

Abstract

We present the verification of the normalisation of a binary decision diagram (BDD). The normalisation follows the original algorithm presented by Bryant in 1986 and transforms an ordered BDD in a reduced, ordered and shared BDD. The verification is based on Hoare logics.

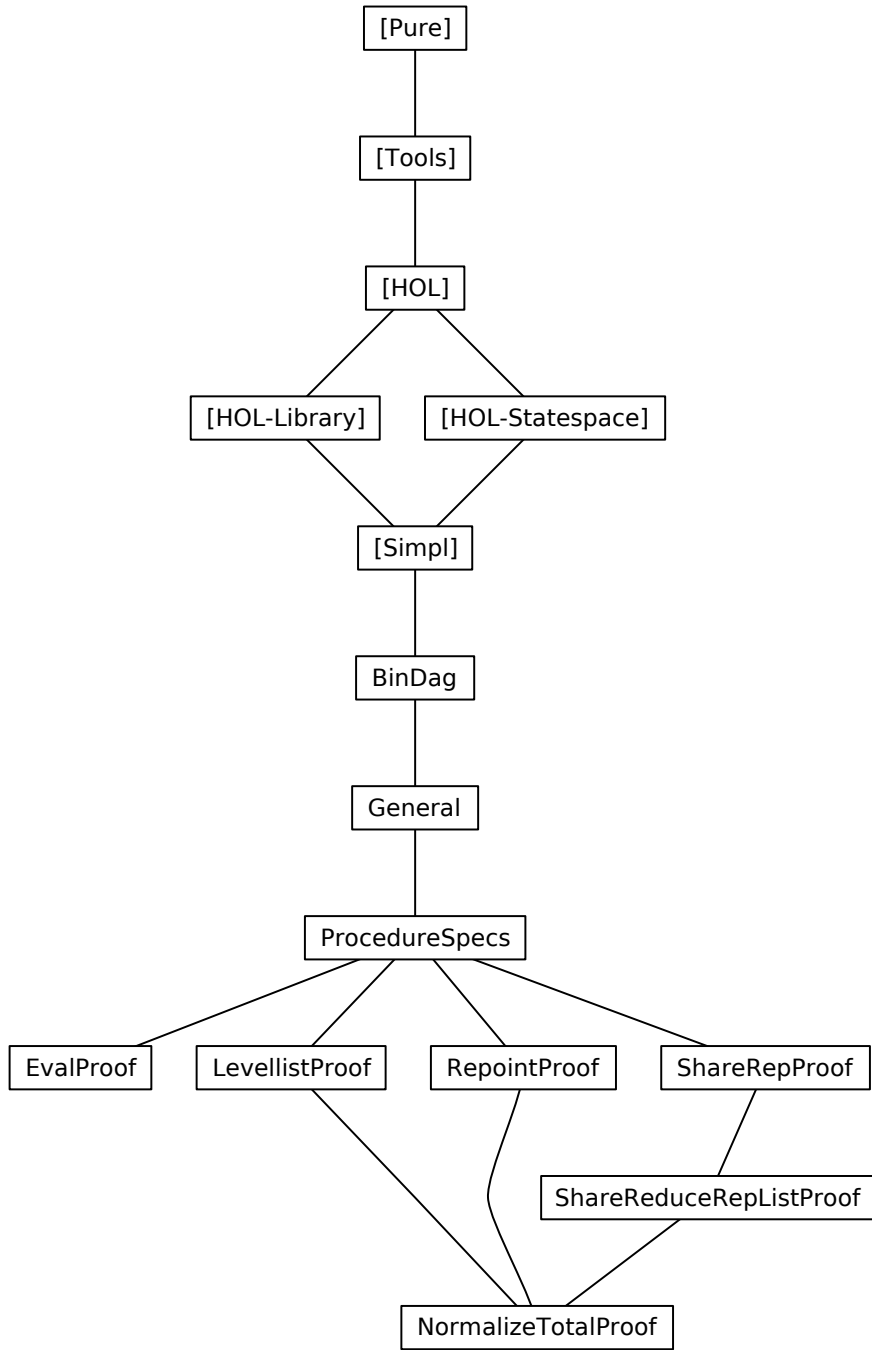
BDD-Normalisation

Veronika Ortner and Norbert Schirmer

May 26, 2024

Contents

1	Introduction	4
2	BDD Abstractions	4
3	General Lemmas on BDD Abstractions	12
4	Definitions of Procedures	50
5	Proof of Procedure Eval	53
6	Proof of Procedure Levellist	53
7	Proof of Procedure ShareRep	79
8	Proof of Procedure ShareReduceRepList	85
9	Proof of Procedure Reprint	128
10	Proof of Procedure Normalize	146



1 Introduction

In [1] we describe the partial correctness proofs for BDD normalisation. We extend this work to total correctness in these theories.

2 BDD Abstractions

```
theory BinDag
imports Simpl.Simpl-Heap
begin

datatype dag = Tip | Node dag ref dag

lemma [simp]: Node lt a rt  $\neq$  lt
  by (induct lt) auto

lemma [simp]: lt  $\neq$  Node lt a rt
  by (induct lt) auto

lemma [simp]: Node lt a rt  $\neq$  rt
  by (induct rt) auto

lemma [simp]: rt  $\neq$  Node lt a rt
  by (induct rt) auto

primrec set-of:: dag  $\Rightarrow$  ref set where
  set-of-Tip: set-of Tip = {}
  | set-of-Node: set-of (Node lt a rt) = {a}  $\cup$  set-of lt  $\cup$  set-of rt

primrec DAG:: dag  $\Rightarrow$  bool where
  DAG Tip = True
  | DAG (Node l a r) = (a  $\notin$  set-of l  $\wedge$  a  $\notin$  set-of r  $\wedge$  DAG l  $\wedge$  DAG r)

primrec subdag:: dag  $\Rightarrow$  dag  $\Rightarrow$  bool where
  subdag Tip t = False
  | subdag (Node l a r) t = (t=l  $\vee$  t=r  $\vee$  subdag l t  $\vee$  subdag r t)

lemma subdag-size: subdag t s  $\Longrightarrow$  size s < size t
  by (induct t) auto

lemma subdag-neq: subdag t s  $\Longrightarrow$  t $\neq$ s
by (induct t) (auto dest: subdag-size)

lemma subdag-trans [trans]: subdag t s  $\Longrightarrow$  subdag s r  $\Longrightarrow$  subdag t r
by (induct t) auto

lemma subdag-NodeD:
```

subdag t (Node lt a rt) \implies subdag t lt \wedge subdag t rt
by (*induct t*) *auto*

lemma *subdag-not-sym*: $\bigwedge t. \llbracket \text{subdag } s \ t; \text{subdag } t \ s \rrbracket \implies P$
by (*induct s*) (*auto dest: subdag-NodeD*)

instantiation *dag :: order*
begin

definition
less-dag-def: $s < (t::dag) \longleftrightarrow \text{subdag } t \ s$

definition
le-dag-def: $s \leq (t::dag) \longleftrightarrow s=t \vee s < t$

lemma *le-dag-refl*: $(x::dag) \leq x$
by (*simp add: le-dag-def*)

lemma *le-dag-trans*:
fixes $x::dag$ **and** y **and** z
assumes $x-y: x \leq y$ **and** $y-z: y \leq z$
shows $x \leq z$
proof (*cases x=y*)
case *True* **with** $y-z$ **show** *?thesis* **by** *simp*
next
case *False*
note $x \neq y = \text{this}$
with $x-y$ **have** $x \text{-less-} y: x < y$ **by** (*simp add: le-dag-def*)
show *?thesis*
proof (*cases y=z*)
case *True*
with $x-y$ **show** *?thesis* **by** *simp*
next
case *False*
with $y-z$ **have** $y < z$ **by** (*simp add: le-dag-def*)
with $x \text{-less-} y$ **have** $x < z$
by (*auto simp add: less-dag-def intro: subdag-trans*)
thus *?thesis*
by (*simp add: le-dag-def*)
qed
qed

lemma *le-dag-antisym*:
fixes $x::dag$ **and** y
assumes $x-y: x \leq y$ **and** $y-x: y \leq x$
shows $x = y$
proof (*cases x=y*)
case *True* **thus** *?thesis* .
next

```

case False
with  $x-y$   $y-x$  show ?thesis
  by (auto simp add: less-dag-def le-dag-def intro: subdag-not-sym)
qed

lemma dag-less-le:
  fixes  $x::dag$  and  $y$ 
  shows  $(x < y) = (x \leq y \wedge x \neq y)$ 
  by (auto simp add: less-dag-def le-dag-def dest: subdag-neq)

instance
  by standard (auto simp add: dag-less-le le-dag-refl intro: le-dag-trans dest: le-dag-antisym)

end

lemma less-dag-Tip [simp]:  $\neg (x < Tip)$ 
  by (simp add: less-dag-def)

lemma less-dag-Node:  $x < (Node\ l\ a\ r) =$ 
   $(x \leq l \vee x \leq r)$ 
  by (auto simp add: order-le-less less-dag-def)

lemma less-dag-Node':  $x < (Node\ l\ a\ r) =$ 
   $(x=l \vee x=r \vee x < l \vee x < r)$ 
  by (simp add: less-dag-def)

lemma less-Node-dag:  $(Node\ l\ a\ r) < x \implies l < x \wedge r < x$ 
  by (auto simp add: less-dag-def dest: subdag-NodeD)

lemma less-dag-set-of:  $x < y \implies set-of\ x \subseteq set-of\ y$ 
  by (unfold less-dag-def, induct y, auto)

lemma le-dag-set-of:  $x \leq y \implies set-of\ x \subseteq set-of\ y$ 
  apply (unfold le-dag-def)
  apply (erule disjE)
  apply simp
  apply (erule less-dag-set-of)
  done

lemma DAG-less:  $DAG\ y \implies x < y \implies DAG\ x$ 
  by (induct y) (auto simp add: less-dag-Node')

lemma less-DAG-set-of:
  assumes  $x-less-y$ :  $x < y$ 
  assumes  $DAG-y$ :  $DAG\ y$ 
  shows  $set-of\ x \subseteq set-of\ y$ 
proof (cases y)
  case Tip with  $x-less-y$  show ?thesis by simp
next

```

```

case (Node l a r)
with DAG-y obtain a: a ∉ set-of l a ∉ set-of r
  by simp
from Node obtain l-less-y: l < y and r-less-y: r < y
  by (simp add: less-dag-Node)
from Node a obtain
  l-subset-y: set-of l ⊆ set-of y and
  r-subset-y: set-of r ⊆ set-of y
  by auto
from Node x-less-y have x=l ∨ x=r ∨ x < l ∨ x < r
  by (simp add: less-dag-Node')
thus ?thesis
proof (elim disjE)
  assume x=l
  with l-subset-y show ?thesis by simp
next
  assume x=r
  with r-subset-y show ?thesis by simp
next
  assume x < l
  hence set-of x ⊆ set-of l
    by (rule less-dag-set-of)
  also note l-subset-y
  finally show ?thesis .
next
  assume x < r
  hence set-of x ⊆ set-of r
    by (rule less-dag-set-of)
  also note r-subset-y
  finally show ?thesis .
qed
qed

```

lemma in-set-of-decomp:

$p \in \text{set-of } t = (\exists l r. t = (\text{Node } l \text{ } p \text{ } r) \vee \text{subdag } t (\text{Node } l \text{ } p \text{ } r))$
 (is ?A = ?B)

```

proof
  assume ?A thus ?B
    by (induct t) auto
next
  assume ?B thus ?A
    by (induct t) auto
qed

```

primrec Dag:: ref ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref) ⇒ dag ⇒ bool

where

Dag p l r Tip = (p = Null) |

Dag p l r (Node lt a rt) = (p = a ∧ p ≠ Null ∧

$$\text{Dag } (l p) l r lt \wedge \text{Dag } (r p) l r rt$$

lemma *Dag-Null* [*simp*]: $\text{Dag Null } l r t = (t = \text{Tip})$
by (*cases t*) *simp-all*

lemma *Dag-Ref* [*simp*]:
 $p \neq \text{Null} \implies \text{Dag } p l r t = (\exists lt rt. t = \text{Node } lt p rt \wedge$
 $\text{Dag } (l p) l r lt \wedge \text{Dag } (r p) l r rt)$
by (*cases t*) *auto*

lemma *Null-notin-Dag* [*simp, intro*]: $\bigwedge p l r. \text{Dag } p l r t \implies \text{Null} \notin \text{set-of } t$
by (*induct t*) *auto*

theorem *notin-Dag-update-l* [*simp*]:
 $\bigwedge p. q \notin \text{set-of } t \implies \text{Dag } p (l(q := y)) r t = \text{Dag } p l r t$
by (*induct t*) *auto*

theorem *notin-Dag-update-r* [*simp*]:
 $\bigwedge p. q \notin \text{set-of } t \implies \text{Dag } p l (r(q := y)) t = \text{Dag } p l r t$
by (*induct t*) *auto*

lemma *Dag-upd-same-l-lemma*: $\bigwedge p. p \neq \text{Null} \implies \neg \text{Dag } p (l(p:=p)) r t$
apply (*induct t*)
apply *simp*
apply (*simp (no-asm-simp) del: fun-upd-apply*)
apply (*simp (no-asm-simp) only: fun-upd-apply refl if-True*)
apply *blast*
done

lemma *Dag-upd-same-l* [*simp*]: $\text{Dag } p (l(p:=p)) r t = (p = \text{Null} \wedge t = \text{Tip})$
apply (*cases p=Null*)
apply *simp*
apply (*fast dest: Dag-upd-same-l-lemma*)
done

Dag-upd-same-l prevents $p \neq \text{Null} \implies \text{Dag } p (l(p := p)) r t = X$ from looping, because of *Dag-Ref* and *fun-upd-apply*.

lemma *Dag-upd-same-r-lemma*: $\bigwedge p. p \neq \text{Null} \implies \neg \text{Dag } p l (r(p:=p)) t$
apply (*induct t*)
apply *simp*
apply (*simp (no-asm-simp) del: fun-upd-apply*)
apply (*simp (no-asm-simp) only: fun-upd-apply refl if-True*)
apply *blast*
done

lemma *Dag-upd-same-r* [*simp*]: $\text{Dag } p l (r(p:=p)) t = (p = \text{Null} \wedge t = \text{Tip})$
apply (*cases p=Null*)
apply *simp*


```

apply (fast dest: Dag-upd-same-r-lemma)
done

lemma Dag-update-l-new [simp]:  $\llbracket \text{set-of } t \subseteq \text{set alloc} \rrbracket$ 
   $\implies \text{Dag } p \ (l(\text{new } (\text{set alloc}) := x)) \ r \ t = \text{Dag } p \ l \ r \ t$ 
by (rule notin-Dag-update-l) fastforce

lemma Dag-update-r-new [simp]:  $\llbracket \text{set-of } t \subseteq \text{set alloc} \rrbracket$ 
   $\implies \text{Dag } p \ l \ (r(\text{new } (\text{set alloc}) := x)) \ t = \text{Dag } p \ l \ r \ t$ 
by (rule notin-Dag-update-r) fastforce

lemma Dag-update-lI [intro]:
   $\llbracket \text{Dag } p \ l \ r \ t; q \notin \text{set-of } t \rrbracket \implies \text{Dag } p \ (l(q := y)) \ r \ t$ 
by simp

lemma Dag-update-rI [intro]:
   $\llbracket \text{Dag } p \ l \ r \ t; q \notin \text{set-of } t \rrbracket \implies \text{Dag } p \ l \ (r(q := y)) \ t$ 
by simp

lemma Dag-unique:  $\bigwedge p \ t2. \text{Dag } p \ l \ r \ t1 \implies \text{Dag } p \ l \ r \ t2 \implies t1=t2$ 
by (induct t1) auto

lemma Dag-unique1:  $\text{Dag } p \ l \ r \ t \implies \exists !t. \text{Dag } p \ l \ r \ t$ 
by (blast intro: Dag-unique)

lemma Dag-subdag:  $\bigwedge p. \text{Dag } p \ l \ r \ t \implies \text{subdag } t \ s \implies \exists q. \text{Dag } q \ l \ r \ s$ 
by (induct t) auto

lemma Dag-root-not-in-subdag-l [simp,intro]:
  assumes Dag (l p) l r t
  shows  $p \notin \text{set-of } t$ 
proof –
  {
    fix lt rt
    assume  $t = \text{Node } lt \ p \ rt$ 
    from assms have Dag (l p) l r lt
      by (clarsimp simp only: t Dag.simps)
    with assms have  $t=lt$ 
      by (rule Dag-unique)
    with t have False
      by simp
  }
moreover
  {
    fix lt rt
    assume subdag: subdag t (Node lt p rt)
    with assms obtain q where Dag q l r (Node lt p rt)
      by (rule Dag-subdag [elim-format]) iprover
    hence Dag (l p) l r lt
  }

```

```

    by auto
  with assms have  $t=lt$ 
    by (rule Dag-unique)
  moreover
  have subdag  $t$   $lt$ 
  proof –
    note subdag
    also have subdag (Node  $lt$   $p$   $rt$ )  $lt$  by simp
    finally show ?thesis .
  qed
  ultimately have False
    by (simp add: subdag-neq)
}
ultimately show ?thesis
  by (auto simp add: in-set-of-decomp)
qed

lemma Dag-root-not-in-subdag-r [simp, intro]:
  assumes Dag ( $r$   $p$ )  $l$   $r$   $t$ 
  shows  $p \notin \text{set-of } t$ 
  proof –
    {
      fix  $lt$   $rt$ 
      assume  $t$ :  $t = \text{Node } lt$   $p$   $rt$ 
      from assms have Dag ( $r$   $p$ )  $l$   $r$   $rt$ 
        by (clarsimp simp only:  $t$  Dag.simps)
      with assms have  $t=rt$ 
        by (rule Dag-unique)
      with  $t$  have False
        by simp
    }
  moreover
  {
    fix  $lt$   $rt$ 
    assume subdag: subdag  $t$  (Node  $lt$   $p$   $rt$ )
    with assms obtain  $q$  where Dag  $q$   $l$   $r$  (Node  $lt$   $p$   $rt$ )
      by (rule Dag-subdag [elim-format]) iprover
    hence Dag ( $r$   $p$ )  $l$   $r$   $rt$ 
      by auto
    with assms have  $t=rt$ 
      by (rule Dag-unique)
    moreover
    have subdag  $t$   $rt$ 
    proof –
      note subdag
      also have subdag (Node  $lt$   $p$   $rt$ )  $rt$  by simp
      finally show ?thesis .
    qed
    ultimately have False

```

```

    by (simp add: subdag-neq)
  }
  ultimately show ?thesis
    by (auto simp add: in-set-of-decomp)
qed

```

lemma *Dag-is-DAG*: $\bigwedge p\ l\ r. \text{Dag } p\ l\ r\ t \implies \text{DAG } t$
 by (induct t) auto

lemma *heaps-eq-Dag-eq*:
 $\bigwedge p. \forall x \in \text{set-of } t. l\ x = l'\ x \wedge r\ x = r'\ x$
 $\implies \text{Dag } p\ l\ r\ t = \text{Dag } p\ l'\ r'\ t$
 by (induct t) auto

lemma *heaps-eq-DagI1*:
 $\llbracket \text{Dag } p\ l\ r\ t; \forall x \in \text{set-of } t. l\ x = l'\ x \wedge r\ x = r'\ x \rrbracket$
 $\implies \text{Dag } p\ l'\ r'\ t$
 by (rule heaps-eq-Dag-eq [THEN iffD1])

lemma *heaps-eq-DagI2*:
 $\llbracket \text{Dag } p\ l'\ r'\ t; \forall x \in \text{set-of } t. l\ x = l'\ x \wedge r\ x = r'\ x \rrbracket$
 $\implies \text{Dag } p\ l\ r\ t$
 by (rule heaps-eq-Dag-eq [THEN iffD2]) auto

lemma *Dag-unique-all-impl-simp* [simp]:
 $\text{Dag } p\ l\ r\ t \implies (\forall t. \text{Dag } p\ l\ r\ t \longrightarrow P\ t) = P\ t$
 by (auto dest: Dag-unique)

lemma *Dag-unique-ex-conj-simp* [simp]:
 $\text{Dag } p\ l\ r\ t \implies (\exists t. \text{Dag } p\ l\ r\ t \wedge P\ t) = P\ t$
 by (auto dest: Dag-unique)

lemma *Dags-eq-hp-eq*:
 $\bigwedge p\ p'. \llbracket \text{Dag } p\ l\ r\ t; \text{Dag } p'\ l'\ r'\ t \rrbracket \implies$
 $p' = p \wedge (\forall no \in \text{set-of } t. l'\ no = l\ no \wedge r'\ no = r\ no)$
 by (induct t) auto

definition *isDag* :: $\text{ref} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{bool}$
 where $\text{isDag } p\ l\ r = (\exists t. \text{Dag } p\ l\ r\ t)$

definition *dag* :: $\text{ref} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{dag}$
 where $\text{dag } p\ l\ r = (\text{THE } t. \text{Dag } p\ l\ r\ t)$

lemma *Dag-conv-isDag-dag*: $\text{Dag } p\ l\ r\ t = (\text{isDag } p\ l\ r \wedge t = \text{dag } p\ l\ r)$
 apply (simp add: isDag-def dag-def)
 apply (rule iffI)
 apply (rule conjI)
 apply blast

```

apply (subst the1-equality)
  apply (erule Dag-unique1)
  apply assumption
apply (rule refl)
apply clarify
apply (rule theI)
  apply assumption
apply (erule (1) Dag-unique)
done

```

lemma *Dag-dag*: $Dag\ p\ l\ r\ t \implies dag\ p\ l\ r = t$
by (*simp add: Dag-conv-isDag-dag*)

end

3 General Lemmas on BDD Abstractions

theory *General* **imports** *BinDag* **begin**

definition *subdag-eq*:: $dag \Rightarrow dag \Rightarrow bool$ **where**
subdag-eq $t_1\ t_2 = (t_1 = t_2 \vee subdag\ t_1\ t_2)$

primrec *root* :: $dag \Rightarrow ref$
where
root *Tip* = *Null* |
root (*Node* $l\ a\ r$) = a

fun *isLeaf* :: $dag \Rightarrow bool$ **where**
isLeaf *Tip* = *False*
| *isLeaf* (*Node* $Tip\ v\ Tip$) = *True*
| *isLeaf* (*Node* (*Node* $l\ v_1\ r$) $v_2\ Tip$) = *False*
| *isLeaf* (*Node* $Tip\ v_1$ (*Node* $l\ v_2\ r$)) = *False*

datatype *bdt* = *Zero* | *One* | *Bdt-Node* $bdt\ nat\ bdt$

fun *bdt-fn* :: $dag \Rightarrow (ref \Rightarrow nat) \Rightarrow bdt\ option$ **where**
bdt-fn *Tip* = ($\lambda bdtvar . None$)
| *bdt-fn* (*Node* $Tip\ vref\ Tip$) =
($\lambda bdtvar .$
(*if* ($bdtvar\ vref = 0$)
then *Some Zero*
else (*if* ($bdtvar\ vref = 1$)
then *Some One*
else *None*)))
| *bdt-fn* (*Node* $Tip\ vref$ (*Node* $l\ vref1\ r$)) = ($\lambda bdtvar . None$)
| *bdt-fn* (*Node* (*Node* $l\ vref1\ r$) $vref\ Tip$) = ($\lambda bdtvar . None$)
| *bdt-fn* (*Node* (*Node* $l1\ vref1\ r1$) $vref$ (*Node* $l2\ vref2\ r2$)) =
($\lambda bdtvar .$

```

(if (bdtvar vref = 0 ∨ bdtvar vref = 1)
  then None
  else
    (case (bdt-fn (Node l1 vref1 r1) bdtvar) of
      None ⇒ None
    |(Some b1) ⇒
      (case (bdt-fn (Node l2 vref2 r2) bdtvar) of
        None ⇒ None
      |(Some b2) ⇒ Some (Bdt-Node b1 (bdtvar vref) b2))))))

```

abbreviation $bdt == bdt\text{-}fn$

primrec $eval :: bdt \Rightarrow bool\ list \Rightarrow bool$

where

$eval\ Zero\ env = False$ |

$eval\ One\ env = True$ |

$eval\ (Bdt\text{-}Node\ l\ v\ r)\ env = (if\ (env\ !\ v)\ then\ eval\ r\ env\ else\ eval\ l\ env)$

fun $ordered\text{-}bdt :: bdt \Rightarrow bool$ **where**

$ordered\text{-}bdt\ Zero = True$

| $ordered\text{-}bdt\ One = True$

| $ordered\text{-}bdt\ (Bdt\text{-}Node\ (Bdt\text{-}Node\ l1\ v1\ r1)\ v\ (Bdt\text{-}Node\ l2\ v2\ r2)) =$

$((v1 < v) \wedge (v2 < v) \wedge$

$(ordered\text{-}bdt\ (Bdt\text{-}Node\ l1\ v1\ r1)) \wedge (ordered\text{-}bdt\ (Bdt\text{-}Node\ l2\ v2\ r2)))$

| $ordered\text{-}bdt\ (Bdt\text{-}Node\ (Bdt\text{-}Node\ l1\ v1\ r1)\ v\ r) =$

$((v1 < v) \wedge (ordered\text{-}bdt\ (Bdt\text{-}Node\ l1\ v1\ r1)))$

| $ordered\text{-}bdt\ (Bdt\text{-}Node\ l\ v\ (Bdt\text{-}Node\ l2\ v2\ r2)) =$

$((v2 < v) \wedge (ordered\text{-}bdt\ (Bdt\text{-}Node\ l2\ v2\ r2)))$

| $ordered\text{-}bdt\ (Bdt\text{-}Node\ l\ v\ r) = True$

fun $ordered :: dag \Rightarrow (ref \Rightarrow nat) \Rightarrow bool$ **where**

$ordered\ Tip = (\lambda\ var.\ True)$

| $ordered\ (Node\ (Node\ l_1\ v_1\ r_1)\ v\ (Node\ l_2\ v_2\ r_2)) =$

$(\lambda\ var.\ (var\ v_1 < var\ v \wedge var\ v_2 < var\ v) \wedge$

$(ordered\ (Node\ l_1\ v_1\ r_1)\ var) \wedge (ordered\ (Node\ l_2\ v_2\ r_2)\ var))$

| $ordered\ (Node\ Tip\ v\ Tip) = (\lambda\ var.\ (True))$

| $ordered\ (Node\ Tip\ v\ r) =$

$(\lambda\ var.\ (var\ (root\ r) < var\ v) \wedge (ordered\ r\ var))$

| $ordered\ (Node\ l\ v\ Tip) =$

$(\lambda\ var.\ (var\ (root\ l) < var\ v) \wedge (ordered\ l\ var))$

primrec $max\text{-}var :: bdt \Rightarrow nat$

where

$max\text{-}var\ Zero = 0$ |

$max\text{-var } One = 1 \mid$
 $max\text{-var } (Bdt\text{-Node } l \ v \ r) = max \ v \ (max \ (max\text{-var } l) \ (max\text{-var } r))$

lemma *eval-zero*: $\llbracket bdt \ (Node \ l \ v \ r) \ var = Some \ x;$
 $var \ (root \ (Node \ l \ v \ r)) = (0::nat) \rrbracket \implies x = Zero$
apply (*cases l*)
apply (*cases r*)
apply *simp*
apply *simp*
apply (*cases r*)
apply *simp*
apply *simp*
done

lemma *bdt-Some-One-iff* [*simp*]:
 $(bdt \ t \ var = Some \ One) = (\exists \ p. \ t = Node \ Tip \ p \ Tip \wedge \ var \ p = 1)$
apply (*induct t rule: bdt-fn.induct*)
apply (*auto split: option.splits*)
done

lemma *bdt-Some-Zero-iff* [*simp*]:
 $(bdt \ t \ var = Some \ Zero) = (\exists \ p. \ t = Node \ Tip \ p \ Tip \wedge \ var \ p = 0)$
apply (*induct t rule: bdt-fn.induct*)
apply (*auto split: option.splits*)
done

lemma *bdt-Some-Node-iff* [*simp*]:
 $(bdt \ t \ var = Some \ (Bdt\text{-Node} \ bdt1 \ v \ bdt2)) =$
 $(\exists \ p \ l \ r. \ t = Node \ l \ p \ r \wedge \ bdt \ l \ var = Some \ bdt1 \wedge \ bdt \ r \ var = Some \ bdt2 \wedge$
 $1 < v \wedge \ var \ p = v)$
apply (*induct t rule: bdt-fn.induct*)
prefer 5
apply (*fastforce split: if-splits option.splits*)
apply *auto*
done

lemma *balanced-bdt*:
 $\bigwedge \ p \ bdt1. \ \llbracket Dag \ p \ low \ high \ t; \ bdt \ t \ var = Some \ bdt1; \ no \in \ set\text{-of} \ t \rrbracket$
 $\implies (low \ no = Null) = (high \ no = Null)$
proof (*induct t*)
case *Tip*
then show *?case by simp*
next
case (*Node lt a rt*)
note *NN= this*
have *bdt1: bdt (Node lt a rt) var = Some bdt1 by fact*
have *no-in-t: no ∈ set-of (Node lt a rt) by fact*
have *p-tree: Dag p low high (Node lt a rt) by fact*

```

from Node.prems obtain
  lt: Dag (low p) low high lt and
  rt: Dag (high p) low high rt
  by auto
show ?case
proof (cases lt)
  case (Node llt l rlt)
  note Nlt=this
  show ?thesis
  proof (cases rt)
  case (Node lrt r rrt)
  note Nrt=this
  from Nlt Nrt bdt1 obtain lbdt rbdt where
    lbdt-def: bdt lt var = Some lbdt and
    rbdt-def: bdt rt var = Some rbdt and
    bdt1-def: bdt1 = Bdt-Node lbdt (var a) rbdt
    by (auto split: if-split-asm option.splits)
  from no-in-t show ?thesis
  proof (simp, elim disjE)
    assume no = a
    with p-tree Nlt Nrt show ?thesis
    by auto
  next
    assume no ∈ set-of lt
    with Node.hyps lbdt-def lt show ?thesis
    by simp
  next
    assume no ∈ set-of rt
    with Node.hyps rbdt-def rt show ?thesis
    by simp
  qed
next
  case Tip
  with Nlt bdt1 show ?thesis
  by simp
  qed
next
  case Tip
  note ltTip=this
  show ?thesis
  proof (cases rt)
  case Tip
  with ltTip bdt1 no-in-t p-tree show ?thesis
  by auto
  next
  case (Node rlt r rrt)
  with bdt1 ltTip show ?thesis
  by simp
  qed

```

qed
qed

primrec *dag-map* :: (ref \Rightarrow ref) \Rightarrow dag \Rightarrow dag

where

dag-map *f* *Tip* = *Tip* |

dag-map *f* (*Node* *l a r*) = (*Node* (*dag-map* *f* *l*) (*f* *a*) (*dag-map* *f* *r*))

definition *wf-marking* :: dag \Rightarrow (ref \Rightarrow bool) \Rightarrow (ref \Rightarrow bool) \Rightarrow bool \Rightarrow bool

where

wf-marking *t* *mark-old* *mark-new* *marked* =

(*case* *t* of *Tip* \Rightarrow *mark-new* = *mark-old*

| (*Node* *lt p rt*) \Rightarrow

(\forall *n*. *n* \notin *set-of* *t* \longrightarrow *mark-new* *n* = *mark-old* *n*) \wedge

(\forall *n*. *n* \in *set-of* *t* \longrightarrow *mark-new* *n* = *marked*))

definition *dag-in-levellist*:: dag \Rightarrow (ref list list) \Rightarrow (ref \Rightarrow nat) \Rightarrow bool

where *dag-in-levellist* *t* *levellist* *var* = (*t* \neq *Tip* \longrightarrow

(\forall *st*. *subdag-eq* *t* *st* \longrightarrow *root* *st* \in *set* (*levellist* ! (*var* (*root* *st*))))))

lemma *set-of-nn*: \llbracket *Dag* *p* *low* *high* *t*; *n* \in *set-of* *t* $\rrbracket \Longrightarrow$ *n* \neq *Null*

apply (*induct* *t*)

apply *simp*

apply *auto*

done

lemma *subnodes-ordered* [*rule-format*]:

\forall *p*. *n* \in *set-of* *t* \longrightarrow *Dag* *p* *low* *high* *t* \longrightarrow *ordered* *t* *var* \longrightarrow *var* *n* \leq *var* *p*

apply (*induct* *t*)

apply *simp*

apply (*intro* *allI*)

apply (*erule-tac* *x=low* *p* **in** *allE*)

apply (*erule-tac* *x=high* *p* **in** *allE*)

apply *clarsimp*

apply (*case-tac* *t1*)

apply (*case-tac* *t2*)

apply *simp*

apply *fastforce*

apply (*case-tac* *t2*)

apply *fastforce*

apply *fastforce*

done

lemma *ordered-set-of*:

\bigwedge *x*. \llbracket *ordered* *t* *var*; *x* \in *set-of* *t* $\rrbracket \Longrightarrow$ *var* *x* \leq *var* (*root* *t*)

apply (*induct* *t*)

apply *simp*


```

apply simp
apply (elim disjE)
apply simp
apply (case-tac t1)
apply simp
apply (case-tac t2)
apply fastforce
apply fastforce
apply (case-tac t2)
apply simp
apply (case-tac t1)
apply fastforce
apply fastforce
done

```

```

lemma dag-setofD:  $\bigwedge p \text{ low high } n. \llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t \rrbracket \implies$ 
  ( $\exists nt. \text{Dag } n \text{ low high } nt$ )  $\wedge$  ( $\forall nt. \text{Dag } n \text{ low high } nt \longrightarrow \text{set-of } nt \subseteq \text{set-of } t$ )
apply (induct t)
apply simp
apply auto
apply fastforce
apply (fastforce dest: Dag-unique)
apply (fastforce dest: Dag-unique)
apply blast
apply blast
done

```

```

lemma dag-setof-exD:  $\llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t \rrbracket \implies \exists nt. \text{Dag } n \text{ low high } nt$ 
apply (simp add: dag-setofD)
done

```

```

lemma dag-setof-subsetD:  $\llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t; \text{Dag } n \text{ low high } nt \rrbracket \implies$ 
   $\text{set-of } nt \subseteq \text{set-of } t$ 
apply (simp add: dag-setofD)
done

```

```

lemma subdag-parentdag-low:  $\text{not } \leq lt \implies \text{not } \leq (\text{Node } lt \text{ } p \text{ } rt)$  for not
apply (cases not = lt)
apply (cases lt)
apply simp
apply (cases rt)
apply simp
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
done

```

```

lemma subdag-parentdag-high:  $not \leq rt \implies not \leq (Node\ lt\ p\ rt)$  for not
apply (cases not = rt)
apply (cases lt)
apply simp
apply (cases rt)
apply simp
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
apply (simp add: le-dag-def less-dag-def)
done

```

```

lemma set-of-subdag:  $\bigwedge p\ not\ no.$ 
 $\llbracket Dag\ p\ low\ high\ t; Dag\ no\ low\ high\ not; no \in set-of\ t \rrbracket \implies not \leq t$ 
proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt po rt)
  note rtNode=this
  from Node.prems have ppo: p=po
  by simp
  show ?case
  proof (cases no = p)
  case True
  with ppo Node.prems have not = (Node lt po rt)
  by (simp add: Dag-unique del: Dag-Ref)
  with Node.prems ppo show ?thesis by (simp add: subdag-eq-def)
next
  assume no  $\neq$  p
  with Node.prems have no-in-ltorrt: no  $\in$  set-of lt  $\vee$  no  $\in$  set-of rt
  by simp
  show ?thesis
  proof (cases no  $\in$  set-of lt)
  case True
  from Node.prems ppo have Dag (low po) low high lt
  by simp
  with Node.prems ppo True have not  $\leq$  lt
  apply –
  apply (rule Node.hyps)
  apply assumption+
  done
  with Node.prems no-in-ltorrt show ?thesis
  apply –
  apply (rule subdag-parentdag-low)
  apply simp
  done
next
  assume no  $\notin$  set-of lt

```

```

with no-in-ltorrt have no-in-rt: no ∈ set-of rt
  by simp
from Node.premis ppo have Dag (high po) low high rt
  by simp
with Node.premis ppo no-in-rt have not <= rt
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
with Node.premis no-in-rt show ?thesis
  apply -
  apply (rule subdag-parentdag-high)
  apply simp
  done
qed
qed
qed

```

lemma *children-ordered*: $\llbracket \text{ordered } (Node\ lt\ p\ rt)\ \text{var} \rrbracket \implies \text{ordered } lt\ \text{var} \wedge \text{ordered } rt\ \text{var}$

```

proof (cases lt)
  case Tip
  note ltTip=this
  assume orderedNode: ordered (Node lt p rt) var
  show ?thesis
  proof (cases rt)
    case Tip
    note rtTip = this
    with ltTip show ?thesis
    by simp
  next
    case (Node lrt r rrt)
    with orderedNode ltTip show ?thesis
    by simp
  qed
next
case (Node llt l rlt)
note ltNode=this
assume orderedNode: ordered (Node lt p rt) var
show ?thesis
proof (cases rt)
  case Tip
  note rtTip = this
  with orderedNode ltNode show ?thesis by simp
next
case (Node lrt r rrt)
note rtNode = this
with orderedNode ltNode show ?thesis by simp
qed

```

```

qed

lemma ordered-subdag:  $\llbracket \text{ordered } t \text{ var}; \text{ not } \leq t \rrbracket \implies \text{ordered not var for not}$ 
proof (induct t)
  case Tip
  then show ?thesis by (simp add: less-dag-def le-dag-def)
next
  case (Node lt p rt)
  show ?case
  proof (cases not = Node lt p rt)
    case True
    with Node.premis show ?thesis by simp
  next
    assume notnt: not  $\neq$  Node lt p rt
    with Node.premis have notstltorrt: not  $\leq$  lt  $\vee$  not  $\leq$  rt
    apply -
    apply (simp add: less-dag-def le-dag-def)
    apply fastforce
    done
  from Node.premis have ord-lt: ordered lt var
  apply -
  apply (drule children-ordered)
  apply simp
  done
  from Node.premis have ord-rt: ordered rt var
  apply -
  apply (drule children-ordered)
  apply simp
  done
  show ?thesis
  proof (cases not  $\leq$  lt)
    case True
    with ord-lt show ?thesis
    apply -
    apply (rule Node.hyps)
    apply assumption+
    done
  next
    assume  $\neg$  not  $\leq$  lt
    with notstltorrt have notinrt: not  $\leq$  rt
    by simp
    from Node.hyps have hyprt:  $\llbracket \text{ordered } rt \text{ var}; \text{ not } \leq rt \rrbracket \implies \text{ordered not var}$ 
  by simp
  from notinrt ord-rt show ?thesis
  apply -
  apply (rule hyprt)
  apply assumption+
  done
qed

```

qed
qed

lemma *subdag-ordered*:

\wedge *not no p. [[ordered t var; Dag p low high t; Dag no low high not;*
 $no \in \text{set-of } t]] \implies \text{ordered not var}$

proof (*induct t*)

case *Tip*

from *Tip.prem*s **show** *?case* **by** *simp*

next

case (*Node lt po rt*)

note *nN=this*

show *?case*

proof (*cases lt*)

case *Tip*

note *ltTip=this*

show *?thesis*

proof (*cases rt*)

case *Tip*

from *Node.prem*s **have** *ppo: p=po*

by *simp*

from *Tip ltTip Node.prem*s **have** *no=p*

by *simp*

with *ppo Node.prem*s **have** *not=(Node lt po rt)*

by (*simp del: Dag-Ref add: Dag-unique*)

with *Node.prem*s **show** *?thesis* **by** *simp*

next

case (*Node lrnot rn rrnot*)

from *Node.prem*s *ltTip Node* **have** *ord-rt: ordered rt var*

by *simp*

from *Node.prem*s **have** *ppo: p=po*

by *simp*

from *Node.prem*s **have** *ponN: po \neq Null*

by *auto*

with *ppo ponN ltTip Node.prem*s **have** **: Dag (high po) low high rt*

by *auto*

show *?thesis*

proof (*cases no=po*)

case *True*

with *ppo Node.prem*s **have** *not = Node lt po rt*

by (*simp del: Dag-Ref add: Dag-unique*)

with *Node.prem*s **show** *?thesis*

by *simp*

next

case *False*

with *Node.prem*s *ltTip* **have** *no \in set-of rt*

by *simp*

with *ord-rt * \langle Dag no low high not \rangle* **show** *?thesis*

```

      by (rule Node.hyps)
    qed
  qed
next
case (Node llt l rlt)
note ltNode=this
show ?thesis
proof (cases rt)
  case Tip
  from Node.prem1 Tip ltNode have ord-lt: ordered lt var
    by simp
  from Node.prem1 have ppo: p=po
    by simp
  from Node.prem1 have ponN: po ≠ Null
    by auto
  with ppo ponN Tip Node.prem1 ltNode have *: Dag (low po) low high lt
    by auto
  show ?thesis
proof (cases no=po)
  case True
  with ppo Node.prem1 have not = (Node lt po rt)
    by (simp del: Dag-Ref add: Dag-unique)
  with Node.prem1 show ?thesis by simp
next
case False
  with Node.prem1 Tip have no ∈ set-of lt
    by simp
  with ord-lt * ⟨Dag no low high not⟩ show ?thesis
    by (rule Node.hyps)
qed
next
case (Node lrt r rrt)
from Node.prem1 have ppo: p=po
  by simp
from Node.prem1 Node ltNode have ord-lt: ordered lt var
  by simp
from Node.prem1 Node ltNode have ord-rt: ordered rt var
  by simp
from Node.prem1 have ponN: po ≠ Null
  by auto
with ppo ponN Node Node.prem1 ltNode have lt-Dag: Dag (low po) low high
lt
  by auto
with ppo ponN Node Node.prem1 ltNode have rt-Dag: Dag (high po) low high
rt
  by auto
show ?thesis
proof (cases no = po)
  case True

```

```

with ppo Node.premis have not = (Node lt po rt)
  by (simp del: Dag-Ref add: Dag-unique)
with Node.premis show ?thesis by simp
next
assume no ≠ po
with Node.premis have no-in-lt-or-rt: no ∈ set-of lt ∨ no ∈ set-of rt
  by simp
show ?thesis
proof (cases no ∈ set-of lt)
  case True
  with ord-lt lt-Dag Node.premis show ?thesis
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
next
assume no ∉ set-of lt
with no-in-lt-or-rt have no-in-rt: no ∈ set-of rt
  by simp
from Node.hyps have hyp2:
   $\bigwedge p \text{ no not. } \llbracket \text{ordered } rt \text{ var; Dag } p \text{ low high } rt; \text{ Dag no low high not; no} \in \text{set-of } rt \rrbracket$ 
 $\implies \text{ordered not var}$ 
  apply -
  apply assumption
  done
from no-in-rt ord-rt rt-Dag Node.premis show ?thesis
  apply -
  apply (rule hyp2)
  apply assumption+
  done
qed
qed
qed
qed
qed

```

lemma *elem-set-of*: $\bigwedge x \text{ st. } \llbracket x \in \text{set-of } st; \text{set-of } st \subseteq \text{set-of } t \rrbracket \implies x \in \text{set-of } t$
by *blast*

definition *wf-ll* :: *dag* \Rightarrow *ref list list* \Rightarrow (*ref* \Rightarrow *nat*) \Rightarrow *bool*

where

wf-ll *t* *levellist* *var* =

$((\forall p. p \in \text{set-of } t \longrightarrow p \in \text{set } (\text{levellist } ! \text{ var } p)) \wedge$
 $(\forall k < \text{length } \text{levellist}. \forall p \in \text{set } (\text{levellist } ! k). p \in \text{set-of } t \wedge \text{var } p = k))$

definition *cong-eval* :: $bdt \Rightarrow bdt \Rightarrow \text{bool}$ (**infix** \sim 60)
where *cong-eval* bdt_1 $bdt_2 = (\text{eval } bdt_1 = \text{eval } bdt_2)$

lemma *cong-eval-sym*: $l \sim r = r \sim l$
by (*auto simp add: cong-eval-def*)

lemma *cong-eval-trans*: $\llbracket l \sim r; r \sim t \rrbracket \Longrightarrow l \sim t$
by (*simp add: cong-eval-def*)

lemma *cong-eval-child-high*: $l \sim r \Longrightarrow r \sim (\text{Bdt-Node } l \ v \ r)$
apply (*simp add: cong-eval-def*)
apply (*rule ext*)
apply *auto*
done

lemma *cong-eval-child-low*: $l \sim r \Longrightarrow l \sim (\text{Bdt-Node } l \ v \ r)$
apply (*simp add: cong-eval-def*)
apply (*rule ext*)
apply *auto*
done

definition *null-comp* :: $(\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref})$ (**infix** \times 60)
where *null-comp* $a \ b = (\lambda p. (\text{if } (b \ p) = \text{Null} \text{ then } \text{Null} \text{ else } ((a \circ b) \ p)))$

lemma *null-comp-not-Null* [*simp*]: $h \ q \neq \text{Null} \Longrightarrow (g \times h) \ q = g \ (h \ q)$
by (*simp add: null-comp-def*)

lemma *id-trans*: $(a \times \text{id}) \ (b \ (c \ p)) = (a \times b) \ (c \ p)$
by (*simp add: null-comp-def*)

definition *Nodes* :: $\text{nat} \Rightarrow \text{ref list list} \Rightarrow \text{ref set}$
where *Nodes* $i \ \text{levellist} = (\bigcup k \in \{k. k < i\} . \text{set } (\text{levellist } ! \ k))$

inductive-set *Dags* :: $\text{ref set} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{dag set}$
for *nodes low high*
where
DagsI: $\llbracket \text{set-of } t \subseteq \text{nodes}; \text{Dag } p \ \text{low high } t; t \neq \text{Tip} \rrbracket$
 $\Longrightarrow t \in \text{Dags } \text{nodes } \text{low high}$

lemma *empty-Dags* [*simp*]: $\text{Dags } \{\} \ \text{low high} = \{\}$
apply *rule*
apply *rule*
apply (*erule Dags.cases*)
apply (*case-tac t*)
apply *simp*

apply *simp*
apply *rule*
done

definition *isLeaf-pt* :: $ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow bool$
where *isLeaf-pt* p *low high* = $(low\ p = Null \wedge high\ p = Null)$

definition *repNodes-eq* :: $ref \Rightarrow ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow bool$

where

repNodes-eq p q *low high rep* ==
 $(rep \times high)\ p = (rep \times high)\ q \wedge (rep \times low)\ p = (rep \times low)\ q$

definition *isomorphic-dags-eq* :: $dag \Rightarrow dag \Rightarrow (ref \Rightarrow nat) \Rightarrow bool$

where

isomorphic-dags-eq st_1 st_2 *var* =
 $(\forall bdt_1\ bdt_2. (bdt\ st_1\ var = Some\ bdt_1 \wedge bdt\ st_2\ var = Some\ bdt_2 \wedge (bdt_1 = bdt_2)))$
 $\longrightarrow st_1 = st_2)$

lemma *isomorphic-dags-eq-sym*: *isomorphic-dags-eq* st_1 st_2 *var* = *isomorphic-dags-eq* st_2 st_1 *var*

by (*auto simp add: isomorphic-dags-eq-def*)

definition *shared* :: $dag \Rightarrow (ref \Rightarrow nat) \Rightarrow bool$

where *shared* t *var* = $(\forall st_1\ st_2. (st_1 \leq t \wedge st_2 \leq t) \longrightarrow isomorphic-dags-eq\ st_1\ st_2\ var)$

fun *reduced* :: $dag \Rightarrow bool$ **where**

reduced *Tip* = *True*
| *reduced* (*Node* *Tip* v *Tip*) = *True*
| *reduced* (*Node* l v r) = $(l \neq r \wedge reduced\ l \wedge reduced\ r)$

primrec *reduced-bdt* :: $bdt \Rightarrow bool$

where

reduced-bdt *Zero* = *True*
| *reduced-bdt* *One* = *True*
| *reduced-bdt* (*Bdt-Node* l v r) =
 $(if\ l = r\ then\ False$
 $else\ (reduced-bdt\ l \wedge reduced-bdt\ r))$

lemma *replicate-elem*: $i < n \implies (replicate\ n\ x\ !i) = x$

```

apply (induct n)
apply simp
apply (cases i)
apply auto
done

```

lemma *no-in-one-ll*:

```

[[wf-ll pret levellista var; i < length levellista; j < length levellista;
  no ∈ set (levellista ! i); i ≠ j]]
  ⇒ no ∉ set (levellista ! j)
apply (unfold wf-ll-def)
apply (erule conjE)
apply (rotate-tac 5)
apply (frule-tac x = i and ?R= no ∈ set-of pret ∧ var no = i in allE)
apply (erule impE)
apply simp
apply (rotate-tac 6)
apply (erule-tac x=no in ballE)
apply assumption
apply simp
apply (cases no ∉ set (levellista ! j))
apply assumption
apply (erule-tac x=j in allE)
apply (erule impE)
apply assumption
apply (rotate-tac 7)
apply (erule-tac x=no in ballE)
prefer 2
apply assumption
apply (elim conjE)
apply (thin-tac ∀ q. q ∈ set-of pret → q ∈ set (levellista ! var q))
apply fastforce
done

```

lemma *nodes-in-wf-ll*:

```

[[wf-ll pret levellista var; i < length levellista; no ∈ set (levellista ! i)]]
  ⇒ var no = i ∧ no ∈ set-of pret
apply (simp add: wf-ll-def)
done

```

lemma *subelem-set-of-low*:

```

∧ p. [[ x ∈ set-of t; x ≠ Null; low x ≠ Null; Dag p low high t ]]
  ⇒ (low x) ∈ set-of t
proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt po rt)
  note tNode=this

```

```

then have ppo: p=po by simp
show ?case
proof (cases x=p)
  case True
  with Node.prem1s have lxrootlt: low x = root lt
  proof (cases lt)
    case Tip
    with True Node.prem1s show ?thesis
      by auto
  next
  case (Node llt l rlt)
  with Node.prem1s True show ?thesis
    by auto
  qed
with True Node.prem1s have low x ∈ set-of (Node lt p rt)
proof (cases lt)
  case Tip
  with lxrootlt Node.prem1s show ?thesis
    by simp
  next
  case (Node llt l rlt)
  with lxrootlt Node.prem1s show ?thesis
    by simp
  qed
with ppo show ?thesis
  by simp
next
assume xnp: x ≠ p
with Node.prem1s have x ∈ set-of lt ∨ x ∈ set-of rt
  by simp
show ?thesis
proof (cases x ∈ set-of lt)
  case True
  note xinlt=this
  from Node.prem1s have Dag (low p) low high lt
    by fastforce
  with Node.prem1s True have low x ∈ set-of lt
    apply -
    apply (rule Node.hyps)
    apply assumption+
    done
  with Node.prem1s show ?thesis
    by auto
  next
  assume xnotinlt: x ∉ set-of lt
  with xnp Node.prem1s have xinrt: x ∈ set-of rt
    by simp
  from Node.prem1s have Dag (high p) low high rt
    by fastforce

```

```

with Node.premis xinrt have low x ∈ set-of rt
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
with Node.premis show ?thesis
  by auto
qed
qed
qed

lemma subelem-set-of-high:
 $\bigwedge p. \llbracket x \in \text{set-of } t; x \neq \text{Null}; \text{high } x \neq \text{Null}; \text{Dag } p \text{ low high } t \rrbracket$ 
 $\implies (\text{high } x) \in \text{set-of } t$ 
proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt po rt)
  note tNode=this
  then have ppo: p=po by simp
  show ?case
  proof (cases x=p)
    case True
    with Node.premis have lxrootlt: high x = root rt
    proof (cases rt)
      case Tip
      with True Node.premis show ?thesis
        by auto
    next
      case (Node lrt l rrt)
      with Node.premis True show ?thesis
        by auto
    qed
  with True Node.premis have high x ∈ set-of (Node lt p rt)
  proof (cases rt)
    case Tip
    with lxrootlt Node.premis show ?thesis
      by simp
  next
    case (Node lrt l rrt)
    with lxrootlt Node.premis show ?thesis
      by simp
  qed
  with ppo show ?thesis
    by simp
next
  assume xnp: x ≠ p
  with Node.premis have x ∈ set-of lt ∨ x ∈ set-of rt

```

```

    by simp
  show ?thesis
proof (cases x ∈ set-of lt)
  case True
  note xinlt=this
  from Node.prem1 have Dag (low p) low high lt
    by fastforce
  with Node.prem1 True have high x ∈ set-of lt
    apply -
    apply (rule Node.hyps)
    apply assumption+
  done
  with Node.prem1 show ?thesis
    by auto
next
  assume xnotinlt: x ∉ set-of lt
  with xnp Node.prem1 have xinrt: x ∈ set-of rt
    by simp
  from Node.prem1 have Dag (high p) low high rt
    by fastforce
  with Node.prem1 xinrt have high x ∈ set-of rt
    apply -
    apply (rule Node.hyps)
    apply assumption+
  done
  with Node.prem1 show ?thesis
    by auto
qed
qed
qed

lemma set-split: {k. k < (Suc n)} = {k. k < n} ∪ {n}
apply auto
done

lemma Nodes-levellist-subset-t:
  [[wf-ll t levellist var; i <= length levellist]] ⇒ Nodes i levellist ⊆ set-of t
proof (induct i)
  case 0
  show ?case by (simp add: Nodes-def)
next
  case (Suc n)
  from Suc.prem1 Suc.hyps have Nodestn-in-t: Nodes n levellist ⊆ set-of t
    by simp
  from Suc.prem1 have ∀ x ∈ set (levellist ! n). x ∈ set-of t
    apply -
    apply (rule ballI)
    apply (simp add: wf-ll-def)

```

```

apply (erule conjE)
apply (thin-tac  $\forall q. q \in \text{set-of } t \longrightarrow q \in \text{set } (\text{levellist } ! \text{ var } q)$ )
apply (erule-tac  $x=n$  in allE)
apply (erule impE)
apply simp
apply fastforce
done
with Suc.prem have  $\text{set } (\text{levellist } ! n) \subseteq \text{set-of } t$ 
apply blast
done
with Suc.prem Nodesn-in-t show ?case
apply (simp add: Nodes-def)
apply (simp add: set-split)
done
qed

```

lemma bdt-child:

```

 $\llbracket \text{bdt } (\text{Node } (\text{Node } \text{llt } l \text{ rlt}) p (\text{Node } \text{lrt } r \text{ rrt})) \text{ var} = \text{Some } \text{bdt1} \rrbracket$ 
 $\implies \exists \text{lbdt } \text{rbdt}. \text{bdt } (\text{Node } \text{llt } l \text{ rlt}) \text{ var} = \text{Some } \text{lbdt} \wedge$ 
 $\text{bdt } (\text{Node } \text{lrt } r \text{ rrt}) \text{ var} = \text{Some } \text{rbdt}$ 
by (simp split: option.splits)

```

lemma subbdt-ex-dag-def:

```

 $\bigwedge \text{bdt1 } p. \llbracket \text{Dag } p \text{ low high } t; \text{bdt } t \text{ var} = \text{Some } \text{bdt1}; \text{Dag } no \text{ low high not};$ 
 $no \in \text{set-of } t \rrbracket \implies \exists \text{bdt2}. \text{bdt } not \text{ var} = \text{Some } \text{bdt2} \text{ for not}$ 
proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt po rt)
  note pNode=this
  with Node.prem have p-po:  $p=po$  by simp
  show ?case
  proof (cases no = po)
    case True
    note no-eq-po=this
    from p-po Node.prem no-eq-po have not = (Node lt po rt) by (simp add:
Dag-unique del: Dag-Ref)
    with Node.prem have bdt not var = Some bdt1 by (simp add: le-dag-def)
    then show ?thesis by simp
  next
  assume no  $\neq$  po
  with Node.prem have no-in-lt-or-rt:  $no \in \text{set-of } lt \vee no \in \text{set-of } rt$  by simp
  show ?thesis
  proof (cases no  $\in$  set-of lt)
    case True
    note no-in-lt=this
    from Node.prem p-po have lt-dag: Dag (low po) low high lt by simp

```

```

from Node.prems have lbdt-def:  $\exists$  lbdt. bdt lt var = Some lbdt
proof (cases lt)
  case Tip
  with Node.prems no-in-lt show ?thesis by (simp add: le-dag-def)
next
  case (Node llt l rlt)
  note lNode=this
  show ?thesis
  proof (cases rt)
    case Tip
    note rNode=this
    with lNode Node.prems show ?thesis by simp
  next
    case (Node lrt r rrt)
    note rNode=this
    with lNode Node.prems show ?thesis by (simp split: option.splits)
  qed
qed
then obtain lbdt where bdt lt var = Some lbdt..
with Node.prems lt-dag no-in-lt show ?thesis
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
next
  assume no  $\notin$  set-of lt
  with no-in-lt-or-rt have no-in-rt: no  $\in$  set-of rt by simp
  from Node.prems p-po have rt-dag: Dag (high po) low high rt by simp
  from Node.hyps have hyp2:  $\bigwedge$  rbdt.  $\llbracket$ Dag (high po) low high rt; bdt rt var =
Some rbdt; Dag no low high not; no  $\in$  set-of rt $\rrbracket \implies \exists$  bdt2. bdt not var = Some
bdt2
    by simp
  from Node.prems have lbdt-def:  $\exists$  rbdt. bdt rt var = Some rbdt
  proof (cases rt)
    case Tip
    with Node.prems no-in-rt show ?thesis by (simp add: le-dag-def)
  next
    case (Node lrt l rrt)
    note rNode=this
    show ?thesis
    proof (cases lt)
      case Tip
      note lTip=this
      with rNode Node.prems show ?thesis by simp
    next
      case (Node llt r rlt)
      note lNode=this
      with rNode Node.prems show ?thesis by (simp split: option.splits)
    qed
  qed

```

```

qed
then obtain rbd1 where bdt1 rt var = Some rbd1..
with Node.prem1 rt-dag no-in-rt show ?thesis
  apply -
  apply (rule hyp2)
  apply assumption+
done
qed
qed
qed

lemma subbd1-ex:
 $\bigwedge$  bdt1.  $\llbracket$  (Node lst stp rst)  $\leq$  t; bdt t var = Some bdt1  $\rrbracket$ 
 $\implies$   $\exists$  bdt2. bdt (Node lst stp rst) var = Some bdt2
proof (induct t)
  case Tip
  then show ?case by (simp add: le-dag-def)
next
  case (Node lt p rt)
  note pNode=this
  show ?case
  proof (cases Node lst stp rst = Node lt p rt)
    case True
    with Node.prem1 show ?thesis by simp
  next
    assume Node lst stp rst  $\neq$  Node lt p rt
    with Node.prem1 have Node lst stp rst < Node lt p rt apply (simp add:
le-dag-def) apply auto done
    then have in-ltrt: Node lst stp rst  $\leq$  lt  $\vee$  Node lst stp rst  $\leq$  rt
      by (simp add: less-dag-Node)
    show ?thesis
    proof (cases Node lst stp rst  $\leq$  lt)
      case True
      note in-lt=this
      from Node.prem1 have lbd1-def:  $\exists$  lbd1. bdt lt var = Some lbd1
      proof (cases lt)
        case Tip
        with Node.prem1 in-lt show ?thesis by (simp add: le-dag-def)
      next
        case (Node llt l rrt)
        note lNode=this
        show ?thesis
        proof (cases rt)
          case Tip
          note rNode=this
          with lNode Node.prem1 show ?thesis by simp
        next
          case (Node lrt r rrt)
          note rNode=this

```



```

    with lNode Node.premis show ?thesis by (simp split: option.splits)
  qed
qed
then obtain lbd where bdt lt var = Some lbd..
with Node.premis in-lt show ?thesis
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
next
assume  $\neg$  Node lst stp rst  $\leq$  lt
with in-ltrt have in-rt: Node lst stp rst  $\leq$  rt by simp
  from Node.hyps have hyp2:  $\bigwedge$  rbd.  $\llbracket$ Node lst stp rst  $\leq$  rt; bdt rt var =
  Some rbd $\rrbracket \implies \exists$  bdt2. bdt (Node lst stp rst) var = Some bdt2
  by simp
  from Node.premis have rbd-def:  $\exists$  rbd. bdt rt var = Some rbd
  proof (cases rt)
    case Tip
      with Node.premis in-rt show ?thesis by (simp add: le-dag-def)
  next
    case (Node lrt l rrt)
      note rNode=this
      show ?thesis
      proof (cases lt)
        case Tip
          note lNode=this
          with rNode Node.premis show ?thesis by simp
      next
        case (Node lrt r rrt)
          note lNode=this
          with rNode Node.premis show ?thesis by (simp split: option.splits)
      qed
    qed
  then obtain rbd where bdt rt var = Some rbd..
  with Node.premis in-rt show ?thesis
    apply -
    apply (rule hyp2)
    apply assumption+
    done
  qed
qed
qed
qed

```

lemma *var-ordered-children*:

$\bigwedge p. \llbracket$ Dag p low high t ; ordered t var; no \in set-of t ;
 low no \neq Null; high no \neq Null \rrbracket
 \implies var (low no) $<$ var no \wedge var (high no) $<$ var no
proof (*induct t*)

```

    case Tip
  then show ?case by simp
next
case (Node lt po rt)
then have ppo: p=po by simp
show ?case
proof (cases no = po)
  case True
  note no-po=this
  from Node.prems have var (low po) < var po ∧ var (high po) < var po
  proof (cases lt)
    case Tip
    note ltTip=this
    with Node.prems no-po ppo show ?thesis by simp
  next
  case (Node llt l rlt)
  note lNode=this
  show ?thesis
  proof (cases rt)
    case Tip
    note rTip=this
    with Node.prems no-po ppo show ?thesis by simp
  next
  case (Node lrt r rrt)
  note rNode=this
  with Node.prems ppo no-po lNode show ?thesis by (simp del: Dag-Ref)
  qed
qed
with no-po show ?thesis by simp
next
assume no ≠ po
with Node.prems have no-in-ltrt: no ∈ set-of lt ∨ no ∈ set-of rt
  by simp
show ?thesis
proof (cases no ∈ set-of lt)
  case True
  note no-in-lt=this
  from Node.prems ppo have lt-dag: Dag (low po) low high lt
  by simp
  from Node.prems have ord-lt: ordered lt var
  apply –
  apply (drule children-ordered)
  apply simp
  done
  from no-in-lt lt-dag ord-lt Node.prems show ?thesis
  apply –
  apply (rule Node.hyps)
  apply assumption+
  done

```

```

next
  assume no  $\notin$  set-of lt
  with no-in-ltrt have no-in-rt: no  $\in$  set-of rt by simp
  from Node.premis ppo have rt-dag: Dag (high po) low high rt by simp
  from Node.hyps have hyp2:  $\llbracket$ Dag (high po) low high rt; ordered rt var; no  $\in$ 
set-of rt; low no  $\neq$  Null; high no  $\neq$  Null $\rrbracket$ 
     $\implies$  var (low no) < var no  $\wedge$  var (high no) < var no
    by simp
  from Node.premis have ord-rt: ordered rt var
  apply -
  apply (drule children-ordered)
  apply simp
  done
  from rt-dag ord-rt no-in-rt Node.premis show ?thesis
  apply -
  apply (rule hyp2)
  apply assumption+
  done
qed
qed
qed

```

lemma nort-null-comp:

```

assumes pret-dag: Dag p low high pret and
  prebdt-pret: bdt pret var = Some prebdt and
  nort-dag: Dag (repc no) (repb  $\times$  low) (repb  $\times$  high) nort and
  ord-pret: ordered pret var and
  wf-llb: wf-ll pret levellistb var and
  nbll: nb < length levellistb and
  repbc-nc:  $\forall$  nt. nt  $\notin$  set (levellistb ! nb)  $\longrightarrow$  repb nt = repc nt and
  xsnb-in-pret:  $\forall$  x  $\in$  set-of nort. var x < nb  $\wedge$  x  $\in$  set-of pret
shows  $\forall$  x  $\in$  set-of nort. ((repc  $\times$  low) x = (repb  $\times$  low) x  $\wedge$ 
(repc  $\times$  high) x = (repb  $\times$  high) x)

```

proof (rule ballI)

```

fix x
assume x-in-nort: x  $\in$  set-of nort
with nort-dag have xnN: x  $\neq$  Null
  apply -
  apply (rule set-of-nn [rule-format])
  apply auto
  done
from x-in-nort xsnb-in-pret have xsnb: var x < nb
  by simp
from x-in-nort xsnb-in-pret have x-in-pret: x  $\in$  set-of pret
  by blast
show (repc  $\times$  low) x = (repb  $\times$  low) x  $\wedge$  (repc  $\times$  high) x = (repb  $\times$  high) x
proof (cases (low x)  $\neq$  Null)
  case True
  with pret-dag prebdt-pret x-in-pret have highnN: (high x)  $\neq$  Null

```

```

    apply -
    apply (drule balanced-bdt)
    apply assumption+
    apply simp
    done
  from x-in-pret ord-pret highnN True have children-var-smaller: var (low x) <
var x  $\wedge$  var (high x) < var x
    apply -
    apply (rule var-ordered-children)
    apply (rule pret-dag)
    apply (rule ord-pret)
    apply (rule x-in-pret)
    apply (rule True)
    apply (rule highnN)
    done
  with xsnb have lowxsnb: var (low x) < nb
    by arith
  from children-var-smaller xsnb have highxsnb: var (high x) < nb
    by arith
  from x-in-pret xnN True pret-dag have lowxinpret: (low x)  $\in$  set-of pret
    apply -
    apply (drule subelem-set-of-low)
    apply assumption
    apply (thin-tac x  $\neq$  Null)
    apply assumption+
    done
  with wf-llb have low x  $\in$  set (levellistb ! (var (low x)))
    by (simp add: wf-ll-def)
  with wf-llb nbsll lowxsnb have low x  $\notin$  set (levellistb ! nb)
    apply -
    apply (rule-tac ?i=(var (low x)) and ?j=nb in no-in-one-ll)
    apply auto
    done
  with repbc-nc have repclow: repc (low x) = repb (low x)
    by auto
  from x-in-pret xnN highnN pret-dag have highxinpret: (high x)  $\in$  set-of pret
    apply -
    apply (drule subelem-set-of-high)
    apply assumption
    apply (thin-tac x  $\neq$  Null)
    apply assumption+
    done
  with wf-llb have high x  $\in$  set (levellistb ! (var (high x)))
    by (simp add: wf-ll-def)
  with wf-llb nbsll highxsnb have high x  $\notin$  set (levellistb ! nb)
    apply -
    apply (rule-tac ?i=(var (high x)) and ?j=nb in no-in-one-ll)
    apply auto
    done

```

```

with repcb-nc have repc: repc (high x) = repb (high x)
  by auto
with repc show ?thesis
  by (simp add: null-comp-def)
next
assume  $\neg \text{low } x \neq \text{Null}$ 
then have lowxNull: low x = Null by simp
with pret-dag x-in-pret prebdt-pret have highxNull: high x = Null
  apply –
  apply (drule balanced-bdt)
  apply simp
  apply simp
  apply simp
  done
from lowxNull have repc lowxNull: (repc  $\times$  low) x = Null
  by (simp add: null-comp-def)
from lowxNull have repblowNull: (repb  $\times$  low) x = Null
  by (simp add: null-comp-def)
with repc lowxNull have lowxrepcb: (repc  $\times$  low) x = (repb  $\times$  low) x
  by simp
from highxNull have repc highxNull: (repc  $\times$  high) x = Null
  by (simp add: null-comp-def)
from highxNull have (repb  $\times$  high) x = Null
  by (simp add: null-comp-def)
with repc highxNull have highxrepcb: (repc  $\times$  high) x = (repb  $\times$  high) x
  by simp
with lowxrepcb show ?thesis
  by simp
qed
qed

```

lemma *wf-ll-Nodes-pret*:

```

[[wf-ll pret levellista var; nb < length levellista; x  $\in$  Nodes nb levellista]]
 $\implies x \in \text{set-of } \text{pret} \wedge \text{var } x < \text{nb}$ 
apply (simp add: wf-ll-def Nodes-def)
apply (erule conjE)
apply (thin-tac  $\forall q. q \in \text{set-of } \text{pret} \longrightarrow q \in \text{set } (\text{levellista } ! \text{ var } q)$ )
apply (erule exE)
apply (elim conjE)
apply (erule-tac x=xa in allE)
apply (erule impE)
apply arith
apply (erule-tac x=x in ballE)
apply auto
done

```

lemma *bdt-Some-var1-One*:

```

 $\wedge x. [[ \text{bdt } t \text{ var} = \text{Some } x; \text{var } (\text{root } t) = 1 ]]$ 
 $\implies x = \text{One} \wedge t = (\text{Node } \text{Tip } (\text{root } t) \text{ Tip})$ 

```

```

proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt p rt)
  note tNode = this
  show ?case
  proof (cases lt)
    case Tip
    note ltTip=this
    show ?thesis
    proof (cases rt)
      case Tip
      note rtTip = this
      with ltTip Node.prems show ?thesis by auto
    next
    case (Node lrt r rrt)
    note rtNode=this
    with Node.prems ltTip show ?thesis by auto
  qed
next
  case (Node llt l rlt)
  note ltNode=this
  show ?thesis
  proof (cases rt)
    case Tip
    with ltNode Node.prems show ?thesis by auto
  next
  case (Node lrt r rrt)
  note rtNode=this
  with ltNode Node.prems show ?thesis by auto
  qed
qed
qed

```

lemma *bdt-Some-var0-Zero*:

$\bigwedge x. \llbracket \text{bdt } t \text{ var} = \text{Some } x; \text{var } (\text{root } t) = 0 \rrbracket$
 $\implies x = \text{Zero} \wedge t = (\text{Node } \text{Tip } (\text{root } t) \text{Tip})$

```

proof (induct t)
  case Tip
  then show ?case by simp
next
  case (Node lt p rt)
  note tNode = this
  show ?case
  proof (cases lt)
    case Tip
    note ltTip=this
    show ?thesis

```

```

proof (cases rt)
  case Tip
  note rtTip = this
  with ltTip Node.premis show ?thesis by auto
next
  case (Node lrt r rrt)
  note rtNode=this
  with Node.premis ltTip show ?thesis by auto
qed
next
  case (Node llt l rlt)
  note ltNode=this
  show ?thesis
  proof (cases rt)
    case Tip
    with ltNode Node.premis show ?thesis by auto
  next
    case (Node lrt r rrt)
    note rtNode=this
    with ltNode Node.premis show ?thesis by auto
  qed
qed
qed

```

lemma *reduced-children-parent*:
 $\llbracket \text{reduced } l; l = (\text{Node } llt \text{ lp } rlt); \text{reduced } r; r = (\text{Node } lrt \text{ rp } rrt);$
 $lp \neq rp \rrbracket$
 $\implies \text{reduced } (\text{Node } l \text{ p } r)$
by *simp*

lemma *Nodes-subset*: $\text{Nodes } i \text{ levellista} \subseteq \text{Nodes } (\text{Suc } i) \text{ levellista}$
apply (*simp add: Nodes-def*)
apply (*simp add: set-split*)
done

lemma *Nodes-levellist*:
 $\llbracket \text{wf-ll pret levellista var; nb} < \text{length levellista}; p \in \text{Nodes } nb \text{ levellista} \rrbracket$
 $\implies p \notin \text{set } (\text{levellista } ! \text{ nb})$
apply (*simp add: Nodes-def*)
apply (*erule exE*)
apply (*rule-tac i=x and j=nb in no-in-one-ll*)
apply *auto*
done

lemma *Nodes-var-pret*:
 $\llbracket \text{wf-ll pret levellista var; nb} < \text{length levellista}; p \in \text{Nodes } nb \text{ levellista} \rrbracket$
 $\implies \text{var } p < nb \wedge p \in \text{set-of pret}$
apply (*simp add: Nodes-def wf-ll-def*)

```

apply (erule conjE)
apply (thin-tac  $\forall q. q \in \text{set-of pret} \longrightarrow q \in \text{set (levellista ! var q)}$ )
apply (erule exE)
apply (erule-tac  $x=x$  in allE)
apply (erule impE)
apply arith
apply (erule-tac  $x=p$  in ballE)
apply arith
apply simp
done

```

lemma *Dags-root-in-Nodes*:

assumes *t-in-DagsSucnb*: $t \in \text{Dags (Nodes (Suc nb) levellista) low high}$

shows $\exists p. \text{Dag } p \text{ low high } t \wedge p \in \text{Nodes (Suc nb) levellista}$

proof –

from *t-in-DagsSucnb* **obtain** *p* **where** *t-dag*: $\text{Dag } p \text{ low high } t$ **and** *t-subset-Nodes*:
 $\text{set-of } t \subseteq \text{Nodes (Suc nb) levellista}$ **and** *t-nTip*: $t \neq \text{Tip}$

by (fastforce elim: *Dags.cases*)

from *t-dag t-nTip* **have** $p \neq \text{Null}$ **by** (cases *t*) auto

with *t-subset-Nodes t-dag* **have** $p \in \text{Nodes (Suc nb) levellista}$

by (cases *t*) auto

with *t-dag* **show** ?thesis

by auto

qed

lemma *subdag-dag*:

$\bigwedge p. \llbracket \text{Dag } p \text{ low high } t; st \leq t \rrbracket \implies \exists stp. \text{Dag } stp \text{ low high } st$

proof (induct *t*)

case *Tip*

then show ?case

by (simp add: less-dag-def le-dag-def)

next

case (Node *lt po rt*)

note *t-Node=this*

with *Node.prem*s **have** $p-po: p=po$

by simp

show ?case

proof (cases $st = \text{Node } lt \ po \ rt$)

case *True*

note *st-t=this*

with *Node.prem*s **show** ?thesis

by auto

next

assume *st-nt*: $st \neq \text{Node } lt \ po \ rt$

with *Node.prem*s *p-po* **have** *st-subdag-lt-rt*: $st \leq lt \vee st \leq rt$

by (auto simp add: le-dag-def less-dag-def)


```

    from Node.premis p-po obtain lp rp where lt-dag: Dag lp low high lt and
rt-dag: Dag rp low high rt
    by auto
  show ?thesis
  proof (cases st<=lt)
    case True
    note st-lt=this
    with lt-dag show ?thesis
    apply-
    apply (rule Node.hyps)
    apply auto
    done
  next
  assume  $\neg st \leq lt$ 
  with st-subdag-lt-rt have st-rt: st <= rt
  by simp
  from Node.hyps have rhyp:  $\llbracket \text{Dag } rp \text{ low high } rt; st \leq rt \rrbracket \implies \exists stp. \text{Dag } stp$ 
low high st
  by simp
  from st-rt rt-dag show ?thesis
  apply -
  apply (rule rhyp)
  apply auto
  done
  qed
  qed
  qed

```

lemma *Dags-subdags:*

assumes *t-in-Dags:* $t \in \text{Dags nodes low high}$ **and**

st-t: $st \leq t$ **and**

st-nTip: $st \neq \text{Tip}$

shows $st \in \text{Dags nodes low high}$

proof –

from *t-in-Dags* obtain p where t-dag: Dag p low high t and t-subset-Nodes:

set-of t \subseteq nodes and t-nTip: $t \neq \text{Tip}$

by (fastforce elim: Dags.cases)

from st-t have set-of st \subseteq set-of t

by (simp add: le-dag-set-of)

with t-subset-Nodes have st-subset-fnctNodes: set-of st \subseteq nodes

by blast

from st-t t-dag obtain stp where Dag stp low high st

apply –

apply (drule subdag-dag)

apply auto

done

with st-subset-fnctNodes st-nTip show ?thesis

apply –

apply (rule DagsI)

```

    apply auto
  done
qed

```

lemma *Dags-Nodes-cases*:

assumes *P-sym*: $\bigwedge t1\ t2. P\ t1\ t2\ var = P\ t2\ t1\ var$ **and**

dags-in-lower-levels:

$\bigwedge t1\ t2. \llbracket t1 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high;$
 $t2 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high \rrbracket$
 $\implies P\ t1\ t2\ var$ **and**

dags-in-mixed-levels:

$\bigwedge t1\ t2. \llbracket t1 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high;$
 $t2 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high;$
 $t2 \notin Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high \rrbracket$
 $\implies P\ t1\ t2\ var$ **and**

dags-in-high-level:

$\bigwedge t1\ t2. \llbracket t1 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high;$
 $t1 \notin Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high;$
 $t2 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high;$
 $t2 \notin Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high \rrbracket$
 $\implies P\ t1\ t2\ var$

shows $\forall t1\ t2. t1 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high \wedge$
 $t2 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high$
 $\longrightarrow P\ t1\ t2\ var$

proof (*intro allI impI , elim conjE*)

fix *t1 t2*

assume *t1-in-higher-levels*: $t1 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high$

assume *t2-in-higher-levels*: $t2 \in Dags\ (fnct\ '(Nodes\ (Suc\ n)\ levellista))\ low\ high$

show $P\ t1\ t2\ var$

proof (*cases* $t1 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high$)

case *True*

note *t1-in-ll = this*

show *?thesis*

proof (*cases* $t2 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high$)

case *True*

note *t2-in-ll=this*

with *t1-in-ll dags-in-lower-levels* **show** *?thesis*

by *simp*

next

assume *t2-notin-ll*: $t2 \notin Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high$

with *t1-in-ll t2-in-higher-levels dags-in-mixed-levels* **show** *?thesis*

by *simp*

qed

next

assume *t1-notin-ll*: $t1 \notin Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high$

show *?thesis*

proof (*cases* $t2 \in Dags\ (fnct\ '(Nodes\ n\ levellista))\ low\ high$)

case *True*

```

note t2-in-ll=this
with dags-in-mixed-levels t1-in-higher-levels t1-notin-ll P-sym show ?thesis
  by auto
next
  assume t2-notin-ll: t2 ∉ Dags (fnct ‘ Nodes n levellista) low high
  with t1-notin-ll t1-in-higher-levels t2-in-higher-levels dags-in-high-level show
?thesis
    by simp
  qed
qed
qed

```

```

lemma Null-notin-Nodes: [[Dag p low high t; nb <= length levellista; wf-ll t level-
lista var]] ⇒ Null ∉ Nodes nb levellista
  apply (simp add: Nodes-def wf-ll-def del: Dag-Ref)
  apply (rule allI)
  apply (rule impI)
  apply (elim conjE)
  apply (thin-tac ∀ q. P q for P)
  apply (erule-tac x=x in allE)
  apply (erule impE)
  apply simp
  apply (erule-tac x=Null in ballE)
  apply (erule conjE)
  apply (drule set-of-nn [rule-format])
  apply auto
done

```

```

lemma Nodes-in-pret: [[wf-ll t levellista var; nb <= length levellista]] ⇒ Nodes
nb levellista ⊆ set-of t
  apply –
  apply rule
  apply (simp add: wf-ll-def Nodes-def)
  apply (erule exE)
  apply (elim conjE)
  apply (thin-tac ∀ q. q ∈ set-of t → q ∈ set (levellista ! var q))
  apply (erule-tac x=xa in allE)
  apply (erule impE)
  apply simp
  apply (erule-tac x=x in ballE)
  apply auto
done

```

lemma *restrict-root-Node:*

```

[[t ∈ Dags (repc ‘Nodes (Suc nb) levellista) (repc ∝ low) (repc ∝ high); t ∉ Dags
(repc ‘Nodes nb levellista) (repc ∝ low) (repc ∝ high)];

```

```

    ordered t var;  $\forall$  no  $\in$  Nodes (Suc nb) levellista. var (repc no)  $\leq$  var no  $\wedge$  repc
(repc no) = repc no; wf-ll pret levellista var; nb < length levellista; repc 'Nodes (Suc
nb) levellista  $\subseteq$  Nodes (Suc nb) levellista]]
 $\implies \exists$  q. Dag (repc q) (repc  $\times$  low) (repc  $\times$  high) t  $\wedge$  q  $\in$  set (levellista ! nb)
proof (elim Dags.cases)
  fix p and ta :: dag
  assume t-notin-DagsNodesnb: t  $\notin$  Dags (repc 'Nodes nb levellista) (repc  $\times$  low)
(repc  $\times$  high)
  assume t-ta: t = ta
  assume ta-in-repc-NodesSucnb: set-of ta  $\subseteq$  repc 'Nodes (Suc nb) levellista
  assume ta-dag: Dag p (repc  $\times$  low) (repc  $\times$  high) ta
  assume ta-nTip: ta  $\neq$  Tip
  assume ord-t: ordered t var
  assume varrep-prop:  $\forall$  no  $\in$  Nodes (Suc nb) levellista. var (repc no)  $\leq$  var no
 $\wedge$  repc (repc no) = repc no
  assume wf-lla: wf-ll pret levellista var
  assume nbslla: nb < length levellista
  assume repcNodes-in-Nodes: repc 'Nodes (Suc nb) levellista  $\subseteq$  Nodes (Suc nb)
levellista
  from ta-nTip ta-dag have p-nNull: p  $\neq$  Null
    by auto
  with ta-nTip ta-dag obtain lt rt where ta-Node: ta = Node lt p rt
    by auto
  with ta-nTip ta-dag have p-in-ta: p  $\in$  set-of ta
    by auto
  with ta-in-repc-NodesSucnb have p-in-repcNodes-Sucnb: p  $\in$  repc 'Nodes (Suc
nb) levellista
    by auto
  show ?thesis
  proof (cases p  $\in$  repc '(set (levellista ! nb)))
    case True
    then obtain q where
      p-repc: p = repc q and
      a-in-llanb: q  $\in$  set (levellista ! nb)
    by auto
  with ta-dag t-ta show ?thesis
    apply -
    apply (rule-tac x=q in exI)
    apply simp
    done
  next
  assume p-notin-repc-llanb: p  $\notin$  repc 'set (levellista ! nb)
    with p-in-repcNodes-Sucnb have p-in-repc-Nodesnb: p  $\in$  repc 'Nodes nb
levellista
    apply -
    apply (erule imageE)
    apply rule
    apply (simp add: Nodes-def)
    apply (simp add: Nodes-def)

```

```

apply (erule exE conjE)
apply (case-tac xa=nb)
apply simp
apply (rule-tac x=xa in exI)
apply auto
done
have  $t \in Dags$  (repc 'Nodes nb levellista) (repc  $\times$  low) (repc  $\times$  high)
proof –
  have set-of  $t \subseteq$  repc 'Nodes nb levellista
  proof (rule)
    fix  $x :: ref$ 
    assume  $x$ -in- $t$ :  $x \in$  set-of  $t$ 
    with ord- $t$  have  $var\ x \leq var$  (root  $t$ )
    apply –
    apply (rule ordered-set-of)
    apply auto
    done
    with  $t$ -ta  $ta$ -Node have  $var\ x$ - $var\ p$ :  $var\ x \leq var\ p$ 
    by auto
    from  $p$ -in-repc-Nodesnb obtain  $k$  where  $ksnb$ :  $k < nb$  and  $p$ -in-repc-llak:
 $p \in$  repc '(set (levellista !  $k$ ))
    by (auto simp add: Nodes-def ImageE)
    then obtain  $q$  where  $p$ -repc $q$ :  $p = repc\ q$  and  $q$ -in-llak:  $q \in$  set (levellista
!  $k$ )
    by auto
    from  $q$ -in-llak  $wf$ -lla  $nbslla$   $ksnb$  have  $var\ q$ :  $var\ q = k$ 
    by (simp add: wf-ll-def)
    have Nodesnb-in-NodesSucnb: Nodes nb levellista  $\subseteq$  Nodes (Suc nb)
levellista
    by (rule Nodes-subset)
    from  $q$ -in-llak  $ksnb$  have  $q \in$  Nodes nb levellista
    by (auto simp add: Nodes-def)
    with  $varrep$ -prop Nodesnb-in-NodesSucnb have  $var$  (repc  $q$ )  $\leq var\ q$ 
    by auto
    with  $var\ q$   $ksnb$   $p$ -repc $q$  have  $var\ p < nb$ 
    by auto
    with  $var\ x$ - $var\ p$  have  $var\ x$ - $snb$ :  $var\ x < nb$ 
    by auto
    from  $x$ -in- $t$   $t$ -ta  $ta$ -in-repc-NodesSucnb obtain  $a$  where
 $x$ -repc $a$ :  $x = repc\ a$  and
 $a$ -in-NodesSucnb:  $a \in$  Nodes (Suc nb) levellista
    by auto
    with  $varrep$ -prop have  $rx$ - $x$ : repc  $x = x$ 
    by auto
    have  $x \in$  set-of pret
    proof –
    from  $wf$ -lla  $nbslla$  have Nodes (Suc nb) levellista  $\subseteq$  set-of pret
    apply –
    apply (rule Nodes-in-pret)

```

```

    apply auto
  done
with  $x$ -in- $t$   $t$ -ta  $ta$ -in- $repc$ - $Nodes$  $Suc$  $nb$   $repc$  $Nodes$ -in- $Nodes$  show ?thesis
  by auto
qed
with  $wf$ - $lla$  have  $x \in set (levellista ! (var x))$ 
  by (auto simp add:  $wf$ - $ll$ - $def$ )
with  $var$ - $x$ - $snb$  have  $x \in Nodes$   $nb$   $levellista$ 
  by (auto simp add:  $Nodes$ - $def$ )
with  $rx$ - $x$  show  $x \in repc$  ' $Nodes$   $nb$   $levellista$ '
  apply -
  apply rule
  apply (subgoal-tac  $x=rep$  $c$   $x$ )
  apply auto
  done
qed
with  $ta$ - $n$  $Tip$   $ta$ - $dag$   $t$ - $ta$  show ?thesis
  apply -
  apply (rule  $DagsI$ )
  apply auto
  done
qed
with  $t$ -notin- $Dags$  $Nodes$  $snb$  show ?thesis
  by auto
qed
qed

```

```

lemma same-bdt-var:  $\llbracket bdt (Node$   $lt1$   $p1$   $rt1) var = Some$   $bdt1$ ;  $bdt (Node$   $lt2$   $p2$ 
 $rt2) var = Some$   $bdt1 \rrbracket$ 
 $\implies var$   $p1 = var$   $p2$ 
proof (induct  $bdt1$ )
  case Zero
  then obtain  $var$ - $p1$ :  $var$   $p1 = 0$  and  $var$ - $p2$ :  $var$   $p2 = 0$ 
    by simp
  then show ?case
    by simp
next
  case One
  then obtain  $var$ - $p1$ :  $var$   $p1 = 1$  and  $var$ - $p2$ :  $var$   $p2 = 1$ 
    by simp
  then show ?case
    by simp
next
  case ( $Bdt$ - $Node$   $l$  $bdt$   $v$   $r$  $bdt$ )
  then obtain  $var$ - $p1$ :  $var$   $p1 = v$  and  $var$ - $p2$ :  $var$   $p2 = v$ 

```

```

    by simp
  then show ?case by simp
qed

```

```

lemma bdt-Some-Leaf-var-le-1:
  [[Dag p low high t; bdt t var = Some x; isLeaf-pt p low high]]
  ⇒ var p ≤ 1
proof (induct t)
  case Tip
  thus ?case by simp
next
  case (Node lt p rt)
  note tNode=this
  from Node.prem1 tNode show ?case
  apply (simp add: isLeaf-pt-def)
  apply (case-tac var p = 0)
  apply simp
  apply (case-tac var p = Suc 0)
  apply simp
  apply simp
  done
qed

```

```

lemma subnode-dag-cons:
  ∧ p. [[Dag p low high t; no ∈ set-of t]] ⇒ ∃ not. Dag no low high not
proof (induct t)
  case Tip
  thus ?case by simp
next
  case (Node lt q rt)
  with Node.prem1 have q-p: p = q
  by simp
  from Node.prem1 have lt-dag: Dag (low p) low high lt
  by auto
  from Node.prem2 have rt-dag: Dag (high p) low high rt
  by auto
  show ?case
  proof (cases no ∈ set-of lt)
  case True
  with Node.hyps lt-dag show ?thesis
  by simp
  next
  assume no-notin-lt: no ∉ set-of lt
  show ?thesis
  proof (cases no=p)
  case True
  with Node.prem1 q-p show ?thesis
  by auto
  next

```

```

    assume no-neq-p: no ≠ p
    with Node.premis no-notin-lt have no-in-rt: no ∈ set-of rt
      by simp
    with rt-dag Node.hyps show ?thesis
      by auto
  qed
qed
qed

```

```

lemma nodes-in-taken-in-takeSucn: no ∈ set (take n nodeslist) ⇒ no ∈ set (take
(Suc n) nodeslist)
proof -
  assume no-in-taken: no ∈ set (take n nodeslist)
  have set (take n nodeslist) ⊆ set (take (Suc n) nodeslist)
    apply -
    apply (rule set-take-subset-set-take)
    apply simp
  done
  with no-in-taken show ?thesis
    by blast
qed

```

```

lemma ind-in-higher-take: ∧n k. [n < k; n < length xs]
⇒ xs ! n ∈ set (take k xs)
apply (induct xs)
apply simp
apply simp
apply (case-tac n)
apply simp
apply (case-tac k)
apply simp
apply simp
apply simp
apply (case-tac k)
apply simp
apply simp
done

```


lemma *take-length-set*: $\bigwedge n. n = \text{length } xs \implies \text{set } (\text{take } n \text{ } xs) = \text{set } xs$
apply (*induct xs*)
apply (*auto simp add: take-Cons split: nat.splits*)
done

lemma *repNodes-eq-ext-rep*: $\llbracket \text{low } no \neq \text{nodeslist! } n; \text{ high } no \neq \text{nodeslist! } n; \text{ low } sn \neq \text{nodeslist! } n; \text{ high } sn \neq \text{nodeslist! } n \rrbracket$
 $\implies \text{repNodes-eq } sn \text{ no low high } \text{repa} = \text{repNodes-eq } sn \text{ no low high } (\text{repa}(\text{nodeslist! } n := \text{repa } (\text{low } (\text{nodeslist! } n))))$
by (*simp add: repNodes-eq-def null-comp-def*)

lemma *filter-not-empty*: $\llbracket x \in \text{set } xs; P \ x \rrbracket \implies \text{filter } P \ xs \neq []$
by (*induct xs*) *auto*

lemma $x \in \text{set } (\text{filter } P \ xs) \implies P \ x$
by *auto*

lemma *hd-filter-in-list*: $\text{filter } P \ xs \neq [] \implies \text{hd } (\text{filter } P \ xs) \in \text{set } xs$
by (*induct xs*) *auto*

lemma *hd-filter-in-filter*: $\text{filter } P \ xs \neq [] \implies \text{hd } (\text{filter } P \ xs) \in \text{set } (\text{filter } P \ xs)$
by (*induct xs*) *auto*

lemma *hd-filter-prop*:
assumes *non-empty*: $\text{filter } P \ xs \neq []$
shows $P \ (\text{hd } (\text{filter } P \ xs))$
proof –
from *non-empty* **have** $\text{hd } (\text{filter } P \ xs) \in \text{set } (\text{filter } P \ xs)$
by (*rule hd-filter-in-filter*)
thus *?thesis*
by *auto*
qed

lemma *index-elem*: $x \in \text{set } xs \implies \exists i < \text{length } xs. x = xs \ ! \ i$
apply (*induct xs*)
apply *simp*
apply (*case-tac x=a*)
apply *auto*
done

lemma *filter-hd-P-rep-indep*:
 $\llbracket \forall x. P \ x \ x; \forall a \ b. P \ x \ a \ \longrightarrow \ P \ a \ b \ \longrightarrow \ P \ x \ b; \text{filter } (P \ x) \ xs \neq [] \rrbracket \implies$
 $\text{hd } (\text{filter } (P \ (\text{hd } (\text{filter } (P \ x) \ xs))) \ xs) = \text{hd } (\text{filter } (P \ x) \ xs)$
apply (*induct xs*)
apply *simp*
apply (*case-tac P x a*)
using $[[\text{simp-depth-limit}=2]]$

```

apply (simp)
apply clarsimp
apply (fastforce dest: hd-filter-prop)
done

```

```

lemma take-Suc-not-last:
 $\bigwedge n. \llbracket x \in \text{set } (\text{take } (\text{Suc } n) \text{ } xs); x \neq xs!n; n < \text{length } xs \rrbracket \implies x \in \text{set } (\text{take } n \text{ } xs)$ 
apply (induct xs)
apply simp
apply (case-tac n)
apply simp
using [simp-depth-limit=2]
apply fastforce
done

```

```

lemma P-eq-list-filter:  $\forall x \in \text{set } xs. P \ x = Q \ x \implies \text{filter } P \ xs = \text{filter } Q \ xs$ 
apply (induct xs)
apply auto
done

```

```

lemma hd-filter-take-more:  $\bigwedge n \ m. \llbracket \text{filter } P \ (\text{take } n \text{ } xs) \neq []; n \leq m \rrbracket \implies$ 
 $\text{hd } (\text{filter } P \ (\text{take } n \text{ } xs)) = \text{hd } (\text{filter } P \ (\text{take } m \text{ } xs))$ 
apply (induct xs)
apply simp
apply (case-tac n)
apply simp
apply (case-tac m)
apply simp
apply clarsimp
done

```

end

4 Definitions of Procedures

```

theory ProcedureSpecs
imports General Simpl.Vcg
begin

```

```

record globals =
  var-' :: ref  $\Rightarrow$  nat
  low-' :: ref  $\Rightarrow$  ref
  high-' :: ref  $\Rightarrow$  ref
  rep-' :: ref  $\Rightarrow$  ref
  mark-' :: ref  $\Rightarrow$  bool
  next-' :: ref  $\Rightarrow$  ref

```

```

record 'g bdd-state = 'g state +
  varval-' :: bool list
  p-' :: ref
  R-' :: bool
  levellist-' :: ref list
  nodeslist-' :: ref
  node-': ref
  m-' :: bool
  n-' :: nat

```

procedures

```

Eval (p, varval | R) =
  IF ('p → 'var = 0) THEN 'R ::= False
  ELSE IF ('p → 'var = 1) THEN 'R ::= True
  ELSE IF ('varval ! ('p → 'var)) THEN CALL Eval ('p → 'high, 'varval, 'R)
  ELSE CALL Eval ('p → 'low, 'varval, 'R)
  FI
  FI
  FI

```

procedures

```

Levellist (p, m, levellist | levellist) =
  IF ('p ≠ Null)
  THEN
    IF ('p → 'mark ≠ 'm)
    THEN
      'levellist ::= CALL Levellist ('p → 'low, 'm, 'levellist);;
      'levellist ::= CALL Levellist ('p → 'high, 'm, 'levellist);;
      'p → 'next ::= 'levellist ! ('p → 'var);;
      'levellist ! ('p → 'var) ::= 'p;;
      'p → 'mark ::= 'm
    FI
  FI

```

procedures

```

ShareRep (nodeslist, p) =
  IF (isLeaf-pt 'p 'low 'high)
  THEN 'p → 'rep ::= 'nodeslist
  ELSE
    WHILE ('nodeslist ≠ Null) DO

```

```

    IF (repNodes-eq 'nodeslist 'p 'low 'high 'rep)
    THEN 'p→'rep ::= 'nodeslist;; 'nodeslist ::= Null
    ELSE 'nodeslist ::= 'nodeslist→'next
    FI
  OD
FI

```

procedures

```

ShareReduceRepList (nodeslist | ) =
'node ::= 'nodeslist;;
WHILE ('node ≠ Null) DO
  IF (¬ isLeaf-pt 'node 'low 'high ∧
      'node → 'low → 'rep = 'node → 'high → 'rep )
  THEN 'node → 'rep ::= 'node → 'low → 'rep
  ELSE CALL ShareRep ('nodeslist , 'node )
  FI;;
  'node ::= 'node → 'next
OD

```

procedures

```

Repoint (p|p) =
IF ( 'p ≠ Null )
THEN
  'p ::= 'p → 'rep;;
  IF ( 'p ≠ Null )
  THEN 'p → 'low ::= CALL Repoint ('p → 'low);;
        'p → 'high ::= CALL Repoint ('p → 'high)
  FI
FI

```

procedures

```

Normalize (p|p) =
'levellist ::= replicate ('p→'var +1) Null;;
'levellist ::= CALL Levellist ('p, (¬ 'p→'mark) , 'levellist);;
('n ::=0;;
WHILE ('n < length 'levellist) DO
  CALL ShareReduceRepList('levellist ! 'n);;
  'n ::= 'n + 1
OD);;
'p ::= CALL Repoint ('p)

```

end

5 Proof of Procedure Eval

theory *EvalProof* **imports** *ProcedureSpecs* **begin**

lemma (**in** *Eval-impl*) *Eval-modifies*:
 shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC Eval } (\prime p, \prime varval, \prime R)$
 $\{t. t \text{ may-not-modify-globals } \sigma\}$
 apply (*hoare-rule HoarePartial.ProcRec1*)
 apply (*vcg spec=modifies*)
 done

lemma (**in** *Eval-impl*) *Eval-spec*:
 shows $\forall \sigma t \text{ bdt1}. \Gamma \vdash$
 $\{\sigma. \text{Dag } \prime p \text{ 'low } \prime \text{high } t \wedge \text{bdt } t \text{ 'var} = \text{Some } \text{bdt1}\}$
 $\prime R := \text{PROC Eval}(\prime p, \prime varval)$
 $\{\prime R = \text{eval } \text{bdt1 } \sigma \text{varval}\}$
 apply (*hoare-rule HoarePartial.ProcRec1*)
 apply *vcg*
 apply *clarsimp*
 apply *safe*
 apply (*case-tac bdt1*)
 apply *simp*
 apply *fastforce*
 apply *fastforce*
 apply *simp*
 apply (*case-tac bdt1*)
 apply *fastforce*
 apply *fastforce*
 apply *fastforce*
 apply (*case-tac bdt1*)
 apply *fastforce*
 apply *fastforce*
 apply *fastforce*
 apply (*case-tac bdt1*)
 apply *fastforce*
 apply *fastforce*
 done

end

6 Proof of Procedure Levellist

theory *LevellistProof* **imports** *ProcedureSpecs* *Simpl.HeapList* **begin**

hide-const (**open**) *DistinctTreeProver.set-of tree.Node tree.Tip*

lemma (in *Levellist-impl*) *Levellist-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ 'levellist} ::= \text{PROC Levellist } ('p, 'm, 'levellist)$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{mark}, \text{next}]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma *all-stop-cong*: $(\forall x. P x) = (\forall x. P x)$
by *simp*

lemma *Dag-RefD*:
 $\llbracket \text{Dag } p \ l \ r \ t; p \neq \text{Null} \rrbracket \implies$
 $\exists lt \ rt. t = \text{Node } lt \ p \ rt \wedge \text{Dag } (l \ p) \ l \ r \ lt \wedge \text{Dag } (r \ p) \ l \ r \ rt$
by *simp*

lemma *Dag-unique-ex-conjI*:
 $\llbracket \text{Dag } p \ l \ r \ t; P \ t \rrbracket \implies (\exists t. \text{Dag } p \ l \ r \ t \wedge P \ t)$
by *simp*

lemma *dag-Null* [*simp*]: $\text{dag } \text{Null} \ l \ r = \text{Tip}$
by (*simp add: dag-def*)

definition *first*:: $\text{ref } list \Rightarrow \text{ref}$ **where**
 $\text{first } ps = (\text{case } ps \text{ of } [] \Rightarrow \text{Null} \mid (p \# rs) \Rightarrow p)$

lemma *first-simps* [*simp*]:
 $\text{first } [] = \text{Null}$
 $\text{first } (r \# rs) = r$
by (*simp-all add: first-def*)

definition *Levellist*:: $\text{ref } list \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref } list \ list) \Rightarrow \text{bool}$ **where**
 $\text{Levellist } hds \ next \ ll \longleftrightarrow (\text{map } \text{first } ll = hds) \wedge$
 $(\forall i < \text{length } hds. \text{List } (hds ! i) \ next \ (ll ! i))$

lemma *Levellist-unique*:
assumes $ll: \text{Levellist } hds \ next \ ll$
assumes $ll': \text{Levellist } hds \ next \ ll'$
shows $ll = ll'$
proof –
from ll **have** $\text{length } ll = \text{length } hds$
by (*clarsimp simp add: Levellist-def*)
moreover
from ll' **have** $\text{length } ll' = \text{length } hds$
by (*clarsimp simp add: Levellist-def*)

```

ultimately have leq: length ll = length ll' by simp
show ?thesis
proof (rule nth-equalityI [OF leq, rule-format])
  fix i
  assume i < length ll
  with ll ll'
  show ll!i = ll'!i
    apply (clarsimp simp add: Levellist-def)
    apply (erule-tac x=i in allE)
    apply (erule-tac x=i in allE)
    apply simp
    by (erule List-unique)
qed
qed

lemma Levellist-unique-ex-conj-simp [simp]:
Levellist hds next ll  $\implies$  ( $\exists ll. \text{Levellist hds next } ll \wedge P ll$ ) = P ll
by (auto dest: Levellist-unique)

```

```

lemma in-set-concat-idx:
  x  $\in$  set (concat xss)  $\implies$   $\exists i < \text{length } xss. x \in \text{set } (xss!i)$ 
apply (induct xss)
apply simp
apply clarsimp
apply (erule disjE)
apply (rule-tac x=0 in exI)
apply simp
apply auto
done

```

```

definition wf-levellist :: dag  $\Rightarrow$  ref list list  $\Rightarrow$  ref list list  $\Rightarrow$ 
  (ref  $\Rightarrow$  nat)  $\Rightarrow$  bool where
wf-levellist t levellist-old levellist-new var =
(case t of Tip  $\Rightarrow$  levellist-old = levellist-new
| (Node lt p rt)  $\Rightarrow$ 
  ( $\forall q. q \in \text{set-of } t \longrightarrow q \in \text{set } (\text{levellist-new } ! (\text{var } q))$ )  $\wedge$ 
  ( $\forall i \leq \text{var } p. (\exists \text{prx}. (\text{levellist-new } ! i) = \text{prx}@(\text{levellist-old } ! i)
    \wedge (\forall \text{pt} \in \text{set } \text{prx}. \text{pt} \in \text{set-of } t \wedge \text{var } \text{pt} = i))$ )  $\wedge$ 
  ( $\forall i. (\text{var } p) < i \longrightarrow (\text{levellist-new } ! i) = (\text{levellist-old } ! i)$ )  $\wedge$ 
  (length levellist-new = length levellist-old))

```

```

lemma wf-levellist-subset:
  assumes wf-ll: wf-levellist t ll ll' var
  shows set (concat ll')  $\subseteq$  set (concat ll)  $\cup$  set-of t
proof (cases t)
  case Tip with wf-ll show ?thesis by (simp add: wf-levellist-def)
next
  case (Node lt p rt)

```

```

show ?thesis
proof -
{
  fix n
  assume n ∈ set (concat ll')
  from in-set-concat-idx [OF this]
  obtain i where i-bound: i < length ll' and n-in: n ∈ set (ll' ! i)
  by blast
  have n ∈ set (concat ll) ∪ set-of t
  proof (cases i ≤ var p)
  case True
  with wf-ll obtain prx where
    ll'-ll: ll' ! i = prx @ ll' ! i and
    prx: ∀ pt ∈ set prx. pt ∈ set-of t and
    leq: length ll' = length ll
  apply (clarsimp simp add: wf-levellist-def Node)
  apply (erule-tac x=i in allE)
  apply clarsimp
  done
  show ?thesis
  proof (cases n ∈ set prx)
  case True
  with prx have n ∈ set-of t
  by simp
  thus ?thesis by simp
  next
  case False
  with n-in ll'-ll
  have n ∈ set (ll' ! i)
  by simp
  with i-bound leq
  have n ∈ set (concat ll)
  by auto
  thus ?thesis by simp
  qed
  next
  case False
  with wf-ll obtain ll'i = ll'i length ll' = length ll
  by (auto simp add: wf-levellist-def Node)
  with n-in i-bound
  have n ∈ set (concat ll)
  by auto
  thus ?thesis by simp
  qed
}
thus ?thesis by auto
qed
qed

```


lemma *Levellist-ext-to-all*: $((\exists ll. \text{Levellist hds next } ll \wedge P ll) \longrightarrow Q)$
 $=$
 $(\forall ll. \text{Levellist hds next } ll \wedge P ll \longrightarrow Q)$
apply *blast*
done

lemma *Levellist-length*: $\text{Levellist hds } p ll \implies \text{length } ll = \text{length hds}$
by $(\text{auto simp add: Levellist-def})$

lemma *map-update*:
 $\bigwedge i. i < \text{length } xss \implies \text{map } f (xss[i := xs]) = (\text{map } f xss) [i := f xs]$
apply *induct xss*
apply *simp*
apply *case-tac i*
apply *simp*
apply *simp*
done

lemma $(\text{in Levellist-impl})$ *Levellist-spec-total'*:
shows $\forall ll \sigma t. \Gamma, \Theta \vdash_t$
 $\{\sigma. \text{Dag } 'p \text{ 'low 'high } t \wedge ('p \neq \text{Null} \longrightarrow ('p \rightarrow 'var) < \text{length } 'levellist) \wedge$
 $\text{ordered } t \text{ 'var} \wedge \text{Levellist } 'levellist \text{ 'next } ll \wedge$
 $(\forall n \in \text{set-of } t.$
 $\text{if } 'mark \ n = 'm$
 $\text{then } n \in \text{set } (ll \ ! \ 'var \ n) \wedge$
 $(\forall nt \ p. \text{Dag } n \text{ 'low 'high } nt \wedge p \in \text{set-of } nt$
 $\longrightarrow 'mark \ p = 'm)$
 $\text{else } n \notin \text{set } (\text{concat } ll))\}$
 $'levellist := \text{PROC Levellist } ('p, 'm, 'levellist)$
 $\{\exists ll'. \text{Levellist } 'levellist \text{ 'next } ll' \wedge \text{wf-levellist } t \ ll \ ll' \sigma \text{var} \wedge$
 $\text{wf-marking } t \ \sigma \text{mark } 'mark \ \sigma m \wedge$
 $(\forall p. p \notin \text{set-of } t \longrightarrow \sigma \text{next } p = 'next \ p)$
 $\}$
apply $(\text{hoare-rule HoareTotal.ProcRec1}$
 $\ [\text{where } r = \text{measure } (\lambda(s,p). \text{size } (\text{dag } s_p \text{ }^s\text{low }^s\text{high}))])$
apply *vcg*
apply *rule conjI*
apply *clarify*
apply *rule conjI*
apply *clarify*
apply $(\text{clarsimp simp del: BinDag.set-of.simps split del: if-split})$
defer
apply *rule impI*

```

apply (clarsimp simp del: BinDag.set-of.simps split del: if-split)
defer
apply (clarsimp simp add: wf-levellist-def wf-marking-def)
apply (simp only: Levellist-ext-to-all )
proof –
  fix ll var low high mark next nexta p levellist m lt rt
  assume pnN: p ≠ Null
  assume mark-p: mark p = (¬ m)
  assume lt: Dag (low p) low high lt
  assume rt: Dag (high p) low high rt
  from pnN lt rt have Dag-p: Dag p low high (Node lt p rt) by simp
  from Dag-p rt
  have size-rt-dec: size (dag (high p) low high) < size (dag p low high)
    by (simp only: Dag-dag) simp
  from Dag-p lt
  have size-lt-dec: size (dag (low p) low high) < size (dag p low high)
    by (simp only: Dag-dag) simp
  assume ll: Levellist levellist next ll

  assume marked-child-ll:
    ∀ n ∈ set-of (Node lt p rt).
      if mark n = m
      then n ∈ set (ll ! var n) ∧
        (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → mark p = m)
      else n ∉ set (concat ll)
  with mark-p have p-notin-ll: p ∉ set (concat ll)
    by auto
  assume varsl': var p < length levellist
  with ll have varsl: var p < length ll
    by (simp add: Levellist-length)
  assume orderedt: ordered (Node lt p rt) var
  show (low p ≠ Null → var (low p) < length levellist) ∧
    ordered lt var ∧
    (∀ n ∈ set-of lt.
      if mark n = m
      then n ∈ set (ll ! var n) ∧
        (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → mark p = m)
      else n ∉ set (concat ll)) ∧
    size (dag (low p) low high) < size (dag p low high) ∧
    (∀ marka nexta levellist lla.
      Levellist levellist nexta lla ∧
      wf-levellist lt ll lla var ∧ wf-marking lt mark marka m ∧
      (∀ p. p ∉ set-of lt → next p = nexta p) →
      (high p ≠ Null → var (high p) < length levellist) ∧
      ordered rt var ∧
      (∃ lla. Levellist levellist nexta lla ∧
        (∀ n ∈ set-of rt.
          if marka n = m
          then n ∈ set (lla ! var n) ∧

```

$$\begin{aligned}
& (\forall nt\ p. \text{Dag } n\ \text{low } high\ nt \wedge p \in \text{set-of } nt \longrightarrow \\
& \quad \text{marka } p = m) \\
& \text{else } n \notin \text{set } (\text{concat } lla) \wedge \\
& \text{size } (\text{dag } (high\ p)\ \text{low } high) < \text{size } (\text{dag } p\ \text{low } high) \wedge \\
& (\forall \text{markb } \text{nextb } \text{levellist } llb. \\
& \quad \text{Levellist } \text{levellist } \text{nextb } llb \wedge \\
& \quad \text{wf-levellist } rt\ lla\ llb\ \text{var} \wedge \\
& \quad \text{wf-marking } rt\ \text{marka } \text{markb } m \wedge \\
& (\forall p. p \notin \text{set-of } rt \longrightarrow \text{nexta } p = \text{nextb } p) \longrightarrow \\
& (\exists ll'. \text{Levellist } (\text{levellist}[var\ p := p]) \\
& \quad (\text{nextb}(p := \text{levellist } !\ \text{var } p))\ ll' \wedge \\
& \quad \text{wf-levellist } (\text{Node } lt\ p\ rt)\ ll\ ll'\ \text{var} \wedge \\
& \quad \text{wf-marking } (\text{Node } lt\ p\ rt)\ \text{mark } (\text{markb}(p := m))\ m \wedge \\
& (\forall pa. pa \notin \text{set-of } (\text{Node } lt\ p\ rt) \longrightarrow \\
& \quad \text{next } pa = \\
& \quad (\text{if } pa = p \text{ then } \text{levellist } !\ \text{var } p \\
& \quad \text{else } \text{nextb } pa))))))
\end{aligned}$$

proof (cases lt)

case Tip

note lt-Tip = this

show ?thesis

proof (cases rt)

case Tip

show ?thesis

using size-rt-dec Tip lt-Tip Tip lt rt

apply clarsimp

subgoal premises prems **for** marka nexta levellista lla markb nextb levellistb

llb

proof –

have lla: Levellist levellista nexta lla **by** fact

have llb: Levellist levellistb nextb llb **by** fact

have wfl-lt: wf-levellist Tip ll lla var

wf-marking Tip mark marka m **by** fact+

then have ll-lla: ll = lla

by (simp add: wf-levellist-def)

moreover

with wfl-lt lt-Tip lt **have** marka = mark

by (simp add: wf-marking-def)

moreover

have wfl-rt: wf-levellist Tip lla llb var

wf-marking Tip marka markb m **by** fact+

then have lla-llb: lla = llb

by (simp add: wf-levellist-def)

moreover

with wfl-rt Tip rt **have** markb = marka

by (simp add: wf-marking-def)

moreover

```

from varsll llb ll-lla lla-llb
obtain var p < length levellistb var p < length llb
  by (simp add: Levellist-length)
with llb pnN
have llc: Levellist (levellistb[var p := p]) (nextb(p := levellistb ! var p))
  (llb[var p := p # llb ! var p])
  apply (clarsimp simp add: Levellist-def map-update)
  apply (erule-tac x=i in allE)
  apply clarsimp
  apply (subgoal-tac p ∉ set (llb ! i) )
  prefer 2
  using p-notin-ll ll-lla lla-llb
  apply simp
  apply (case-tac i=var p)
  apply simp
  apply simp
  done
ultimately
show ?thesis
  using lt-Tip Tip varsll
  apply (clarsimp simp add: wf-levellist-def wf-marking-def)
proof –
  fix i
  assume varsllb: var p < length llb
  assume i ≤ var p
  show ∃ prx. llb[var p := p # llb ! var p]!i = prx @ llb!i ∧
    (∀ pt ∈ set prx. pt = p ∧ var pt = i)
  proof (cases i = var p)
    case True
      with pnN lt rt varsllb lt-Tip Tip show ?thesis
      apply –
      apply (rule-tac x=[p] in exI)
      apply (simp add: subdag-eq-def)
      done
    next
      assume i ≠ var p
      with varsllb show ?thesis
      apply –
      apply (rule-tac x=[] in exI)
      apply (simp add: subdag-eq-def)
      done
  qed
qed
qed
done
next
case (Node dag1 a dag2)
have rt-node: rt = Node dag1 a dag2 by fact
with rt have high-p: high p = a

```

```

  by simp
have s:  $\bigwedge nexta. (\forall p. next\ p = nexta\ p) = (next = nexta)$ 
  by auto
show ?thesis
using size-rt-dec size-lt-dec rt-node lt-Tip Tip lt rt
apply (clarsimp simp del: set-of-Node split del: if-split simp add: s)
subgoal premises prems for marka levellista lla
proof -
  have lla: Levellist levellista next lla by fact
  have wfl-lt: wf-levellist Tip ll lla var
    wf-marking Tip mark marka m by fact+
  from this have ll-lla: ll = lla
    by (simp add: wf-levellist-def)
  moreover
  from wfl-lt lt-Tip lt have marklrec: marka = mark
    by (simp add: wf-marking-def)
  from orderedt varsl ll lla ll-lla rt-node lt-Tip high-p
  have var-highp-bound: var (high p) < length levellista
    by (auto simp add: Levellist-length)
  from orderedt high-p rt-node lt-Tip
  have ordered-rt: ordered (Node dag1 (high p) dag2) var
    by simp
  from high-p marklrec marked-child-ll lt rt lt-Tip rt-node ll-lla
  have mark-rt:  $(\forall n \in \text{set-of } (Node\ dag1\ (high\ p)\ dag2).$ 
    if marka n = m
    then  $n \in \text{set } (lla\ !\ var\ n) \wedge$ 
       $(\forall nt\ p. Dag\ n\ low\ high\ nt \wedge p \in \text{set-of } nt \longrightarrow marka\ p = m)$ 
    else  $n \notin \text{set } (concat\ lla)$ 
  apply (simp only: BinDag.set-of.simps)
  apply clarify
  apply (drule-tac x=n in bspec)
  apply blast
  apply assumption
  done
show ?thesis
  apply (rule conjI)
  apply (rule var-highp-bound)
  apply (rule conjI)
  apply (rule ordered-rt)
  apply (rule conjI)
  apply (rule conjI)
  apply (rule mark-rt)
  apply clarify
  apply clarsimp
subgoal premises prems for markb nextb levellistb llb
proof -
  have llb: Levellist levellistb nextb llb by fact
  have wfl-rt: wf-levellist (Node dag1 (high p) dag2) lla llb var by fact
  have wfmarking-rt: wf-marking (Node dag1 (high p) dag2) marka markb
m by fact

```

```

from wfl-rt varsll llb ll-lla
obtain var-p-bounds: var p < length levellistb var p < length llb
  by (simp add: Levellist-length wf-levellist-def)
with p-notin-ll ll-lla wfl-rt
have p-notin-llb:  $\forall i < \text{length } llb. p \notin \text{set } (llb ! i)$ 
  apply –
  apply (intro allI impI)
  apply (clarsimp simp add: wf-levellist-def)
  apply (case-tac i ≤ var (high p))
  apply (drule-tac x=i in spec)
  using orderedt rt-node lt-Tip high-p
  apply clarsimp
  apply (drule-tac x=i in spec)
  apply (drule-tac x=i in spec)
  apply clarsimp
  done
with llb pnN var-p-bounds
have llc: Levellist (levellistb[var p := p])
  (nextb(p := levellistb ! var p))
  (llb[var p := p # llb ! var p])
  apply (clarsimp simp add: Levellist-def map-update)
  apply (erule-tac x=i in allE)
  apply (erule-tac x=i in allE)
  apply clarsimp
  apply (case-tac i=var p)
  apply simp
  apply simp
  done
then show ?thesis
  apply simp
  using wfl-rt wfmarking-rt
  lt-Tip rt-node varsll orderedt lt rt pnN ll-lla marklrec
  apply (clarsimp simp add: wf-levellist-def wf-marking-def)
  apply (intro conjI)
  apply (rule allI)
  apply (rule conjI)
  apply (erule-tac x=q in allE)
  apply (case-tac var p = var q)
  apply fastforce
  apply fastforce
  apply (case-tac var p = var q)
  apply hypsubst-thin
  apply fastforce
  apply fastforce
  apply (rule allI)
  apply (rotate-tac 4)
  apply (erule-tac x=i in allE)
  apply (case-tac i=var p)
  apply simp

```

```

    apply (case-tac var (high p) < i)
    apply simp
    apply simp
    apply (erule exE)
    apply (rule-tac x=prx in exI)
    apply (intro conjI)
    apply simp
    apply clarify
    apply (rotate-tac 15)
    apply (erule-tac x=pt in ballE)
    apply fastforce
    apply fastforce
    done
  qed
done
qed
done
qed
next
case (Node llt l rlt)
have lt-Node: lt = Node llt l rlt by fact
from orderedt lt varsll' lt-Node
obtain ordered-lt:
  ordered lt var (low p ≠ Null → var (low p) < length levellist)
  by (cases rt) auto
from lt lt-Node marked-child-ll
have mark-lt: ∀ n ∈ set-of lt.
  if mark n = m
  then n ∈ set (ll ! var n) ∧
    (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → mark p = m)
  else n ∉ set (concat ll)
  apply (simp only: BinDag.set-of.simps)
  apply clarify
  apply (drule-tac x=n in bspec)
  apply blast
  apply assumption
done
show ?thesis
  apply (intro conjI ordered-lt mark-lt size-lt-dec)
  apply (clarify)
  apply (simp add: size-rt-dec split del: if-split)
  apply (simp only: Levellist-ext-to-all)
subgoal premises prems for marka nexta levellista lla
proof -
  have lla: Levellist levellista nexta lla by fact
  have wfl-lt: wf-levellist lt ll lla var by fact
  have wfmarking-lt: wf-marking lt mark marka m by fact
  from wfl-lt lt-Node
  have lla-eq-ll: length lla = length ll

```

```

  by (simp add: wf-levellist-def)
with ll lla have lla-eq-ll': length levellista = length levellist
  by (simp add: Levellist-length)
with orderedt rt lt-Node lt varsl'
obtain ordered-rt:
  ordered rt var (high p ≠ Null → var (high p) < length levellista)
  by (cases rt) auto
from wfl-ll lt-Node
have nodes-in-lla: ∀ q. q ∈ set-of lt → q ∈ set (lla ! (q→var))
  by (simp add: wf-levellist-def)
from wfl-ll lt-Node lt
have lla-st: (∀ i ≤ (low p)→var.
  (∃ prx. (lla ! i) = prx@(ll ! i) ∧
    (∀ pt ∈ set prx. pt ∈ set-of lt ∧ pt→var = i)))
  by (simp add: wf-levellist-def)
from wfl-ll lt-Node lt
have lla-nc: ∀ i. ((low p)→var) < i → (lla ! i) = (ll ! i)
  by (simp add: wf-levellist-def)
from wfmarking-ll lt-Node lt
have mot-nc: ∀ n. n ∉ set-of lt → mark n = marka n
  by (simp add: wf-marking-def)
from wfmarking-ll lt-Node lt
have mit-marked: ∀ n. n ∈ set-of lt → marka n = m
  by (simp add: wf-marking-def)
from marked-child-ll nodes-in-lla mot-nc mit-marked lla-st
have mark-rt: ∀ n ∈ set-of rt.
  if marka n = m
  then n ∈ set (lla ! var n) ∧
    (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → marka p = m)
  else n ∉ set (concat lla)
  apply –
  apply (rule ballI)
  apply (drule-tac x=n in bspec)
  apply (simp)
proof –
  fix n
  assume nodes-in-lla: ∀ q. q ∈ set-of lt → q ∈ set (lla ! var q)
  assume mot-nc: ∀ n. n ∉ set-of lt → mark n = marka n
  assume mit-marked: ∀ n. n ∈ set-of lt → marka n = m
  assume marked-child-ll: if mark n = m
    then n ∈ set (ll ! var n) ∧
      (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → mark p = m)
    else n ∉ set (concat ll)
  assume lla-st: ∀ i ≤ var (low p).
    ∃ prx. lla ! i = prx @ ll ! i ∧
      (∀ pt ∈ set prx. pt ∈ set-of lt ∧ var pt = i)

```



```

assume n-in-rt:  $n \in \text{set-of } rt$ 
show n-in-lla-marked: if marka  $n = m$ 
  then  $n \in \text{set } (lla \ ! \ \text{var } n) \wedge$ 
    ( $\forall nt \ p. \text{Dag } n \ \text{low } \text{high } nt \wedge p \in \text{set-of } nt \longrightarrow \text{marka } p = m$ )
  else  $n \notin \text{set } (\text{concat } lla)$ 
proof (cases  $n \in \text{set-of } lt$ )
  case True
  from True nodes-in-lla have n-in-ll:  $n \in \text{set } (lla \ ! \ \text{var } n)$ 
    by simp
  moreover
  from True wfmarking-lt
  have marka  $n = m$ 
    apply (cases lt)
    apply (auto simp add: wf-marking-def)
  done
  moreover
  {
    fix nt p
    assume Dag n low high nt
    with lt True have subset-nt-lt:  $\text{set-of } nt \subseteq \text{set-of } lt$ 
      by (rule dag-setof-subsetD)
    moreover assume  $p \in \text{set-of } nt$ 
    ultimately have  $p \in \text{set-of } lt$ 
      by blast
    with mit-marked have marka  $p = m$ 
      by simp
  }
  ultimately show ?thesis
    using n-in-rt
    apply clarsimp
  done
next
assume n-notin-lt:  $n \notin \text{set-of } lt$ 
show ?thesis
proof (cases marka  $n = m$ )
  case True
  from n-notin-lt mot-nc have marka-eq-mark: mark  $n = \text{marka } n$ 
    by simp
  from marka-eq-mark True have n-marked: mark  $n = m$ 
    by simp
  from rt n-in-rt have nnN:  $n \neq \text{Null}$ 
    apply –
    apply (rule set-of-nn [rule-format])
    apply fastforce
    apply assumption
  done
  from marked-child-ll n-in-rt marka-eq-mark nnN n-marked
  have n-in-ll:  $n \in \text{set } (ll \ ! \ \text{var } n)$ 
    by fastforce

```

m

```

from marked-child-ll n-in-rt marka-eq-mark nnN n-marked lt rt
have nt-mark:  $\forall nt p. \text{Dag } n \text{ low high } nt \wedge p \in \text{set-of } nt \longrightarrow \text{mark } p =$ 

  by simp
from nodes-in-lla n-in-ll lla-st
have n-in-lla:  $n \in \text{set } (lla ! \text{ var } n)$ 
proof (cases var (low p) < (var n))
  case True
  with lla-nc have (lla ! var n) = (ll ! var n)
    by fastforce
  with n-in-ll show ?thesis
    by fastforce
next
assume varnslp:  $\neg \text{var } (low p) < \text{var } n$ 
with lla-st
have ll-in-lla:  $\exists prx. lla ! (\text{var } n) = prx @ ll ! (\text{var } n)$ 
  apply -
  apply (erule-tac x=var n in allE)
  apply fastforce
  done
with n-in-ll show ?thesis
  by fastforce
qed
{
  fix nt pt
  assume nt-Dag: Dag n low high nt
  assume pt-in-nt: pt  $\in$  set-of nt
  have marka pt = m
  proof (cases pt  $\in$  set-of lt)
  case True
  with mit-marked show ?thesis
    by fastforce
  next
  assume pt-notin-lt: pt  $\notin$  set-of lt
  with mot-nc have mark pt = marka pt
    by fastforce
  with nt-mark nt-Dag pt-in-nt show ?thesis
    by fastforce
  qed
}
then have nt-marka:
   $\forall nt pt. \text{Dag } n \text{ low high } nt \wedge pt \in \text{set-of } nt \longrightarrow \text{marka } pt = m$ 
  by fastforce
with n-in-lla nt-marka True show ?thesis
  by fastforce
next
case False
note n-not-marka = this
with wfmarking-lt n-notin-lt

```

```

have mark n ≠ m
  by (simp add: wf-marking-def lt-Node)
with marked-child-ll
have n-notin-ll: n ∉ set (concat ll)
  by simp
show ?thesis
proof (cases n ∈ set (concat lla))
  case False with n-not-marka show ?thesis by simp
next
  case True
  with wf-levellist-subset [OF wfl-ll] n-notin-ll
  have n ∈ set-of lt
    by blast
  with n-notin-ll have False by simp
  thus ?thesis ..
qed
qed
qed
show ?thesis
  apply (intro conjI ordered-rt mark-rt)
  apply clarify
  subgoal premises prems for markb nextb levellistb llb
  proof -
    have llb: Levellist levellistb nextb llb by fact
    have wfl-rt: wf-levellist rt lla llb var by fact
    have wfmarking-rt: wf-marking rt marka markb m by fact
    show ?thesis
  proof (cases rt)
    case Tip
    from wfl-rt Tip have lla-llb: lla = llb
      by (simp add: wf-levellist-def)
    moreover
    from wfmarking-rt Tip rt have markb = marka
      by (simp add: wf-marking-def)
    moreover
    from wfl-ll varsll llb lla-llb
    obtain var-p-bounds: var p < length levellistb var p < length llb
      by (simp add: Levellist-length wf-levellist-def lt-Node Tip)
    with p-notin-ll lla-llb wfl-ll
    have p-notin-llb: ∀ i < length llb. p ∉ set (llb ! i)
    apply -
    apply (intro allI impI)
    apply (clarsimp simp add: wf-levellist-def lt-Node)
    apply (case-tac i ≤ var l)
    apply (drule-tac x=i in spec)
    using orderedt Tip lt-Node
    applyclarsimp
    apply (drule-tac x=i in spec)

```

```

  apply (drule-tac x=i in spec)
  apply clarsimp
  done
with llb pnN var-p-bounds
have llc: Levellist (levellistb[var p := p])
  (nextb(p := levellistb ! var p))
  (llb[var p := p # llb ! var p])
  apply (clarsimp simp add: Levellist-def map-update)
  apply (erule-tac x=i in allE)
  apply (erule-tac x=i in allE)
  apply clarsimp
  apply (case-tac i=var p)
  apply simp
  apply simp
  done
ultimately show ?thesis
using Tip lt-Node varsll orderedt lt rt pnN wfll-lt wfmarking-lt
apply (clarsimp simp add: wf-levellist-def wf-marking-def)
apply (intro conjI)
apply (rule allI)
apply (rule conjI)
apply (erule-tac x=q in allE)
apply (case-tac var p = var q)
apply fastforce
apply fastforce
apply (case-tac var p = var q)
apply hypsubst-thin
apply fastforce
apply fastforce
apply (rule allI)
apply (rotate-tac 4)
apply (erule-tac x=i in allE)
apply (case-tac i=var p)
apply simp
apply (case-tac var (low p) < i)
apply simp
apply simp
apply (erule exE)
apply (rule-tac x=prx in exI)
apply (intro conjI)
apply simp
apply clarify
apply (rotate-tac 15)
apply (erule-tac x=pt in ballE)
apply fastforce
apply fastforce
done
next
case (Node lrt r rrt)

```

```

have rt-Node:  $rt = \text{Node } lrt \ r \ rrt$  by fact
from wfl-rt rt-Node
have llb-eq-lla:  $\text{length } llb = \text{length } lla$ 
  by (simp add: wf-levellist-def)
with llb lla
have llb-eq-lla':  $\text{length } levellistb = \text{length } levellista$ 
  by (simp add: Levellist-length)
from wfl-rt rt-Node
have nodes-in-llb:  $\forall q. q \in \text{set-of } rt \longrightarrow q \in \text{set } (llb ! (q \rightarrow var))$ 
  by (simp add: wf-levellist-def)
from wfl-rt rt-Node rt
have llb-st:  $(\forall i \leq (\text{high } p) \rightarrow var. (\exists prx. (llb ! i) = prx @ (lla ! i) \wedge (\forall pt \in \text{set } prx. pt \in \text{set-of } rt \wedge pt \rightarrow var = i)))$ 
  by (simp add: wf-levellist-def)
from wfl-rt rt-Node rt
have llb-nc:
   $\forall i. ((\text{high } p) \rightarrow var) < i \longrightarrow (llb ! i) = (lla ! i)$ 
  by (simp add: wf-levellist-def)
from wfmarking-rt rt-Node rt
have mort-nc:  $\forall n. n \notin \text{set-of } rt \longrightarrow \text{marka } n = \text{markb } n$ 
  by (simp add: wf-marking-def)
from wfmarking-rt rt-Node rt
have mirt-marked:  $\forall n. n \in \text{set-of } rt \longrightarrow \text{markb } n = m$ 
  by (simp add: wf-marking-def)
with p-notin-ll wfl-rt wfl-lt
have p-notin-llb:  $\forall i < \text{length } llb. p \notin \text{set } (llb ! i)$ 
  apply –
  apply (intro allI impI)
  apply (clarsimp simp add: wf-levellist-def lt-Node rt-Node)
  apply (case-tac i ≤ var r)
  apply (drule-tac x=i in spec)
  using orderedt rt-Node lt-Node
  apply clarsimp
  apply (erule disjE)
  apply clarsimp
  apply (case-tac i ≤ var l)
  apply (drule-tac x=i in spec)
  apply clarsimp
  apply clarsimp
  apply (subgoal-tac llb ! i = lla ! i)
  prefer 2
  apply clarsimp
  apply (case-tac i ≤ var l)
  apply (drule-tac x=i in spec, erule impE, assumption)
  apply clarsimp
  using orderedt rt-Node lt-Node
  apply clarsimp
  apply clarsimp

```

```

done
from wfl-lt wfl-rt varsll lla llb
obtain var-p-bounds: var p < length levellistb var p < length llb
  by (simp add: Levellist-length wf-levellist-def lt-Node rt-Node)
with p-notin-llb llb pnN var-p-bounds
have llc: Levellist (levellistb[var p := p])
  (nextb(p := levellistb ! var p))
  (llb[var p := p # llb ! var p])
  apply (clarsimp simp add: Levellist-def map-update)
  apply (erule-tac x=i in allE)
  apply (erule-tac x=i in allE)
  apply clarsimp
  apply (case-tac i=var p)
  apply simp
  apply simp
done
then show ?thesis
proof (clarsimp)
show wf-levellist (Node lt p rt) ll (llb[var p := p#llb ! var p]) var ∧
  wf-marking (Node lt p rt) mark (markb(p := m)) m
proof -
have nodes-in-upllb: ∀ q. q ∈ set-of (Node lt p rt)
  → q ∈ set (llb[var p := p # llb ! var p] ! (var q))
  apply -
  apply (rule allI)
  apply (rule impI)
proof -
fix q
assume q-in-t: q ∈ set-of (Node lt p rt)
show q-in-upllb:
  q ∈ set (llb[var p := p # llb ! var p] ! (var q))
proof (cases q ∈ set-of rt)
case True
with nodes-in-llb have q-in-llb: q ∈ set (llb ! (var q))
  by fastforce
from orderedt rt-Node lt-Node lt rt
have ordered-rt: ordered rt var
  by fastforce
from True rt ordered-rt rt-Node lt lt-Node have var q ≤ var r
  apply -
  apply (drule subnodes-ordered)
  apply fastforce
  apply fastforce
  apply fastforce
done
with orderedt rt lt rt-Node lt-Node have var q < var p
  by fastforce
then have
  llb[var p := p#llb ! var p] ! var q =

```

```

    llb ! var q
  by fastforce
with q-in-llb show ?thesis
  by fastforce
next
assume q-notin-rt: q ∉ set-of rt
show q ∈ set (llb[var p :=p # llb ! var p] ! var q)
proof (cases q ∈ set-of lt)
  case True
  assume q-in-lt: q ∈ set-of lt
  with nodes-in-lla have q-in-lla: q ∈ set (lla ! (var q))
  by fastforce
  from orderedt rt-Node lt-Node lt rt
  have ordered-lt: ordered lt var
  by fastforce
  from q-in-lt lt ordered-lt rt-Node rt lt-Node
  have var q ≤ var l
  apply –
  apply (drule subnodes-ordered)
  apply fastforce
  apply fastforce
  apply fastforce
  done
with orderedt rt lt rt-Node lt-Node have qsp: var q < var p
  by fastforce
then show ?thesis
proof (cases var q ≤ var (high p))
  case True
  with llb-st
  have ∃ prx. (llb ! (var q)) = prx@(lla ! (var q))
  by fastforce
  with nodes-in-lla q-in-lla
  have q-in-llb: q ∈ set (llb ! (var q))
  by fastforce
  from qsp
  have llb[var p :=p # llb ! var p]!var q =
    llb ! (var q)
  by fastforce
  with q-in-llb show ?thesis
  by fastforce
next
assume ¬ var q ≤ var (high p)
with llb-nc have llb ! (var q) = lla ! (var q)
  by fastforce
with q-in-lla have q-in-llb: q ∈ set (llb ! (var q))
  by fastforce
from qsp have
  llb[var p :=p # llb ! var p] ! var q = llb ! (var q)
  by fastforce

```

```

    with  $q$ -in-llb show ?thesis
      by fastforce
  qed
next
  assume  $q$ -notin- $lt$ :  $q \notin \text{set-of } lt$ 
  with  $q$ -notin- $rt$   $rt$   $lt$   $rt$ -Node  $lt$ -Node  $q$ -in- $t$  have  $qp$ :  $q = p$ 
    by fastforce
  with varsll  $lla$ -eq- $ll$   $llb$ -eq- $lla$  have  $var\ p < \text{length } llb$ 
    by fastforce
  with  $qp$  show ?thesis
    by simp
  qed
qed
qed
have  $prx$ - $ll$ - $st$ :  $\forall i \leq var\ p$ .
  ( $\exists prx$ .  $llb[var\ p := p \# llb!var\ p]!i = prx@ll!i \wedge$ 
    ( $\forall pt \in \text{set } prx$ .  $pt \in \text{set-of } (Node\ lt\ p\ rt) \wedge var\ pt = i$ ))
  apply -
  apply (rule allI)
  apply (rule impI)
proof -
  fix  $i$ 
  assume  $isep$ :  $i \leq var\ p$ 
  show  $\exists prx$ .  $llb[var\ p := p \# llb!var\ p]!i = prx@ll!i \wedge$ 
    ( $\forall pt \in \text{set } prx$ .  $pt \in \text{set-of } (Node\ lt\ p\ rt) \wedge var\ pt = i$ )
  proof (cases  $i = var\ p$ )
    case True
    with orderedt  $lt$   $lt$ -Node  $rt$   $rt$ -Node
    have  $lpsp$ :  $var\ (low\ p) < var\ p$ 
      by fastforce
    with orderedt  $lt$   $lt$ -Node  $rt$   $rt$ -Node
    have  $hpsp$ :  $var\ (high\ p) < var\ p$ 
      by fastforce
    with  $lpsp$   $lla$ -nc
    have  $llall$ :  $lla ! var\ p = ll ! var\ p$ 
      by fastforce
    with  $hpsp$   $llb$ -nc have  $llb ! var\ p = ll ! var\ p$ 
      by fastforce
    with  $llb$ -eq- $lla$   $lla$ -eq- $ll$   $isep$  True varsll  $lt$   $rt$  show ?thesis
      apply -
      apply (rule-tac  $x=[p]$  in  $exI$ )
      apply (rule conjI)
      apply simp
      apply (rule ballI)
      apply fastforce
    done
  next
    assume  $inp$ :  $i \neq var\ p$ 
    show ?thesis

```



```

proof (cases var (low p) < i)
  case True
  with lla-nc have llall: lla ! i = ll ! i
    by fastforce
  assume vpsi: var (low p) < i
  show ?thesis
  proof (cases var (high p) < i)
    case True
    with llall llb-nc have llb ! i = ll ! i
      by fastforce
    with inp True vpsi varsll lt rt show ?thesis
      apply -
      apply (rule-tac x=[] in exI)
      apply (rule conjI)
      apply simp
      apply (rule ballI)
      apply fastforce
      done
  next
  assume isehp: ¬ var (high p) < i
  with vpsi lla-nc have lla-ll: lla ! i = ll ! i
    by fastforce
  with isehp llb-st
  have prx-lla: ∃ prx. llb ! i = prx @ lla ! i ∧
    (∀ pt ∈ set prx. pt ∈ set-of rt ∧ var pt = i)
    apply -
    apply (erule-tac x=i in allE)
    apply simp
    done
  with lla-ll inp rt show ?thesis
    apply -
    apply (erule exE)
    apply (rule-tac x=prx in exI)
    apply simp
    done
  qed
next
  assume iselp: ¬ var (low p) < i
  show ?thesis
  proof (cases var (high p) < i)
    case True
    with llb-nc have llb-ll: llb ! i = lla ! i
      by fastforce
    with iselp lla-st
  have prx-ll: ∃ prx. lla ! i = prx @ ll ! i ∧
    (∀ pt ∈ set prx. pt ∈ set-of lt ∧ var pt = i)
    apply -
    apply (erule-tac x=i in allE)
    apply simp

```

```

    done
  with llb-ll inp lt show ?thesis
    apply -
    apply (erule exE)
    apply (rule-tac x=prx in exI)
    apply simp
    done
next
assume isehp:  $\neg \text{var } (\text{high } p) < i$ 
from iselp lla-st
have prxl:  $\exists \text{prx}. \text{lla} ! i = \text{prx} @ \text{ll} ! i \wedge$ 
  ( $\forall \text{pt} \in \text{set } \text{prx}. \text{pt} \in \text{set-of } \text{lt} \wedge \text{var } \text{pt} = i$ )
  by fastforce
from isehp llb-st
have prxh:  $\exists \text{prx}. \text{llb} ! i = \text{prx} @ \text{lla} ! i \wedge$ 
  ( $\forall \text{pt} \in \text{set } \text{prx}. \text{pt} \in \text{set-of } \text{rt} \wedge \text{var } \text{pt} = i$ )
  by fastforce
with prxl inp lt pnN rt show ?thesis
  apply -
  apply (elim exE)
  apply (rule-tac x=prxa @ prx in exI)
  apply simp
  apply (elim conjE)
  apply fastforce
  done
qed
qed
qed
qed
have big-Nodes-nc:  $\forall i. (\text{p} \rightarrow \text{var}) < i$ 
   $\rightarrow (\text{llb}[\text{var } p := p \# \text{llb} ! \text{var } p]) ! i = \text{ll} ! i$ 
  apply -
  apply (rule allI)
  apply (rule impI)
proof -
  fix i
  assume psi:  $\text{var } p < i$ 
  with orderedt lt rt lt-Node rt-Node have lpsi:  $\text{var } (\text{low } p) < i$ 
    by fastforce
  with lla-nc have lla-ll:  $\text{lla} ! i = \text{ll} ! i$ 
    by fastforce
  from psi orderedt lt rt lt-Node rt-Node have hpsi:  $\text{var } (\text{high } p) < i$ 
    by fastforce
  with llb-nc have llb-lla:  $\text{llb} ! i = \text{lla} ! i$ 
    by fastforce
  from psi
  have upllb-llb:  $\text{llb}[\text{var } p := p \# \text{llb} ! \text{var } p] ! i = \text{llb} ! i$ 
    by fastforce
  from upllb-llb llb-lla lla-ll

```

```

    show llb[var p :=p # llb ! var p] ! i = ll ! i
      by fastforce
  qed
  from lla-eq-ll llb-eq-lla
  have length-eq: length (llb[var p :=p # llb ! var p]) = length ll
    by fastforce
  from length-eq big-Nodes-nc prx-ll-st nodes-in-upllb
  have wf-ll-upllb:
    wf-levellist (Node lt p rt) ll (llb[var p :=p # llb ! var p]) var
    by (simp add: wf-levellist-def)
  have mark-nc:
     $\forall n. n \notin \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow (\text{markb}(p:=m)) \ n = \text{mark } n$ 
    apply -
    apply (rule allI)
    apply (rule impI)
  proof -
    fix n
    assume nnit:  $n \notin \text{set-of } (\text{Node } lt \ p \ rt)$ 
    with lt rt have nnilt:  $n \notin \text{set-of } lt$ 
      by fastforce
    from nnit lt rt have nnirt:  $n \notin \text{set-of } rt$ 
      by fastforce
    with nnilt mot-nc mort-nc have mb-eq-m:  $\text{markb } n = \text{mark } n$ 
      by fastforce
    from nnit have n≠p
      by fastforce
    then have upmarkb-markb:  $(\text{markb}(p :=m)) \ n = \text{markb } n$ 
      by fastforce
    with mb-eq-m show  $(\text{markb}(p :=m)) \ n = \text{mark } n$ 
      by fastforce
  qed
  have mark-c:  $\forall n. n \in \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow (\text{markb}(p :=m)) \ n$ 
= m
    apply -
    apply (intro allI)
    apply (rule impI)
  proof -
    fix n
    assume nint:  $n \in \text{set-of } (\text{Node } lt \ p \ rt)$ 
    show  $(\text{markb}(p :=m)) \ n = m$ 
    proof (cases n=p)
      case True
        then show ?thesis
          by fastforce
      next
        assume nnp:  $n \neq p$ 
        show ?thesis
        proof (cases n ∈ set-of rt)
          case True

```

```

    with mirt-marked have markb n = m
      by fastforce
    with nnp show ?thesis
      by fastforce
  next
    assume nninrt:  $n \notin \text{set-of } rt$ 
    with nint nnp have ninlt:  $n \in \text{set-of } lt$ 
      by fastforce
    with mit-marked have marka-m:  $\text{marka } n = m$ 
      by fastforce
    from mort-nc nninrt have  $\text{marka } n = \text{markb } n$ 
      by fastforce
    with marka-m have  $\text{markb } n = m$ 
      by fastforce
    with nnp show ?thesis
      by fastforce
  qed
qed
qed
from mark-c mark-nc
have wf-mark: wf-marking (Node lt p rt) mark (markb(p := m)) m
  by (simp add: wf-marking-def)
with wf-ll-upllb show ?thesis
  by fastforce
qed
qed
qed
done
qed
done
qed
next
fix var low high p lt rt and levellist and
  ll::ref list list and mark::ref  $\Rightarrow$  bool and next
assume pnN:  $p \neq \text{Null}$ 
assume ll: Levellist levellist next ll
assume vpsll:  $\text{var } p < \text{length } \text{levellist}$ 
assume orderedt: ordered (Node lt p rt) var
assume marked-child-ll:  $\forall n \in \text{set-of } (\text{Node } lt \ p \ rt).$ 
  if  $\text{mark } n = \text{mark } p$ 
    then  $n \in \text{set } (ll \ ! \ \text{var } n) \wedge$ 
       $(\forall nt \ pa. \ \text{Dag } n \ \text{low } \text{high } \ nt \wedge \ pa \in \text{set-of } \ nt \longrightarrow \text{mark } pa = \text{mark } p)$ 
    else  $n \notin \text{set } (\text{concat } ll)$ 
assume lt: Dag (low p) low high lt
assume rt: Dag (high p) low high rt
show wf-levellist (Node lt p rt) ll ll var  $\wedge$ 
  wf-marking (Node lt p rt) mark mark (mark p)
proof –

```

from *marked-child-ll* pnN *lt rt* **have** *marked-st*:
 $(\forall pa. pa \in \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow \text{mark } pa = \text{mark } p)$
apply –
apply (*drule-tac* $x=p$ **in** *bspec*)
apply *simp*
apply (*clarsimp*)
apply (*erule-tac* $x=(\text{Node } lt \ p \ rt)$ **in** *allE*)
apply *simp*
done
have *nodest-in-ll*:
 $\forall q. q \in \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow q \in \text{set } (ll \ ! \ \text{var } q)$
proof –
from *marked-child-ll* pnN **have** *pinll*: $p \in \text{set } (ll \ ! \ \text{var } p)$
apply –
apply (*drule-tac* $x=p$ **in** *bspec*)
apply *simp*
apply *fastforce*
done
from *marked-st* *marked-child-ll* *lt rt* **show** *?thesis*
apply –
apply (*rule allI*)
apply (*erule-tac* $x=q$ **in** *allE*)
apply (*rule impI*)
apply (*erule impE*)
apply *assumption*
apply (*drule-tac* $x=q$ **in** *bspec*)
apply *simp*
apply *fastforce*
done
qed
have *levellist-nc*: $\forall i \leq \text{var } p. (\exists prx. ll \ ! \ i = prx@ll \ ! \ i) \wedge$
 $(\forall pt \in \text{set } prx. pt \in \text{set-of } (\text{Node } lt \ p \ rt) \wedge \text{var } pt = i)$
apply –
apply (*rule allI*)
apply (*rule impI*)
apply (*rule-tac* $x=[]$ **in** *exI*)
apply *fastforce*
done
have *ll-nc*: $\forall i. (\text{var } p) < i \longrightarrow ll \ ! \ i = ll \ ! \ i$
by *fastforce*
have *length-ll*: $\text{length } ll = \text{length } ll$
by *fastforce*
with *ll-nc* *levellist-nc* *nodest-in-ll*
have *wf*: *wf-levellist* (*Node* *lt p rt*) *ll ll var*
by (*simp add*: *wf-levellist-def*)
have *m-nc*: $\forall n. n \notin \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow \text{mark } n = \text{mark } n$
by *fastforce*
from *marked-st* **have** $\forall n. n \in \text{set-of } (\text{Node } lt \ p \ rt) \longrightarrow \text{mark } n = \text{mark } p$
by *fastforce*

with *m-nc* **have** *wf-marking* (*Node lt p rt*) *mark mark* (*mark p*)
by (*simp add: wf-marking-def*)
with *wf* **show** *?thesis*
by *fastforce*
qed
qed

lemma *allD*: $\forall ll. P ll \implies P ll$
by *blast*

lemma *replicate-spec*: $\llbracket \forall i < n. xs ! i = x; n = \text{length } xs \rrbracket$
 $\implies \text{replicate } (\text{length } xs) x = xs$
apply *hypsubst-thin*
apply (*induct xs*)
apply *simp*
apply *force*
done

lemma (**in** *Levellist-impl*) *Levellist-spec-total*:

shows $\forall \sigma t. \Gamma, \Theta \vdash_t$
 $\{ \sigma. \text{Dag } 'p \text{ 'low 'high } t \wedge (\forall i < \text{length } 'levellist. 'levellist ! i = \text{Null}) \wedge$
 $\text{length } 'levellist = 'p \rightarrow 'var + 1 \wedge$
 $\text{ordered } t \text{ 'var} \wedge (\forall n \in \text{set-of } t. 'mark n = (\neg 'm)) \}$
 $'levellist := \text{PROC } \text{Levellist } ('p, 'm, 'levellist)$
 $\{ \exists ll. \text{Levellist } 'levellist \text{ 'next } ll \wedge \text{wf-ll } t ll \sigma \text{var} \wedge$
 $\text{length } 'levellist = \sigma p \rightarrow \sigma \text{var} + 1 \wedge$
 $\text{wf-marking } t \sigma \text{mark } 'mark \sigma m \wedge$
 $(\forall p. p \notin \text{set-of } t \longrightarrow \sigma \text{next } p = \text{'next } p) \}$
apply (*hoare-rule HoareTotal.conseq*)
apply (*rule-tac ll=replicate* ($\sigma p \rightarrow \sigma \text{var} + 1$) \llbracket **in** *allD* [*OF* *Levellist-spec-total* \rrbracket)
apply (*intro allI impI*)
apply (*rule-tac x=σ in exI*)
apply (*rule-tac x=t in exI*)
apply (*rule conjI*)
apply (*clarsimp split:if-split-asm simp del: concat-replicate-trivial*)
apply (*frule replicate-spec [symmetric]*)
apply (*simp*)
apply (*clarsimp simp add: Levellist-def*)
apply (*case-tac i*)
apply *simp*
apply *simp*
apply (*simp add: Collect-conv-if split:if-split-asm*)
apply *vcg-step*
apply (*elim exE conjE*)
apply (*rule-tac x=ll' in exI*)
apply *simp*
apply (*thin-tac* $\forall p. p \notin \text{set-of } t \longrightarrow \text{next } p = \text{nexta } p$)
apply (*simp add: wf-levellist-def wf-ll-def*)
apply (*case-tac t = Tip*)

```

apply simp
apply (rule conjI)
apply clarsimp
apply (case-tac k)
apply simp
apply simp
apply (subgoal-tac length ll'=Suc (var Null))
apply (simp add: Levellist-length)
apply fastforce
apply (split dag.splits)
apply simp
apply (elim conjE)
apply (intro conjI)
apply (rule allI)
apply (erule-tac x=pa in allE)
apply clarify
prefer 2
apply (simp add: Levellist-length)
apply (rule allI)
apply (rule impI)
apply (rule ballI)
apply (rotate-tac 11)
apply (erule-tac x=k in allE)
apply (rename-tac dag1 ref dag2 k pa)
apply (subgoal-tac k <= var ref)
prefer 2
apply (subgoal-tac ref = p)
apply simp
apply clarify
apply (erule-tac ?P = Dag p low high (Node dag1 ref dag2) in rev-mp)
apply (simp (no-asm))
apply (rotate-tac 14)
apply (erule-tac x=k in allE)
apply clarify
apply (erule-tac x=k in allE)
apply clarify
apply (case-tac k)
apply simp
apply simp
done

end

```

7 Proof of Procedure ShareRep

theory *ShareRepProof* **imports** *ProcedureSpecs Simpl.HeapList* **begin**

lemma (**in** *ShareRep-impl*) *ShareRep-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC ShareRep } ('nodeslist, 'p)$

$\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [rep]\}$
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

lemma *hd-filter-cons*:

$\bigwedge i. \llbracket P (xs ! i) p; i < \text{length } xs; \forall no \in \text{set } (\text{take } i \text{ } xs). \neg P \text{ no } p; \forall a b. P a b = P b a \rrbracket$

$\implies xs ! i = \text{hd } (\text{filter } (P p) \text{ } xs)$

apply (induct xs)
apply simp
apply (case-tac P a p)
apply simp
apply (case-tac i)
apply simp
apply simp
apply (case-tac i)
apply simp
apply auto
done

lemma (in ShareRep-impl) *ShareRep-spec-total*:

shows

$\forall \sigma \text{ ns. } \Gamma, \Theta \vdash_t$
 $\{\sigma. \text{List } 'nodeslist \text{ } 'next \text{ } ns \wedge$
 $(\forall no \in \text{set } ns. no \neq \text{Null} \wedge$
 $((no \rightarrow \text{low} = \text{Null}) = (no \rightarrow \text{high} = \text{Null})) \wedge$
 $(\text{isLeaf-pt } 'p \text{ } 'low \text{ } 'high \longrightarrow \text{isLeaf-pt } no \text{ } 'low \text{ } 'high) \wedge$
 $no \rightarrow \text{var} = 'p \rightarrow \text{var}) \wedge$
 $'p \in \text{set } ns\}$

PROC ShareRep ('nodeslist, 'p)

$\{\sigma_p \rightarrow 'rep = \text{hd } (\text{filter } (\lambda sn. \text{repNodes-eq } sn \text{ } \sigma_p \text{ } \sigma_{low} \text{ } \sigma_{high} \text{ } \sigma_{rep}) \text{ } ns) \wedge$
 $(\forall pt. pt \neq \sigma_p \longrightarrow pt \rightarrow \sigma_{rep} = pt \rightarrow 'rep) \wedge$
 $(\sigma_p \rightarrow 'rep \rightarrow \sigma_{var} = \sigma_p \rightarrow \sigma_{var})\}$

apply (hoare-rule HoareTotal.ProcNoRec1)

apply (hoare-rule anno=)

IF (isLeaf-pt 'p 'low 'high)
THEN 'p \rightarrow 'rep ::= 'nodeslist
ELSE

WHILE ('nodeslist \neq Null)

INV $\{\exists prx \text{ } sfx. \text{List } 'nodeslist \text{ } 'next \text{ } sfx \wedge ns = prx @ sfx \wedge$

$\neg \text{isLeaf-pt } 'p \text{ } 'low \text{ } 'high \wedge$

$(\forall no \in \text{set } ns. no \neq \text{Null} \wedge$

$((no \rightarrow \sigma_{low} = \text{Null}) = (no \rightarrow \sigma_{high} = \text{Null})) \wedge$

$(\text{isLeaf-pt } \sigma_p \text{ } \sigma_{low} \text{ } \sigma_{high} \longrightarrow \text{isLeaf-pt } no \text{ } \sigma_{low} \text{ } \sigma_{high}) \wedge$

$no \rightarrow \sigma_{var} = \sigma_p \rightarrow \sigma_{var}) \wedge$

$\sigma_p \in \text{set } ns \wedge$

$(\exists pt \in \text{set } prx. \text{repNodes-eq } pt \text{ } \sigma_p \text{ } \sigma_{low} \text{ } \sigma_{high} \text{ } \sigma_{rep})$


```

    → 'rep σp = hd (filter (λ sn. repNodes-eq sn σp σlow σhigh σrep) prx) ∧
      (∀ pt. pt ≠ σp → pt→σrep = pt→'rep) ∧
      ((∀ pt ∈ set prx. ¬ repNodes-eq pt σp σlow σhigh σrep) → σrep = 'rep) ∧
      ('nodeslist ≠ Null →
        (∀ pt ∈ set prx. ¬ repNodes-eq pt σp σlow σhigh σrep) ∧
        ('p = σp ∧ 'high = σhigh ∧ 'low = σlow))
  VAR MEASURE (length (list 'nodeslist 'next))
  DO
    IF (repNodes-eq 'nodeslist 'p 'low 'high 'rep)
      THEN 'p→'rep ::= 'nodeslist;; 'nodeslist ::= Null
      ELSE 'nodeslist ::= 'nodeslist→'next
    FI
  OD
  FI in HoareTotal.annotateI)
apply vcg
using [[simp-depth-limit = 2]]
apply (rule conjI)
apply clarify
apply (simp (no-asm-use))
prefer 2
apply clarify
apply (rule-tac x=[] in exI)
apply (rule-tac x=ns in exI)
apply (simp (no-asm-use))
prefer 2
apply clarify
apply (rule conjI)
apply clarify
apply (rule conjI)
apply (clarsimp simp add: List-list)
apply (simp (no-asm-use))
apply (rule conjI)
apply assumption
prefer 2
apply clarify
apply (simp (no-asm-use))
apply (rule conjI)
apply (clarsimp simp add: List-list)
apply (simp only: List-not-Null simp-thms triv-forall-equality)
apply clarify
apply (simp only: triv-forall-equality)
apply (rename-tac sfx)
apply (rule-tac x=prx@[nodeslist] in exI)
apply (rule-tac x=sfx in exI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply simp
prefer 4

```

```

apply (elim exE conjE)
apply (simp (no-asm-use))
apply hypsubst
using [[simp-depth-limit = 100]]
proof -

  fix ns var low high rep next p nodeslist
  assume ns: List nodeslist next ns
  assume no-prop:  $\forall no \in set\ ns.$ 
    no  $\neq$  Null  $\wedge$ 
    (low no = Null) = (high no = Null)  $\wedge$ 
    (isLeaf-pt p low high  $\longrightarrow$  isLeaf-pt no low high)  $\wedge$  var no = var p
  assume p-in-ns: p  $\in$  set ns
  assume p-Leaf: isLeaf-pt p low high
  show nodeslist = hd [sn $\leftarrow$ ns . repNodes-eq sn p low high rep]  $\wedge$ 
    var nodeslist = var p
  proof -
    from p-in-ns no-prop have p-not-Null: p  $\neq$  Null
      using [[simp-depth-limit=2]]
      by auto
    from p-in-ns have ns  $\neq$  []
      by (cases ns) auto
    with ns obtain ns' where ns': ns = nodeslist#ns'
      by (cases nodeslist=Null) auto
    with no-prop p-Leaf obtain
      isLeaf-pt nodeslist low high and
      var-eq: var nodeslist = var p and
      nodeslist $\neq$ Null
      using [[simp-depth-limit=2]]
      by auto
    with p-not-Null p-Leaf have repNodes-eq nodeslist p low high rep
      by (simp add: repNodes-eq-def isLeaf-pt-def null-comp-def)
    with ns' var-eq
    show ?thesis
      by simp
  qed
next

  fix var::ref $\Rightarrow$ nat and low high rep repa p prx sfx next
  assume sfx: List Null next sfx
  assume p-in-ns: p  $\in$  set (prx @ sfx)
  assume no-props:  $\forall no \in set\ (prx\ @\ sfx).$ 
    no  $\neq$  Null  $\wedge$ 
    (low no = Null) = (high no = Null)  $\wedge$ 
    (isLeaf-pt p low high  $\longrightarrow$  isLeaf-pt no low high)  $\wedge$  var no = var p
  assume match-prx: ( $\exists pt \in set\ prx.$  repNodes-eq pt p low high rep)  $\longrightarrow$ 
    repa p = hd [sn $\leftarrow$ prx . repNodes-eq sn p low high rep]  $\wedge$ 
    ( $\forall pt.$  pt  $\neq$  p  $\longrightarrow$  rep pt = repa pt)
  show repa p = hd [sn $\leftarrow$ prx @ sfx . repNodes-eq sn p low high rep]  $\wedge$ 

```

$(\forall pt. pt \neq p \longrightarrow rep\ pt = repa\ pt) \wedge var\ (repa\ p) = var\ p$
proof –
from *sfx*
have *sfx-Nil*: *sfx*=[]
by *simp*
with *p-in-ns* **have** *ex-match*: $(\exists pt \in set\ prx. repNodes\text{-}eq\ pt\ p\ low\ high\ rep)$
apply –
apply (*rule-tac* *x=p* **in** *bexE*)
apply (*simp* *add*: *repNodes-eq-def*)
apply *simp*
done
hence *not-empty*: $[sn \leftarrow prx . repNodes\text{-}eq\ sn\ p\ low\ high\ rep] \neq []$
apply –
apply (*erule* *bexE*)
apply (*rule* *filter-not-empty*)
apply *auto*
done
from *ex-match* *match-prx* **obtain**
found: $repa\ p = hd\ [sn \leftarrow prx . repNodes\text{-}eq\ sn\ p\ low\ high\ rep]$ **and**
unmodif: $\forall pt. pt \neq p \longrightarrow rep\ pt = repa\ pt$
by *blast*
from *hd-filter-in-list* [*OF* *not-empty*] *found*
have $repa\ p \in set\ prx$
by *simp*
with *no-props*
have $var\ (repa\ p) = var\ p$
using [*simp-depth-limit*=2]
by *simp*
with *found* *unmodif* *sfx-Nil*
show *?thesis*
by *simp*
qed
next

fix *var* *low* *high* *p* *repa* *next* *nodeslist* *prx* *sfx*
assume *nodeslist-not-Null*: *nodeslist* \neq *Null*
assume *p-no-Leaf*: $\neg isLeaf\text{-}pt\ p\ low\ high$
assume *no-props*: $\forall no \in set\ prx \cup set\ (nodeslist\ \# \ sfx).$
 $no \neq Null \wedge (low\ no = Null) = (high\ no = Null) \wedge var\ no = var\ p$
assume *p-in-ns*: $p \in set\ prx \vee p \in set\ (nodeslist\ \# \ sfx)$
assume *match-prx*: $(\exists pt \in set\ prx. repNodes\text{-}eq\ pt\ p\ low\ high\ repa) \longrightarrow$
 $repa\ p = hd\ [sn \leftarrow prx . repNodes\text{-}eq\ sn\ p\ low\ high\ repa]$
assume *nomatch-prx*: $\forall pt \in set\ prx. \neg repNodes\text{-}eq\ pt\ p\ low\ high\ repa$
assume *nomatch-nodeslist*: $\neg repNodes\text{-}eq\ nodeslist\ p\ low\ high\ repa$
assume *sfx*: *List* (*next* *nodeslist*) *next* *sfx*
show $(\forall no \in set\ prx \cup set\ (nodeslist\ \# \ sfx).$
 $no \neq Null \wedge (low\ no = Null) = (high\ no = Null) \wedge var\ no = var\ p) \wedge$
 $((\exists pt \in set\ (prx\ @ \ [nodeslist]). repNodes\text{-}eq\ pt\ p\ low\ high\ repa) \longrightarrow$
 $repa\ p = hd\ [sn \leftarrow prx\ @ \ [nodeslist] . repNodes\text{-}eq\ sn\ p\ low\ high\ repa]) \wedge$

```

      (next nodeslist ≠ Null →
        (∀ pt∈set (prx @ [nodeslist]). ¬ repNodes-eq pt p low high repa))
proof –
  from nomatch-prx nomatch-nodeslist
  have ((∃ pt∈set (prx @ [nodeslist]). repNodes-eq pt p low high repa) →
    repa p = hd [sn←prx @ [nodeslist] . repNodes-eq sn p low high repa])
    by auto
  moreover
  from nomatch-prx nomatch-nodeslist
  have (next nodeslist ≠ Null →
    (∀ pt∈set (prx @ [nodeslist]). ¬ repNodes-eq pt p low high repa))
    by auto
  ultimately show ?thesis
    using no-props
    by (intro conjI)
qed
next

fix var low high p repa next nodeslist prx sfx
assume nodeslist-not-Null: nodeslist ≠ Null
assume sfx: List nodeslist next sfx
assume p-not-Leaf: ¬ isLeaf-pt p low high
assume no-props: ∀ no∈set prx ∪ set sfx.
  no ≠ Null ∧
  (low no = Null) = (high no = Null) ∧
  (isLeaf-pt p low high → isLeaf-pt no low high) ∧ var no = var p
assume p-in-ns: p ∈ set prx ∨ p ∈ set sfx
assume match-prx: (∃ pt∈set prx. repNodes-eq pt p low high repa) →
  repa p = hd [sn←prx . repNodes-eq sn p low high repa]
assume nomatch-prx: ∀ pt∈set prx. ¬ repNodes-eq pt p low high repa
assume match: repNodes-eq nodeslist p low high repa
show (∀ no∈set prx ∪ set sfx.
  no ≠ Null ∧
  (low no = Null) = (high no = Null) ∧
  (isLeaf-pt p low high → isLeaf-pt no low high) ∧ var no = var p) ∧
  (p ∈ set prx ∨ p ∈ set sfx) ∧
  ((∃ pt∈set prx ∪ set sfx. repNodes-eq pt p low high repa) →
  nodeslist =
  hd ([sn←prx . repNodes-eq sn p low high repa] @
    [sn←sfx . repNodes-eq sn p low high repa])) ∧
  ((∀ pt∈set prx ∪ set sfx. ¬ repNodes-eq pt p low high repa) →
  repa = repa(p := nodeslist))
proof –
  from nodeslist-not-Null sfx
  obtain sfx' where sfx': sfx=nodeslist#sfx'
  by (cases nodeslist=Null) auto
  from nomatch-prx match sfx'
  have hd: hd ([sn←prx . repNodes-eq sn p low high repa] @
    [sn←sfx . repNodes-eq sn p low high repa]) = nodeslist

```

```

    by simp
  from match sfx'
  have triv: (( $\forall pt \in \text{set } prx \cup \text{set } sfx. \neg \text{repNodes-eq } pt \ p \ \text{low } \text{high } \text{repa}$ )  $\longrightarrow$ 
     $\text{repa} = \text{repa}(p := \text{nodeslist})$ )
    by simp
  show ?thesis
    apply (rule conjI)
    apply (rule no-props)
    apply (intro conjI)
    apply (rule p-in-ns)
    apply (simp add: hd)
    apply (rule triv)
  done
qed
qed
end

```

8 Proof of Procedure ShareReduceRepList

theory *ShareReduceRepListProof* **imports** *ShareRepProof* **begin**

lemma (in *ShareReduceRepList-impl*) *ShareReduceRepList-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC } \text{ShareReduceRepList } (\text{'nodeslist})$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{rep}]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma *hd-filter-app*: $\llbracket \text{filter } P \ xs \neq []; \text{zs} = \text{xs} @ \text{ys} \rrbracket \implies$
 $\text{hd } (\text{filter } P \ \text{zs}) = \text{hd } (\text{filter } P \ \text{xs})$
by (*induct xs arbitrary: n m*) *auto*

lemma (in *ShareReduceRepList-impl*) *ShareReduceRepList-spec-total*:
defines $\text{var-eq} \equiv (\lambda \text{ns } \text{var}. (\forall \text{no1} \in \text{set } \text{ns}. \forall \text{no2} \in \text{set } \text{ns}. \text{no1} \rightarrow \text{var} = \text{no2} \rightarrow \text{var}))$
shows

```

 $\forall \sigma \ \text{ns}. \Gamma \vdash_t$ 
 $\{\sigma. \text{List } \text{'nodeslist } \text{'next } \text{ns} \wedge$ 
 $(\forall \text{no} \in \text{set } \text{ns}.$ 
 $\text{no} \neq \text{Null} \wedge ((\text{no} \rightarrow \text{'low} = \text{Null}) = (\text{no} \rightarrow \text{'high} = \text{Null})) \wedge$ 
 $\text{no} \rightarrow \text{'low} \notin \text{set } \text{ns} \wedge \text{no} \rightarrow \text{'high} \notin \text{set } \text{ns} \wedge$ 
 $(\text{isLeaf-pt } \text{no } \text{'low } \text{'high} = (\text{no} \rightarrow \text{'var} \leq 1)) \wedge$ 
 $(\text{no} \rightarrow \text{'low} \neq \text{Null} \longrightarrow (\text{no} \rightarrow \text{'low}) \rightarrow \text{'rep} \neq \text{Null}) \wedge$ 
 $((\text{'rep} \times \text{'low}) \ \text{no} \notin \text{set } \text{ns})) \wedge$ 
 $\text{var-eq } \text{ns } \text{'var}\}$ 
 $\text{PROC } \text{ShareReduceRepList } (\text{'nodeslist})$ 
 $\llbracket (\forall \text{no}. \text{no} \notin \text{set } \text{ns} \longrightarrow \text{no} \rightarrow^\sigma \text{rep} = \text{no} \rightarrow \text{'rep}) \wedge$ 
 $(\forall \text{no} \in \text{set } \text{ns}. \text{no} \rightarrow \text{'rep} \neq \text{Null} \wedge$ 

```

```

    (if (( $\sigma_{low}$  rep) no = ( $\sigma_{high}$  rep) no  $\wedge$  no  $\rightarrow$   $\sigma_{low}$   $\neq$  Null)
      then (no  $\rightarrow$  rep = ( $\sigma_{low}$  rep) no )
      else ((no  $\rightarrow$  rep)  $\in$  set ns  $\wedge$  no  $\rightarrow$  rep  $\rightarrow$  rep = no  $\rightarrow$  rep  $\wedge$ 
        ( $\forall$  no1  $\in$  set ns.
          (( $\sigma_{high}$  rep) no1 = ( $\sigma_{high}$  rep) no  $\wedge$ 
            ( $\sigma_{low}$  rep) no1 = ( $\sigma_{low}$  rep) no) = (no  $\rightarrow$  rep = no1  $\rightarrow$  rep))))))}
apply (hoare-rule HoareTotal.ProcNoRec1)
apply (hoare-rule anno=
  'node ::= 'nodeslist;;
  WHILE ('node  $\neq$  Null )
  INV { $\exists$  prx sfx. List 'node 'next sfx  $\wedge$ 
    List 'nodeslist 'next ns  $\wedge$  ns=prx@sfx  $\wedge$ 
    ( $\forall$  no  $\in$  set ns.
      no  $\neq$  Null  $\wedge$  ((no  $\rightarrow$   $\sigma_{low}$  = Null) = (no  $\rightarrow$   $\sigma_{high}$  = Null))  $\wedge$ 
      no  $\rightarrow$   $\sigma_{low}$   $\notin$  set ns  $\wedge$  no  $\rightarrow$   $\sigma_{high}$   $\notin$  set ns  $\wedge$ 
      (isLeaf-pt no  $\sigma_{low}$   $\sigma_{high}$  = (no  $\rightarrow$   $\sigma_{var}$   $\leq$  1))  $\wedge$ 
      (no  $\rightarrow$   $\sigma_{low}$   $\neq$  Null  $\rightarrow$  (no  $\rightarrow$   $\sigma_{low}$ )  $\rightarrow$  rep  $\neq$  Null)  $\wedge$ 
      (( $\sigma_{low}$  rep) no  $\notin$  set ns))  $\wedge$ 
      var-eq ns 'var  $\wedge$ 
      ( $\forall$  no. no  $\notin$  set prx  $\rightarrow$  no  $\rightarrow$  rep = no  $\rightarrow$  rep)  $\wedge$ 
      ( $\forall$  no  $\in$  set prx. no  $\rightarrow$  rep  $\neq$  Null  $\wedge$ 
        (if (( $\sigma_{low}$  rep) no = ( $\sigma_{high}$  rep) no  $\wedge$  no  $\rightarrow$   $\sigma_{low}$   $\neq$  Null)
          then (no  $\rightarrow$  rep = ( $\sigma_{low}$  rep) no )
          else ((no  $\rightarrow$  rep)=hd (filter ( $\lambda$ sn. repNodes-eq sn no  $\sigma_{low}$   $\sigma_{high}$  'rep)
            prx)  $\wedge$ 
            ((no  $\rightarrow$  rep)  $\rightarrow$  rep) = no  $\rightarrow$  rep  $\wedge$ 
            ( $\forall$  no1  $\in$  set prx.
              (( $\sigma_{high}$  rep) no1 = ( $\sigma_{high}$  rep) no  $\wedge$ 
                ( $\sigma_{low}$  rep) no1 = ( $\sigma_{low}$  rep) no) =
                (no  $\rightarrow$  rep = no1  $\rightarrow$  rep))))))  $\wedge$ 
            'nodeslist=  $\sigma_{nodeslist}$   $\wedge$  'high= $\sigma_{high}$   $\wedge$  'low= $\sigma_{low}$   $\wedge$  'var= $\sigma_{var}$ }}
  VAR MEASURE (length (list 'node 'next))
  DO
  IF ( $\neg$  isLeaf-pt 'node 'low 'high  $\wedge$ 
    'node  $\rightarrow$  'low  $\rightarrow$  rep = 'node  $\rightarrow$  'high  $\rightarrow$  rep )
  THEN 'node  $\rightarrow$  rep ::= 'node  $\rightarrow$  'low  $\rightarrow$  rep
  ELSE CALL ShareRep ('nodeslist , 'node)
  FI;;
  'node ::= 'node  $\rightarrow$  'next
  OD in HoareTotal.annotateI)
apply (vcg spec=spec-total)
apply (rule-tac x=[] in exI)
apply (rule-tac x=ns in exI)
using [[simp-depth-limit = 2]]
apply (simp (no-asm-use))
prefer 2
using [[simp-depth-limit = 4]]
apply (clarsimp)
prefer 2

```

```

apply (rule conjI)
apply clarify
apply (rule conjI)
apply (clarsimp simp add: List-list)
apply (simp only: List-not-Null simp-thms triv-forall-equality)
apply clarify
apply (simp only: triv-forall-equality)
apply (rename-tac sfx)
apply (rule-tac x=prx@[node] in exI)
apply (rule-tac x=sfx in exI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (simp (no-asm))
apply (rule conjI)
apply (assumption)
prefer 2
apply clarify
apply (simp only: List-not-Null simp-thms triv-forall-equality)
apply clarify
apply (simp only: triv-forall-equality)
apply (rename-tac sfx)
apply (rule-tac x=prx@node#sfx in exI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (rule ballI)
apply (frule-tac x=no in bspec, assumption)
apply (drule-tac x=node in bspec)
apply (simp (no-asm-use))
apply (elim conjE)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (unfold var-eq-def)
apply (drule-tac x=node in bspec, simp)
apply (drule-tac x=no in bspec, assumption)
apply (simp add: isLeaf-pt-def )
apply (rule conjI)
apply (simp (no-asm))
apply (clarify)
apply (rule conjI)
apply (subgoal-tac List node next (node#sfx))
apply (simp only: List-list)
apply (simp (no-asm))
apply (simp (no-asm-simp))
apply (rule-tac x=prx@[node] in exI)
apply (rule-tac x=sfx in exI)

```

```

apply (rule conjI)
apply assumption
apply (rule conjI)
apply (simp (no-asm))
apply (rule conjI)
apply (assumption)
using [[simp-depth-limit = 100]]
proof –
  fix var low high rep nodeslist ns repa next no
  assume ns: List nodeslist next ns
  assume no-in-ns: no ∈ set ns
  assume while-inv: ∀ no ∈ set ns.
    repa no ≠ Null ∧
    (if (repa ∝ low) no = (repa ∝ high) no ∧ high no ≠ Null
     then repa no = (repa ∝ low) no
     else repa no = hd [sn←ns . repNodes-eq sn no low high repa] ∧
     repa (repa no) = repa no ∧
     (∀ no1 ∈ set ns.
      ((repa ∝ high) no1 = (repa ∝ high) no ∧
       (repa ∝ low) no1 = (repa ∝ low) no) =
       (repa no = repa no1)))
  assume pre: ∀ no ∈ set ns.
    no ≠ Null ∧
    (low no = Null) = (high no = Null) ∧
    low no ∉ set ns ∧
    high no ∉ set ns ∧
    isLeaf-pt no low high = (var no ≤ Suc 0) ∧
    (low no ≠ Null → rep (low no) ≠ Null) ∧ (rep ∝ low) no ∉ set ns
  assume same-var: ∀ no1 ∈ set ns. ∀ no2 ∈ set ns. var no1 = var no2
  assume share-case: (repa ∝ low) no = (repa ∝ high) no → high no = Null
  assume unmodif: ∀ no. no ∉ set ns → rep no = repa no
  show hd [sn←ns . repNodes-eq sn no low high repa] ∈ set ns ∧
    repa (hd [sn←ns . repNodes-eq sn no low high repa]) =
    hd [sn←ns . repNodes-eq sn no low high repa]
  proof –
    from no-in-ns pre obtain
      no-nNull: no ≠ Null and
      no-balanced: (low no = Null) = (high no = Null) and
      isLeaf-var: isLeaf-pt no low high = (var no ≤ Suc 0)
    by blast
  have repNodes-eq-same-node: repNodes-eq no no low high repa
  by (simp add: repNodes-eq-def)
  from no-in-ns have ns-nempty: ns ≠ []
  by auto
  from no-in-ns repNodes-eq-same-node
  have repNodes-not-empty: [sn←ns . repNodes-eq sn no low high repa] ≠ []
  by (rule filter-not-empty)
  then have hd-term-in-ns: hd [sn←ns . repNodes-eq sn no low high repa] ∈ set
ns

```



```

    by (rule hd-filter-in-list)
  with while-inv obtain
    repa-hd-nNull: repa (hd [sn←ns . repNodes-eq sn no low high repa]) ≠ Null
  by auto
  let ?hd = hd [sn←ns . repNodes-eq sn no low high repa]
  from hd-term-in-ns pre obtain
    hd-nNull: ?hd ≠ Null and
    hd-balanced:
      (low (hd [sn←ns . repNodes-eq sn no low high repa]) = Null) =
      (high (hd [sn←ns . repNodes-eq sn no low high repa]) = Null) and
    hd-isLeaf-var:
      isLeaf-pt (hd [sn←ns . repNodes-eq sn no low high repa]) low high =
      (var (hd [sn←ns . repNodes-eq sn no low high repa]) ≤ Suc 0)
  by blast
  have repa (hd [sn←ns . repNodes-eq sn no low high repa]) =
    hd [sn←ns . repNodes-eq sn no low high repa]
  proof (cases high no = Null)
  case True
  with no-balanced have low no = Null
  by simp
  with True have no-Leaf: isLeaf-pt no low high
  by (simp add: isLeaf-pt-def)
  with isLeaf-var have varno: var no ≤ 1
  by simp
  from same-var [rule-format, OF no-in-ns hd-term-in-ns] varno
  have var (hd [sn←ns . repNodes-eq sn no low high repa]) ≤ 1
  by simp
  with hd-isLeaf-var have
    isLeaf-pt (hd [sn←ns . repNodes-eq sn no low high repa]) low high
  by simp
  with while-inv hd-term-in-ns repNodes-not-empty show ?thesis
  apply (simp add: isLeaf-pt-def)
  apply (erule-tac x=
    hd [sn←ns . repNodes-eq sn no low high repa] in ballE)
  prefer 2
  apply simp
  apply (simp (no-asm-use) add: repNodes-eq-def)
  apply (rule filter-hd-P-rep-indep)
  apply (simp (no-asm-simp))
  apply (simp (no-asm-simp))
  apply assumption
  done
next
assume hno-nNull: high no ≠ Null
with share-case have repchildren-neq: (repa × low) no ≠ (repa × high) no
  by simp
from repNodes-not-empty have
  repNodes-eq (hd [sn←ns . repNodes-eq sn no low high repa]) no low high
repa

```

```

    by (rule hd-filter-prop)
  then
  have (repa  $\times$  low) (hd [sn $\leftarrow$ ns . repNodes-eq sn no low high repa]) =
    (repa  $\times$  low) no  $\wedge$ 
    (repa  $\times$  high) (hd [sn $\leftarrow$ ns . repNodes-eq sn no low high repa]) =
    (repa  $\times$  high) no
  by (simp add: repNodes-eq-def)
  with repchildren-neq have
    (repa  $\times$  low) (hd [sn $\leftarrow$ ns . repNodes-eq sn no low high repa])
     $\neq$  (repa  $\times$  high) (hd [sn $\leftarrow$ ns . repNodes-eq sn no low high repa])
  by simp
  with while-inv hd-term-in-ns repNodes-not-empty show ?thesis
  apply (simp add: isLeaf-pt-def)
  apply (erule-tac x=
    hd [sn $\leftarrow$ ns . repNodes-eq sn no low high repa] in ballE)
  prefer 2
  apply simp
  apply (simp (no-asm-use) add: repNodes-eq-def)
  apply (rule filter-hd-P-rep-indep)
  apply simp
  apply fastforce
  apply fastforce
  done
qed
with hd-term-in-ns
show ?thesis
by simp
qed
next

```

```

fix var low high rep nodeslist repa next node prx sfx
assume ns: List nodeslist next (prx @ node # sfx)
assume sfx: List (next node) next sfx
assume node-not-Null: node  $\neq$  Null
assume nodes-balanced-ordered:  $\forall no \in \text{set } (prx @ node \# sfx).$ 
  no  $\neq$  Null  $\wedge$ 
  (low no = Null) = (high no = Null)  $\wedge$ 
  low no  $\notin$  set (prx @ node # sfx)  $\wedge$ 
  high no  $\notin$  set (prx @ node # sfx)  $\wedge$ 
  isLeaf-pt no low high = (var no  $\leq$  (1::nat))  $\wedge$ 
  (low no  $\neq$  Null  $\longrightarrow$  rep (low no)  $\neq$  Null)  $\wedge$ 
  (rep  $\times$  low) no  $\notin$  set (prx @ node # sfx)
assume all-nodes-same-var:  $\forall no1 \in \text{set } (prx @ node \# sfx).$ 
   $\forall no2 \in \text{set } (prx @ node \# sfx).$  var no1 = var no2
assume rep-repa-nc:  $\forall no.$  no  $\notin$  set prx  $\longrightarrow$  rep no = repa no
assume while-inv:  $\forall no \in \text{set } prx.$ 
  repa no  $\neq$  Null  $\wedge$ 
  (if (repa  $\times$  low) no = (repa  $\times$  high) no  $\wedge$  low no  $\neq$  Null
  then repa no = (repa  $\times$  low) no

```

else $\text{repa } no = \text{hd } [sn \leftarrow prx . \text{repNodes-eq } sn \ no \ low \ high \ \text{repa}] \wedge$
 $\text{repa } (\text{repa } no) = \text{repa } no \wedge$
 $(\forall no1 \in \text{set } prx.$
 $\quad ((\text{repa } \times \ high) \ no1 = (\text{repa } \times \ high) \ no \wedge$
 $\quad (\text{repa } \times \ low) \ no1 = (\text{repa } \times \ low) \ no) =$
 $\quad (\text{repa } no = \text{repa } no1)))$

assume *not-Leaf*: $\neg \text{isLeaf-pt } node \ low \ high$
assume *repchildren-eq-nln*: $\text{repa } (\text{low } node) = \text{repa } (\text{high } node)$
show $(\forall no. no \notin \text{set } (prx \ @ \ [node]) \longrightarrow$
 $\quad \text{rep } no = (\text{repa}(node := \text{repa } (\text{high } node))) \ no) \wedge$
 $\quad (\forall no \in \text{set } (prx \ @ \ [node]).$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node))) \ no \neq \text{Null} \wedge$
 $\quad (\text{if } (\text{repa}(node := \text{repa } (\text{high } node)) \times \ low) \ no =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)) \times \ high) \ no \wedge$
 $\quad \text{low } no \neq \text{Null}$
 $\quad \text{then } (\text{repa}(node := \text{repa } (\text{high } node))) \ no =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)) \times \ low) \ no$
 $\quad \text{else } (\text{repa}(node := \text{repa } (\text{high } node))) \ no =$
 $\quad \text{hd } [sn \leftarrow prx \ @ \ [node] .$
 $\quad \text{repNodes-eq } sn \ no \ low \ high$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)))] \wedge$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)))$
 $\quad ((\text{repa}(node := \text{repa } (\text{high } node))) \ no) =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node))) \ no \wedge$
 $\quad (\forall no1 \in \text{set } (prx \ @ \ [node]).$
 $\quad ((\text{repa}(node := \text{repa } (\text{high } node)) \times \ high) \ no1 =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)) \times \ high) \ no \wedge$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)) \times \ low) \ no1 =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node)) \times \ low) \ no) =$
 $\quad ((\text{repa}(node := \text{repa } (\text{high } node))) \ no =$
 $\quad (\text{repa}(node := \text{repa } (\text{high } node))) \ no1))))$

(is ?NodesUnmodif \wedge ?NodesModif)
proof –
 — This proof was originally conducted without the substitution $\text{repa } (\text{low } node)$
 $= \text{repa } (\text{high } node)$ preformed. So don't be confused if we show everythin for repa
 $(\text{low } node)$.

from *rep-repa-nc*
have *nodes-unmodif*: *?NodesUnmodif*
by *auto*
hence *rep-Sucna-nc*:
 $(\forall no. no \notin \text{set } (prx \ @ \ [node])$
 $\longrightarrow \text{rep } no = (\text{repa}(node := \text{repa } (\text{low } (node)))) \ no)$
by *auto*
have *nodes-modif*: *?NodesModif* (**is** $\forall no \in \text{set } (prx \ @ \ [node]). \ ?P \ no \ \wedge \ ?Q \ no$)
proof (*rule ballI*)
fix *no*
assume *no-in-take-Sucna*: $no \in \text{set } (prx \ @ \ [node])$
show $\ ?P \ no \ \wedge \ ?Q \ no$
proof (*cases no = node*)

```

case False
note no-noteq-nln=this
with no-in-take-Sucna
have no-in-take-n: no ∈ set prx
  by auto
with no-in-take-n while-inv obtain
  repa-no-nNull: repa no ≠ Null and
  repa-cases: (if (repa ∝ low) no = (repa ∝ high) no ∧ low no ≠ Null
  then repa no = (repa ∝ low) no
  else repa no = hd [sn←prx . repNodes-eq sn no low high repa]
  ∧ repa (repa no) = repa no ∧
  (∀ no1 ∈ set prx. ((repa ∝ high) no1 = (repa ∝ high) no
  ∧ (repa ∝ low) no1 = (repa ∝ low) no
  = (repa no = repa no1)))
  using [[simp-depth-limit = 2]]
  by auto
from no-in-take-n
have no-in-nodeslist: no ∈ set (prx @ node # sfx)
  by auto
from repa-no-nNull no-noteq-nln have ext-repa-nNull: ?P no
  by auto
from no-in-nodeslist nodes-balanced-ordered obtain
  nln-nNull: node ≠ Null and
  nln-balanced-children: (low node = Null) = (high node = Null) and
  lnl-notin-nodeslist: low node ∉ set (prx @ node # sfx) and
  hnl-notin-nodeslist: high node ∉ set (prx @ node # sfx) and
  isLeaf-var-nln: isLeaf-pt node low high = (var node ≤ 1) and
  node-nNull-rap-nNull-nln: (low node ≠ Null
  → rep (low node) ≠ Null) and
  nln-varrep-le-var: (rep ∝ low) node ∉ set (prx @ node # sfx)
  apply (erule-tac x=node in ballE)
  apply auto
  done
from no-in-nodeslist nodes-balanced-ordered no-in-take-Sucna
obtain
  no-nNull: no ≠ Null and
  balanced-children: (low no = Null) = (high no = Null) and
  lno-notin-nodeslist: low no ∉ set (prx @ node # sfx) and
  hno-notin-nodeslist: high no ∉ set (prx @ node # sfx) and
  isLeaf-var-no: isLeaf-pt no low high = (var no ≤ 1) and
  node-nNull-rep-nNull: (low no ≠ Null → rep (low no) ≠ Null) and
  varrep-le-var: (rep ∝ low) no ∉ set (prx @ node # sfx)
  apply –
  apply (erule-tac x=no in ballE)
  apply auto
  done
from lno-notin-nodeslist
have ext-rep-null-comp-low:
  (repa (node := repa (low node)) ∝ low) no = (repa ∝ low) no

```

```

    by (auto simp add: null-comp-def)
  from hno-notin-nodeslist
  have ext-rep-null-comp-high:
    (repa (node := repa (low node))  $\times$  high) no = (repa  $\times$  high) no
    by (auto simp add: null-comp-def)
  have share-reduce-if: ?Q no
  proof (cases (repa (node := repa (low node))  $\times$  low) no =
    (repa (node := repa (low node))  $\times$  high) no  $\wedge$  low no  $\neq$  Null)
  case True
  then obtain
    red-case: (repa (node := repa (low node))  $\times$  low) no =
    (repa (node := repa (low node))  $\times$  high) no and
    lno-nNull: low no  $\neq$  Null
    by simp
  from lno-nNull balanced-children have hno-nNull: high no  $\neq$  Null
    by simp
  from True ext-rep-null-comp-low ext-rep-null-comp-high
  have repchildren-eq-no: (repa  $\times$  low) no = (repa  $\times$  high) no
    by simp
  with repa-cases lno-nNull have repa no = (repa  $\times$  low) no
    by auto
  with ext-rep-null-comp-low no-noteq-nln
  have (repa (node := repa (low node))) no =
    (repa (node := repa (low node))  $\times$  low) no
    by simp
  with True repchildren-eq-nln show ?thesis
    by auto
next
  assume share-case-ext:
     $\neg$  ((repa (node := repa (low node))  $\times$  low) no =
    (repa (node := repa (low node))  $\times$  high) no  $\wedge$  low no  $\neq$  Null)
  from not-Leaf isLeaf-var-nln
  have 1 < var node
    by simp
  with all-nodes-same-var
  have all-nodes-nl-Suc0-l-var:  $\forall x \in \text{set } (\text{prx} @ \text{node} \# \text{sfx}). 1 < \text{var } x$ 
    using [[simp-depth-limit=1]]
    by auto
  with nodes-balanced-ordered
  have all-nodes-nl-noLeaf:
     $\forall x \in \text{set } (\text{prx} @ \text{node} \# \text{sfx}). \neg \text{isLeaf-pt } x \text{ low high}$ 
    apply  $-$ 
    apply rule
    apply (drule-tac x=x in bspec, assumption)
    apply (drule-tac x=x in bspec, assumption)
    apply auto
  done
  from nodes-balanced-ordered
  have all-nodes-nl-balanced:

```

```

     $\forall x \in \text{set } (\text{prx} @ \text{node} \# \text{sfx}). (\text{low } x = \text{Null}) = (\text{high } x = \text{Null})$ 
    apply -
    apply rule
    apply (drule-tac x=x in bspec,assumption)
    apply auto
    done
  from all-nodes-nl-Suc0-l-var no-in-nodeslist
  have Suc0-l-var-no: 1 < var no
    by auto
  with isLeaf-var-no have no-nLeaf:  $\neg \text{isLeaf-pt } no \text{ low high}$ 
    by simp
  with balanced-children have lno-nNull:  $\text{low } no \neq \text{Null}$ 
    by (simp add: isLeaf-pt-def)
  with balanced-children have hno-nNull:  $\text{high } no \neq \text{Null}$ 
    by simp
  with share-case-ext ext-rep-null-comp-low ext-rep-null-comp-high lno-nNull

  have repchildren-neq-no:  $(\text{repa} \times \text{low}) \text{ no} \neq (\text{repa} \times \text{high}) \text{ no}$ 
    by (simp add: null-comp-def)
  with repa-cases
  have share-case-inv:
    repa no = hd [sn←prx . repNodes-eq sn no low high repa]  $\wedge$ 
    repa (repa no) = repa no  $\wedge$ 
    ( $\forall no1 \in \text{set } \text{prx}. ((\text{repa} \times \text{high}) \text{ no1} = (\text{repa} \times \text{high}) \text{ no}) \wedge$ 
    ( $\text{repa} \times \text{low}) \text{ no1} = (\text{repa} \times \text{low}) \text{ no}) = (\text{repa } no = \text{repa } no1))$ 
    by auto
  then have repa-no: repa no = hd [sn←prx . repNodes-eq sn no low high
repa]
    by simp
  from Suc0-l-var-no have  $\forall x \in \text{set } (\text{prx} @ \text{node} \# \text{sfx}). 1 < \text{var } no$ 
    by auto
  from no-in-take-n have [sn←prx . repNodes-eq sn no low high repa]  $\neq []$ 
    apply -
    apply (rule filter-not-empty)
    apply (auto simp add: repNodes-eq-def)
    done
  then have repNodes-eq
    (hd [sn←prx . repNodes-eq sn no low high repa]) no low high repa
    by (rule hd-filter-prop)
  with repa-no
  have rep-children-eq-no-repa-no:
     $(\text{repa} \times \text{low}) (\text{repa } no) = (\text{repa} \times \text{low}) \text{ no} \wedge$ 
     $(\text{repa} \times \text{high}) (\text{repa } no) = (\text{repa} \times \text{high}) \text{ no}$ 
    by (simp add: repNodes-eq-def)
  from lno-notin-nodeslist rep-repa-nc
  have rep-repa-nc-low-no:  $\text{rep } (\text{low } no) = \text{repa } (\text{low } no)$ 
    apply -
    apply (erule-tac x=low no in allE)
    apply auto

```

```

done
have  $\forall x \in \text{set } (\text{prx} @ [\text{node}])$ .
   $\text{repNodes-eq } x \text{ no low high } (\text{repa}(\text{node} := \text{repa } (\text{low node}))) =$ 
   $\text{repNodes-eq } x \text{ no low high repa}$ 
proof (rule ballI, unfold repNodes-eq-def)
  fix x
  assume x-in-take-Sucn:  $x \in \text{set } (\text{prx} @ [\text{node}])$ 
  hence x-in-nodelist:  $x \in \text{set } (\text{prx} @ \text{node} \# \text{sfx})$ 
  by auto
  with all-nodes-nl-noLeaf nodes-balanced-ordered
  have children-nNull-x:  $\text{low } x \neq \text{Null} \wedge \text{high } x \neq \text{Null}$ 
  apply -
  apply (drule-tac x=x in bspec.assumption)
  apply (drule-tac x=x in bspec.assumption)
  apply (auto simp add: isLeaf-pt-def)
  done
from x-in-nodelist nodes-balanced-ordered
have  $\text{low } x \notin \text{set } (\text{prx} @ \text{node} \# \text{sfx}) \wedge \text{high } x \notin \text{set } (\text{prx} @ \text{node} \# \text{sfx})$ 
  apply -
  apply (drule-tac x=x in bspec.assumption)
  apply auto
  done
with lno-notin-nodelist hno-notin-nodelist
  children-nNull-x lno-nNull hno-nNull
show  $((\text{repa}(\text{node} := \text{repa } (\text{low node})) \times \text{high}) x =$ 
   $(\text{repa}(\text{node} := \text{repa } (\text{low node})) \times \text{high}) \text{ no} \wedge$ 
   $(\text{repa}(\text{node} := \text{repa } (\text{low node})) \times \text{low}) x =$ 
   $(\text{repa}(\text{node} := \text{repa } (\text{low node})) \times \text{low}) \text{ no}) =$ 
   $((\text{repa} \times \text{high}) x = (\text{repa} \times \text{high}) \text{ no} \wedge$ 
   $(\text{repa} \times \text{low}) x = (\text{repa} \times \text{low}) \text{ no})$ 
  by (simp add: null-comp-def)
qed
then have filter-extrep-rep:
   $[\text{sn} \leftarrow (\text{prx} @ [\text{node}]). \text{repNodes-eq } \text{sn} \text{ no low high}$ 
   $(\text{repa}(\text{node} := \text{repa } (\text{low node})))] =$ 
   $[\text{sn} \leftarrow (\text{prx} @ [\text{node}]) . \text{repNodes-eq } \text{sn} \text{ no low high repa}]$ 
  by (rule P-eq-list-filter)
from no-in-take-n
have filter-n-notempty:  $[\text{sn} \leftarrow \text{prx}. \text{repNodes-eq } \text{sn} \text{ no low high repa}] \neq []$ 
  apply (rule filter-not-empty)
  apply (simp add: repNodes-eq-def)
  done
then have hd  $[\text{sn} \leftarrow \text{prx}. \text{repNodes-eq } \text{sn} \text{ no low high repa}] =$ 
   $\text{hd } [\text{sn} \leftarrow \text{prx} @ [\text{node}]. \text{repNodes-eq } \text{sn} \text{ no low high repa}]$ 
  by auto
with no-noteq-nln filter-extrep-rep repa-no
have ext-repa-no:  $(\text{repa}(\text{node} := \text{repa } (\text{low node}))) \text{ no} =$ 
   $\text{hd } [\text{sn} \leftarrow \text{prx} @ [\text{node}]. \text{repNodes-eq } \text{sn} \text{ no low high}$ 
   $(\text{repa}(\text{node} := \text{repa } (\text{low node})))]$ 

```

```

by simp
have (repa(node := repa (low node))) (repa no) = repa no
proof (cases repa no = node)
  case True
  note rno-nln=this
  from rep-repa-nc-low-no rep-children-eq-no-repa-no lno-nNull
    node-nNull-rep-nNull
  have low-rep-no-nNull: low (repa no) ≠ Null
  apply (simp add: null-comp-def)
  apply auto
  done
  with nodes-balanced-ordered rno-nln
  have high-rap-no-nNull: high (repa no) ≠ Null
  apply -
  apply (drule-tac x=repa no in bspec)
  apply auto
  done
  with low-rep-no-nNull rno-nln rep-children-eq-no-repa-no
  have repa (low node) = (repa ∘ low) no ∧
    repa (high node) = (repa ∘ high) no
  by (simp add: null-comp-def)
  with repchildren-eq-nln have (repa ∘ low) no = (repa ∘ high) no
  by simp
  with repchildren-neq-no show ?thesis
  by simp
next
assume rno-not-nln: repa no ≠ node
from share-case-inv have repa (repa no) = repa no
  by auto
with rno-not-nln show ?thesis
  by simp
qed
with no-noteq-nln have ext-repa-ext-repa:
  (repa(node := repa (low node)))
  ((repa(node := repa (low node))) no)
  = (repa(node := repa (low node))) no
  by simp
have (∀ no1 ∈ set (prx@[node]).
  ((repa(node := repa (low node)) ∘ high) no1 =
  (repa(node := repa (low node)) ∘ high) no ∧
  (repa(node := repa (low node)) ∘ low) no1 =
  (repa(node := repa (low node)) ∘ low) no) =
  ((repa(node := repa (low node))) no =
  (repa(node := repa (low node))) no1))
proof (rule ballI)
  fix no1
  assume no1-in-take-Sucn: no1 ∈ set (prx@[node])
  hence no1-in-nodeslist: no1 ∈ set (prx @ node # sfx)
  by auto

```



```

show ((repa(node := repa (low node))  $\times$  high) no1 =
  (repa(node := repa (low node))  $\times$  high) no  $\wedge$ 
  (repa(node := repa (low node))  $\times$  low) no1 =
  (repa(node := repa (low node))  $\times$  low) no) =
  ((repa(node := repa (low node))) no =
  (repa(node := repa (low node))) no1)
proof (cases no1 = node)
  case True
  show ?thesis
  proof (rule, elim conjE)
    assume ext-repa-no-no1:
      (repa(node := repa (low node))) no =
      (repa(node := repa (low node))) no1
    with True no-noteq-nln
    have repa-no-repa-low-nln: repa no = repa (low node)
      by simp
    from filter-n-notempty
    have repa-no-in-take-n:
      hd [sn←prx. repNodes-eq sn no low high repa]
       $\in$  set prx
    apply -
    apply (rule hd-filter-in-list)
    apply auto
    done
  with repa-no
  have repa-no-in-nodeslist: repa no  $\in$  set (prx @ node # sfx)
    by auto
  from lnl-n-notin-nodeslist rep-repa-nc
  have rep-repa-low-nln: rep (low node) = repa (low node)
    by auto
  from all-nodes-nl-noLeaf nln-balanced-children
  have low node  $\neq$  Null
    by (auto simp add: isLeaf-pt-def)
  with rep-repa-low-nln lnl-n-notin-nodeslist lno-nNull
    nln-varrep-le-var
  have repa (low node)  $\notin$  set (prx @ node # sfx)
    by (simp add: null-comp-def)
  with repa-no-repa-low-nln repa-no-in-nodeslist
  show (repa(node := repa (low node))  $\times$  high) no1 =
    (repa(node := repa (low node))  $\times$  high) no  $\wedge$ 
    (repa(node := repa (low node))  $\times$  low) no1 =
    (repa(node := repa (low node))  $\times$  low) no
    by simp
  next
  assume no-no1-high:
    (repa(node := repa (low node))  $\times$  high) no1 =
    (repa(node := repa (low node))  $\times$  high) no
  assume no-no1-low:
    (repa(node := repa (low node))  $\times$  low) no1 =

```

```

    (repa(node := repa (low node))  $\times$  low) no
  from True repchildren-eq-nln
  have repachildren-eq-no1: repa (low no1) = repa (high no1)
    by simp
  from not-Leaf True nln-balanced-children
  have children-nNull-no1: (low no1)  $\neq$  Null  $\wedge$  high no1  $\neq$  Null
    by (simp add: isLeaf-pt-def)
  with repachildren-eq-no1
  have repchildren-eq-no1: (repa  $\times$  low) no1 = (repa  $\times$  high) no1
    by (simp add: null-comp-def)
  from no-no1-low children-nNull-no1 lno-nNull
    lnl-notin-nodeslist lno-notin-nodeslist True
  have rep-low-eq-no-no1: (repa  $\times$  low) no1 = (repa  $\times$  low) no
    by (simp add: null-comp-def)
  from no-no1-high children-nNull-no1 hno-nNull
    hnl-notin-nodeslist hno-notin-nodeslist True
  have rep-high-eq-no-no1: (repa  $\times$  high) no1 = (repa  $\times$  high) no
    by (simp add: null-comp-def)
  with rep-low-eq-no-no1 repchildren-eq-no1
  have (repa  $\times$  low) no = (repa  $\times$  high) no
    by simp
  with repchildren-neq-no
  show (repa(node := repa (low node))) no =
    (repa(node := repa (low node))) no1
    by simp
qed
next
assume no1-neq-nln: no1  $\neq$  node
from no1-in-nodeslist nodes-balanced-ordered
have children-notin-nl-no1:
low no1  $\notin$  set (prx @ node # sfx)  $\wedge$  high no1  $\notin$  set (prx @ node # sfx)
  apply -
  apply (drule-tac x=no1 in bspec,assumption)
  by auto
from no1-neq-nln no1-in-take-Sucn
have no1-in-take-n: no1  $\in$  set prx
  by auto
from no1-in-nodeslist all-nodes-nl-noLeaf all-nodes-nl-balanced
have children-nNull-no1: (low no1)  $\neq$  Null  $\wedge$  high no1  $\neq$  Null
  by (auto simp add: isLeaf-pt-def)
show ?thesis
proof (rule, elim conjE)
  assume ext-repa-high-no1-no:
    (repa(node := repa (low node))  $\times$  high) no1
    = (repa(node := repa (low node))  $\times$  high) no
  assume ext-repa-low-no1-no:
    (repa(node := repa (low node))  $\times$  low) no1
    = (repa(node := repa (low node))  $\times$  low) no
  from children-nNull-no1 hno-nNull ext-repa-high-no1-no

```

```

    children-notin-nl-no1
    hno-notin-nodeslist
  have repa-high-no1-no: (repa  $\times$  high) no1 = (repa  $\times$  high) no
    by (simp add: null-comp-def)
  from children-nNull-no1 lno-nNull ext-repa-low-no1-no
    children-notin-nl-no1 lno-notin-nodeslist
  have repa-low-no1-no: (repa  $\times$  low) no1 = (repa  $\times$  low) no
    by (simp add: null-comp-def)
  from repchildren-neq-no repa-high-no1-no repa-low-no1-no
  have (repa  $\times$  low) no1  $\neq$  (repa  $\times$  high) no1
    by simp
  from no1-in-take-n share-case-inv repa-high-no1-no repa-low-no1-no
  have repa no = repa no1
    by auto
  with no-noteq-nln no1-neq-nln
  show (repa(node := repa (low node))) no =
    (repa(node := repa (low node))) no1
    by simp
next
  assume (repa(node := repa (low node))) no =
    (repa(node := repa (low node))) no1
  with no-noteq-nln no1-neq-nln
  have repa no = repa no1
    by simp
  with share-case-inv no1-in-take-n
  have ((repa  $\times$  high) no1 = (repa  $\times$  high) no  $\wedge$ 
    (repa  $\times$  low) no1 = (repa  $\times$  low) no)
    by auto
  with children-notin-nl-no1 children-nNull-no1 lno-notin-nodeslist
    hno-notin-nodeslist lno-nNull hno-nNull
  show (repa(node := repa (low node))  $\times$  high) no1 =
    (repa(node := repa (low node))  $\times$  high) no  $\wedge$ 
    (repa(node := repa (low node))  $\times$  low) no1 =
    (repa(node := repa (low node))  $\times$  low) no
    by (auto simp add: null-comp-def)
  qed
  qed
  from ext-repa-ext-repa ext-repa-no share-case-ext repchildren-eq-nln this
  show ?thesis
    using [[simp-depth-limit=4]]
    by auto
  qed
  with ext-repa-nNull show ?thesis
    by auto
next
  assume no-nln: no = node
  hence no-in-nodeslist: no  $\in$  set (prx @ node # sfx)
    by simp

```

```

from no-nln not-Leaf no-in-nodeslist
  nodes-balanced-ordered [rule-format, OF this] obtain
  low-no-nNull: low no  $\neq$  Null and
  high-no-nNull: high no  $\neq$  Null and
  rep-low-no-nNull: rep (low no)  $\neq$  Null and
  lno-notin-nl: low no  $\notin$  set (prx @ node # sfx) and
  hno-notin-nl: high no  $\notin$  set (prx @ node # sfx) and
  children-nNull-no: (low no  $\neq$  Null)  $\wedge$  (high no  $\neq$  Null)
apply (unfold isLeaf-pt-def)
apply blast
done
then have low no  $\notin$  set prx
  by auto
with rep-repa-nc no-nln rep-low-no-nNull
have (repa(node := repa (low node)) no  $\neq$  Null)
  by simp
moreover
have (if (repa(node := repa (low node))  $\times$  low) no =
  (repa(node := repa (low node))  $\times$  high) no  $\wedge$  low no  $\neq$  Null)
  then (repa(node := repa (low node)) no =
  (repa(node := repa (low node))  $\times$  low) no
  else (repa(node := repa (low node)) no =
  hd [sn $\leftarrow$ prx@[node]. repNodes-eq sn no low high
  (repa(node := repa (low node)))]  $\wedge$ 
  (repa(node := repa (low node))))
  ((repa(node := repa (low node)) no) =
  (repa(node := repa (low node)) no)  $\wedge$ 
  ( $\forall no1 \in$  set (prx@[node])).
  ((repa(node := repa (low node))  $\times$  high) no1 =
  (repa(node := repa (low node))  $\times$  high) no  $\wedge$ 
  (repa(node := repa (low node))  $\times$  low) no1 =
  (repa(node := repa (low node))  $\times$  low) no) =
  ((repa(node := repa (low node)) no =
  (repa(node := repa (low node)) no1)))
proof (cases (repa(node := repa (low node))  $\times$  low) no =
  (repa(node := repa (low node))  $\times$  high) no  $\wedge$  low no  $\neq$  Null)
  case True
  note red-case=this
  with children-nNull-no lno-notin-nl hno-notin-nl
  have (repa  $\times$  low) no = (repa  $\times$  high) no
  by (auto simp add: null-comp-def)
  from children-nNull-no lno-notin-nl
  have ext-repa-eq-repa-low: (repa(node := repa (low node))  $\times$  low) no
  = (repa  $\times$  low) no
  by (auto simp add: null-comp-def)
  from children-nNull-no hno-notin-nl
  have ext-repa-eq-repa-high:
  (repa(node := repa (low node))  $\times$  high) no
  = (repa  $\times$  high) no

```

```

    by (auto simp add: null-comp-def)
  from no-nln children-nNull-no
  have repa (low node) = (repa  $\times$  low) no
    by (simp add: null-comp-def)
  with red-case ext-repa-eq-repa-high ext-repa-eq-repa-low no-nln
  show ?thesis
    using [[simp-depth-limit=2]]
    by (auto simp del: null-comp-not-Null)
next
  assume share-case:  $\neg ((\text{repa}(\text{node} := \text{repa}(\text{low node})) \times \text{low}) \text{ no} = (\text{repa}(\text{node} := \text{repa}(\text{low node})) \times \text{high}) \text{ no} \wedge \text{low no} \neq \text{Null})$ 
  with low-no-nNull have (repa(node := repa (low node))  $\times$  low) no  $\neq$  (repa(node := repa (low node))  $\times$  high) no
    by simp
  with children-nNull-no lno-notin-nl hno-notin-nl
  have (repa  $\times$  low) no  $\neq$  (repa  $\times$  high) no
    by (auto simp add: null-comp-def)
  with children-nNull-no have repa (low no)  $\neq$  repa (high no)
    by (simp add: null-comp-def)
  with repchildren-eq-nln no-nln show ?thesis
    by simp
qed
ultimately show ?thesis
  using repchildren-eq-nln
  apply -
  apply (simp only:)
  apply (simp (no-asm))
  done
qed
qed
from nodes-unmodif nodes-modif
show ?thesis by iprover
qed
next
fix var low high rep nodeslist repa next node prx sfx repb
assume ns: List nodeslist next (prx @ node # sfx)
assume sfx: List (next node) next sfx
assume nodes-balanced-ordered:  $\forall \text{no} \in \text{set} (\text{prx} @ \text{node} \# \text{sfx}). \text{no} \neq \text{Null} \wedge (\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \wedge \text{low no} \notin \text{set} (\text{prx} @ \text{node} \# \text{sfx}) \wedge \text{high no} \notin \text{set} (\text{prx} @ \text{node} \# \text{sfx}) \wedge \text{isLeaf-pt no low high} = (\text{var no} \leq (1::\text{nat})) \wedge (\text{low no} \neq \text{Null} \longrightarrow \text{rep}(\text{low no}) \neq \text{Null}) \wedge (\text{rep} \times \text{low}) \text{ no} \notin \text{set} (\text{prx} @ \text{node} \# \text{sfx})$ 
assume all-nodes-same-var:  $\forall \text{no1} \in \text{set} (\text{prx} @ \text{node} \# \text{sfx}). \forall \text{no2} \in \text{set} (\text{prx} @ \text{node} \# \text{sfx}). \text{var no1} = \text{var no2}$ 
assume rep-repa-nc:  $\forall \text{no}. \text{no} \notin \text{set prx} \longrightarrow \text{rep no} = \text{repa no}$ 
assume while-inv:  $\forall \text{no} \in \text{set prx}. \text{no} \notin \text{set prx} \longrightarrow \text{rep no} = \text{repa no}$ 

```

$repa\ no \neq\ Null \wedge$
 $(if\ (repa \times low)\ no = (repa \times high)\ no \wedge low\ no \neq\ Null$
 $then\ repa\ no = (repa \times low)\ no$
 $else\ repa\ no = hd\ [sn \leftarrow prx . repNodes-eq\ sn\ no\ low\ high\ repa] \wedge$
 $repa\ (repa\ no) = repa\ no \wedge$
 $(\forall\ no1 \in set\ prx.$
 $((repa \times high)\ no1 = (repa \times high)\ no \wedge$
 $(repa \times low)\ no1 = (repa \times low)\ no) =$
 $(repa\ no = repa\ no1)))$

assume share-cond:
 $\neg (\neg\ isLeaf-pt\ node\ low\ high \wedge repa\ (low\ node) = repa\ (high\ node))$

assume repb-node:
 $repb\ node = hd\ [sn \leftarrow prx \ @\ node \ \# \ sfx . repNodes-eq\ sn\ node\ low\ high\ repa]$

assume repa-repb-nc: $\forall\ pt. pt \neq\ node \longrightarrow repa\ pt = repb\ pt$

assume var-repb-node: $var\ (repb\ node) = var\ node$

show $(\forall\ no. no \notin set\ (prx \ @\ [node]) \longrightarrow rep\ no = repb\ no) \wedge$
 $(\forall\ no \in set\ (prx \ @\ [node]).$
 $repb\ no \neq\ Null \wedge$
 $(if\ (repb \times low)\ no = (repb \times high)\ no \wedge low\ no \neq\ Null$
 $then\ repb\ no = (repb \times low)\ no$
 $else\ repb\ no =$
 $hd\ [sn \leftarrow prx \ @\ [node] . repNodes-eq\ sn\ no\ low\ high\ repb] \wedge$
 $repb\ (repb\ no) = repb\ no \wedge$
 $(\forall\ no1 \in set\ (prx \ @\ [node]).$
 $((repb \times high)\ no1 = (repb \times high)\ no \wedge$
 $(repb \times low)\ no1 = (repb \times low)\ no) =$
 $(repb\ no = repb\ no1))))$

proof –

have $rep-repb-nc: (\forall\ no. no \notin set\ (prx \ @\ [node]) \longrightarrow rep\ no = repb\ no)$

proof (*intro allI impI*)

fix no

assume $no-notin-take-Sucn: no \notin set\ (prx \ @\ [node])$

with $rep-repa-nc$

have $rep-repa-nc-Sucn: rep\ no = repa\ no$

by *auto*

from $no-notin-take-Sucn$ **have** $no \neq\ node$

by *auto*

with $rep-repb-nc$ **have** $repa\ no = repb\ no$

by *auto*

with $rep-repa-nc-Sucn$ **show** $rep\ no = repb\ no$

by *simp*

qed

moreover

have $repb-no-share-def:$
 $(\forall\ no \in set\ (prx \ @\ [node]).$
 $\neg ((repb \times low)\ no = (repb \times high)\ no \wedge low\ no \neq\ Null) \longrightarrow$
 $repb\ no = hd\ [sn \leftarrow (prx \ @\ [node]) . repNodes-eq\ sn\ no\ low\ high\ repb])$

proof (*intro ballI impI*)

fix no

```

assume no-in-take-Sucn:  $no \in \text{set } (prx \text{ @ } [node])$ 
assume share-prop:  $\neg ((repb \times low) no = (repb \times high) no \wedge low no \neq Null)$ 
from share-prop have share-or:
   $(repb \times low) no \neq (repb \times high) no \vee low no = Null$ 
  using [[simp-depth-limit=2]]
  by simp
from no-in-take-Sucn have no-in-nl:  $no \in \text{set } (prx \text{ @ } node \# sfx)$ 
  by auto
from nodes-balanced-ordered [rule-format, OF this] obtain
  no-nNull:  $no \neq Null$  and
  balanced-no:  $(low no = Null) = (high no = Null)$  and
  lno-notin-nl:  $low no \notin \text{set } (prx \text{ @ } node \# sfx)$  and
  hno-notin-nl:  $high no \notin \text{set } (prx \text{ @ } node \# sfx)$  and
  isLeaf-var-no:  $isLeaf-pt no low high = (var no \leq 1)$ 
  by auto
have nodes-notin-nl-neg-nln:  $\forall p. p \notin \text{set } (prx \text{ @ } node \# sfx) \longrightarrow p \neq node$ 
  by auto
show  $repb no = hd [sn \leftarrow (prx \text{ @ } [node]). repNodes-eq sn no low high repb]$ 
proof (cases no = node)
  case False
  note no-notin-nl=this
  with no-in-take-Sucn have no-in-take-n:  $no \in \text{set } prx$ 
    by auto
  from False repa-repb-nc have repb-repa-no:  $repb no = repa no$ 
    by auto
  with while-inv [rule-format, OF no-in-take-n] no-in-take-n obtain
    repa-no-nNull:  $repa no \neq Null$  and
    while-share-red-exp:
      (if  $(repa \times low) no = (repa \times high) no \wedge low no \neq Null$ 
        then  $repa no = (repa \times low) no$ 
        else  $repa no = hd [sn \leftarrow prx . repNodes-eq sn no low high repa] \wedge$ 
           $repa (repa no) = repa no \wedge$ 
           $(\forall no1 \in \text{set } prx. ((repa \times high) no1 = (repa \times high) no \wedge$ 
             $(repa \times low) no1 = (repa \times low) no) = (repa no = repa no1)))$ )
    using [[simp-depth-limit = 2]]
    by auto
  from no-in-take-n
  have filter-take-n-notempty:  $[sn \leftarrow prx.$ 
     $repNodes-eq sn no low high repa] \neq []$ 
    apply –
    apply (rule filter-not-empty)
    apply (auto simp add: repNodes-eq-def)
    done
  then have hd-term-n-Sucn:
     $hd [sn \leftarrow prx. repNodes-eq sn no low high repa]$ 
     $= hd [sn \leftarrow prx @ [node] . repNodes-eq sn no low high repa]$ 
    by auto
  thus ?thesis
proof (cases low no = Null)

```

```

case True
note lno-Null=this
with balanced-no have hno-Null: high no = Null
  by simp
from lno-Null hno-Null have isLeaf-no: isLeaf-pt no low high
  by (simp add: isLeaf-pt-def)
from True while-share-red-exp
have while-low-Null:
  repa no = hd [sn←prx. repNodes-eq sn no low high repa] ∧
  repa (repa no) = repa no ∧
  (∀ no1 ∈ set prx. ((repa ∝ high) no1 = (repa ∝ high) no
  ∧ (repa ∝ low) no1 = (repa ∝ low) no) = (repa no = repa no1))
  by auto
have all-nodes-in-nl-Leafs:
  ∀ x ∈ set (prx @ node # sfx). isLeaf-pt x low high
proof (intro ballI)
  fix x
  assume x-in-nodeslist: x ∈ set (prx @ node # sfx)
  from isLeaf-no isLeaf-var-no have var no ≤ 1
  by simp
  with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
  have var x ≤ 1
  by simp
  with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
  show isLeaf-pt x low high
  by (auto simp add: isLeaf-pt-def)
qed
have ∀ x ∈ set (prx@[node]). repNodes-eq x no low high repb
  = repNodes-eq x no low high repa
proof (rule ballI)
  fix x
  assume x-in-take-Sucn: x ∈ set (prx@[node])
  hence x-in-nodeslist: x ∈ set (prx @ node # sfx)
  by auto
  with all-nodes-in-nl-Leafs have isLeaf-pt x low high
  by auto
  with isLeaf-no repa-repb-nc show repNodes-eq x no low high repb
  = repNodes-eq x no low high repa
  by (simp add: repNodes-eq-def null-comp-def isLeaf-pt-def)
qed
then have [sn←(prx@[node]). repNodes-eq sn no low high repa]
  = [sn←(prx@[node]) . repNodes-eq sn no low high repb]
  apply -
  apply (rule P-eq-list-filter)
  apply simp
  done
with hd-term-n-Sucn while-low-Null repb-repa-no show ?thesis
  by auto
next

```



```

assume lno-nNull: low no ≠ Null
with balanced-no have hno-nNull: high no ≠ Null
  by simp
with lno-nNull have no-nLeaf: ¬ isLeaf-pt no low high
  by (simp add: isLeaf-pt-def)
with isLeaf-var-no have Sucn-s-varno: 1 < var no
  by auto
with no-in-nl all-nodes-same-var
have all-nodes-nl-var: ∀ x ∈ set (prx @ node # sfx). 1 < var x
  apply -
  apply (rule ballI)
  apply (drule-tac x=no in bspec,assumption)
  apply (drule-tac x=x in bspec,assumption)
  apply auto
  done
with nodes-balanced-ordered
have all-nodes-nl-nLeaf:
  ∀ x ∈ set (prx @ node # sfx). ¬ isLeaf-pt x low high
  apply -
  apply (rule ballI)
  apply (drule-tac x=x in bspec,assumption)
  apply (drule-tac x=x in bspec,assumption)
  apply auto
  done
from lno-nNull share-or
have repbchildren-eq-no: (repb ∝ low) no ≠ (repb ∝ high) no
  by simp
with lno-nNull hno-nNull lno-notin-nl hno-notin-nl repa-repb-nc
  nodes-notin-nl-neq-nln
have repachildren-eq-no: (repa ∝ low) no ≠ (repa ∝ high) no
  using [[simp-depth-limit=2]]
  by (simp add: null-comp-def)
with while-share-red-exp
have repa-no-def:
  repa no = hd [sn←prx . repNodes-eq sn no low high repa]
  by auto
with no-notin-nl repa-repb-nc
have repb no = hd [sn←prx. repNodes-eq sn no low high repa]
  by simp
with hd-term-n-Sucn
have repb-no-hd-term-repa: repb no =
  hd [sn←prx@[node] . repNodes-eq sn no low high repa]
  by simp
have ∀ x ∈ set (prx@[node]).
  repNodes-eq x no low high repa = repNodes-eq x no low high repb
proof (intro ballI)
  fix x
  assume x-in-take-Sucn: x ∈ set (prx@[node])
  hence x-in-nodeslist: x ∈ set (prx @ node # sfx)

```

```

    by auto
  with all-nodes-nl-nLeaf have x-nLeaf:  $\neg$  isLeaf-pt x low high
    by auto
  from nodes-balanced-ordered [rule-format, OF x-in-nodeslist] obtain
    balanced-x: (low x = Null) = (high x = Null) and
    lx-notin-nl: low x  $\notin$  set (prx @ node # sfx) and
    hx-notin-nl: high x  $\notin$  set (prx @ node # sfx)
    by auto
  with nodes-notin-nl-neq-nln lno-notin-nl hno-notin-nl lno-nNull
    hno-nNull repa-repb-nc
  show repNodes-eq x no low high repa = repNodes-eq x no low high repb
    by (simp add: repNodes-eq-def null-comp-def)
qed
then have [sn $\leftarrow$ (prx@[node]). repNodes-eq sn no low high repa] =
  [sn $\leftarrow$ (prx@[node]). repNodes-eq sn no low high repb]
  apply -
  apply (rule P-eq-list-filter)
  apply auto
  done
with repb-no-hd-term-repa show ?thesis
  by simp
qed
next
assume no-nln: no = node
with repb-node have repb-no-def: repb no =
  hd [sn $\leftarrow$ (prx @ node # sfx). repNodes-eq sn node low high repa]
  by simp
show ?thesis
proof (cases isLeaf-pt no low high)
  case True
  note isLeaf-no=this
  have  $\forall x \in$  set (prx @ node # sfx). repNodes-eq x no low high repb
    = repNodes-eq x no low high repa
  proof (rule ballI)
    fix x
    assume x-in-nodeslist:  $x \in$  set (prx @ node # sfx)
    have all-nodes-in-nl-Leafs:
       $\forall x \in$  set (prx @ node # sfx). isLeaf-pt x low high
    proof (intro ballI)
      fix x
      assume x-in-nodeslist:  $x \in$  set (prx @ node # sfx)
      from isLeaf-no isLeaf-var-no have var no  $\leq$  1
        by simp
      with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
      have var x  $\leq$  1
        by simp
      with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
      show isLeaf-pt x low high
        by (auto simp add: isLeaf-pt-def)
    
```

```

qed
with x-in-nodeslist have isLeaf-pt x low high
  by auto
with isLeaf-no repa-repb-nc
show repNodes-eq x no low high repb = repNodes-eq x no low high repa
  by (simp add: repNodes-eq-def null-comp-def isLeaf-pt-def)
qed
with repb-no-def no-nln have repb-no-whole-nl: repb no =
  hd [sn← (prx @ node # sfx). repNodes-eq sn node low high repb]
apply -
apply (subgoal-tac
  [sn← (prx@node#sfx). repNodes-eq sn node low high repa]
  = [sn←(prx @ node # sfx) . repNodes-eq sn node low high repb])
apply simp
apply (rule P-eq-list-filter)
apply auto
done
from no-in-take-Sucn no-nln
have [sn← (prx@[node]). repNodes-eq sn node low high repb] ≠ []
  apply -
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
done
then
have hd [sn←(prx@[node]). repNodes-eq sn node low high repb] =
  hd [sn←(prx @ node # sfx). repNodes-eq sn node low high repb]
  apply -
  apply (rule hd-filter-app [symmetric])
  apply auto
done
with repb-no-whole-nl no-nln show ?thesis
  by simp
next
assume no-nLeaf: ¬ isLeaf-pt no low high
with share-or balanced-no have (repb ∝ low) no ≠ (repb ∝ high) no
  using [[simp-depth-limit=2]]
  by (simp add: isLeaf-pt-def)
from no-nLeaf share-cond no-nln have repa (low no) ≠ repa (high no)
  by auto
with no-nLeaf balanced-no have (repa ∝ low) no ≠ (repa ∝ high) no
  by (simp add: null-comp-def isLeaf-pt-def)
have  $\forall x \in \text{set } (prx@node\#sfx). repNodes-eq x no low high repb$ 
  = repNodes-eq x no low high repa
proof (rule ballI)
  fix x
  assume x-in-nodeslist: x ∈ set (prx@node#sfx)
  have all-nodes-in-nl-Leafs:
     $\forall x \in \text{set } (prx@node\#sfx). \neg isLeaf-pt x low high$ 
  proof (intro ballI)

```

```

fix  $x$ 
assume  $x$ -in-nodeslist:  $x \in \text{set } (\text{prx}@node\#sfx)$ 
from no-nLeaf isLeaf-var-no have  $1 < \text{var } no$ 
  by simp
with all-nodes-same-var [rule-format, OF  $x$ -in-nodeslist no-in-nl]
have  $1 < \text{var } x$ 
  by auto
with nodes-balanced-ordered [rule-format, OF  $x$ -in-nodeslist]
show  $\neg \text{isLeaf-pt } x \text{ low high}$ 
  apply (unfold isLeaf-pt-def)
  apply fastforce
  done
qed
with  $x$ -in-nodeslist have  $x$ -nLeaf:  $\neg \text{isLeaf-pt } x \text{ low high}$ 
  by auto
from nodes-balanced-ordered [rule-format, OF  $x$ -in-nodeslist]
have ( $\text{low } x = \text{Null}$ ) = ( $\text{high } x = \text{Null}$ )
   $\wedge \text{low } x \notin \text{set } (\text{prx}@node\#sfx) \wedge \text{high } x \notin \text{set } (\text{prx}@node\#sfx)$ 
  by auto
with  $x$ -nLeaf balanced-no no-nLeaf repa-repb-nc
  nodes-notin-nl-neq-nln lno-notin-nl hno-notin-nl
show  $\text{repNodes-eq } x \text{ no low high repb} = \text{repNodes-eq } x \text{ no low high repa}$ 
  using [[simp-depth-limit=2]]
  by (simp add: repNodes-eq-def null-comp-def isLeaf-pt-def)
qed
with repb-no-def no-nln
have repb-no-whole-nl:
   $\text{repb } no = \text{hd } [\text{sn} \leftarrow (\text{prx}@node\#sfx). \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repb}]$ 
  apply –
  apply (subgoal-tac)
   $[\text{sn} \leftarrow (\text{prx}@node\#sfx). \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repa}]$ 
   $= [\text{sn} \leftarrow (\text{prx}@node\#sfx). \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repb}]$ 
  apply simp
  apply (rule P-eq-list-filter)
  apply auto
  done
from no-in-take-Sucn no-nln
have  $[\text{sn} \leftarrow (\text{prx}@[node]) . \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repb}] \neq []$ 
  apply –
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have
   $\text{hd } [\text{sn} \leftarrow (\text{prx}@[node]) . \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repb}] =$ 
   $\text{hd } [\text{sn} \leftarrow (\text{prx}@node\#sfx) . \text{repNodes-eq } \text{sn } \text{node } \text{low } \text{high } \text{repb}]$ 
  apply –
  apply (rule hd-filter-app [symmetric])
  apply auto
  done

```

```

    with repb-no-whole-nl no-nln show ?thesis
      by simp
  qed
  qed
  qed
  have repb-no-red-def: (∀ no ∈ set (prx@[node]).(repb ∘ low) no = (repb ∘ high)
no
  ∧ low no ≠ Null → repb no = (repb ∘ low) no)
  proof (intro ballI impI)
    fix no
    assume no-in-take-Sucn: no ∈ set (prx@[node])
    assume red-cond-no: (repb ∘ low) no = (repb ∘ high) no ∧ low no ≠ Null
    from no-in-take-Sucn have no-in-nl: no ∈ set (prx@node#sfx)
      by auto
    from nodes-balanced-ordered [rule-format, OF this] obtain
      no-nNull: no ≠ Null and
      balanced-no: (low no = Null) = (high no = Null) and
      lno-notin-nl: low no ∉ set (prx@node#sfx) and
      hno-notin-nl: high no ∉ set (prx@node#sfx) and
      isLeaf-var-no: isLeaf-pt no low high = (var no ≤ 1)
    by auto
    have nodes-notin-nl-neq-nln: ∀ p. p ∉ set (prx@node#sfx) → p ≠ node
      by auto
    show repb no = (repb ∘ low) no
    proof (cases no = node)
      case False
        note no-notin-nl=this
        with no-in-take-Sucn have no-in-take-n: no ∈ set prx
          by auto
        from False repa-repb-nc have repb-repa-no: repb no = repa no
          by auto
        with while-inv [rule-format, OF no-in-take-n] obtain
          repa-no-nNull: repa no ≠ Null and
          while-share-red-exp:
            (if (repa ∘ low) no = (repa ∘ high) no ∧ low no ≠ Null
            then repa no = (repa ∘ low) no
            else repa no = hd [sn ← prx. repNodes-eq sn no low high repa] ∧
            repa (repa no) = repa no ∧
            (∀ no1 ∈ set prx. ((repa ∘ high) no1 = (repa ∘ high) no ∧
            (repa ∘ low) no1 = (repa ∘ low) no) = (repa no = repa no1)))
          using [[simp-depth-limit=2]]
          by auto
        from red-cond-no nodes-notin-nl-neq-nln lno-notin-nl
          hno-notin-nl while-share-red-exp balanced-no repa-repb-nc
        have red-repa-no: repa no = (repa ∘ low) no
          by (auto simp add: null-comp-def)
        from red-cond-no nodes-notin-nl-neq-nln lno-notin-nl repa-repb-nc
        have (repb ∘ low) no = (repa ∘ low) no
          by (auto simp add: null-comp-def)

```

```

with red-repa-no no-notin-nl balanced-no repa-repb-nc
have repb no = (repb  $\times$  low) no
  by auto
with red-cond-no show ?thesis
  by auto
next
assume no = node
with share-cond
have share-cond-pre:
  isLeaf-pt no low high  $\vee$  repa (low no)  $\neq$  repa (high no)
  by simp
show ?thesis
proof (cases isLeaf-pt no low high)
  case True
  with red-cond-no show ?thesis
    by (simp add: isLeaf-pt-def)
  next
  assume no-nLeaf:  $\neg$  isLeaf-pt no low high
  with share-cond-pre
  have repa (low no)  $\neq$  repa (high no)
    by simp
  with no-nLeaf lno-notin-nl hno-notin-nl nodes-notin-nl-neq-nln
  balanced-no repa-repb-nc
  have repb (low no)  $\neq$  repb (high no)
    using [[simp-depth-limit=2]]
    by (auto simp add: isLeaf-pt-def)
  with no-nLeaf balanced-no have (repb  $\times$  low) no  $\neq$  (repb  $\times$  high) no
    by (simp add: null-comp-def isLeaf-pt-def)
  with red-cond-no show ?thesis
    by simp
qed
qed
qed
have while-while: ( $\forall$  no  $\in$  set (prx@[node])).
  repb no  $\neq$  Null  $\wedge$ 
  (if (repb  $\times$  low) no = (repb  $\times$  high) no  $\wedge$  low no  $\neq$  Null
  then repb no = (repb  $\times$  low) no
  else repb no = hd [sn  $\leftarrow$  (prx@[node]). repNodes-eq sn no low high repb]  $\wedge$ 
  repb (repb no) = repb no  $\wedge$ 
  ( $\forall$  no1  $\in$  set ((prx@[node])). ((repb  $\times$  high) no1 = (repb  $\times$  high) no
   $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no) = (repb no = repb no1)))
(is  $\forall$  no  $\in$  set (prx@[node]). ?P no  $\wedge$  ?Q no)
proof (intro ballI)
  fix no
  assume no-in-take-Sucn: no  $\in$  set (prx@[node])
  hence no-in-nl: no  $\in$  set (prx@node#sfx)
    by auto
  from nodes-balanced-ordered [rule-format, OF this] obtain
  no-nNull: no  $\neq$  Null and

```

```

balanced-no: (low no = Null) = (high no = Null) and
lno-notin-nl: low no ∉ set (prx@node#sfx) and
hno-notin-nl: high no ∉ set (prx@node#sfx) and
isLeaf-var-no: isLeaf-pt no low high = (var no ≤ 1)
by auto
from no-in-take-Sucn
have filter-take-Sucn-not-empty:
  [sn←(prx@[node]). repNodes-eq sn no low high repb] ≠ []
  apply -
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have hd-filter-Sucn-in-Sucn:
  hd [sn←(prx@[node]). repNodes-eq sn no low high repb] ∈
  set (prx@[node])
  by (rule hd-filter-in-list)
have nodes-notin-nl-neg-nln: ∀ p. p ∉ set (prx@node#sfx) → p ≠ node
  by auto
show ?P no ∧ ?Q no
proof (cases no = node)
  case False
  note no-notin-nl=this
  with no-in-take-Sucn
  have no-in-take-n: no ∈ set prx
    by auto
  from False repa-repb-nc have repb-repa-no: repb no = repa no
    by auto
  with while-inv [rule-format, OF no-in-take-n] obtain
    repa-no-nNull: repa no ≠ Null and
    while-share-red-exp:
      (if (repa × low) no = (repa × high) no ∧ low no ≠ Null
        then repa no = (repa × low) no
        else repa no = hd [sn←prx. repNodes-eq sn no low high repa] ∧
        repa (repa no) = repa no ∧
        (∀ no1 ∈ set prx. ((repa × high) no1 = (repa × high) no ∧
        (repa × low) no1 = (repa × low) no) = (repa no = repa no1)))
    using [[simp-depth-limit=2]]
    by auto
  from repb-repa-no repa-no-nNull have repb-no-nNull: ?P no
    by simp
  have ?Q no
  proof (cases (repb × low) no = (repb × high) no ∧ low no ≠ Null)
    case True
    with no-in-take-Sucn repb-no-red-def show ?thesis
      by auto
  next
  assume share-case-repb:
    ¬ ((repb × low) no = (repb × high) no ∧ low no ≠ Null)
  with repb-no-share-def no-in-take-Sucn

```

```

have repb-no-def: repb no = hd [sn← (prx@[node]).
  repNodes-eq sn no low high repb]
  by auto
with share-case-repb
have (repb  $\times$  low) no  $\neq$  (repb  $\times$  high) no  $\vee$  low no = Null
  using [[simp-depth-limit=2]]
  by simp
thus ?thesis
proof (cases low no = Null)
  case True
  note lno-Null=this
  with balanced-no have hno-Null: high no = Null
    by simp
  from lno-Null hno-Null have isLeaf-no: isLeaf-pt no low high
    by (simp add: isLeaf-pt-def)
  from True while-share-red-exp
  have while-low-Null:
    repa no = hd [sn←prx. repNodes-eq sn no low high repa]  $\wedge$ 
    repa (repa no) = repa no  $\wedge$ 
    ( $\forall$  no1  $\in$  set prx. ((repa  $\times$  high) no1 = (repa  $\times$  high) no
     $\wedge$  (repa  $\times$  low) no1 = (repa  $\times$  low) no) = (repa no = repa no1))
    by auto
  from no-in-take-n
  have [sn←prx. repNodes-eq sn no low high repa]  $\neq$  []
    apply -
    apply (rule filter-not-empty)
    apply (auto simp add: repNodes-eq-def)
    done
  then have hd-term-n-Sucn: hd [sn←prx. repNodes-eq sn no low high
repa] =
    hd [sn←(prx@[node]) . repNodes-eq sn no low high repa]
    apply -
    apply (rule hd-filter-app [symmetric])
    apply auto
    done
  have all-nodes-in-nl-Leafs:
     $\forall x \in$  set (prx@[node#sfx]). isLeaf-pt x low high
  proof (intro ballI)
    fix x
    assume x-in-nodeslist: x  $\in$  set (prx@[node#sfx])
    from isLeaf-no isLeaf-var-no have var no  $\leq$  1
      by simp
    with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
    have var x  $\leq$  1
      by simp
    with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
    show isLeaf-pt x low high
      by (auto simp add: isLeaf-pt-def)
  qed

```



```

from no-in-take-Sucn have
  filter-Sucn-no-notempty:
  [sn←(prx@[node]). repNodes-eq sn no low high repb] ≠ []
  apply –
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have hd-term-in-take-Sucn:
  hd [sn←(prx@[node]) . repNodes-eq sn no low high repb]
  ∈ set (prx@[node])
  by (rule hd-filter-in-list)
then have hd-term-in-nl:
  hd [sn←(prx@[node]) . repNodes-eq sn no low high repb]
  ∈ set (prx@node#sfx)
  by auto
with all-nodes-in-nl-Leafs
have hd-term-Leaf: isLeaf-pt (hd [sn← (prx@[node]).
  repNodes-eq sn no low high repb]) low high
  by auto
from while-low-Null have repa (repa no) = repa no
  by auto
with no-notin-nl repa-repb-nc
have repa-repb-no-repb: repa (repb no) = repb no
  by auto
have repb-repb-no: repb (repb no) = repb no
proof (cases repb no = node)
  case False
  with repa-repb-nc repa-repb-no-repb show ?thesis
  by auto
next
assume repb-no-nln: repb no = node
with hd-term-Leaf isLeaf-no all-nodes-in-nl-Leafs
have nested-hd-repa-repb:
  hd [sn←(prx@node#sfx). repNodes-eq sn
  (hd [sn←(prx@[node]) . repNodes-eq sn no low high repb])
  low high repa] =
  hd [sn←(prx@node#sfx). repNodes-eq sn
  ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb])
  low high repb]
  by (simp add: isLeaf-pt-def repNodes-eq-def null-comp-def)
from hd-term-in-take-Sucn
have [sn←(prx@[node]). repNodes-eq sn
  (hd [sn←(prx@[node]). repNodes-eq sn no low high repb])
  low high repb] ≠ []
  apply –
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have hd [sn←(prx@[node]). repNodes-eq sn

```

```

      ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
        low high repb] =
      hd [sn←(prx@node#sfx). repNodes-eq sn
        ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
          low high repb]
apply -
apply (rule hd-filter-app [symmetric])
apply auto
done
then have hd-term-nodeslist-Sucn:
  hd [sn←(prx@node#sfx). repNodes-eq sn
    ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
      low high repb] =
  hd [sn←(prx@[node]). repNodes-eq sn
    ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
      low high repb]
  by simp
from no-in-take-Sucn filter-Sucn-no-notempty
have filter-filter: hd [sn←(prx@[node]). repNodes-eq sn
  (hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
    low high repb] =
  hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
  apply -
  apply (rule filter-hd-P-rep-indep)
  apply (auto simp add: repNodes-eq-def)
  done
from repb-no-def repb-no-nln repb-node
have repb (repb no) = hd [sn←(prx@node#sfx). repNodes-eq sn
  ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
    low high repa]
  by simp
with nested-hd-repa-repb
have repb (repb no) = hd [sn←(prx@node#sfx). repNodes-eq sn
  (hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
    low high repb]
  by simp
with hd-term-nodeslist-Sucn
have repb (repb no) = hd [sn←(prx@[node]). repNodes-eq sn
  ( hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
    low high repb]
  by simp
with filter-filter
have repb (repb no) = hd [sn←(prx@[node]).
  repNodes-eq sn no low high repb]
  by simp
with repb-no-def show ?thesis
  by simp
qed
have two-nodes-repb: (∀ no1 ∈ set (prx@[node]).

```

```

      ((repb  $\times$  high) no1 = (repb  $\times$  high) no
       $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no) = (repb no = repb no1))
proof (intro ballI)
  fix no1
  assume no1-in-take-Sucn: no1  $\in$  set (prx@[node])
  then have no1  $\in$  set (prx@node#sfx) by auto
  with all-nodes-in-nl-Leafs
  have isLeaf-no1: isLeaf-pt no1 low high
    by auto
  with isLeaf-no
  have repbchildren-eq-no-no1: (repb  $\times$  high) no1 = (repb  $\times$  high) no
     $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no
    by (simp add: null-comp-def isLeaf-pt-def)
  from isLeaf-no1 isLeaf-no
  have repachildren-eq-no-no1: (repa  $\times$  high) no1 = (repa  $\times$  high) no
     $\wedge$  (repa  $\times$  low) no1 = (repa  $\times$  low) no
    by (simp add: null-comp-def isLeaf-pt-def)
  from while-low-Null
  have while-low-same-rep: ( $\forall$  no1  $\in$  set prx.
    ((repa  $\times$  high) no1 = (repa  $\times$  high) no
     $\wedge$  (repa  $\times$  low) no1 = (repa  $\times$  low) no) = (repa no = repa no1))
    by auto
  show ((repb  $\times$  high) no1 = (repb  $\times$  high) no  $\wedge$ 
    (repb  $\times$  low) no1 = (repb  $\times$  low) no) = (repb no = repb no1)
  proof (cases no1 = node)
    case False
    with no1-in-take-Sucn have no1  $\in$  set prx
      by auto
    with while-low-same-rep repachildren-eq-no-no1
    have repa no = repa no1
      by auto
    with repa-repb-nc no-notin-nl False repbchildren-eq-no-no1
    show ?thesis
      by auto
  next
  assume no1-nln: no1 = node
  hence no1-in-take-Sucn: no1  $\in$  set (prx@[node])
    by auto
  hence no1-in-nl: no1  $\in$  set (prx@node#sfx)
    by auto
  from nodes-balanced-ordered [rule-format, OF this] have
    balanced-no1: (low no1 = Null) = (high no1 = Null)
    by auto
  with no1-in-take-Sucn repb-no-share-def isLeaf-no1
  have repb-no1: repb no1 = hd [sn $\leftarrow$ (prx@[node]).
    repNodes-eq sn no1 low high repb]
    by (auto simp add: isLeaf-pt-def)
  from balanced-no1 isLeaf-no1 isLeaf-no balanced-no
  have repbchildren-eq-no1-no: (repb  $\times$  high) no1 = (repb  $\times$  high) no

```

```

       $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no
    by (simp add: null-comp-def isLeaf-pt-def)
  have  $\forall x \in \text{set } (\text{prx}@[\text{node}]). \text{repNodes-eq } x \text{ no low high repb}$ 
    = repNodes-eq x no1 low high repb
  proof (intro ballI)
    fix x
    assume x-in-take-Sucn:  $x \in \text{set } (\text{prx}@[\text{node}])$ 
    with repbchildren-eq-no1-no show repNodes-eq x no low high repb
      = repNodes-eq x no1 low high repb
      by (simp add: repNodes-eq-def)
    qed
  then have [sn $\leftarrow$ (prx@[node]). repNodes-eq sn no low high repb]
    = [sn $\leftarrow$ (prx@[node]). repNodes-eq sn no1 low high repb]
    by (rule P-eq-list-filter)
  with repb-no-def repb-no1 have repb-no-no1: repb no = repb no1
    by simp
  with repbchildren-eq-no1-no show ?thesis
    by simp
  qed
  qed
  with repb-repb-no repb-no-share-def no-in-take-Sucn share-case-repb
  show ?thesis
    using [[simp-depth-limit=4]]
    by auto
next
  assume lno-nNull: low no  $\neq$  Null
  with share-case-repb
  have repbchildren-neq-no: (repb  $\times$  low) no  $\neq$  (repb  $\times$  high) no
    by auto
  from balanced-no lno-nNull
  have hno-nNull: high no  $\neq$  Null
    by simp
  with repbchildren-neq-no lno-nNull repa-repb-nc
    lno-notin-nl hno-notin-nl nodes-notin-nl-neq-nln
  have repachildren-neq-no: (repa  $\times$  low) no  $\neq$  (repa  $\times$  high) no
    using [[simp-depth-limit=2]]
    by (auto simp add: null-comp-def)
  with while-share-red-exp
  have repa-while-inv: repa (repa no) = repa no
     $\wedge$  ( $\forall \text{no1} \in \text{set } \text{prx}. ((\text{repa} \times \text{high}) \text{no1} = (\text{repa} \times \text{high}) \text{no})$ 
     $\wedge$  ( $\text{repa} \times \text{low}) \text{no1} = (\text{repa} \times \text{low}) \text{no}) = (\text{repa no} = \text{repa no1}))$ 
    by auto
  from lno-nNull hno-nNull
  have no-nLeaf:  $\neg \text{isLeaf-pt no low high}$ 
    by (simp add: isLeaf-pt-def)
  have all-nodes-in-nl-nLeafs:
     $\forall x \in \text{set } (\text{prx}@[\text{node}\#\text{sfx}]). \neg \text{isLeaf-pt } x \text{ low high}$ 
  proof (intro ballI)
    fix x

```

```

assume x-in-nodeslist:  $x \in \text{set } (\text{prx}@node\#sfx)$ 
from no-nLeaf isLeaf-var-no have  $1 < \text{var } no$ 
  by simp
with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
have  $1 < \text{var } x$ 
  by simp
with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
show  $\neg \text{isLeaf-pt } x \text{ low high}$ 
  using [simp-depth-limit = 2]
  by (auto simp add: isLeaf-pt-def)
qed
have repb-repb-no:  $\text{repb } (\text{repb } no) = \text{repb } no$ 
proof -
  from repa-while-inv no-notin-nl repa-repb-nc
  have repa ( $\text{repb } no$ ) =  $\text{repb } no$ 
    by simp
  from hd-filter-Sucn-in-Sucn repb-no-def
  have repb-no-in-take-Sucn:  $\text{repb } no \in \text{set } (\text{prx}@[node])$ 
    by simp
  hence repb-no-in-nl:  $\text{repb } no \in \text{set } (\text{prx}@node\#sfx)$ 
    by auto
  from all-nodes-in-nl-nLeafs repb-no-in-nl
  have repb-no-nLeaf:  $\neg \text{isLeaf-pt } (\text{repb } no) \text{ low high}$ 
    by auto
  from nodes-balanced-ordered [rule-format, OF repb-no-in-nl]
  have ( $\text{low } (\text{repb } no) = \text{Null}$ ) = ( $\text{high } (\text{repb } no) = \text{Null}$ )
     $\wedge \text{low } (\text{repb } no) \notin \text{set } (\text{prx}@node\#sfx) \wedge$ 
     $\text{high } (\text{repb } no) \notin \text{set } (\text{prx}@node\#sfx)$ 
    by auto
  from filter-take-Sucn-not-empty
  have repNodes-eq ( $\text{hd } [sn \leftarrow (\text{prx}@[node])]$ ).
    repNodes-eq sn no low high repb)  $\text{no low high repb}$ 
    by (rule hd-filter-prop)
  with repb-no-def have repNodes-eq ( $\text{repb } no$ )  $\text{no low high repb}$ 
    by simp
  then have ( $\text{repb } \times \text{low}$ ) ( $\text{repb } no$ ) = ( $\text{repb } \times \text{low}$ )  $\text{no}$ 
     $\wedge (\text{repb } \times \text{high}) (\text{repb } no) = (\text{repb } \times \text{high}) \text{no}$ 
    by (simp add: repNodes-eq-def)
  with repbchildren-neq-no have ( $\text{repb } \times \text{low}$ ) ( $\text{repb } no$ )
     $\neq (\text{repb } \times \text{high}) (\text{repb } no)$ 
    by simp
  with repb-no-in-take-Sucn repb-no-share-def
  have repb-repb-no-double-hd:
     $\text{repb } (\text{repb } no) = \text{hd } [sn \leftarrow (\text{prx}@[node])]$ .
    repNodes-eq sn (repb no) low high repb]
    by auto
  from filter-take-Sucn-not-empty
  have hd [ $sn \leftarrow (\text{prx}@[node])$ ].
    repNodes-eq sn (repb no) low high repb] =  $\text{repb } no$ 

```

```

    apply (simp only: repb-no-def )
    apply (rule filter-hd-P-rep-indep)
    apply (auto simp add: repNodes-eq-def)
  done
with repb-repb-no-double-hd show ?thesis
  by simp
qed
have (∀ no1 ∈ set (prx@[node]).
  ((repb × high) no1 = (repb × high) no ∧
  (repb × low) no1 = (repb × low) no) = (repb no = repb no1))
proof (intro ballI)
  fix no1
  assume no1-in-take-Sucn: no1 ∈ set (prx@[node])
  hence no1-in-nl: no1 ∈ set (prx@node#sfx)
    by auto
  from all-nodes-in-nl-nLeafs no1-in-nl
  have no1-nLeaf: ¬ isLeaf-pt no1 low high
    by auto
  from nodes-balanced-ordered [rule-format, OF no1-in-nl]
  have no1-props: (low no1 = Null) = (high no1 = Null)
    ∧ low no1 ∉ set (prx@node#sfx) ∧ high no1 ∉ set (prx@node#sfx)
    by auto
  show ((repb × high) no1 = (repb × high) no
    ∧ (repb × low) no1 = (repb × low) no) = (repb no = repb no1)
  proof (cases no1 = node)
    case False
    note no1-neq-nln=this
    with no1-in-take-Sucn
    have no1-in-take-n: no1 ∈ set prx
      by auto
    with repa-while-inv have ((repa × high) no1 = (repa × high) no
      ∧ (repa × low) no1 = (repa × low) no) = (repa no = repa no1)
      by fastforce
    with no1-props no1-nLeaf no-nLeaf balanced-no lno-notin-nl
      hno-notin-nl nodes-notin-nl-neq-nln no-notin-nl
      no1-neq-nln repa-repb-nc
    show ?thesis
      using [[simp-depth-limit=1]]
      by (auto simp add: null-comp-def isLeaf-pt-def)
  next
  assume no1-nln: no1 = node
  show ?thesis
  proof
    assume repbchildren-eq-no1-no:
      (repb × high) no1 = (repb × high) no
      ∧ (repb × low) no1 = (repb × low) no
    with repbchildren-neq-no
    have (repb × high) no1 ≠ (repb × low) no1
      by auto
  end
end

```

```

with repb-no-share-def no1-in-take-Sucn
have repb-no1-def: repb no1 = hd [sn←(prx@[node]).
  repNodes-eq sn no1 low high repb]
  by auto
have filter-no1-eq-filter-no: [sn←(prx@[node]).
  repNodes-eq sn no1 low high repb] =
  [sn←(prx@[node]). repNodes-eq sn no low high repb]
proof –
  have  $\forall x \in \text{set } (prx@[node])$ .
    repNodes-eq x no1 low high repb =
    repNodes-eq x no low high repb
  proof (intro ballI)
    fix x
    assume x-in-take-Sucn:  $x \in \text{set } (prx@[node])$ 
    with repbchildren-eq-no1-no
    show repNodes-eq x no1 low high repb =
      repNodes-eq x no low high repb
    by (simp add: repNodes-eq-def)
  qed
then show ?thesis
  by (rule P-eq-list-filter)
qed
with repb-no1-def repb-no-def show repb no = repb no1
  by simp
next
assume repb-no-no1-eq: repb no = repb no1
from no1-nln repb-node repb-no-def have repb-no1-def:
  repb no1 =
  hd [sn←(prx@node#sfx). repNodes-eq sn node low high repa]
  by auto
with no1-nln repb-no-def repb-no-no1-eq
have repb-Sucn-repa-nl-hd: hd [sn←(prx@[node]).
  repNodes-eq sn no low high repb] =
  hd [sn←(prx@node#sfx). repNodes-eq sn no1 low high repa]
  by simp
from filter-take-Sucn-not-empty
have hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
  = hd [sn←(prx@node#sfx) . repNodes-eq sn no low high repb]
  apply –
  apply (rule hd-filter-app [symmetric])
  apply auto
  done
then have hd-Sucn-hd-whole-list:
  hd [sn←(prx@[node]) .
  repNodes-eq sn no low high repb] =
  hd [sn← (prx@node#sfx). repNodes-eq sn no low high repb]
  by simp
have hd-nl-repb-repa:
  [sn← (prx@node#sfx). repNodes-eq sn no low high repb]

```

```

= [sn←(prx@node#sfx). repNodes-eq sn no low high repa]
proof –
  have  $\forall x \in \text{set } (prx@node\#sfx).$ 
    repNodes-eq x no low high repb =
    repNodes-eq x no low high repa
  proof (intro ballI)
    fix x
    assume x-in-nl:  $x \in \text{set } (prx@node\#sfx)$ 
    from all-nodes-in-nl-nLeafs x-in-nl
    have x-nLeaf:  $\neg \text{isLeaf-pt } x \text{ low high}$ 
      by auto
    from nodes-balanced-ordered [rule-format, OF x-in-nl]
    have x-props:  $(\text{low } x = \text{Null}) = (\text{high } x = \text{Null}) \wedge$ 
       $\text{low } x \notin \text{set } (prx@node\#sfx) \wedge \text{high } x \notin \text{set } (prx@node\#sfx)$ 
      by auto
    with x-nLeaf lno-nNull hno-nNull lno-notin-nl hno-notin-nl
      nodes-notin-nl-neq-nln repa-repb-nc
    show repNodes-eq x no low high repb =
      repNodes-eq x no low high repa
      using [[simp-depth-limit=1]]
      by (simp add: repNodes-eq-def isLeaf-pt-def null-comp-def)
    qed
  then show ?thesis
    by (rule P-eq-list-filter)
qed
with repb-Sucn-repa-nl-hd hd-Sucn-hd-whole-list
have filter-nl-no-no1:
  hd [sn←(prx@node#sfx). repNodes-eq sn no low high repa]
= hd [sn←(prx@node#sfx). repNodes-eq sn no1 low high repa]
by simp
from no-in-nl have filter-no-not-empty:
[sn←(prx@node#sfx). repNodes-eq sn no low high repa]  $\neq []$ 
apply –
apply (rule filter-not-empty)
apply (auto simp add: repNodes-eq-def)
done
from no1-in-nl have filter-no1-not-empty:
[sn←(prx@node#sfx). repNodes-eq sn no1 low high repa]  $\neq []$ 
apply –
apply (rule filter-not-empty)
apply (auto simp add: repNodes-eq-def)
done
from repb-no-def hd-Sucn-hd-whole-list hd-nl-repb-repa
have repb no =
  hd [sn←(prx@node#sfx). repNodes-eq sn no low high repa]
by simp
with hd-filter-prop [OF filter-no-not-empty ]
have repNodes-no-repa: repNodes-eq (repb no) no low high repa
by auto

```



```

from repb-no1-def no1-nln
have
  repb no1 = hd [sn←(prx@node#sfx). repNodes-eq sn no1
    low high repa]
  by simp
with hd-filter-prop [OF filter-no1-not-empty ]
have repNodes-eq (repb no1) no1 low high repa
  by auto
with filter-nl-no-no1 repNodes-no-repa repb-no-no1-eq
have (repa  $\times$  high) no1 =
  (repa  $\times$  high) no  $\wedge$  (repa  $\times$  low) no1 = (repa  $\times$  low) no
  by (simp add: repNodes-eq-def)
with hno-nNull no1-props no1-nLeaf lno-nNull lno-notin-nl
  hno-notin-nl nodes-notin-nl-neq-nln repa-repb-nc
show (repb  $\times$  high) no1 =
  (repb  $\times$  high) no  $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no
  using [[simp-depth-limit=1]]
  by (auto simp add: isLeaf-pt-def null-comp-def)
qed
qed
qed
with repb-repb-no repb-no-share-def share-case-repb no-in-take-Sucn
show ?thesis
  using [[simp-depth-limit=1]]
  by auto
qed
qed
with repb-no-nNull show ?thesis
  by simp
next
assume no-nln: no = node
with repb-node have repb-no-def:
  repb no = hd [sn←(prx@node#sfx). repNodes-eq sn no low high repa]
  by simp
from no-nln have no  $\in$  set (prx@node#sfx)
  by auto
then have filter-nl-repa-not-empty:
  [sn←(prx@node#sfx). repNodes-eq sn no low high repa]  $\neq$  []
  apply -
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have hd-filter-nl-in-nl:
  hd [sn←(prx@node#sfx). repNodes-eq sn no low high repa]  $\in$  set (prx@node#sfx)
  by (rule hd-filter-in-list)
with repb-no-def
have repb-no-in-nodeslist: repb no  $\in$  set (prx@node#sfx)
  by simp
from nodes-balanced-ordered [rule-format,OF this]

```

```

have repb-no-nNull: repb no ≠ Null
  by auto
from share-cond no-nln have share-cond-or:
  isLeaf-pt no low high ∨ repa (low no) ≠ repa (high no)
  by auto
have share-reduce-if: (if (repb ∝ low) no = (repb ∝ high) no ∧ low no ≠
Null
  then repb no = (repb ∝ low) no
  else repb no = hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
∧
  repb (repb no) = repb no
  ∧ (∀ no1 ∈ set (prx@[node]). ((repb ∝ high) no1 = (repb ∝ high) no
  ∧ (repb ∝ low) no1 = (repb ∝ low) no) = (repb no = repb no1)))
proof (cases isLeaf-pt no low high)
  case True
  note isLeaf-no=this
  then have lno-Null: low no = Null by (simp add: isLeaf-pt-def)
  from isLeaf-no no-in-take-Sucn repb-no-share-def
  have repb-no-repb-def: repb no
    = hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
    by (auto simp add: isLeaf-pt-def)
  from isLeaf-no nodes-balanced-ordered [rule-format, OF no-in-nl]
  have var-no: var no ≤ 1
    by auto
  have all-nodes-nl-var-l-1: ∀ x ∈ set (prx@node#sfx). var x ≤ 1
  proof (intro ballI)
    fix x
    assume x-in-nl: x ∈ set (prx@node#sfx)
    from all-nodes-same-var [rule-format, OF x-in-nl no-in-nl] var-no
    show var x ≤ 1
      by auto
  qed
  have all-nodes-nl-Leafs: ∀ x ∈ set (prx@node#sfx). isLeaf-pt x low high
  proof (intro ballI)
    fix x
    assume x-in-nl: x ∈ set (prx@node#sfx)
    with all-nodes-nl-var-l-1 have var x ≤ 1
      by auto
    with nodes-balanced-ordered [rule-format, OF x-in-nl ]
    show isLeaf-pt x low high
      by auto
  qed
  have repb-repb-no: repb (repb no) = repb no
  proof –
    from repb-no-share-def no-in-take-Sucn lno-Null
    have repb-no-def: repb no =
      hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
      by auto
    with hd-filter-Sucn-in-Sucn

```

```

have repb-no-in-take-Sucn:  $\text{repb } no \in \text{set } (prx@[node])$ 
  by simp
hence repb-no-in-nl:  $\text{repb } no \in \text{set } (prx@[node])$ 
  by auto
with all-nodes-nl-Leafs
have repb-no-Leaf:  $\text{isLeaf-pt } (repb \ no) \ \text{low } \text{high}$ 
  by auto
with repb-no-in-take-Sucn repb-no-share-def
have repb-repb-no-def:  $\text{repb } (repb \ no) =$ 
   $\text{hd } [sn \leftarrow (prx@[node]). \ \text{repNodes-eq } sn \ (repb \ no) \ \text{low } \text{high } \text{repb}]$ 
  by (auto simp add: isLeaf-pt-def)
from filter-take-Sucn-not-empty
show ?thesis
  apply (simp only: repb-repb-no-def )
  apply (simp only: repb-no-def)
  apply (rule filter-hd-P-rep-indep)
  apply (auto simp add: repNodes-eq-def)
done
qed
have two-nodes-repb:  $(\forall \ no1 \in \text{set } (prx@[node]).$ 
   $(\text{repb } \times \ \text{high}) \ no1 = (\text{repb } \times \ \text{high}) \ no \wedge$ 
   $(\text{repb } \times \ \text{low}) \ no1 = (\text{repb } \times \ \text{low}) \ no) = (\text{repb } \ no = \text{repb } \ no1))$ 
proof (intro ballI)
  fix no1
  assume no1-in-take-Sucn:  $\text{no1} \in \text{set } (prx@[node])$ 
  from no1-in-take-Sucn
  have  $\text{no1} \in \text{set } (prx@node\#sfx)$ 
  by auto
  with all-nodes-nl-Leafs
  have isLeaf-no1:  $\text{isLeaf-pt } \text{no1} \ \text{low } \text{high}$ 
  by auto
  with repb-no-share-def no1-in-take-Sucn
  have repb-no1-def:  $\text{repb } \text{no1} =$ 
   $\text{hd } [sn \leftarrow (prx@[node]). \ \text{repNodes-eq } sn \ \text{no1} \ \text{low } \text{high } \text{repb}]$ 
  by (auto simp add: isLeaf-pt-def)
  show  $(\text{repb } \times \ \text{high}) \ \text{no1} = (\text{repb } \times \ \text{high}) \ no$ 
   $\wedge (\text{repb } \times \ \text{low}) \ \text{no1} = (\text{repb } \times \ \text{low}) \ no) = (\text{repb } \ no = \text{repb } \ \text{no1})$ 
proof
  assume repbchildren-eq-no1-no:  $(\text{repb } \times \ \text{high}) \ \text{no1} = (\text{repb } \times \ \text{high}) \ no$ 
   $\wedge (\text{repb } \times \ \text{low}) \ \text{no1} = (\text{repb } \times \ \text{low}) \ no$ 
  have  $[sn \leftarrow (prx@[node]). \ \text{repNodes-eq } sn \ \text{no1} \ \text{low } \text{high } \text{repb}]$ 
   $= [sn \leftarrow (prx@[node]). \ \text{repNodes-eq } sn \ no \ \text{low } \text{high } \text{repb}]$ 
proof –
  have  $\forall x \in \text{set } (prx@[node]).$ 
   $\text{repNodes-eq } x \ \text{no1} \ \text{low } \text{high } \text{repb} = \text{repNodes-eq } x \ no \ \text{low } \text{high } \text{repb}$ 
proof (intro ballI)
  fix x
  assume x-in-take-Sucn:  $x \in \text{set } (prx@[node])$ 
  with repbchildren-eq-no1-no

```

```

      show repNodes-eq x no1 low high repb = repNodes-eq x no low high
    repb
      by (simp add: repNodes-eq-def)
    qed
    then show ?thesis
      by (rule P-eq-list-filter)
    qed
    with repb-no1-def repb-no-repb-def
    show repb no = repb no1
      by simp
    next
    assume repb-no-no1: repb no = repb no1
    with isLeaf-no isLeaf-no1
    show (repb  $\times$  high) no1 = (repb  $\times$  high) no
       $\wedge$  (repb  $\times$  low) no1 = (repb  $\times$  low) no
      by (simp add: null-comp-def isLeaf-pt-def)
    qed
  qed
  with repb-repb-no lno-Null no-in-take-Sucn repb-no-share-def show ?thesis
    by auto
  next
  assume no-nLeaf:  $\neg$  isLeaf-pt no low high
  with balanced-no obtain
    lno-nNull: low no  $\neq$  Null and
    hno-nNull: high no  $\neq$  Null
    by (simp add: isLeaf-pt-def)
  from no-nLeaf nodes-balanced-ordered [rule-format, OF no-in-nl]
  have var-no:  $1 < \text{var } no$ 
    by auto
  have all-nodes-nl-var-l-1:  $\forall x \in \text{set } (\text{prx}@node\#sfx). 1 < \text{var } x$ 
  proof (intro ballI)
    fix x
    assume x-in-nl:  $x \in \text{set } (\text{prx}@node\#sfx)$ 
    with all-nodes-same-var [rule-format, OF x-in-nl no-in-nl] var-no
    show  $1 < \text{var } x$ 
      by simp
  qed
  have all-nodes-nl-nLeafs:  $\forall x \in \text{set } (\text{prx}@node\#sfx). \neg \text{isLeaf-pt } x \text{ low}$ 
  high
  proof (intro ballI)
    fix x
    assume x-in-nl:  $x \in \text{set } (\text{prx}@node\#sfx)$ 
    with all-nodes-nl-var-l-1 have  $1 < \text{var } x$ 
      by auto
    with nodes-balanced-ordered [rule-format, OF x-in-nl] show  $\neg \text{isLeaf-pt}$ 
  x low high
    by auto
  qed
  from no-nLeaf share-cond-or

```

```

have repchildren-neq-no: repa (low no) ≠ repa (high no)
  by auto
with lno-nNull hno-nNull
have (repa ∝ low) no ≠ (repa ∝ high) no
  by (simp add: null-comp-def)
with repa-repb-nc lno-notin-nl hno-notin-nl
  nodes-notin-nl-neq-nln lno-nNull hno-nNull
have repbchildren-neq-no: (repb ∝ low) no ≠ (repb ∝ high) no
  using [[simp-depth-limit=1]]
  by (auto simp add: null-comp-def)
have repb-repb-no: repb (repb no) = repb no
proof –
  from repb-no-share-def no-in-take-Sucn repbchildren-neq-no
  have repb-no-def: repb no =
    hd [sn←(prx@[node]). repNodes-eq sn no low high repb]
  by auto
  from filter-take-Sucn-not-empty
  have repNodes-eq (repb no) no low high repb
  apply (simp only: repb-no-def)
  apply (rule hd-filter-prop)
  apply simp
  done
with repbchildren-neq-no
have repbchildren-neq-repb-no: (repb ∝ low) (repb no) ≠ (repb ∝ high)
  (repb no)
  by (simp add: repNodes-eq-def)
  from filter-take-Sucn-not-empty
  have repb no ∈ set (prx@[node])
  apply (simp only: repb-no-def)
  apply (rule hd-filter-in-list)
  apply simp
  done
with repbchildren-neq-repb-no repb-no-share-def
have repb-repb-no-def: repb (repb no) =
  hd [sn←(prx@[node]) . repNodes-eq sn (repb no) low high repb]
  by auto
  from filter-take-Sucn-not-empty show ?thesis
  apply (simp only: repb-repb-no-def)
  apply (simp only: repb-no-def)
  apply (rule filter-hd-P-rep-indep)
  apply (auto simp add: repNodes-eq-def)
  done
qed
have two-nodes-repb: (∀ no1 ∈ set (prx@[node]).
  ((repb ∝ high) no1 = (repb ∝ high) no ∧
  (repb ∝ low) no1 = (repb ∝ low) no) = (repb no = repb no1))
  (is (∀ no1 ∈ set (prx@[node]). ?P no1))
proof (intro ballI)
  fix no1

```

```

assume no1-in-take-Sucn:  $no1 \in \text{set } (prx@[node])$ 
hence no1-in-nodelist:  $no1 \in \text{set } (prx@node\#sfx)$ 
  by auto
with all-nodes-nl-nLeafs
have no1-nLeaf:  $\neg \text{isLeaf-pt } no1 \text{ low high}$ 
  by auto
show ?P no1
proof
assume repbchildren-eq-no1-no:  $(repb \times high) no1 = (repb \times high) no$ 
   $\wedge (repb \times low) no1 = (repb \times low) no$ 
with repbchildren-neq-no have  $(repb \times high) no1 \neq (repb \times low) no1$ 
  by auto
with no1-in-take-Sucn repb-no-share-def have repb-no1-def:  $repb no1$ 
=
  hd [sn←(prx@[node])]. repNodes-eq sn no1 low high repb]
  by auto
from repb-no-share-def no-in-take-Sucn repbchildren-neq-no
have repb-no-def:  $repb no =$ 
  hd [sn←(prx@[node])]. repNodes-eq sn no low high repb]
  by auto
have [sn←(prx@[node])]. repNodes-eq sn no1 low high repb] =
  [sn←(prx@[node])]. repNodes-eq sn no low high repb]
proof –
  have  $\forall x \in \text{set } (prx@[node])$ .
    repNodes-eq x no1 low high repb = repNodes-eq x no low high repb
  proof (intro ballI)
    fix x
    assume x-in-take-Sucn:  $x \in \text{set } (prx@[node])$ 
    with repbchildren-eq-no1-no
    show repNodes-eq x no1 low high repb = repNodes-eq x no low high
repb
    by (simp add: repNodes-eq-def)
  qed
then show ?thesis
  by (rule P-eq-list-filter)
qed
with repb-no-def repb-no1-def show  $repb no = repb no1$ 
  by simp
next
assume repb-no-no1:  $repb no = repb no1$ 
from repb-no-share-def no-in-take-Sucn repbchildren-neq-no
have repb-no-def:  $repb no =$ 
  hd [sn←(prx@[node])]. repNodes-eq sn no low high repb]
  by auto
from filter-take-Sucn-not-empty
have  $repb no \in \text{set } (prx@[node])$ 
  apply (simp only: repb-no-def)
  apply (rule hd-filter-in-list)
  apply simp

```

```

done
then have repb-no-in-nl: repb no ∈ set (prx@node#sfx)
  by auto
from filter-take-Sucn-not-empty
have repNodes-repb-no: repNodes-eq (repb no) no low high repb
  apply (simp only: repb-no-def)
  apply (rule hd-filter-prop)
  apply simp
done
show (repb × high) no1 = (repb × high) no
  ∧ (repb × low) no1 = (repb × low) no
proof (cases (repb × low) no1 = (repb × high) no1)
case True
note red-cond=this
from no1-in-nodeslist all-nodes-nl-nLeafs
have no1-nLeaf: ¬ isLeaf-pt no1 low high
  by auto
from nodes-balanced-ordered [rule-format, OF no1-in-nodeslist]
have no1-props: (low no1 ∉ set (prx@node#sfx))
  ∧ (high no1 ∉ set (prx@node#sfx)) ∧ (low no1 = Null) = (high
no1 = Null)
  ∧ ((repb × low) no1 ∉ set (prx@node#sfx))
  by auto
with red-cond no1-nLeaf no1-in-take-Sucn repb-no-red-def
have repb-no1-def: repb no1 = (repb × low) no1
  by (auto simp add: isLeaf-pt-def)
with no1-nLeaf no1-props have repb no1 = repb (low no1)
  by (simp add: null-comp-def isLeaf-pt-def)
from no1-props no1-nLeaf have rep (low no1) ∉ set (prx@node#sfx)
  by (auto simp add: isLeaf-pt-def null-comp-def)
with rep-repb-nc no1-props
have repb (low no1) ∉ set (prx@node#sfx)
  by auto
with repb-no1-def repb-no-no1 no1-props no1-nLeaf
have repb no ∉ set (prx@node#sfx)
  by (simp add: isLeaf-pt-def null-comp-def)
with repb-no-in-nl show ?thesis
  by simp
next
assume (repb × low) no1 ≠ (repb × high) no1
with repb-no-share-def no1-in-take-Sucn
have repb-no1-def: repb no1 =
  hd [sn←(prx@[node]). repNodes-eq sn no1 low high repb]
  by auto
from no1-in-take-Sucn
have [sn←(prx@[node]). repNodes-eq sn no1 low high repb] ≠ []
  apply -
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)

```

```

      done
    then
      have repNodes-repb-no1: repNodes-eq (repb no1) no1 low high repb
        apply (simp only: repb-no1-def )
        apply (rule hd-filter-prop)
        apply simp
      done
      with repNodes-repb-no repb-no-no1
      have repNodes-eq no1 no low high repb
        by (simp add: repNodes-eq-def)
      then show ?thesis
        by (simp add: repNodes-eq-def)
    qed
  qed
  with repb-repb-no repb-no-share-def no-in-take-Sucn repbchildren-neq-no
  show ?thesis
    using [[simp-depth-limit=2]]
    by fastforce
  qed
  with repb-no-nNull show ?thesis
    by simp
  qed
  with rep-repb-nc show ?thesis
    by (intro conjI)
  qed
end

```

9 Proof of Procedure Repoint

theory *RepointProof* **imports** *ProcedureSpecs* **begin**

hide-const (**open**) *DistinctTreeProver.set-of tree.Node tree.Tip*

lemma (**in** *Repoint-impl*) *Repoint-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ 'p} ::= \text{PROC Repoint ('p)}$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [low, high]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma *low-high-exchange-dag*:

assumes *pt-same*: $\forall pt. pt \notin \text{set-of } lt \longrightarrow \text{low } pt = \text{lowa } pt \wedge \text{high } pt = \text{higha } pt$

assumes *pt-changed*: $\forall pt \in \text{set-of } lt. \text{lowa } pt = (\text{rep} \circ \text{low}) \text{ pt} \wedge$
 $\text{higha } pt = (\text{rep} \circ \text{high}) \text{ pt}$

assumes *rep-pt*: $\forall pt \in \text{set-of } rt. \text{rep } pt = pt$


```

shows  $\bigwedge q. \text{Dag } q \text{ (rep } \times \text{ low) (rep } \times \text{ high) } rt \implies$ 
            $\text{Dag } q \text{ (rep } \times \text{ lowa) (rep } \times \text{ higha) } rt$ 
using rep-pt
proof (induct rt)
  case Tip thus ?case by simp
next
  case (Node lrt q' rrt)
  have  $\text{Dag } q \text{ (rep } \times \text{ low) (rep } \times \text{ high) (Node lrt q' rrt)$  by fact
  then obtain
     $q': q = q'$  and
     $q\text{-notNull}: q \neq \text{Null}$  and
     $lrt: \text{Dag } ((\text{rep } \times \text{ low}) q) \text{ (rep } \times \text{ low) (rep } \times \text{ high) } lrt$  and
     $rrt: \text{Dag } ((\text{rep } \times \text{ high}) q) \text{ (rep } \times \text{ low) (rep } \times \text{ high) } rrt$ 
    by auto
  have  $rlowa\text{-}rlow: ((\text{rep } \times \text{ lowa}) q) = ((\text{rep } \times \text{ low}) q)$ 
  proof (cases q  $\in$  set-of lrt)
    case True
    note q-in-lt=this
    with pt-changed have  $lowa\text{-}q: lowa\ q = (\text{rep } \times \text{ low})\ q$ 
      by simp
    thus  $(\text{rep } \times \text{ lowa})\ q = (\text{rep } \times \text{ low})\ q$ 
    proof (cases low q = Null)
      case True
      with  $lowa\text{-}q$  have  $lowa\ q = \text{Null}$ 
        by (simp add: null-comp-def)
      with True show ?thesis
        by (simp add: null-comp-def)
    next
    assume  $lq\text{-}n\text{Null}: low\ q \neq \text{Null}$ 
    show ?thesis
    proof (cases (rep } \times \text{ low) } q = \text{Null})
      case True
      with  $lowa\text{-}q$  have  $lowa\ q = \text{Null}$ 
        by simp
      with True show ?thesis
        by (simp add: null-comp-def)
    next
    assume  $rlq\text{-}n\text{Null}: (\text{rep } \times \text{ low})\ q \neq \text{Null}$ 
    with  $lrt\ lowa\text{-}q$  have  $lowa\ q \in \text{set-of } lrt$ 
      by auto
    with Node.premis Node have  $lowa\ q \in \text{set-of } (\text{Node } lrt\ q'\ rrt)$ 
      by simp
    with Node.premis have  $rep\ (lowa\ q) = lowa\ q$ 
      by auto
    with  $lowa\text{-}q\ rlq\text{-}n\text{Null}$  show ?thesis
      by (simp add: null-comp-def)
  qed
qed
next

```

```

assume q-notin-lt:  $q \notin \text{set-of } lt$ 
with pt-same have  $low\ q = low_a\ q$ 
  by auto
thus ?thesis
  by (simp add: null-comp-def)
qed
have rhigha-rhigh:  $((rep \ \times\ high_a)\ q) = ((rep \ \times\ high)\ q)$ 
proof (cases q \in set-of lt)
  case True
  note q-in-lt=this
  with pt-changed have higha-q:  $high_a\ q = (rep \ \times\ high)\ q$ 
  by simp
  thus ?thesis
proof (cases high q = Null)
  case True
  with higha-q have  $high_a\ q = Null$ 
  by (simp add: null-comp-def)
  with True show ?thesis
  by (simp add: null-comp-def)
next
assume hq-nNull:  $high\ q \neq Null$ 
show ?thesis
proof (cases (rep \ \times\ high)\ q = Null)
  case True
  with higha-q have  $high_a\ q = Null$ 
  by simp
  with True show ?thesis
  by (simp add: null-comp-def)
next
assume rhq-nNull:  $(rep \ \times\ high)\ q \neq Null$ 
with rrt higha-q have  $high_a\ q \in \text{set-of } rrt$ 
  by auto
with Node.prems Node have  $high_a\ q \in \text{set-of } (Node\ lrt\ q'\ rrt)$ 
  by simp
with Node.prems have  $rep\ (high_a\ q) = high_a\ q$ 
  by auto
with higha-q rhq-nNull show ?thesis
  by (simp add: null-comp-def)
qed
qed
next
assume q-notin-lt:  $q \notin \text{set-of } lt$ 
with pt-same have  $high\ q = high_a\ q$ 
  by auto
thus ?thesis
  by (simp add: null-comp-def)
qed
with rrt have rhigha-mixed-dag:
   $Dag\ ((rep \ \times\ high_a)\ q)\ ((rep \ \times\ low)\ (rep \ \times\ high)\ rrt)$ 

```

```

  by simp
  from lrt rlowa-rlow have rlowa-mixed-dag:
    Dag ((rep  $\times$  lowa) q) (rep  $\times$  low) (rep  $\times$  high) lrt
  by simp
  from Node.premis have lrt-rep-eq:  $\forall pt \in \text{set-of } lrt. \text{rep } pt = pt$ 
  by simp
  from Node.premis have rrt-rep-eq:  $\forall pt \in \text{set-of } rrt. \text{rep } pt = pt$ 
  by simp
  from rlowa-mixed-dag lrt-rep-eq have lowa-lrt:
    Dag ((rep  $\times$  lowa) q) (rep  $\times$  lowa) (rep  $\times$  higha) lrt
  apply -
  apply (rule Node.hyps)
  apply auto
  done
  from rhigha-mixed-dag rrt-rep-eq have higha-rrt:
    Dag ((rep  $\times$  higha) q) (rep  $\times$  lowa) (rep  $\times$  higha) rrt
  apply -
  apply (rule Node.hyps)
  apply auto
  done
  with lowa-lrt q' q-notNull
  show Dag q (rep  $\times$  lowa) (rep  $\times$  higha) (Node lrt q' rrt)
  by simp
qed

```

lemma (in *Repoint-impl*) *Repoint-spec'*:
shows

```

 $\forall \sigma. \Gamma \vdash \{\sigma\}$ 
 $\text{'p} ::= \text{PROC } \text{Repoint } (\text{'p})$ 
 $\{\forall \text{rept}. ((\text{Dag } ((\sigma_{\text{rep}} \times \text{id}) \sigma_{\text{p}}) (\sigma_{\text{rep}} \times \sigma_{\text{low}}) (\sigma_{\text{rep}} \times \sigma_{\text{high}}) \text{rept})$ 
 $\wedge (\forall \text{no} \in \text{set-of } \text{rept}. \sigma_{\text{rep}} \text{no} = \text{no}))$ 
 $\longrightarrow \text{Dag } \text{'p } \text{'low } \text{'high } \text{rept} \wedge$ 
 $(\forall \text{pt}. \text{pt} \notin \text{set-of } \text{rept} \longrightarrow \sigma_{\text{low}} \text{pt} = \text{'low } \text{pt} \wedge \sigma_{\text{high}} \text{pt} = \text{'high } \text{pt}))\}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply vcg
  apply (rule conjI)
  apply clarify
  prefer 2
  apply (intro impI allI )
  apply (simp add: null-comp-def)
  apply (rule conjI)
  prefer 2
  apply (clarsimp)
  apply clarify
  proof -

```

```

fix low high p rep lowa higha pa lowb highb pb rept
assume p-nNull: p ≠ Null
assume rp-nNull: rep p ≠ Null
assume rec-spec-lrept:
  ∀ rept. Dag ((rep ∘ id) (low (rep p))) (rep ∘ low) (rep ∘ high) rept
  ∧ (∀ no ∈ set-of rept. rep no = no)
  → Dag pa lowa higha rept ∧
    (∀ pt. pt ∉ set-of rept → low pt = lowa pt ∧ high pt = higha pt)
assume rec-spec-rrept:
  ∀ rept. Dag ((rep ∘ id) (higha (rep p))) (rep ∘ lowa(rep p := pa)) (rep ∘ higha)
rept
  ∧ (∀ no ∈ set-of rept. rep no = no)
  → Dag pb lowb highb rept ∧
    (∀ pt. pt ∉ set-of rept → (lowa(rep p := pa)) pt = lowb pt ∧ higha pt =
highb pt)
assume rept-dag: Dag ((rep ∘ id) p) (rep ∘ low) (rep ∘ high) rept
assume rno-rept: ∀ no ∈ set-of rept. rep no = no
show Dag (rep p) lowb (highb(rep p := pb)) rept ∧
  (∀ pt. pt ∉ set-of rept → low pt = lowb pt ∧ high pt = (highb(rep p := pb))
pt)
proof –
from rp-nNull rept-dag p-nNull obtain lrept rrept where
  rept-def: rept = Node lrept (rep p) rrept
by auto
with rept-dag p-nNull have lrept-dag:
  Dag ((rep ∘ low) (rep p)) (rep ∘ low) (rep ∘ high) lrept
by simp
from rept-def rept-dag p-nNull have rrept-dag:
  Dag ((rep ∘ high) (rep p)) (rep ∘ low) (rep ∘ high) rrept
by simp
from rno-rept rept-def have rno-lrept: ∀ no ∈ set-of lrept. rep no = no
by auto
from rno-rept rept-def have rno-rrept: ∀ no ∈ set-of rrept. rep no = no
by auto
have repoint-post-low:
  Dag pa lowa higha lrept ∧
  (∀ pt. pt ∉ set-of lrept → low pt = lowa pt ∧ high pt = higha pt)
proof –
from lrept-dag have Dag ((rep ∘ id) (low (rep p))) (rep ∘ low) (rep ∘ high)
lrept
by (simp add: id-trans)
with rec-spec-lrept rno-lrept show ?thesis
apply –
apply (erule-tac x=lrept in allE)
apply (erule impE)
apply simp
apply assumption
done
qed

```

hence $low\text{-}lowa\text{-}nc$: $(\forall pt. pt \notin set\text{-}of\ lrept \longrightarrow low\ pt = lowa\ pt \wedge high\ pt = higha\ pt)$
by *simp*
from *lrept-dag repoint-post-low* **obtain**
 $pa\text{-}def$: $pa = (rep \times low)\ (rep\ p)$ **and**
 $lowa\text{-}higha\text{-}def$: $(\forall no \in set\text{-}of\ lrept. lowa\ no = (rep \times low)\ no \wedge higha\ no = (rep \times high)\ no)$
apply $-$
apply (*drule Dags-eq-hp-eq*)
apply *auto*
done
from *rept-dag* **have** *rept-DAG*: $DAG\ rept$
by (*rule Dag-is-DAG*)
with *rept-def* **have** *rp-notin-lrept*: $rep\ p \notin set\text{-}of\ lrept$
by *simp*
from *rept-DAG rept-def* **have** *rp-notin-rrept*: $rep\ p \notin set\text{-}of\ rrept$
by *simp*
have *Dag* $((rep \times id)\ (higha\ (rep\ p)))\ (rep \times lowa(rep\ p := pa))\ (rep \times higha)$
rrept
proof $-$
from *low-lowa-nc rp-notin-lrept* **have** $(rep \times high)\ (rep\ p) = (rep \times higha)$
 $(rep\ p)$
by (*auto simp add: null-comp-def*)
with *rrept-dag* **have** *higha-mixed-rrept*:
 $Dag\ ((rep \times id)\ (higha\ (rep\ p)))\ (rep \times low)\ (rep \times high)\ rrept$
by (*simp add: id-trans*)
thm *low-high-exchange-dag*
with *low-lowa-nc lowa-higha-def rno-rrept* **have** *lowa-higha-rrept*:
 $Dag\ ((rep \times id)\ (higha\ (rep\ p)))\ (rep \times lowa)\ (rep \times higha)\ rrept$
apply $-$
apply (*rule low-high-exchange-dag*)
apply *auto*
done
have *Dag* $((rep \times id)\ (higha\ (rep\ p)))\ (rep \times lowa)\ (rep \times higha)\ rrept =$
 $Dag\ ((rep \times id)\ (higha\ (rep\ p)))\ (rep \times lowa(rep\ p := pa))\ (rep \times higha)$
rrept
proof $-$
have $\forall no \in set\text{-}of\ rrept. (rep \times lowa)\ no = (rep \times lowa(rep\ p := pa))\ no$
 \wedge
 $(rep \times higha)\ no = (rep \times higha)\ no$
proof
fix *no*
assume *no-in-rrept*: $no \in set\text{-}of\ rrept$
with *rp-notin-rrept* **have** $no \neq rep\ p$
by *blast*
thus $(rep \times lowa)\ no = (rep \times lowa(rep\ p := pa))\ no \wedge$
 $(rep \times higha)\ no = (rep \times higha)\ no$
by (*simp add: null-comp-def*)
qed

```

    thus ?thesis
      by (rule heaps-eq-Dag-eq)
  qed
  with lowa-higha-rrept show ?thesis
    by simp
  qed
  with rec-spec-rrept rno-rrept have repoint-rrept: Dag pb lowb highb rrept  $\wedge$ 
    ( $\forall pt. pt \notin \text{set-of } rrept \longrightarrow$ 
      ( $\text{lowa}(rep\ p := pa) pt = \text{lowb } pt \wedge \text{higha } pt = \text{highb } pt$ )
    )
  apply -
  apply (erule-tac x=rrept in allE)
  apply (erule impE)
  apply simp
  apply assumption
  done
  then have ab-nc: ( $\forall pt. pt \notin \text{set-of } rrept \longrightarrow$ 
    ( $\text{lowa}(rep\ p := pa) pt = \text{lowb } pt \wedge \text{higha } pt = \text{highb } pt$ )
  )
  by simp
  from repoint-rrept rrept-dag obtain
    pb-def:  $pb = ((rep \times high) (rep\ p))$  and
    lowb-highb-def: ( $\forall no \in \text{set-of } rrept. \text{lowb } no = (rep \times low) no \wedge \text{highb } no =$ 
      ( $rep \times high) no$ )
  apply -
  apply (erule Dags-eq-hp-eq)
  apply auto
  done
  have rept-end-dag: Dag (rep p) lowb (highb(rep p := pb)) rept
  proof -
    have  $\forall no \in \text{set-of } rept. \text{lowb } no = (rep \times low) no \wedge (\text{highb}(rep\ p := pb)) no$ 
      = ( $rep \times high) no$ 
    proof
      fix no
      assume no-in-rept:  $no \in \text{set-of } rept$ 
      show  $\text{lowb } no = (rep \times low) no \wedge (\text{highb}(rep\ p := pb)) no = (rep \times high)$ 
        no
    proof (cases  $no \in \text{set-of } rrept$ )
      case True
      with lowb-highb-def pb-def show ?thesis
        by simp
    next
      assume no-notin-rrept:  $no \notin \text{set-of } rrept$ 
      show ?thesis
      proof (cases  $no \in \text{set-of } lrept$ )
        case True
        with no-notin-rrept rp-notin-lrept ab-nc
        have ab-nc-no:  $\text{lowa } no = \text{lowb } no \wedge \text{higha } no = \text{highb } no$ 
        apply -
        apply (erule-tac x=no in allE)
        apply (erule impE)

```

```

    apply simp
    apply (subgoal-tac no ≠ rep p)
    apply simp
    apply blast
    done
  from lowa-higha-def True have
    lowa no = (rep ∝ low) no ∧ higha no = (rep ∝ high) no
    by auto
  with ab-nc-no have lowb no = (rep ∝ low) no ∧ highb no = (rep ∝ high)
no
    by simp
  with rp-notin-lrept True show ?thesis
    apply (subgoal-tac no ≠ rep p)
    apply simp
    apply blast
    done
next
  assume no-notin-lrept: no ∉ set-of lrept
  with no-in-rept rept-def no-notin-rrept have no-rp: no = rep p
    by simp
  with rp-notin-lrept low-low-a-nc have a-nc:
    low no = lowa no ∧ high no = higha no
    by auto
  from rp-notin-rrept no-rp ab-nc have
    (lowa(rep p := pa)) no = lowb no ∧ higha no = highb no
    by auto
  with a-nc pa-def no-rp have (rep ∝ low) no = lowb no ∧ high no =
highb no
    by auto
  with pb-def no-rp show ?thesis
    by simp
  qed
  qed
  qed
  with rept-dag have Dag (rep p) lowb (highb(rep p := pb)) rept =
    Dag (rep p) (rep ∝ low) (rep ∝ high) rept
  apply -
  thm heaps-eq-Dag-eq
  apply (rule heaps-eq-Dag-eq)
  apply auto
  done
  with rept-dag p-nNull show ?thesis
    by simp
  qed
  have (∀ pt. pt ∉ set-of rept → low pt = lowb pt ∧ high pt = (highb(rep p :=
pb)) pt)
  proof (intro allI impI)
    fix pt
    assume pt-notin-rept: pt ∉ set-of rept

```

with *rept-def* **obtain**
pt-notin-lrept: $pt \notin \text{set-of } lrept$ **and**
pt-notin-rrept: $pt \notin \text{set-of } rrept$ **and**
pt-neq-rp: $pt \neq rep\ p$
by *simp*
with *low-lowa-nc ab-nc* **show** $low\ pt = lowb\ pt \wedge high\ pt = (highb(rep\ p := pb))\ pt$
by *auto*
qed
with *rept-end-dag* **show** *?thesis*
by *simp*
qed
qed

lemma (in *Repoint-impl*) *Repoint-spec*:

shows

$\forall \sigma\ rept. \Gamma \vdash \{\sigma. Dag\ ((rep \times id)\ 'p)\ (rep \times low)\ (rep \times high)\ rept$
 $\wedge (\forall no \in \text{set-of } rept. rep\ no = no)\ \}$
 $'p := PROC\ Repoint\ ('p)$
 $\{\{Dag\ 'p\ 'low\ 'high\ rept \wedge$
 $(\forall pt. pt \notin \text{set-of } rept \longrightarrow \sigma low\ pt = 'low\ pt \wedge \sigma high\ pt = 'high\ pt)\ \}$

apply (*hoare-rule HoarePartial.ProcRec1*)

apply *vcg*

apply (*rule conjI*)

prefer 2

apply (*clarsimp simp add: null-comp-def*)

apply *clarify*

apply (*rule conjI*)

prefer 2

apply *clarsimp*

apply *clarify*

proof –

fix *rept low high rep p*

assume *rept-dag*: $Dag\ ((rep \times id)\ p)\ (rep \times low)\ (rep \times high)\ rept$

assume *rno-rept*: $\forall no \in \text{set-of } rept. rep\ no = no$

assume *p-nNull*: $p \neq Null$

assume *rp-nNull*: $rep\ p \neq Null$

show $\exists lrept.$

$Dag\ ((rep \times id)\ (low\ (rep\ p)))\ (rep \times low)\ (rep \times high)\ lrept \wedge$

$(\forall no \in \text{set-of } lrept. rep\ no = no) \wedge$

$(\forall lowa\ higha\ pa.$

$Dag\ pa\ lowa\ higha\ lrept \wedge$

$(\forall pt. pt \notin \text{set-of } lrept \longrightarrow$

$low\ pt = lowa\ pt \wedge high\ pt = higha\ pt) \longrightarrow$

$(\exists rrept.$

$Dag\ ((rep \times id)\ (higha\ (rep\ p)))\ (rep \times lowa(rep\ p := pa))$

$(rep \times higha)\ rrept \wedge$

$(\forall no \in \text{set-of } rrept. rep\ no = no) \wedge$

$(\forall lowb\ highb\ pb.$

$$\begin{aligned}
& \text{Dag } pb \text{ lowb highb rrept } \wedge \\
& (\forall pt. pt \notin \text{set-of rrept} \longrightarrow \\
& \quad (\text{lowa}(\text{rep } p := pa)) \text{ pt} = \text{lowb } pt \wedge \\
& \quad \text{higha } pt = \text{highb } pt) \longrightarrow \\
& \text{Dag } (\text{rep } p) \text{ lowb } (\text{highb}(\text{rep } p := pb)) \text{ rept } \wedge \\
& (\forall pt. pt \notin \text{set-of rept} \longrightarrow \\
& \quad \text{low } pt = \text{lowb } pt \wedge \\
& \quad \text{high } pt = (\text{highb}(\text{rep } p := pb)) \text{ pt})))
\end{aligned}$$

proof –

from *rp-nNull rept-dag p-nNull* **obtain** *lrept rrept* **where**

rept-def: $\text{rept} = \text{Node } lrept (\text{rep } p) \text{ rrept}$

by *auto*

with *rept-dag p-nNull* **have** *lrept-dag*:

$\text{Dag } ((\text{rep } \times \text{low}) (\text{rep } p)) (\text{rep } \times \text{low}) (\text{rep } \times \text{high}) \text{ lrept}$

by *simp*

from *rept-def rept-dag p-nNull* **have** *rrept-dag*:

$\text{Dag } ((\text{rep } \times \text{high}) (\text{rep } p)) (\text{rep } \times \text{low}) (\text{rep } \times \text{high}) \text{ rrept}$

by *simp*

from *rno-rept rept-def* **have** *rno-lrept*: $\forall no \in \text{set-of lrept}. \text{rep } no = no$

by *auto*

from *rno-rept rept-def* **have** *rno-rrept*: $\forall no \in \text{set-of rrept}. \text{rep } no = no$

by *auto*

show *?thesis*

apply (*rule-tac x=lrept in exI*)

apply (*rule conjI*)

apply (*simp add: id-trans lrept-dag*)

apply (*rule conjI*)

apply (*rule rno-lrept*)

apply *clarify*

subgoal premises *prems* **for** *lowa higha pa*

proof –

have *lrepta*: $\text{Dag } pa \text{ lowa higha lrept}$ **by** *fact*

have *low-lowanc*:

$\forall pt. pt \notin \text{set-of lrept} \longrightarrow \text{low } pt = \text{lowa } pt \wedge \text{high } pt = \text{higha } pt$ **by** *fact*

from *lrept-dag lrepta* **obtain**

pa-def: $pa = (\text{rep } \times \text{low}) (\text{rep } p)$ **and**

lowa-higha-def: $\forall no \in \text{set-of lrept}.$

$\text{lowa } no = (\text{rep } \times \text{low}) \text{ no} \wedge \text{higha } no = (\text{rep } \times \text{high}) \text{ no}$

apply –

apply (*drule Dags-eq-hp-eq*)

apply *auto*

done

from *rept-dag* **have** *rept-DAG*: $\text{DAG } \text{rept}$

by (*rule Dag-is-DAG*)

with *rept-def* **have** *rp-notin-lrept*: $\text{rep } p \notin \text{set-of lrept}$

by *simp*

from *rept-DAG rept-def* **have** *rp-notin-rrept*: $\text{rep } p \notin \text{set-of rrept}$

by *simp*

have *rrepta*: $\text{Dag } ((\text{rep } \times \text{id}) (\text{higha } (\text{rep } p)))$

```

      (rep  $\times$  lowa(rep p := pa)) (rep  $\times$  higha) rrept
proof –
  from low-low-a-nc rp-notin-lrept
  have (rep  $\times$  high) (rep p) = (rep  $\times$  higha) (rep p)
    by (auto simp add: null-comp-def)
  with rrept-dag have higha-mixed-rrept:
    Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  low) (rep  $\times$  high) rrept
    by (simp add: id-trans)
  thm low-high-exchange-dag
  with low-low-a-nc lowa-higha-def rno-rrept
  have lowa-higha-rrept:
    Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  lowa) (rep  $\times$  higha) rrept
  apply –
  apply (rule low-high-exchange-dag)
  apply auto
  done
have Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  lowa) (rep  $\times$  higha) rrept =
  Dag ((rep  $\times$  id) (higha (rep p)))
  (rep  $\times$  lowa(rep p := pa)) (rep  $\times$  higha) rrept
proof –
  have  $\forall no \in \text{set-of } rrept.$ 
    (rep  $\times$  lowa) no = (rep  $\times$  lowa(rep p := pa)) no  $\wedge$ 
    (rep  $\times$  higha) no = (rep  $\times$  higha) no
  proof
    fix no
    assume no-in-rrept: no  $\in$  set-of rrept
    with rp-notin-rrept have no  $\neq$  rep p
      by blast
    thus (rep  $\times$  lowa) no = (rep  $\times$  lowa(rep p := pa)) no  $\wedge$ 
      (rep  $\times$  higha) no = (rep  $\times$  higha) no
      by (simp add: null-comp-def)
    qed
  thus ?thesis
    by (rule heaps-eq-Dag-eq)
  qed
with lowa-higha-rrept show ?thesis
  by simp
qed
show ?thesis
  apply (rule-tac x=rrept in exI)
  apply (rule conjI)
  apply (rule rrepta)
  apply (rule conjI)
  apply (rule rno-rrept)
  apply clarify
  subgoal premises prems for lowb highb pb
  proof –
    have rreptb: Dag pb lowb highb rrept by fact
    have ab-nc:  $\forall pt. pt \notin \text{set-of } rrept \longrightarrow$ 

```

$(lowa(rep\ p := pa))\ pt = lowb\ pt \wedge higha\ pt = highb\ pt$ **by**

fact

from *rreptb rrept-dag* **obtain**
pb-def: $pb = ((rep \times high)\ (rep\ p))$ **and**
lowb-highb-def: $\forall no \in set-of\ rrept.$
 $lowb\ no = (rep \times low)\ no \wedge highb\ no = (rep \times high)\ no$

apply –
apply (*drule Dags-eq-hp-eq*)
apply *auto*
done

have *rept-end-dag*: $Dag\ (rep\ p)\ lowb\ (highb(rep\ p := pb))\ rept$
proof –
have $\forall no \in set-of\ rept.$
 $lowb\ no = (rep \times low)\ no \wedge (highb(rep\ p := pb))\ no = (rep \times$

high)\ no

proof
fix *no*
assume *no-in-rept*: $no \in set-of\ rept$
show $lowb\ no = (rep \times low)\ no \wedge$
 $(highb(rep\ p := pb))\ no = (rep \times high)\ no$
proof (*cases no ∈ set-of rrept*)
case *True*
with *lowb-highb-def pb-def* **show** *?thesis*
by *simp*

next
assume *no-notin-rrept*: $no \notin set-of\ rrept$
show *?thesis*
proof (*cases no ∈ set-of lrept*)
case *True*
with *no-notin-rrept rp-notin-lrept ab-nc*
have *ab-nc-no*: $lowa\ no = lowb\ no \wedge higha\ no = highb\ no$
apply –
apply (*erule-tac x=no in alle*)
apply (*erule impE*)
apply *simp*
apply (*subgoal-tac no ≠ rep p*)
apply *simp*
apply *blast*
done

from *lowa-higha-def True* **have**
 $lowa\ no = (rep \times low)\ no \wedge higha\ no = (rep \times high)\ no$
by *auto*

with *ab-nc-no*
have $lowb\ no = (rep \times low)\ no \wedge highb\ no = (rep \times high)\ no$
by *simp*

with *rp-notin-lrept True* **show** *?thesis*
apply (*subgoal-tac no ≠ rep p*)
apply *simp*
apply *blast*

```

    done
  next
  assume no-notin-lrept:  $no \notin \text{set-of } lrept$ 
  with no-in-rept rept-def no-notin-rrept have no-rp:  $no = rep\ p$ 
    by simp
  with rp-notin-lrept low-lowa-nc
  have a-nc:  $low\ no = lowa\ no \wedge high\ no = higha\ no$ 
    by auto
  from rp-notin-rrept no-rp ab-nc
  have  $(lowa(rep\ p := pa))\ no = lowb\ no \wedge higha\ no = highb\ no$ 
    by auto
  with a-nc pa-def no-rp
  have  $(rep \times low)\ no = lowb\ no \wedge high\ no = highb\ no$ 
    by auto
  with pb-def no-rp show ?thesis
    by simp
  qed
  qed
  qed
  with rept-dag
  have Dag  $(rep\ p)\ lowb\ (highb(rep\ p := pb))\ rept =$ 
     $Dag\ (rep\ p)\ (rep \times low)\ (rep \times high)\ rept$ 
  apply –
  apply (rule heaps-eq-Dag-eq)
  apply auto
  done
  with rept-dag p-nNull show ?thesis
    by simp
  qed
  have  $(\forall pt.\ pt \notin \text{set-of } rept \longrightarrow low\ pt = lowb\ pt \wedge$ 
     $high\ pt = (highb(rep\ p := pb))\ pt)$ 
  proof (intro allI impI)
    fix pt
    assume pt-notin-rept:  $pt \notin \text{set-of } rept$ 
    with rept-def obtain
      pt-notin-lrept:  $pt \notin \text{set-of } lrept$  and
      pt-notin-rrept:  $pt \notin \text{set-of } rrept$  and
      pt-neq-rp:  $pt \neq rep\ p$ 
      by simp
    with low-lowa-nc ab-nc
    show  $low\ pt = lowb\ pt \wedge high\ pt = (highb(rep\ p := pb))\ pt$ 
      by auto
    qed
  with rept-end-dag show ?thesis
    by simp
  qed
  done
  qed
  done

```

qed
qed

lemma (in *Repoint-impl*) *Repoint-spec-total*:

shows

$$\begin{aligned} & \forall \sigma \text{ rept. } \Gamma \vdash_t \{ \sigma. \text{Dag } ((\text{'rep} \times \text{id}) \text{'p}) (\text{'rep} \times \text{'low}) (\text{'rep} \times \text{'high}) \text{rept} \\ & \wedge (\forall \text{no} \in \text{set-of rept. } \text{'rep no} = \text{no}) \} \\ & \text{'p} ::= \text{PROC Repoint } (\text{'p}) \\ & \{ \text{Dag } \text{'p } \text{'low } \text{'high } \text{rept} \wedge \\ & (\forall \text{pt. pt} \notin \text{set-of rept} \longrightarrow \sigma \text{low pt} = \text{'low pt} \wedge \sigma \text{high pt} = \text{'high pt}) \} \end{aligned}$$

apply (*hoare-rule HoareTotal.ProcRec1*

[**where** $r = \text{measure } (\lambda(s,p). \text{size } (\text{dag } ((\text{'srep} \times \text{id}) \text{'s}p) (\text{'srep} \times \text{'slow}) (\text{'srep} \times \text{'shigh})))$]])

apply *vcg*

apply (*rule conjI*)

prefer 2

apply (*clarsimp simp add: null-comp-def*)

apply *clarify*

apply (*rule conjI*)

prefer 2

apply *clarsimp*

apply *clarify*

proof –

fix *rept low high rep p*

assume *rept-dag*: $\text{Dag } ((\text{rep} \times \text{id}) p) (\text{rep} \times \text{low}) (\text{rep} \times \text{high}) \text{rept}$

assume *rno-rept*: $\forall \text{no} \in \text{set-of rept. } \text{rep no} = \text{no}$

assume *p-nNull*: $p \neq \text{Null}$

assume *rp-nNull*: $\text{rep } p \neq \text{Null}$

show $\exists \text{lrept.}$

$$\begin{aligned} & \text{Dag } ((\text{rep} \times \text{id}) (\text{low } (\text{rep } p))) (\text{rep} \times \text{low}) (\text{rep} \times \text{high}) \text{lrept} \wedge \\ & (\forall \text{no} \in \text{set-of lrept. } \text{rep no} = \text{no}) \wedge \\ & \text{size } (\text{dag } ((\text{rep} \times \text{id}) (\text{low } (\text{rep } p))) (\text{rep} \times \text{low}) (\text{rep} \times \text{high})) \\ & < \text{size } (\text{dag } ((\text{rep} \times \text{id}) p) (\text{rep} \times \text{low}) (\text{rep} \times \text{high})) \wedge \\ & (\forall \text{lowa higha pa.} \end{aligned}$$

$$\text{Dag } \text{pa } \text{lowa } \text{higha } \text{lrept} \wedge$$

$$(\forall \text{pt. pt} \notin \text{set-of lrept} \longrightarrow$$

$$\text{low pt} = \text{lowa pt} \wedge \text{high pt} = \text{higha pt}) \longrightarrow$$

$$(\exists \text{rrept.}$$

$$\text{Dag } ((\text{rep} \times \text{id}) (\text{higha } (\text{rep } p))) (\text{rep} \times \text{lowa}(\text{rep } p := \text{pa}))$$

$$(\text{rep} \times \text{higha}) \text{rrept} \wedge$$

$$(\forall \text{no} \in \text{set-of rrept. } \text{rep no} = \text{no}) \wedge$$

$$\text{size } (\text{dag } ((\text{rep} \times \text{id}) (\text{higha } (\text{rep } p)))$$

$$(\text{rep} \times \text{lowa}(\text{rep } p := \text{pa})) (\text{rep} \times \text{higha}))$$

$$< \text{size } (\text{dag } ((\text{rep} \times \text{id}) p) (\text{rep} \times \text{low}) (\text{rep} \times \text{high})) \wedge$$

$$(\forall \text{lowb highb pb.}$$

$$\text{Dag } \text{pb } \text{lowb } \text{highb } \text{rrept} \wedge$$

$$(\forall \text{pt. pt} \notin \text{set-of rrept} \longrightarrow$$

$$(\text{lowa}(\text{rep } p := \text{pa})) \text{pt} = \text{lowb pt} \wedge$$

$$\begin{aligned}
& \text{higha } pt = \text{highb } pt) \longrightarrow \\
& \text{Dag } (\text{rep } p) \text{ lowb } (\text{highb}(\text{rep } p := pb)) \text{ rept} \wedge \\
& (\forall pt. pt \notin \text{set-of } \text{rept} \longrightarrow \\
& \quad \text{low } pt = \text{lowb } pt \wedge \\
& \quad \text{high } pt = (\text{highb}(\text{rep } p := pb)) \text{ pt})))
\end{aligned}$$

proof –

from *rp-nNull rept-dag p-nNull* **obtain** *lrept rrept* **where**

rept-def: *rept = Node lrept (rep p) rrept*

by *auto*

with *rept-dag p-nNull* **have** *lrept-dag*:

Dag ((rep \times low) (rep p)) (rep \times low) (rep \times high) lrept

by *simp*

from *rept-def rept-dag p-nNull* **have** *rrept-dag*:

Dag ((rep \times high) (rep p)) (rep \times low) (rep \times high) rrept

by *simp*

from *rno-rept rept-def* **have** *rno-lrept*: $\forall no \in \text{set-of } \text{lrept}. \text{rep } no = no$

by *auto*

from *rno-rept rept-def* **have** *rno-rrept*: $\forall no \in \text{set-of } \text{rrept}. \text{rep } no = no$

by *auto*

show *?thesis*

apply (*rule-tac x=lrept in exI*)

apply (*rule conjI*)

apply (*simp add: id-trans lrept-dag*)

apply (*rule conjI*)

apply (*rule rno-lrept*)

apply (*rule conjI*)

using *rept-dag rept-def*

apply (*simp only: Dag-dag*)

apply (*clarsimp simp add: id-trans Dag-dag*)

apply *clarify*

subgoal **premises** *prems* **for** *lowa higha pa*

proof –

have *lrepta*: *Dag pa lowa higha lrept* **by** *fact*

have *low-lowanc*:

$\forall pt. pt \notin \text{set-of } \text{lrept} \longrightarrow \text{low } pt = \text{lowa } pt \wedge \text{high } pt = \text{higha } pt$ **by** *fact*

from *lrept-dag lrepta* **obtain**

pa-def: *pa = (rep \times low) (rep p)* **and**

lowa-higha-def: $\forall no \in \text{set-of } \text{lrept}.$

lowa no = (rep \times low) no \wedge higha no = (rep \times high) no

apply –

apply (*drule Dags-eq-hp-eq*)

apply *auto*

done

from *rept-dag* **have** *rept-DAG*: *DAG rept*

by (*rule Dag-is-DAG*)

with *rept-def* **have** *rp-notin-lrept*: *rep p \notin set-of lrept*

by *simp*

from *rept-DAG rept-def* **have** *rp-notin-rrept*: *rep p \notin set-of rrept*

by *simp*

```

have rrepta: Dag ((rep  $\times$  id) (higha (rep p)))
                (rep  $\times$  lowa(rep p := pa)) (rep  $\times$  higha) rrept
proof –
  from low-low-a-nc rp-notin-lrept
  have (rep  $\times$  high) (rep p) = (rep  $\times$  higha) (rep p)
    by (auto simp add: null-comp-def)
  with rrept-dag have higha-mixed-rrept:
    Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  low) (rep  $\times$  high) rrept
    by (simp add: id-trans)
  thm low-high-exchange-dag
  with low-low-a-nc lowa-higha-def rno-rrept
  have lowa-higha-rrept:
    Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  lowa) (rep  $\times$  higha) rrept
  apply –
  apply (rule low-high-exchange-dag)
  apply auto
  done
have Dag ((rep  $\times$  id) (higha (rep p))) (rep  $\times$  lowa) (rep  $\times$  higha) rrept =
    Dag ((rep  $\times$  id) (higha (rep p)))
        (rep  $\times$  lowa(rep p := pa)) (rep  $\times$  higha) rrept
proof –
  have  $\forall no \in \text{set-of } rrept.$ 
    (rep  $\times$  lowa) no = (rep  $\times$  lowa(rep p := pa)) no  $\wedge$ 
    (rep  $\times$  higha) no = (rep  $\times$  higha) no
  proof
    fix no
    assume no-in-rrept: no  $\in$  set-of rrept
    with rp-notin-rrept have no  $\neq$  rep p
      by blast
    thus (rep  $\times$  lowa) no = (rep  $\times$  lowa(rep p := pa)) no  $\wedge$ 
      (rep  $\times$  higha) no = (rep  $\times$  higha) no
      by (simp add: null-comp-def)
    qed
    thus ?thesis
      by (rule heaps-eq-Dag-eq)
    qed
  with lowa-higha-rrept show ?thesis
    by simp
qed
show ?thesis
  apply (rule-tac x=rrept in exI)
  apply (rule conjI)
  apply (rule rrepta)
  apply (rule conjI)
  apply (rule rno-rrept)
  apply (rule conjI)
  using rept-dag rept-def rrepta
  apply (simp only: Dag-dag)
  apply (clarsimp simp add: id-trans Dag-dag)

```

```

apply clarify
subgoal premises prems for lowb highb pb
proof –
  have rreptb: Dag pb lowb highb rrept by fact
  have ab-nc:  $\forall pt. pt \notin \text{set-of } rrept \longrightarrow$ 
     $(\text{lowa}(\text{rep } p := pa)) \text{ pt} = \text{lowb } pt \wedge \text{higha } pt = \text{highb } pt$  by
fact

from rreptb rrept-dag obtain
  pb-def:  $pb = ((\text{rep } \times \text{high}) (\text{rep } p))$  and
  lowb-highb-def:  $\forall no \in \text{set-of } rrept.$ 
     $\text{lowb } no = (\text{rep } \times \text{low}) \text{ no} \wedge \text{highb } no = (\text{rep } \times \text{high}) \text{ no}$ 

  apply –
  apply (drule Dags-eq-hp-eq)
  apply auto
  done

have rept-end-dag: Dag (rep p) lowb (highb(rep p := pb)) rept
proof –
  have  $\forall no \in \text{set-of } rept.$ 
     $\text{lowb } no = (\text{rep } \times \text{low}) \text{ no} \wedge (\text{highb}(\text{rep } p := pb)) \text{ no} = (\text{rep } \times$ 
high) no

  proof
    fix no
    assume no-in-rept:  $no \in \text{set-of } rept$ 
    show  $\text{lowb } no = (\text{rep } \times \text{low}) \text{ no} \wedge$ 
       $(\text{highb}(\text{rep } p := pb)) \text{ no} = (\text{rep } \times \text{high}) \text{ no}$ 
    proof (cases no ∈ set-of rrept)
      case True
      with lowb-highb-def pb-def show ?thesis
      by simp
    next
    assume no-notin-rrept:  $no \notin \text{set-of } rrept$ 
    show ?thesis
    proof (cases no ∈ set-of lrept)
      case True
      with no-notin-rrept rp-notin-lrept ab-nc
      have ab-nc-no:  $\text{lowa } no = \text{lowb } no \wedge \text{higha } no = \text{highb } no$ 
      apply –
      apply (erule-tac x=no in alle)
      apply (erule impE)
      apply simp
      apply (subgoal-tac no ≠ rep p)
      apply simp
      apply blast
      done
    from lowa-higha-def True have
       $\text{lowa } no = (\text{rep } \times \text{low}) \text{ no} \wedge \text{higha } no = (\text{rep } \times \text{high}) \text{ no}$ 
      by auto
    with ab-nc-no
    have  $\text{lowb } no = (\text{rep } \times \text{low}) \text{ no} \wedge \text{highb } no = (\text{rep } \times \text{high}) \text{ no}$ 

```



```

    by simp
  with rp-notin-lrept True show ?thesis
    apply (subgoal-tac no ≠ rep p)
    apply simp
    apply blast
  done
next
  assume no-notin-lrept: no ∉ set-of lrept
  with no-in-rept rept-def no-notin-rrept have no-rp: no = rep p
    by simp
  with rp-notin-lrept low-lowa-nc
  have a-nc: low no = lowa no ∧ high no = higha no
    by auto
  from rp-notin-rrept no-rp ab-nc
  have (lowa(rep p := pa)) no = lowb no ∧ higha no = highb no
    by auto
  with a-nc pa-def no-rp
  have (rep ∝ low) no = lowb no ∧ high no = highb no
    by auto
  with pb-def no-rp show ?thesis
    by simp
qed
qed
qed
with rept-dag
have Dag (rep p) lowb (highb(rep p := pb)) rept =
  Dag (rep p) (rep ∝ low) (rep ∝ high) rept
  apply –
  apply (rule heaps-eq-Dag-eq)
  apply auto
done
with rept-dag p-nNull show ?thesis
  by simp
qed
have (∀ pt. pt ∉ set-of rept → low pt = lowb pt ∧
  high pt = (highb(rep p := pb)) pt)
proof (intro allI impI)
  fix pt
  assume pt-notin-rept: pt ∉ set-of rept
  with rept-def obtain
    pt-notin-lrept: pt ∉ set-of lrept and
    pt-notin-rrept: pt ∉ set-of rrept and
    pt-neq-rp: pt ≠ rep p
    by simp
  with low-lowa-nc ab-nc
  show low pt = lowb pt ∧ high pt = (highb(rep p := pb)) pt
    by auto
qed
with rept-end-dag show ?thesis

```

```

      by simp
    qed
  done
  qed
done
qed
qed
end

```

10 Proof of Procedure Normalize

```

theory NormalizeTotalProof imports LevellistProof ShareReduceRepListProof
  ReprintProof begin

```

```

hide-const (open) DistinctTreeProver.set-of tree.Node tree.Tip

```

```

lemma (in Normalize-impl) Normalize-modifies:

```

```

shows

```

```

   $\forall \sigma. \Gamma \vdash \{\sigma\} \ 'p ::= PROC\ Normalize\ ('p)$ 
   $\{t. t\ \text{may-only-modify-globals}\ \sigma\ \text{in}\ [rep, mark, low, high, next]\}$ 

```

```

apply (hoare-rule HoarePartial.ProcRec1)

```

```

apply (vcg spec=modifies)

```

```

done

```

```

lemma (in Normalize-impl) Normalize-spec:

```

```

shows  $\forall \sigma\ pret\ prebdt. \Gamma \vdash_t$ 

```

```

 $\{\sigma. Dag\ 'p\ 'low\ 'high\ pret \wedge ordered\ pret\ 'var \wedge$ 
 $'p \neq Null \wedge (\forall n. n \in set-of\ pret \longrightarrow 'mark\ n = 'mark\ 'p) \wedge$ 
 $bdt\ pret\ 'var = Some\ prebdt\}\}$ 

```

```

 $'p ::= PROC\ Normalize\ ('p)$ 

```

```

 $\{\{\forall pt. pt \notin set-of\ pret$ 
 $\longrightarrow \sigma_{rep}\ pt = 'rep\ pt \wedge \sigma_{low}\ pt = 'low\ pt \wedge \sigma_{high}\ pt = 'high\ pt \wedge$ 
 $\sigma_{mark}\ pt = 'mark\ pt \wedge \sigma_{next}\ pt = 'next\ pt\} \wedge$ 

```

```

 $(\exists postt. Dag\ 'p\ 'low\ 'high\ postt \wedge reduced\ postt \wedge$ 

```

```

 $shared\ postt\ \sigma_{var} \wedge ordered\ postt\ \sigma_{var} \wedge$ 

```

```

 $set-of\ postt \subseteq set-of\ pret \wedge$ 

```

```

 $(\exists postbdt. bdt\ postt\ \sigma_{var} = Some\ postbdt \wedge prebdt \sim postbdt)\} \wedge$ 

```

```

 $(\forall no. no \in set-of\ pret \longrightarrow 'mark\ no = (\neg \sigma_{mark}\ 'p))\ \}\}$ 

```

```

apply (hoare-rule HoareTotal.ProcNoRec1)

```

```

apply (hoare-rule anno=

```

```

  'levellist ::= replicate ('p  $\rightarrow$  'var + 1) Null;;

```

```

  'levellist ::= CALL Levellist ('p, ( $\neg$  'p  $\rightarrow$  'mark), 'levellist);;

```

```

  (ANNO ( $\tau, ll$ ).  $\{\tau. Levellist\ 'levellist\ 'next\ ll \wedge$ 

```

```

    Dag  $\sigma_p\ \sigma_{low}\ \sigma_{high}\ pret \wedge ordered\ pret\ \sigma_{var} \wedge \sigma_p \neq Null \wedge$ 
    (bdt pret  $\sigma_{var} = Some\ prebdt) \wedge$ 

```

```

    wf-ll pret ll  $\sigma_{var} \wedge$ 

```

```

    length 'levellist = (( $\sigma_p \rightarrow \sigma_{var}$ ) + 1)  $\wedge$ 

```

```

    wf-marking pret  $\sigma_{mark}\ 'mark\ (\neg \sigma_{mark}\ \sigma_p) \wedge$ 

```

$$\begin{aligned}
& (\forall pt. pt \notin \text{set-of pret} \longrightarrow \sigma_{\text{next pt}} = \text{'next pt}) \wedge \\
& \text{'low} = \sigma_{\text{low}} \wedge \text{'high} = \sigma_{\text{high}} \wedge \text{'p} = \sigma_p \wedge \text{'rep} = \sigma_{\text{rep}} \wedge \\
& \text{'var} = \sigma_{\text{var}} \} \\
\text{'n} ::= 0; \\
\text{WHILE } (\text{'n} < \text{length 'levellist}) \\
\text{INV } \{ & \text{Levellist 'levellist 'next ll} \wedge \\
& \text{Dag } \sigma_p \sigma_{\text{low}} \sigma_{\text{high}} \text{pret} \wedge \text{ordered pret } \sigma_{\text{var}} \wedge \sigma_p \neq \text{Null} \wedge \\
& (\text{bdt pret } \sigma_{\text{var}} = \text{Some prebdt}) \wedge \text{wf-ll pret ll } \sigma_{\text{var}} \wedge \\
& \text{length } \tau_{\text{levellist}} = ((\sigma_p \rightarrow \sigma_{\text{var}}) + 1) \wedge \\
& \text{wf-marking pret } \sigma_{\text{mark}} \tau_{\text{mark}} (\neg \sigma_{\text{mark}} \sigma_p) \wedge \\
& \tau_{\text{low}} = \sigma_{\text{low}} \wedge \tau_{\text{high}} = \sigma_{\text{high}} \wedge \tau_p = \sigma_p \wedge \tau_{\text{rep}} = \sigma_{\text{rep}} \wedge \tau_{\text{var}} = \sigma_{\text{var}} \wedge \\
& \text{'n} \leq \text{length } \tau_{\text{levellist}} \wedge \\
& (\forall pt \ i. (pt \notin \text{set-of pret} \vee (\text{'n} \leq i \wedge pt \in \text{set} (\text{ll} ! i) \wedge \\
& \quad i < \text{length } \tau_{\text{levellist}}) \\
& \quad \longrightarrow \sigma_{\text{rep pt}} = \text{'rep pt})) \wedge \\
& \text{'rep 'Nodes 'n ll} \subseteq \text{Nodes 'n ll} \wedge \\
& (\forall no \in \text{Nodes 'n ll}. \\
& \quad \text{no} \rightarrow \text{'rep} \rightarrow \sigma_{\text{var}} \leq \text{no} \rightarrow \sigma_{\text{var}} \wedge \\
& \quad (\exists \text{not nort. Dag ('rep no) ('rep } \times \sigma_{\text{low}}) ('rep } \times \sigma_{\text{high}}) \text{nort} \wedge \\
& \quad \text{Dag no } \sigma_{\text{low}} \sigma_{\text{high}} \text{not} \wedge \text{reduced nort} \wedge \\
& \quad \text{ordered nort } \sigma_{\text{var}} \wedge \text{set-of nort} \subseteq \text{'rep 'Nodes 'n ll} \wedge \\
& \quad (\forall no \in \text{set-of nort. 'rep no} = \text{no}) \wedge \\
& \quad (\exists \text{nobdt norbdt. bdt not } \sigma_{\text{var}} = \text{Some nobdt} \wedge \\
& \quad \quad \text{bdt nort } \sigma_{\text{var}} = \text{Some norbdt} \wedge \text{nobdt} \sim \text{norbdt})) \wedge \\
& (\forall t1 \ t2. \\
& \quad t1 \in \text{Dags ('rep ('Nodes 'n ll)) ('rep } \times \sigma_{\text{low}}) ('rep } \times \sigma_{\text{high}}) \wedge \\
& \quad t2 \in \text{Dags ('rep ('Nodes 'n ll)) ('rep } \times \sigma_{\text{low}}) ('rep } \times \sigma_{\text{high}}) \\
& \quad \longrightarrow \\
& \quad \text{isomorphic-dags-eq } t1 \ t2 \ \sigma_{\text{var}}) \wedge \\
& \text{'levellist} = \tau_{\text{levellist}} \wedge \text{'next} = \tau_{\text{next}} \wedge \text{'mark} = \tau_{\text{mark}} \wedge \text{'low} = \sigma_{\text{low}} \wedge \\
& \text{'high} = \sigma_{\text{high}} \wedge \text{'p} = \sigma_p \wedge \text{'var} = \sigma_{\text{var}} \} \\
\text{VAR MEASURE } (\text{length 'levellist} - \text{'n}) \\
\text{DO} \\
\text{CALL } \text{ShareReduceRepList}(\text{'levellist} ! \text{'n}); \\
\text{'n} ::= \text{'n} + 1 \\
\text{OD} \\
\{ & (\exists \text{postnormt. Dag ('rep } \sigma_p) ('rep } \times \sigma_{\text{low}}) ('rep } \times \sigma_{\text{high}}) \text{postnormt} \wedge \\
& \text{reduced postnormt} \wedge \text{shared postnormt } \sigma_{\text{var}} \wedge \\
& \text{ordered postnormt } \sigma_{\text{var}} \wedge \text{set-of postnormt} \subseteq \text{set-of pret} \wedge \\
& (\exists \text{postnormbdt. bdt postnormt } \sigma_{\text{var}} = \text{Some postnormbdt} \wedge \text{prebdt} \sim \text{post-} \\
& \text{normbdt}) \wedge \\
& (\forall no \in \text{set-of postnormt. ('rep no} = \text{no})) \wedge \\
& \text{ordered pret } \sigma_{\text{var}} \wedge \sigma_p \neq \text{Null} \wedge \\
& (\forall pt. pt \notin \text{set-of pret} \longrightarrow \sigma_{\text{rep pt}} = \text{'rep pt}) \wedge \\
& \text{'levellist} = \tau_{\text{levellist}} \wedge \text{'next} = \tau_{\text{next}} \wedge \text{'mark} = \tau_{\text{mark}} \wedge \text{'low} = \sigma_{\text{low}} \wedge \text{'high} \\
= & \sigma_{\text{high}} \wedge \\
& \text{'p} = \sigma_p \wedge (\forall no. no \in \text{set-of pret} \longrightarrow \text{'mark no} = (\neg \sigma_{\text{mark}} \sigma_p)) \} \\
& ;; \\
& \text{'p} ::= \text{CALL } \text{Repoint} (\text{'p})
\end{aligned}$$

```

    in HoareTotal.annotateI)
  apply (vcg spec=spec-total)
  prefer 2

  apply (simp add: Nodes-def null-comp-def)

  apply (rule-tac x=pret in exI)
  apply clarify
  apply (rule conjI)
  apply clarsimp
  apply (case-tac i)
  apply simp
  apply simp
  apply (rule conjI)
  apply simp
  apply (rule conjI)
  apply simp
  apply (rule conjI)
  apply simp
  apply clarify
  apply (simp (no-asm-use) only: simp-thms)
  apply (rule-tac x=ll in exI)
  apply (rule conjI)
  apply assumption
  apply clarify
  apply (simp only: simp-thms triv-forall-equality True-implies-equals)
  apply (rule-tac x=postnormt in exI)
  apply (rule conjI)
  apply simp
  apply (rule conjI)
  apply simp
  apply clarify
  apply (simp (no-asm-simp))
  prefer 2

  apply clarify
  apply (simp only: simp-thms triv-forall-equality True-implies-equals)
  apply (rule-tac x=ll!n in exI)
  apply (rule conjI)
  apply (simp add: Levellist-def)
  prefer 3

  apply (clarify)
  apply (simp (no-asm-use) only: simp-thms triv-forall-equality True-implies-equals)

proof -
  — End of while (invariant + false condition) to end of inner SPEC
  fix var p rep mark vara lowa higha pa levellista repa marka nexta varb ll
    nb pret prebdt and low :: ref ⇒ ref and

```

high :: *ref* ⇒ *ref* **and** *repb* :: *ref* ⇒ *ref*
assume *ll*: *Levellist* *levellista* *nexta* *ll*
assume *wf-lla*: *wf-ll* *pret* *ll* *var*
assume *length-lla*: *length* *levellista* = *var* *p* + 1
assume *ord-pret*: *ordered* *pret* *var*
assume *pnN*: *p* ≠ *Null*
assume *rep-repb-nc*:
 $\forall pt\ i. pt \notin \text{set-of } pret \vee nb \leq i \wedge pt \in \text{set } (ll\ !\ i) \wedge$
 $i < \text{length } levellista$
 $\longrightarrow rep\ pt = repb\ pt$

assume *wf-marking-prop*: *wf-marking* *pret* *mark* *marka* (\neg *mark* *p*)
assume *pret-dag*: *Dag* *p* *low* *high* *pret*
assume *prebdt*: *bdt* *pret* *var* = *Some* *prebdt*
assume *not-nbslla*: $\neg nb < \text{length } levellista$
assume *nb-le-lla*: $nb \leq \text{length } levellista$

assume *normalize-prop*: $\forall no \in \text{Nodes } nb\ ll.$
 $var\ (repb\ no) \leq var\ no \wedge$
 $(\exists not\ nort. Dag\ (repb\ no)\ (repb\ \times\ low)\ (repb\ \times\ high)\ nort \wedge$
 $Dag\ no\ low\ high\ not \wedge reduced\ nort \wedge ordered\ nort\ var \wedge$
 $\text{set-of } nort \subseteq repb\ \text{' } Nodes\ nb\ ll \wedge$
 $(\forall no \in \text{set-of } nort. repb\ no = no) \wedge$
 $(\exists nobdt\ norbdt. bdt\ not\ var = Some\ nobdt \wedge$
 $bdt\ nort\ var = Some\ norbdt \wedge nobdt \sim norbdt))$

assume *repbNodes-in-Nodes*: $repb\ \text{' } Nodes\ nb\ ll \subseteq Nodes\ nb\ ll$
assume *shared-mult-dags*:
 $\forall t1\ t2. t1 \in Dags\ (repb\ \text{' } Nodes\ nb\ ll)\ (repb\ \times\ low)\ (repb\ \times\ high) \wedge$
 $t2 \in Dags\ (repb\ \text{' } Nodes\ nb\ ll)\ (repb\ \times\ low)\ (repb\ \times\ high)$
 $\longrightarrow isomorphic-dags-eq\ t1\ t2\ var$

show $(\exists postnormt. Dag\ (repb\ p)\ (repb\ \times\ low)\ (repb\ \times\ high)\ postnormt \wedge$
 $reduced\ postnormt \wedge shared\ postnormt\ var \wedge$
 $ordered\ postnormt\ var \wedge \text{set-of } postnormt \subseteq \text{set-of } pret \wedge$
 $(\exists postnormbdt.$
 $bdt\ postnormt\ var = Some\ postnormbdt \wedge prebdt \sim postnormbdt) \wedge$
 $(\forall no \in \text{set-of } postnormt. repb\ no = no)) \wedge$
 $ordered\ pret\ var \wedge p \neq Null \wedge$
 $(\forall pt. pt \notin \text{set-of } pret \longrightarrow rep\ pt = repb\ pt) \wedge$
 $(\forall no. no \in \text{set-of } pret \longrightarrow marka\ no = (\neg\ mark\ p))$

proof –

from *ll* **have** *length-ll-eq*: *length* *levellista* = *length* *ll*
by (*simp* *add*: *Levellist-length*)
from *rep-repb-nc* **have** *rep-nc-post*: $\forall pt. pt \notin \text{set-of } pret \longrightarrow rep\ pt = repb\ pt$
by *auto*
from *pnN* *pret-dag* **obtain** *lt* *rt* **where** *pret-def*: *pret* = *Node* *lt* *p* *rt*
by *auto*
from *wf-marking-prop* *pret-def*
have *marking-inverted*: $(\forall no. no \in \text{set-of } pret \longrightarrow marka\ no = (\neg\ mark\ p))$

by (simp add: wf-marking-def)
 from not-nbslla nb-le-lla have nb-length-lla: $nb = \text{length } \text{levellista}$
 by simp
 with length-lla have varp-s-nb: $\text{var } p < nb$
 by simp
 from pret-def have p-in-pret: $p \in \text{set-of } \text{pret}$
 by simp
 with wf-lla have $p \in \text{set } (\text{ll } ! (\text{var } p))$
 by (simp add: wf-ll-def)
 with varp-s-nb have p-in-Nodes: $p \in \text{Nodes } nb \ \text{ll}$
 by (auto simp add: Nodes-def)
 with normalize-prop obtain not nort where
 varrepno-varno: $\text{var } (\text{repb } p) \leq \text{var } p$ and
 nort-dag: $\text{Dag } (\text{repb } p) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ \text{nort}$ and
 not-dag: $\text{Dag } p \ \text{low } \text{high } \ \text{not}$ and
 red-nort: $\text{reduced } \ \text{nort}$ and
 ord-nort: $\text{ordered } \ \text{nort } \ \text{var}$ and
 nort-in-repNodes: $\text{set-of } \ \text{nort} \subseteq \text{repb } ' \ \text{Nodes } nb \ \text{ll}$ and
 nort-repb: $(\forall \text{no} \in \text{set-of } \ \text{nort}. \ \text{repb } \ \text{no} = \text{no})$ and
 bdt-prop: $\exists \text{nobdt } \ \text{norbdtdt}. \ \text{bdt } \ \text{not } \ \text{var} = \text{Some } \ \text{nobdt} \wedge \text{bdt } \ \text{nort } \ \text{var} = \text{Some}$
 $\text{norbdtdt} \wedge$
 $\text{nobdt} \sim \text{norbdtdt}$
 by auto

 from wf-lla nb-length-lla have Nodes-in-pret: $\text{Nodes } nb \ \text{ll} \subseteq \text{set-of } \ \text{pret}$
 apply –
 apply (rule Nodes-in-pret)
 apply (auto simp add: length-ll-eq)
 done
 from pret-dag wf-lla nb-length-lla have $\text{Null} \notin \text{Nodes } nb \ \text{ll}$
 apply –
 apply (rule Null-notin-Nodes)
 apply (auto simp add: length-ll-eq)
 done
 with p-in-Nodes repbNodes-in-Nodes have rp-nNull: $\text{repb } p \neq \text{Null}$
 by auto
 with nort-dag have nort-nTip: $\text{nort} \neq \text{Tip}$
 by auto
 have $\exists \text{postnormt}. \ \text{Dag } (\text{repb } p) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ \text{postnormt} \wedge$
 $\text{reduced } \ \text{postnormt} \wedge \text{shared } \ \text{postnormt } \ \text{var} \wedge$
 $\text{ordered } \ \text{postnormt } \ \text{var} \wedge \text{set-of } \ \text{postnormt} \subseteq \text{set-of } \ \text{pret} \wedge$
 $(\exists \text{postnormbdt}. \ \text{bdt } \ \text{postnormt } \ \text{var} = \text{Some } \ \text{postnormbdt} \wedge \text{prebdt} \sim \text{postnormbdt}) \wedge$
 $(\forall \text{no} \in \text{set-of } \ \text{postnormt}. \ \text{repb } \ \text{no} = \text{no})$
 proof (rule-tac $x=\text{nort}$ in exI)
 from nort-in-repNodes repbNodes-in-Nodes Nodes-in-pret
 have nort-in-pret: $\text{set-of } \ \text{nort} \subseteq \text{set-of } \ \text{pret}$
 by blast
 from not-dag pret-dag have not-pret: $\text{not} = \text{pret}$

```

by (simp add: Dag-unique)
with bdt-prop prebdt have pret-bdt-prop:
   $\exists$  postnormbdt.
  bdt nort var = Some postnormbdt  $\wedge$  prebdt  $\sim$  postnormbdt
by auto
from shared-mult-dags have shared nort var
proof (auto simp add: shared-def isomorphic-dags-eq-def)
fix st1 st2 bdt1
assume shared-imp:
   $\forall t1 t2. t1 \in Dags (repb \text{ `Nodes nb ll} ) (repb \propto low) (repb \propto high) \wedge$ 
   $t2 \in Dags (repb \text{ `Nodes nb ll} ) (repb \propto low) (repb \propto high)$ 
   $\longrightarrow$ 
   $(\exists bdt1. bdt t1 var = Some bdt1 \wedge bdt t2 var = Some bdt1) \longrightarrow t1 = t2$ 
assume st1-nort: st1  $\leq$  nort
assume st2-nort: st2  $\leq$  nort
assume bdt-st1: bdt st1 var = Some bdt1
assume bdt-st2: bdt st2 var = Some bdt1
from nort-in-repNodes nort-dag nort-nTip
have nort-in-DagsrNodes:
  nort  $\in$  Dags (repb `Nodes nb ll) (repb  $\propto$  low) (repb  $\propto$  high)
apply  $-$ 
apply (rule DagsI)
apply auto
done
show st1 = st2
proof (cases st1)
  case Tip
  note st1-Tip=this
  with bdt-st1 bdt-st2 show ?thesis
  by auto
next
  case (Node lst1 st1p rst1)
  note st1-Node=this
  then have st1-nTip: st1  $\neq$  Tip
  by simp
  show ?thesis
  proof (cases st2)
    case Tip
    with bdt-st1 bdt-st2 show ?thesis
    by auto
  next
    case (Node lst2 st2p rst2)
    note st2-Node=this
    then have st2-nTip: st2  $\neq$  Tip
    by simp
    from nort-in-DagsrNodes st1-nort ord-nort wf-lla st1-nTip
    have st1-in-Dags:
      st1  $\in$  Dags (repb `Nodes nb ll) (repb  $\propto$  low) (repb  $\propto$  high)
    apply  $-$ 

```

```

    apply (rule Dags-subdags)
    apply auto
    done
  from nort-in-DagsrNodes st2-nort ord-nort wf-lla st2-nTip
  have st2-in-Dags:
    st2 ∈ Dags (repb ‘ Nodes nb ll) (repb ∝ low) (repb ∝ high)
    apply –
    apply (rule Dags-subdags)
    apply auto
    done
  from st1-in-Dags st2-in-Dags bdt-st1 bdt-st2 shared-imp show st1=st2
  by simp
qed
qed
qed
with nort-dag red-nort ord-nort nort-in-pret pret-bdt-prop nort-repb
show Dag (repb p) (repb ∝ low) (repb ∝ high) nort ∧
  reduced nort ∧ shared nort var ∧ ordered nort var ∧
  set-of nort ⊆ set-of pret ∧
  (∃ postnormbdt.
    bdt nort var = Some postnormbdt ∧ prebdt ∼ postnormbdt) ∧
  (∀ no ∈ set-of nort. repb no = no)
  apply –
  apply (intro conjI)
  apply assumption+
  done
qed
with wf-lla length-lla ord-pret pnN rep-nc-post marking-inverted
show ?thesis
  by simp
qed
next
— From postcondition inner SPEC to final postcondition
fix var low high p rep levellist marka next
  nexta lowb highb pb levellista ll repa pret prebdt
  and mark::ref⇒bool and postnormt postnormbdt
assume ll: Levellist levellista nexta ll
assume repoint-spec:
  Dag pb lowb highb postnormt
  ∀ pt. pt ∉ set-of postnormt → low pt = lowb pt ∧ high pt = highb pt
assume pret-dag: Dag p low high pret
assume ord-pret: ordered pret var
assume pnN: p ≠ Null
assume onemark-pret:
  ∀ n. n ∈ set-of pret → mark n = mark p (is ∀ n. ?in-pret n → ?eq-mark-p n)
assume pret-bdt: bdt pret var = Some prebdt

assume wf-ll: wf-ll pret ll var and
  length-ll:length levellist = var p + 1 and

```


wf-marking-ll: *wf-marking pret mark marka* (\neg *mark p*)
assume
postnormt-dag: *Dag (repa p) (repa \times low) (repa \times high) postnormt* **and**
reduced-postnormt: *reduced postnormt* **and**
shared-postnormt: *shared postnormt var* **and**
ordered-postnormt: *ordered postnormt var* **and**
subset-pret: *set-of postnormt \subseteq set-of pret***and**
sim-bdt: *bdt postnormt var = Some postnormbdt prebdt \sim postnormbdt* **and**
postdag-repa: $\forall no \in \text{set-of postnormt. repa } no = no$ **and**
rep-eq: $\forall pt. pt \notin \text{set-of pret} \longrightarrow \text{rep } pt = \text{repa } pt$ **and**
pret-marked: $\forall no. no \in \text{set-of pret} \longrightarrow \text{marka } no = (\neg \text{mark } p)$
assume *unmodif-next*: $\forall p. p \notin \text{set-of pret} \longrightarrow \text{next } p = \text{nexta } p$
show ($\forall pt. pt \notin \text{set-of pret}$
 $\longrightarrow \text{low } pt = \text{lowb } pt \wedge \text{high } pt = \text{highb } pt \wedge$
 $\text{mark } pt = \text{marka } pt$)

proof –
from *ll* **have** *length-ll-eq*: *length levellista = length ll*
by (*simp add: Levellist-length*)
from *repoint-spec pnN subset-pret*
have *repoint-nc*: ($\forall pt. pt \notin \text{set-of pret}$
 $\longrightarrow \text{low } pt = \text{lowb } pt \wedge \text{high } pt = \text{highb } pt$) \wedge *Dag pb lowb highb postnormt*
by *auto*
then have *lowhigh-b-eq*: $\forall pt. pt \notin \text{set-of pret}$
 $\longrightarrow \text{low } pt = \text{lowb } pt \wedge \text{high } pt = \text{highb } pt$
by *fastforce*
from *wf-marking-ll pret-dag pnN*
have *mark-b-eq*: $\forall pt. pt \notin \text{set-of pret} \longrightarrow \text{mark } pt = \text{marka } pt$
apply –
apply (*simp add: wf-marking-def*)
apply (*split dag.splits*)
apply *simp*
apply (*rule allI*)
apply (*rule impI*)
apply (*elim conjE*)
apply (*erule-tac x=pt in allE*)
apply *fastforce*
done
with *lowhigh-b-eq rep-eq unmodif-next*
have *pret-nc*: $\forall pt. pt \notin \text{set-of pret}$
 $\longrightarrow \text{rep } pt = \text{repa } pt \wedge \text{low } pt = \text{lowb } pt \wedge \text{high } pt = \text{highb } pt \wedge$
 $\text{mark } pt = \text{marka } pt \wedge \text{next } pt = \text{nexta } pt$
by *blast*

from *pret-nc*
show *?thesis*
by *fastforce*
qed
next

— invariant to invariant

fix *var low high p rep mark pret prebdt levellist ll next marka n repc*
and *repb :: ref \Rightarrow ref*
assume *ll: Levellist levellist next ll*
assume *pret-dag: Dag p low high pret*
assume *ord-pret: ordered pret var*
assume *pnN: p \neq Null*
assume *prebdt-pret: bdt pret var = Some prebdt*
assume *wf-ll: wf-ll pret ll var*
assume *lll: length levellist = var p + 1*
assume *n-Suc-var-p: n < var p + 1*
assume *wf-marking-m-ma: wf-marking pret mark marka (\neg mark p)*

assume *rep-nc: $\forall pt i.$
 $pt \notin \text{set-of } pret \vee n \leq i \wedge pt \in \text{set } (ll ! i) \wedge i < \text{var } p + 1 \longrightarrow$
 $rep \ pt = repb \ pt$*

assume *repbNodes-in-Nodes: repb ‘ Nodes n ll \subseteq Nodes n ll*
assume
*normalize-prop: $\forall no \in \text{Nodes } n \ ll.$
 $var (repb \ no) \leq var \ no \wedge$
 $(\exists \text{not } nort. \text{Dag } (repb \ no) (repb \ \alpha \ low) (repb \ \alpha \ high) \ nort \wedge$
 $\text{Dag } no \ low \ high \ not \wedge \text{reduced } nort \wedge \text{ordered } nort \ var \wedge$
 $\text{set-of } nort \subseteq \text{repb ‘ Nodes } n \ ll \wedge$
 $(\forall no \in \text{set-of } nort. \text{repb } no = no) \wedge$
 $(\exists \text{nobdt. bdt } not \ var = \text{Some } nobdt \wedge$
 $(\exists \text{norbdt. bdt } nort \ var = \text{Some } norbdt \wedge$
 $nobdt \sim norbdt)))$*

assume
*isomorphic-dags-eq:
 $\forall t1 \ t2. t1 \in \text{Dags } (repb \ ‘ \ \text{Nodes } n \ ll) (repb \ \alpha \ low) (repb \ \alpha \ high) \wedge$
 $t2 \in \text{Dags } (repb \ ‘ \ \text{Nodes } n \ ll) (repb \ \alpha \ low) (repb \ \alpha \ high)$
 $\longrightarrow \text{isomorphic-dags-eq } t1 \ t2 \ var$*

show $(\forall no \in \text{set } (ll ! n).$
 $no \neq \text{Null} \wedge$
 $(low \ no = \text{Null}) = (high \ no = \text{Null}) \wedge$
 $low \ no \notin \text{set } (ll ! n) \wedge$
 $high \ no \notin \text{set } (ll ! n) \wedge$
 $\text{isLeaf-pt } no \ low \ high = (var \ no \leq 1) \wedge$
 $(low \ no \neq \text{Null} \longrightarrow \text{repb } (low \ no) \neq \text{Null}) \wedge (repb \ \alpha \ low) \ no \notin \text{set } (ll$
 $! \ n)) \wedge$
 $(\forall no1 \in \text{set } (ll ! n). \forall no2 \in \text{set } (ll ! n). \text{var } no1 = \text{var } no2) \wedge$
 $(\forall \text{repa}. (\forall no. no \notin \text{set } (ll ! n) \longrightarrow \text{repb } no = \text{repa } no) \wedge$
 $(\forall no \in \text{set } (ll ! n).$
 $\text{repa } no \neq \text{Null} \wedge$
 $(\text{if } (repb \ \alpha \ low) \ no = (repb \ \alpha \ high) \ no \wedge \text{low } no \neq \text{Null}$
 $\text{then } \text{repa } no = (repb \ \alpha \ low) \ no$
 $\text{else } \text{repa } no \in \text{set } (ll ! n) \wedge$
 $\text{repa } (\text{repa } no) = \text{repa } no \wedge$

$(\forall no1 \in set (ll ! n).$
 $((repa \times high) no1 = (repa \times high) no \wedge$
 $(repa \times low) no1 = (repa \times low) no) =$
 $(repa no = repa no1)))) \longrightarrow$
 $var p + 1 - (n + 1) < var p + 1 - n \wedge$
 $n + 1 \leq var p + 1 \wedge$
 $(\forall pt \ i. pt \notin set-of pret \vee (n + 1 \leq i \wedge pt \in set (ll ! i) \wedge i < var p$
 $+ 1) \longrightarrow$
 $rep pt = repa pt) \wedge$
 $repa \text{ ' } Nodes (n + 1) ll \subseteq Nodes (n + 1) ll \wedge$
 $(\forall no \in Nodes (n + 1) ll.$
 $var (repa no) \leq var no \wedge$
 $(\exists not nort.$
 $Dag (repa no) (repa \times low) (repa \times high) nort \wedge$
 $Dag no low high not \wedge$
 $reduced nort \wedge$
 $ordered nort var \wedge$
 $set-of nort \subseteq repa \text{ ' } Nodes (n + 1) ll \wedge$
 $(\forall no \in set-of nort. repa no = no) \wedge$
 $(\exists nobdt.$
 $bdt not var = Some nobdt \wedge$
 $(\exists norbdt. bdt nort var = Some norbdt \wedge nobdt \sim norbdt))))$
 \wedge
 $(\forall t1 \ t2.$
 $t1 \in Dags (repa \text{ ' } Nodes (n + 1) ll) (repa \times low) (repa \times high) \wedge$
 $t2 \in Dags (repa \text{ ' } Nodes (n + 1) ll) (repa \times low) (repa \times high)$
 \longrightarrow
 $isomorphic-dags-eq t1 t2 var))$
proof –
from ll **have** $length-ll-eq$: $length\ lellist = length\ ll$
by (*simp add: Levellist-length*)
from n -*Suc-var-p* lll **have** nsl : $n < length\ lellist$ **by** *simp*
hence $nseqll$: $n \leq length\ lellist$ **by** *simp*
have $srrel-precond$: $(\forall no \in set (ll ! n).$
 $no \neq Null \wedge$
 $(low\ no = Null) = (high\ no = Null) \wedge$
 $low\ no \notin set (ll ! n) \wedge$
 $high\ no \notin set (ll ! n) \wedge$
 $isLeaf-pt\ no\ low\ high = (var\ no \leq 1) \wedge$
 $(low\ no \neq Null \longrightarrow repb (low\ no) \neq Null) \wedge$
 $(repb \times low) no \notin set (ll ! n))$
proof (*intro ballI*)
fix no
assume $no-in-lln$: $no \in set (ll ! n)$
with $wf-ll\ nsl$ **have** $no-in-pret-var$: $no \in set-of\ pret \wedge var\ no = n$
by (*simp add: wf-ll-def length-ll-eq*)
with $pret-dag$ **have** $no-nNull$: $no \neq Null$
apply –
apply (*rule set-of-nn*)

```

    apply auto
  done
from pret-dag prebdt-pret no-in-pret-var
have balanced-no: (low no = Null) = (high no = Null)
  apply -
  apply (erule conjE)
  apply (rule-tac p=p and low=low in balanced-bdt)
  apply auto
  done
have low-no-notin-lln: low no  $\notin$  set (ll ! n)
proof (cases low no = Null)
  case True
  note lno-Null=this
  with balanced-no have hno-Null: high no = Null
  by simp
  show ?thesis
  proof (cases low no  $\in$  set (ll ! n))
    case True
    with wf-ll nsll have low no  $\in$  set-of pret  $\wedge$  var (low no) = n
      by (auto simp add: wf-ll-def length-ll-eq)
    with pret-dag have low no  $\neq$  Null
    apply -
    apply (rule set-of-nn)
    apply auto
    done
  with lno-Null show ?thesis
  by simp
  next
  assume lno-notin-lln: low no  $\notin$  set (ll ! n)
  then show ?thesis
  by simp
qed
next
assume lno-nNull: low no  $\neq$  Null
with balanced-no have hno-nNull: high no  $\neq$  Null
  by simp
with lno-nNull pret-dag ord-pret no-in-pret-var
have var-children-smaller: var (low no) < var no  $\wedge$  var (high no) < var no
  apply -
  apply (rule var-ordered-children)
  apply auto
  done
show ?thesis
proof (cases low no  $\in$  set (ll ! n))
  case True
  with wf-ll nsll have low no  $\in$  set-of pret  $\wedge$  var (low no) = n
    by (simp add: wf-ll-def length-ll-eq)
  with var-children-smaller no-in-pret-var show ?thesis
  by simp

```

```

next
  assume  $low\ no \notin set\ (ll\ !\ n)$ 
  thus ?thesis
  by simp
qed
qed
have  $high\ no\ notin\ ll\ n: high\ no \notin set\ (ll\ !\ n)$ 
proof (cases  $high\ no = Null$ )
  case True
  note  $hno\ -Null = this$ 
  with  $balanced\ no$  have  $lno\ -Null: low\ no = Null$ 
  by simp
  show ?thesis
  proof (cases  $high\ no \in set\ (ll\ !\ n)$ )
    case True
    with  $wf\ ll\ nsll$  have  $high\ no \in set\ of\ pret \wedge var\ (high\ no) = n$ 
    by (auto simp add:  $wf\ ll\ def\ length\ ll\ eq$ )
    with  $pret\ dag$  have  $high\ no \neq Null$ 
    apply -
    apply (rule  $set\ of\ nn$ )
    apply auto
    done
  with  $hno\ -Null$  show ?thesis
  by simp
  next
  assume  $hno\ notin\ ll\ n: high\ no \notin set\ (ll\ !\ n)$ 
  then show ?thesis
  by simp
  qed
next
next
assume  $hno\ n\ -Null: high\ no \neq Null$ 
with  $balanced\ no$  have  $lno\ n\ -Null: low\ no \neq Null$ 
  by simp
with  $hno\ n\ -Null\ pret\ dag\ ord\ pret\ no\ in\ pret\ var$ 
have  $var\ children\ smaller: var\ (low\ no) < var\ no \wedge var\ (high\ no) < var\ no$ 
  apply -
  apply (rule  $var\ ordered\ children$ )
  apply auto
  done
show ?thesis
proof (cases  $high\ no \in set\ (ll\ !\ n)$ )
  case True
  with  $wf\ ll\ nsll$  have  $high\ no \in set\ of\ pret \wedge var\ (high\ no) = n$ 
  by (simp add:  $wf\ ll\ def\ length\ ll\ eq$ )
  with  $var\ children\ smaller\ no\ in\ pret\ var$  show ?thesis
  by simp
  next
  assume  $high\ no \notin set\ (ll\ !\ n)$ 
  thus ?thesis

```

```

    by simp
  qed
qed
from no-in-pret-var pret-dag no-nNull obtain not where
  no-dag-ex: Dag no low high not
  apply -
  apply (rotate-tac 2)
  apply (drule subnode-dag-cons)
  apply (auto simp del: Dag-Ref)
  done
with pret-dag prebdt-pret no-in-pret-var obtain nobdt
  where nobdt-ex:
    bdt not var = Some nobdt
  apply -
  apply (drule subbdt-ex-dag-def)
  apply auto
  done
have isLeaf-var: isLeaf-pt no low high = (var no ≤ 1)
proof
  assume no-isLeaf: isLeaf-pt no low high
  from nobdt-ex no-dag-ex no-isLeaf show var no ≤ 1
  apply -
  apply (rule bdt-Some-Leaf-var-le-1)
  apply auto
  done
next
  assume varno-le-1: var no ≤ 1
  show isLeaf-pt no low high
  proof (cases var no = 0)
    case True
      with nobdt-ex no-nNull no-dag-ex have not = Node Tip no Tip
        apply -
        apply (drule bdt-Some-var0-Zero)
        apply auto
        done
      with no-dag-ex show isLeaf-pt no low high
        by (simp add: isLeaf-pt-def)
    next
      assume var no ≠ 0
      with varno-le-1 have var no = 1
        by simp
      with nobdt-ex no-nNull no-dag-ex have not = Node Tip no Tip
        apply -
        apply (drule bdt-Some-var1-One)
        apply auto
        done
      with no-dag-ex show isLeaf-pt no low high
        by (simp add: isLeaf-pt-def)
  qed

```

```

qed
have repb-low-nNull: (low no  $\neq$  Null  $\longrightarrow$  repb (low no)  $\neq$  Null)
proof
  assume lno-nNull: low no  $\neq$  Null
  with no-nNull no-in-pret-var pret-dag have lno-in-pret: low no  $\in$  set-of pret
  apply -
  apply (rule-tac low=low in subelem-set-of-low)
  apply auto
  done
  from lno-nNull balanced-no have hno-nNull: high no  $\neq$  Null
  by simp
  with lno-nNull pret-dag ord-pret no-in-pret-var
  have var-children-smaller: var (low no) < var no  $\wedge$  var (high no) < var no
  apply -
  apply (rule var-ordered-children)
  apply auto
  done
  with no-in-pret-var have var-lno-l-n: var (low no) < n
  by simp
  with wf-ll lno-in-pret nsll have low no  $\in$  set (ll ! (var (low no)))
  by (simp add: wf-ll-def length-ll-eq)
  with lno-in-pret var-lno-l-n have low no  $\in$  Nodes n ll
  apply (simp add: Nodes-def)
  apply (rule-tac x=var (low no) in exI)
  apply simp
  done
  hence repb (low no)  $\in$  repb ' Nodes n ll
  by simp
  with repbNodes-in-Nodes have repb-lno-in-Nodes:
    repb (low no)  $\in$  Nodes n ll
  by blast
  from pret-dag wf-ll nsll have Null  $\notin$  Nodes n ll
  apply -
  apply (rule Null-notin-Nodes)
  apply (auto simp add: length-ll-eq)
  done
  with repb-lno-in-Nodes show repb (low no)  $\neq$  Null
  by auto
qed
have Null-notin-lln: Null  $\notin$  set (ll ! n)
proof (cases Null  $\in$  set (ll ! n))
  case True
  with wf-ll nsll have Null  $\in$  set-of pret  $\wedge$  var (Null) = n
  by (simp add: wf-ll-def length-ll-eq)
  with pret-dag have Null  $\neq$  Null
  apply -
  apply (rule set-of-nn)
  apply auto
  done

```

```

    thus ?thesis
      by auto
  next
    assume  $Null \notin set\ ll\ !\ n$ 
    thus ?thesis
      by simp
  qed
  have  $(repb \times low)\ no \notin set\ ll\ !\ n$ 
  proof (cases  $low\ no = Null$ )
    case True
    with  $Null\ notin\ ll\ n$  show ?thesis
      by (simp add: null-comp-def)
  next
    assume  $lno\ n\ Null: low\ no \neq Null$ 
    with  $no\ n\ Null\ no\ in\ pret\ var\ pret\ dag$  have  $lno\ in\ pret: low\ no \in set\ of\ pret$ 
    apply -
    apply (rule-tac  $low = low$  in subelem-set-of-low)
    apply auto
    done
  from  $lno\ n\ Null$  have propto-eq-comp:  $(repb \times low)\ no = repb\ (low\ no)$ 
    by (simp add: null-comp-def)
  from  $lno\ n\ Null\ balanced\ no$  have  $hno\ n\ Null: high\ no \neq Null$ 
    by simp
  with  $lno\ n\ Null\ pret\ dag\ ord\ pret\ no\ in\ pret\ var$ 
  have  $var\ children\ smaller: var\ (low\ no) < var\ no \wedge var\ (high\ no) < var\ no$ 
    apply -
    apply (rule var-ordered-children)
    apply auto
    done
  with  $no\ in\ pret\ var$  have  $var\ lno\ l\ n: var\ (low\ no) < n$ 
    by simp
  with  $wf\ ll\ lno\ in\ pret\ nsll$  have  $low\ no \in set\ ll\ !\ (var\ (low\ no))$ 
    by (simp add: wf-ll-def length-ll-eq)
  with  $lno\ in\ pret\ var\ lno\ l\ n$  have  $lno\ in\ Nodes\ n: low\ no \in Nodes\ n\ ll$ 
    apply (simp add: Nodes-def)
    apply (rule-tac  $x = var\ (low\ no)$  in exI)
    apply simp
    done
  hence  $repb\ (low\ no) \in repb\ 'Nodes\ n\ ll$ 
    by simp
  with  $repb\ Nodes\ in\ Nodes$  have  $repb\ lno\ in\ Nodes:$ 
     $repb\ (low\ no) \in Nodes\ n\ ll$ 
    by blast
  with  $lno\ in\ Nodes\ n\ normalize\ prop$  have  $var\ (repb\ (low\ no)) \leq var\ (low$ 
no)
    by auto
  with  $var\ lno\ l\ n$  have  $var\ rep\ lno\ l\ n: var\ (repb\ (low\ no)) < n$ 
    by simp
  with  $repb\ lno\ in\ Nodes$  have  $\exists k < n. repb\ (low\ no) \in set\ ll\ !\ k$ 

```



```

    by (auto simp add: Nodes-def)
  with wf-ll propto-eq-comp nsll show (repb  $\propto$  low) no  $\notin$  set (ll ! n)
  apply -
  apply (erule exE)
  apply (rule-tac i=k and j=n in no-in-one-ll)
  apply (auto simp add: length-ll-eq)
  done
qed
with no-nNull balanced-no low-no-notin-lln high-no-notin-lln isLeaf-var repb-low-nNull
show no  $\neq$  Null  $\wedge$ 
  (low no = Null) = (high no = Null)  $\wedge$ 
  low no  $\notin$  set (ll ! n)  $\wedge$  high no  $\notin$  set (ll ! n)  $\wedge$ 
  isLeaf-pt no low high = (var no  $\leq$  1)  $\wedge$ 
  (low no  $\neq$  Null  $\longrightarrow$  repb (low no)  $\neq$  Null)  $\wedge$ 
  (repb  $\propto$  low) no  $\notin$  set (ll ! n)
  by auto
qed
have all-nodes-same-var:  $\forall$  no1  $\in$  set (ll ! n).  $\forall$  no2  $\in$  set (ll ! n). var no1 =
var no2
proof (intro ballI impI)
  fix no1 no2
  assume no1  $\in$  set (ll ! n)
  with wf-ll nsll have var-lln-i: var no1 = n
  by (simp add: wf-ll-def length-ll-eq)
  assume no2  $\in$  set (ll ! n)
  with wf-ll nsll have var no2 = n
  by (simp add: wf-ll-def length-ll-eq)
  with var-lln-i show var no1 = var no2
  by simp
qed
have ( $\forall$  repa. ( $\forall$  no. no  $\notin$  set (ll ! n)  $\longrightarrow$  repb no = repa no)  $\wedge$ 
  ( $\forall$  no $\in$ set (ll ! n).
    repa no  $\neq$  Null  $\wedge$ 
    (if (repa  $\propto$  low) no = (repa  $\propto$  high) no  $\wedge$  low no  $\neq$  Null
    then repa no = (repa  $\propto$  low) no
    else repa no  $\in$  set (ll ! n)  $\wedge$ 
    repa (repa no) = repa no  $\wedge$ 
    ( $\forall$  no1 $\in$ set (ll ! n).
      ((repa  $\propto$  high) no1 = (repa  $\propto$  high) no  $\wedge$ 
      (repa  $\propto$  low) no1 = (repa  $\propto$  low) no) =
      (repa no = repa no1))))  $\longrightarrow$ 
  var p + 1 - (n + 1) < var p + 1 - n  $\wedge$ 
  n + 1  $\leq$  var p + 1  $\wedge$ 
  ( $\forall$  pt i. pt  $\notin$  set-of pret  $\vee$  (n + 1  $\leq$  i  $\wedge$  pt  $\in$  set (ll ! i)  $\wedge$  i < var p
+ 1)  $\longrightarrow$ 
  rep pt = repa pt)  $\wedge$ 
  repa 'Nodes (n + 1) ll  $\subseteq$  Nodes (n + 1) ll  $\wedge$ 
  ( $\forall$  no $\in$ Nodes (n + 1) ll.
    var (repa no)  $\leq$  var no  $\wedge$ 

```

$(\exists \text{ not nort.}$
 $\text{Dag (repa no) (repa } \times \text{ low) (repa } \times \text{ high) nort } \wedge$
 $\text{Dag no low high not } \wedge$
 $\text{reduced nort } \wedge$
 $\text{ordered nort var } \wedge$
 $\text{set-of nort } \subseteq \text{repa 'Nodes (n + 1) ll } \wedge$
 $(\forall \text{ no} \in \text{set-of nort. repa no = no) } \wedge$
 $(\exists \text{ nobdt.}$
 $\text{ bdt not var = Some nobdt } \wedge$
 $(\exists \text{ norbdt. bdt nort var = Some norbdt } \wedge \text{ nobdt } \sim \text{ norbdt})))$

\wedge

$(\forall t1 t2.$
 $t1 \in \text{Dags (repa 'Nodes (n + 1) ll) (repa } \times \text{ low) (repa } \times \text{ high) } \wedge$
 $t2 \in \text{Dags (repa 'Nodes (n + 1) ll) (repa } \times \text{ low) (repa } \times \text{ high)}$

\longrightarrow

$\text{isomorphic-dags-eq t1 t2 var))}$
 $(\text{is } (\forall \text{ repc. ?srrl-post repc } \longrightarrow \text{?norm-inv repc))$
proof $(\text{intro allI impI, elim conjE})$
fix repc
assume $\text{repbc-nc: } \forall \text{ no. no } \notin \text{set (ll ! n) } \longrightarrow \text{repb no = repc no}$
assume $\text{rep-prop: } \forall \text{ no} \in \text{set (ll ! n).}$
 $\text{repc no } \neq \text{Null } \wedge$
 $(\text{if (repc } \times \text{ low) no = (repc } \times \text{ high) no } \wedge \text{low no } \neq \text{Null}$
 $\text{then repc no = (repc } \times \text{ low) no}$
 $\text{else repc no } \in \text{set (ll ! n) } \wedge$
 $\text{repc (repc no) = repc no } \wedge$
 $(\forall \text{ no1} \in \text{set (ll ! n).}$
 $((\text{repc } \times \text{ high) no1 = (repc } \times \text{ high) no } \wedge$
 $(\text{repc } \times \text{ low) no1 = (repc } \times \text{ low) no) =}$
 $(\text{repc no = repc no1})))$

show ?norm-inv repc
proof $-$
from $\text{n-Suc-var-p have termi: var p + 1 - (n + 1) < var p + 1 - n}$
by arith
from $\text{wf-ll repbc-nc nsll}$
have $\text{Nodes-n-rep-nc: } \forall p. p \in \text{Nodes n ll } \longrightarrow \text{repb p = repc p}$
apply $-$
apply (rule allI)
apply (rule impI)
apply $(\text{simp add: Nodes-def})$
apply (erule exE)
apply $(\text{erule-tac x=p in allE})$
apply $(\text{drule-tac i=x and j=n in no-in-one-ll})$
apply $(\text{auto simp add: length-ll-eq})$
done
from $\text{repbNodes-in-Nodes}$
have $\text{Nodes-n-rep-in-Nodesn:}$
 $\forall p. p \in \text{Nodes n ll } \longrightarrow \text{repb p } \in \text{Nodes n ll}$
by auto

```

from wf-ll nsll have Nodes n ll  $\subseteq$  set-of pret
  apply –
  apply (rule Nodes-in-pret)
  apply (auto simp add: length-ll-eq)
  done
with Nodes-n-rep-in-Nodesn
have Nodes-n-rep-in-pret:  $\forall p. p \in \text{Nodes } n \text{ ll} \longrightarrow \text{repb } p \in \text{set-of pret}$ 
  apply –
  apply (intro allI impI)
  apply blast
  done
have Nodes-repbc-Dags-eq:  $\forall p t. p \in \text{Nodes } n \text{ ll} \longrightarrow \text{Dag } (\text{repb } p) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) t = \text{Dag } (\text{repc } p) (\text{repc } \times \text{low}) (\text{repc } \times \text{high}) t$ 
proof (intro allI impI)
  fix p t
  assume p-in-Nodes:  $p \in \text{Nodes } n \text{ ll}$ 
  then have repp-nc:  $\text{repb } p = \text{repc } p$ 
    by (rule Nodes-n-rep-nc [rule-format])
  from p-in-Nodes normalize-prop obtain nort where
    nort-repb-dag:  $\text{Dag } (\text{repb } p) (\text{repb } \times \text{low}) (\text{repb } \times \text{high})$  nort and
    nort-in-repbNodes:  $\text{set-of nort} \subseteq \text{repb } \text{ ` Nodes } n \text{ ll}$ 
  apply –
  apply (erule-tac x=p in ballE)
  prefer 2
  apply auto
  done
from nort-in-repbNodes repbNodes-in-Nodes
have nort-in-Nodesn:  $\text{set-of nort} \subseteq \text{Nodes } n \text{ ll}$ 
  by blast
from pret-dag wf-ll nsll have Null  $\notin$  Nodes n ll
  apply –
  apply (rule Null-notin-Nodes)
  apply (auto simp add: length-ll-eq)
  done
with p-in-Nodes repbNodes-in-Nodes have repp-nNull:  $\text{repb } p \neq \text{Null}$ 
  by auto
from nort-repb-dag repp-nc
have nort-repbc-dag:  $\text{Dag } (\text{repc } p) (\text{repb } \times \text{low}) (\text{repb } \times \text{high})$  nort
  by simp
from nort-in-Nodesn have  $\forall x \in \text{set-of nort}. x \in \text{Nodes } n \text{ ll}$ 
  apply –
  apply (rule ballI)
  apply blast
  done
with wf-ll nsll have  $\forall x \in \text{set-of nort}. x \in \text{set-of pret} \wedge \text{var } x < n$ 
  apply –
  apply (rule ballI)
  apply (rule wf-ll-Nodes-pret)

```

```

    apply (auto simp add: length-ll-eq)
  done
with pret-dag prebdt-pret nort-repbc-dag ord-pret wf-ll nsll repbc-nc
have
   $\forall x \in \text{set-of nort. } (\text{repc} \times \text{low}) x = (\text{repb} \times \text{low}) x \wedge$ 
   $(\text{repc} \times \text{high}) x = (\text{repb} \times \text{high}) x$ 
  apply -
  apply (rule nort-null-comp)
  apply (auto simp add: length-ll-eq)
  done
with nort-repbc-dag repp-nc
have Dag (repc p) (repb  $\times$  low) (repb  $\times$  high) nort =
  Dag (repc p) (repc  $\times$  low) (repc  $\times$  high) nort
  apply -
  apply (rule heaps-eq-Dag-eq)
  apply (rule ballI)
  apply (erule-tac x=x in ballE)
  apply (elim conjE)
  apply (rule conjI)
  apply auto
  done
with nort-repbc-dag repp-nc show
  Dag (repb p) (repb  $\times$  low) (repb  $\times$  high) t =
  Dag (repc p) (repc  $\times$  low) (repc  $\times$  high) t
  apply auto
  apply (rotate-tac 2)
  apply (frule-tac Dag-unique)
  apply (rotate-tac 1)
  apply simp
  apply simp
  apply (frule Dag-unique)
  apply (rotate-tac 3)
  apply simp
  apply simp
  done
qed
from rep-prop have repbc-changes:  $\forall no \in \text{set } (ll ! n).$ 
  repc no  $\neq$  Null  $\wedge$ 
  (if (repc  $\times$  low) no = (repc  $\times$  high) no  $\wedge$  low no  $\neq$  Null
  then repc no = (repc  $\times$  low) no
  else repc no  $\in$  set (ll ! n)  $\wedge$  repc (repc no) = repc no  $\wedge$ 
  ( $\forall no1 \in \text{set } (ll ! n).$  ((repc  $\times$  high) no1 = (repc  $\times$  high) no  $\wedge$ 
  (repc  $\times$  low) no1 = (repc  $\times$  low) no) = (repc no = repc no1)))
  by blast
from nsll lll have n-var-prop:  $n + 1 \leq \text{var } p + 1$ 
  by simp
from rep-nc have Sucn-repb-nc: ( $\forall pt. pt \notin \text{set-of pret} \vee$ 
  ( $\exists i. n + 1 \leq i \wedge pt \in \text{set } (ll ! i) \wedge i < \text{var } p + 1$ )
   $\longrightarrow \text{rep } pt = \text{repb } pt$ )

```

```

apply –
apply (intro allI impI)
apply (erule-tac x=pt in allE)
apply auto
apply (rule-tac x=i in exI)
apply auto
done
have repc-nc:
  ( $\forall pt. pt \notin \text{set-of pret} \vee$ 
  ( $\exists i. n + 1 \leq i \wedge pt \in \text{set} (ll ! i) \wedge i < \text{var } p + 1$ )
   $\longrightarrow \text{rep } pt = \text{repc } pt$ )
proof (intro allI impI)
  fix pt
  assume pt-notin-lower-ll:  $pt \notin \text{set-of pret} \vee$ 
    ( $\exists i. n + 1 \leq i \wedge pt \in \text{set} (ll ! i) \wedge i < \text{var } p + 1$ )
  show  $\text{rep } pt = \text{repc } pt$ 
  proof (cases pt  $\notin$  set-of pret)
    case True
      with wf-ll nsll have  $pt \notin \text{set} (ll ! n)$ 
        apply (simp add: wf-ll-def length-ll-eq)
        apply (case-tac pt  $\in$  set (ll ! n))
        apply (subgoal-tac pt  $\in$  set-of pret)
        apply (auto)
        done
      with repb-nc have  $\text{repb } pt = \text{repc } pt$ 
        by auto
      with Sucn-repb-nc True show ?thesis
        by auto
    next
      assume pt-in-pret:  $\neg pt \notin \text{set-of pret}$ 
      with pt-notin-lower-ll have pt-in-higher-ll:
        ( $\exists i. n + 1 \leq i \wedge pt \in \text{set} (ll ! i) \wedge i < \text{var } p + 1$ )
        by simp
      with nsll wf-ll lll have pt-notin-lln:  $pt \notin \text{set} (ll ! n)$ 
        apply –
        apply (erule exE)
        apply (rule-tac i=i and j=n in no-in-one-ll)
        apply (auto simp add: length-ll-eq)
        done
      with repb-nc have  $\text{repb } pt = \text{repc } pt$ 
        by auto
      with Sucn-repb-nc pt-in-higher-ll show ?thesis
        by auto
    qed
  qed
from wf-ll nsll
have Nodesn-notin-lln:  $\forall no \in \text{Nodes } n \text{ ll. } no \notin \text{set} (ll ! n)$ 
  apply (simp add: Nodes-def)
  apply clarify

```

```

apply (drule no-in-one-ll)
apply (auto simp add: length-ll-eq)
done
with repb-nc
have Nodesn-repnc:  $\forall no \in \text{Nodes } n \text{ ll. } \text{repb } no = \text{repc } no$ 
  apply –
  apply (rule ballI)
  apply (erule-tac x=no in allE)
  apply simp
  done
then have repbNodes-repcNodes:
  repb ‘(Nodes n ll) = repc ‘(Nodes n ll)
  apply –
  apply rule
  apply blast
  apply rule
  apply (erule imageE)
  apply (erule-tac x=xa in ballE)
  prefer 2
  apply simp
  apply rule
  apply auto
  done
have repcNodes-in-Nodes:
  repc ‘Nodes (n + 1) ll  $\subseteq$  Nodes (n + 1) ll
proof
  fix x
  assume x-in-repcNodesSucn:  $x \in \text{repc } \text{‘Nodes } (n + 1) \text{ ll}$ 
  show  $x \in \text{Nodes } (n + 1) \text{ ll}$ 
  proof (cases x ∈ repc ‘Nodes n ll)
    case True
    with repbNodes-repcNodes repbNodes-in-Nodes have  $x \in \text{Nodes } n \text{ ll}$ 
    by auto
    with Nodes-subset show ?thesis
    by auto
  next
  assume  $x \notin \text{repc } \text{‘Nodes } n \text{ ll}$ 
  with x-in-repcNodesSucn have x-in-repclln:  $x \in \text{repc } \text{‘set } (ll ! n)$ 
  apply (auto simp add: Nodes-def)
  apply (case-tac k < n)
  apply auto
  apply (case-tac k = n)
  apply simp
  apply arith
  done
from x-in-repclln show ?thesis
proof (elim imageE)
  fix y
  assume x-repcy:  $x = \text{repc } y$ 

```

```

assume y-in-repclln:  $y \in \text{set } (ll \ ! \ n)$ 
from rep-prop y-in-repclln obtain
  repcy-nNull:  $\text{repc } y \neq \text{Null}$  and
  red-prop:  $(\text{repc } \times \text{low}) \ y = (\text{repc } \times \text{high}) \ y \wedge$ 
  low  $y \neq \text{Null} \longrightarrow \text{repc } y = (\text{repc } \times \text{high}) \ y$  and
  share-prop:  $((\text{repc } \times \text{low}) \ y = (\text{repc } \times \text{high}) \ y \longrightarrow \text{low } y = \text{Null})$ 
   $\longrightarrow \text{repc } y \in \text{set } (ll \ ! \ n) \wedge \text{repc } (\text{repc } y) = \text{repc } y \wedge$ 
   $(\forall \text{no1} \in \text{set } (ll \ ! \ n).$ 
   $(\text{repc } \times \text{high}) \ \text{no1} = (\text{repc } \times \text{high}) \ y \wedge$ 
   $(\text{repc } \times \text{low}) \ \text{no1} = (\text{repc } \times \text{low}) \ y) = (\text{repc } y = \text{repc } \text{no1}))$ 
  using  $[[\text{simp-depth-limit} = 4]]$ 
  by auto
from wf-ll nsll y-in-repclln obtain
  y-in-pret:  $y \in \text{set-of-pret}$  and
  vary-n:  $\text{var } y = n$ 
  by (auto simp add: wf-ll-def length-ll-eq)
from y-in-pret pret-dag have y-nNull:  $y \neq \text{Null}$ 
  apply  $-$ 
  apply (rule set-of-nn)
  apply auto
  done
show  $x \in \text{Nodes } (n + 1) \ ll$ 
proof (cases low y = Null)
  case True
  from pret-dag prebdt-pret True y-in-pret
  have highy-Null:  $\text{high } y = \text{Null}$ 
  apply  $-$ 
  apply (drule balanced-bdt)
  apply auto
  done
with share-prop True obtain
  repcy-in-llb:  $\text{repc } y \in \text{set } (ll \ ! \ n)$  and
  rry-ry:  $\text{repc } (\text{repc } y) = \text{repc } y$  and
  y-other-node-prop:  $\forall \text{no1} \in \text{set } (ll \ ! \ n).$ 
   $((\text{repc } \times \text{high}) \ \text{no1} = (\text{repc } \times \text{high}) \ y \wedge$ 
   $(\text{repc } \times \text{low}) \ \text{no1} = (\text{repc } \times \text{low}) \ y) = (\text{repc } y = \text{repc } \text{no1})$ 
  by simp
from repcy-in-llb x-repcy show ?thesis
  by (auto simp add: Nodes-def)
next
assume lowy-nNull:  $\text{low } y \neq \text{Null}$ 
with pret-dag prebdt-pret y-in-pret
have highy-nNull:  $\text{high } y \neq \text{Null}$ 
  apply  $-$ 
  apply (drule balanced-bdt)
  apply auto
  done
show ?thesis
proof (cases (repc  $\times$  low) y = (repc  $\times$  high) y)

```

```

case True
with red-prop lowy-nNull have repc y = (repc  $\times$  high) y
  by auto
with highy-nNull have red-repc-y: repc y = repc (high y)
  by (simp add: null-comp-def)
from pret-dag ord-pret y-in-pret lowy-nNull highy-nNull

have var (low y) < var y  $\wedge$  var (high y) < var y
  apply –
  apply (rule var-ordered-children)
  apply auto
  done
with vary-n have varhighy: var (high y) < n
  by auto
from y-in-pret y-nNull highy-nNull pret-dag
have high y  $\in$  set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
  done
with wf-ll varhighy have high y  $\in$  Nodes n ll
  by (auto simp add: wf-ll-def Nodes-def)
with red-repc-y have repc y  $\in$  repc 'Nodes n ll
  by simp
with x-repcy have x  $\in$  repc 'Nodes n ll
  by simp
with repbNodes-repcNodes repbNodes-in-Nodes
have x  $\in$  Nodes n ll
  by auto
with Nodes-subset show ?thesis
  by auto
next
assume (repc  $\times$  low) y  $\neq$  (repc  $\times$  high) y
with share-prop obtain
  repcy-in-llbn: repc y  $\in$  set (ll ! n) and
  rry-ry: repc (repc y) = repc y and
  y-other-node-share:  $\forall no1 \in set (ll ! n).$ 
  ((repc  $\times$  high) no1 = (repc  $\times$  high) y  $\wedge$ 
  (repc  $\times$  low) no1 = (repc  $\times$  low) y) = (repc y = repc no1)
  by auto
with repcy-in-llbn x-repcy have x  $\in$  set (ll ! n)
  by auto
then show ?thesis
  by (auto simp add: Nodes-def)
qed
qed
qed
qed
qed

```



```

have ( $\forall no \in Nodes (n + 1) ll.$ 
   $var (repc\ no) \leq var\ no \wedge$ 
   $(\exists not\ nort.\ Dag\ (repc\ no)\ (repc\ \times\ low)\ (repc\ \times\ high)\ nort \wedge$ 
   $Dag\ no\ low\ high\ not \wedge$ 
   $reduced\ nort \wedge ordered\ nort\ var \wedge$ 
   $set-of\ nort \subseteq repc\ 'Nodes\ (n + 1)\ ll \wedge$ 
   $(\forall no \in set-of\ nort.\ repc\ no = no) \wedge$ 
   $(\exists nobdt.\ bdt\ not\ var = Some\ nobdt \wedge$ 
   $(\exists norbdt.\ bdt\ nort\ var = Some\ norbdt \wedge nobdt \sim norbdt))))$ 
(is  $\forall no \in Nodes (n + 1) ll.\ ?Q\ i\ no)$ 
proof (intro ballI)
  fix  $no$ 
  assume  $no-in-Nodes: no \in Nodes (n + 1) ll$ 
  from  $wf-ll\ no-in-Nodes\ nsll$  have  $no-in-pret: no \in set-of\ pret$ 
    apply (simp add: wf-ll-def Nodes-def length-ll-eq)
    apply (erule conjE)
    apply (thin-tac  $\forall q.\ q \in set-of\ pret \longrightarrow q \in set (ll ! var\ q)$ )
    apply (erule exE)
    apply (erule-tac  $x=x$  in allE)
    apply (erule impE)
    apply arith
    apply (erule-tac  $x=no$  in ballE)
    apply auto
    done
  from  $pret-dag\ no-in-pret$  have  $nonNull: no \neq Null$ 
    apply  $-$ 
    apply (rule set-of-nn [rule-format])
    apply auto
    done
  show  $?Q\ i\ no$ 
  proof (cases  $no \in Nodes\ n\ ll$ )
    case True
      note  $no-in-Nodesn=this$ 
      with  $wf-ll\ nsll\ no-in-Nodes$ 
      have  $no-notin-llbn: no \notin set (ll ! n)$ 
        apply  $-$ 
        apply (simp add: Nodes-def length-ll-eq)
        apply (elim exE)
        apply (drule-tac  $?i=xa$  and  $?j=n$  in  $no-in-one-ll$ )
        apply arith
        apply simp
        apply auto
        done
      with  $repcb-nc$  have  $repb-no-eq-repc-no: repb\ no = repc\ no$ 
        by simp
      from  $repcb-nc\ no-in-Nodes\ no-notin-llbn\ normalize-prop\ True$ 
      have  $varrep-eq-var: var (repc\ no) \leq var\ no$ 
        apply  $-$ 
        apply (erule-tac  $x=no$  in ballE)

```

```

    prefer 2
    apply simp
    apply (erule-tac x=no in allE)
    apply simp
    done
  moreover
  from True normalize-prop no-in-Nodes obtain not nort where
    nort-dag: Dag (repb no) (repb  $\times$  low) (repb  $\times$  high) nort and
    ord-nort: ordered nort var and
    subset-nort-not: set-of nort  $\subseteq$  repb '(Nodes n ll) and
    not-dag: Dag no low high not and
    red-nort: reduced nort and
    nort-repb: ( $\forall no \in \text{set-of nort. repb no} = no$ ) and
    bdt-prop:  $\exists nobdt norbdt. bdt \text{ not var} = \text{Some nobdt} \wedge$ 
    bdt nort var = Some norbdt  $\wedge nobdt \sim norbdt$ 
  by blast
  moreover
  from Nodesn-notin-lln repbc-nc nort-repb subset-nort-not repbN-
odes-in-Nodes
  have nort-repc:
    ( $\forall no \in \text{set-of nort. repc no} = no$ )
  apply auto
  apply (subgoal-tac no  $\in$  Nodes n ll)
  apply fastforce
  apply blast
  done
  moreover
  from nort-dag have nortnodesnN: ( $\forall no. no \in \text{set-of nort} \longrightarrow no \neq$ 
Null)
  apply –
  apply (rule allI)
  apply (rule impI)
  apply (rule set-of-nn)
  apply auto
  done
  moreover
  have Dag (repc no) (repc  $\times$  low) (repc  $\times$  high) nort
  proof –
    from no-notin-llbn repbc-nc have repbc-no: repc no = repb no
    by fastforce
  with nort-dag
  have nortrepbc-dag: Dag (repc no) (repb  $\times$  low) (repb  $\times$  high) nort
  by simp
  from wf-ll nseqll have Nodes n ll  $\subseteq$  set-of pret
  apply –
  apply (rule Nodes-levellist-subset-t)
  apply assumption+
  apply (simp add: length-ll-eq)
  done

```

```

with repbNodes-in-Nodes subset-nort-not
have subset-nort-pret: set-of nort  $\subseteq$  set-of pret
  by simp
have vxsn-in-pret:  $\forall x \in \text{set-of nort}. \text{var } x < n \wedge x \in \text{set-of pret}$ 
proof (rule ballI)
  fix x
  assume x-in-nort:  $x \in \text{set-of nort}$ 
  from x-in-nort subset-nort-not repbNodes-in-Nodes
  have x  $\in$  Nodes n ll
  by blast
with wf-ll nll have xsn: var x < n
  apply (simp add: wf-ll-def Nodes-def length-ll-eq)
  apply (erule conjE)
  apply (thin-tac  $\forall q. q \in \text{set-of pret} \longrightarrow q \in \text{set } (ll ! \text{var } q)$ )
  apply (erule exE)
  apply (erule-tac x=xa in allE)
  apply (erule impE)
  apply arith
  apply (erule-tac x=x in ballE)
  apply auto
  done
from x-in-nort subset-nort-pret have x-in-pret:  $x \in \text{set-of pret}$ 
  by blast
with xsn show var x < n  $\wedge$  x  $\in$  set-of pret by simp
qed
with pret-dag prebdt-pret nortrepcb-dag ord-pret wf-ll nll
  repcb-nc
have  $\forall x \in \text{set-of nort}. ((\text{repc} \times \text{low}) x = (\text{repb} \times \text{low}) x \wedge$ 
   $(\text{repc} \times \text{high}) x = (\text{repb} \times \text{high}) x)$ 
  apply  $-$ 
  apply (rule nort-null-comp)
  apply (auto simp add: length-ll-eq)
  done
with nort-dag
have Dag (repc no) (repc  $\times$  low) (repc  $\times$  high) nort =
  Dag (repc no) (repb  $\times$  low) (repb  $\times$  high) nort
  apply  $-$ 
  apply (rule heaps-eq-Dag-eq)
  apply simp
  done
with nortrepcb-dag show ?thesis
  by simp
qed
moreover
have set-of nort  $\subseteq$  repc '(Nodes (n + 1) ll)
proof  $-$ 
  have Nodesn-in-NodesSucn: Nodes n ll  $\subseteq$  Nodes (n + 1) ll
  by (simp add: Nodes-def set-split)
  then have repbNodesn-in-repbNodesSucn:

```

```

    repb '(Nodes n ll) ⊆ repb '(Nodes (n + 1) ll)
  by blast
from wf-ll nsll
have Nodes-n-notin-lln: ∀ no ∈ Nodes n ll. no ∉ set (ll ! n)
  apply (simp add: Nodes-def length-ll-eq)
  apply clarify
  apply (drule no-in-one-ll)
  apply auto
  done
with repbc-nc have ∀ no ∈ Nodes n ll. repb no = repc no
  apply -
  apply (rule ballI)
  apply (erule-tac x=no in allE)
  apply simp
  done
then have repbNodes-repcNodes:
  repb '(Nodes n ll) = repc '(Nodes n ll)
  apply -
  apply rule
  apply blast
  apply rule
  apply (erule imageE)
  apply (erule-tac x=xa in ballE)
  prefer 2
  apply simp
  apply rule
  apply auto
  done
from Nodesn-in-NodesSucn
have repc '(Nodes n ll) ⊆ repc '(Nodes (n + 1) ll)
  by blast
with repbNodes-repcNodes subset-nort-not repbNodesn-in-repbNodesSucn

  show ?thesis by simp
qed
ultimately show ?thesis
  by blast
next
assume no ∉ Nodes n ll
with no-in-Nodes have no-in-llbn: no ∈ set (ll ! n)
  apply (simp add: Nodes-def)
  apply (erule exE)
  apply (erule-tac x=x in allE)
  apply (case-tac x<n)
  apply simp
  apply simp
  apply (elim conjE)
  apply (case-tac x=n)
  apply simp

```

```

apply arith
done
with wf-ll nsll have varno: var no = n
  by (simp add: wf-ll-def length-ll-eq)
from repcb-changes no-in-llbn
have repcno-changes: repc no ≠ Null  $\wedge$ 
  ((repc  $\times$  low) no = (repc  $\times$  high) no  $\wedge$  low no  $\neq$  Null)
   $\longrightarrow$  repc no = (repc  $\times$  high) no  $\wedge$ 
  (((repc  $\times$  low) no = (repc  $\times$  high) no  $\longrightarrow$  low no = Null)
   $\longrightarrow$  repc no  $\in$  set (ll ! n)  $\wedge$  repc (repc no) = repc no  $\wedge$ 
  ( $\forall$  no1  $\in$  set (ll ! n). ((repc  $\times$  high) no1 = (repc  $\times$  high) no  $\wedge$ 
  (repc  $\times$  low) no1 = (repc  $\times$  low) no) = (repc no = repc no1)))
  (is ?rnonN  $\wedge$  ?repreduce  $\wedge$  ?repshare)
  using [simp-depth-limit=4]
  by (simp split: if-split)
then obtain
  rnonN: ?rnonN and
  repreduce: ?repreduce and
  repshare: ?repshare
  by blast
have repcn-normalize: var (repc no)  $\leq$  var no  $\wedge$ 
  ( $\exists$  not nort. Dag (repc no) (repc  $\times$  low) (repc  $\times$  high) nort  $\wedge$ 
  Dag no low high not  $\wedge$  reduced nort  $\wedge$  ordered nort var  $\wedge$ 
  set-of nort  $\subseteq$  repc 'Nodes (n + 1) ll  $\wedge$ 
  ( $\forall$  no  $\in$  set-of nort. repc no = no)  $\wedge$ 
  ( $\exists$  nobdt. bdt not var = Some nobdt  $\wedge$ 
  ( $\exists$  norbdt. bdt nort var = Some norbdt  $\wedge$  nobdt  $\sim$  norbdt)))
  (is ?varrep  $\wedge$  ?repcn-prop)
  is ?varrep  $\wedge$ 
  ( $\exists$  not nort. ?nort-dag nort  $\wedge$  ?not-dag not  $\wedge$  ?red nort  $\wedge$ 
  ?ord nort  $\wedge$  ?nort-in-Nodes nort  $\wedge$  ?repcno-no-n nort  $\wedge$  ?bdt-equ not
nort))
proof (cases high no = Null)
case True
note highnoNull=this
with pret-dag prebdt-pret no-in-pret
have lownoNull: low no = Null
  apply  $-$ 
  apply (drule balanced-bdt)
  apply assumption+
  apply simp
  done
with repshare have repcnoinlln: repc no  $\in$  set (ll ! n)
  by simp
with wf-ll nsll have varrno-n: var (repc no) = n
  by (simp add: wf-ll-def length-ll-eq)
with varno have varrep: ?varrep
  by simp
from wf-ll nsll no-in-llbn varrno-n

```

```

have varrno-varno: var (repc no) = var no
  by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsl repcnoinlln
have rno-in-pret: repc no ∈ set-of pret
  by (simp add: wf-ll-def length-ll-eq)
from repcnoinlln repshare lownoNull
have reprep-eq-rep: repc (repc no) = repc no
  by simp
with repcnoinlln repshare lownoNull
have repchildreneq:
  ((repc ∝ high) (repc no) = (repc ∝ high) no ∧
   (repc ∝ low) (repc no) = (repc ∝ low) no)
  by simp
have repcn-prop: ?repcn-prop
  apply –
  apply (rule-tac x=(Node Tip no Tip) in exI)
  apply (rule-tac x=(Node Tip (repc no) Tip) in exI)
  apply (intro conjI)
  apply simp
  prefer 3
  apply simp
  prefer 3
  apply simp
proof –
  from pret-dag pnN rno-in-pret have rnonN: repc no ≠ Null
    apply (case-tac repc no = Null)
    apply auto
    done
  from highnoNull repchildreneq
  have rhighNull: (repc ∝ high) (repc no) = Null
    by (simp add: null-comp-def)
  from lownoNull repchildreneq
  have rlowNull: (repc ∝ low) (repc no) = Null
    by (simp add: null-comp-def)
  with rhighNull rnonN
  show repc no ≠ Null ∧ (repc ∝ low) (repc no) = Null ∧
    (repc ∝ high) (repc no) = Null
    by simp
next
  from nonNull lownoNull highnoNull
  show ?not-dag (Node Tip no Tip)
    by simp
next
  from no-in-Nodes
  show set-of (Node Tip (repc no) Tip) ⊆ repc ‘ Nodes (n + 1) ll
    by simp
next
  show  $\forall no \in \text{set-of } (Node\ Tip\ (repc\ no)\ Tip). repc\ no = no$ 
proof

```

```

    fix pt
    assume pt-in-repcLeaf: pt ∈ set-of (Node Tip (repc no) Tip)
    with reprep-eq-rep show repc pt = pt
      by simp
  qed
next
show ?bdt-equ (Node Tip no Tip) (Node Tip (repc no) Tip)
proof (cases var no = 0)
  case True
  note vno-Null=this
  then have nobdt: bdt (Node Tip no Tip) var = Some Zero by simp
  from varrep vno-Null have varrho: var (repc no) = 0 by simp
  then have norbdt: bdt (Node Tip (repc no) Tip) var = Some Zero
by simp
  from nobdt norbdt vno-Null varrho show ?thesis
  by (simp add: cong-eval-def)
next
assume vno-not-Null: var no ≠ 0
show ?thesis
proof (cases var no = 1)
  case True
  note vno-One=this
  then have nobdt: bdt (Node Tip no Tip) var = Some One by simp
  from varrho-varno vno-One
  have bdt (Node Tip (repc no) Tip) var = Some One by simp
  with nobdt show ?thesis by (auto simp add: cong-eval-def)
next
assume vno-nOne: var no ≠ 1
with vno-not-Null have onesvno: 1 < var no by simp
from nonNull lownoNull highnoNull
have no-dag: Dag no low high (Node Tip no Tip)
  by simp
with pret-dag no-in-pret have not-in-pret: (Node Tip no Tip) ≤
pret
  by (metis set-of-subdag)
with prebdt-pret have ∃ bdt2. bdt (Node Tip no Tip) var = Some
bdt2
  by (metis subbdt-ex)
with onesvno show ?thesis
  by simp
qed
qed
qed
with varrep reprep-eq-rep show ?thesis by simp
next
assume hno-nNull: high no ≠ Null
with pret-dag prebdt-pret no-in-pret have lno-nNull: low no ≠ Null
  by (metis balanced-bdt)

```

```

from no-in-pret nonNull hno-nNull pret-dag
have hno-in-pret: high no ∈ set-of pret
  by (metis subelem-set-of-high)
with wf-ll
have hno-in-ll: high no ∈ set (ll ! (var (high no)))
  by (simp add: wf-ll-def)
from pret-dag ord-pret no-in-pret lno-nNull hno-nNull

have varhnos-varno: var (high no) < var no
  by (metis var-ordered-children)
with varno have varhnos-n: var (high no) < n by simp
with hno-in-ll have hno-in-Nodesn: high no ∈ Nodes n ll
  apply (simp add: Nodes-def)
  apply (rule-tac x=var (high no) in exI)
  apply simp
  done
from wf-ll nsll hno-in-ll varhnos-n
have high no ∉ set (ll ! n)
  apply –
  apply (rule no-in-one-ll)
  apply (auto simp add: length-ll-eq)
  done
with repcb-nc have repb-repc-high: repb (high no) = repc (high no) by

```

simp

```

with normalize-prop hno-in-Nodesn varhnos-varno varno
have high-normalize: var (repc (high no)) ≤ var (high no) ∧
  (∃ not nort. Dag (repc (high no)) (repb ∝ low) (repb ∝ high) nort ∧
  Dag (high no) low high not ∧ reduced nort ∧
  ordered nort var ∧ set-of nort ⊆ repb '(Nodes n ll) ∧
  (∀ no ∈ set-of nort. repb no = no) ∧
  (∃ nobdt norbdt. bdt not var = Some nobdt ∧ bdt nort var =
  Some norbdt ∧ nobdt ~ norbdt))
  is ?varrep-high ∧
  (∃ not nort. ?repbchigh-dag nort ∧ ?high-dag not ∧
  ?redhigh nort ∧ ?ordhigh nort ∧ ?rephigh-in-Nodes nort ∧
  ?repbno-no nort ∧ ?highdd-prop not nort)
  is ?varrep-high ∧ ?not-nort-prop)
apply simp
apply (erule-tac x=high no in ballE)
apply (simp del: Dag-Ref)
apply simp
done
then have varrep-high: ?varrep-high by simp
from varhnos-n varrep-high have varrephno-s-n:
  var (repc (high no)) < n
  by simp
from Nodes-subset
have Nodes n ll ⊆ Nodes (Suc n) ll

```



```

  by auto
with hno-in-Nodesn repcNodes-in-Nodes
have repc (high no) ∈ Nodes (Suc n) ll
  apply simp
  apply blast
  done
with wf-ll nsll have repc (high no) ∈ set-of pret
  apply (simp add: wf-ll-def Nodes-def length-ll-eq)
  apply (elim conjE exE)
  apply (thin-tac ∀ q. q ∈ set-of pret → q ∈ set (ll ! var q))
  apply (erule-tac x=x in allE)
  apply (erule impE)
  apply simp
  apply (erule-tac x=repc (high no) in ballE)
  apply auto
  done
with wf-ll varrephno-s-n
have ∃ k < n. repc (high no) ∈ set (ll ! k)
  by (auto simp add: wf-ll-def)
with wf-ll nsll have repc (high no) ∉ set (ll ! n)
  apply -
  apply (erule exE)
  apply (rule-tac i=k and j=n in no-in-one-ll)
  apply (auto simp add: length-ll-eq)
  done
with repbc-nc
have repbhigh-idem: repb (repc (high no)) = repc (repc (high no))
  by auto
from high-normalize
have not-nort-prop-high: ?not-nort-prop by (simp del: Dag-Ref)
from not-nort-prop-high obtain hnot where high-dag: ?high-dag hnot
  by auto
from wf-ll nsll
have ∀ no ∈ Nodes n ll. no ∉ set (ll ! n)
  apply (simp add: Nodes-def length-ll-eq)
  apply clarify
  apply (drule no-in-one-ll)
  apply auto
  done
with repbc-nc have ∀ no ∈ Nodes n ll. repb no = repc no
  apply -
  apply (rule ballI)
  apply (erule-tac x=no in allE)
  apply simp
  done
then
have repbNodes-repcNodes:
  repb '(Nodes n ll) = repc '(Nodes n ll)
  apply -

```

```

apply rule
apply blast
apply rule
apply (erule imageE)
apply (erule-tac x=xa in ballE)
prefer 2
apply simp
apply rule
apply auto
done
then have repcNodes-repbNodes:
  repc '(Nodes n ll) = repb '(Nodes n ll)
  by simp
from pret-dag nll wf-ll
have null-notin-Nodesn: Null  $\notin$  Nodes n ll
  apply –
  apply (rule Null-notin-Nodes)
  apply (auto simp add: length-ll-eq)
  done
from hno-in-Nodesn have repc (high no)  $\in$  repc '(Nodes n ll)
  by blast
with repbNodes-in-Nodes repcNodes-repbNodes
have repc (high no)  $\in$  Nodes n ll
  apply simp
  apply blast
  done
with null-notin-Nodesn have rhn-nNull: repc (high no)  $\neq$  Null
  by auto

```

```

from no-in-pret nonNull lno-nNull pret-dag
have lno-in-pret: low no  $\in$  set-of pret
  by (rule subelem-set-of-low)
with wf-ll
have lno-in-ll: low no  $\in$  set (ll ! (var (low no)))
  by (simp add: wf-ll-def)
from pret-dag ord-pret no-in-pret lno-nNull hno-nNull
have varlnos-varno: var (low no) < var no
  apply –
  apply (drule var-ordered-children)
  apply assumption+
  apply auto
  done
with varno have varlnos-n: var (low no) < n by simp
with lno-in-ll have lno-in-Nodesn: low no  $\in$  Nodes n ll
  apply (simp add: Nodes-def)
  apply (rule-tac x=var (low no) in exI)
  apply simp

```

```

done
from varlnos-n wf-ll nsl lno-in-ll
have low no  $\notin$  set (ll ! n)
  apply -
  apply (rule no-in-one-ll)
  apply (auto simp add: length-ll-eq)
done
with repbc-nc have repb-repc-low: repb (low no) = repc (low no) by
simp
with normalize-prop lno-in-Nodesn varlnos-varno varno
have low-normalize: var (repc (low no))  $\leq$  var (low no)  $\wedge$ 
  ( $\exists$  not nort. Dag (repc (low no)) (repb  $\propto$  low) (repb  $\propto$  high) nort  $\wedge$ 
  Dag (low no) low high not  $\wedge$  reduced nort  $\wedge$  ordered nort var  $\wedge$ 
  set-of nort  $\subseteq$  repb '(Nodes n ll)  $\wedge$ 
  ( $\forall$  no $\in$ set-of nort. repb no = no)  $\wedge$ 
  ( $\exists$  nobdt norbdt. bdt not var = Some nobdt  $\wedge$  bdt nort var = Some
norbdt  $\wedge$ 
  nobdt  $\sim$  norbdt))
(is ?varrep-low  $\wedge$ 
  ( $\exists$  not nort. ?repbc-low-dag nort  $\wedge$  ?low-dag not  $\wedge$  ?redhigh nort  $\wedge$ 
  ?ordhigh nort  $\wedge$  ?replow-in-Nodes nort  $\wedge$  ?low-repno-no nort
 $\wedge$  ?lowdd-prop not nort)
  is ?varrep-low  $\wedge$  ?not-nort-prop-low)
apply simp
apply (erule-tac x=low no in ballE)
apply (simp del: Dag-Ref)
apply simp
done
then have varrep-low: ?varrep-low by simp
from low-normalize have not-nort-prop-low: ?not-nort-prop-low
  by (simp del: Dag-Ref)
from lno-in-Nodesn have repc (low no)  $\in$  repc '(Nodes n ll)
  by blast
with repbNodes-in-Nodes repcNodes-repbNodes
have repc (low no)  $\in$  Nodes n ll
  apply simp
  apply blast
done
with null-notin-Nodesn have rln-nNull: repc (low no)  $\neq$  Null
  by auto

show ?thesis
proof (cases repc (low no) = repc (high no))
case True
note red-case=this
with reproduce lno-nNull hno-nNull
have rno-eq-hrno: repc no = repc (high no)
  by (simp add: null-comp-def)

```

simp

```

from varhnos-varno rno-eq-hrno varrep-high have varrep: ?varrep by
from not-nort-prop-high not-nort-prop-low have repcn-prop: ?repcn-prop
  apply -
  apply (elim exE)
  apply (rename-tac rnot lnot rnort lnort )
  apply (rule-tac x=(Node lnot no rnot) in exI)
  apply (rule-tac x=rnort in exI)
  apply (elim conjE)
  apply (intro conjI)
  prefer 7
  apply (elim exE)
apply (rename-tac rnot lnot rnort lnort rnobdt lnobdt rnorbdtd lnorbdt)
  apply (elim conjE)
  apply (case-tac Suc 0 < var no)
  apply (rule-tac x=(Bdt-Node lnobdt (var no) rnobdt) in exI)
  apply (rule conjI)
  prefer 2
  apply (rule-tac x=rnorbdtd in exI)
  apply (rule conjI)
proof -
  fix rnot lnot rnort lnort
  assume highnort-dag:
    Dag (repc (high no)) (repb  $\times$  low) (repb  $\times$  high) rnort
  assume ord-nort: ordered rnort var
  assume rnort-in-repNodes: set-of rnort  $\subseteq$  repb ‘ Nodes n ll
  from rnort-in-repNodes repbNodes-in-Nodes
  have nort-in-Nodes: set-of rnort  $\subseteq$  Nodes n ll
  by blast
  from varhnos-n varrep-high
  have vrhnos-n: var (repc (high no)) < n by simp
  from rhn-nNull highnort-dag
  have  $\exists$  lno rno. rnort = Node lno (repc (high no)) rno by fastforce
  with highnort-dag rhn-nNull have root rnort = repc (high no) by
 $\text{auto}$ 
  with ord-nort have  $\forall x \in \text{set-of rnort. var } x \leq \text{var (repc (high no))}$ 
  apply -
  apply (rule ballI)
  apply (drule ordered-set-of)
  apply auto
  done
with vrhnos-n have vxsn:  $\forall x \in \text{set-of rnort. var } x < n$ 
  by fastforce
from nort-in-Nodes have  $\forall x \in \text{set-of rnort. } x \in \text{Nodes } n \text{ ll}$ 
  by auto
with wf-ll nsll
have x-in-pret:  $\forall x \in \text{set-of rnort. } x \in \text{set-of pret}$ 
  apply -

```

```

    apply (rule ballI)
    apply (drule wf-ll-Nodes-pret)
    apply (auto simp add: length-ll-eq)
    done
  from vxsn x-in-pret
  have vxsn-in-nort:  $\forall x \in \text{set-of rnort}. \text{var } x < n \wedge x \in \text{set-of pret}$ 
    by auto
  with pret-dag prebdt-pret highnort-dag ord-pret wf-ll nsll
    repbc-nc
  have  $\forall x \in \text{set-of rnort}. (\text{repc } \times \text{low}) x = (\text{repb } \times \text{low}) x \wedge$ 
     $(\text{repc } \times \text{high}) x = (\text{repb } \times \text{high}) x$ 
    apply -
    apply (rule nort-null-comp)
    apply (auto simp add: length-ll-eq)
    done
  with rno-eq-hrno
  have Dag (repc no) (repc  $\times$  low) (repc  $\times$  high) rnort =
    Dag (repb no) (repb  $\times$  low) (repb  $\times$  high) rnort
    apply -
    apply (rule heaps-eq-Dag-eq)
    apply simp
    done
  with highnort-dag rno-eq-hrno
  show Dag (repc no) (repc  $\times$  low) (repc  $\times$  high) rnort by simp
next
  fix rnot lnot rnort lnot
  assume lnot-dag: Dag (low no) low high lnot
  assume rnot-dag: Dag (high no) low high rnot
  with lnot-dag nonNull
  show Dag no low high (Node lnot no rnot) by simp
next
  fix rnot lnot rnort lnot
  assume reduced rnort
  then show reduced rnort by simp
next
  fix rnort
  assume ordered rnort var
  then show ordered rnort var by simp
next
  fix rnort
  assume rnort-in-Nodes:  $\text{set-of rnort} \subseteq \text{repb } \text{' Nodes } n \text{ ll}$ 
  have Nodes n ll  $\subseteq$  Nodes (n + 1) ll
    by (simp add: Nodes-def set-split)
  then have  $\text{repc } \text{' Nodes } n \text{ ll} \subseteq \text{repc } \text{' Nodes } (n + 1) \text{ ll}$ 
    by blast
  with rnort-in-Nodes repbNodes-repcNodes
  show  $\text{set-of rnort} \subseteq \text{repc } \text{' Nodes } (n + 1) \text{ ll}$ 
    by (simp add: Nodes-def)
next

```

```

fix rnort rnorbd
assume bdt rnort var = Some rnorbd
then show bdt rnort var = Some rnorbd by simp
next
fix rnot lnot rnort lnort rnobdt knobdt rnorbd lnorbdt
assume rcongeval: rnobdt ~ rnorbd
assume lnort-dag:
  Dag (repc (low no)) (repb  $\times$  low) (repb  $\times$  high) lnort
assume rnort-dag:
  Dag (repc (high no)) (repb  $\times$  low) (repb  $\times$  high) rnort
assume lnorbdt-def: bdt lnot var = Some lnorbdt
assume rnorbdt-def: bdt rnort var = Some rnorbd
assume lcongeval:knobdt ~ lnorbdt
from red-case lnort-dag rnort-dag
have lnort-rnort: lnort = rnort
  by (simp add: Dag-unique del: Dag-Ref)
with lnorbdt-def lcongeval rnorbd-def
have knobdt-rnorbd: knobdt ~ rnorbd by simp
with rcongeval have knobdt ~ rnobdt
  apply –
  apply (rule cong-eval-trans)
  apply (auto simp add: cong-eval-sym)
  done
then have Bdt-Node knobdt (var no) rnobdt ~ rnorbd
  apply –
  apply (simp add: cong-eval-sym [rule-format])
  apply (rule cong-eval-child-high)
  apply assumption
  done
with rcongeval show Bdt-Node knobdt (var no) rnobdt ~ rnorbd
  apply –
  apply (rotate-tac 1)
  apply (rule cong-eval-trans)
  apply auto
  done
next
fix lnot rnot knobdt rnobdt
assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume knobdt-def: bdt lnot var = Some knobdt
assume rnobdt-def: bdt rnot var = Some rnobdt
assume onesvarno: Suc 0 < var no
with rnobdt-def lnot-dag rnot-dag knobdt-def
show bdt (Node lnot no rnot) var =
  Some (Bdt-Node knobdt (var no) rnobdt)
  by simp
next
fix rnot lnot rnort lnort rnobdt knobdt rnorbd lnorbdt
assume lnorbdt-def: bdt lnot var = Some lnorbdt

```

```

assume rnobdt-def: bdt rnot var = Some rnobdt
assume rnorbdt-def: bdt rnort var = Some rnorbdt
assume cong-rno-rnor: rnobdt ~ rnorbdt
assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume  $\neg \text{Suc } 0 < \text{var no}$ 
then have varnoseq1: var no = 0  $\vee$  var no = 1 by auto
show  $\exists \text{ nobdt. bdt (Node lnot no rnot) var = Some nobdt} \wedge$ 
 $(\exists \text{ norbdt. bdt rnort var = Some norbdt} \wedge \text{ nobdt} \sim \text{ norbdt})$ 
proof (cases var no = 0)
  case True
    note vnoNull=this
    with pret-dag ord-pret no-in-pret lno-nNull hno-nNull
    show ?thesis
      apply  $-$ 
      apply (drule var-ordered-children)
      apply auto
      done
  next
    assume var no  $\neq$  0
    with varnoseq1 have vnoOne: var no = 1 by simp
    from pret-dag ord-pret no-in-pret lno-nNull hno-nNull
      vnoOne
    have vlvrNull: var (low no) = 0  $\wedge$  var (high no) = 0
      apply  $-$ 
      apply (drule var-ordered-children)
      apply auto
      done
    then have vlNull: var (low no) = 0 by simp
    from vlvrNull have vrNull: var (high no) = 0 by simp
    from lnobdt-def lnot-dag vlNull lno-nNull
    have lnobdt-Zero: lnobdt = Zero
      apply  $-$ 
      apply (drule bdt-Some-var0-Zero)
      apply auto
      done
    from rnobdt-def rnot-dag vrNull hno-nNull
    have rnobdt-Zero: rnobdt = Zero
      apply  $-$ 
      apply (drule bdt-Some-var0-Zero)
      apply auto
      done
    from lnobdt-Zero lnobdt-def have bdt lnot var = Some Zero by
simp
    with lnot-dag vlNull
    have lnot-Node: lnot = (Node Tip (low no) Tip)
      by auto
    from rnobdt-Zero rnobdt-def rnot-dag vrNull
    have rnot-Node: rnot = (Node Tip (high no) Tip)

```

```

    by auto
  from pret-dag no-in-pret obtain not where
    not-ex: Dag no low high not
  apply -
  apply (drule dag-setof-exD)
  apply auto
  done
with pret-dag no-in-pret have not-ex-in-pret: not <= pret
  apply -
  apply (rule set-of-subdag)
  apply auto
  done
from not-ex lnot-dag rnot-dag nonNull
have not-def: not = (Node lnot no rnot)
  by (simp add: Dag-unique del: Dag-Ref)
with not-ex-in-pret prebdt-pret
have nobdt-ex:  $\exists$  nobdt. bdt (Node lnot no rnot) var = Some nobdt
  apply -
  apply (rule subbdt-ex)
  apply auto
  done
then obtain nobdt where
  nobdt-def: bdt (Node lnot no rnot) var = Some nobdt by auto
from not-def have root not = no by simp
with nobdt-def vnoOne not-def have not = (Node Tip no Tip)
  apply -
  apply (drule bdt-Some-var1-One)
  apply auto
  done
with not-def have rnot = Tip by simp
with rnot-Node show ?thesis by simp
qed
next
fix rnot lnot rnort lnot
assume rnort-in-repb-Nodesn: set-of rnort  $\subseteq$  repb 'Nodes n ll
assume rnort-repb-no:  $\forall$  no $\in$ set-of rnort. repb no = no
from repbNodes-in-Nodes rnort-in-repb-Nodesn
have rnort-in-Nodesn: set-of rnort  $\subseteq$  Nodes n ll
  by blast
show  $\forall$  no $\in$ set-of rnort. repc no = no
proof
  fix pt
  assume pt-in-rnort: pt  $\in$  set-of rnort
  with rnort-in-Nodesn have pt  $\in$  Nodes n ll
  by blast
  with Nodesn-notin-lln have pt  $\notin$  set (ll ! n)
  by auto
  with repbc-nc have repb pt = repc pt
  by auto

```



```

    with rnort-repb-no pt-in-rnort show repc pt = pt
      by auto
    qed
  qed
  with varrep show ?thesis by simp
next
assume share-case-cond: repc (low no) ≠ repc (high no)
with lno-nNull hno-nNull
have share-case-cond-propto: (repc × low) no ≠ (repc × high) no
  by (simp add: null-comp-def)
with repshare obtain
  rno-in-llbn: repc no ∈ set (ll ! n) and
  rrno-eq-rno: repc (repc no) = repc no and
  twonodes-in-llbn-prop:  $(\forall no1 \in \text{set } (ll ! n)).$ 
   $((repc \times high) no1 = (repc \times high) no \wedge$ 
   $(repc \times low) no1 = (repc \times low) no) = (repc no = repc no1)$ 
  by auto
from wf-ll rno-in-llbn nsll
have varrepno-n: var (repc no) = n
  by (simp add: wf-ll-def length-ll-eq)
with varno have varrep: ?varrep
  by simp
from not-nort-prop-high not-nort-prop-low have repcn-prop: ?repcn-prop
  apply-
  apply (elim exE)
  apply (rename-tac rnot lnot rnort lnort)
  apply (rule-tac x=Node lnot no rnot in exI)
  apply (rule-tac x=Node lnort (repc no) rnort in exI)
  apply (elim conjE)
  apply (intro conjI)
  prefer 7
  apply (elim exE)
  apply (rename-tac rnot lnot rnort lnort rnobdt lnobdt rnorbdtd lnorbdt)
  apply (elim conjE)
  apply (case-tac Suc 0 < var no)
  apply (rule-tac x=(Bdt-Node lnobdt (var no) rnobdt) in exI)
  apply (rule conjI)
  prefer 2
  apply (rule-tac x=(Bdt-Node lnorbdt (var (repc no)) rnorbdtd) in
exI)

  apply (rule conjI)
proof -
  fix rnot lnot rnort lnort
  assume rnort-dag:
    Dag (repc (high no)) (repc × low) (repc × high) rnort
  assume lnort-dag:
    Dag (repc (low no)) (repc × low) (repc × high) lnort
  assume rnort-in-repNodes: set-of rnort ⊆ repb ‘ Nodes n ll
  assume lnort-in-repNodes: set-of lnort ⊆ repb ‘ Nodes n ll

```

```

from rnort-in-repNodes repbNodes-in-Nodes
have rnort-in-Nodes: set-of rnort  $\subseteq$  Nodes n ll
  by simp
from lnort-in-repNodes repbNodes-in-Nodes
have lnort-in-Nodes: set-of lnort  $\subseteq$  Nodes n ll
  by simp
from rnort-in-Nodes
have rnortx-in-Nodes:  $\forall x \in \text{set-of } rnort. x \in \text{Nodes } n ll$ 
  by blast
with wf-ll nsll
have  $\forall x \in \text{set-of } rnort. x \in \text{set-of } pret \wedge \text{var } x < n$ 
  apply –
  apply (rule ballI)
  apply (rule wf-ll-Nodes-pret)
  apply (auto simp add: length-ll-eq)
  done
with pret-dag prebdt-pret rnort-dag ord-pret wf-ll nsll
  repbc-nc
have  $\forall x \in \text{set-of } rnort. (rep_c \times low) x = (rep_b \times low) x \wedge$ 
   $(rep_c \times high) x = (rep_b \times high) x$ 
  apply –
  apply (rule nort-null-comp)
  apply (auto simp add: length-ll-eq)
  done
then have Dag (rep_c (high no)) (rep_c  $\times$  low) (rep_c  $\times$  high) rnort =
  Dag (rep_c (high no)) (rep_b  $\times$  low) (rep_b  $\times$  high) rnort
  apply –
  apply (rule heaps-eq-Dag-eq)
  apply assumption
  done
with rnort-dag
have rnort-dag-repc:
  Dag (rep_c (high no)) (rep_c  $\times$  low) (rep_c  $\times$  high) rnort
  by simp
from lnort-in-Nodes
have lnortx-in-Nodes:  $\forall x \in \text{set-of } lnort. x \in \text{Nodes } n ll$ 
  by blast
with wf-ll nsll
have  $\forall x \in \text{set-of } lnort. x \in \text{set-of } pret \wedge \text{var } x < n$ 
  apply –
  apply (rule ballI)
  apply (rule wf-ll-Nodes-pret)
  apply (auto simp add: length-ll-eq)
  done
with pret-dag prebdt-pret lnort-dag ord-pret wf-ll nsll
  repbc-nc
have  $\forall x \in \text{set-of } lnort. (rep_c \times low) x = (rep_b \times low) x \wedge$ 
   $(rep_c \times high) x = (rep_b \times high) x$ 
  apply –

```

```

    apply (rule nort-null-comp)
    apply (auto simp add: length-ll-eq)
  done
then have
  Dag (repc (low no)) (repc  $\times$  low) (repc  $\times$  high) lnort =
  Dag (repc (low no)) (repb  $\times$  low) (repb  $\times$  high) lnort
  apply -
  apply (rule heaps-eq-Dag-eq)
  apply assumption
  done
with lnort-dag
have lnort-dag-repc:
  Dag (repc (low no)) (repc  $\times$  low) (repc  $\times$  high) lnort
  by simp
from lno-nNull hno-nNull
have propto-comp: (repc  $\times$  low) no = repc (low no)  $\wedge$ 
  (repc  $\times$  high) no = repc (high no)
  by (simp add: null-comp-def)
from rno-in-llbn twonodes-in-llbn-prop rrno-eq-rno
have (repc  $\times$  high) (repc no) = (repc  $\times$  high) no  $\wedge$ 
  (repc  $\times$  low) (repc no) = (repc  $\times$  low) no
  by simp
with propto-comp lnort-dag-repc rnort-dag-repc lno-nNull hno-nNull

  rnonN
  show Dag(repc no)(repc  $\times$  low)(repc  $\times$  high)(Node lnort (repc no)
rnot)

  by auto
next
fix rnot lnot rnot lnort
assume rnot-dag: Dag (high no) low high rnot
assume lnot-dag: Dag (low no) low high lnot
with rnot-dag nonNull
show Dag no low high (Node lnot no rnot)
  by simp
next
fix rnort lnort
assume rnort-dag:
  Dag (repc (high no)) (repb  $\times$  low) (repb  $\times$  high) rnort
assume lnort-dag:
  Dag (repc (low no)) (repb  $\times$  low) (repb  $\times$  high) lnort
assume red-rnort: reduced rnort
assume red-lnort: reduced lnort
from rhn-nNull rnort-dag obtain lnort rrnort where
  rnort-Node: rnort = (Node lnort (repc (high no)) rrnort)
  by auto
from rln-nNull lnort-dag obtain llnort rlnort where
  lnort-Node: lnort = (Node llnort (repc (low no)) rlnort)
  by auto

```

lno-nNull

```

from twonodes-in-llbn-prop rrno-eq-rno rno-in-llbn hno-nNull
have  $((\text{repc} \times \text{high}) (\text{repc no})) = \text{repc} (\text{high no}) \wedge$ 
 $((\text{repc} \times \text{low}) (\text{repc no})) = \text{repc} (\text{low no})$ 
apply –
apply (erule-tac x=repc no in ballE)
apply (auto simp add: null-comp-def)
done
with share-case-cond
have  $((\text{repc} \times \text{high}) (\text{repc no})) \neq ((\text{repc} \times \text{low}) (\text{repc no}))$ 
by auto
with red-lnort red-rnort rnort-Node lnort-Node share-case-cond
show reduced (Node lnort (repc no) rnort)
apply –
apply (rule-tac lp=repc (low no) and rp=repc (high no) and
 $llt=llnort$  and  $rlt = rlnort$  and  $lrt=lrnort$  and  $rrt=rrnort$ 
in reduced-children-parent)
apply auto
done
next
fix lnort rnort
assume lnort-dag:
 $Dag (\text{repc} (\text{low no})) (\text{repc} \times \text{low}) (\text{repc} \times \text{high}) \text{lnort}$ 
assume ord-lnort: ordered lnort var
assume rnort-dag:
 $Dag (\text{repc} (\text{high no})) (\text{repc} \times \text{low}) (\text{repc} \times \text{high}) \text{rnort}$ 
assume ord-rnort: ordered rnort var
assume lnort-in-repNodes: set-of lnort  $\subseteq$  repb ‘Nodes n ll
assume rnort-in-repNodes: set-of rnort  $\subseteq$  repb ‘Nodes n ll
from lnort-in-repNodes repbNodes-in-Nodes
have lnort-in-Nodes: set-of lnort  $\subseteq$  Nodes n ll
by simp
from rnort-in-repNodes repbNodes-in-Nodes
have rnort-in-Nodes: set-of rnort  $\subseteq$  Nodes n ll
by simp

from rhn-nNull rnort-dag obtain lnort rrnort where
 $rnort\text{-Node: } rnort = (\text{Node } lrnort (\text{repc} (\text{high no})) rrnort)$ 
by auto
from rln-nNull lnort-dag obtain llnort rlnort where
 $lnort\text{-Node: } lnort = (\text{Node } llnort (\text{repc} (\text{low no})) rlnort)$ 
by auto
from lnort-dag rln-nNull lnort-in-Nodes
have  $\text{repc} (\text{low no}) \in \text{set-of } lnort$ 
by auto
with lnort-in-Nodes
have  $\text{repc} (\text{low no}) \in \text{Nodes } n \ ll$ 
by blast
with wf-ll nsll

```

```

have vrlno-sn: var (repc (low no)) < n
  apply –
  apply (drule wf-ll-Nodes-pret)
  apply (auto simp add: length-ll-eq)
  done
from rnort-dag rhn-nNull rnort-in-Nodes
have repc (high no) ∈ set-of rnort
  by auto
with rnort-in-Nodes
have repc (high no) ∈ Nodes n ll
  by blast
with wf-ll nsll have vrhno-sn: var (repc (high no)) < n
  apply –
  apply (drule wf-ll-Nodes-pret)
  apply (auto simp add: length-ll-eq)
  done
with varrepno-n vrlno-sn lnort-dag ord-lnort rnort-dag rnort-Node
  lnort-Node ord-rnort
show ordered (Node lnort (repc no) rnort) var
  by auto
next
fix lnort rnort
assume lnort-in-Nodes: set-of lnort ⊆ repb ‘Nodes n ll
assume rnort-in-Nodes: set-of rnort ⊆ repb ‘Nodes n ll
from lnort-in-Nodes repbNodes-repcNodes
have lnort-in-repcNodes: set-of lnort ⊆ repc ‘Nodes n ll
  by simp
from rnort-in-Nodes repbNodes-repcNodes
have rnort-in-repcNodes: set-of rnort ⊆ repc ‘Nodes n ll
  by simp
have nNodessubset: Nodes n ll ⊆ Nodes (n+1) ll
  by (simp add: Nodes-subset)
then have repc-Nodes-subset:
  repc ‘Nodes n ll ⊆ repc ‘Nodes (n+1) ll
  by blast
from no-in-Nodes have repc no ∈ repc ‘Nodes (n+1) ll
  by blast
with repc-Nodes-subset lnort-in-repcNodes rnort-in-repcNodes
show set-of (Node lnort (repc no) rnort) ⊆
  repc ‘Nodes (n + 1) ll
  apply simp
  apply blast
  done
next
fix rnot lnot rnort lnort rnobdt lnobdt rnorbdt lnorbdt
assume lnobdt-def: bdt lnot var = Some lnobdt
assume rnobdt-def: bdt rnot var = Some rnobdt
assume rnorbdt-def: bdt rnort var = Some rnorbdt
assume cong-rno-rnor: rnobdt ∼ rnorbdt

```

```

assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume  $\neg \text{Suc } 0 < \text{var } no$ 
then have varnoseq1: var no = 0  $\vee$  var no = 1 by auto
show  $\exists nobdt. bdt$  (Node lnot no rnot) var = Some nobdt  $\wedge$ 
  ( $\exists norbdt. bdt$  (Node lnort (repc no) rnort) var = Some norbdt  $\wedge$ 
nobdt  $\sim$  norbdt)
proof (cases var no = 0)
  case True
    note vnoNull=this
    with pret-dag ord-pret no-in-pret lno-nNull hno-nNull
    show ?thesis
      apply  $-$ 
      apply (drule var-ordered-children)
      apply auto
      done
  next
    assume var no  $\neq$  0
    with varnoseq1 have vnoOne: var no = 1 by simp
    from pret-dag ord-pret no-in-pret lno-nNull hno-nNull
      vnoOne
    have vlvrNull: var (low no) = 0  $\wedge$  var (high no) = 0
      apply  $-$ 
      apply (drule var-ordered-children)
      apply auto
      done
    then have vlNull: var (low no) = 0 by simp
    from vlvrNull have vrNull: var (high no) = 0 by simp
    from lnobdt-def lnot-dag vlNull lno-nNull
    have lnobdt-Zero: lnobdt = Zero
      apply  $-$ 
      apply (drule bdt-Some-var0-Zero)
      apply auto
      done
    from rnobdt-def rnot-dag vrNull hno-nNull
    have rnobdt-Zero: rnobdt = Zero
      apply  $-$ 
      apply (drule bdt-Some-var0-Zero)
      apply auto
      done
    from lnobdt-Zero lno-nNull
    have lnot-Node: lnot = (Node Tip (low no) Tip)
      by auto
    from rnobdt-Zero rnobdt-def rnot-dag vrNull
    have rnot-Node: rnot = (Node Tip (high no) Tip)
      by auto
    from pret-dag no-in-pret obtain not

```

```

    where not-ex: Dag no low high not
    apply -
    apply (drule dag-setof-exD)
    apply auto
    done
  with pret-dag no-in-pret have not-ex-in-pret: not <= pret
    apply -
    apply (rule set-of-subdag)
    apply auto
    done
  from not-ex lnot-dag rnot-dag nonNull
  have not-def: not = (Node lnot no rnot)
    by (simp add: Dag-unique del: Dag-Ref)
  with not-ex-in-pret prebdt-pret
  have nobdt-ex:  $\exists$  nobdt. bdt (Node lnot no rnot) var = Some nobdt
    apply -
    apply (rule subbdt-ex)
    apply auto
    done
  then obtain nobdt where
    nobdt-def: bdt (Node lnot no rnot) var = Some nobdt by auto
  from not-def have root not = no by simp
  with nobdt-def vnoOne not-def
  have not = (Node Tip no Tip)
    apply -
    apply (drule bdt-Some-var1-One)
    apply auto
    done
  with not-def have rnot = Tip by simp
  with rnot-Node show ?thesis by simp
qed
next
fix lnot rnot lnobdt rnobdt
assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume lnobdt-def: bdt lnot var = Some lnobdt
assume rnobdt-def: bdt rnot var = Some rnobdt
assume onesvarno: Suc 0 < var no
with rnobdt-def lnot-dag rnot-dag lnobdt-def
show bdt (Node lnot no rnot) var =
  Some (Bdt-Node lnobdt (var no) rnobdt) by simp
next
fix rnot lnot rnort lnort rnobdt lnobdt rnorbdt lnorbdt
assume rnort-dag:
  Dag (repc (high no)) (repb  $\times$  low) (repb  $\times$  high) rnort
assume lnort-dag:
  Dag (repc (low no)) (repb  $\times$  low) (repb  $\times$  high) lnort
assume rnorbdt-def: bdt rnort var = Some rnorbdt
assume lnorbdt-def: bdt lnort var = Some lnorbdt

```

```

assume varno-bOne: Suc 0 < var no
with varno have Suc 0 < n by simp
with varrepno-n have Suc 0 < var (repc no) by simp
with rnorbd-def lnorbd-def
show bdt (Node lnort (repc no) rnort) var =
  Some (Bdt-Node lnorbd (var (repc no)) rnorbd)
by simp
next
fix rnobdt knobdt rnorbd lnorbd
assume lcong-eval: knobdt ~ lnorbd
assume rcong-eval: rnobdt ~ rnorbd
from varno varrepno-n have var (repc no) = var no by simp
with lcong-eval rcong-eval
show Bdt-Node knobdt (var no) rnobdt ~
  Bdt-Node lnorbd (var (repc no)) rnorbd
apply (unfold cong-eval-def)
apply (rule ext)
by simp
next
fix rnot lnot rnort lnort
assume lnort-repb:  $\forall no \in \text{set-of } lnort. \text{repb } no = no$ 
assume rnort-repb:  $\forall no \in \text{set-of } rnort. \text{repb } no = no$ 
assume rnort-in-repb-Nodesn:  $\text{set-of } rnort \subseteq \text{repb } \text{'Nodes } n \ ll$ 
assume lnort-in-repb-Nodesn:  $\text{set-of } lnort \subseteq \text{repb } \text{'Nodes } n \ ll$ 
from repbNodes-in-Nodes rnort-in-repb-Nodesn
have rnort-in-Nodesn:  $\text{set-of } rnort \subseteq \text{Nodes } n \ ll$ 
by blast
from repbNodes-in-Nodes lnort-in-repb-Nodesn
have lnort-in-Nodesn:  $\text{set-of } lnort \subseteq \text{Nodes } n \ ll$ 
by blast
have rnort-repc:  $\forall no \in \text{set-of } rnort. \text{repc } no = no$ 
proof
fix pt
assume pt-in-rnort:  $pt \in \text{set-of } rnort$ 
with rnort-in-Nodesn have  $pt \in \text{Nodes } n \ ll$ 
by blast
with Nodesn-notin-lln have  $pt \notin \text{set } (ll \ ! \ n)$ 
by auto
with repbc-nc have  $\text{repb } pt = \text{repc } pt$ 
by auto
with rnort-repb pt-in-rnort show  $\text{repc } pt = pt$ 
by auto
qed
have lnort-repc:  $\forall no \in \text{set-of } lnort. \text{repc } no = no$ 
proof
fix pt
assume pt-in-lnort:  $pt \in \text{set-of } lnort$ 
with lnort-in-Nodesn have  $pt \in \text{Nodes } n \ ll$ 
by blast

```



```

with Nodesn-notin-lln have  $pt \notin \text{set } (ll \ ! \ n)$ 
  by auto
with repbc-nc have  $\text{repb } pt = \text{repc } pt$ 
  by auto
with lnort-repb pt-in-lnort show  $\text{repc } pt = pt$ 
  by auto
qed
show  $\forall no \in \text{set-of } (\text{Node } lnort \ (\text{repc } no) \ rnort). \text{repc } no = no$ 
proof
  fix pt
  assume pt-in-rept:  $pt \in \text{set-of } (\text{Node } lnort \ (\text{repc } no) \ rnort)$ 
  show  $\text{repc } pt = pt$ 
  proof (cases  $pt \in \text{set-of } lnort$ )
    case True
    with lnort-repc show ?thesis
    by auto
  next
  assume pt-notin-lnort:  $pt \notin \text{set-of } lnort$ 
  show ?thesis
  proof (cases  $pt \in \text{set-of } rnort$ )
    case True
    with rnort-repc show ?thesis
    by auto
  next
  assume pt-notin-rnort:  $pt \notin \text{set-of } rnort$ 
  with pt-notin-lnort pt-in-rept have  $pt = \text{repc } no$ 
  by simp
  with rrno-eq-rno show  $\text{repc } pt = pt$ 
  by simp
qed
qed
qed
qed

with varrep rrno-eq-rno show ?thesis by simp
qed
qed
with rnonN show ?thesis by simp
qed
note while-while-prop=this
from wf-ll nsll
have  $\forall no \in \text{Nodes } n \ ll. no \notin \text{set } (ll \ ! \ n)$ 
  apply (simp add: Nodes-def length-ll-eq)
  apply clarify
  apply (drule no-in-one-ll)
  apply auto
done
with repbc-nc have  $\forall no \in \text{Nodes } n \ ll. \text{repb } no = \text{repc } no$ 

```

```

apply –
apply (rule ballI)
apply (erule-tac x=no in allE)
apply simp
done
then have repbNodes-repcNodes:
  repc ‘(Nodes n ll) = repc ‘(Nodes n ll)
apply –
apply rule
apply blast
apply rule
apply (erule imageE)
apply (erule-tac x=xa in ballE)
prefer 2
apply simp
apply rule
apply auto
done
then have repcNodes-repbNodes:
  repc ‘(Nodes n ll) = repb ‘(Nodes n ll)
by simp
have repb-c-dags-eq:
  Dags (repc ‘ Nodes n ll) (repc  $\times$  low) (repc  $\times$  high) =
  Dags (repb ‘ Nodes n ll) (repb  $\times$  low) (repb  $\times$  high)
apply –
apply rule
apply rule
apply (erule Dags.cases)
apply (rule DagsI)
prefer 4
apply rule
apply (erule Dags.cases)
apply (rule DagsI)
proof –
  fix x p t
  assume t-in-repcNodes: set-of t  $\subseteq$  repc ‘ Nodes n ll
  assume x-t: x=t
  with t-in-repcNodes repcNodes-repbNodes
  show set-of x  $\subseteq$  repb ‘ Nodes n ll
    by simp
next
  fix x p t
  assume t-in-repcNodes: set-of t  $\subseteq$  repc ‘ Nodes n ll
  assume t-dag: Dag p (repc  $\times$  low) (repc  $\times$  high) t
  assume t-nTip: t  $\neq$  Tip
  assume x-t: x=t
  from t-nTip t-dag have p  $\neq$  Null
    apply –
    apply (case-tac p=Null)

```

```

    apply auto
  done
with  $t\text{-nTip}$   $t\text{-dag}$  obtain  $lt$   $rt$  where  $t\text{-Node}: t = \text{Node } lt \ p \ rt$ 
  by auto
from  $t\text{-in-repcNodes}$   $t\text{-dag}$   $t\text{-nTip}$   $t\text{-Node}$  obtain  $q$  where
   $rq\text{-}p: \text{repc } q = p$  and  $q\text{-in-Nodes}: q \in \text{Nodes } n \ ll$ 
  apply simp
  apply (elim conjE)
  apply (erule imageE)
  apply auto
  done
from  $q\text{-in-Nodes}$  have  $\text{repb } q = \text{repc } q$ 
  by (rule Nodes-n-rep-nc [rule-format])
with  $rq\text{-}p$  have  $\text{repb}q\text{-}p: \text{repb } q = p$  by simp
from  $q\text{-in-Nodes}$ 
have  $\text{Dag } (\text{repb } q) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ t =$ 
   $\text{Dag } (\text{repc } q) (\text{repc } \times \text{low}) (\text{repc } \times \text{high}) \ t$ 
  by (rule Nodes-repb-Dags-eq [rule-format])
with  $t\text{-dag}$   $rq\text{-}p$  have  $\text{Dag } (\text{repb } q) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ t$  by simp
with  $\text{repb}q\text{-}p$   $x\text{-}t$  show  $\text{Dag } p (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ x$ 
  by simp
next
fix  $x \ p \ t$ 
assume  $t\text{-in-repcNodes}: \text{set-of } t \subseteq \text{repb } \text{' } \text{Nodes } n \ ll$ 
assume  $x\text{-}t: x = t$ 
with  $t\text{-in-repcNodes}$   $\text{repbNodes-repcNodes}$ 
show  $\text{set-of } x \subseteq \text{repc } \text{' } \text{Nodes } n \ ll$ 
  by simp
next
fix  $x \ p \ t$ 
assume  $t\text{-in-repcNodes}: \text{set-of } t \subseteq \text{repb } \text{' } \text{Nodes } n \ ll$ 
assume  $t\text{-dag}: \text{Dag } p (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \ t$ 
assume  $t\text{-nTip}: t \neq \text{Tip}$ 
assume  $x\text{-}t: x = t$ 
from  $t\text{-nTip}$   $t\text{-dag}$  have  $p \neq \text{Null}$ 
  apply -
  apply (case-tac  $p = \text{Null}$ )
  apply auto
  done
with  $t\text{-nTip}$   $t\text{-dag}$  obtain  $lt$   $rt$  where  $t\text{-Node}: t = \text{Node } lt \ p \ rt$ 
  by auto
from  $t\text{-in-repcNodes}$   $t\text{-dag}$   $t\text{-nTip}$   $t\text{-Node}$  obtain  $q$  where
   $rq\text{-}p: \text{repb } q = p$  and  $q\text{-in-Nodes}: q \in \text{Nodes } n \ ll$ 
  apply simp
  apply (elim conjE)
  apply (erule imageE)
  apply auto
  done
from  $q\text{-in-Nodes}$  have  $\text{repb } q = \text{repc } q$ 

```

```

    by (rule Nodes-n-rep-nc [rule-format])
  with rq-p have repbq-p: repc q = p by simp
  from q-in-Nodes
  have Dag (repb q) (repb  $\times$  low) (repb  $\times$  high) t =
    Dag (repc q) (repc  $\times$  low) (repc  $\times$  high) t
    by (rule Nodes-repbc-Dags-eq [rule-format])
  with t-dag rq-p have Dag (repc q) (repc  $\times$  low) (repc  $\times$  high) t by simp
  with repbq-p x-t show Dag p (repc  $\times$  low) (repc  $\times$  high) x
    by simp
next
  fix x p and t :: dag
  assume x-t: x = t
  assume t-nTip: t  $\neq$  Tip
  with x-t show x  $\neq$  Tip by simp
next
  fix x p and t :: dag
  assume x-t: x = t
  assume t-nTip: t  $\neq$  Tip
  with x-t show x  $\neq$  Tip by simp
qed
from pret-dag wf-ll nsll
have null-notin-Nodes-Suc-n: Null  $\notin$  Nodes (Suc n) ll
  by - (rule Null-notin-Nodes,auto simp add: length-ll-eq)
{ fix t1 t2
  assume t1-in-DagsNodesn:
    t1  $\in$  Dags (repc ' Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  assume t2-notin-DagsNodesn:
    t2  $\notin$  Dags (repc ' Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  assume t2-in-DagsNodesSucn:
    t2  $\in$  Dags (repc ' Nodes (Suc n) ll) (repc  $\times$  low) (repc  $\times$  high)
  assume isomorphic-dags-eq-asm:
     $\forall t1 t2. t1 \in Dags (repb ' Nodes n ll) (repb \times low) (repb \times high)$ 
     $\wedge t2 \in Dags (repb ' Nodes n ll) (repb \times low) (repb \times high)$ 
     $\longrightarrow isomorphic-dags-eq t1 t2 var$ 
  assume repbc-Dags:
    Dags (repc ' Nodes n ll) (repc  $\times$  low) (repc  $\times$  high) =
    Dags (repb ' Nodes n ll) (repb  $\times$  low) (repb  $\times$  high)
  from t1-in-DagsNodesn repbc-Dags
  have t1-repb-subnode:
    t1  $\in$  Dags (repb ' Nodes n ll) (repb  $\times$  low) (repb  $\times$  high)
    by simp
  from t2-in-DagsNodesSucn
  have t2-in-DagsNodesSucn:
    t2  $\in$  Dags (repc ' Nodes (Suc n) ll) (repc  $\times$  low) (repc  $\times$  high)
    by simp
  from repbNodes-in-Nodes repbNodes-repcNodes
  have repcNodesn-in-Nodesn: repc ' Nodes n ll  $\subseteq$  Nodes n ll
    by simp
  from t1-in-DagsNodesn obtain q where

```

Dag-q-Nodes-n:
Dag (repc q) (repc \times low) (repc \times high) t1 \wedge q \in Nodes n ll

proof (*elim Dags.cases*)

fix p t

assume t1-t: t1 = t

assume t-in-repcNodesn: set-of t \subseteq repc ' Nodes n ll

assume t-dag: Dag p (repc \times low) (repc \times high) t

assume t-nTip: t \neq Tip

assume obtain-prop: \bigwedge q. Dag (repc q) (repc \times low) (repc \times high) t1 \wedge q \in Nodes n ll \implies ?thesis

from t-nTip t-dag **have** p \neq Null

apply –

apply (*case-tac p=Null*)

apply *auto*

done

with t-nTip t-dag **obtain** lt rt **where** t-Node: t=Node lt p rt

by *auto*

from t-in-repcNodesn t-dag t-nTip t-Node **obtain** k **where** rk-p: repc k = p **and** k-in-Nodes: k \in Nodes n ll

apply *simp*

apply (*elim conjE*)

apply (*erule imageE*)

apply *auto*

done

with t1-t t-dag obtain-prop rk-p k-in-Nodes **show** ?thesis

by *auto*

qed

with wf-ll nsll **have** varq-sn: (var q < n)

apply (*simp add: Nodes-def*)

apply (*elim conjE*)

apply (*erule exE*)

apply (*simp add: wf-ll-def length-ll-eq*)

apply (*elim conjE*)

apply (*thin-tac \forall q. q \in set-of pret \longrightarrow q \in set (ll ! var q)*)

apply (*erule-tac x=x in allE*)

apply *auto*

done

from Dag-q-Nodes-n **have** q-in-Nodesn: q \in Nodes n ll

by *simp*

then **have** \exists k < n. q \in set (ll ! k)

by (*simp add: Nodes-def*)

with wf-ll nsll **have** q \notin set (ll ! n)

apply –

apply (*erule exE*)

apply (*rule-tac i=k and j=n in no-in-one-ll*)

apply (*auto simp add: length-ll-eq*)

done

with repbc-nc **have** repbc-q: repc q = repb q

apply –

```

apply (erule-tac  $x=q$  in allE)
apply auto
done
from normalize-prop q-in-Nodesn have  $\text{var } (\text{repc } q) \leq \text{var } q$ 
  apply –
  apply (erule-tac  $x=q$  in ballE)
  apply auto
  done
with repcb-q have  $\text{var-repc-q: var } (\text{repc } q) \leq \text{var } q$ 
  by simp
with varq-sn have  $\text{var-repc-q-n: var } (\text{repc } q) < n$ 
  by simp

from Nodes-subset Dag-q-Nodes-n while-while-prop
have ord-t1-var-rep: ordered t1 var  $\wedge \text{var } (\text{repc } q) \leq \text{var } q$ 
  apply (elim conjE)
  apply (erule-tac  $x=q$  in ballE)
  apply auto
  done
then have ord-t1: ordered t1 var by simp
from ord-t1-var-rep have varrep-q: var  $(\text{repc } q) \leq \text{var } q$  by simp
from t2-in-DagsNodesSucn have ord-t2: ordered t2 var
proof (elim Dags.cases)
  fix  $p t$ 
  assume t-in-repcNodes: set-of t  $\subseteq \text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  assume t-nTip: t  $\neq \text{Tip}$ 
  assume t2t: t2  $= t$ 
  assume t-dag: Dag p  $(\text{repc } \times \text{low}) (\text{repc } \times \text{high}) t$ 
  from t-in-repcNodes have x-in-repcNodesSucn:
     $\forall x. x \in \text{set-of } t \longrightarrow x \in \text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  apply –
  apply auto
  done
from t-nTip t-dag have  $p \neq \text{Null}$ 
  apply –
  apply (case-tac p=Null)
  apply auto
  done
with t-nTip t-dag obtain  $lt rt$  where t-Node: t=Node lt p rt
  by auto
then have  $p \in \text{set-of } t$ 
  by auto
with x-in-repcNodesSucn have  $p \in \text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  by simp
then obtain  $a$  where repca-p: p=repc a and
  a-in-NodesSucn: a  $\in \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  by auto
with repca-p while-while-prop t-dag t2t show ?thesis
  apply –

```

```

    apply (erule-tac x=a in ballE)
    apply (elim conjE exE)
    apply (subgoal-tac nort = t)
    prefer 2
    apply (simp add: Dag-unique)
    apply auto
    done
qed
from while-while-prop have while-prop-part:
   $\forall no \in Nodes (Suc n) ll.$ 
  var (repc no)  $\leq$  var no
  by auto
from while-while-prop have rep-rep-nort:
 $\forall no \in Nodes (n + 1) ll. (\exists nort. Dag (repc no) (repc \times low) (repc \times high)$ 
nort  $\wedge$ 
  ( $\forall no \in set-of nort. repc no = no$ )
  by auto
from repcNodes-in-Nodes null-notin-Nodes-Suc-n
have  $\forall no \in Nodes (n+1) ll. repc no \neq Null$ 
  by auto
with rep-rep-nort have  $\forall no \in Nodes (n+1) ll. repc (repc no) = (repc$ 
no)
  apply –
  apply (rule ballI)
  apply (erule-tac x=no in ballE)
  prefer 2
  apply simp
  apply (erule-tac x=no in ballE)
  apply (erule exE)
  apply (subgoal-tac repc no  $\in$  set-of nort)
  apply (elim conjE)
  apply (erule-tac x=repc no in ballE)
  apply simp
  apply simp
  apply (simp)
  apply (elim conjE)
  apply (thin-tac  $\forall no \in set-of nort. repc no = no$ )
  apply auto
  done
with t2-in-DagsNodesSucn t2-notin-DagsNodesn ord-t2 while-prop-part
wf-ll nsll repcNodes-in-Nodes obtain a where
t2-repc-dag:  $Dag (repc a) (repc \times low) (repc \times high) t2$  and
a-in-lln:  $a \in set (ll ! n)$ 
  apply –
  apply (drule restrict-root-Node)
  apply (auto simp add: length-ll-eq)
  done
with wf-ll nsll have a-in-pret:  $a \in set-of pret$ 
  by (simp add: wf-ll-def length-ll-eq)

```

```

from wf-ll nsl a-in-lln have vara-n: var a = n
  by (simp add: wf-ll-def length-ll-eq)
from a-in-lln rep-prop obtain
  repp-nNull: repc a ≠ Null and
  repp-reduce: (repc ∘ low) a = (repc ∘ high) a ∧ low a ≠ Null
  → repc a = (repc ∘ high) a and
  repp-share: ((repc ∘ low) a = (repc ∘ high) a → low a = Null)
  → repc a ∈ set (ll ! n) ∧
  repc (repc a) = repc a ∧
  (∀ no1 ∈ set (ll ! n). ((repc ∘ high) no1 = (repc ∘ high) a ∧
  (repc ∘ low) no1 = (repc ∘ low) a) = (repc a = repc no1))
  using [[simp-depth-limit=4]]
  by auto
from t2-repc-dag a-in-lln repp-nNull obtain lt2 rt2 where
  t2-Node: t2 = (Node lt2 (repc a) rt2)
  by auto
have isomorphic-dags-eq t1 t2 var
proof (cases (repc ∘ low) a = (repc ∘ high) a ∧ low a ≠ Null)
  case True
  note red=this
  then have red-case: (repc ∘ low) a = (repc ∘ high) a
    by simp
  from red have low-nNull: low a ≠ Null
    by simp
  with pret-dag prebdt-pret a-in-pret have highp-nNull: high a ≠ Null
    apply –
    apply (drule balanced-bdt)
    apply auto
    done
  from pret-dag ord-pret a-in-pret low-nNull highp-nNull
  have var-children-smaller: var (low a) < var a ∧ var (high a) < var a
    apply –
    apply (rule var-ordered-children)
    apply auto
    done
  from pret-dag a-in-pret have a-nNull: a ≠ Null
    apply –
    apply (rule set-of-nn [rule-format])
    apply auto
    done
  with a-in-pret highp-nNull pret-dag have high a ∈ set-of pret
    apply –
    apply (drule subelem-set-of-high)
    apply auto
    done
  with wf-ll have high a ∈ set (ll ! (var (high a)))
    by (simp add: wf-ll-def)
  with a-in-lln t2-repc-dag var-children-smaller vara-n
  have ∃ k < n. (high a) ∈ set (ll ! k)

```



```

    by auto
  then have higha-in-Nodesn:  $(high\ a) \in Nodes\ n\ ll$ 
    by (simp add: Nodes-def)
  then have rhigha-in-rNodesn:  $repc\ (high\ a) \in repc\ 'Nodes\ n\ ll$ 
    by simp
  from higha-in-Nodesn normalize-prop obtain rt where
    rt-dag:  $Dag\ (repb\ (high\ a))\ (repb\ \times\ low)\ (repb\ \times\ high)\ rt$  and
    rt-in-repbNort:  $set-of\ rt \subseteq repb\ 'Nodes\ n\ ll$ 
    apply -
    apply (erule-tac x=high a in ballE)
    apply auto
    done
  from rt-in-repbNort repbNodes-repcNodes
  have rt-in-repcNodesn:  $set-of\ rt \subseteq repc\ 'Nodes\ n\ ll$ 
    by blast
  from rt-dag higha-in-Nodesn
  have repcrt-dag:  $Dag\ (repc\ (high\ a))\ (repc\ \times\ low)\ (repc\ \times\ high)\ rt$ 
    apply -
    apply (drule Nodes-repbc-Dags-eq [rule-format])
    apply auto
    done
  have rt-nTip:  $rt \neq Tip$ 
  proof -
    have repc (high a) ≠ Null
    proof -
      note rhigha-in-rNodesn
      also have  $repc\ 'Nodes\ n\ ll \subseteq repc\ 'Nodes\ (Suc\ n)\ ll$ 
        using Nodes-subset
        by blast
      also have  $\dots \subseteq Nodes\ (Suc\ n)\ ll$ 
        using repcNodes-in-Nodes
        by simp
      finally show ?thesis
        using null-notin-Nodes-Suc-n
        by auto
    qed
  with repcrt-dag show ?thesis by auto
  qed
  from a-nNull a-in-pret low-nNull pret-dag have  $low\ a \in set-of\ pret$ 
    apply -
    apply (drule subelem-set-of-low)
    apply auto
    done
  with wf-ll have  $low\ a \in set\ (ll\ !\ (var\ (low\ a)))$ 
    by (simp add: wf-ll-def)
  with a-in-lln t2-repc-dag var-children-smaller vara-n
  have  $\exists k < n. (low\ a) \in set\ (ll\ !\ k)$ 
    by auto
  then have  $(low\ a) \in Nodes\ n\ ll$ 

```

```

    by (simp add: Nodes-def)
  then have rlow-in-rNodesn:  $\text{repc } (low\ a) \in \text{repc } \text{' } Nodes\ n\ ll$ 
    by simp
  show ?thesis
  proof -
    from repp-reduce low-nNull highp-nNull red-case
    have repc-p-def:  $\text{repc } a = \text{repc } (high\ a)$ 
      by (simp add: null-comp-def)
    with rt-in-repcNodesn repcrt-dag rhigha-in-rNodesn a-in-lln t2-repc-dag
      repc-p-def rt-nTip
    have t2-in-Dags-Nodesn:
       $t2 \in Dags\ (\text{repc } \text{' } Nodes\ n\ ll)\ (\text{repc } \propto\ low)\ (\text{repc } \propto\ high)$ 
      apply -
      apply (rule DagsI)
      apply simp
      apply (subgoal-tac t2=rt)
      apply (auto simp add: Dag-unique)
      done
      from t1-in-DagsNodesn t2-in-Dags-Nodesn repbc-dags-eq isomor-
    phic-dags-eq-asm
      show shared-t1-t2: isomorphic-dags-eq t1 t2 var
        apply -
        apply (erule-tac x=t1 in allE)
        apply (erule-tac x=t2 in allE)
        apply simp
        done
    qed
  next
  assume share:  $\neg ((\text{repc } \propto\ low)\ a = (\text{repc } \propto\ high)\ a \wedge low\ a \neq Null)$ 
  then
  have share:  $(\text{repc } \propto\ low)\ a \neq (\text{repc } \propto\ high)\ a \vee low\ a = Null$ 
    using [[simp-depth-limit=1]]
    by simp
  with repp-share obtain
    ra-in-llbn:  $\text{repc } a \in \text{set } (ll\ !\ n)$  and
    rra-ra:  $\text{repc } (\text{repc } a) = \text{repc } a$  and
    two-nodes-share:  $(\forall no1 \in \text{set } (ll\ !\ n).$ 
       $((\text{repc } \propto\ high)\ no1 = (\text{repc } \propto\ high)\ a \wedge$ 
       $(\text{repc } \propto\ low)\ no1 = (\text{repc } \propto\ low)\ a) = (\text{repc } a = \text{repc } no1))$ 
    using [[simp-depth-limit=3]]
    by simp
  from wf-ll ra-in-llbn nsl1 have var-repc-a-n:  $\text{var } (\text{repc } a) = n$ 
    by (auto simp add: wf-ll-def length-ll-eq)
  show ?thesis
  proof (auto simp add: isomorphic-dags-eq-def)
    fix bdt1
    assume bdt-t1:  $bdt\ t1\ var = \text{Some } bdt1$ 
    assume bdt-t2:  $bdt\ t2\ var = \text{Some } bdt1$ 
    show  $t1 = t2$ 

```

```

proof (cases t1)
  case Tip
    with bdt-t1 show ?thesis
    by simp
  next
    case (Node lt1 p1 rt1)
    note t1-Node=this
    with Dag-q-Nodes-n have p1=(repc q)
    by simp
    with t2-Node bdt-t1 bdt-t2 t1-Node have var (repc q) = var (repc a)
    apply -
    apply (rule same-bdt-var)
    apply auto
    done
    with var-repc-q-n var-repc-a-n show ?thesis
    by simp
  qed
qed
qed }
note mixed-Dags-case = this
from repbc-dags-eq isomorphic-dags-eq
have dags-shared:
   $\forall t1 t2. t1 \in \text{Dags } (\text{repc } \text{' Nodes } (Suc\ n)\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high) \wedge$ 
   $t2 \in \text{Dags } (\text{repc } \text{' Nodes } (Suc\ n)\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
   $\longrightarrow \text{isomorphic-dags-eq } t1\ t2\ var$ 
  apply -
  apply (rule Dags-Nodes-cases)
  apply (rule isomorphic-dags-eq-sym)
proof -
  fix t1 t2
  assume t1-in-Dagsn:
     $t1 \in \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
  assume t2-in-Dagsn:
     $t2 \in \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
  assume isomorphic-dags-eq-asm:
     $\forall t1 t2. t1 \in \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high) \wedge$ 
     $t2 \in \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
     $\longrightarrow \text{isomorphic-dags-eq } t1\ t2\ var$ 
  assume repb-repc-Dags:
     $\text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high) =$ 
     $\text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
  with t1-in-Dagsn t2-in-Dagsn isomorphic-dags-eq-asm
  show isomorphic-dags-eq t1 t2 var by simp
next
  fix t1 t2
  assume t1-in-DagsNodesn:
     $t1 \in \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 
  assume t2-notin-DagsNodesn:
     $t2 \notin \text{Dags } (\text{repc } \text{' Nodes } n\ ll) (\text{repc } \alpha\ low) (\text{repc } \alpha\ high)$ 

```

```

assume t2-in-DagsNodesSucn:
   $t2 \in \text{Dags } (\text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high})$ 
assume isomorphic-dags-eq-asm:
   $\forall t1\ t2. t1 \in \text{Dags } (\text{repb } ' \text{Nodes } n \text{ ll}) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \wedge$ 
   $t2 \in \text{Dags } (\text{repb } ' \text{Nodes } n \text{ ll}) (\text{repb } \times \text{low}) (\text{repb } \times \text{high})$ 
   $\longrightarrow \text{isomorphic-dags-eq } t1\ t2\ \text{var}$ 
assume repcb-Dags:
   $\text{Dags } (\text{repc } ' \text{Nodes } n \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high}) =$ 
   $\text{Dags } (\text{repb } ' \text{Nodes } n \text{ ll}) (\text{repb } \times \text{low}) (\text{repb } \times \text{high})$ 
from t1-in-DagsNodesn t2-notin-DagsNodesn t2-in-DagsNodesSucn
  isomorphic-dags-eq-asm repcb-Dags
show isomorphic-dags-eq t1 t2 var
  apply –
  apply (rule mixed-Dags-case)
  apply auto
  done
next
fix t1 t2
assume t1-in-DagsNodesSucn:
   $t1 \in \text{Dags } (\text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high})$ 
assume t1-notin-DagsNodesn:
   $t1 \notin \text{Dags } (\text{repc } ' \text{Nodes } n \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high})$ 
assume t2-in-DagsNodesSucn:
   $t2 \in \text{Dags } (\text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high})$ 
assume t2-notin-DagsNodesn:
   $t2 \notin \text{Dags } (\text{repc } ' \text{Nodes } n \text{ ll}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high})$ 

from t1-in-DagsNodesSucn have ord-t1: ordered t1 var
proof (elim Dags.cases)
  fix p t
  assume t-in-repcNodes: set-of t  $\subseteq$  repc ' Nodes (Suc n) ll
  assume t-nTip: t  $\neq$  Tip
  assume t2t: t1 = t
  assume t-dag: Dag p (repc  $\times$  low) (repc  $\times$  high) t
  from t-in-repcNodes
  have x-in-repcNodesSucn:
     $\forall x. x \in \text{set-of } t \longrightarrow x \in \text{repc } ' \text{Nodes } (\text{Suc } n) \text{ ll}$ 
    apply –
    apply auto
    done
  from t-nTip t-dag have p  $\neq$  Null
  apply –
  apply (case-tac p=Null)
  apply auto
  done
  with t-nTip t-dag obtain lt rt where t-Node: t=Node lt p rt
  by auto

```

```

then have  $p \in \text{set-of } t$ 
  by auto
with  $x\text{-in-repcNodesSucn}$  have  $p \in \text{repc } \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  by simp
then obtain  $a$  where
   $\text{repca-p: } p = \text{repc } a$  and  $a\text{-in-NodesSucn: } a \in \text{Nodes } (\text{Suc } n) \text{ ll}$ 
  by auto
with  $\text{repca-p}$   $\text{while-while-prop}$   $t\text{-dag}$   $t2t$  show ?thesis
  apply –
  apply (erule-tac  $x=a$  in ballE)
  apply (elim conjE exE)
  apply (subgoal-tac  $\text{nort} = t$ )
  prefer 2
  apply (simp add: Dag-unique)
  apply auto
  done
qed
from  $\text{while-while-prop}$ 
have  $\text{while-prop-part: } \forall \text{no} \in \text{Nodes } (\text{Suc } n) \text{ ll.}$ 
   $\text{var } (\text{repc } \text{no}) \leq \text{var } \text{no}$ 
  by auto
from  $\text{while-while-prop}$  have  $\text{rep-rep-nort:}$ 
   $\forall \text{no} \in \text{Nodes } (n + 1) \text{ ll.}$ 
   $(\exists \text{nort. Dag } (\text{repc } \text{no}) (\text{repc } \times \text{low}) (\text{repc } \times \text{high}) \text{nort} \wedge$ 
   $(\forall \text{no} \in \text{set-of } \text{nort. repc } \text{no} = \text{no}))$ 
  by auto
from  $\text{repcNodes-in-Nodes}$   $\text{null-notin-Nodes-Suc-n}$ 
have  $\forall \text{no} \in \text{Nodes } (n+1) \text{ ll. repc } \text{no} \neq \text{Null}$ 
  by auto
with  $\text{rep-rep-nort}$ 
have  $\text{rep-rep-no: } \forall \text{no} \in \text{Nodes } (n+1) \text{ ll. repc } (\text{repc } \text{no}) = (\text{repc } \text{no})$ 
  apply –
  apply (rule ballI)
  apply (erule-tac  $x=\text{no}$  in ballE)
  prefer 2
  apply simp
  apply (erule-tac  $x=\text{no}$  in ballE)
  apply (erule exE)
  apply (subgoal-tac  $\text{repc } \text{no} \in \text{set-of } \text{nort}$ )
  apply (elim conjE)
  apply (erule-tac  $x=\text{repc } \text{no}$  in ballE)
  apply simp
  apply simp
  apply (simp)
  apply (elim conjE)
  apply (thin-tac  $\forall \text{no} \in \text{set-of } \text{nort. repc } \text{no} = \text{no}$ )
  apply auto
  done
with  $t1\text{-in-DagsNodesSucn}$   $t1\text{-notin-DagsNodesn}$   $\text{ord-t1}$   $\text{while-prop-part}$ 

```

wf-ll

```

nsl repcNodes-in-Nodes obtain a1 where
t1-repc-dag: Dag (repc a1) (repc  $\times$  low) (repc  $\times$  high) t1 and
a1-in-lln: a1  $\in$  set (ll ! n)
apply –
apply (drule restrict-root-Node)
apply (auto simp add: length-ll-eq)
done
with wf-ll nsl have a1-in-pret: a1  $\in$  set-of pret
by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsl a1-in-lln have vara1-n: var a1 = n
by (simp add: wf-ll-def length-ll-eq)
from a1-in-lln rep-prop obtain
repa1-nNull: repc a1  $\neq$  Null and
repa1-reduce: (repc  $\times$  low) a1 = (repc  $\times$  high) a1  $\wedge$  low a1  $\neq$  Null
 $\longrightarrow$  repc a1 = (repc  $\times$  high) a1 and
repa1-share: ((repc  $\times$  low) a1 = (repc  $\times$  high) a1  $\longrightarrow$  low a1 = Null)
 $\longrightarrow$  repc a1  $\in$  set (ll ! n)  $\wedge$  repc (repc a1) = repc a1  $\wedge$ 
( $\forall$  no1  $\in$  set (ll ! n). ((repc  $\times$  high) no1 = (repc  $\times$  high) a1  $\wedge$ 
(repc  $\times$  low) no1 = (repc  $\times$  low) a1) = (repc a1 = repc no1))
using [[simp-depth-limit=4]]
by auto
from t1-repc-dag a1-in-lln repa1-nNull obtain lt1 rt1 where
t1-Node: t1 = (Node lt1 (repc a1) rt1)
by auto

```

```

from t2-in-DagsNodesSucn have ord-t2: ordered t2 var
proof (elim Dags.cases)
fix p t
assume t-in-repcNodes: set-of t  $\subseteq$  repc ' Nodes (Suc n) ll
assume t-nTip: t  $\neq$  Tip
assume t2t: t2 = t
assume t-dag: Dag p (repc  $\times$  low) (repc  $\times$  high) t
from t-in-repcNodes
have x-in-repcNodesSucn:
 $\forall$  x. x  $\in$  set-of t  $\longrightarrow$  x  $\in$  repc ' Nodes (Suc n) ll
apply –
apply auto
done
from t-nTip t-dag have p  $\neq$  Null
apply –
apply (case-tac p=Null)
apply auto
done
with t-nTip t-dag obtain lt rt where t-Node: t=Node lt p rt
by auto

```

```

then have  $p \in \text{set-of } t$ 
  by auto
with  $x\text{-in-repcNodesSucn}$  have  $p \in \text{repc } \text{'Nodes (Suc n) ll}$ 
  by simp
then obtain  $a$  where
   $\text{repca-p: } p = \text{repc } a$  and  $a\text{-in-NodesSucn: } a \in \text{Nodes (Suc n) ll}$ 
  by auto
with  $\text{repca-p while-while-prop t-dag t2t}$  show ?thesis
  apply –
  apply (erule-tac x=a in ballE)
  apply (elim conjE exE)
  apply (subgoal-tac nort = t)
  prefer 2
  apply (simp add: Dag-unique)
  apply auto
  done
qed
from  $\text{rep-rep-no t2-in-DagsNodesSucn t2-notin-DagsNodesn ord-t2 while-prop-part}$ 
wf-ll
   $\text{nsll repcNodes-in-Nodes}$  obtain  $a2$  where
   $t2\text{-repc-dag: Dag (repc } a2) (\text{repc } \times \text{low}) (\text{repc } \times \text{high}) t2$  and
   $a2\text{-in-lln: } a2 \in \text{set (ll ! n)}$ 
  apply –
  apply (drule restrict-root-Node)
  apply (auto simp add: length-ll-eq)
  done
with  $\text{wf-ll nsll}$  have  $a2\text{-in-pret: } a2 \in \text{set-of pret}$ 
  by (simp add: wf-ll-def length-ll-eq)
from  $\text{wf-ll nsll } a2\text{-in-lln}$  have  $\text{vara2-n: var } a2 = n$ 
  by (simp add: wf-ll-def length-ll-eq)
from  $a2\text{-in-lln rep-prop}$  obtain
   $\text{repa2-nNull: repc } a2 \neq \text{Null}$  and
   $\text{repa2-reduce: } (\text{repc } \times \text{low}) a2 = (\text{repc } \times \text{high}) a2 \wedge \text{low } a2 \neq \text{Null}$ 
   $\longrightarrow \text{repc } a2 = (\text{repc } \times \text{high}) a2$  and
   $\text{repa2-share: } ((\text{repc } \times \text{low}) a2 = (\text{repc } \times \text{high}) a2 \longrightarrow \text{low } a2 = \text{Null})$ 
   $\longrightarrow \text{repc } a2 \in \text{set (ll ! n)} \wedge \text{repc (repc } a2) = \text{repc } a2 \wedge$ 
   $(\forall \text{no1} \in \text{set (ll ! n)}. ((\text{repc } \times \text{high}) \text{no1} = (\text{repc } \times \text{high}) a2 \wedge$ 
   $(\text{repc } \times \text{low}) \text{no1} = (\text{repc } \times \text{low}) a2) = (\text{repc } a2 = \text{repc no1}))$ 
  using  $[[\text{simp-depth-limit} = 4]]$ 
  by auto
from  $t2\text{-repc-dag } a2\text{-in-lln repa2-nNull}$  obtain  $lt2 rt2$  where
   $t2\text{-Node: } t2 = (\text{Node } lt2 (\text{repc } a2) rt2)$ 
  by auto
show isomorphic-dags-eq t1 t2 var
proof (cases (repc  $\times$  low) a1 = (repc  $\times$  high) a1  $\wedge$  low a1  $\neq$  Null)
  case True
  note t1-red-cond=this
  with  $t1\text{-red-cond}$  have  $t1\text{-red-case: } (\text{repc } \times \text{low}) a1 = (\text{repc } \times \text{high}) a1$ 
  by simp

```

```

from t1-red-cond have lowa1-nNull: low a1  $\neq$  Null
  by simp
with pret-dag prebdt-pret a1-in-pret have higha1-nNull: high a1  $\neq$  Null
  apply –
  apply (drule balanced-bdt)
  apply auto
  done
from pret-dag ord-pret a1-in-pret lowa1-nNull higha1-nNull
have var-children-smaller-a1: var (low a1) < var a1  $\wedge$  var (high a1) <
var a1
  apply –
  apply (rule var-ordered-children)
  apply auto
  done
from pret-dag a1-in-pret have a1-nNull: a1  $\neq$  Null
  apply –
  apply (rule set-of-nn [rule-format])
  apply auto
  done
with a1-in-pret higha1-nNull pret-dag have high a1  $\in$  set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
  done
with wf-ll have high a1  $\in$  set (ll ! (var (high a1)))
  by (simp add: wf-ll-def)
with a1-in-lln t1-repc-dag var-children-smaller-a1 vara1-n
have  $\exists k < n. (high a1) \in set (ll ! k)$ 
  by auto
then have higha1-in-Nodesn: (high a1)  $\in$  Nodes n ll
  by (simp add: Nodes-def)
then have rhigha1-in-rNodesn: repc (high a1)  $\in$  repc 'Nodes n ll
  by simp
from higha1-in-Nodesn normalize-prop obtain rt1 where
  rt1-dag: Dag (repb (high a1)) (repb  $\times$  low) (repb  $\times$  high) rt1 and
  rt1-in-repbNort: set-of rt1  $\subseteq$  repb 'Nodes n ll
  apply –
  apply (erule-tac x=high a1 in ballE)
  apply auto
  done
from rt1-in-repbNort repbNodes-repcNodes
have rt1-in-repcNodesn: set-of rt1  $\subseteq$  repc 'Nodes n ll
  by blast
from rt1-dag higha1-in-Nodesn
have repcrt1-dag: Dag (repc (high a1)) (repc  $\times$  low) (repc  $\times$  high) rt1
  apply –
  apply (drule Nodes-repbc-Dags-eq [rule-format])
  apply auto
  done

```



```

have rt1-nTip: rt1  $\neq$  Tip
proof –
  have repc (high a1)  $\neq$  Null
  proof –
    note rhigha1-in-rNodesn
    also have repc ‘Nodes n ll  $\subseteq$  repc ‘Nodes (Suc n) ll
      using Nodes-subset
      by blast
    also have  $\dots \subseteq$  Nodes (Suc n) ll
      using repcNodes-in-Nodes
      by simp
    finally show ?thesis
      using null-notin-Nodes-Suc-n
      by auto
  qed
  with repcrt1-dag show ?thesis by auto
qed
from repa1-reduce lowa1-nNull higha1-nNull t1-red-case
have repc-a1-def: repc a1 = repc (high a1)
  by (simp add: null-comp-def)
with rt1-in-repcNodesn repcrt1-dag rhigha1-in-rNodesn a1-in-lln
  t1-repc-dag repc-a1-def rt1-nTip
have t1-in-Dags-Nodesn:
  t1  $\in$  Dags (repc ‘Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  apply –
  apply (rule DagsI)
  apply simp
  apply (subgoal-tac t1=rt1)
  apply (auto simp add: Dag-unique)
  done
show ?thesis
proof (cases (repc  $\times$  low) a2 = (repc  $\times$  high) a2  $\wedge$  low a2  $\neq$  Null)
  case True
  note t2-red-cond=this
  with t2-red-cond have t2-red-case: (repc  $\times$  low) a2 = (repc  $\times$  high) a2
    by simp
  from t2-red-cond have lowa2-nNull: low a2  $\neq$  Null
    by simp
  with pret-dag prebdt-pret a2-in-pret have higha2-nNull: high a2  $\neq$  Null

  apply –
  apply (drule balanced-bdt)
  apply auto
  done
from pret-dag ord-pret a2-in-pret lowa2-nNull higha2-nNull
have var-children-smaller-a2:
  var (low a2) < var a2  $\wedge$  var (high a2) < var a2
  apply –
  apply (rule var-ordered-children)

```

```

    apply auto
  done
from pret-dag a2-in-pret have a2-nNull: a2 ≠ Null
  apply –
  apply (rule set-of-nn [rule-format])
  apply auto
  done
with a2-in-pret higha2-nNull pret-dag have high a2 ∈ set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
  done
with wf-ll have high a2 ∈ set (ll ! (var (high a2)))
  by (simp add: wf-ll-def)
with a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n
have ∃ k < n. (high a2) ∈ set (ll ! k)
  by auto
then have higha2-in-Nodesn: (high a2) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rhigha2-in-rNodesn: repc (high a2) ∈ repc ‘Nodes n ll
  by simp
from higha2-in-Nodesn normalize-prop obtain rt2 where
  rt2-dag: Dag (repc (high a2)) (repc ∘ low) (repc ∘ high) rt2 and
  rt2-in-repbNort: set-of rt2 ⊆ repb ‘Nodes n ll
  apply –
  apply (erule tac x=high a2 in ballE)
  apply auto
  done
from rt2-in-repbNort repbNodes-repcNodes
have rt2-in-repcNodesn: set-of rt2 ⊆ repc ‘Nodes n ll
  by blast
from rt2-dag higha2-in-Nodesn
have repcrt2-dag: Dag (repc (high a2)) (repc ∘ low) (repc ∘ high) rt2
  apply –
  apply (drule Nodes-repbc-Dags-eq [rule-format])
  apply auto
  done
have rt2-nTip: rt2 ≠ Tip
proof –
  have repc (high a2) ≠ Null
  proof –
    note rhigha2-in-rNodesn
    also have repc ‘Nodes n ll ⊆ repc ‘Nodes (Suc n) ll
      using Nodes-subset
      by blast
    also have ... ⊆ Nodes (Suc n) ll
      using repcNodes-in-Nodes
      by simp
    finally show ?thesis

```

```

    using null-notin-Nodes-Suc-n
    by auto
  qed
  with repcrt2-dag show ?thesis by auto
  qed
  from repa2-reduce lowa2-nNull higha2-nNull t2-red-case
  have repc-a2-def: repc a2 = repc (high a2)
    by (simp add: null-comp-def)
  with rt2-in-repcNodesn repcrt2-dag rhigha2-in-rNodesn a2-in-lln
  t2-repc-dag repc-a2-def rt2-nTip
  have t2-in-Dags-Nodesn:
    t2 ∈ Dags (repc ‘ Nodes n ll) (repc ∝ low) (repc ∝ high)
  apply –
  apply (rule DagsI)
  apply simp
  apply (subgoal-tac t2=rt2)
  apply (auto simp add: Dag-unique)
  done
  from isomorphic-dags-eq t1-in-Dags-Nodesn t2-in-Dags-Nodesn
repcb-dags-eq
  show ?thesis
  by auto
next
  assume t2-share-cond:
    ¬ ((repc ∝ low) a2 = (repc ∝ high) a2 ∧ low a2 ≠ Null)
  from t1-in-Dags-Nodesn t2-notin-DagsNodesn t2-in-DagsNodesSucn
  isomorphic-dags-eq repbc-dags-eq
  show ?thesis
  apply –
  apply (rule mixed-Dags-case)
  apply auto
  done
  qed
next
  assume t1-share-cond:
    ¬ ((repc ∝ low) a1 = (repc ∝ high) a1 ∧ low a1 ≠ Null)
  with repa1-share obtain
    repca1-in-llbn: repc a1 ∈ set (ll ! n) and
    reprepa1: repc (repc a1) = repc a1 and
    twonodes-llbn-a1:
      (∀ no1 ∈ set (ll ! n). ((repc ∝ high) no1 = (repc ∝ high) a1 ∧
        (repc ∝ low) no1 = (repc ∝ low) a1) = (repc a1 = repc no1))
  using [[simp-depth-limit=2]]
  by auto
  show ?thesis
  proof (cases (repc ∝ low) a2 = (repc ∝ high) a2 ∧ low a2 ≠ Null)
  case True
    note t2-red-cond=this
  with t2-red-cond have t2-red-case: (repc ∝ low) a2 = (repc ∝ high) a2

```

```

    by simp
  from t2-red-cond have lowa2-nNull: low a2 ≠ Null
    by simp
  with pret-dag prebdt-pret a2-in-pret have higha2-nNull: high a2 ≠ Null

    apply –
    apply (drule balanced-bdt)
    apply auto
    done
  from pret-dag ord-pret a2-in-pret lowa2-nNull higha2-nNull
  have var-children-smaller-a2:
    var (low a2) < var a2 ∧ var (high a2) < var a2
    apply –
    apply (rule var-ordered-children)
    apply auto
    done
  from pret-dag a2-in-pret have a2-nNull: a2 ≠ Null
    apply –
    apply (rule set-of-nn [rule-format])
    apply auto
    done
  with a2-in-pret higha2-nNull pret-dag have high a2 ∈ set-of pret
    apply –
    apply (drule subelem-set-of-high)
    apply auto
    done
  with wf-ll
  have high a2 ∈ set (ll ! (var (high a2)))
    by (simp add: wf-ll-def)
  with a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n
  have ∃ k < n. (high a2) ∈ set (ll ! k)
    by auto
  then have higha2-in-Nodesn: (high a2) ∈ Nodes n ll
    by (simp add: Nodes-def)
  then have rhiga2-in-rNodesn: repc (high a2) ∈ repc ‘Nodes n ll
    by simp
  from higha2-in-Nodesn normalize-prop obtain rt2 where
    rt2-dag: Dag (repb (high a2)) (repb ∝ low) (repb ∝ high) rt2 and
    rt2-in-repbNort: set-of rt2 ⊆ repb ‘Nodes n ll
    apply –
    apply (erule-tac x=high a2 in ballE)
    apply auto
    done
  from rt2-in-repbNort repbNodes-repcNodes
  have rt2-in-repcNodesn: set-of rt2 ⊆ repc ‘Nodes n ll
    by blast
  from rt2-dag higha2-in-Nodesn
  have repcrt2-dag: Dag (repc (high a2)) (repc ∝ low) (repc ∝ high) rt2
    apply –

```

```

apply (drule Nodes-repbc-Dags-eq [rule-format])
apply auto
done
have rt2-nTip:  $rt2 \neq Tip$ 
proof –
  have rep (high a2)  $\neq Null$ 
  proof –
    note rhigha2-in-rNodesn
    also have rep ‘Nodes n ll  $\subseteq$  rep ‘Nodes (Suc n) ll
      using Nodes-subset
      by blast
    also have  $\dots \subseteq$  Nodes (Suc n) ll
      using repNodes-in-Nodes
      by simp
    finally show ?thesis
      using null-notin-Nodes-Suc-n
      by auto
  qed
with repert2-dag show ?thesis by auto
qed
from repa2-reduce lowa2-nNull higha2-nNull t2-red-case
have rep-a2-def:  $rep\ a2 = rep\ (high\ a2)$ 
  by (simp add: null-comp-def)
with rt2-in-repNodesn repert2-dag rhigha2-in-rNodesn a2-in-lln
  t2-rep-dag rep-a2-def rt2-nTip
have t2-in-Dags-Nodesn:
   $t2 \in Dags\ (rep\ 'Nodes\ n\ ll)\ (rep\ \times\ low)\ (rep\ \times\ high)$ 
  apply –
  apply (rule DagsI)
  apply simp
  apply (subgoal-tac t2=rt2)
  apply (auto simp add: Dag-unique)
  done
from t2-in-Dags-Nodesn t1-notin-DagsNodesn t1-in-DagsNodesSucn
  isomorphic-dags-eq repbc-dags-eq
have isomorphic-dags-eq t2 t1 var
  apply –
  apply (rule mixed-Dags-case)
  apply auto
  done
then show ?thesis
  by (simp add: isomorphic-dags-eq-sym)
next
assume t2-shared-cond:
 $\neg ((rep\ \times\ low)\ a2 = (rep\ \times\ high)\ a2 \wedge low\ a2 \neq Null)$ 
with repa2-share obtain
  repca2-in-llbn:  $rep\ a2 \in set\ (ll\ !\ n)$  and
  reprepa2:  $rep\ (rep\ a2) = rep\ a2$  and
  twonodes-llbn-a2:  $(\forall no1 \in set\ (ll\ !\ n).$ 

```

```

((repc  $\times$  high) no1 = (repc  $\times$  high) a2  $\wedge$ 
(repc  $\times$  low) no1 = (repc  $\times$  low) a2) = (repc a2 = repc no1))
using [[simp-depth-limit=2]]
by auto
from twonodes-llbn-a2 a1-in-lln
have share-a1-a2:
  ((repc  $\times$  high) a1 = (repc  $\times$  high) a2  $\wedge$ 
  (repc  $\times$  low) a1 = (repc  $\times$  low) a2) = (repc a2 = repc a1)
  by auto
from twonodes-llbn-a1 repca1-in-llbn reprep1
have children-repc-eq-a1: (repc  $\times$  high) (repc a1) = (repc  $\times$  high) a1
 $\wedge$ 
  (repc  $\times$  low) (repc a1) = (repc  $\times$  low) a1
  by auto
from twonodes-llbn-a2 repca2-in-llbn reprep2
have children-repc-eq-a2: (repc  $\times$  high) (repc a2) = (repc  $\times$  high) a2
 $\wedge$ 
  (repc  $\times$  low) (repc a2) = (repc  $\times$  low) a2
  by auto
from t1-Node t2-Node show ?thesis
proof (clarsimp simp add: isomorphic-dags-eq-def)
  fix bdt1
  assume t1-bdt: bdt (Node lt1 (repc a1) rt1) var = Some bdt1
  assume t2-bdt: bdt (Node lt2 (repc a2) rt2) var = Some bdt1
  show lt1 = lt2  $\wedge$  repc a1 = repc a2  $\wedge$  rt1 = rt2
  proof (cases bdt1)
    case Zero
    with t1-bdt t1-Node obtain
      lt1-Tip: lt1 = Tip and
      rt1-Tip: rt1 = Tip
    by simp
    from Zero t2-bdt t2-Node obtain
      lt2-Tip: lt2 = Tip and
      rt2-Tip: rt2 = Tip
    by simp
    from t1-repc-dag t1-Node lt1-Tip have (repc  $\times$  low) (repc a1) =
Null
      by simp
    with children-repc-eq-a1
    have repc-low-a1-Null: (repc  $\times$  low) a1 = Null
    by simp
    from t1-repc-dag t1-Node rt1-Tip
    have (repc  $\times$  high) (repc a1) = Null
    by simp
    with children-repc-eq-a1
    have repc-high-a1-Null: (repc  $\times$  high) a1 = Null
    by simp
    from t2-repc-dag t2-Node lt2-Tip have (repc  $\times$  low) (repc a2) =
Null

```

```

    by simp
  with children-repc-eq-a2
  have repc-low-a2-Null: (repc  $\times$  low) a2 = Null
    by simp
  from t2-repc-dag t2-Node rt2-Tip
  have (repc  $\times$  high) (repc a2) = Null
    by simp
  with children-repc-eq-a2
  have repc-high-a2-Null: (repc  $\times$  high) a2 = Null
    by simp
  with share-a1-a2 repc-low-a1-Null repc-high-a1-Null
    repc-low-a2-Null repc-high-a2-Null
  have repc a2 = repc a1
    by auto
  with lt1-Tip rt1-Tip lt2-Tip rt2-Tip show ?thesis
    by auto
next
case One
with t1-bdt t1-Node obtain
  lt1-Tip: lt1 = Tip and
  rt1-Tip: rt1 = Tip
  by simp
from One t2-bdt t2-Node obtain
  lt2-Tip: lt2 = Tip and
  rt2-Tip: rt2 = Tip
  by simp
from t1-repc-dag t1-Node lt1-Tip have (repc  $\times$  low) (repc a1) =
Null

  by simp
  with children-repc-eq-a1
  have repc-low-a1-Null: (repc  $\times$  low) a1 = Null
    by simp
  from t1-repc-dag t1-Node rt1-Tip have (repc  $\times$  high) (repc a1) =
Null

  by simp
  with children-repc-eq-a1
  have repc-high-a1-Null: (repc  $\times$  high) a1 = Null
    by simp
  from t2-repc-dag t2-Node lt2-Tip have (repc  $\times$  low) (repc a2) =
Null

  by simp
  with children-repc-eq-a2
  have repc-low-a2-Null: (repc  $\times$  low) a2 = Null
    by simp
  from t2-repc-dag t2-Node rt2-Tip have (repc  $\times$  high) (repc a2) =
Null

  by simp
  with children-repc-eq-a2
  have repc-high-a2-Null: (repc  $\times$  high) a2 = Null

```

```

    by simp
  with share-a1-a2 repc-low-a1-Null repc-high-a1-Null
    repc-low-a2-Null repc-high-a2-Null
  have repc a2 = repc a1
    by auto
  with lt1-Tip rt1-Tip lt2-Tip rt2-Tip show ?thesis
    by auto
next
case (Bdt-Node lbd t v rbd t)
note bdt-Node-case=this
with t1-bdt t1-Node obtain
  lbd-def-lt1: bdt lt1 var = Some lbd and
  rbd-def-rt1: bdt rt1 var = Some rbd
  by auto
from t2-bdt bdt-Node-case t2-Node obtain
  lbd-def-lt2: bdt lt2 var = Some lbd and
  rbd-def-rt2: bdt rt2 var = Some rbd
  by auto
from lbd-def-lt1 t1-Node t1-repc-dag children-repc-eq-a1
have (repc  $\times$  low) a1  $\neq$  Null
  by auto
then have low-a1-nNull: (low a1)  $\neq$  Null
  by (auto simp: null-comp-def)
from rbd-def-rt1 t1-Node t1-repc-dag children-repc-eq-a1
have (repc  $\times$  high) a1  $\neq$  Null
  by auto
then have high-a1-nNull: (high a1)  $\neq$  Null
  by (auto simp: null-comp-def)
from lbd-def-lt2 t2-Node t2-repc-dag children-repc-eq-a2
have (repc  $\times$  low) a2  $\neq$  Null
  by auto
then have low-a2-nNull: (low a2)  $\neq$  Null
  by (auto simp: null-comp-def)
from rbd-def-rt2 t2-Node t2-repc-dag children-repc-eq-a2
have (repc  $\times$  high) a2  $\neq$  Null
  by auto
then have high-a2-nNull: (high a2)  $\neq$  Null
  by (auto simp: null-comp-def)

from pret-dag ord-pret a1-in-pret low-a1-nNull high-a1-nNull
have var-children-smaller-a1:
  var (low a1) < var a1  $\wedge$  var (high a1) < var a1
  apply -
  apply (rule var-ordered-children)
  apply auto
  done
from pret-dag a1-in-pret have a1-nNull: a1  $\neq$  Null
  apply -

```



```

apply (rule set-of-nn [rule-format])
apply auto
done

with a1-in-pret high-a1-nNull pret-dag have high a1 ∈ set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
  done
with wf-ll
have high a1 ∈ set (ll ! (var (high a1)))
  by (simp add: wf-ll-def)
with a1-in-lln t1-repc-dag var-children-smaller-a1 vara1-n
have ∃ k < n. (high a1) ∈ set (ll ! k)
  by auto
then have higha1-in-Nodesn: (high a1) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rhiga1-in-rNodesn:
  repc (high a1) ∈ repc ‘Nodes n ll
  by simp
from higha1-in-Nodesn normalize-prop obtain rt1h where
rt1-dag: Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) rt1h and
rt1-in-repbNort: set-of rt1h ⊆ repc ‘Nodes n ll
  apply –
  apply (erule-tac x=high a1 in ballE)
  apply auto
  done
from rt1-in-repbNort repbNodes-repcNodes
have rt1-in-repcNodesn: set-of rt1h ⊆ repc ‘Nodes n ll
  by blast
from rt1-dag higha1-in-Nodesn
have repcrt1-dag:
  Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) rt1h
  apply –
  apply (drule Nodes-repbc-Dags-eq [rule-format])
  apply auto
  done
from t1-Node t1-repc-dag high-a1-nNull children-repc-eq-a1
have Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) rt1
  by (auto simp add: null-comp-def)
with repcrt1-dag have rt1h-rt1: rt1h = rt1 by (simp add: Dag-unique)
have rt1-nTip: rt1 ≠ Tip
proof –
  have repc (high a1) ≠ Null
  proof –
  note rhiga1-in-rNodesn
  also have
  repc ‘Nodes n ll ⊆ repc ‘Nodes (Suc n) ll

```

```

    using Nodes-subset
    by blast
  also have ...  $\subseteq$  Nodes (Suc n) ll
    using repcNodes-in-Nodes
    by simp
  finally show ?thesis
    using null-notin-Nodes-Suc-n
    by auto
qed
with repcrt1-dag rt1h-rt1 show ?thesis by auto
qed
with rt1-in-repcNodesn repcrt1-dag rhiga1-in-rNodesn a1-in-lln
  t1-repc-dag rt1h-rt1
have rt1-in-Dags-Nodesn:
  rt1  $\in$  Dags (repc 'Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  apply -
  apply (rule DagsI)
  apply auto
done

from a1-nNull a1-in-pret low-a1-nNull pret-dag
have low a1  $\in$  set-of pret
  apply -
  apply (drule subelem-set-of-low)
  apply auto
done
with wf-ll have
  low a1  $\in$  set (ll ! (var (low a1))) by (simp add: wf-ll-def)
with a1-in-lln t1-repc-dag var-children-smaller-a1 vara1-n
have  $\exists k < n. (low a1) \in set (ll ! k)$ 
  by auto
then have lowa1-in-Nodesn: (low a1)  $\in$  Nodes n ll
  by (simp add: Nodes-def)
then have rlowa1-in-rNodesn: repc (low a1)  $\in$  repc 'Nodes n ll
  by simp
from lowa1-in-Nodesn normalize-prop obtain lt1h where
  lt1-dag: Dag (repb (low a1)) (repb  $\times$  low) (repb  $\times$  high) lt1h and
  lt1-in-repbNort: set-of lt1h  $\subseteq$  repb 'Nodes n ll
  apply -
  apply (erule-tac x=low a1 in ballE)
  apply auto
done
from lt1-in-repbNort repbNodes-repcNodes
have lt1-in-repcNodesn: set-of lt1h  $\subseteq$  repc 'Nodes n ll
  by blast
from lt1-dag lowa1-in-Nodesn
have repclt1-dag: Dag (repc (low a1)) (repc  $\times$  low) (repc  $\times$  high)

```

lt1h

```

apply –
apply (drule Nodes-repbc-Dags-eq [rule-format])
apply auto
done
from t1-Node t1-repc-dag low-a1-nNull children-repc-eq-a1
have Dag (repc (low a1)) (repc  $\times$  low) (repc  $\times$  high) lt1
  by (auto simp add: null-comp-def)
with repclt1-dag have lt1h-lt1: lt1h = lt1 by (simp add: Dag-unique)
have lt1-nTip: lt1  $\neq$  Tip
proof –
  have repc (low a1)  $\neq$  Null
  proof –
    note rlowa1-in-rNodesn
    also have
      repc ‘Nodes n ll  $\subseteq$  repc ‘Nodes (Suc n) ll
      using Nodes-subset
      by blast
    also have ...  $\subseteq$  Nodes (Suc n) ll
      using repcNodes-in-Nodes
      by simp
    finally show ?thesis
      using null-notin-Nodes-Suc-n
      by auto
  qed
with repclt1-dag lt1h-lt1 show ?thesis by auto
qed
with lt1-in-repcNodesn repclt1-dag rlowa1-in-rNodesn a1-in-lln
  t1-repc-dag lt1h-lt1
have lt1-in-Dags-Nodesn:
  lt1  $\in$  Dags (repc ‘Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  apply –
  apply (rule DagsI)
  apply auto
  done

from pret-dag ord-pret a2-in-pret low-a2-nNull high-a2-nNull
have var-children-smaller-a2:
  var (low a2) < var a2  $\wedge$  var (high a2) < var a2
  apply –
  apply (rule var-ordered-children)
  apply auto
  done
from pret-dag a2-in-pret have a2-nNull: a2  $\neq$  Null
  apply –
  apply (rule set-of-nn [rule-format])
  apply auto
```

```

done

with a2-in-pret high-a2-nNull pret-dag have high a2 ∈ set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
  done
with wf-ll have high a2 ∈ set (ll ! (var (high a2)))
  by (simp add: wf-ll-def)
with a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n
have ∃ k < n. (high a2) ∈ set (ll ! k)
  by auto
then have higha2-in-Nodesn: (high a2) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rhigha2-in-rNodesn:
  repc (high a2) ∈ repc ‘Nodes n ll
  by simp
from higha2-in-Nodesn normalize-prop obtain rt2h where
rt2-dag: Dag (repc (high a2)) (repc ∝ low) (repc ∝ high) rt2h and
rt2-in-repbNort: set-of rt2h ⊆ repb ‘Nodes n ll
  apply –
  apply (erule-tac x=high a2 in ballE)
  apply auto
  done
from rt2-in-repbNort repbNodes-repcNodes
have rt2-in-repcNodesn: set-of rt2h ⊆ repc ‘Nodes n ll
  by blast
from rt2-dag higha2-in-Nodesn
have repcrt2-dag:
  Dag (repc (high a2)) (repc ∝ low) (repc ∝ high) rt2h
  apply –
  apply (drule Nodes-repbC-Dags-eq [rule-format])
  apply auto
  done
from t2-Node t2-repc-dag high-a2-nNull children-repc-eq-a2
have Dag (repc (high a2)) (repc ∝ low) (repc ∝ high) rt2
  by (auto simp add: null-comp-def)
with repcrt2-dag have rt2h-rt2: rt2h = rt2 by (simp add: Dag-unique)
have rt2-nTip: rt2 ≠ Tip
proof –
  have repc (high a2) ≠ Null
  proof –
    note rhigha2-in-rNodesn
  also have
    repc ‘Nodes n ll ⊆ repc ‘Nodes (Suc n) ll
  using Nodes-subset
  by blast
  also have ... ⊆ Nodes (Suc n) ll

```

```

    using repcNodes-in-Nodes
    by simp
    finally show ?thesis
    using null-notin-Nodes-Suc-n
    by auto
qed
with repcrt2-dag rt2h-rt2 show ?thesis by auto
qed
with rt2-in-repcNodesn repcrt2-dag rhiga2-in-rNodesn a2-in-lln
t2-repc-dag rt2h-rt2
have rt2-in-Dags-Nodesn:
  rt2 ∈ Dags (repc ‘ Nodes n ll) (repc ∝ low) (repc ∝ high)
  apply –
  apply (rule DagsI)
  apply auto
done

```

```

from a2-nNull a2-in-pret low-a2-nNull pret-dag
have low a2 ∈ set-of pret
  apply –
  apply (drule subelem-set-of-low)
  apply auto
done
with wf-ll have low a2 ∈ set (ll ! (var (low a2)))
  by (simp add: wf-ll-def)
with a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n
have ∃ k < n. (low a2) ∈ set (ll ! k)
  by auto
then have lowa2-in-Nodesn: (low a2) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rlowa2-in-rNodesn: repc (low a2) ∈ repc ‘ Nodes n ll
  by simp
from lowa2-in-Nodesn normalize-prop obtain lt2h where
  lt2-dag: Dag (repb (low a2)) (repb ∝ low) (repb ∝ high) lt2h and
  lt2-in-repbNort: set-of lt2h ⊆ repb ‘ Nodes n ll
  apply –
  apply (erule-tac x=low a2 in ballE)
  apply auto
done
from lt2-in-repbNort repbNodes-repcNodes
have lt2-in-repcNodesn: set-of lt2h ⊆ repc ‘ Nodes n ll
  by blast
from lt2-dag lowa2-in-Nodesn
have repclt2-dag: Dag (repc (low a2)) (repc ∝ low) (repc ∝ high)
  apply –
  apply (drule Nodes-repb-Dags-eq [rule-format])

```

lt2h

```

    apply auto
  done
  from t2-Node t2-repc-dag low-a2-nNull children-repc-eq-a2
  have Dag (repc (low a2)) (repc  $\times$  low) (repc  $\times$  high) lt2
    by (auto simp add: null-comp-def)
  with repclt2-dag have lt2h-lt2: lt2h = lt2 by (simp add: Dag-unique)
  have lt2-nTip: lt2  $\neq$  Tip
  proof -
    have repc (low a2)  $\neq$  Null
    proof -
      note rlowa2-in-rNodesn
      also have
        repc 'Nodes n ll  $\subseteq$  repc 'Nodes (Suc n) ll
        using Nodes-subset
        by blast
      also have ...  $\subseteq$  Nodes (Suc n) ll
        using repcNodes-in-Nodes
        by simp
      finally show ?thesis
        using null-notin-Nodes-Suc-n
        by auto
    qed
  with repclt2-dag lt2h-lt2 show ?thesis by auto
  qed
  with lt2-in-repcNodesn repclt2-dag rlowa2-in-rNodesn a2-in-lln
  t2-repc-dag lt2h-lt2
  have lt2-in-Dags-Nodesn:
    lt2  $\in$  Dags (repc 'Nodes n ll) (repc  $\times$  low) (repc  $\times$  high)
  apply -
  apply (rule DagsI)
  apply auto
  done

  from isomorphic-dags-eq lt1-in-Dags-Nodesn lt2-in-Dags-Nodesn
  repbc-dags-eq
  have shared-lt1-lt2: isomorphic-dags-eq lt1 lt2 var
  by auto
  from isomorphic-dags-eq rt1-in-Dags-Nodesn rt2-in-Dags-Nodesn
  repbc-dags-eq
  have shared-rt1-rt2: isomorphic-dags-eq rt1 rt2 var
  by auto

  from shared-lt1-lt2 lbd-def-lt1 lbd-def-lt2 have lt1-lt2: lt1 = lt2
  by (auto simp add: isomorphic-dags-eq-def)
  then have root-lt1-lt2: root lt1 = root lt2
  by auto
  from lt1-nTip t1-repc-dag t1-Node have (repc  $\times$  low) (repc a1)  $\neq$ 
  Null

```

by auto
 with *lt1-nTip* *t1-repc-dag* *t1-Node* obtain *llt1* *lt1p* *rlt1* where
 lt1-Node: *lt1* = *Node* *llt1* *lt1p* *rlt1*
 by auto
 with *t1-repc-dag* *t1-Node* *children-repc-eq-a1* *lt1-nTip*
 have *root-llt1*: *root* *llt1* = (*repc* \times *low*) *a1*
 by auto
 from *lt2-nTip* *t2-repc-dag* *t2-Node* have (*repc* \times *low*) (*repc* *a2*) \neq

Null

by auto
 with *lt2-nTip* *t2-repc-dag* *t2-Node* obtain *llt2* *lt2p* *rlt2* where
 lt2-Node: *lt2* = *Node* *llt2* *lt2p* *rlt2*
 by auto
 with *t2-repc-dag* *t2-Node* *children-repc-eq-a2* *lt2-nTip*
 have *root-llt2*: *root* *llt2* = (*repc* \times *low*) *a2*
 by auto
 from *root-llt1-llt2* *root-llt2* *root-llt1*
 have *low-a1-a2*: (*repc* \times *low*) *a1* = (*repc* \times *low*) *a2*
 by auto
 from *shared-rt1-rt2* *rbd-def-rt1* *rbd-def-rt2* have *rt1-rt2*: *rt1* = *rt2*
 by (*auto simp add: isomorphic-dags-eq-def*)
 then have *root-rt1-rt2*: *root* *rt1* = *root* *rt2*
 by auto
 from *rt1-nTip* *t1-repc-dag* *t1-Node* have (*repc* \times *high*) (*repc* *a1*) \neq

Null

by auto
 with *rt1-nTip* *t1-repc-dag* *t1-Node* obtain *lrt1* *rt1p* *rrt1* where
 rt1-Node: *rt1* = *Node* *lrt1* *rt1p* *rrt1*
 by auto
 with *t1-repc-dag* *t1-Node* *children-repc-eq-a1* *rt1-nTip*
 have *root-rt1*: *root* *rt1* = (*repc* \times *high*) *a1*
 by auto
 from *rt2-nTip* *t2-repc-dag* *t2-Node*
 have (*repc* \times *high*) (*repc* *a2*) \neq *Null*
 by auto
 with *rt2-nTip* *t2-repc-dag* *t2-Node* obtain *lrt2* *rt2p* *rrt2* where
 rt2-Node: *rt2* = *Node* *lrt2* *rt2p* *rrt2*
 by auto
 with *t2-repc-dag* *t2-Node* *children-repc-eq-a2* *rt2-nTip*
 have *root-rt2*: *root* *rt2* = (*repc* \times *high*) *a2*
 by auto
 from *root-rt1-rt2* *root-rt2* *root-rt1*
 have *high-a1-a2*: (*repc* \times *high*) *a1* = (*repc* \times *high*) *a2*
 by auto
 from *low-a1-a2* *high-a1-a2* *share-a1-a2*
 have *repc* *a1* = *repc* *a2*
 by auto
 with *llt1-llt2* *rt1-rt2* show *?thesis*
 by auto

```

      qed
    qed
  qed
  qed
  qed
  from termi dags-shared while-while-prop repcNodes-in-Nodes repc-nc n-var-prop

    wf-marking-m-ma
  show ?thesis
    by auto
  qed
  qed
  with srrl-precond all-nodes-same-var
  show ?thesis
    apply –
    apply (intro conjI)
    apply assumption+
    done
  qed
  qed

end

```

References

- [1] V. Ortner and N. Schirmer. Verification of BDD normalization. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 2005*, volume 3603 of *LNCS*, pages 261–277. Springer, 2005.