof} \]
\[ \text{have } \text{will-find: } \text{lookup} \ \text{fst} \ i ?t = \text{Some } (i,a) \]
\[ \text{proof} \]
\[ \text{from } \text{setSpec} \ \text{have } \text{kvin: } (i,a) : \text{setOf} \ ?t \text{ by simp} \]
\[ \text{have } \text{binsert-sorted-inst: } \text{sortedTree} \ \text{fst} \ t \ 
\text{sortedTree} \ \text{fst} \ ?t \text{ by (insert binsert-sorted [of fst t (i,a)], assumption)} \]
\[ \text{from } v \ \text{binsert-sorted-inst} \ \text{have } \text{rs: } \text{sortedTree} \ \text{fst} \ ?t \text{ by simp} \]
\[ \text{have } \text{lookup-finds-inst: } \text{sortedTree} \ \text{fst} \ ?t \& \ (i,a) : \text{setOf} \ ?t \ 
\text{lookup} \ \text{fst} \ i ?t = \text{Some } (i,a) \]
\[ \text{by (insert lookup-finds [of fst ?t (i,a)], simp)} \]
\[ \text{from } \text{rs} \ \text{kvin} \ \text{lookup-finds-inst} \ \text{show } \text{thesis by simp} \]
\[ \text{qed} \]
\[ \text{from } \text{upres will-find show } \text{thesis by (simp add: mapOf-def)} \]
\[ \text{qed} \]
\[ \text{from } \text{lhs-res rhs-res i2ei show } \text{thesis by simp}\]
\[ \text{next case False note } i2nei = \text{this} \]
\[ \text{from } i2nei \ \text{have } \text{rhs-res: } ((\text{mapOf} \ t)(i \mid-> a)) \ i2 = \text{mapOf} \ t \ i2 \text{ by auto} \]
\[ \text{have } \text{lhs-res: } \text{mapOf} \ (\text{mupdate} \ i \ a \ t) \ i2 = \text{mapOf} \ t \ i2 \]
\[ \text{proof} \ (\text{cases mapOf } t \ i2) \]
\[ \text{case None from } \text{this have } \text{mapNone: } \text{mapOf} \ t \ i2 = \text{None by simp} \]
\[ \text{from } v \ \text{mapNone mapset-none have } \text{i2nin: } \text{ALL } a. \ (i2,a) \ 
\text{\sim: } \text{setOf} \ t \text{ by fastforce} \]
\[ \text{have } \text{noneIn: } \text{ALL } b. \ (i2,b) \ 
\text{\sim: } \text{setOf} \ ?t \]
\[ \text{proof} \]
\[ \text{fix } b \]
\[ \text{from } v \ \text{binsert-set} \]
\[ \text{have } \text{setOf} \ ?t = \text{setOf} \ t - \text{eqs} \ \text{fst} \ (i,a) \ \text{Un} \ \{(i,a)\} \]
\[ \text{by fastforce} \]
\[ \text{from } \text{this } i2nei \ \text{i2nin show } (i2,b) \ 
\text{\sim: } \text{setOf} \ ?t \text{ by fastforce} \]
qed
have mapset-none-inst:
valid-tmap ?tr -->> (mapOf ?tr i2 = None) = (ALL 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 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: ALL a. (i,a) ~: setOf ?tr
    proof
fix a
from remSet show (i,a) ~: setOf ?tr by (simp add: eqs-def)
qed
from v remove-sort have vr: valid-tmap ?tr by fastforce
have mapset-none-inst: valid-tmap ?tr ==>
(mapOf ?tr i = None) = (ALL a. (i,a) ~: setOf ?tr)
bysimp
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
\(\text{memb} \ e \ \text{Tip} = \text{False} \)
\[
\text{memb} \ e \ (\text{Nd} \ t1 \ x \ t2) = \\
(\text{if } e < x \ \text{then memb} \ e \ t1 \\
\text{else if } x < e \ \text{then memb} \ e \ t2 \\
\text{else True})
\]

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

15 Insertion operation

**primrec** \(\text{binsert} :: \text{nat} \Rightarrow \text{tree} \Rightarrow \text{tree}\)
— Insert a node into sorted tree.

**where**

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

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

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

**corollary** binsert-spec:
\(\text{sorted} \ t \Rightarrow\)
\(\text{sorted} \ (\text{binsert} \ x \ t) \land\)
\(\text{set-of} \ (\text{binsert} \ x \ t) = \text{set-of} \ t \cup \{x\}\)
by (simp add: binsert-sorted)

16 Remove operation

**primrec** \(\text{rm} :: \text{tree} \Rightarrow \text{nat}\)
— find the rightmost element in the tree

**where**

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

**primrec** \(\text{rem} :: \text{tree} \Rightarrow \text{tree}\)
— find the tree without the rightmost element

**where**

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

**primrec**
\(\text{remove} :: \text{nat} \Rightarrow \text{tree} \Rightarrow \text{tree}\)
— remove a node from sorted tree

**where**
remove x Tip = Tip
| remove x (Nd l y r) =
  (if x < y then Nd (remove x l) y r else
   if y < x then Nd l y (remove x r) else
   if l = Tip then r
   else Nd (rem l) (rm l) r)

lemma rm-in-set-of: t ~= Tip ==> rm t : set-of t
by (induct t) auto

lemma set-of-rem: t ~= Tip ==> set-of t = set-of(rem t) Un {rm t}
by (induct t) auto

lemma [simp]: [] t ~= Tip; sorted t [] ==> sorted(rem t)
by (induct t) (auto simp add:set-of-rem)

lemma sorted-rem: [] t ~= Tip; x ∈ set-of(rem t); sorted t [] ==> x < rm t
by (induct t) (auto simp add:set-of-rem simp:if-splits)

theorem set-of-remove [simp]: sorted t ==> set-of(remove x t) = 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
  apply(fastforce simp:set-of-rem)
done

theorem remove-sorted: sorted t ==> sorted(remove x t)
by (induct t) (auto intro: less-trans rm-in-set-of sorted-rem)

corollary remove-spec: summary specification of remove
sorted t ==> sorted (remove x t) &
sorted-rem (remove x t) = set-of t - \{x\}
by (simp add: remove-sorted)

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

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

lemma t ~= Tip ==> remrm t = (rem t, 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) (binset 4 (binset 3 Tip)))}

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

\texttt{end}