

BinarySearchTree

Larry Paulson

September 13, 2023

Contents

1	Isar-style Reasoning for Binary Tree Operations	1
2	Tree Definition	1
3	Tree Lookup	2
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	17
7	Map implementation and an abstraction function	17
8	Auxiliary Properties of our Implementation	18
8.1	Lemmas <i>mapset-none</i> and <i>mapset-some</i> establish a relation between the set and map abstraction of the tree	18
9	Empty Map	20
10	Map Update Operation	20
11	Map Remove Operation	21
12	Tactic-Style Reasoning for Binary Tree Operations	22
13	Definition of a sorted binary tree	22
14	Tree Membership	23
15	Insertion operation	23
16	Remove operation	23

1 Isar-style Reasoning for Binary Tree Operations

theory *BinaryTree* **imports** *Main* **begin**

We prove correctness of operations on binary search tree implementing a set.

This document is LGPL.

Author: Viktor Kuncak, MIT CSAIL, November 2003

2 Tree Definition

datatype *'a Tree* = *Tip* | *T 'a Tree 'a 'a Tree*

primrec

setOf :: *'a Tree* => *'a set*
— set abstraction of a tree

where

setOf Tip = {}
| *setOf (T t1 x t2)* = (*setOf t1*) *Un* (*setOf t2*) *Un* {*x*}

type-synonym

— we require index to have an irreflexive total order <
— apart from that, we do not rely on index being int
index = *int*

type-synonym — hash function type

'a hash = *'a* => *index*

definition *eqs* :: *'a hash* => *'a* => *'a set* **where**

— equivalence class of elements with the same hash code
eqs h x == {*y. h y* = *h x*}

primrec

sortedTree :: *'a hash* => *'a Tree* => *bool*
— check if a tree is sorted

where

sortedTree h Tip = *True*
| *sortedTree h (T t1 x t2)* =
 (*sortedTree h t1* &
 ($\forall l \in \text{setOf } t1. h\ l < h\ x$) &
 ($\forall r \in \text{setOf } t2. h\ x < h\ r$) &
 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

$lookup :: 'a \text{ hash} \Rightarrow index \Rightarrow 'a \text{ Tree} \Rightarrow 'a \text{ option}$

where

$lookup\ h\ k\ Tip = None$
 $| lookup\ h\ k\ (T\ t1\ x\ t2) =$
 $\quad (if\ k < h\ x\ then\ lookup\ h\ k\ t1$
 $\quad \quad else\ if\ h\ x < k\ then\ lookup\ h\ k\ t2$
 $\quad \quad else\ Some\ x)$

lemma *lookup-none*:

$sortedTree\ h\ t \ \&\ (lookup\ h\ k\ t = None) \dashrightarrow (\forall x \in setOf\ t. h\ x \sim k)$

by (*induct t, auto*)

lemma *lookup-some*:

$sortedTree\ h\ t \ \&\ (lookup\ h\ k\ t = Some\ x) \dashrightarrow x : setOf\ t \ \&\ h\ x = k$

apply (*induct t*)

— Just auto will do it, but very slowly

apply (*simp*)

apply (*clarify, auto*)

apply (*simp-all split: if-split-asm*)

done

definition *sorted-distinct-pred* :: $'a \text{ hash} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \text{ Tree} \Rightarrow bool$ **where**

— No two elements have the same hash code

$sorted-distinct-pred\ h\ a\ b\ t == sortedTree\ h\ t \ \&$
 $\quad a : setOf\ t \ \&\ b : setOf\ t \ \&\ h\ a = h\ b \dashrightarrow$
 $\quad a = b$

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

— for case analysis on three cases

lemma *cases3*: $[[\ C1 \implies G; C2 \implies G; C3 \implies G;$
 $\quad C1 \mid C2 \mid C3 \]] \implies 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 $t1 :: 'a \text{ Tree}$ **assume** $h1: ?P\ t1$

fix $t2 :: 'a \text{ Tree}$ **assume** $h2: ?P\ t2$

fix $x :: 'a$

show $?P\ (T\ t1\ x\ t2)$

proof (*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:  $h\ a = h\ b$ 
from s have s1: sortedTree h t1 by auto
from s have s2: 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  $a : \text{setOf } t1 \mid a = x \mid a : \text{setOf } t2$  by auto
moreover { assume adef1:  $a : \text{setOf } t1$ 
  have ?thesis
  proof —
  — three cases for b
  from bdef have  $b : \text{setOf } t1 \mid b = x \mid b : \text{setOf } t2$  by auto
  moreover { assume bdef1:  $b : \text{setOf } t1$ 
    from s1 adef1 bdef1 hahb h1 have ?thesis by simp }
  moreover { assume bdef1:  $b = x$ 
    from adef1 bdef1 s have  $h\ a < h\ b$  by auto
    from this hahb have ?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 hahb have ?thesis by simp } — case impossible
  ultimately show ?thesis by blast
  qed
}
moreover { assume adef1:  $a = x$ 
  have ?thesis
  proof —
  — three cases for b
  from bdef have  $b : \text{setOf } t1 \mid b = x \mid b : \text{setOf } t2$  by auto
  moreover { assume bdef1:  $b : \text{setOf } t1$ 
    from this s have  $h\ b < h\ x$  by auto
    from this adef1 have  $h\ b < h\ a$  by auto
    from hahb this have ?thesis by simp } — case impossible
  moreover { assume bdef1:  $b = x$ 
    from adef1 bdef1 have ?thesis by simp }
  moreover { assume bdef1:  $b : \text{setOf } t2$ 
    from this s have  $h\ x < h\ b$  by auto
    from this adef1 have  $h\ a < h\ b$  by simp
    from hahb this have ?thesis by simp } — case impossible
  ultimately show ?thesis by blast
  qed
}
moreover { assume adef1:  $a : \text{setOf } t2$ 
  have ?thesis
  proof —
  — three cases for b
  from bdef have  $b : \text{setOf } t1 \mid b = x \mid b : \text{setOf } t2$  by auto

```

```

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

```

lemma *tlookup-finds*: — if a node is in the tree, lookup finds it

```

sortedTree h t & y:setOf t -->
  tlookup h (h y) t = Some y
proof safe
  assume s: sortedTree h t
  assume yint: y : setOf t
  show tlookup h (h y) t = Some y
  proof (cases tlookup h (h y) t)
  case None note res = this
    from s res have sortedTree h t & (tlookup h (h y) t = None) by simp
    from this have o1:  $\forall x \in \text{setOf } t. h x \sim h y$  by (simp add: tlookup-none)
    from o1 yint have h y  $\sim$  h y by fastforce
    from this show ?thesis by simp
  next case (Some z) note res = this
    have 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

```

```

| Some z => (x=z))

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 tlookup-none have o1:  $\forall y \in \text{setOf } t. h y \sim h x$  by fastforce
  have notIn:  $x \sim \text{setOf } t$ 
  proof
    assume h: x : 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 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)
    from res zin xez show ?thesis by simp
  next case False note xnez = this
    from tSome xnez have res:  $\sim \text{memb h x t}$  by (simp add: memb-def)
    have x  $\sim \text{setOf } t$ 
    proof
      assume xin: x : 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
      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

```

$$(T\ t1\ x\ (binsert\ h\ e\ t2)) \\ else\ (T\ t1\ e\ t2)))$$

A technique for proving disjointness of sets.

lemma *disjCond*: $[[\text{!! } x. [x:A; x:B] ==> False]]$ $==> A \text{ Int } B = \{\}$
by *fastforce*

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

lemma *binsert-set*: *sortedTree* *h* *t* $-->$
 $setOf\ (binsert\ h\ e\ t) = (setOf\ t) - (eqs\ h\ e)\ Un\ \{e\}$
 (**is** *?P* *t*)
proof (*induct* *t*)
 — base case
show *?P* *Tip* **by** (*simp* *add*: *eqs-def*)
 — induction step
fix *t1* :: '*a* *Tree* **assume** *h1*: *?P* *t1*
fix *t2* :: '*a* *Tree* **assume** *h2*: *?P* *t2*
fix *x* :: '*a*
show *?P* (*T* *t1* *x* *t2*)
proof
assume *s*: *sortedTree* *h* (*T* *t1* *x* *t2*)
from *s* **have** *s1*: *sortedTree* *h* *t1* **by** (*rule* *sortLemmaL*)
from *s1* **and** *h1* **have** *c1*: $setOf\ (binsert\ h\ e\ t1) = setOf\ t1 - eqs\ h\ e\ Un\ \{e\}$
by *simp*
from *s* **have** *s2*: *sortedTree* *h* *t2* **by** (*rule* *sortLemmaR*)
from *s2* **and** *h2* **have** *c2*: $setOf\ (binsert\ h\ e\ t2) = setOf\ t2 - eqs\ h\ e\ Un\ \{e\}$
by *simp*
show $setOf\ (binsert\ h\ e\ (T\ t1\ x\ t2)) =$
 $setOf\ (T\ t1\ x\ t2) - eqs\ h\ e\ Un\ \{e\}$
proof (*cases* *h* *e* < *h* *x*)
case *True* **note** *eLess* = *this*
from *eLess* **have** *res*: $binsert\ h\ e\ (T\ t1\ x\ t2) = (T\ (binsert\ h\ e\ t1)\ x\ t2)$ **by**
simp
show $setOf\ (binsert\ h\ e\ (T\ t1\ x\ t2)) =$
 $setOf\ (T\ t1\ x\ t2) - eqs\ h\ e\ Un\ \{e\}$
proof (*simp* *add*: *res* *eLess* *c1*)
show $insert\ x\ (insert\ e\ (setOf\ t1 - eqs\ h\ e\ Un\ setOf\ t2)) =$
 $insert\ e\ (insert\ x\ (setOf\ t1\ Un\ setOf\ t2) - eqs\ h\ e)$
proof —
have *eqsLessX*: $\forall el \in eqs\ h\ e. h\ el < h\ x$ **by** (*simp* *add*: *eqs-def* *eLess*)
from *this* **have** *eqsDisjX*: $\forall el \in eqs\ h\ e. h\ el \sim = h\ x$ **by** *fastforce*
from *s* **have** *xLessT2*: $\forall r \in setOf\ t2. h\ x < h\ r$ **by** *auto*
have *eqsLessT2*: $\forall el \in eqs\ h\ e. \forall r \in setOf\ t2. h\ el < h\ r$
proof *safe*
fix *el* **assume** *hel*: *el* : *eqs* *h* *e*
from *hel* *eqs-def* **have** *o1*: *h* *el* = *h* *e* **by** *fastforce*
fix *r* **assume** *hr*: *r* : *setOf* *t2*

```

      from  $xLessT2$   $hr$   $o1$   $eLess$  show  $h\ el < h\ r$  by auto
    qed
    from  $eqsLessT2$  have  $eqsDisjT2$ :  $\forall el \in eqs\ h\ e. \forall r \in setOf\ t2. h\ el \sim =$ 
   $h\ r$ 
    by fastforce
    from  $eqsDisjX$   $eqsDisjT2$  show ?thesis by fastforce
  qed
  qed
  next case False note  $eNotLess = this$ 
  show  $setOf\ (bininsert\ h\ e\ (T\ t1\ x\ t2)) = setOf\ (T\ t1\ x\ t2) - eqs\ h\ e\ Un\ \{e\}$ 
  proof (cases  $h\ x < h\ e$ )
    case True note  $xLess = this$ 
    from  $xLess$  have  $res$ :  $bininsert\ h\ e\ (T\ t1\ x\ t2) = (T\ t1\ x\ (bininsert\ h\ e\ t2))$  by
  simp
    show  $setOf\ (bininsert\ h\ e\ (T\ t1\ x\ t2)) =$ 
       $setOf\ (T\ t1\ x\ t2) - eqs\ h\ e\ Un\ \{e\}$ 
    proof (simp add:  $res\ xLess\ eNotLess\ c2$ )
      show  $insert\ x\ (insert\ e\ (setOf\ t1\ Un\ (setOf\ t2 - eqs\ h\ e))) =$ 
         $insert\ e\ (insert\ x\ (setOf\ t1\ Un\ setOf\ t2) - eqs\ h\ e)$ 
      proof -
        have  $XLessEqs$ :  $\forall el \in eqs\ h\ e. h\ x < h\ el$  by (simp add:  $eqs-def\ xLess$ )
        from this have  $eqsDisjX$ :  $\forall el \in eqs\ h\ e. h\ el \sim = h\ x$  by auto
        from  $s$  have  $t1LessX$ :  $\forall l \in setOf\ t1. h\ l < h\ x$  by auto
        have  $T1lessEqs$ :  $\forall el \in eqs\ h\ e. \forall l \in setOf\ t1. h\ l < h\ el$ 
        proof safe
          fix  $el$  assume  $hel$ :  $el : eqs\ h\ e$ 
          fix  $l$  assume  $hl$ :  $l : setOf\ t1$ 
          from  $hel$   $eqs-def$  have  $o1$ :  $h\ el = h\ e$  by fastforce
          from  $t1LessX$   $hl$   $o1$   $xLess$  show  $h\ l < h\ el$  by auto
        qed
        from  $T1lessEqs$  have  $T1disjEqs$ :  $\forall el \in eqs\ h\ e. \forall l \in setOf\ t1. h\ el \sim =$ 
   $h\ l$ 
          by fastforce
          from  $eqsDisjX$   $T1lessEqs$  show ?thesis by auto
        qed
      qed
    next case False note  $xNotLess = this$ 
    from  $xNotLess$   $eNotLess$  have  $xqe$ :  $h\ x = h\ e$  by simp
    from  $xqe$  have  $res$ :  $bininsert\ h\ e\ (T\ t1\ x\ t2) = (T\ t1\ e\ t2)$  by simp
    show  $setOf\ (bininsert\ h\ e\ (T\ t1\ x\ t2)) =$ 
       $setOf\ (T\ t1\ x\ t2) - eqs\ h\ e\ Un\ \{e\}$ 
    proof (simp add:  $res\ eNotLess\ xqe$ )
      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:  $eqs-def\ xqe$ )

```



```

moreover have (setOf t1) Int (eqs h e) = {}
proof (rule disjCond)
  fix w
  assume whSet: w : setOf t1
  assume whEq: w : eqs h e
  from whSet s have o1: h w < h x by simp
  from whEq eqs-def have o2: h w = h e by fastforce
  from o2 xege have o3: ~ h w < h x by simp
  from o1 o3 show False by contradiction
qed
moreover have (setOf t2) Int (eqs h e) = {}
proof (rule disjCond)
  fix w
  assume whSet: w : setOf t2
  assume whEq: w : eqs h e
  from whSet s have o1: h x < h w by simp
  from whEq eqs-def have o2: h w = h e by fastforce
  from o2 xege have o3: ~ h x < h w by simp
  from o1 o3 show False by contradiction
qed
ultimately show ?thesis by auto
qed
from this show ?thesis by simp
qed
qed
qed
qed
qed
qed

```

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

lemma *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*: $\text{sortedTree } h \ t \longrightarrow$
 $\text{sortedTree } h \ (\text{binsert } h \ x \ t) \ \&$
 $\text{setOf } (\text{binsert } h \ e \ t) = (\text{setOf } t) - (\text{eqs } h \ e) \cup \{e\}$
by (simp add: binsert-set binsert-sorted)

5 Removing an element from a tree

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

primrec
 $\text{rm} :: 'a \ \text{hash} \Rightarrow 'a \ \text{Tree} \Rightarrow 'a$
 — rightmost element of a tree

where

$rm\ h\ (T\ t1\ x\ t2) =$
 $(if\ t2 = Tip\ then\ x\ else\ rm\ h\ t2)$

primrec

$wrm :: 'a\ hash \Rightarrow 'a\ Tree \Rightarrow 'a\ Tree$
 — 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

$wrmrm :: 'a\ hash \Rightarrow 'a\ Tree \Rightarrow 'a\ Tree * 'a$
 — 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

$remove :: 'a\ hash \Rightarrow 'a \Rightarrow 'a\ Tree \Rightarrow 'a\ Tree$
 — 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 $wrmrm\text{-}decomp: t \sim = Tip \dashrightarrow wrmrm\ h\ t = (wrm\ h\ t, rm\ h\ t)$

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

apply $simp\text{-}all$

done

lemma $rm\text{-}set: t \sim = Tip \ \&\ sortedTree\ h\ t \dashrightarrow rm\ h\ t : setOf\ t$

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

apply $simp\text{-}all$

done

lemma $wrm\text{-}set: t \sim = Tip \ \&\ sortedTree\ h\ t \dashrightarrow$

$setOf\ (wrm\ h\ t) = setOf\ t - \{rm\ h\ t\}\ (is\ ?P\ t)$

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 ?P (T t1 x t2)
proof (rule impI, erule conjE)
  assume s: sortedTree h (T t1 x t2)
  show setOf (wrn 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 (wrn h t2) by simp
    from s have s2: sortedTree h t2 by simp
    from h2 t2nTip s2
    have o1: setOf (wrn 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:  $\forall l \in \text{setOf } t1. h\ l < h\ (\text{rm } h\ t2)$ 
      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: $t \sim = \text{Tip} \ \& \ \text{sortedTree } h\ t \longrightarrow \text{setOf } (\text{wrn } h\ t) \leq \text{setOf } t$
by (auto simp add: wrm-set)

lemma wrm-sort: $t \sim = \text{Tip} \ \& \ \text{sortedTree } h\ t \longrightarrow \text{sortedTree } h\ (\text{wrn } h\ t)$ (**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
    assume s: sortedTree h (T t1 x t2)

```

```

show sortedTree h (wrn h (T t1 x t2))
proof (cases t2 = Tip)
case True note t2tip = this
  from t2tip have res: wrn 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: wrn h (T t1 x t2) = T t1 x (wrn h t2) by simp
  from s have s1: sortedTree h t1 by simp
  from s have s2: sortedTree h t2 by simp
  from s2 h2 t2nTip have o1: sortedTree h (wrn h t2) by simp
  from s2 t2nTip wrm-set1 have o2: setOf (wrn h t2) <= setOf t2 by auto
  from s o2 have o3:  $\forall r \in \text{setOf } (\text{wrn } h \ t2). h \ x < h \ r$  by auto
  from s1 o1 o3 res s show sortedTree h (wrn h (T t1 x t2)) by simp
qed
qed
qed

lemma wrm-less-rm:
  t ~ = Tip & sortedTree h t -->
    ( $\forall l \in \text{setOf } (\text{wrn } h \ t). h \ l < h \ (\text{rm } h \ t)$ ) (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:  $l : \text{setOf } (\text{wrn } h \ (T \ t1 \ x \ t2))$ 
  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  $h \ l < h \ (\text{rm } h \ (T \ t1 \ x \ t2))$ 
  proof (cases t2 = Tip)
  case True note t2tip = this
    from t2tip have rm-res:  $\text{rm } h \ (T \ t1 \ x \ t2) = x$  by simp
    from t2tip have wrm-res:  $\text{wrn } h \ (T \ t1 \ x \ t2) = t1$  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 t2nTip = this
    from t2nTip have rm-res:  $\text{rm } h \ (T \ t1 \ x \ t2) = \text{rm } h \ t2$  by simp
    from t2nTip have wrm-res:  $\text{wrn } h \ (T \ t1 \ x \ t2) = T \ t1 \ x \ (\text{wrn } h \ t2)$  by simp
    from ldef wrm-res
    have l-scope:  $l : \{x\} \cup \text{setOf } t1 \cup \text{setOf } (\text{wrn } h \ t2)$  by simp
    have hLess:  $h \ l < h \ (\text{rm } h \ t2)$ 
    proof (cases l = x)
    case True note lx = this
      from s t2nTip 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 lnx = this

```

```

show ?thesis
proof (cases l : setOf t1)
case True note l-in-t1 = this
  from s t2nTip rm-set s2 have o1: h x < h (rm h t2) by auto
  from l-in-t1 s have o2: h l < h x by auto
  from o1 o2 show ?thesis by simp
next case False note l-notin-t1 = this
  from l-scope lnx l-notin-t1
  have l-in-res: l : setOf (wrn 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
qed

lemma remove-set: sortedTree h t -->
  setOf (remove h e t) = setOf t - eqs h e (is ?P t)
proof (induct t)
show ?P Tip by auto
fix t1 :: 'a Tree assume h1: ?P t1
fix t2 :: 'a Tree assume h2: ?P t2
fix x :: 'a
show ?P (T t1 x t2)
proof
  assume s: sortedTree h (T t1 x t2)
  show setOf (remove h e (T t1 x t2)) = setOf (T t1 x t2) - eqs h e
  proof (cases h e < h x)
  case True note elx = this
    from elx have res: remove h e (T t1 x t2) = T (remove h e t1) x t2
    by simp
    from s have s1: sortedTree h t1 by simp
    from s1 h1 have o1: setOf (remove h e t1) = setOf t1 - eqs h e by simp
    show ?thesis
  proof (simp add: o1 elx)
    show insert x (setOf t1 - eqs h e Un setOf t2) =
      insert x (setOf t1 Un setOf t2) - eqs h e
    proof -
      have xOk: x ~: eqs h e
      proof
        assume h: x : eqs h e
        from h have o1: ~ (h e < h x) by (simp add: eqs-def)
        from elx o1 show False by contradiction
      qed
      have t2Ok: (setOf t2) Int (eqs h e) = {}
      proof (rule disjCond)
        fix y :: 'a
        assume y-in-t2: y : setOf t2

```

```

    assume y-in-eq: y : eqs h e
    from y-in-t2 s have xly: h x < h y by auto
    from y-in-eq have eey: h y = h e by (simp add: eqs-def)
    from xly eey have nelx: ~ (h e < h x) by simp
    from nelx elx show False by contradiction
  qed
  from xOk t2Ok show ?thesis by auto
  qed
  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 nyle: ~ (h x < h e) by simp
      from nyle xle show False by contradiction
    qed
    from xOk t1Ok show ?thesis by auto
  qed
  next case False note nyle = this
  from nelx nyle have ex: h e = h x by simp
  have t2Ok: (setOf t2) Int (eqs h e) = {}
  proof (rule disjCond)
    fix y :: 'a
    assume y-in-t2: y : setOf t2
    assume y-in-eq: y : eqs h e
    from y-in-t2 s have xly: h x < h y by auto

```

```

    from y-in-eq have eey:  $h\ y = h\ e$  by (simp add: eqs-def)
    from y-in-eq ex eey have nxly:  $\sim (h\ x < h\ y)$  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:  $\text{remove } h\ e\ (T\ t1\ x\ t2) = t2$  by simp
  show ?thesis
  proof (simp add: res t1tip ex)
    show  $\text{setOf } t2 = \text{insert } x\ (\text{setOf } t2) - \text{eqs } h\ e$ 
    proof -
      from ex have x-in-eqs:  $x : \text{eqs } h\ e$  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:  $\text{remove } h\ e\ (T\ t1\ x\ t2) =$ 
     $T\ (\text{wrm } h\ t1)\ (\text{rm } h\ t1)\ t2$ 
  by (simp add: Let-def wrmm-decomp)
  from res show ?thesis
  proof simp
    from s have s1: sortedTree h t1 by simp
    show  $\text{insert } (\text{rm } h\ t1)\ (\text{setOf } (\text{wrm } h\ t1)\ Un\ \text{setOf } t2) =$ 
       $\text{insert } x\ (\text{setOf } t1\ Un\ \text{setOf } t2) - \text{eqs } h\ e$ 
    proof (simp add: t1nTip s1 rm-set wrm-set)
      show  $\text{insert } (\text{rm } h\ t1)\ (\text{setOf } t1 - \{\text{rm } h\ t1\}\ Un\ \text{setOf } t2) =$ 
         $\text{insert } x\ (\text{setOf } t1\ Un\ \text{setOf } t2) - \text{eqs } h\ e$ 
      proof -
        from t1nTip s1 rm-set
        have o1:  $\text{insert } (\text{rm } h\ t1)\ (\text{setOf } t1 - \{\text{rm } h\ t1\}\ Un\ \text{setOf } t2) =$ 
           $\text{setOf } t1\ Un\ \text{setOf } t2$  by auto
        have o2:  $\text{insert } x\ (\text{setOf } t1\ Un\ \text{setOf } t2) - \text{eqs } h\ e =$ 
           $\text{setOf } t1\ Un\ \text{setOf } t2$ 
        proof -
          from ex have xOk:  $x : \text{eqs } h\ e$  by (simp add: eqs-def)
          have t1Ok:  $(\text{setOf } t1)\ \text{Int}\ (\text{eqs } h\ e) = \{\}$ 
          proof (rule disjCond)
            fix y :: 'a
            assume y-in-t1:  $y : \text{setOf } t1$ 
            assume y-in-eq:  $y : \text{eqs } h\ e$ 
            from y-in-t1 s ex have o1:  $h\ y < h\ e$  by auto
            from y-in-eq have o2:  $\sim (h\ y < h\ e)$  by (simp add: eqs-def)
            from o1 o2 show False by contradiction
          qed
          from xOk t1Ok t2Ok show ?thesis by auto
        qed
      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 =  $\forall l \in \text{setOf } (\text{remove h e t1}). h\ l < h\ x$ 
  let ?C2 =  $\forall r \in \text{setOf } t2. h\ x < h\ r$ 
  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 =  $\forall l \in \text{setOf } t1. h\ l < h\ x$ 

```



```

    let ?C2 =  $\forall r \in \text{setOf } (\text{remove } h \ e \ t2). \ h \ x < h \ r$ 
    have o2: ?C2
    proof -
      from s2 have setOf (remove h e t2) = setOf t2 - eqs h e by (simp add:
remove-set)
      from s this show ?thesis by auto
    qed
    from o2 s show ?C1 & ?C2 by auto
  qed
next case False note nxle = this
from nelx nxle have ex: h e = h x by simp
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 by (simp add: res t1tip ex s2)
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
  let ?C1 = sortedTree h (wrm h t1)
  let ?C2 =  $\forall l \in \text{setOf } (\text{wrm } h \ t1). \ h \ l < h \ (\text{rm } h \ t1)$ 
  let ?C3 =  $\forall r \in \text{setOf } t2. \ h \ (\text{rm } h \ t1) < h \ r$ 
  let ?C4 = sortedTree h t2
  from s1 t1nTip have o1: ?C1 by (simp add: wrm-sort)
  from s1 t1nTip have o2: ?C2 by (simp add: wrm-less-rm)
  have o3: ?C3
  proof
    fix r :: 'a
    assume rt2: r : setOf t2
    from s rm-set s1 t1nTip have o1: h (rm h t1) < h x by auto
    from rt2 s have o2: h x < h r by auto
    from o1 o2 show h (rm h t1) < h r by simp
  qed
  from o1 o2 o3 s2 show ?C1 & ?C2 & ?C3 & ?C4 by simp
qed
qed
qed
qed
qed
qed

```

We summarize the specification of remove as follows.

corollary *remove-spec*: $\text{sortedTree } h \ t \dashrightarrow$
 $\text{sortedTree } h \ (\text{remove } h \ e \ t) \ \&$
 $\text{setOf } (\text{remove } h \ e \ t) = \text{setOf } t - \text{eqs } h \ e$

```

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
 — ==>
 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:  $\forall a. (i,a) \notin \text{setOf } t$ 
show  $\text{mapOf } t \ i = \text{None}$ 
proof (cases  $\text{mapOf } t \ i$ )
case None then show ?thesis .
next case (Some a) note mapsome = this
  from v mapsome have o1:  $\text{tlookup fst } i \ t = \text{Some } (i,a)$  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:  $(\text{mapOf } t \ i = \text{Some } a) = ((i,a) : \text{setOf } t)$ 
proof
— ==>
  assume mapsome:  $\text{mapOf } t \ i = \text{Some } a$ 
  from v mapsome have o1:  $\text{tlookup fst } i \ t = \text{Some } (i,a)$  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:  $(i,a) : \text{setOf } t$ 
  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  $\text{mapOf } t \ i = \text{Some } a$  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:  $\text{mapOf } \text{Tip } k = \text{None}$ 
by (simp add: mapOf-def)

```

10 Map Update Operation

definition $mupdate :: index \Rightarrow 'a \Rightarrow 'a \text{ tarray} \Rightarrow 'a \text{ tarray}$ **where**
 $mupdate\ i\ a\ t == \text{binsert}\ \text{fst}\ (i,a)\ t$

lemma assumes $v: \text{valid-tmap}\ t$

shows $mupdate\text{-map}: \text{mapOf}\ (mupdate\ i\ a\ t) = (\text{mapOf}\ t)(i \mid\!-\!>\ a)$

proof

fix $i2$

let $?tr = \text{binsert}\ \text{fst}\ (i,a)\ t$

have $\text{upres}: mupdate\ i\ a\ t = ?tr$ **by** $(\text{simp}\ \text{add}: mupdate\text{-def})$

from v **binsert-set**

have $\text{setSpec}: \text{setOf}\ ?tr = \text{setOf}\ t - \text{eqs}\ \text{fst}\ (i,a)\ \text{Un}\ \{(i,a)\}$ **by** fastforce

from v **binsert-sorted** **have** $vr: \text{valid-tmap}\ ?tr$ **by** fastforce

show $\text{mapOf}\ (mupdate\ i\ a\ t)\ i2 = (\text{mapOf}\ t)(i \mid\!-\!>\ a)\ i2$

proof $(\text{cases}\ i = i2)$

case True **note** $i2ei = \text{this}$

from $i2ei$ **have** $\text{rhs-res}: (\text{mapOf}\ t)(i \mid\!-\!>\ a)\ i2 = \text{Some}\ a$ **by** simp

have $\text{lhs-res}: \text{mapOf}\ (mupdate\ i\ a\ t)\ i = \text{Some}\ a$

proof $-$

have $\text{will-find}: \text{tlookup}\ \text{fst}\ i\ ?tr = \text{Some}\ (i,a)$

proof $-$

from setSpec **have** $kvin: (i,a) : \text{setOf}\ ?tr$ **by** simp

have $\text{binsert-sorted-inst}: \text{sortedTree}\ \text{fst}\ t \dashv\!\!\!\rightarrow$

$\text{sortedTree}\ \text{fst}\ ?tr$

by $(\text{insert}\ \text{binsert-sorted}\ [\text{of}\ \text{fst}\ t\ (i,a)],\ \text{assumption})$

from v **binsert-sorted-inst** **have** $rs: \text{sortedTree}\ \text{fst}\ ?tr$ **by** simp

have $\text{tlookup-finds-inst}: \text{sortedTree}\ \text{fst}\ ?tr \ \&\ (i,a) : \text{setOf}\ ?tr \dashv\!\!\!\rightarrow$

$\text{tlookup}\ \text{fst}\ i\ ?tr = \text{Some}\ (i,a)$

by $(\text{insert}\ \text{tlookup-finds}\ [\text{of}\ \text{fst}\ ?tr\ (i,a)],\ \text{simp})$

from $rs\ kvin\ \text{tlookup-finds-inst}$ **show** $?thesis$ **by** simp

qed

from $\text{upres}\ \text{will-find}$ **show** $?thesis$ **by** $(\text{simp}\ \text{add}: \text{mapOf-def})$

qed

from $\text{lhs-res}\ \text{rhs-res}\ i2ei$ **show** $?thesis$ **by** simp

next case False **note** $i2nei = \text{this}$

from $i2nei$ **have** $\text{rhs-res}: (\text{mapOf}\ t)(i \mid\!-\!>\ a)\ i2 = \text{mapOf}\ t\ i2$ **by** auto

have $\text{lhs-res}: \text{mapOf}\ (mupdate\ i\ a\ t)\ i2 = \text{mapOf}\ t\ i2$

proof $(\text{cases}\ \text{mapOf}\ t\ i2)$

case None **from** this **have** $\text{mapNone}: \text{mapOf}\ t\ i2 = \text{None}$ **by** simp

from v **mapNone** **mapset-none** **have** $i2nin: \forall a. (i2,a) \notin \text{setOf}\ t$ **by** fastforce

have $\text{noneIn}: \forall b. (i2,b) \notin \text{setOf}\ ?tr$

proof

fix b

from v **binsert-set**

have $\text{setOf}\ ?tr = \text{setOf}\ t - \text{eqs}\ \text{fst}\ (i,a)\ \text{Un}\ \{(i,a)\}$

by fastforce

from $\text{this}\ i2nei\ i2nin$ **show** $(i2,b) \sim: \text{setOf}\ ?tr$ **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 setSpec i2nei have (i2,z) : setOf ?tr by (simp add: eqs-def)
from this vr mapset-some have mapOf ?tr i2 = Some z by fastforce
from this upres mapSome show ?thesis by simp
qed
from lhs-res rhs-res show ?thesis by simp
qed
qed

```

```

lemma assumes v: valid-tmap t
  shows mupdate-valid: valid-tmap (mupdate i a t)
proof -
  let ?tr = bininsert fst (i,a) t
  have upres: mupdate i a t = ?tr by (simp add: mupdate-def)
  from v bininsert-sorted have vr: valid-tmap ?tr by fastforce
  from vr upres show ?thesis by simp
qed

```

11 Map Remove Operation

definition mremove :: index => 'a tarray => 'a tarray **where**
 mremove i t == remove fst (i, undefined) t

```

lemma assumes v: valid-tmap t
  shows mremove-valid: valid-tmap (mremove i t)
proof (simp add: mremove-def)
  from v remove-sort
  show sortedTree fst (remove fst (i, undefined) t) by fastforce
qed

```

```

lemma assumes v: valid-tmap t
  shows mremove-map: mapOf (mremove i t) i = None
proof (simp add: mremove-def)
  let ?tr = remove fst (i, undefined) t
  show mapOf ?tr i = None
proof -
  from v remove-spec
  have remSet: setOf ?tr = setOf t - eqs fst (i, undefined)
  by fastforce
  have noneIn: ∀ a. (i,a) ∉ setOf ?tr
proof
  fix a
  from remSet show (i,a) ∼: setOf ?tr by (simp add: eqs-def)

```

```

    qed
  from v remove-sort have vr: valid-tmap ?tr by fastforce
  have mapset-none-inst: valid-tmap ?tr ==>
    (mapOf ?tr i = None) = (∀ a. (i,a) ∉ setOf ?tr)
  by (insert mapset-none [of ?tr i], simp)
  from vr this have (mapOf ?tr i = None) = (∀ a. (i,a) ∉ setOf ?tr) by fastforce
  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* ∪ *set-of r* ∪ {*x*}

primrec *sorted* :: *tree* => *bool*

— Tree is sorted

where

sorted Tip = *True*
 | *sorted (Nd l y r)* =
 (*sorted l* & *sorted r* & (∀ *x* ∈ *set-of l*. *x* < *y*) & (∀ *z* ∈ *set-of r*. *y* < *z*))

14 Tree Membership

primrec

memb :: *nat* => *tree* => *bool*

where

memb e Tip = *False*
 | *memb e (Nd t1 x t2)* =
 (*if e* < *x* then *memb e t1*

else if $x < e$ then memb $e\ t2$
else True)

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

15 Insertion operation

primrec *binsert :: nat => tree => tree*

— Insert a node into sorted tree.

where

binsert x Tip = (Nd Tip x Tip)
| binsert x (Nd t1 y t2) = (if $x < y$ then
(Nd (binsert x t1) y t2)
else
(if $y < x$ then
(Nd t1 y (binsert x t2))
else (Nd t1 y t2)))

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

theorem *binsert-sorted: sorted t \dashrightarrow sorted (binsert x t)*

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

corollary *binsert-spec:*

sorted t ==>
sorted (binsert x t) &
set-of (binsert x t) = set-of t Un {x}
by (*simp add: binsert-sorted*)

16 Remove operation

primrec

rm :: tree => nat — find the rightmost element in the tree

where

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

primrec

rem :: tree => tree — find the tree without the rightmost element

where

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

primrec

remove :: nat => tree => tree — remove a node from sorted tree

where

remove x Tip = Tip
| remove x (Nd l y r) =
(if $x < y$ then Nd (remove x l) y r else


```

    if  $y < x$  then  $Nd\ l\ y\ (remove\ x\ r)$  else
    if  $l = Tip$  then  $r$ 
    else  $Nd\ (rem\ l)\ (rm\ l)\ r$ 

```

lemma *rm-in-set-of*: $t \sim = Tip \implies rm\ t : set-of\ t$
by (*induct t*) *auto*

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

lemma [*simp*]: $[\mid t \sim = Tip; sorted\ t \mid] \implies sorted\ (rem\ t)$
by (*induct t*) (*auto simp add:set-of-rem*)

lemma *sorted-rem*: $[\mid t \sim = Tip; x \in set-of\ (rem\ t); sorted\ t \mid] \implies x < rm\ t$
by (*induct t*) (*auto simp add:set-of-rem split:if-splits*)

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

theorem *remove-sorted*: $sorted\ t \implies sorted\ (remove\ x\ t)$
by (*induct t*) (*auto intro: less-trans rm-in-set-of sorted-rem*)

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

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

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

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

We can test this implementation by generating code.

definition *test* = *memb* 4 (*remove* (3::nat) (*binsert* 4 (*binsert* 3 *Tip*)))

```
export-code test  
  in SML module-name BinaryTree-TacticStyle-Code file ⟨BinaryTree-TacticStyle-Code.ML⟩  
end
```