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

August 28, 2014

Contents

1 Introduction 3
2 BDD Abstractions 3
3 General Lemmas on BDD Abstractions 12
4 Definitions of Procedures 50
5 Proof of Procedure Eval 52
6 Proof of Procedure Levellist 53
7 Proof of Procedure ShareRep 79
8 Proof of Procedure ShareReduceRepList 85
9 Proof of Procedure Repoint 128
10 Proof of Procedure Normalize 145
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 ≠ lt
by (induct lt) auto

lemma [simp]: lt ≠ Node lt a rt
by (induct lt) auto

lemma [simp]: Node lt a rt ≠ rt
by (induct rt) auto

lemma [simp]: rt ≠ Node lt a rt
by (induct rt) auto

primrec set-of:: dag ⇒ ref set where
  set-of-Tip: set-of Tip = { }
  | set-of-Node: set-of (Node lt a rt) = { a } ∪ set-of lt ∪ set-of rt

primrec DAG:: dag ⇒ bool where
  DAG Tip = True
  | DAG (Node l a r) = (a ∉ set-of l ∧ a ∉ set-of r ∧ DAG l ∧ DAG r)

primrec subdag:: dag ⇒ dag ⇒ bool where
  subdag Tip t = False
  | subdag (Node l a r) t = (t=l ∨ t=r ∨ subdag l t ∨ subdag r t)

lemma subdag-size: subdag t s ⇒ size s < size t
by (induct t) auto

lemma subdag-neq: subdag t s ⇒ t ≠ s
by (induct t) (auto dest: subdag-size)

lemma subdag-trans [trans]: subdag t s ⇒ subdag s r ⇒ subdag t r
by (induct t) auto

lemma subdag-NodeD:
  subdag t (Node lt a rt) ⇒ subdag t lt ∧ subdag t rt
by (induct t) auto

lemma subdag-not-sym: ∃ t. [subdag s t; subdag t s] ⇒ P
by (induct s) (auto dest: subdag-NodeD)

instantiation dag :: order
begin

definition
  less-dag-def: s < (t::dag) ⟷ subdag t s
definition
le-dag-def: \( s \leq (t::\text{dag}) \iff s=t \lor s < t \)

lemma le-dag-refl: \((x::\text{dag}) \leq x\)
  by (simp add: le-dag-def)

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

lemma le-dag-antisym:
  fixes \(x::\text{dag} \) and \(y\)
  assumes \(x\leq y\) and \(y\leq x\)
  shows \(x = y\)
proof (cases \(x=y\))
  case True thus \(\text{thesis}\).
next
  case False
  with \(x\leq y\) \(y\leq x\) show \(\text{thesis}\)
  by (auto simp add: less-dag-def le-dag-def intro: subdag-not-sym)
qed

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

instance
  by default (auto simp add: dag-less-le le-dag-refl intro: le-dag-trans dest: le-dag-antisym)
lemma less-dag-Tip [simp]: \( \neg (x < \text{Tip}) \)
  by (simp add: less-dag-def)

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

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

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

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

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

lemma less-DAG-set-of:
  assumes x-less-y: \( x < y \)
  assumes DAG-y: \( \text{DAG} \; y \)
  shows \( \text{set-of} \; x \subset \text{set-of} \; y \)
proof (cases y)
  case Tip with x-less-y show ?thesis by simp
next
case (Node \; l \; a \; r)
  with DAG-y obtain a: \( a \notin \text{set-of} \; l \; a \notin \text{set-of} \; r \)
    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: \( \text{set-of} \; l \subseteq \text{set-of} \; y \)
    r-subset-y: \( \text{set-of} \; r \subseteq \text{set-of} \; y \)
    by auto
  from Node x-less-y have \( x=l \vee x=r \vee x < l \vee x < r \)
    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

lemma in-set-of-decomp:
  p ∈ set-of t = (∃ l r. t=(Node l p r) ∨ subdag t (Node l p 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 ∧
                           Dag (l p) l r lt ∧ Dag (r p) l r rt)

lemma Dag-Null [simp]: Dag Null l r t = (t = Tip)
    by (cases t) simp-all

lemma Dag-Ref [simp]:
  p≠Null ==> Dag p l r t = (∃ lt rt. t=Node lt p rt ∧
                          Dag (l p) l r lt ∧ Dag (r p) l r rt)
    by (cases t) auto

lemma Null-notin-Dag [simp, intro]: ∀ p l r. Dag p l r t ==> Null ∉ set-of t
    by (induct t) auto
theorem notin-Dag-update-l [simp]:
\[ \forall p, q \notin \text{set-of } t \Rightarrow \text{Dag } p \ (l(q := y)) \ r \ t = \text{Dag } p \ l \ r \ t \]
by (induct t) auto

theorem notin-Dag-update-r [simp]:
\[ \forall p, q \notin \text{set-of } t \Rightarrow \text{Dag } p \ l \ (r(q := y)) \ t = \text{Dag } p \ l \ r \ t \]
by (induct t) auto

lemma Dag-upd-same-l-lemma: \[ \forall p. p \neq \text{Null} \Rightarrow \neg \text{Dag } p \ (l(p := p)) \ r \ t \]
apply (induct t)
apply simp
apply (simp (no-asmp-simp) del: fun-upd-apply)
apply (simp (no-asmp-simp) only: fun-upd-apply refl if-True)
apply blast
done

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

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

lemma Dag-upd-same-r-lemma: \[ \forall p. p \neq \text{Null} \Rightarrow \neg \text{Dag } p \ l \ (r(p := p)) \ t \]
apply (induct t)
apply simp
apply (simp (no-asmp-simp) del: fun-upd-apply)
apply (simp (no-asmp-simp) only: fun-upd-apply refl if-True)
apply blast
done

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

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

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

lemma Dag-update-lI [intro]:

8
\[ \text{Dag } p \ l \ r \ t; \ q \notin \text{set-of } t \Rightarrow \text{Dag } p \ (l(q := y)) \ r \ t \]

by simp

**Lemma** \(\text{Dag-update-rI} \ [\text{intro}]:\)
\[ \text{Dag } p \ l \ r \ t; \ q \notin \text{set-of } t \Rightarrow \text{Dag } p \ l \ (r(q := y)) \ t \]
by simp

**Lemma** \(\text{Dag-unique}: \bigwedge p t_2. \text{Dag } p \ l \ r \ t_1 \Rightarrow \text{Dag } p \ l \ r \ t_2 \Rightarrow t_1 = t_2\)
by (induct \(t_1\) auto)

**Lemma** \(\text{Dag-unique1}: \text{Dag } p \ l \ r \ t \Rightarrow \exists! t. \text{Dag } p \ l \ r \ t\)
by (blast intro: \(\text{Dag-unique}\))

**Lemma** \(\text{Dag-subdag}: \bigwedge p. \text{Dag } p \ l \ r \ t \Rightarrow \text{subdag } t \ s \Rightarrow \exists q. \text{Dag } q \ l \ r \ s\)
by (induct \(t\) auto)

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

\begin{definition}

\textbf{lemma} \textit{Dag-root-not-in-subdag-r} [simp, intro]:
\begin{itemize}
  \item \textbf{assumes} \( \text{Dag (r p) l r t} \)
  \item \textbf{shows} \( p \notin \text{set-of t} \)
\end{itemize}
\end{definition}

\begin{proof}

\begin{itemize}
  \item \textbf{fix} \( \text{lt rt} \)
  \item \textbf{assume} \( t: t = \text{Node lt p rt} \)
  \item \textbf{from} \( \text{assms have} \ \text{Dag (r p) l r rt} \)
    \item \textbf{by} (clarsimp simp only: t Dag.simps)
  \item \textbf{with} \( \text{assms have} \ t = \text{rt} \)
    \item \textbf{by} (rule Dag-unique)
  \item \textbf{with} \( t \) \textbf{have} \( \text{False} \)
    \item \textbf{by} simp
\end{itemize}

\begin{itemize}
  \item \textbf{moreover}
  \item \textbf{fix} \( \text{lt rt} \)
  \item \textbf{assume} \( \text{subdag: subdag t (Node lt p rt)} \)
  \item \textbf{with} \( \text{assms obtain} \ q \ \text{where} \ \text{Dag q l r (Node lt p rt)} \)
    \item \textbf{by} (rule Dag-subdag [elim-format]) iprover
  \item \textbf{hence} \( \text{Dag (r p) l r rt} \)
    \item \textbf{by} auto
  \item \textbf{with} \( \text{assms have} \ t = \text{rt} \)
    \item \textbf{by} (rule Dag-unique)
  \item \textbf{moreover}
    \item \textbf{have} \( \text{subdag t rt} \)
    \item \textbf{proof} –
      \item \textbf{note} \( \text{subdag} \)
      \item \textbf{also have} \( \text{subdag (Node lt p rt) rt by simp} \)
      \item \textbf{finally show} \( \text{thesis} \).
  \item \textbf{qed}
  \item \textbf{ultimately have} \( \text{False} \)
    \item \textbf{by} (simp add: subdag-neq)
\end{itemize}

\begin{itemize}
\end{itemize}

ultimately show \( \text{thesis} \)
  by (auto simp add: in-set-of-decomp)
qed

\begin{definition}

\textbf{lemma} \textit{Dag-is-DAG}:
\begin{itemize}
  \item \( \text{\wedge p l r \ Dag p l r t =\Rightarrow DAG t} \)
\end{itemize}
\end{definition}

\begin{proof}

\begin{itemize}
  \item \textbf{induct} \( t \) \textbf{auto}
\end{itemize}

\begin{definition}

\textbf{lemma} \textit{heaps-eq-Dag-eq}:
\begin{itemize}
  \item \( \text{\wedge p. \forall x \in \text{set-of t}. \ l x = l' x \land r x = r' x} \)
\end{itemize}
\end{definition}
\[ \begin{align*}
\implies & \text{ Dag p l r t = Dag p l' r' t} \\
\text{by (induct t) auto}
\end{align*} \]

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

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

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

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

**Lemma** Dags-eq-hp-eq:
\[ \forall p p'. [\text{Dag p l r t; Dag p' l' r' t}] \implies p' = p \land (\forall no \in \text{set-of } t. l' no = l no \land r' no = r no) \]
by (induct t) auto

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

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

**Lemma** Dag-conv-isDag-dag:
\[ \text{Dag p l r t } = (\text{isDag p l r } \land t = \text{dag p l r}) \]
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:
\[ \text{Dag p l r t } \implies \text{dag p l r } = t \]
by (simp add: Dag-conv-isDag-dag)
3 General Lemmas on BDD Abstractions

definition subdag-eq :: dag ⇒ dag ⇒ bool where
subdag-eq t₁ t₂ = (t₁ = t₂ ∨ subdag t₁ t₂)

primrec root :: dag ⇒ ref
where
root Tip = Null |
root (Node l a r) = a

fun isLeaf :: dag ⇒ bool where
isLeaf Tip = False |
isLeaf (Node Tip v Tip) = True |
isLeaf (Node (Node l v₁ r) v₂ Tip) = False |
isLeaf (Node Tip v₁ (Node l v₂ r)) = False

datatype bdt = Zero | One | Bdt-Node bdt nat bdt

fun bdt-fn :: dag ⇒ (ref ⇒ nat) ⇒ bdt option where
bdt-fn Tip = (λbdtvar. None) |
bdt-fn (Node Tip vref Tip) =
    (λbdtvar.
        (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 r₁)) = (λbdtvar . None) |
bdt-fn (Node (Node l vref₁ r₁) vref Tip) = (λbdtvar . None) |
bdt-fn (Node (Node l₁ vref₁ r₁) vref (Node l₂ vref₂ r₂)) =
    (λbdtvar .
        (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-fn
primrec eval :: bdt ⇒ bool list ⇒ bool
where
  eval Zero env = False |
  eval One env = True |
  eval (Bdt-Node l v r) env = (if (env ! v) then eval r env else eval l env)

fun ordered-bdt :: bdt ⇒ bool where
  ordered-bdt Zero = True
  ordered-bdt One = True
  ordered-bdt (Bdt-Node (Bdt-Node l1 v1 r1) v (Bdt-Node l2 v2 r2)) =
    ((v1 < v) ∧ (v2 < v) ∧
     (ordered-bdt (Bdt-Node l1 v1 r1)) ∧ (ordered-bdt (Bdt-Node l2 v2 r2)))
  ordered-bdt (Bdt-Node (Bdt-Node l1 v1 r1) v r) =
    ((v1 < v) ∧ (ordered-bdt (Bdt-Node l1 v1 r1)))
  ordered-bdt (Bdt-Node l v (Bdt-Node l2 v2 r2)) =
    ((v2 < v) ∧ (ordered-bdt (Bdt-Node l2 v2 r2)))
  ordered-bdt (Bdt-Node l v r) = True

fun ordered :: dag ⇒ (ref⇒nat) ⇒ bool where
  ordered Tip = (λ var. True)
  ordered (Node (Node l1 v1 r1) v (Node l2 v2 r2)) =
    (λ var. (var v1 < var v ∧ var v2 < var v) ∧
     (ordered (Node l1 v1 r1) var) ∧ (ordered (Node l2 v2 r2) var))
  ordered (Node Tip v Tip) = (λ var. (True))
  ordered (Node Tip v r) =
    (λ var. (var (root r) < var v) ∧ (ordered r var))
  ordered (Node l v Tip) =
    (λ var. (var (root l) < var v) ∧ (ordered l var))

primrec max-var :: bdt ⇒ nat
where
  max-var Zero = 0 |
  max-var One = 1 |
  max-var (Bdt-Node l v r) = max v (max (max-var l) (max-var r))

lemma eval-zero: ![bdt (Node l v r) var = Some x; var (root (Node l v r)) = (0::nat)] !⇒ x = Zero
apply (cases l)
apply (cases r)
apply simp
apply simp
apply (cases r)
apply simp
apply simp
apply simp
apply simp

13
done

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

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

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

lemma balanced-bdt:
\[ p \ bdt1. [ \text{Dag p low high t}; \text{bdt t var } = \text{Some bdt1}; \text{no } \notin \text{set-of t}] \implies (\text{low no } = \text{Null}) = (\text{high no } = \text{Null}) \]
proof (induct t)
case Tip
then show ?case by simp
next
case (Node lt a rt)
note NN = this
have bdt1: \text{bdt (Node lt a rt) var } = \text{Some bdt1} by fact
have no-in-t: \text{no } \notin \text{set-of (Node lt a rt)} by fact
have p-tree: \text{Dag p low high (Node lt a rt)} by fact
from Node.prems obtain
  lt: \text{Dag (low p) low high lt and}
  rt: \text{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: split-if-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

primrec dag-map :: (ref ⇒ ref) ⇒ dag ⇒ 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 ⇒ (ref ⇒ bool) ⇒ (ref ⇒ bool) ⇒ bool ⇒ bool
where
  wf-marking t mark-old mark-new marked =
(case t of Tip ⇒ mark-new = mark-old
| (Node lt p rt) ⇒
  (∀ n. n ∈ set-of t → mark-new n = mark-old n) ∧
  (∀ n. n ∈ set-of t → mark-new n = marked))

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

**where**

dag-in-levellist t levellist var = (t ≠ Tip →
  (∀ st. subdag-eq t st → root st ∈ set (levellist ! (var (root st)))))

**lemma** set-of-nn: [\[ Dag p low high t; n ∈ set-of t \] \implies n ≠ Null

**apply** (induct t)

**apply** simp

**apply** auto

**done**

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

∀ p. n ∈ set-of t → Dag p low high t → ordered t var → var n ≤ 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

**apply** fastforce

**done**

**lemma** ordered-set-of:

\[ x. \text{[ordered t var; } x ∈ \text{set-of t}] \implies var x ≤ var (\text{root t})\]

**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

**apply** fastforce

**apply** fastforce

**16**
apply fastforce
done

lemma dag-setofD: \( \forall p \ low \ high \ n. \ [ \text{Dag} \ p \ low \ high \ t \ ; \ n \in \text{set-of} \ t] \implies \)
  \( (\exists nt. \text{Dag} \ n \ low \ high \ nt) \land (\forall nt. \text{Dag} \ n \ low \ high \ nt \implies \text{set-of} \ nt \subseteq \text{set-of} \ t) \)
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: \( \exists nt. \text{Dag} \ n \ low \ high \ nt \implies \text{set-of} \ nt \subseteq \text{set-of} \ t \)
apply (simp add: dag-setofD)
done

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

lemma subdag-parentdag-low: not <= lt \implies \ not <= (Node lt p rt)
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 <= rt \implies \ not <= (Node lt p rt)
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: \( \forall p \not= no \).
\[ \text{Dag} p \text{ low high } t; \text{Dag} no \text{ low high } not; \not\in \text{set-of } t \implies \not< t \]
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 \(\not=\) p
  with Node.prems have no-in-ltorrt: no \(\not\in\) set-of lt \(\lor\) no \(\not\in\) set-of rt
  by simp
  show \(?thesis\) proof (cases no \(\not\in\) 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 \(<\) 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 \(\not\in\) set-of lt
  with no-in-ltorrt have no-in-rt: no \(\not\in\) set-of rt
  by simp
  from Node.prems ppo have Dag (high po) low high rt
  by simp
  with Node.prems ppo no-in-rt have not \(<\) rt
  apply –
  apply (rule Node.hyps)
  apply assumption+
  done
  with Node.prems no-in-rt show \(?thesis\)
  apply –
  apply (rule subdag-parentdag-high)
apply simp
done
qed
qed
qed

lemma children-ordered: \[\text{\text{ordered}} (\text{Node} \; \text{lt} \; p \; \text{rt}) \; \text{var} \] \implies
\text{\text{ordered}} \; \text{lt} \; \text{var} \land \text{\text{ordered}} \; \text{rt} \; \text{var}
proof (cases \text{lt})
  case \text{Tip}
  note \text{ltTip=this}
  assume \text{orderedNode}: \text{\text{ordered}} (\text{Node} \; \text{lt} \; p \; \text{rt}) \; \text{var}
  show \; ?\text{thesis}
  proof (cases \text{rt})
    case \text{Tip}
    note \text{rtTip = this}
    with \text{ltTip} show \; ?\text{thesis}
    by simp
  next
    case (\text{Node} \; \text{lrt} \; l \; \text{rlt})
    note \text{ltNode} = this
    assume \text{orderedNode}: \text{\text{ordered}} (\text{Node} \; \text{lt} \; p \; \text{rt}) \; \text{var}
    show \; ?\text{thesis}
    proof (cases \text{rt})
      case \text{Tip}
      note \text{rtTip = this}
      with \text{orderedNode} \; \text{ltNode} show \; ?\text{thesis} \text{ by simp}
    next
      case (\text{Node} \; \text{lrt} \; r \; \text{rrt})
      note \text{rtNode} = this
      with \text{orderedNode} \; \text{ltNode} show \; ?\text{thesis} \text{ by simp}
  qed
qed

lemma ordered-subdag: \[\text{\text{ordered}} \; \text{t} \; \text{var}; \; \text{not} \; \leq \; \text{t}\] \implies
\text{\text{ordered}} \; \text{not} \; \text{var}
proof (induct \text{t})
  case \text{Tip}
  then show \; ?\text{thesis} \text{ by (simp add: less-dag-def le-dag-def)}
next
  case (\text{Node} \; \text{lt} \; p \; \text{rt})
  show \; ?\text{thesis}
  proof (cases \text{not} = \text{Node} \; \text{lt} \; p \; \text{rt})
    case \text{True}
    with \text{Node.prems} show \; ?\text{thesis} \text{ by simp}
next
  assume notnt: not \neq \text{Node \textit{lt} p \textit{rt}}
  with Node.prems have notstlrorrt: not \leq \textit{lt} \lor not \leq \textit{rt}
    apply -
    apply (simp add; less-dag-def le-dag-def)
    apply fastforce
    done
  from Node.prems have ord-lt: ordered \textit{lt} \textit{var}
    apply -
    apply (drule children-ordered)
    apply simp
    done
  from Node.prems have ord-rt: ordered \textit{rt} \textit{var}
    apply -
    apply (drule children-ordered)
    apply simp
    done
  show ?thesis
  proof (cases not \leq \textit{lt})
    case True
    with ord-lt show ?thesis
      apply -
      apply (rule Node.hyps)
      apply assumption+
      done
  next
    assume \neg not \leq \textit{lt}
    with notstltlorrt have notirt: not \leq \textit{rt}
      by simp
    from Node.hyps have hyprt: \[[\text{ordered \textit{rt} \textit{var}; not \leq \textit{rt}]} \implies \text{ordered not \textit{var}}\]
      by simp
    from notirt ord-rt show ?thesis
      apply -
      apply (rule hyprt)
      apply assumption+
      done
  qed
  qed
  qed

lemma subdag-ordered:
\land \text{not no p. [ordered \textit{t} \textit{var}; Dag \textit{p} low high \textit{t}; Dag no low high not; no \in set-of \textit{t}] \implies \text{ordered not \textit{var}}}
proof (induct \textit{t})
  case Tip
    from Tip.prems show ?case by simp
next
  case (Node \textit{lt} po \textit{rt})
note \( nN = \text{this} \)
show \(?\text{thesis}\)
proof (cases \( \text{lt} \))
  case Tip
  note \( \text{ltTip} = \text{this} \)
  show \(?\text{thesis}\)
  proof (cases \( \text{rt} \))
    case Tip
    from \( \text{Node.prems} \) have \( \text{ppo} : p = \text{po} \)
      by simp
    from \( \text{Tip ltTip Node.prems} \) have \( \text{no} = p \)
      by simp
    with \( \text{ppo Node.prems} \) have \( \text{not} = (\text{Node lt po rt}) \)
      by (simp del: Dag-Ref add: Dag-unique)
    with \( \text{Node.prems} \) show \(?\text{thesis}\) by simp
  next
    case (\( \text{Node lnot rn rrnot} \))
    from \( \text{Node.prems} \) \( \text{ltTip Node have} \) \( \text{ord-rt}: \text{ordered rt var} \)
      by simp
    from \( \text{Node.prems have ppo} : p = \text{po} \)
      by simp
    from \( \text{Node.prems have ponn} : \text{po} \neq \text{Null} \)
      by auto
    with \( \text{ppo ponn ltTip Node.prems} \) have \( \ast : \text{Dag} (\text{high po}) \text{ low high rt} \)
      by auto
    show \(?\text{thesis}\)
    proof (cases \( \text{no} = \text{po} \))
      case True
      with \( \text{ppo Node.prems} \) have \( \text{not} = \text{Node lt po rt} \)
        by (simp del: Dag-Ref add: Dag-unique)
      with \( \text{Node.prems show} \)?thesis
        by simp
    next
      case False
      with \( \text{Node.prems ltTip have no} \in \text{set-of rt} \)
        by simp
      with \( \text{ord-rt} \ast (\text{Dag no low high not}) \) show \(?\text{thesis}\)
        by (rule Node.hyps)
    qed
    qed
  next
    case (\( \text{Node llt l rl} \))
    note \( \text{ltnode} = \text{this} \)
    show \(?\text{thesis}\)
    proof (cases \( \text{rt} \))
      case Tip
      from \( \text{Node.prems Tip ltnode have} \) \( \text{ord-rt}: \text{ordered rt var} \)
        by simp
      from \( \text{Node.prems have ppo} : p = \text{po} \)
by simp
from Node.prems have ponN: po ≠ Null
  by auto
with ppo ponN Tip Node.prems ltNode have *: Dag (low po) low high lt
  by auto
show ?thesis
proof (cases no=po)
  case True
    with ppo Node.prems have not = (Node lt po rt)
    by (simp del: Dag-Ref add: Dag-unique)
    with Node.prems show ?thesis by simp
next
  case False
    with Node.prems 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.prems have ppo: p=po
  by simp
from Node.prems Node ltNode have ord-lt: ordered lt var
  by simp
from Node.prems Node ltNode have ord-rt: ordered rt var
  by simp
from Node.prems have ponN: po ≠ Null
  by auto
with ppo ponN Node Node.prems ltNode have lt-Dag: Dag (low po) low high lt
  by auto
with ppo ponN Node Node.prems ltNode have rt-Dag: Dag (high po) low high rt
  by auto
show ?thesis
proof (cases no ∈ set-of lt)
  case True
    with ord-lt lt-Dag Node.prems show ?thesis
next
  assume no ≠ po
  with Node.prems have no-in-ll-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.prems 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:
   \( \forall p \text{ no not. } [\text{ordered rt } \text{var; Dag p low high rt; Dag no low high not; no } \in \text{set-of rt}] \implies \text{ordered not var} \)
apply -
apply assumption
done
from no-in-rt ord-rt rt-Dag Node.prems show ?thesis
apply -
apply (rule hyp2)
apply assumption+
done
qed
qed
qed
qed
qed

lemma elem-set-of: \( \forall x. [x \in \text{set-of st}; \text{set-of st} \subseteq \text{set-of t}] \implies x \in \text{set-of t} \)
by blast

definition wf-ll :: dag ⇒ ref list list ⇒ (ref ⇒ nat ⇒ bool)
where
wf-ll t levellist var =
   \( (\forall p. p \in \text{set-of t} \implies p \in \text{set (levellist ! var p)}) \land \)
   \( (\forall k < \text{length levellist}. \forall p \in \text{set (levellist ! k)}. p \in \text{set-of t} \land \text{var p} = k) \)
definition cong-eval :: bdt ⇒ bdt ⇒ bool (infix ~ 60)
where cong-eval bdt1 bdt2 = (eval bdt1 = eval bdt2)

lemma cong-eval-sym: l ~ r = r ~ l
by (auto simp add: cong-eval-def)

lemma cong-eval-trans: l ~ r; r ~ t] ⇒ l ~ t
by (simp add: cong-eval-def)

lemma cong-eval-child-high: l ~ r ⇒ r ~ (Bdt-Node l v r)
apply (simp add: cong-eval-def)
apply (rule ext)
lemma cong-eval-child-low: \( l \sim r \Rightarrow l \sim (\text{Bdt-Node } l v r) \)
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 \(\propto\) 60)
where null-comp a b = (\(\lambda\) p. (if (b p) = Null then Null else ((a o b) p)))

lemma null-comp-not-Null [simp]: \( h q \neq Null \Rightarrow (g \propto h) q = g (h q) \)
by (simp add: null-comp-def)

lemma id-trans: (a \(\propto\) id) (b (c p)) = (a \(\propto\) 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 levellist = (\(\bigcup\) k \(\in\) \{k. k < i\} . set (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: [ set-of t \(\subseteq\) nodes; Dag p low high t; t \(\neq\) Tip] 
\[\Rightarrow\] t \(\in\) Dags nodes low high

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

definition isLeaf-pt :: \(\text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{bool}\)
where isLeaf-pt p low high = (low p = Null \(\land\) high p = Null)

definition repNodes-eq :: \(\text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{ref} \Rightarrow \text{bool}\)
where repNodes-eq p q low high rep ==
\((\text{rep } \propto \text{high }) p = (\text{rep } \propto \text{high }) q \land (\text{rep } \propto \text{low }) p = (\text{rep } \propto \text{low }) q\)
**Definition** `isomorphic-dags-eq :: dag ⇒ dag ⇒ (ref ⇒ nat) ⇒ bool`

**Where**

\[
\text{isomorphic-dags-eq } \mathbf{st}_1 \mathbf{st}_2 \mathbf{var} = \\
(\forall \mathbf{bdt}_1 \mathbf{bdt}_2. \; (\mathbf{bdt} \mathbf{st}_1 \mathbf{var} = \text{Some } \mathbf{bdt}_1 \land \mathbf{bdt} \mathbf{st}_2 \mathbf{var} = \text{Some } \mathbf{bdt}_2 \land (\mathbf{bdt}_1 = \mathbf{bdt}_2))) \\
\implies \mathbf{st}_1 = \mathbf{st}_2
\]

**Lemma** `isomorphic-dags-eq-sym`: `isomorphic-dags-eq \mathbf{st}_1 \mathbf{st}_2 \mathbf{var} = isomorphic-dags-eq \mathbf{st}_2 \mathbf{st}_1 \mathbf{var}`

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

**Definition** `shared :: dag ⇒ (ref ⇒ nat) ⇒ bool`

**Where**

\[
\text{shared } \mathbf{t} \mathbf{var} = (\forall \mathbf{st}_1 \mathbf{st}_2. \; (\mathbf{st}_1 < \mathbf{t} \land \mathbf{st}_2 < \mathbf{t}) \implies \text{isomorphic-dags-eq } \mathbf{st}_1 \mathbf{st}_2 \mathbf{var})
\]

**Fun** `reduced :: dag ⇒ bool`

**Where**

\[
\text{reduced } \text{Tip} = \text{True} \\
| \text{reduced } (\text{Node } \text{Tip} \mathbf{v} \text{Tip}) = \text{True} \\
| \text{reduced } (\text{Node } \mathbf{l} \mathbf{v} \mathbf{r}) = (\mathbf{l} \neq \mathbf{r} \land \text{reduced } \mathbf{l} \land \text{reduced } \mathbf{r})
\]

**Primrec** `reduced-bdt :: bdt ⇒ bool`

**Where**

\[
\text{reduced-bdt } \text{Zero} = \text{True} \\
| \text{reduced-bdt } \text{One} = \text{True} \\
| \text{reduced-bdt } (\text{Bdt-Node } \mathbf{lbdt} \mathbf{v} \mathbf{rbdt}) = \\
(\text{if } \mathbf{lbdt} = \mathbf{rbdt} \text{ then False} \\
\text{else } (\text{reduced-bdt } \mathbf{lbdt} \land \text{reduced-bdt } \mathbf{rbdt}))
\]

**Lemma** `replicate-elem`: \(i < n \implies (\text{replicate } n \mathbf{x} \! i) = \mathbf{x}\)

**Apply** `(induct n)`

**Apply** `simp`

**Apply** `(cases i)`

**Apply** `auto`

**Done**

**Lemma** `no-in-one-ll`:

\[
[\text{wf-ll } \text{pret } \text{levellista } \mathbf{var}; \; i < \text{length } \text{levellista}; \; j < \text{length } \text{levellista}; \; \mathbf{no} \in \text{set } (\text{levellista } \! i); \; i \neq j] \\
\implies \mathbf{no} \notin \text{set } (\text{levellista } \! j)
\]

**Apply** `(unfold wf-ll-def)`

**Apply** `(erule conjE)`

25
apply (rotate-tac 5)
apply (frule-tac x = \( i \) and \( ?R = no \in \text{set-of} \ pret \land \text{var no} = i \text{ in} \ allE)\)
apply (erule impE)
apply simp
apply (rotate-tac 6)
apply (erule-tac x = \( i \) in \ allE)
apply assumption
apply simp
apply (cases no \not\in \text{set} \ (\text{levellista} \ ! \ j))
apply assumption
apply (erule-tac x = \( j \) in \ allE)
apply assumption
apply (rotate-tac 7)
apply (erule-tac x = \( no \) in \ \text{ballE})
prefer 2
apply assumption
apply (elim conjE)
apply (thin-tac \( \forall q. q \in \text{set-of} \ pret \rightarrow q \in \text{set} \ (\text{levellista} \ ! \ var \ q)\))
apply fastforce
done

lemma nodes-in-wf-ll:
\[ [wf-ll \ pret \ \text{levellista} \ var; i < \text{length} \ \text{levellista}; no \in \text{set} \ (\text{levellista} \ ! \ i)] \]
\[ = \Rightarrow \text{var no} = i \land no \in \text{set-of} \ pret\]
apply (simp add: wf-ll-def)
done

lemma subelem-set-of-low:
\[ \forall p \in \text{set-of} t; x \neq \text{Null}; \text{low} x \neq \text{Null}; \text{Dag} p \text{ low} \text{ high} t \]
\[ \Rightarrow (\text{low} x) \in \text{set-of} t\]
proof (induct t)
case Tip
then show ?case by simp
next
case (Node lt po rt)
note\ tNode=\text{this}
then have ppo: p=po by simp
show ?case
proof (cases x=p)
case True
with Node.prems have lrootlt: low x = root lt
proof (cases lt)
case Tip
with True Node.prems show ?thesis
  by auto
next
case (Node lt l rlt)
with Node.prems True show ?thesis
by auto
qed
with True Node.prems have low \( x \in \text{set-of} \ (\text{Node} \ p \ rt) \)
proof (cases \text{lt})
  case \text{Tip}
  with \text{lxrootlt} Node.prems show \( ?\text{thesis} \)
  
  by simp
next
  case (Node \text{lt} l \text{rlt})
  with \text{lxrootlt} Node.prems show \( ?\text{thesis} \)
  
  by simp
qed
with \text{ppo} show \( ?\text{thesis} \)
by simp
next
assume \text{xnp}: \( x \neq p \)
with Node.prems have \( x \in \text{set-of} \text{lt} \lor x \in \text{set-of} \text{rt} \)
by simp
show \( ?\text{thesis} \)
proof (cases \( x \in \text{set-of} \text{lt} \))
  case True
  note \text{xinlt}=this
  from Node.prems have \text{Dag} (\text{low} p) \text{low} \text{high} \text{lt}
  
  by fastforce
  with Node.prems \text{True} have \text{low} x \in \text{set-of} \text{lt}
  apply –
  apply (rule Node.hyps)
  apply assumption+
  done
  with Node.prems show \( ?\text{thesis} \)
  
  by auto
next
assume \text{xnotinlt}: \( x \notin \text{set-of} \text{lt} \)
with \text{xnp} Node.prems have \text{xinrt}: \( x \in \text{set-of} \text{rt} \)
by simp
from Node.prems have \text{Dag} (\text{high} p) \text{low} \text{high} \text{rt}
by fastforce
with Node.prems \text{xinrt} have \text{low} x \in \text{set-of} \text{rt}
apply –
apply (rule Node.hyps)
apply assumption+
done
with Node.prems show \( ?\text{thesis} \)
by auto
qed
qed

lemma subelem-set-of-high:
\( \forall p. [ \; x \in \text{set-of} \; t; \; x \neq \text{Null}; \; \text{high} \; x \neq \text{Null}; \; \text{Dag} \; p \; \text{low} \; \text{high} \; t \; ] \Rightarrow (\text{high} \; x) \in \text{set-of} \; t \)

**proof**  
(\text{induct} \; t)

- **case**  \( \text{Tip} \)
  - then show  \(?\text{case by simp}\)

- **next**
  - **case**  \( (\text{Node} \; \text{lt} \; \text{po} \; \text{rt}) \)
    - note  \( \text{tNode} = \text{this} \)
    - then have  \( \text{ppo: p=po by simp} \)
    - show  \(?\text{case}\)
      - **proof**  \( (\text{cases x=p}) \)
        - **case**  \( \text{True} \)
          - with  \( \text{Node.prems have lxrootlt: high} \; x = \text{root} \; \text{rt} \)
          - **proof**  \( (\text{cases rt}) \)
            - **case**  \( \text{Tip} \)
              - with  \( \text{True Node.prems show } ?\text{thesis} \)
                - by  \( \text{auto} \)
            - **next**
              - **case**  \( (\text{Node} \; \text{lrt} \; l \; \text{rrt}) \)
                - with  \( \text{Node.prems True show } ?\text{thesis} \)
                  - by  \( \text{auto} \)
          - **qed**
        - with  \( \text{True Node.prems have high} \; x \in \text{set-of} \; (\text{Node} \; \text{lt} \; \text{p} \; \text{rt}) \)
          - **proof**  \( (\text{cases rt}) \)
            - **case**  \( \text{Tip} \)
              - with  \( \text{lxrootlt Node.prems show } ?\text{thesis} \)
                - by  \( \text{simp} \)
            - **next**
              - **case**  \( (\text{Node} \; \text{lrl} \; l \; \text{rrt}) \)
                - with  \( \text{lxrootlt Node.prems show } ?\text{thesis} \)
                  - by  \( \text{simp} \)
          - **qed**
        - with  \( \text{ppo show } ?\text{thesis} \)
          - by  \( \text{simp} \)
      - **next**
        - assume  \( \text{xnp: x \neq p} \)
        - with  \( \text{Node.prems have x \in set-of} \; \text{lt} \lor x \in \text{set-of} \; \text{rt} \)
          - by  \( \text{simp} \)
        - show  \(?\text{thesis}\)
          - **proof**  \( (\text{cases x \in set-of} \; \text{lt}) \)
            - **case**  \( \text{True} \)
              - note  \( \text{xinlt=this} \)
              - from  \( \text{Node.prems have Dag (low} \; p) \; \text{low} \; \text{high} \; \text{lt} \)
                - by  \( \text{fastforce} \)
              - with  \( \text{Node.prems True have high} \; x \in \text{set-of} \; \text{lt} \)
                - apply  \( \text{--} \)
                - apply  \( \text{(rule Node.hyps)} \)
                - apply  \( \text{assumption+} \)
                - done
with Node.prems show thesis
  by auto

next
  assume xnotinlt: x \notin \text{set-of lt}
with xnp Node.prems have xinrt: x \in \text{set-of rt}
  by simp
from Node.prems have Dag (high p) low high rt
  by fastforce
with Node.prems xinrt have high x \in \text{set-of rt}
  apply -
  apply (rule Node.hyps)
  apply assumption+
  done
with Node.prems show thesis
  by auto
qed
qed
qed

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

lemma Nodes-levellist-subset-t:
\[\text{wf-ll t levellist var; } i\leq \text{length levellist} \implies \text{Nodes i levellist } \subseteq \text{set-of t}\]
proof (induct i)
  case 0
  show ?case by (simp add: Nodes-def)
next
  case (Suc n)
from Suc.prems Suc.hyps have Nodens-in-t: Nodes n levellist \subseteq \text{set-of t}
  by simp
from Suc.prems have \forall x \in \text{set (levellist ! n)}, x \in \text{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 (levellist ! var q)})
  apply (erule-tac x=n in allE)
  apply (erule impE)
  apply simp
  apply fastforce
  done
with Suc.prems have set (levellist ! n) \subseteq \text{set-of t}
  apply blast
  done
with Suc.prems Nodens-in-t show ?case
  apply (simp add: Nodes-def)
apply (simp add: set-split)
done
qed

lemma bdt-child:
\[ [ \text{bdt} (\text{Node } (\text{Node } llt l rlt) p (\text{Node } lrt r rrt)) \text{ var } = \text{Some } bdt1] \]
\[ \implies \exists \text{ lbdt rbdt. } \text{bdt} (\text{Node } llt l rlt) \text{ var } = \text{Some } lbdt \land \text{bdt} (\text{Node } lrt r rrt) \text{ var } = \text{Some } rbdt \]
by (simp split: option.splits)

lemma subbdt-ex-dag-def:
\[ \land \text{bdt1 p. } [\text{Dag } p \text{ low high } t; \text{bdt } t \text{ var } = \text{Some } bdt1; \text{Dag no low high not}; \text{no } \in \text{set-of } t] \implies \exists \text{ bdt2. } \text{bdt not var } = \text{Some } bdt2 \]
proof (induct t)
case Tip
then show ?case by simp
next
case (\text{Node } lt po rt)
  note pNode=this
  with Node.premss have p-po: p=po by simp
  show ?case
  proof (cases no \neq po)
    case True
    note no-eq-po=this
    from p-po Node.premss no-eq-po have not = (\text{Node } lt po rt) by (simp add: Dag-unique del: Dag-Ref)
    with Node.premss have bdt not var = Some bdt1 by (simp add: le-dag-def)
    then show ?thesis by simp
  next
    assume no \neq po
    with Node.premss have no-in-lt-or-rt: no \in \text{set-of } lt \lor no \in \text{set-of } rt by simp
    show ?thesis
    proof (cases no \in \text{set-of } lt)
      case True
      note no-in-lt=this
      from Node.premss p-po have lt-dag: Dag (low po) low high lt by simp
      from Node.premss have lbdt-def: \exists lbdt. bdt lt var = Some lbdt
      proof (cases lt)
        case Tip
        with Node.premss no-in-lt show ?thesis by (simp add: le-dag-def)
      next
        case (\text{Node } llt l rlt)
        note lNode=this
        show ?thesis
        proof (cases rt)
          case Tip
          note rNode=this
          with lNode Node.premss 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 @$\notin@$ 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: @$\land@$ rbdt. [Dag (high po) low high rt; bdt rt var = Some rbdt; Dag no low high not; no @$\notin@$ set-of rt] $\implies$ @$\exists@$ 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
  next
    case (Node llt r rlt)
      note lNode=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
then obtain rbdt where bdt rt var = Some rbdt..
  with Node.prems rt-dag no-in-rt show ?thesis
    apply -
    apply (rule hyp2)
    apply assumption+
    done
  qed
  qed

lemma subbdt-ex:
\[ \text{bddt1} \cdot [(\text{Node lst stp rst}) \leq t; \text{bdd t var} = \text{Some bdt1}] \]
\[ \implies \exists \text{bdd2}. \text{bdd} (\text{Node lst stp rst}) \text{ var} = \text{Some bdt2} \]

**proof** (induct t)

- **case** Tip
  - then show \(?case\) by (simp add: le-dag-def)

**next**

- **case** (Node lt p rt)
  - note pNode\ =\ this
  - show \(?thesis\) proof (cases Node lst stp rst = Node lt p rt)
    - case True
      - with Node.prems show \(?thesis\) by simp
  - **next**
    - assume Node lst stp rst \(\neq\) Node lt p rt
      - with Node.prems 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.prems have lbdt-def: \(\exists\) lbdt. bdt lt var = Some lbdt
      - proof (cases lt)
        - case Tip
          - with Node.prems in-lt show \(?thesis\) by simp
      - **next**
        - case (Node llt l rlt)
          - note lNode\ \=\ this
          - show \(?thesis\) proof (cases r)
            - 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 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: \( \forall \text{Node lst stp rst} \subseteq \text{rt}; \ bdt \text{ rt var} = \text{Some rbdt} \) \( \implies \exists \text{bdt2}. \ bdt (\text{Node lst stp rst}) \ bdt \text{ var} = \text{Some bdt2} \)
by simp
from Node.prems have rbdt-def: \( \exists \text{rbdt}. \ bdt \text{ rt var} = \text{Some rbdt} \)
proof (cases rt)
case Tip
with Node.prems in-rt show \(?thesis\) by simp
qed
next
case (Node lrt l rrt)
proof
  cases llt
  case Tip
  note lNode=this
  show \(?thesis\)
  proof (cases llt)
  case Tip
  note lNode=this
  with rNode Node.prems show \(?thesis\) by simp
  qed
  qed
next
case (Node lrt r rrt)
proof
  cases lrt
  case Tip
  then show \(?case\) by simp
qed
next
case (Node llt po rt)
then have ppo: \(p = \text{po}\) by simp
proof (cases no = po)
case True
  note no-po=this
from Node.prems have \(\text{var}\ (\text{low po}) < \text{var po} \wedge \text{var}\ (\text{high po}) < \text{var po}\)
proof (cases llt)
case Tip

lemma var-ordered-children:
\( \forall p. [\text{Dag p low high t; ordered t var; no} \in \text{set-of t; low no} \neq \text{Null; high no} \neq \text{Null}] \implies \text{var}\ (\text{low no}) < \text{var no} \wedge \text{var}\ (\text{high no}) < \text{var no}\)
proof (induct t)
case Tip
then show \(?case\) by simp
next
case (Node llt po rt)
then have ppo: \(p = \text{po}\) by simp
proof (cases no = po)
case True
  note no-po=this
from Node.prems have \(\text{var}\ (\text{low po}) < \text{var po} \wedge \text{var}\ (\text{high po}) < \text{var po}\)
proof (cases llt)
case Tip

33
note \( \text{ltTip}=\text{this} \)
with Node.prems no-po ppo show \( \text{thesis} \) by simp
next
case \( \text{Node llt l rlt} \)
  note lNode=\text{this}
  show \( \text{thesis} \)
proof (cases rt)
  case Tip
  with Node.prems no-po ppo show \( \text{thesis} \) by simp
next
case \( \text{Node lrt r rrt} \)
  note rNode=\text{this}
  with Node.prems ppo no-po lNode show \( \text{thesis} \) by (simp del: Dag-Ref)
qed
qed
with no-po show \( \text{thesis} \) by simp
next
assume \( \text{no} \neq \text{po} \)
with Node.prems have no-in-ltrt: \( \text{no} \notin \text{set-of lt} \lor \text{no} \notin \text{set-of rt} \)
  by simp
show \( \text{thesis} \)
proof (cases \( \text{no} \notin \text{set-of lt} \))
  case True
  note no-in-.lt=\text{this}
  from Node.prems ppo have \( \text{lt-dag}: \text{Dag} (\text{low po}) \text{ low high lt} \)
    by simp
  from Node.prems have ord-lt: ordered lt var
    apply
    apply (erule children-ordered)
    apply simp
    done
  from no-in-lt lt-dag ord-lt Node.prems show \( \text{thesis} \)
    apply
    apply (rule Node.hyps)
    apply assumption+
    done
next
assume \( \text{no} \notin \text{set-of rt} \)
with no-in-lrtr have no-in-rt: \( \text{no} \notin \text{set-of rt} \) by simp
from Node.prems ppo have rt-dag: \( \text{Dag} (\text{high po}) \text{ low high rt} \) by simp
from Node.prems have hyp2: \([\text{Dag} (\text{high po}) \text{ low high rt}; \text{ordered rt var}; \text{no} \in \text{set-of rt}; \text{low no} \neq \text{Null}; \text{high no} \neq \text{Null}]\)
  \( \Rightarrow \text{var (low no)} < \text{var no} \land \text{var (high no)} < \text{var no} \)
    by simp
from Node.prems have ord-rt: ordered rt var
  apply
  apply (erule children-ordered)
  apply simp

34
done
from rt-dag ord-rt no-in-rt Node.prems 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 ∝ low) (repb ∝ high) nort and
ord-pret: ordered pret var and
wf-llb: wf-ll pret levellistb var and
nbll: nb < length levellistb and
repcb-c: ∀ nt. nt ≠ set (levellistb ! nb) → repb nt = repc nt and
xsnb-in-pret: ∀ x ∈ set-of nort. var x < nb ∧ x ∈ set-of pret
shows ∀ x ∈ set-of nort. ((repc ∝ low) x = (repb ∝ low) x ∧
(repc ∝ high) x = (repb ∝ high) x)
proof (rule ballI)
fix x
assume x-in-nort: x ∈ set-of nort
with nort-dag have xnN: x ≠ 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 ∈ set-of pret
by blast
show (repc ∝ low) x = (repb ∝ low) x ∧ (repc ∝ high) x = (repb ∝ high) x
proof (cases (low x) ≠ Null)
case True
with pret-dag prebdt-pret x-in-pret have highN: (high x) ≠ Null
apply –
apply (drule balanced-bdt)
apply assumption+
apply simp
done
from x-in-pret ord-pret highN True have children-var-smaller: var (low x) <
var x ∧ 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) ∈ set-of pret
  apply
  apply (drule subelem-set-of-low)
  apply assumption
  apply (thin-tac x ≠ Null)
  apply assumption+
  done
with wf-llb have low x ∈ set (levellistb ! (var (low x)))
  by (simp add: wf-ll-def)
with wf-llb nbsll lowxsnb have low x ∉ 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) ∈ set-of pret
  apply
  apply (drule subelem-set-of-high)
  apply assumption
  apply (thin-tac x ≠ Null)
  apply assumption+
  done
with wf-llb have high x ∈ set (levellistb ! (var (high x)))
  by (simp add: wf-ll-def)
with wf-llb nbsll highxsnb have high x ∉ set (levellistb ! nb)
  apply
  apply (rule-tac $?i=(var (high x)) and $?j=nb in no-in-one-ll)
  apply auto
  done
with repbc-nc have repchigh: repc (high x) = repb (high x)
  by auto
with repclow show ?thesis
  by (simp add: null-comp-def)
next
  assume ¬ low x ≠ 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 repclowNull: (repc \propto low) x = Null
  by (simp add: null-comp-def)
from lowxNull have repbNull: (repb \propto low) x = Null
  by (simp add: null-comp-def)
with repclowNull have lowxrepbc: (repc \propto low) x = (repb \propto low) x
  by simp
from highxNull have repchighNull: (repc \propto high) x = Null
  by (simp add: null-comp-def)
from highxNull have repbNull: (repb \propto high) x = Null
  by (simp add: null-comp-def)
with repchighNull have highxrepbc: (repc \propto high) x = (repb \propto high) x
  by simp
with lowxrepbc 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 set-of pret \land var x < nb
apply (simp add: wf-ll-def Nodes-def)
apply (erule conjE)
apply (thin-tac \forall q. q \in set-of pret \implies q \in set (levellista \setminus var q))
apply (erule exE)
apply (elim conjE)
apply (erule_tac x=k in allE)
apply (erule impE)
apply arith
apply (erule-tac x=x in ballE)
apply auto
done

lemma bdt-Some-var1-One:
\forall x. [bdt t var = Some x; var (root t) = 1]
  \implies x = One \land t = (Node Tip (root t) Tip)
proof (induct t)
case Tip
  then show thesis
  by simp
next
case (Node lt p rt)
ote tNode = this
show thesis
proof (cases lt)
case Tip
  note ltTip= this
  show thesis
  proof (cases rt)
  qed
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:
\[ x \cdot [ bdt \ t \ var = \text{Some} \ x; \ var \ (root \ t) = 0 ] \implies x = \text{Zero} \land t = (Node \ Tip \ (root \ t) \ Tip) \]
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 \( \text{ltNode \ Node.prems} \) show \( ?thesis \) by auto
next
  case (Node lrt r rrt)
  note rtNode=this
  with \( \text{ltNode \ Node.prems} \) show \( ?thesis \) by auto
qed
qed

lemma reduced-children-parent:
\[
\begin{align*}
\text{reduced } l; \ l &= (\text{Node } llt lp rlt) \; \text{reduced } r; \ r = (\text{Node } lrt rp rrt); \\
lp \neq rp & \implies \text{reduced } (\text{Node } l p r)
\end{align*}
\]
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:
\[
\begin{align*}
\text{wf-ll } \text{pret } \text{levellista } \text{var}; \ nb < \text{length } \text{levellista} \; p \in \text{Nodes } nb \text{ levellista} & \implies p \notin \text{set } (\text{levellista } \setminus \text{nb}) \\
\end{align*}
\]
apply (simp add: Nodes-def)
apply (erule exE)
apply (rule-tac i=k and j=nb in no-in-one-ll)
apply auto
done

lemma Nodes-var-pret:
\[
\begin{align*}
\text{wf-ll } \text{pret } \text{levellista } \text{var}; \ nb < \text{length } \text{levellista} \; p \in \text{Nodes } nb \text{ levellista} & \implies \text{var } p < nb \land p \in \text{set-of } \text{pret} \\
\end{align*}
\]
apply (simp add: Nodes-def wf-ll-def)
apply (erule conjE)
apply (thin-tac \( \forall q. \ q \in \text{set-of } \text{pret} \implies q \in \text{set } (\text{levellista } \setminus \text{var } q) \))
apply (erule exE)
apply (erule-tac x=k 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 $\text{DagsSucnb}$: $t \in \text{Dags (Nodes (Suc nb) levellista)}$ low high
shows $\exists \ p \ . \ \text{Dag p low high t} \land p \in \text{Nodes (Suc nb) levellista}$
proof —
from $t$ in $\text{DagsSucnb}$ obtain $p$ where $t$-dag: $\text{Dag p low high t}$ and $t$-subset-Nodes: 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 $\text{subdag-dag}$:
$\forall p \ . \ [[\text{Dag p low high t}; st < = t]] \implies \exists \ st p. \ \text{Dag stp low high st}$
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 $\text{Node.prems}$ have $p$-po: $p = po$
by simp
show ?case
proof (cases $st = \text{Node lt po rt}$)
case True
note $st$-t=this
with $\text{Node.prems}$ show ?thesis
by auto
next
assume $st$-nt: $st \neq \text{Node lt po rt}$
with $\text{Node.prems}$ $p$-po have $st$-subdag-lt-$rt$: $st \leq lt \lor st \leq rt$
by (auto simp add:le-dag-def less-dag-def)
from $\text{Node.prems}$ $p$-po obtain $lp$ $rp$ where $lt$-dag: $\text{Dag lp low high lt}$ and $rt$-dag: $\text{Dag rp low high rt}$
by auto
show ?thesis
proof (cases $st \leq lt$)
case True
note $st$-lt=this
with $lt$-dag show ?thesis
apply
apply (rule Node.hyps)
apply auto
done
next  
  assume ¬ st ≤ lt  
  with st-subdag-lt-rt have st-rt: st <= rt  
    by simp  
  from Node.hyps have rhyp: [Dag rp low high rt; st ≤ rt] ⇒ ∃ stp. 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 ∈ Dags nodes low high and  
st-t: st <= t and  
st-nTip: st ≠ Tip  
shows st ∈ 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 ⊆ nodes and t-nTip: t ≠ Tip  
    by (fastforce elim: Dags.cases)  
  from st-t have set-of st ⊆ set-of t  
    by (simp add: le-dag-set-of)  
  with t-subset-Nodes have st-subset-fnctNodes: set-of st ⊆ 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: ∅ t1 t2. P t1 t2 var = P t2 t1 var and  
dags-in-lower-levels:  
    ∨ t1 t2. [t1 ∈ Dags (fact ‘(Nodes n levellista)) low high;  
      t2 ∈ Dags (fact ‘(Nodes n levellista)) low high]  
        ⇒ P t1 t2 var and  
dags-in-mixed-levels:
\[ \forall t1 \ t2. \ (t1 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ n \ \text{levellista})) \ \text{low high}; \\
    t2 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high}; \\
    t2 \notin \text{Dags}(\text{fnct}'(\text{Nodes} \ n \ \text{levellista})) \ \text{low high}] \\
\implies \ P \ t1 \ t2 \ \text{var and dags-in-high-level:} \\
\forall t1 \ t2. \ (t1 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high}; \\
    t1 \notin \text{Dags}(\text{fnct}'(\text{Nodes} \ n \ \text{levellista})) \ \text{low high}; \\
    t2 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high}; \\
    t2 \notin \text{Dags}(\text{fnct}'(\text{Nodes} \ n \ \text{levellista})) \ \text{low high}] \\
\implies \ P \ t1 \ t2 \ \text{var} \\
\text{shows} \ \forall \ t1 \ t2. \ (t1 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high} \land \\
    t2 \in \text{Dags}(\text{fnct}'(\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high}] \\
\implies \ P \ t1 \ t2 \ \text{var} \\
\text{proof (intro \ allI \ impI, \ elim \ conjE)} \\
\text{fix \ t1 \ t2} \\
\text{assume \ t1-in-higher-levels: \ } t1 \in \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high} \\
\text{assume \ t2-in-higher-levels: \ } t2 \in \text{Dags}(\text{fnct}' \ (\text{Nodes} \ (\text{Suc} \ n) \ \text{levellista})) \ \text{low high} \\
\text{show \ } P \ t1 \ t2 \ \text{var} \\
\text{proof (cases \ } t1 \in \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high)} \\
\text{case \ True} \\
\text{note \ t1-in-ll} = \text{this} \\
\text{show \ ?thesis} \\
\text{proof (cases \ } t2 \in \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high)} \\
\text{case \ True} \\
\text{note \ t2-in-ll} = \text{this} \\
\text{with \ } t1-in-ll \ \text{dags-in-lower-levels} \ \text{show \ ?thesis} \\
\text{by \ simp} \\
\text{next} \\
\text{assume \ } t2-notin-ll: \ t2 \notin \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high} \\
\text{with \ } t1-in-ll \ \text{t2-in-higher-levels} \ \text{dags-in-mixed-levels} \ \text{show \ ?thesis} \\
\text{by \ simp} \\
\text{qed} \\
\text{next} \\
\text{assume \ } t1-notin-ll: \ t1 \notin \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high} \\
\text{show \ ?thesis} \\
\text{proof (cases \ } t2 \in \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high)} \\
\text{case \ True} \\
\text{note \ t2-in-ll} = \text{this} \\
\text{with \ } \text{dags-in-mixed-levels} \ \text{t1-in-higher-levels} \ \text{t1-notin-ll} \ \text{P-sym} \ \text{show \ ?thesis} \\
\text{by \ auto} \\
\text{next} \\
\text{assume \ } t2-notin-ll: \ t2 \notin \text{Dags}(\text{fnct}' \ (\text{Nodes} \ n \ \text{levellista})) \ \text{low high} \\
\text{with \ } t1-notin-ll \ \text{t1-in-higher-levels} \ \text{t2-in-higher-levels} \ \text{dags-in-high-level} \ \text{show \ ?thesis} \\
\text{by \ simp} \\
\text{qed} \\
\text{qed}
**Lemma** Null-notin-Nodes: \[\text{Dag} p \text{ low high } t; \text{ nb } \leq \text{ length levellista}; \text{ wf-ll } t \text{ levellista } \text{ var}\] \[\Rightarrow \text{ Null } \notin \text{ Nodes } \text{ nb levellista}\]

**Apply** (simp add: Nodes-def wf-ll-def del: Dag-Ref)

**Apply** (rule allI)

**Apply** (rule impI)

**Apply** (elim conjE)

**Apply** (thin-tac \( \forall q \). \( q \in \text{ var} \))

**Apply** (erule-tac \( x=k \) in allE)

**Apply** (erule impE)

**Apply** simp

**Apply** (erule-tac \( x=\text{Null} \) in ballE)

**Apply** (erule conjE)

**Apply** (drule set-of-nn [rule-format])

**Apply** auto

**Done**

**Lemma** Nodes-in-pret: \[\text{wf-ll } t \text{ levellista } \text{ var}\] \[\Rightarrow \text{ Nodes } \text{ nb levellista } \subseteq \text{ set-of } t\]

**Apply** –

**Apply** rule

**Apply** (simp add: wf-ll-def Nodes-def)

**Apply** (erule exE)

**Apply** (elim conjE)

**Apply** (thin-tac \( \forall q \). \( q \in \text{ set-of } t \) \( \Rightarrow q \in \text{ set } \) (levellista ! var \( q \)))

**Apply** (erule-tac \( x=k \) in allE)

**Apply** (erule impE)

**Apply** simp

**Apply** (erule-tac \( x=x \) in ballE)

**Apply** auto

**Done**

**Lemma** restrict-root-Node:

\[ t \in \text{ Dags } (\text{repc } '\text{Nodes } (\text{Suc } \text{ nb} ) \text{ levellista} ) \text{ (repc } \propto \text{ low} ) \text{ (repc } \propto \text{ high} ); \text{ t } \notin \text{ Dags } (\text{repc } '\text{Nodes } \text{ nb levellista} ) \text{ (repc } \propto \text{ low} ) \text{ (repc } \propto \text{ high} ); \text{ ordered } t \text{ var}; \forall \text{ no } \in \text{ Nodes } (\text{Suc } \text{ nb} ) \text{ levellista}. \text{ var (repc no) } \leq \text{ var no } \land \text{ repc (repc no) = repc no}; \text{ wf-ll pret levellista } \text{ var}; \text{ nb } \leq \text{ length levellista};\text{ repc } '\text{Nodes } (\text{Suc } \text{ nb} ) \text{ levellista } \subseteq \text{ Nodes } (\text{Suc } \text{ nb} ) \text{ levellista}\]

\[\Rightarrow \exists q. \text{ Dag } (\text{repc q) (repc } \propto \text{ low} ) \text{ (repc } \propto \text{ high} ) t \land q \in \text{ set } (\text{levellista } ! \text{ nb})\]

**Proof** (elim Dags.cases)

**Fix** \( p \) and ta :: dag

**Assume** t-notin-DagsNodesnb: \( t \notin \text{ Dags } (\text{repc } '\text{Nodes } \text{ nb levellista} ) \text{ (repc } \propto \text{ low} ) \text{ (repc } \propto \text{ high} )\)

**Assume** t-ta: \( t = \text{ ta} \)

**Assume** ta-in-repc-NodesSucnb: set-of ta \( \subseteq \text{ repc } '\text{Nodes } (\text{Suc } \text{ nb} ) \text{ levellista} \)

**Assume** ta-dag: \( \text{Dag } p \text{ (repc } \propto \text{ low} ) \text{ (repc } \propto \text{ high} ) t a\)

**Assume** ta-nTip: \( t a \neq \text{ Tip} \)
assume ord-t: ordered t var
assume varrep-prop: \( \forall \ no \in \text{Nodes} \ (\text{Suc} \ nb) \ levellista. \ var \ (\text{repc} \ no) \leq \ var \ no \)
\& \text{repc} \ (\text{repc} \ no) = \text{repc} \ no
assume wf-lla: wf-ll \ pret \ levellista \ var
assume nb<lla: nb < length levellista
assume repcNodes-in-Nodes: \( \text{repc} \ '\text{Nodes} \ (\text{Suc} \ nb) \ levellista \subseteq \text{Nodes} \ (\text{Suc} \ nb) \)
levellista
from ta-nTip ta-dag have p-nNull: p\# \ 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 \text{repc} \ '\text{Nodes} \ (\text{Suc} \ nb) \ levellista
by auto
show \( ?\)thesis
proof (cases p \in \text{repc} \ '(\text{set} \ (\text{levellista} \ ! \ nb)))
case True
then obtain q where
  p-repca: p=\text{repc} \ q \ and
  a-in-llaq: q \in \text{set} \ (\text{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-llaq: p \notin \text{repc} \ '(\text{set} \ (\text{levellista} \ ! \ nb))
with p-in-repc-NodesSucnb have p-in-repc-Nodes-Sucnb: p \in \text{repc} \ '\text{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 k=nb)
apply simp
apply (rule-tac x=k in exI)
apply auto
done
have t \in \text{Dags} \ (\text{repc} \ '\text{Nodes} \ nb \ levellista) \ (\text{repc} \propto \text{low}) \ (\text{repc} \propto \text{high})
proof –
  have set-of t \subseteq \text{repc} \ '\text{Nodes} \ nb \ levellista
  proof (rule)
  \fix \ x :: \text{ref}
  \assume x-in-t: x \in \set-of t

44
with ord-t have var x <= var (root t)
apply –
apply (rule ordered-set-of)
apply auto
done
with t-ta ta-Node have var-varp: var x <= var p
by auto
from p-in-repc-Nodesnb obtain k where ksnb: k < nb and p-in-repc-lla:
p ∈ repc '(set (levellista ! k))
by (auto simp add: Nodes-def ImageE)
then obtain q where p-repq: p=repc q and q-in-lla: q ∈ set (levellista ! k)
by auto
from q-in-lla wf-lla nbslla have varqk: var q = k
by (simp add: wf-ll-def)
have Nodesnb-in-NodesSucnb: Nodes nb levellista ⊆ Nodes (Suc nb) levellista
by (rule Nodes-subset)
from q-in-lla ksnb have q ∈ Nodes nb levellista
by (auto simp add: Nodes-def)
with varrep-prop Nodesnb-in-NodesSucnb have var (repc q) <= var q
by auto
with varqk ksnb p-repq have var p < nb
by auto
with varx-varp have varx-snb: var x < nb
by auto
from x-in-t t-ta ta-in-repc-NodesSucnb obtain a where
x-repa: x = repc a and
a-in-NodesSucnb: a ∈ Nodes (Suc nb) levellista
by auto
with varrep-prop have rx-x: repc x = x
by auto
have x ∈ set-of pret
proof –
from wf-lla nbslla have Nodes (Suc nb) levellista ⊆ set-of pret
apply –
apply (rule Nodes-in-pret)
apply auto
done
with x-in-t t-ta ta-in-repc-NodesSucnb repcNodes-in-Nodes show ?thesis
by auto
qed
with wf-lla have x ∈ set (levellista ! (var x))
by (auto simp add: wf-ll-def)
with varx-snb have x ∈ Nodes nb levellista
by (auto simp add: Nodes-def)
with rx-x show x ∈ repc ‘Nodes nb levellista
apply –
apply rule
apply (subgoal-tac x=repc x)
apply auto
done
qed
with ta-nTip ta-dag t-ta show ?thesis
apply –
apply (rule DagsI)
apply auto
done
qed
with t-notin-DagsNodesnb show ?thesis
by auto
qed
qed

lemma same-bdt-var: \[ bdt (Node lt1 p1 rt1) \var = Some bdt1; bdt (Node lt2 p2 rt2) \var = Some bdt1] \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 lbdt v rbdt) 
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] 
\implies \var p \leq 1 
proof (induct t)
case Tip 
thus ?case by simp
next
case (Node lt p rt)
note \( \text{tNode} = \text{this} \)
from Node.prems tNode show \(?\text{case}\)
  apply (simp add: isLeaf-pt-def)
  apply (case-tac var \( p = 0 \))
  apply simp
  apply (case-tac var \( p = \text{Suc} \ 0 \))
  apply simp
  apply simp
  done
qed

lemma subnode-dag-cons:
\( \forall \ p. \ [\text{Dag} \ p \ \text{low} \ \text{high} \ t; \ \text{no} \ \in \ \text{set-of} \ t] \implies \exists \ \text{not}. \ \text{Dag} \ \text{no} \ \text{low} \ \text{high} \ \text{not} \)
proof (induct \( t \))
  case Tip
  thus \(?\text{case}\) by simp
next
  case (Node \( \text{lt} \ q \ \text{rt} \))
  with Node.prems have \( q-p: p = q \)
    by simp
  from Node.prems have \( \text{lt-dag}: \ \text{Dag} (\text{low} p) \ \text{low} \ \text{high} \ \text{lt} \)
    by auto
  from Node.prems have \( \text{rt-dag}: \ \text{Dag} (\text{high} p) \ \text{low} \ \text{high} \ \text{rt} \)
    by auto
  show \(?\text{case}\)
    proof (cases \( \text{no} \ \in \ \text{set-of} \ \text{lt} \))
      case True
      with Node.hyps \( \text{lt-dag}\) show \(?\text{thesis}\)
        by simp
    next
      assume \( \text{no-notin-lt}: \ \text{no} \ \notin \ \text{set-of} \ \text{lt} \)
      show \(?\text{thesis}\)
        proof (cases \( \text{no} = p \))
          case True
          with Node.prems \( q-p\) show \(?\text{thesis}\)
            by auto
        next
          assume \( \text{no-neq-p}: \ \text{no} \neq p \)
          with Node.prems \( \text{no-notin-lt}\) have \( \text{no-in-rt}: \ \text{no} \ \in \ \text{set-of} \ \text{rt} \)
            by simp
          with \( \text{rt-dag} \) Node.hyps show \(?\text{thesis}\)
            by auto
      qed
    qed
qed

47
lemma nodes-in-taken-in-takeSuc: \( no \in set (take \ n \ \text{nodelist}) \implies no \in set (take \ (Suc \ n) \ \text{nodelist}) \)
proof –
  assume no-in-taken: \( no \in set (take \ n \ \text{nodelist}) \)
  have set (take n nodeslist) \( \subseteq \) 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: \( \forall \ n \ k. [n < k; \ n < \text{length} \ \text{xs}] \)
  \( \implies \) \( \text{xs} ! n \in set (take \ k \ \text{xs}) \)
apply (induct \text{xs})
apply simp
apply simp
apply (case-tac \ n)
apply simp
apply (case-tac \ k)
apply simp
apply simp
apply simp
apply simp
apply simp
apply simp
apply simp
done

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

lemma repNodes-eq-ext-rep: [low no \( \neq \) nodeslist! n; high no \( \neq \) nodeslist ! n;
low sn \( \neq \) nodeslist ! n; high sn \( \neq \) nodeslist ! n]
\( \implies \) repNodes-eq sn no low high repa = repNodes-eq sn no low high (repa(nodeslist ! n := repa (low (nodeslist ! n))))
by (simp add: repNodes-eq-def null-comp-def)
lemma filter-not-empty: \[ x \in \text{set} \, xs; \, P \, x \] \implies \, \text{filter} \, P \, xs \neq [] \\
by (induct \, xs) \, \text{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) \, \text{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) \, \text{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 \text{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: \\
\[ \forall \, x. \, P \, x \, x; \forall \, a \, b. \, P \, x \, a \longrightarrow P \, a \, b \longrightarrow P \, x \, b; \, \text{filter} \, (P \, x) \, xs \neq [] \] \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 [[simp-depth-limit=2]] \\
apply (simp) \\
applyclarsimp \\
apply (fastforce dest: hd-filter-prop) \\
done

lemma take-Suc-not-last: \\
\forall \, n. \, \exists \, i < \text{length} \, xs. \, x \neq xs \, !i; \, n < \text{length} \, xs \] \implies \, x \in \text{set} \, (\text{take} \, n \, 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: ∀ x ∈ set xs. P x = Q x → filter P xs = filter Q xs
  apply (induct xs)
  apply auto
  done

lemma hd-filter-take-more: ∀ n m. [filter P (take n xs) ≠ []; n ≤ m] →
  hd (filter P (take n xs)) = hd (filter P (take m 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 ⇒ nat
  low′:: ref ⇒ ref
  high′:: ref ⇒ ref
  rep′:: ref ⇒ ref
  mark′:: ref ⇒ bool
  next′:: ref ⇒ 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

50
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-p t 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 | ) =
\begin{verbatim}

node := nodeslist;
WHILE (node \neq Null) DO
  IF (\neg isLeaf-p't node' low \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

procedures
Repoint (p|p) =
IF (p \neq Null )
  THEN
    p := p \rightarrow rep;
    IF (p \neq Null )
      THEN p \rightarrow low := CALL Repoint (p \rightarrow low);
      THEN p \rightarrow high := CALL Repoint (p \rightarrow high)
    FI
  FI
FI

procedures
Normalize (p|p) =
  levellist := replicate (p\rightarrow var +1) Null;;
  levellist := CALL Levellist (p, (\neg p\rightarrow 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\} \ PROC Eval ('p, 'varval, 'R)
           \{t. t may-not-modify-globals \sigma\}
apply (hoare-rule HoarePartial.ProcRec1)
apply (vfg spec=modifies)
done

lemma (in Eval-impl) Eval-spec:

\end{verbatim}
shows $\forall \sigma \ t \ bdt1. \ \Gamma \vdash$

\begin{align*}
\{\sigma. \ Dag \ 'p \ 'low \ 'high \ t \land bdt \ t \ 'var = Some \ bdt1\} \\
'R := \ PROC \ Eval(\ 'p, \ 'varval) \\
\{ 'R = eval \ bdt1 \ \sigma'varval \}
\end{align*}

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 fastforce
apply (case-tac bdt1)
apply fastforce
apply fastforce
apply fastforce
apply fastforce
apply fastforce
apply fastforce
apply fastforce
done

end

6 Proof of Procedure Levellist

theory LevellistProof imports ProcedureSpecs ../Simpl/HeapList begin


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

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

53
lemma \textit{Dag-RefD}: 
\[\text{Dag } p \ l \ r \ t; \ p \neq \text{Null} \Rightarrow \exists lt \ rt. \ t = \text{Node} \ lt \ p \ rt \land \text{Dag} \ (l \ p) \ l \ rt \land \text{Dag} \ (r \ p) \ l \ rt\]
by simp

lemma \textit{Dag-unique-ex-conjI}: 
\[\text{Dag } p \ l \ r \ t; \ P t \Rightarrow (\exists t. \text{Dag} \ p \ l \ r \ t \land P t)\]
by simp

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

lemma \textit{list-ext}: 
\[
\forall \ys. \text{length } xs = \text{length } ys; \forall i < \text{length } xs. \text{xs}!i=\ys!i \Rightarrow xs = ys
\]
apply (induct \(xs\))
apply simp
apply (case_tac \(ys\))
apply simp
apply force
done

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

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

definition \textit{Levellist:: ref list \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref list list) \Rightarrow bool where}
Levellist \(hds\) next \(ll\) \(\leftrightarrow\) (map first \(ll\) = \(hds\)) \land
(\(\forall i < \text{length } hds. \text{List} (hds \downarrow i) \text{ next } (\text{ll}!i)\))

lemma \textit{Levellist-unique}: 
assumes \(ll\): Levellist \(hds\) next \(ll\)
assumes \(ll'\): 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 \text{leq: length } ll = \text{length } ll' by simp
show \(\text{thesis}\)
proof (rule list-ext [OF \text{leq, rule-format}])
fix \(i\)

54
assume  \( i < \text{length } ll \)
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

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

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

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

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

55
assume $n \in \text{set} (\text{concat } ll')$
from \text{in-set-concat-idx [OF this]}
\begin{align*}
\text{obtain } i \text{ where } i\text{-bound}: i < \text{length } ll' & \text{ and } n\text{-in}: n \in \text{set} (ll' ! i) \\
\text{by } \text{blast} \\
\text{have } n \in \text{set} (\text{concat } ll) & \cup \text{set-of } t \\
\end{align*}
\begin{proof} (cases $i \leq \text{var } p$)
\begin{enumerate}
\item case $\text{True}$
\begin{align*}
\text{with } \text{wf-ll obtain } prx \text{ where } \\
ll'\text{-ll}: ll' ! i = prx \oplus ll ! i & \text{ and } \\
prx: \forall pt \in \text{set } prx. pt \in \text{set-of } t & \text{ and } \\
\text{leq: length } ll' = \text{length } ll \\
\text{apply } (\text{clarsimp simp add: } \text{wf-levellist-def } \text{Node}) \\
\text{apply } (\text{erule-tac } x=i \text{ in allE}) \\
\text{apply clarsimp} \\
\text{done}
\end{align*}
\end{enumerate}
\begin{enumerate}
\item case $\text{False}$
\begin{align*}
\text{with } \text{n-in } ll'\text{-ll} \\
\text{have } n \in \text{set } (ll ! i) & \text{ by simp} \\
\text{with } \text{i-bound leq} \\
\text{have } n \in \text{set } (\text{concat } ll) & \text{ by auto} \\
\text{thus } ?\text{thesis by simp}
\end{align*}
\end{enumerate}
\begin{enumerate}
\item case $\text{False}$
\begin{align*}
\text{with } \text{wf-ll obtain } ll' ! i = ll ! i & \text{length } ll' = \text{length } ll \\
\text{by } (\text{auto simp add: } \text{wf-levellist-def } \text{Node}) \\
\text{with } \text{n-in } i\text{-bound} \\
\text{have } n \in \text{set } (\text{concat } ll) & \text{ by auto} \\
\text{thus } ?\text{thesis by simp}
\end{align*}
\end{enumerate}
\end{proof}
}\} \\
\text{thus } ?\text{thesis by auto}
\text{qed}
\text{qed}
\text{qed}
\text{qed}
lemma Levellist-ext-to-all: \((\exists \mathbf{ll}. \text{Levellist hds next ll } \land P \mathbf{ll}) \rightarrow Q)\)

\[=\]
\((\forall \mathbf{ll}. \text{Levellist hds next ll } \land P \mathbf{ll} \rightarrow Q)\)

apply blast

done

lemma Levellist-length: \(\text{Levellist hds p ll } \implies \text{length ll } = \text{length hds}\)

by (auto simp add: Levellist-def)

lemma map-update:

\[\forall i. \ i < \text{length xss } \implies \text{map } f \ (\text{xss}[i := \mathbf{xs}]) = (\text{map } f \text{xss}) \ [i := f \mathbf{xs}]\]

apply (induct xss)

apply simp

apply (case-tac i)

apply simp

apply simp

apply simp

done

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

shows \(\forall \mathbf{ll} \sigma t. \Gamma , \Theta \vdash t \{\sigma. \text{Dag 'p 'low 'high t } \land (\text{'p } \neq \text{Null } \implies (\text{'p } \rightarrow \text{'var}) < \text{length 'levellist}) \land \text{ordered 'var } \land \text{Levellist 'levellist 'next ll } \land \)

\((\forall n \in \text{set-of } t). \)

\((\text{if 'mark n } = \text{'m})\)

\(\text{then } n \in \text{set (ll ! 'var n)} \land \)

\((\forall nt p. \text{Dag n 'low 'high nt } \land p \in \text{set-of nt})\)

\(\rightarrow \text{'mark p } = \text{'m})\)

\(\text{else } n \notin \text{set (concat ll)})\)

\(\text{'levellist } := \text{PROC Levellist ('p, 'm, 'levellist)}\)

\[\{\exists \mathbf{ll}'. \text{Levellist 'levellist 'next ll'}' \land \text{wf-levellist t ll ll'} \sigma_{\text{var}} \land \text{wf-marking t } \sigma_{\text{mark}} \text{'mark } \sigma_{\text{m}} \land \)

\((\forall p. p \notin \text{set-of } t) \rightarrow \sigma_{\text{next p }} = \text{'next p})\}\]

apply (hoare-rule HoareTotal.ProcRec1

[where \tau=\text{measure (\lambda(s,p). size (dag s p 'low 'high))}])

apply vcg

apply (rule conjI)

apply clarify

apply (rule conjI)

apply clarify

apply (clarsimp simp del: BinDag.set-of.simps split del: split-if)

defer

apply (rule impl)

apply (clarsimp simp del: BinDag.set-of.simps split del: split-if)

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 lt have size-rt-dec: size (dag (high p) low high) < size (dag p low high)
  by (simp only: Dag-dag) simp
from Dag-p rt 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 varsll’: var p < length levellist
with ll have varsll: 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) ∧
          (∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → marka p = m)
        else n ∉ set (concat lla)) ∧
      size (dag (high p) low high) < size (dag p low high) ∧

58
∀ markb nextb levellist llb.
  Levellist levellist nextb llb ∧
  wf-levelist rt lla llb var ∧
  wf-marking rt marka markb m ∧
  (∀ p. p ∈ set-of rt ⇒ nexta p = nextb p) →
  (∃ ll'. Levellist (levellist(var p := p))
    (nextb(p := levellist! var p)) ll' ∧
    wf-levelist (Node lt p rt) ll ll' var ∧
    wf-marking (Node lt p rt) mark (markb(p := m)) m ∧
    (∀ pa. pa /∈ set-of (Node lt p rt) →
      next pa =
      (if pa = p then levellist! var p
       else nextb pa)))

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
proof (clarsimp)
case (goal1 marka nexta levellista lla markb nextb levellistb llb)
  have lla: Levellist levellista nexta lla by fact
  have llb: Levellist levellistb nextb llb by fact
  have wfl-lt: wf-levelist Tip lt lla var
    wf-marking Tip mark marka m by fact+
  then have l-lla: ll = lla
    by (simp add: wf-levelist-def)
  moreover
    with wfl-lt lt-Tip lt have marka = mark
    by (simp add: wf-marking-def)
  moreover
    have wfl-rt: wf-levelist Tip lla llb var
      wf-marking Tip marka markb m by fact+
    then have lla-llb: lla = llb
      by (simp add: wf-levelist-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 \( (\text{clarsimp simp add: } \text{Levellist-def map-update}) \)
apply \( (\text{erule-tac } x = i \text{ in allE}) \)
apply \( \text{clarsimp} \)
apply \( (\text{subgoal-tac } p \notin \text{set (llb } ! i )) \)
pref 2
using \( \text{p-notin-l} \text{ lll-a } \text{ lla-llb} \)
apply \( \text{simp} \)
apply \( \text{clarsimp } i = \text{var } p \)
apply \( \text{simp} \)
done
ultimately
show \( \text{?case} \)
using \( \text{lt-Tip Tip varsll} \)
apply \( (\text{clarsimp simp add: } \text{wf-levellist-def } \text{wf-marking-def}) \)
proof –
fix i
assume varsllb: \( \text{var } p < \text{length llb} \)
assume i \( \leq \) \( \text{var } p \)
show \( \exists \text{prx. llb} [\text{var } p := p \# \text{llb}! \text{var } p]! i = \text{prx} @ \text{llb}! i \wedge \) \((\forall \text{pt} \in \text{set prx. pt} = p \wedge \text{var pt} = i)\)
proof (cases \( i = \text{var } p \))
case True
with pnN lt rt varsllb lt-Tip Tip show \( \text{?thesis} \)
apply –
apply \( (\text{rule-tac } x = [p] \text{ in exI}) \)
apply \( (\text{simp add: subdag-eq-def}) \)
done
next
assume i \( \neq \) \( \text{var } p \)
with varsllb show \( \text{?thesis} \)
apply –
apply \( (\text{rule-tac } x = [] \text{ in exI}) \)
apply \( (\text{simp add: subdag-eq-def}) \)
done
qed
qed
qed
next
case (Node dag1 a dag2)
have rt-node: \( \text{rt} = \text{Node dag1 a dag2} \) by fact
with \( \text{rt} \) have high-p: \( \text{high } p = a \)
by simp
have s: \( \bigwedge \text{nexta. } (\forall \text{p. next } p = \text{nexta } p) = (\text{next} = \text{nexta}) \)
by auto
show \( \text{?thesis} \)
using size-rt-dec size-lt-dec rt-node lt-Tip Tip lt rt
apply \( (\text{clarsimp simp del: set-of-Node split del: split-if simp add: s}) \)
proof –
case (goal1 marka levellista lla)
have lla: Levellist levellista next lla by fact
have wfll-ll: 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 wfll-ll lt-Tip lt have marklrec: marka = mark
  by (simp add: wf-marking-def)
from orderedt varsll 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) \land
    (\forall nt. Dag n low high nt \land p \in \text{set-of} nt \rightarrow marka p = m)
  else n /\in \text{set} (\text{concat} lla))
apply (simp only: BinDag.set-of.simps)
apply clarify
apply (drule-tac x=n in bspec)
apply blast
apply assumption
done
show ?case
apply (rule conjI)
apply (rule var-highp-bound)
apply (rule conjI)
apply (rule ordered-rt)
apply (rule conjI)
apply (rule mark-rt)
apply clarify
apply clarsimp
proof —
case (goal1 markb nextb levellistb llb)
have llb: Levellist levellistb nextb llb by fact
have wfll-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 wfll-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-lia wfll-rt
have p-notin-llb: \forall i < length llb. p \notin \text{set} (llb ! i)
  apply —
  apply (intro allI impI)
apply (clarsimp simp add: wf-levellist-def)
apply (erule-tac x=i in spec)
using orderedt rt-node lt-Tip high-p
apply clarsimp
apply (erule-tac x=i in spec)
apply clarsimp
apply clarsimp
apply clarsimp
done

with lb pnN var-p-bounds
have lhc: Levelist (levellistb[var p := p])
  (nextb(p := levellistb ! var p))
  (lb[var p := p ≠ lb ! 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 ?case
apply simp
using wfl-rt wflmarking-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 (erule exI)
apply (intro conjI)
apply simp

62
apply clarify
apply (rotate-tac 15)
apply (erule-tac x=pt in ballE)
apply fastforce
apply fastforce
done
qed
qed
qed
next
case (Node ll l rt)
  have lt-Node: l = Node ll l rt by fact
  from orderedl l varsll' lt-Node
  obtain ordered-l:
    ordered lt var (low p \neq \text{Null} \rightarrow \text{var} (\text{low} p) < \text{length levellist})
    by (cases rt) auto
  from lt lt-Node marked-child-l
  have mark-l: \forall n \in \text{set-of} \text{lt}.
    if mark n = m then n \in \text{set} (\text{concat} ll)
  else n \notin \text{set} (\text{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: split-if)
apply (simp only: Levellist-ext-to-all)
proof
  case (goal1 marka nexta levellista lla)
  have lla: Levellist levellista nexta lla by fact
  have wfl-lt: wfl-levellist lt ll lla var by fact
  have wfnmarking-lt: wfn-marking lt mark marka m by fact
  from wfl-lt lt-Node
  have lla-eq-l: length lla = length ll
    by (simp add: wfl-levellist-def)
  with ll lla have 'lla-eq-l': length levellista = length levellist
    by (simp add: Levellist-length)
  with ordered-lt lt-Node lt varsll' 
  obtain ordered-r:
    ordered lt var (high p \neq \text{Null} \rightarrow \text{var} (\text{high} p) < \text{length levellista})
    by (cases rt) auto
  from wfl-lt lt-Node
  have nodes-in-lla: \forall q. q \in \text{set-of} \text{lt} \rightarrow q \in \text{set} (lla ! (q\rightarrow\text{var}))
by (simp add: wf-levellist-def)
from wfll-lt lt-Node lt
have lla-st: \( \forall i \leq (\text{low } p) \rightarrow \text{var}. \)
  \[ (\exists \text{prx. } (\text{lla } i) = \text{prx} \circ (\text{ll } i) \wedge \]
  \[ (\forall \text{pt } \in \text{set prx. } \text{pt } \in \text{set-of } \text{lt } \wedge \text{pt} \rightarrow \text{var } = i)) \]
  by (simp add: wf-levellist-def)
from wfll-lt lt-Node lt
have lla-nc: \( \forall i. ((\text{low } p) \rightarrow \text{var}) < i \rightarrow (\text{lla } i) = (\text{ll } i) \)
  by (simp add: wf-levellist-def)
from wfmarking-lt lt-Node lt
have mot-nc: \( \forall n. n \notin \text{set-of } \text{lt } \rightarrow \text{mark } n = \text{marka } n \)
  by (simp add: wf-marking-def)
from wfmarking-lt lt-Node lt
have mit-marked: \( \forall n. n \in \text{set-of } \text{lt } \rightarrow \text{marka } n = m \)
  by (simp add: wf-marking-def)
from marked-child-ll nodes-in-lla mot-nc mit-marked lla-st
have mark-rt: \( \forall n \in \text{set-of } \text{rt}. \)
  if marka \( n = m \)
  then \( n \in \text{set } (\text{lla } \text{var } n) \) \
  \[ (\forall \text{nt } p. \text{Dag } n \text{ low high } \text{nt } \wedge \text{p } \in \text{set-of } \text{nt } \rightarrow \text{marka } p = m) \]
  else \( n \notin \text{set } (\text{concat ll}) \)
apply 
apply (rule ballI)
apply (drule_tac x=n in bspec)
apply (simp)
proof 
  fix \( n \)

assume nodes-in-lla: \( \forall q. q \in \text{set-of } \text{lt } \rightarrow q \in \text{set } (\text{lla } \text{var } q) \)
assume mot-nc: \( \forall n. n \notin \text{set-of } \text{lt } \rightarrow \text{mark } n = \text{marka } n \)
assume mit-marked: \( \forall n. n \in \text{set-of } \text{lt } \rightarrow \text{marka } n = m \)
assume marked-child-ll: if marka \( n = m \)
  then \( n \in \text{set } (\text{ll } \text{var } n) \) \
  \[ (\forall \text{nt } p. \text{Dag } n \text{ low high } \text{nt } \wedge \text{p } \in \text{set-of } \text{nt } \rightarrow \text{marka } p = m) \]
  else \( n \notin \text{set } (\text{concat ll}) \)
assume lla-st: \( \forall i \leq \text{var } (\text{low } p). \)
  \[ \exists \text{prx. } (\text{lla } i) = \text{prx} \circ (\text{ll } i) \wedge \]
  \[ (\forall \text{pt } \in \text{set prx. } \text{pt } \in \text{set-of } \text{lt } \wedge \text{var } \text{pt } = i) \]
assume n-in-rt: \( n \in \text{set-of } \text{rt} \)
show n-in-lla-marked: if marka \( n = m \)
  then \( n \in \text{set } (\text{lla } \text{var } n) \) \
  \[ (\forall \text{nt } p. \text{Dag } n \text{ low high } \text{nt } \wedge \text{p } \in \text{set-of } \text{nt } \rightarrow \text{marka } p = m) \]
  else \( n \notin \text{set } (\text{concat ll}) \)
proof (cases \( n \in \text{set-of } \text{lt} \))
  case True
  from True nodes-in-lla have n-in-ll: \( n \in \text{set } (\text{lla } \text{var } n) \)
  by simp

64
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: set-of nt ⊆ set-of lt
    by (rule dag-setof-subsetD)
  moreover assume p ∈ set-of nt
  ultimately have p ∈ 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 /∈ set-of lt
show ?thesis
proof (cases marka n = m)
  case True
  from n-notin-lt mot-nc have marka-eq-mark: mark n = 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 ≠ 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 ∈ set (ll ! var n)
  by fastforce
from marked-child-ll n-in-rt marka-eq-mark nnN n-marked lt rt
have nt-mark: ∀ nt p. Dag n low high nt ∧ p ∈ set-of nt → mark p = m
  by simp
from nodes-in-lla n-in-lla lla-st
have n-in-lla: n ∈ set (lla ! var n)
proof (cases var (low p) < (var n))
  case True
  with lla-nc have (lla ! var n) = (ll ! var n)
    by fastforce

65
with n-in-ll show thesis
  by fastforce
next
  assume varsnap: ¬ var (low p) < var n
  with ll-tll
  have ll-in-tll: ∃ prx. ll ! (var n) = prx @ ll ! (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 ∈ set-of nt
  have marka pt = m
  proof (cases pt ∈ set-of lt)
    case True
      with mit-marked show thesis
      by fastforce
    next
      assume pt-notin-lt: pt /∈ 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:
  ∀ nt pt. Dag n low high nt ∧ pt ∈ set-of nt → marka pt = m
  by fastforce
with n-in-lla nt-marka True show thesis
  by fastforce
next
  note n-not-marka = this
with wmarking-nil n-notin-nil
  have mark n ≠ m
    by (simp add: wmarking-def lt-Node)
with marked-child-nil
  have n-notin-nil: n /∈ set (concat ll)
    by simp
  show thesis
proof (cases n ∈ set (concat llla))
  case False with n-not-marka show thesis by simp
next
  case True
with wf-levellist-subset [OF wfll-lt] n-notin-ll
have n ∈ set-of lt
  by blast
with n-notin-lt have False by simp
thus ?thesis ..
qed
qed
qed

show ?case
apply (intro conj1 ordered-rt mark-rt)
apply clarify

proof (cases rt)
case Tip
  from wfll-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 wfll-lt 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 wfll-lt
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 (erule-tac x = i in spec)
  using orderedt Tip lt-Node
  apply clarsimp
  apply (erule-tac x = i in spec)
  apply (erule-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 varsil orderedt lt rt pnN afll-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=\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=\in ballE)
apply fastforce
apply fastforce
done

next
  case (Node lrt r rrt)
  have rt-Node: rt = Node lrt r rrt by fact
  from afll-rt rt-Node
  have llb-eq-lla: length llb = length lla
    by (simp add: wf-levellist-def)
  with llb lla
  have llb-eq-lia: length levellistb = length levellista
    by (simp add: Levellist-length)
  from afll-rt rt-Node
  have nodes-in-llb: \forall q. q \in set-of rt \rightarrow q \in set (llb ! (q\rightarrow var))
by (simp add: wf-levellist-def)
from wfll-rt rt-Node rt
have llb-st: (\forall i \leq (high p)\rightarrow var.
  (\exists prx. (llb ! i) = prx@(lla ! i) \land
   (\forall pt \in \text{set prx}. pt \in \text{set-of rt} \land pt \rightarrow \text{var} = i)))
  by (simp add: wf-levellist-def)
from wfll-rt rt-Node rt
have llb-nc: (\forall i. ((high p)\rightarrow \text{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 wfll-rt wfll-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 \leq \text{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 \leq \text{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 \leq \text{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 wfll-lt wfll-rt varsll lla llb
obtain var-p-bounds: var p < \text{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[\text{var p}]))
  (llb[var p := p \neq llb[\text{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-levelist (Node lt p rt) ll (llb[\text{var } p := \text{p} # llb ! \text{var } p]) \text{var} \land
    wf-marking (Node lt p rt) mark (markb(p := m)) m
proof –
  have nodes-in-upllb: \forall q. q \in \text{set-of} (Node lt p rt)
    \rightarrow q \in \text{set} (llb[\text{var } p := \text{p} # llb ! \text{var } p] ! (\text{var } q))
    apply –
    apply (rule allI)
    apply (rule impI)
proof –
  fix q
  assume q-in-t: q \in \text{set-of} (Node lt p rt)
  show q-in-upllb:
    q \in \text{set} (llb[\text{var } p := \text{p} # llb ! \text{var } p] ! (\text{var } q))
    proof (cases q \in \text{set-of rt})
      case True
      with nodes-in-llb have q-in-llb: q \in \text{set} (llb ! (\text{var } q))
        by fastforce
      from orderedrt rt-Node lt-Node lt rt
      have ordered-rt: ordered rt var
        by fastforce
      from True rt ordered-rt rt-Node lt-Node have var q \leq var r
        apply –
        apply (erule subnodes-ordered)
        apply fastforce
        apply fastforce
        apply fastforce
        done
      with orderedrt rt lt rt-Node lt-Node have var q < var p
        by fastforce
      then have
        llb[\text{var } p := \text{p} # llb ! \text{var } p] ! \text{var } q =
        llb ! \text{var } q
        by fastforce
      with q-in-llb show ?thesis
        by fastforce
  next
  assume q-notin-rt: q \notin \text{set-of} rt
  show q \in \text{set} (llb[\text{var } p := \text{p} # llb ! \text{var } p] ! \text{var } q)
    proof (cases q \in \text{set-of lt})
      case True
assume \( q \text{-in-}lt \): \( q \in \text{set-of} \ lt \)
with nodes-in-lla have \( q \text{-in-lla} \): \( q \in \text{set} (lla ! (\text{var} \ q)) \)
by fastforce

from orderedlt rt-Node lt-Node lt rt
have ordered-lt: ordered \ lt \ var
by fastforce

from \( q \text{-in-}lt \) \( lt \) ordered-lt rt-Node rt lt-Node
have \( \text{var} \ q \leq \text{var} \ l \)
apply \( - \)
apply (drule subnodes-ordered)
apply fastforce
apply fastforce
apply fastforce
done

with orderedlt rt lt rt-Node lt-Node have \( qsp \): \( \text{var} \ q < \text{var} \ p \)
by fastforce

then show \(?\text{thesis}\)
proof (cases \( \text{var} \ q \leq \text{var} \ (\text{high} \ p) \))
case True
with llb-st
have \( \exists \text{prx}. \ (llb ! (\text{var} \ q)) = \text{prx} \circ (lla ! (\text{var} \ q)) \)
by fastforce

with nodes-in-lla q-in-lla
have \( q \text{-in-llb} \): \( q \in \text{set} (llb ! (\text{var} \ q)) \)
by fastforce

from \( qsp \)
have \( llb[\text{var} \ p := p \# llb ! (\text{var} \ p)! \text{var} \ q = llb ! (\text{var} \ q) \)
by fastforce

with \( q \text{-in-llb} \) show \(?\text{thesis}\)
by fastforce

next
assume \( \neg \text{var} \ q \leq \text{var} \ (\text{high} \ p) \)
with llb-nc have \( llb ! (\text{var} \ q) = lla ! (\text{var} \ q) \)
by fastforce

with q-in-lla have \( q \text{-in-lla} \): \( q \in \text{set} (lla ! (\text{var} \ q)) \)
by fastforce

from \( qsp \) have
\( llb[\text{var} \ p := p \# llb ! (\text{var} \ p)! \text{var} \ q = llb ! (\text{var} \ q) \)
by fastforce

with \( q \text{-in-llb} \) show \(?\text{thesis}\)
by fastforce

qed

next
assume \( q \text{-notin-lt} \): \( q \notin \text{set-of} \ lt \)
with q-notin-rt rt lt rt-Node lt-Node q-in-t have \( qp \): \( q = p \)
by fastforce

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

72
by fastforce

with inp True vpsi vars ll 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: ¬ var (high p) < i
from iselp lla-st
have prxl: \exists\text{prx}. lla! i = prx @ ll ! i ∧
  (\forall pt \in \text{set prx}. pt \in \text{set-of lt} ∧ \text{var pt} = i)
by fastforce
from isehp llb-st
have prxh: \exists\text{prx}. llb! i = prx @ lla! i ∧
  (\forall pt \in \text{set prx}. pt \in \text{set-of rt} ∧ \text{var pt} = i)
by fastforce
with prxl inp lt pnN rt show \text{?thesis}
  apply –
  apply (elim exE)
  apply (rule-tac \text{x=prxa @ prx in exI})
  apply simp
  apply (elim conjE)
  apply fastforce
done
qed

have big-Nodes-nc: \forall i. (p->\text{var}) < i
  \rightarrow (llb[\text{var p :=p # llb ! var p}]) ! i = 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 (low p) < i}
by fastforce
with lla-nc have lla-lt: lla! i = ll! i
by fastforce
from psi orderedt lt rt lt-Node rt-Node have hpsi: \text{var (high p) < i}
by fastforce
with llb-nc have llb-lla: llb! i = lla! i
by fastforce
from psi have upllb-llb: llb[\text{var p :=p # llb ! var p}]! i = llb!i
by fastforce
from upllb-llb llb-lla lla-ll
show llb[\text{var p :=p # llb ! var p}]! i = ll ! i
by fastforce
qed
from lla-eq-ll llb-eq-lla
have length-eq: length (llb[\text{var p :=p # llb ! var p}]) = length ll
by fastforce
from length-eq big-Nodes-nc prx-ll-st nodes-in-upllb
have wfl-ll-upllb: wfl-levellist (Node lt p rt ll (llb[\text{var p :=p # llb ! var p}]) var
by (simp add: wf-levelist-def)

have mark-nc:
     \forall n. n \notin \text{set-of} (\text{Node} \; \text{lt} \; p \; \text{rt}) \rightarrow (\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} \; \text{lt} \; p \; \text{rt})
with lt rt have nnilt: n \notin \text{set-of} \; \text{lt}
    by fastforce
from nnit lt rt have nnirt: n \notin \text{set-of} \; \text{rt}
    by fastforce
with nnilt mot-nc mort-nc have mb-eq-m: markb \; n = \text{mark} \; n
    by fastforce
from nnit have n\neq p
    by fastforce
then have upmarkb-markb: (markb(p:=m)) \; n = \text{markb} \; n
    by fastforce
with mb-eq-m show (markb(p:=m)) \; n = \text{mark} \; n
    by fastforce
qed

have mark-c: \forall n. n \in \text{set-of} (\text{Node} \; \text{lt} \; p \; \text{rt}) \rightarrow (\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} \; \text{lt} \; p \; \text{rt})
show (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 \in \text{set-of} \; \text{rt})
  case True
    with mirt-marked have markb \; n = m
    by fastforce
    with nnp show ?thesis
    by fastforce
next
assume nnirt: n \notin \text{set-of} \; \text{rt}
with nint nnp have nnilt: n \in \text{set-of} \; \text{lt}
    by fastforce
with mit-marked have marka-m: marka \; n = m
by fastforce
from mort-nc nninrt have marka n = markb n
by fastforce
with marka-m have markb n = m
by fastforce
with mnp 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
qed
qed
qed

next
fix var low high p lt rt and levellist and
lt::ref list list and mark::ref ⇒ bool and next
assume pnN: p ≠ Null
assume ll: Levellist levellist next ll
assume vpsll: var p < length levellist
assume orderedt: ordered (Node lt p rt) var
assume marked-child-ll: ∀ n∈set-of (Node lt p rt).
  if mark n = mark p
  then n ∈ set (ll ! var n) ∧
  (∀ nt pa. Dag n low high nt ∧ pa ∈ set-of nt → mark pa = mark p)
  else n /∈ set (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 ∧
  wf-marking (Node lt p rt) mark mark (mark p)
proof －
from marked-child-ll pnN lt rt have marked-st:
  (∀ pa. pa ∈ set-of (Node lt p rt) → mark pa = mark p)
  apply －
  apply (drule-tac x=p in bspec)
  apply simp
  apply (clarsimp)
  apply (erule-tac x=(Node lt p rt) in allE)
  apply simp
done
have nodest-in-ll:
  ∀ q. q ∈ set-of (Node lt p rt) → q ∈ set (ll ! var q)
proof
from marked-child-ll pnN have pinll: \( p \in \text{set } (ll \downarrow \text{var } p) \)
  apply -
  apply (drule-tac \( x=p \) in bspec)
  apply simp
  apply fastforce
done
from marked-st marked-child-ll lt rt show \(?\text{thesis}\)
  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: \( \forall \, i \leq \text{var } p, (\exists \text{prx. } ll ! i = \text{prx}@(ll ! i)) \land \\
(\forall \, pt \in \text{set prx. } pt \in \text{set-of } (\text{Node } pt p \text{ rt}) \land \text{var } pt = i) \)
  apply -
  apply (rule allI)
  apply (rule impI)
  apply (rule-tac \( x=[] \) in exI)
  apply fastforce
done

have ll-nc: \( \forall \, i. \text{var } p < i \rightarrow ll ! i = ll ! i \)
  by fastforce

have length-ll: \( \text{length } ll = \text{length } ll \)
  by fastforce

with ll-nc levellist-nc nodest-in-ll
have wf: \( \text{wf-levellist } (\text{Node } lt p \text{ rt}) \text{ ll var } \)
  by (simp add: wf-levellist-def)

have m-nc: \( \forall \, n. n \notin \text{set-of } (\text{Node } pt p \text{ rt}) \rightarrow \text{mark } n = \text{mark } n \)
  by fastforce
from marked-st have \( \forall \, n. n \in \text{set-of } (\text{Node } pt p \text{ rt}) \rightarrow \text{mark } n = \text{mark } p \)
  by fastforce

with m-nc have \( \text{wf-marking } (\text{Node } pt p \text{ rt}) \text{ mark mark } (\text{mark } p) \)
  by (simp add: wf-marking-def)

with wf show \(?\text{thesis}\)
  by fastforce
qed

lemma allD: \( \forall \, ll. P \text{ ll } \rightarrow P \text{ ll } \)
  by blast

lemma replicate-spec: \( [\forall \, i < n. \text{xs} ! i = x; n=\text{length } \text{xs}] \)
⇒ replicate (length xs) x = xs
apply hypsubst-thin
apply (induct xs)
apply simp
apply force
done

lemma (in Levellist-impl) Levellist-spec-total:
shows ∀σ t. Γ,Θ ⊢ t{|σ. Dag `p `low `high t ∧ (∀i < length `levellist. `levellist ! i = Null) ∧
length `levellist = `p → `var + 1 ∧
ordered t `var ∧ (∀n ∈ set-of t. `mark n = (¬ `m) )}}
`levellist ::= PROC Levellist (`p, `m, `levellist)
∃ll. Levellist `levellist `next ll ∧ wf-ll t ll `var ∧
length `levellist = σ_p → σ_var + 1 ∧
wf-marking t σ_mark `mark σ_m ∧
(∀p. p /∈ set-of t −→ σ_next p = `next p)}
apply (hoare-rule HoareTotal.conseq)
apply (rule-tac ll=replicate (σ_p→σ_var + 1) [] in allD [OF Levellist-spec-total])
apply (intro allI impI)
apply (rule-tac x=σ in exI)
apply (rule-tac x=t in exI)
apply (rule conjI)
apply (clarsimp)
apply (case-tac i)
apply simp
apply simp
apply (simp add: Collect-conv-if split-if-asm simp del: concat-replicate-trivial)
aply (clarsimp simp add: Levellist-def)
aply simp
apply simp
apply simp
apply (simp add: Collect-conv-if split-if-asm)
aply vcg-step
apply (elim exE conjE)
apply (rule-tac x=ll in exI)
aply simp
apply (thin-tac ∀p. p /∈ set-of t −→ next p = `nexta p)
aply (simp add: wf-levellist-def wf-ll-def)
aply (case-tac t = `Tip)
aply simp
apply (rule conjI)
aply clarsimp
apply (case-tac k)
aply simp
apply simp
apply (subgoal-tac length ll=Suc (var Null))
apply (simp add: Levellist-length)
aply fastforce
apply (split dag.splits)
aply simp

78
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 (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 ∀ σ. Γ ⊢ {σ} PROC ShareRep (nodeslist, p)
{t. t may-only-modify-globals σ in [rep]} 
apply (hoare-rule HoarePartial.ProcRec1)
apply (vec spec=modifies)
done

lemma hd-filter-cons:
∀ i. [ P (xs ! i) p; i < length xs; ∀ no ∈ set (take i xs). ¬ P no p; ∀ a b. P a b = P b a ]
⇒ xs ! i = hd (filter (P p) xs)
apply (induct xs)
apply simp

79
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
\( \forall \sigma \ ns. \ \Gamma, \Theta \vdash \)
\((\forall no \in set ns. \ \sigma \neq Null \wedge
((no \rightarrow 'low = Null) = (no \rightarrow 'high = Null)) \wedge
(isLeaf-pt \ 'p \ 'low \ 'high \rightarrow isLeaf-pt \ \sigma \ 'low \ 'high) \wedge
\quad no \rightarrow \ 'var = \ 'p \rightarrow \ 'var) \wedge
\quad 'p \in set ns)\)
PROC ShareRep ('nodeslist, 'p)
\[ (\sigma_p \rightarrow \ 'rep = \ hd (\ \lambda sn. \ \text{repNodes-eq} \ sn \ \sigma_p \ \sigma \ 'low \ \sigma \ 'high \ \sigma \ 'rep) \ ns)) \wedge
(\forall pt. \ pt \neq \sigma_p \rightarrow pt \rightarrow \sigma \ 'rep = pt \rightarrow \ 'rep) \wedge
(\sigma_p \rightarrow \ 'rep \rightarrow \sigma \ 'var = \sigma_p \rightarrow \sigma \ 'var)\]
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 \[ (\exists prx \ sfx. \ \text{List} \ \text{nodeslist} \ \text{next} \ ns=prx@sfx \wedge
\quad \neg \text{isLeaf-pt} \ 'p \ 'low \ 'high \wedge
\quad (\forall no \in set ns. \ \sigma \neq Null \wedge
\quad ((no \rightarrow \sigma \ 'low = Null) = (no \rightarrow \sigma \ 'high = Null)) \wedge
\quad (isLeaf-pt \ \sigma_p \ \sigma \ 'low \ \sigma \ 'high \rightarrow isLeaf-pt \ \sigma \ 'low \ \sigma \ 'high) \wedge
\quad \neg \text{nodeslist} \neq Null \wedge
\quad (\exists pt \in set prx. \ \neg \text{repNodes-eq} \ pt \ \sigma_p \ \sigma \ 'low \ \sigma \ 'high \ \sigma \ 'rep) \wedge
\quad \rightarrow \ 'rep \ \sigma_p = \ hd (\ \lambda sn. \ \text{repNodes-eq} \ sn \ \sigma_p \ \sigma \ 'low \ \sigma \ 'high \ \sigma \ 'rep) \ prx) \wedge
\quad (\forall pt. \ pt \neq \sigma_p \rightarrow pt \rightarrow \sigma \ 'rep = pt \rightarrow \sigma \ 'rep) \wedge
\quad (\neg \text{nodeslist} \neq Null \rightarrow
\quad ((\forall pt \in set prx. \ \neg \text{repNodes-eq} \ pt \ \sigma_p \ \sigma \ 'low \ \sigma \ 'high \ \sigma \ 'rep) 
\quad \rightarrow \sigma \ 'rep = \ 'rep) \wedge
\quad ('p = \sigma_p \wedge \ 'high = \sigma \ 'high \wedge \ 'low = \sigma \ 'low))\]
VAR MEASURE (length (list 'nodeslist 'next))
DO
IF (repNodes-eq 'nodeslist 'p 'low 'high 'rep)
THEN \ 'p \rightarrow \ 'rep ::= \ 'nodeslist ;; 'nodeslist ::= Null
ELSE 'nodeslist ::= 'nodeslist \rightarrow '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: ∀ no∈set ns.
   no ≠ Null ∧
   (low no = Null) = (high no = Null) ∧
   (isLeaf-pt p low high → isLeaf-pt no low high) ∧ var no = var p

81
assume p-in-ns: p ∈ set ns
assume p-Leaf: isLeaf-pt p low high
show nodeslist = hd [sn←ns . repNodes-eq sn p low high rep] ∧
  var nodeslist = var p

proof =
  from p-in-ns no-prop have p-not-Null: p≠Null
    using [[simp-depth-limit=2]]
    by auto
  from p-in-ns have ns ≠ []
    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≠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⇒nat and low high rep repa p prx sfx next
assume sfx: List Null next sfx
assume p-in-ns: p ∈ set (prx @ sfx)
assume no-props: ∀ no∈set (prx @ sfx).
  no ≠ Null ∧
  (low no = Null) = (high no = Null) ∧
  (isLeaf-pt p low high −→ isLeaf-pt no low high) ∧ var no = var p
assume match-prx: (∃ pt∈set prx. repNodes-eq pt p low high rep) −→
  repa p = hd [sn←prx @ sfx . repNodes-eq sn p low high rep] ∧
  (∀ pt. pt ≠ p −→ rep pt = repa pt)
show repa p = hd [sn←prx @ sfx . repNodes-eq sn p low high rep] ∧
  (∀ pt. pt ≠ p −→ rep pt = repa pt) ∧ var (repa p) = var p

proof =
  from sfx
  have sfx-Nil: sfx=[]
    by simp
  with p-in-ns have ex-match: (∃ pt∈set prx. repNodes-eq pt p low high rep)
    apply −
    apply (rule-tac x=p in bexI)
    apply (simp add: repNodes-eq-def)
    apply simp
    done
  hence not-empty: [sn←prx . repNodes-eq sn p low high rep] ≠ []
apply 
apply (erule bexE) 
apply (rule filter-not-empty) 
apply auto 
done
from ex-match match-prx obtain
  found: repa p = hd [sn←prx . repNodes-eq sn p low high rep] and
  unmodif: ∀ pt. pt ≠ p → rep pt = repa pt 
by blast
from hd-filter-in-list [OF not-empty] found
have repa p ∈ 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 ≠ Null 
assume p-no-Leaf: ¬ isLeaf-pt p low high 
assume no-props: ∀ no∈set prx ∪ set (nodeslist ≠ sfx).
  no ≠ Null ∧ (low no = Null) = (high no = Null) ∧ var no = var p 
assume p-in-ns: p ∈ set prx ∨ p ∈ set (nodeslist ≠ 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 nomatch-nodeslist: ¬ repNodes-eq nodeslist p low high repa 
assume sfx: List (next nodeslist) next sfx 
show (∀ no∈set prx ∪ set (nodeslist ≠ sfx).
  no ≠ Null ∧ (low no = Null) = (high no = Null) ∧ var no = var p) ∧
  ((∃ pt∈set (prx @ [nodeslist]). repNodes-eq pt p low high repa) →
  repa p = hd [sn←prx @ [nodeslist] . repNodes-eq sn p low high repa]) ∧
  (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

83
ultimately show \(?thesis
  using no-props
by (intro conjI)
qed

next

fix var low high p repa next nodeslist prx sfz
assume nodeslist-not-Null: nodeslist \neq Null
assume sfz: List nodeslist next sfz
assume p-not-Leaf: \neg \text{isLeaf-pt } p \text{ low high}
assume no-props: \forall no \in \text{set prx} \cup \text{set sfz}.
  no \neq Null \land
  (\text{isLeaf-pt } p \text{ low high} \implies \text{isLeaf-pt } no \text{ low high}) \land
  \text{var no } = \text{var p}
assume p-in-ns: p \in \text{set prx} \lor p \in \text{set sfz}
assume match-prx: (\exists pt \in \text{set prx}. \text{repNodes-eq pt } p \text{ low high repa}) \implies
  repa p = \text{hd } ([\text{sn} \leftarrow \text{prx}. \text{repNodes-eq sn } p \text{ low high repa}])
assume nomatch-prx: \forall pt \in \text{set prx}. \neg \text{repNodes-eq pt } p \text{ low high repa}
assume match: \text{repNodes-eq nodeslist } p \text{ low high repa}
show (\forall no \in \text{set prx} \cup \text{set sfz}.
  no \neq Null \land
  (\text{isLeaf-pt } p \text{ low high} \implies \text{isLeaf-pt } no \text{ low high}) \land
  (p \in \text{set prx} \lor p \in \text{set sfz}) \land
  ((\exists pt \in \text{set prx} \cup \text{set sfz}. \text{repNodes-eq pt } p \text{ low high repa}) \implies
    \text{nodeslist} =
    \text{hd } ([\text{sn} \leftarrow \text{prx}. \text{repNodes-eq sn } p \text{ low high repa}] @
    [\text{sn} \leftarrow \text{sfz}. \text{repNodes-eq sn } p \text{ low high repa}])) \land
  ((\forall pt \in \text{set prx} \cup \text{set sfz}. \neg \text{repNodes-eq pt } p \text{ low high repa}) \implies
    repa = \text{repa}(p := \text{nodeslist}))
proof
  from nodeslist-not-Null sfz
obtain sfz': sfz' = nodeslist \# sfz'
  by (cases nodeslist=Null) auto
from nomatch-prx match sfz'
have hd: \text{hd } ([\text{sn} \leftarrow \text{prx}. \text{repNodes-eq sn } p \text{ low high repa}] @
  [\text{sn} \leftarrow \text{sfz}. \text{repNodes-eq sn } p \text{ low high repa}]) = \text{nodeslist}
  by simp
from match sfz'
have triv: (\forall pt \in \text{set prx} \cup \text{set sfz}. \neg \text{repNodes-eq pt } p \text{ low high repa}) \implies
  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)
8 Proof of Procedure ShareReduceRepList

theory ShareReduceRepListProof imports ShareRepProof begin

lemma (in ShareReduceRepList-impl) ShareReduceRepList-modifies: shows \( \forall \sigma. \Gamma \vdash \{\sigma\} \text{PROC ShareReduceRepList} (\text{´nodeslist}) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{rep}]\} \)
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

lemma hd-filter-app: \([\text{filter } P \text{ xs } \neq []; \text{ zs } = \text{ xs } \circ \text{ ys}] \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 var-eq \equiv (\lambda ns \text{ var}. (\forall \text{no1 } \in \text{ set ns}. \forall \text{no2 } \in \text{ set ns}. \text{no1 } \rightarrow \text{var } = \text{no2 } \rightarrow \text{var}))
shows \( \forall \sigma \text{ ns. } \Gamma \vdash _\sigma \text{List } \text{´nodeslist } \text{´next } \text{ns} \land \)
\( (\forall \text{no } \in \text{ set ns}. \)
\( \quad \text{no } \neq \text{Null } \land ((\text{no } \rightarrow \text{low } = \text{Null }) = (\text{no } \rightarrow \text{high } = \text{Null })) \land \)
\( \quad \text{no } \rightarrow \text{low } \notin \text{ set ns } \land \text{no } \rightarrow \text{high } \notin \text{ set ns } \land \)
\( \quad (\text{isLeafPt } \text{no } \rightarrow \text{low } \neq \text{null } \land \text{no } \rightarrow \text{high } \neq \text{null } \land \)
\( \quad (\text{no } \rightarrow \text{low } = \text{null } \land \text{no } \rightarrow \text{high } = \text{null } \land \)
\( \quad (\text{isLeafPt } \text{no } \rightarrow \text{low } \neq \text{null } \land \text{no } \rightarrow \text{high } \neq \text{null } \land \)
\( \quad (\forall \text{no1 } \in \text{ set ns}. ((\text{rep } \circ \text{low }) \text{no } = (\text{rep } \circ \text{low }) \text{no1}) = (\text{rep } \circ \text{low }) \text{no1})) \land \)
\( \forall \text{no } \in \text{ set ns}. \text{no } \rightarrow \text{rep } \neq \text{Null } \land \)
\( (\text{if } ((\text{rep } \circ \text{low }) \text{no} = (\text{rep } \circ \text{high }) \text{no } \land \text{no } \rightarrow \text{low } = \text{null }) \land \)
\( \text{then } (\text{no } \rightarrow \text{rep} = (\text{rep } \circ \text{low }) \text{no }) \quad \text{else } ((\text{no } \rightarrow \text{rep}) \in \text{ set ns } \land \text{no } \rightarrow \text{rep } \rightarrow \text{rep } = \text{no } \rightarrow \text{rep } \land \)
\( (\forall \text{no1 } \in \text{ set ns}. \quad ((\text{rep } \circ \text{high }) \text{no1} = (\text{rep } \circ \text{high }) \text{no } \land \)
\( (\text{rep } \circ \text{low }) \text{no1} = (\text{rep } \circ \text{low }) \text{no}) = (\text{no } \rightarrow \text{rep } = \text{no1 } \rightarrow \text{rep}))))))\)
apply (hoare-rule HoareTotal.ProcNoRec1)
apply (hoare-rule anno= 
\text{node } := \text{´nodeslist};
\text{WHILE } (\text{node } \neq \text{Null })
\text{INV } \{\exists \text{prx } sfx. \text{List } \text{´node } \text{´next } sfx \land
List 'nodeslist' next ns ∧ ns=prx≤sfx ∧ 

(∀ no ∈ set ns.
  no ≠ Null ∧ ((no→low = Null) = (no→high = Null)) ∧
  no→low ≠ set ns ∧ no→high ≠ set ns ∧
  (isLeaf-no no σlow (no→high = (no→var ≤ 1)) ∧
   (no→low ≠ Null → (no→low)→σrep ≠ Null) ∧
   ((σrep ∆ σlow) no ≠ set ns)) ∧

var-eq ns 'var' ∧
(∀ no. no ≠ set prx → no→σrep = no→'rep') ∧
(∀ no ∈ set prx. no→'rep' ≠ Null ∧
  (if (((′rep ∆ σlow) no = (′rep ∆ σhigh) no ∧ no→σlow ≠ Null)
    then (no→'rep' = (′rep ∆ σlow) no )
    else ((no→'rep')=hd (filter (λsn. repNodes-eq sn no σlow σhigh ′rep) prx) ∧
     ((no→'rep')→'rep' = no→'rep' ∧
      (∀ no1 ∈ set prx.
       ((′rep ∆ σhigh) no1 = (′rep ∆ σhigh) no ∧
        (′rep ∆ σlow) no1 = (′rep ∆ σlow) no) =
       (no→'rep' = no1→'rep')))) ∧
    'nodeslist'="nodeslist ∧ high=σhigh ∧ low=σlow ∧ var=σvar"

VAR MEASURE (length (list 'node' 'next'))
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 in HoareTotal.annotateI)

apply (veg 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 simp)
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 (rule conjI)
apply assumption
apply (rule conjI)
apply (simp (no-asm))
apply (rule conjI)
apply (assumption)
pref 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-asn-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-nsl: no ∈ set ns
assume while-inv: \( \forall \text{no} \in \text{set ns} \).
\[
\text{repa no} \neq \text{Null} \land \\
\text{(if } (\text{repa} \propto \text{low}) \text{ no} = (\text{repa} \propto \text{high}) \text{ no} \land \text{high no} \neq \text{Null} \text{ then } \text{repa no} = (\text{repa} \propto \text{low}) \text{ no} \text{ else } \text{repa no} = \text{hd} [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}] \land \\
\text{repa (repa no)} = \text{repa no} \land \\
(\forall \text{no1} \in \text{set ns}. \\
((\text{repa} \propto \text{high}) \text{ no1} = (\text{repa} \propto \text{high}) \text{ no} \land \\
(\text{repa} \propto \text{low}) \text{ no1} = (\text{repa} \propto \text{low}) \text{ no}) = \\
(\text{repa no} = \text{repa no1})))
\]

assume pre: \( \forall \text{no} \in \text{set ns} \).
\[
\text{no} \neq \text{Null} \land \\
(\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \land \\
\text{low no} \notin \text{ set ns} \land \\
\text{high no} \notin \text{ set ns} \land \\
\text{isLeaf-pt no low high} = (\text{var no} \leq \text{Suc 0}) \land \\
(\text{low no} \neq \text{Null} \longrightarrow \text{rep (low no)} \neq \text{Null}) \land (\text{rep no low}) \neq \text{set ns}
\]

assume same-var: \( \forall \text{no1} \in \text{set ns}. \forall \text{no2} \in \text{set ns}. \text{var no1} = \text{var no2} \)

assume share-case: \( (\text{repa} \propto \text{low}) \text{ no} = (\text{repa} \propto \text{high}) \text{ no} \longrightarrow \text{high no} = \text{Null} \)

assume unmodif: \( \forall \text{no}. \text{no} \notin \text{set ns} \longrightarrow \text{rep no} = \text{repa no} \)

show \( \text{hd} [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}] \in \text{set ns} \land \\
\text{repa (hd [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}])} = \\
\text{hd [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}]}
\]

proof

from no-in-ns pre obtain
\[
\text{no-nNull: } \text{no} \neq \text{Null} \land \\
\text{no-balanced: } (\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \land \\
\text{isLeaf-var: } \text{isLeaf-pt no low high} = (\text{var no} \leq \text{Suc 0}) \land \\
\text{by blast}
\]

have repNodes-eq-same-node: \( \text{repNodes-eq no no low high repa} \)
\text{by (simp add: repNodes-eq-def)}

from no-in-ns have ns-nempty: \( \text{ns} \neq [] \)
\text{by auto}

from no-in-ns repNodes-eq-same-node

have repNodes-not-empty: \( [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}] \neq [] \)
\text{by (rule filter-not-empty)}

then have hd-term-in-ns: \( \text{hd} [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}] \in \text{set ns} \)
\text{by (rule hd-filter-in-list)}

with while-inv obtain
\[
\text{repa-hd-nNull: } \text{repa (hd [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}])} \neq \text{Null} \\
\text{by auto}
\]

let \( ?\text{hd} = \text{hd} [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}] \)

from hd-term-in-ns pre obtain
\[
\text{hd-nNull: } ?\text{hd} \neq \text{Null} \land \\
\text{hd-balanced:} \\
(\text{low (hd [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}])} = \text{Null}) = \\
(\text{high (hd [\text{sn} \leftarrow \text{ns} \cdot \text{repNodes-eq sn no low high repa}])} = \text{Null}) \land \\
\text{hd-isLeaf-var:}
\]

88
isLeaf-pt \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})] \) \( \text{low} \ \text{high} = \)
\( (\text{var} \ (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \leq \text{Suc} \ 0) \)
by blast
have repa \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) = \)
\( \text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa}] \)
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 \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \leq 1 \)
by simp
with hd-isLeaf-var have
isLeaf-pt \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \) \( \text{low} \ \text{high} \)
by simp
with while-inv hd-term-in-ns repNodes-not-empty show \( \text{thesis} \)
apply (simp add: isLeaf-pt-def)
apply (erule-tac x = \( \text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa}] \) 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 \( \neq \) Null
with share-case have repchildren-neq: \( (\text{repa} \ \text{x} \ \text{low}) \) no \( \neq \) \( (\text{repa} \ \text{x} \ \text{high}) \) no
by simp
from repNodes-not-empty have
repNodes-eq \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \) \( \text{no} \ \text{low} \ \text{high} \)
repa
by (rule hd-filter-prop)
then
have \( (\text{repa} \ \text{x} \ \text{low}) \) \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) = \)
\( (\text{repa} \ \text{x} \ \text{low}) \) no \( \wedge \)
\( (\text{repa} \ \text{x} \ \text{high}) \) \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) = \)
\( (\text{repa} \ \text{x} \ \text{high}) \) no
by (simp add: repNodes-eq-def)
with repchildren-neq have
\( (\text{repa} \ \text{x} \ \text{low}) \) \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \)
\( \neq \) \( (\text{repa} \ \text{x} \ \text{high}) \) \( (\text{hd} \ [\text{sn} \leftarrow \text{ns} \ . \ \text{repNodes-eq} \ \text{sn} \ \text{no} \ \text{low} \ \text{high} \ \text{repa})]) \)
by simp

89
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
apply fastforce
apply fastforce
done
qed

with hd-term-in-ns
show ?thesis
by simp
qed

next

fix var low high rep nodestlist 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 ≠ Null
assume nodes-balanced-ordered: ∀ no∈set (prx @ node # sfx).
no ≠ Null ∧
(low no = Null) = (high no = Null) ∧
low no ∉ set (prx @ node # sfx) ∧
high no ∉ set (prx @ node # sfx) ∧
isLeaf-pt no low high = (var no ≤ (1::nat)) ∧
(low no ≠ Null → rep (low no) ≠ Null) ∧
(repa ∞ low) no ∉ set (prx @ node # sfx)
assume all-nodes-same-var: \( ∀ no1∈set (prx @ node # sfx) \). var no1 = var no2
assume rep-repa-nc: ∀ no. no ∉ set prx → rep no = repa no
assume while-inv: ∀ no∈set prx.
repa no ≠ Null ∧
(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))))
assume not-Leaf: ¬ isLeaf-pt node low high
assume repchildren-eq-ahn: repa (low node) = repa (high node)
show (\( ∀ no. no ∉ set (prx @ [node]) \) →
repa no = (repa(node := repa (high node))) no) ∧
(∀ no∈set (prx @ [node]).

90
\[
\begin{align*}
\text{repa}(\text{node} := \text{repa} (\text{high node}))) \neq \text{Null} & \land \\
\text{if} (\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{low}) \text{ no } = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{high}) \text{ no } & \land \\
\text{low no} \neq \text{Null} \\
\text{then} (\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no } = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{low}) \text{ no} \\
\text{else} (\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no } = \\
\text{hd} [\text{sn}\leftarrow \text{prx} @ [\text{node}]. \\
\text{repNodes-eq sn no low high}] \land \\
(\text{repa}(\text{node} := \text{repa} (\text{high node})))) \\
((\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no}) = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no } \land \\
(\forall \text{no1} \in \text{set} (\text{prx} @ [\text{node}]), \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{high}) \text{ no1} = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{high}) \text{ no } \land \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{low}) \text{ no1} = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \propto \text{low}) \text{ no} \\
((\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no}) = \\
(\text{repa}(\text{node} := \text{repa} (\text{high node}))) \text{ no1})) \\
\text{(is } \text{?NodesUnmodif } \land \text{?NodesModif})
\end{align*}
\]

\text{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 \text{repa} (\text{low node}).

\text{from rep-repa-nc}
\text{have nodes-unmodif: ?NodesUnmodif}
\text{by auto}
\text{hence rep-Sucna-nc:}
\text{(\forall \text{no}. \text{no} \notin \text{set} (\text{prx} @ [\text{node}])}
\text{\rightarrow \text{rep no} = (\text{repa(\text{node} := \text{repa} (\text{low (node)))}) no})
\text{by auto}
\text{have nodes-modif: ?NodesModif (is \forall \text{no} \in \text{set} (\text{prx} @ [\text{node}]). ?P \text{ no } \land \text{?Q no})}
\text{proof (rule ballI)}
\text{fix no}
\text{assume no-in-take-Sucna: no \in \text{set (prx} @ [\text{node}])}
\text{show ?P \text{ no } \land \text{?Q no}}
\text{proof (cases no = node)}
\text{case False}
\text{note no-noteq-nln=this}
\text{with no-in-take-Sucna}
\text{have no-in-take-n: no \in \text{set prx}}
\text{by auto}
\text{with no-in-take-n while-inv obtain}
\text{repa-no-nNull: repa no \neq \text{Null } \land}
\text{repa-cases: (if (repa \propto \text{low}) no = (repa \propto \text{high}) no ) \land \text{low no } \neq \text{Null}
\text{then repa no} = (repa \propto \text{low}) \text{ no}
\text{else repa no} = \text{hd} [\text{sn}\leftarrow \text{prx} . \text{repNodes-eq sn no low high repa}]
\wedge \text{repa (repa no)} = \text{repa no } \land
∀ 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 no = Null) = (high no = Null) and
lnln-notin-nodeslist: low no ∉ set (prx @ node # sfx) and
hno-notin-nodeslist: high no ∉ set (prx @ node # sfx) and
isLeaf-var-no: isLeaf-pt node low high = (var node ≤ 1) and
node-nNull-rep-nNull-nln: (low no ≠ Null
→ rep (low node) ≠ Null) and
nln-varrep-le-var: (repa ∝ low) node ∉ set (prx @ node # sfx)

apply −
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
hno-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: (repa ∝ low) no ∉ set (prx @ node # sfx)

apply −
apply (erule-tac x=no in ballE)

apply auto
done

from hno-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)) ∝ high) no = (repa ∝ high) no
by (auto simp add: null-comp-def)

have share-reduce-if: ?Q no
proof (cases (repa (node := repa (low node)) ∝ low) no =
(repa (node := repa (low node)) ∝ high) no ∧ low no ≠ Null)

case True
then obtain
red-case: (repa (node := repa (low node)) ∝ low) no =
(repa(node := repa (low node)) \propto 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 \propto low) no = (repa \propto high) no
by simp
with repa-cases lno-nNull have repa no = (repa \propto 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)) \propto low) no
by simp
with True repchildren-eq-nln show ?thesis
by auto
next
assume share-case-ext:
\neg ((repa(node := repa (low node)) \propto low) no =
(repa(node := repa (low node)) \propto high) no \land low no \neq Null)
from not-Leaf isLeaf-var-nln
have I < var node
by simp
with all-nodes-same-var
have all-nodes-nl-Suc0-l-var: \forall x \in set (prx @ node # sfx). I < var x
using [[simp-depth-limit=I]]
by auto
with nodes-balanced-ordered
have all-nodes-nl-noLeaf:
\forall x \in set (prx @ node # sfx). \neg isLeaf-pt x low high
apply \neg
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 set (prx @ node # sfx). (low x = Null) = (high x = Null)
apply \neg
apply rule
apply (drule-tac x=x in bspec, assumption)
apply auto
done
from all-nodes-nl-Suc0-l-var no-in-nodesslist
have Suc0-l-var-no: I < var no
by auto
with isLeaf-var-no have no-nLeaf: \neg isLeaf-pt no low high
by simp
with balanced-children have lno-nNull: low no ≠ Null
by (simp add: isLeaf-pt-def)
with balanced-children have hno-nNull: high no ≠ Null
by simp
with share-case-ext ext-rep-null-comp-low ext-rep-null-comp-high lno-nNull
have repchildren-neq-no: (repa ∞ low) no ≠ (repa ∞ high) 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] ∧
  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
then have repa-no: repa no = hd [sn← prx . repNodes-eq sn no low high repa]
by simp
from Suc0-l-var-no have ∀ x ∈ set (prx @ node ≠ sfx). 1 < var no
by auto
from no-in-take-n have [sn← prx . repNodes-eq sn no low high repa] ≠ []
  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:
  (repa ∞ low) (repa no) = (repa ∞ low) no ∧
  (repa ∞ high) (repa no) = (repa ∞ high) no
by (simp add: repNodes-eq-def)
from lno-notin-nodeslist rep-repa-nc
have rep-repa-nc-low-no: rep (low no) = repa (low no)
  apply –
  apply (erule-tac x=low no in allE)
  apply auto
done
have ∀ x ∈ set (prx @ [node]).
  repNodes-eq x no low high (repa(node := repa (low node))) =
  repNodes-eq x no low high repa
proof (rule ballI, unfold repNodes-eq-def)
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-noLeaf nodes-balanced-ordered
have children-nNull-x: low x ≠ Null ∧ high x ≠ Null

94
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-nodeslist nodes-balanced-ordered
have low x \notin set (prx @ node # sfx) \land high x \notin set (prx @ node # sfx)
  apply −
  apply (drule-tac x=x in bspec,assumption)
  apply auto
done
with lno-notin-nodeslist hno-notin-nodeslist
children-nNull-x lno-nNull hno-nNull
show ((repa(node := repa (low node)) \propto high) x =
  (repa(node := repa (low node)) \propto high) no \land
  (repa(node := repa (low node)) \propto low) x =
  (repa(node := repa (low node)) \propto low) no) =
  ((repa \propto high) x = (repa \propto high) no \land
  (repa \propto low) x = (repa \propto low) no)
  by (simp add: null-comp-def)
qed
then have filter-extrep-rep:
  \[ sn \leftarrow (prx @ [node]). \ repNodes-eq sn no low high \]
  \[ \left\langle \begin{array}{c}
  (repa(node := repa (low node)))
  \end{array} \right\rangle = \]
  \[ \left\langle \begin{array}{c}
  (sn \leftarrow (prx @ [node]). \ repNodes-eq sn no low high repa)
  \end{array} \right\rangle \]
  by (rule P-eq-list-filter)
from no-in-take-n
have filter-n-notempty: \[ sn \leftarrow prx. \ repNodes-eq sn no low high repa \neq [] \]
  apply (rule filter-not-empty)
  apply (simp add: repNodes-eq-def)
done
then have 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
with no-noteq-nln filter-extrep-rep repa-no
have ext-repa-no: (repa(node:= repa (low node))) no =
  hd \[ sn \leftarrow prx@ [node]. \ repNodes-eq sn no low high \]
  (repa(node := repa (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) \neq Null
  apply (simp add: null-comp-def)
  apply auto
done

95
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 # sfs)
    by auto
show ((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 (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]
   ∈ set prx
apply −
apply (rule hd-filter-in-list)
apply auto
done
with repa-no
have repa-no-in-nodeslist: repa no ∈ set (prx @ node # sfx)
   by auto
from lnln-notin-nodeslist rep-repa-nc
have rep-repa-low-nln: rep (low node) = repa (low node)
   by auto
from all-nodes-nl-noLeaf nl-balanced-children
have low node ≠ Null
   by (auto simp add: isLeaf-pt-def)
with rep-repa-low-nln lnln-notin-nodeslist nln-varrep-le-var
have repa (low node) ∉ 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)) ∝ high) no1 =
   (repa(node := repa (low node)) ∝ high) no ∧
   (repa(node := repa (low node)) ∝ low) no1 =
   (repa(node := repa (low node)) ∝ low) no
by simp
next
assume no-no1-high:
   (repa(node := repa (low node)) ∝ high) no1 =
   (repa(node := repa (low node)) ∝ high) no
assume no-no1-low:
   (repa(node := repa (low node)) ∝ low) no1 =
   (repa(node := repa (low node)) ∝ low) no
from True repchildren-eq-nln
have repachildren-eq-no1: repa (low no1) = repa (high no1)
   by simp
from not-Leaf True nl-balanced-children
have children-nNull-no1: (low no1) ≠ Null ∧ high no1 ≠ Null
   by (simp add: isLeaf-pt-def)
with repachildren-eq-no1
have repchildren-eq-no1: (repa ∝ low) no1 = (repa ∝ high) no1
   by (simp add: null-comp-def)
from no-no1-low children-nNull-no1 lno-nNull
have rep-low-eq-no-no1: (repa ∞ low) no1 = (repa ∞ low) no
  by (simp add: null-comp-def)
from no-no1-high children-nNull-no1 hno-nNull
  hln-notin-nodeslist hno-notin-nodeslist True
have rep-high-eq-no-no1: (repa ∞ high) no1 = (repa ∞ high) no
  by (simp add: null-comp-def)
with rep-low-eq-no-no1 repchildren-eq-no1
have (repa ∞ low) no = (repa ∞ 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 ≠ node
from no1-in-nodeslist nodes-balanced-ordered
have children-notin-nl-no1:
  low no1 ∉ set (prx @ node # sfx) ∧ high no1 ∉ 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 ∈ set prx
  by auto
from no1-in-nodeslist all-nodes-nl-noLeaf all-nodes-nl-balanced
have children-nNull-no1: (low no1) ≠ Null ∧ high no1 ≠ 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)) ∞ high) no1
    = (repa(node := repa (low node)) ∞ high) no
  assume ext-repa-low-no1-no:
    (repa(node := repa (low node)) ∞ low) no1
    = (repa(node := repa (low node)) ∞ 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 ∞ high) no1 = (repa ∞ high) no
  by (simp add: null-comp-def)
from children-nNull-no1 hno-nNull ext-repa-low-no1-no
  children-notin-nl-no1 hno-notin-nodeslist
have repa-low-no1-no: (repa ∞ low) no1 = (repa ∞ low) no
  by (simp add: null-comp-def)
from repchildren-neq-no repa-high-no1-no repa-low-no1-no
have (repa ∞ low) no1 ≠ (repa ∞ 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 ∞ high) no1 = (repa ∞ high) no ∧
    (repa ∞ low) no1 = (repa ∞ low) no)
    by auto
  with children-notin-nl-no1 children-nNull-no1 lno-notin-nodeslist
    hno-notin-nodeslist hno-nNull hno-nNull
  show (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
  by (auto simp add: null-comp-def)
qed
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 ∈ set (prx @ node # sfx)
    by simp
  from no-nln no-Leaf no-in-nodeslist
  nodes-balanced-ordered [rule-format, OF this] obtain
    low-no-nNull: low no ≠ Null and
    high-no-nNull: high no ≠ Null and
    rep-low-no-nNull: rep (low no) ≠ Null and
    lno-notin-nl: low no ∉ set (prx @ node # sfx) and
    hno-notin-nl: high no ∉ set (prx @ node # sfx) and
    children-nNull-no: (low no ≠ Null) ∧ (high no ≠ Null)
  apply (unfold isLeaf-pt-def)
  apply blast
then have \( \text{low} \notin \text{set} \ prx \)
by auto

with rep-repa-nc no-nln rep-low-no-nNull
have \((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \neq \text{Null}\)
by simp

moreover
have \(\text{if} \ (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no \(\neq \text{Null}\)
by simp

then \( (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no \(\land \text{low no} \neq \text{Null} \)
else \( (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no

\( (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no \(\neq \text{Null} \)

\( (\forall \text{no1} \in \text{set} \ (\text{prx}[\text{node}]) \). \ (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no1 \(\neq \text{Null} \)
(\( (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no1 \(\neq \text{Null} \) ) \(
\neq (\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no1 \)

\( \text{proof} \) (cases \((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no =
\((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no \(\land \text{low no} \neq \text{Null} \)

\( \text{case True} \)

note red-case=this
with children-nNull-no lno-notin-nl hno-notin-nl
have \((\text{repa} \propto \text{low}) \) no = \((\text{repa} \propto \text{high}) \) no
by (auto simp add: null-comp-def)

from children-nNull-no lno-notin-nl
have ext-repa-eq-repa-low: \((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no
= \((\text{repa} \propto \text{low}) \) no
by (auto simp add: null-comp-def)

from children-nNull-no hno-notin-nl
have ext-repa-eq-repa-high:
\((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no
= \((\text{repa} \propto \text{high}) \) no
by (auto simp add: null-comp-def)

from no-nln children-nNull-no
have \( \text{repa} (\text{low node}) = (\text{repa} \propto \text{low}) \) 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: \( \neg ((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no

\text{proof} \) (cases \((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no =
\((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no \(\land \text{low no} \neq \text{Null} \)

\( \text{case True} \)

note red-case=this
with children-nNull-no lno-notin-nl hno-notin-nl
have \((\text{repa} \propto \text{low}) \) no = \((\text{repa} \propto \text{high}) \) no
by (auto simp add: null-comp-def)

from children-nNull-no lno-notin-nl
have ext-repa-eq-repa-low: \((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no
= \((\text{repa} \propto \text{low}) \) no
by (auto simp add: null-comp-def)

from children-nNull-no hno-notin-nl
have ext-repa-eq-repa-high:
\((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{high} \) no
= \((\text{repa} \propto \text{high}) \) no
by (auto simp add: null-comp-def)

from no-nln children-nNull-no
have \( \text{repa} (\text{low node}) = (\text{repa} \propto \text{low}) \) 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: \( \neg ((\text{repa}(\text{node} := \text{repa} (\text{low node})) ) \propto \text{low} \) no

100
\[(\text{repa(node := repa (low node)}) \propto \text{high}) \text{ no} \wedge \text{low no} \neq \text{Null}\]

**with** low-no-nNull **have** (repa(node := repa (low node)) \propto \text{low}) \text{ no} \\
\neq (\text{repa(node := repa (low node)}) \propto \text{high}) \text{ no} \\
\text{by simp}

**with** children-nNull-no lno-notin-nl **have** (repa \propto \text{low}) \text{ no} \neq (\text{repa \propto high}) \text{ no} \\
\text{by (auto simp add: null-comp-def)}

**with** children-nNull-No have repa (low no) \neq \text{repa (high no)} \\
\text{by (simp add: null-comp-def)}

**with** repchildren-eq-nln no-nln show ?thesis \\
\text{by simp}

**qed**

ultimately show ?thesis \\
\text{using repchildren-eq-nln}

apply –
apply (simp only:)
apply (simp (no-asm))
doné

**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: \( \forall \text{no} \in \text{set (prx @ node # sfx)}. \) \\
(\text{no} \neq \text{Null} \wedge \\
(\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \wedge \\
\text{low no} \notin \text{set (prx @ node # sfx)} \wedge \\
\text{high no} \notin \text{set (prx @ node # sfx)} \wedge \\
isLeaf-p\text{t no} \text{ low high} = (\text{var no} \leq (1::nat)) \wedge \\
(\text{low no} \neq \text{Null} \rightarrow \text{repa (low no)} \neq \text{Null}) \wedge \\
(\text{rep \propto low}) \text{no} \notin \text{set (prx @ node # sfx)}

**assume** all-nodes-same-var: \( \forall \text{no1} \in \text{set (prx @ node # sfx)} . \) \\
\( \forall \text{no2} \in \text{set (prx @ node # sfx)}, \) \text{var no1} = \text{var no2}

**assume** rep-repa-nc: \( \forall \text{no}. \text{no} \notin \text{set prx \rightarrow rep no} = \text{repa no} 

**assume** while-inv: \( \forall \text{no} \in \text{set prx}. \) \\
\text{repa no} \neq \text{Null} \wedge \\
(\text{if (repa \propto low}) \text{no} = (\text{repa \propto high}) \text{ no} \wedge \text{low no} \neq \text{Null} \\
\text{then repa no} = (\text{repa \propto low}) \text{ no} \\
\text{else repa no} = \text{hd [sn←prx . repNodes-eq sn no low high repa] \wedge} \\
\text{repa (repa no)} = \text{repa no} \wedge \\
(\forall \text{no1} \in \text{set prx}. \\
\text{(repa \propto high}) \text{no1} = (\text{repa \propto high}) \text{ no} \wedge \\
(\text{repa \propto low}) \text{no1} = (\text{repa \propto low}) \text{ no}) = \\
(\text{repa no} = \text{repa no1}))

**assume** share-cond:
\[ ¬(\neg \text{isLeaf-pt node low high} \land \text{repa\ (low node)} = \text{repa\ (high node)}) \]

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

**assume** repa-repb-nc: \(\forall pt, pt \neq \text{node} \rightarrow \text{repa\ pt} = \text{repb\ pt}\)

**assume** var-repb-node: \(\text{var\ (repa\ node)} = \text{var\ node}\)

**show** \((\forall no, no \notin \text{set (prx @ [node])} \rightarrow \text{repa\ no} = \text{repb\ no}) \land
(\forall no\in\text{set (prx @ [node])}.
\text{repa\ no} \neq \text{Null} \land
(\text{if (repa\ \propto \text{low})\ no} = (\text{repa\ \propto \text{high})\ no} \land \text{low no} \neq \text{Null}
\text{then repa\ no} = (\text{repa\ \propto \text{low})\ no}
\text{else repa\ no} =
\text{hd \[sn\leftarrow\text{prx} @ \text{node}. \text{repNodes-eq sn node low high repa}\] \land
\text{repa\ (repa\ no)} = \text{repa\ no} \land
(\forall no1\in\text{set (prx @ [node])}.
(\text{repa\ \propto \text{high})\ no1} = (\text{repa\ \propto \text{high})\ no} \land
(\text{repa\ \propto \text{low})\ no1} = (\text{repa\ \propto \text{low})\ no) =
(\text{repa\ no} = (\text{repa\ no1})))))\)

**proof**

**have** repa-repb-nc: \((\forall no, no \notin \text{set (prx @ [node])} \rightarrow \text{repa\ no} = \text{repb\ no})\)

**proof** (intro allI impI)

**fix** no

**assume** no-notin-take-Sucn: \(no \notin \text{set (prx @ [node])}\)

**with** repa-repa-nc

**have** repa-repa-nc-Sucn: \(\text{repa\ no} = \text{repa\ no}\)

by auto

**from** no-notin-take-Sucn **have** no \(\neq \text{node}\)

by auto

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

by auto

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

by simp

qed

moreover

**have** repb-no-share-def:
\((\forall no\in\text{set (prx @ [node])}.\neg\text{(repa\ \propto \text{low}) no} = (\text{repa\ \propto \text{high}) no} \land \text{low no} \neq \text{Null}) \rightarrow
\text{repa\ no} = \text{hd \[sn\leftarrow\text{prx} @ \text{node}. \text{repNodes-eq sn node low high repa}\]}\)

**proof** (intro allI impI)

**fix** no

**assume** no-in-take-Sucn: \(no \in \text{set (prx @ [node])}\)

**assume** share-prop: \(\neg\text{(repa\ \propto \text{low}) no} = (\text{repa\ \propto \text{high}) no} \land \text{low no} \neq \text{Null})\)

**from** share-prop **have** share-or:
\(\text{repa\ \propto \text{low}) no} \neq (\text{repa\ \propto \text{high}) no} \lor \text{low no} = \text{Null}\)

**using** [[simp-depth-limit=2]]

by simp

**from** no-in-take-Sucn **have** no-in-nl: \(no \in \text{set (prx @ node \# sfx)}\)

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 reb no = hd [sn←(prx [node]). repNodes-eq sn no low high reb]

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] 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 no = repa no)))

using [[simp-depth-limit = 2]] by auto

from no-in-take-n have filter-take-n-notempty: [sn←prx.

repNodes-eq sn no low high repa] ≠ []

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]

by auto

thus ?thesis

proof (cases low no = Null)

case True

note hno-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] ∧
\( \text{repa} (\text{repa} \; \text{no}) = \text{repa} \; \text{no} \; \land \\
(\forall \; \text{no1} \in \text{set} \; \text{prx}. \; ((\text{repa} \; \propto \; \text{high}) \; \text{no1} = (\text{repa} \; \propto \; \text{high}) \; \text{no} \\
\land \; (\text{repa} \; \propto \; \text{low}) \; \text{no1} = (\text{repa} \; \propto \; \text{low}) \; \text{no}) = (\text{repa} \; \text{no} = \text{repa} \; \text{no1})) \\
\text{by} \; \text{auto} \\
\text{have} \; \text{all-nodes-in-nl-Leafs:} \\
\forall \; x \in \text{set} \; (\text{prx} \; @ \; \text{node} \; \# \; \text{sfx}). \; \text{isLeaf-pt} \; x \; \text{low} \; \text{high} \\
\text{proof} \; (\text{intro} \; \text{ballI}) \\
\text{fix} \; x \\
\text{assume} \; \text{x-in-nodeslist}: \; x \in \text{set} \; (\text{prx} \; @ \; \text{node} \; \# \; \text{sfx}) \\
\text{from} \; \text{isLeaf-no} \; \text{isLeaf-var-no} \; \text{have} \; \text{var} \; \text{no} \leq 1 \\
\text{by} \; \text{simp} \\
\text{with} \; \text{all-nodes-same-var} \; [\text{rule-format}, \; \text{OF} \; \text{x-in-nodeslist} \; \text{no-in-nl}] \\
\text{have} \; \text{var} \; x \leq 1 \\
\text{by} \; \text{simp} \\
\text{with} \; \text{nodes-balanced-ordered} \; [\text{rule-format}, \; \text{OF} \; \text{x-in-nodeslist}] \\
\text{show} \; \text{isLeaf-pt} \; x \; \text{low} \; \text{high} \\
\text{by} \; (\text{auto} \; \text{simp} \; \text{add:} \; \text{isLeaf-pt-def}) \\
\text{qed} \\
\text{have} \; \forall \; x \in \text{set} \; (\text{prx}@]\text{node}[\text{sfx}]. \; \text{repNodes-eq x no low high repb} \\
= \; \text{repNodes-eq x no low high repa} \\
\text{proof} \; (\text{rule} \; \text{ballI}) \\
\text{fix} \; x \\
\text{assume} \; \text{x-in-take-Sucn}: \; x \in \text{set} \; (\text{prx}@]\text{node}[\text{sfx}] \\
\text{hence} \; \text{x-in-nodeslist}: \; x \in \text{set} \; (\text{prx} \; @ \; \text{node} \; \# \; \text{sfx}) \\
\text{by} \; \text{auto} \\
\text{with} \; \text{all-nodes-in-nl-Leafs} \; \text{have} \; \text{isLeaf-pt} \; x \; \text{low} \; \text{high} \\
\text{by} \; \text{auto} \\
\text{with} \; \text{isLeaf-no} \; \text{repa-repb-nc} \; \text{show} \; \text{repNodes-eq x no low high repb} \\
= \; \text{repNodes-eq x no low high repa} \\
\text{by} \; (\text{simp} \; \text{add:} \; \text{repNodes-eq-def} \; \text{null-comp-def} \; \text{isLeaf-pt-def}) \\
\text{qed} \\
\text{then have} \; [\text{sn} \leftarrow (\text{prx}@]\text{node}[\text{sfx}]. \; \text{repNodes-eq sn no low high repa}] \\
= \; [\text{sn} \leftarrow (\text{prx}@]\text{node}[\text{sfx}]. \; \text{repNodes-eq sn no low high repb}] \\
\text{apply} \; \neg \\
\text{apply} \; (\text{rule} \; \text{P-eq-list-filter}) \\
\text{apply} \; \text{simp} \\
\text{done} \\
\text{with} \; \text{hd-term-n-Sucn} \; \text{while-low-Null} \; \text{repb-repa-no} \; \text{show} \; ?\text{thesis} \\
\text{by} \; \text{auto} \\
\text{next} \\
\text{assume} \; \text{lno-nNull}: \; \text{low} \; \text{no} \neq \; \text{Null} \\
\text{with} \; \text{balanced-no} \; \text{have} \; \text{lno-nNull}: \; \text{high} \; \text{no} \neq \; \text{Null} \\
\text{by} \; \text{simp} \\
\text{with} \; \text{lno-nNull} \; \text{have} \; \text{no-nLeaf}: \; \neg \; \text{isLeaf-pt} \; \text{no low high} \\
\text{by} \; (\text{simp} \; \text{add:} \; \text{isLeaf-pt-def}) \\
\text{with} \; \text{isLeaf-var-no} \; \text{have} \; \text{Sucn-s-varno}: \; 1 \; < \; \text{var} \; \text{no} \\
\text{by} \; \text{auto} \\
\text{with} \; \text{no-in-nl} \; \text{all-nodes-same-var} \\
\text{have} \; \text{all-nodes-nl-var}: \; \forall \; x \in \text{set} \; (\text{prx} \; @ \; \text{node} \; \# \; \text{sfx}). \; 1 \; < \; \text{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:
  \( \forall x \in \text{set (prx @ node # sfx)}. \sim \text{isLeaf-pt x low high} \)
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 repachildren-eq-no: (repb \( \propto \) low) no \( \neq \) (repb \( \propto \) 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 \( \propto \) low) no \( \neq \) (repa \( \propto \) 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 \( \forall x \in \text{set (prx@[(node)]).} \)
  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 \( \in \text{set (prx@[(node)])} \)
hence x-in-nodeslist: x \( \in \text{set (prx @ node # sfx)} \)
  by auto
with all-nodes-nl-nLeaf have x-nLeaf: \( \sim \text{isLeaf-pt x low high} \)
  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

105
null 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← (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 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← (prx@[node # sfz]). 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 ∀ x ∈ set (prx @ node # sfz). 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 # sfz)
        have all-nodes-in-nl-Leafs:
          ∀ x ∈ set (prx @ node # sfz). isLeaf-pt x low high
          proof (intro ballI)
            fix x
            assume x-in-nodeslist: x ∈ set (prx @ node # sfz)
            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
        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 # sfz]). repNodes-eq sn node low high repb]

106
apply −
apply (subgoal-tac
[sn← (prx@node#sfx). repNodes-eq sn node low high repa]
= [sn←(prx @ node # sfz) . 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 # sfz). 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 ∀ x ∈ set (prx@node#sfz). 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#sfz)

have all-nodes-in-nl-Leafs:
∀ x ∈ set (prx@node#sfz). ¬isLeaf-pt x low high
proof (intro ballI)
fix x
assume x-in-nodeslist: x ∈ set (prx@node#sfz)
from no-nLeaf isLeaf-var-no have 1 < var no
by simp

with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
have 1 < var x
by auto

with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
show ¬isLeaf-pt x low high

107
apply (unfold isLeaf-pt-def)
apply fastforce
done
 qed
with x-in-nodeslist have x-nLeaf: ¬ isLeaf-pt x low high
  by auto
from nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
  have (low x = Null) = (high x = Null)
    ∧ low x ∉ set (prx[node#sfx]) ∧ high x ∉ set (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 repNodes-eq x no low high repb = repNodes-eq x 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:
    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
  qed
  qed
  qed
  have repb-no-red-def: (∀ no∈set (prx[node]) . (repb ∞ low) no = (repb ∞ high)
    ∧ low no ≠ Null → repb no = (repb ∞ low) no)
proof (intro ballyl impl)
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: high 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 ∝ 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 ∨ repa (low no) ≠ 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: ¬ isLeaf-pt no low high
with share-cond-pre
have repa (low no) ≠ repa (high no)
by simp
with no-nLeaf balanced-no have (repa ∝ low) no ≠ (repa ∝ high) no
by (simp add: null-comp-def isLeaf-pt-def)
with red-cond-no show ?thesis
by simp
qed
have while-while: (∀ no ∈ set (prx@node)).
repa no ≠ Null ∧
(if (repa ∝ low) no = (repa ∝ high) no ∧ low no ≠ Null
then repa no = (repa ∝ low) no
else repa no = hd [sn←(prx@node)]. repNodes-eq sn no low high repb] ∧
repa (repa no) = repb no ∧
(∀ no1 ∈ set ((prx@node))). ((repa ∝ high) no1 = (repa ∝ high) no
∧ (repa ∝ low) no1 = (repa ∝ low) no = (repa no = repb no1)))
(is ∀ no ∈ set (prx@node). ?P no ∧ ?Q no)
proof (intro ballI)
fix no
assume no-in-take-Sucn: no ∈ set (prx@node)
hence 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
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-neq-nln: ∀ p. p ∉ set (prx@node#sfr) → 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
  repb-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 (repa ∝ low) no = (repa ∝ 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:
  ¬ ((repa ∝ low) no = (repa ∝ 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 (repa ∝ low) no ≠ (repa ∝ high) no ∨ low no = Null
  using [[simp-depth-limit=2]]
  by simp
thus ?thesis
proof (cases low no = Null)
\[
\text{case } \text{True} \\
\text{note } \text{lno-Null} = \text{this} \\
\text{with } \text{balanced-no} \ \text{have } \text{hno-Null}: \text{high no} = \text{Null} \\
\quad \text{by } \text{simp} \\
\text{from } \text{hno-Null} \ \text{hno-Null} \ \text{have } \text{isLeaf-no}: \text{isLeaf-pt no low high} \\
\quad \text{by } \text{(simp add: isLeaf-pt-def)} \\
\text{from } \text{True while-share-red-exp} \\
\text{have } \text{while-low-Null}: \\
\quad \text{repa no} = \text{hd } [\text{sn} \leftarrow \text{prx}. \ \text{repNodes-eq sn no low high repa}] \land \\
\quad \text{rep} (\text{rep no}) = \text{rep no} \land \\
\quad (\forall \text{no1} \in \text{set prx}. ((\text{rep} \propto \text{high}) \ \text{no1} = (\text{rep} \propto \text{high}) \ \text{no}) \land (\text{rep} \propto \text{low}) \ \text{no1} = (\text{rep} \propto \text{low}) \ \text{no}) = (\text{rep no} = \text{rep no1})) \\
\quad \text{by } \text{auto} \\
\text{from } \text{no-in-take-n} \\
\text{have } [\text{sn} \leftarrow \text{prx}. \ \text{repNodes-eq sn no low high repa}] \neq [] \\
\quad \text{apply } - \\
\quad \text{apply (rule filter-not-empty}) \\
\quad \text{apply (auto simp add: repNodes-eq-def)} \\
\quad \text{done} \\
\text{then have } \text{hd-term-n-Sucn: } \text{hd } [\text{sn} \leftarrow \text{prx}. \ \text{repNodes-eq sn no low high repa}] = \\
\quad \text{hd } [\text{sn} \leftarrow (\text{prx}@[\text{node}]) . \ \text{repNodes-eq sn no low high repa}] \\
\quad \text{apply } - \\
\quad \text{apply (rule hd-filter-app [symmetric])} \\
\quad \text{apply auto} \\
\quad \text{done} \\
\text{have } \text{all-nodes-in-nl-Leafs:} \\
\quad (\forall x \in \text{set } (\text{prx}@[\text{node}#sfx]). \ \text{isLeaf-pt x low high)} \\
\text{proof (intro ballI)} \\
\quad \text{fix } x \\
\quad \text{assume } x-in-nodeslist: \ x \in \text{set } (\text{prx}@[\text{node}#sfx}) \\
\text{from } \text{isLeaf-no} \ \text{isLeaf-var-no} \ \text{have } \text{var no} \leq 1 \\
\quad \text{by } \text{simp} \\
\text{with } \text{all-nodes-same-var } \text{[rule-format, OF x-in-nodeslist no-in-nl]} \\
\text{have } \text{var x} \leq 1 \\
\quad \text{by } \text{simp} \\
\text{with } \text{nodes-balanced-ordered } \text{[rule-format, OF x-in-nodeslist]} \\
\text{show } \text{isLeaf-pt x low high} \\
\quad \text{by } \text{(auto simp add: isLeaf-pt-def)} \\
\text{qed} \\
\text{from } \text{no-in-take-Sucn} \ \text{have } \\
\text{filter-Sucn-no-notempty:} \\
\quad [\text{sn} \leftarrow (\text{prx}@[\text{node}]) . \ \text{repNodes-eq sn no low high repb}] \neq [] \\
\quad \text{apply } - \\
\quad \text{apply (rule filter-not-empty}) \\
\quad \text{apply (auto simp add: repNodes-eq-def)} \\
\quad \text{done} \\
\text{then have } \text{hd-term-in-take-Sucn:} \\
\quad \text{hd } [\text{sn} \leftarrow (\text{prx}@[\text{node}]) . \ \text{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@#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
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

113
then have \( \text{hd-term-nodeslist-Sucn} \): 
\[
\begin{align*}
\text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node}\#\text{sfx} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb} = \\
\text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb}
\end{align*}
\]
by simp
from no-in-take-Sucn filter-Sucn-no-notempty
have filter-filter: \( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb} = \\
\text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{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) = \( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node}\#\text{sfx} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb}
by simp
with nested-hd-repb-repb
have repb (repb no) = \( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node}\#\text{sfx} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb}
by simp
with hd-term-nodeslist-Sucn
have repb (repb no) = \( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn} \\
( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn no low high repb}) \\
\text{low high repb}
by simp
with filter-filter
have repb (repb no) = \( \text{hd } [\text{sn} \leftarrow \langle \text{prx}@\text{node} \rangle]. \text{repNodes-eq sn} \\
\text{repNodes-eq sn no low high repb}
by simp
with repb-no-def show thesis
by simp
qed
have two-nodes-repb: \( \forall \text{no1} \in \text{set } \langle \text{prx}@\text{node} \rangle \).
\( (\text{repb } \& \text{ high }) \text{no1} = (\text{repb } \& \text{ high }) \text{no} \)
\& (\text{repb } \& \text{ low }) \text{no1} = (\text{repb } \& \text{ low }) \text{no} = (\text{repb no } = \text{repb no1})
proof (intro ballI)
fix no1
assume no1-in-take-Sucn: \( \text{no1} \in \text{set } \langle \text{prx}@\text{node} \rangle \)
then have \( \text{no1} \in \text{set } \langle \text{prx}@\text{node}\#\text{sfx} \rangle \) by auto
with all-nodes-in-nl-Leafs
have isLeaf-no1: \( \text{isLeaf-pt no1 low high} \)
by auto

114
with isLeaf-no
have repchildren-eq-no-no1: (repb ∝ high) no1 = (repb ∝ high) no
  ∧ (repb ∝ low) no1 = (repb ∝ low) no
  by (simp add: null-comp-def isLeaf-pt-def)
from isLeaf-no1 isLeaf-no
have repchildren-eq-no-no1: (repa ∝ high) no1 = (repa ∝ high) no
  ∧ (repa ∝ low) no1 = (repa ∝ low) no
  by (simp add: null-comp-def isLeaf-pt-def)
from while-low-Null
have while-low-same-rep: (∀ no1∈ set prx.
  ((repa ∝ high) no1 = (repa ∝ high) no
   ∧ (repa ∝ low) no1 = (repa ∝ low) no) = (repa no = repa no1))
  by auto
show ((repa ∝ high) no1 = (repa ∝ high) no ∧
  (repa ∝ low) no1 = (repa ∝ low) no) = (repa no = repa no1)
proof (cases no1 = node)
case False
  with no1-in-take-Sucn have no1 ∈ set prx
    by auto
  with while-low-same-rep repchildren-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 ∈ set (prx@node)
    by auto
  hence no1-in-nl: no1 ∈ 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←(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 repchildren-eq-no1-no: (repb ∝ high) no1 = (repb ∝ high) no
    ∧ (repb ∝ low) no1 = (repb ∝ low) no
    by (simp add: null-comp-def isLeaf-pt-def)
  have ∀ x ∈ set (prx@node). repNodes-eq x no low high repb
    = repNodes-eq x no1 low high repb
proof (intro ballI)
  fix x
  assume x-in-take-Sucn: x ∈ set (prx@node)
  with repchildren-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 \[ sn \leftarrow (prx@\textup{node}) \cdot \text{repNodes-eq sn no low high repb} \]
\[ = \left[ sn \leftarrow (prx@\textup{node}) \cdot \text{repNodes-eq sn no1 low high repb} \right] \]
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 \textit{thesis}
by simp

qed

next

assume lno-nNull: low no \neq Null

with share-case-repb

have repbchildren-neq-no: (repa \propto low) no \neq (repa \propto 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 \propto low) no \neq (repa \propto 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
\land (\forall no1 \in set prx. ((repa \propto high) no1 = (repa \propto high) no
\land (repa \propto low) no1 = (repa \propto low) no = (repa no = 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 set (prx@\textup{node}#sfx). \neg isLeaf-pt x low high

proof (intro ballI)

fix x

assume x-in-nodeslist: x \in set (prx@\textup{node}#sfx)

from no-nLeaf isLeaf-var-no have 1 < var no
by simp

with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]

have 1 < var x
by simp

with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]

show \neg isLeaf-pt x low high
using [[simp-depth-limit = 2]]
by (auto simp add: isLeaf-pt-def)
qed
have repb-repb-no: repb (repb no) = repb no
proof –
  from repa-while-inv no-notin-nl repa-repb-nc
  have repa (repb no) = repb no
    by simp
  from hd-filter-Sucn-in-Sucn repb-no-def
  have repb-no-in-take-Sucn: repb no ∈ set (prx@[node])
    by simp
  hence repb-no-in-nl: repb no ∈ set (prx=node#sfx)
    by auto
  from all-nodes-in-nl-nLeafs repb-no-in-nl
  have repb-no-nLeaf: ¬isLeaf-pt (repb no) low high
    by auto
  from nodes-balanced-ordered [rule-format, OF repb-no-in-nl]
  have (low (repb no) = Null) = (high (repb no) = Null)
    ∧ low (repb no) ∉ set (prx@node#sfx) ∧
    high (repb no) ∉ set (prx@node#sfx)
    by auto
  from filter-take-Sucn-not-empty
  have repNodes-eq (hd [sn←(prx@[node])],
    repNodes-eq sn no low high repb] no low high repb
    by (rule hd-filter-prop)
  with repb-no-def have repNodes-eq (repb no) no low high repb
    by simp
  then have (repb ∞ low) (repb no) = (repb ∞ low) no
    ∧ (repb ∞ high) (repb no) = (repb ∞ high) no
    by (simp add: repNodes-eq-def)
  with repbchildren-neq-no have (repb ∞ low) (repb no)
    ≠ (repb ∞ high) (repb no)
    by simp
  with repb-no-in-take-Sucn repb-no-share-def
  have repb-repb-no-double-hd:
    repb (repb no) = hd [sn←(prx@[node])],
    repNodes-eq sn (repb no) low high repb]
    by auto
  from filter-take-Sucn-not-empty
  have hd [sn←(prx@[node])],
    repNodes-eq sn (repb no) low high repb] = 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 ∧
  117
(repb \propto low) no1 = (repb \propto low) no = (repb no = repb no1))

proof (intro ballI)
  fix no1
  assume no1-in-take-Sucn: no1 \in set (prx[@node])
  hence no1-in-nl: no1 \in set (prx@node#sfx)
    by auto
  from all-nodes-in-nl-nLeafs no1-in-nl
  have no1-nLeaf: \neg isLeaf-pt no1 low high
    by auto
  from all-nodes-in-nl-nLeafs no1-in-nl
  have no1-nLeaf: \neg isLeaf-pt no1 low high
    by auto
  from all-nodes-in-nl-nLeafs no1-in-nl
  have no1-nLeaf: \neg isLeaf-pt no1 low high
    by auto
  from all-nodes-in-nl-nLeafs no1-in-nl
  have no1-nLeaf: \neg isLeaf-pt no1 low high
    by auto
  have no1-nprops: (low no1 = Null) = (high no1 = Null)
    \land low no1 \notin set (prx@node#sfx) \land high no1 \notin set (prx@node#sfx)
    by auto
  show ((repb \propto high) no1 = (repb \propto high) no
    \land (repb \propto low) no1 = (repb \propto 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 \in set prx
        by auto
      with repo-while-inv have ((repo \propto high) no1 = (repo \propto high) no
        \land (repo \propto low) no1 = (repo \propto low) no) = (repo no = repo 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 repo-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 \propto high) no1 = (repb \propto high) no
            \land (repb \propto low) no1 = (repb \propto low) no
          with repbchildren-neq-no
          have (repb \propto high) no1 \neq (repb \propto low) no1
            by auto
          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 no1 low high repb
          proof
            have \forall x \in 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 ∈ 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 ∀ x ∈ 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 ∈ set (prx@node#sfx)
    from all-nodes-in-nl-nLeafs x-in-nl

119
have x-nLeaf: ¬ isLeaf-pt x low high
  by auto
from nodes-balanced-ordered [rule-format, OF x-in-nl]
have x-props: (low x = Null) = (high x = Null) ∧
  low x ∉ set (prx@node#sfx) ∧ high x ∉ 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] ≠ []
  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] ≠ []
  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 \land (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 \land (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 \neq Null
by auto
from share-cond no-nln have share-cond-or:
isLeaf-pt no low high \lor repa (low no) \neq repa (high no)
by auto
have share-reduce-if: (if (repb :\times: low) no = (repb :\times: high) no \land low no \neq Null
  then repb no = (repb :\times: low) no
else repb no = hd [sn←(prx@node)]. repNodes-eq sn no low high repa]
\[\begin{align*}
\text{repb (repb no)} &= \text{repb no} \\
\land \ (\forall \text{no} \in \text{set (prx@\{node\})}. \ ((\text{repb} \propto \text{high}) \text{no}1 = (\text{repb} \propto \text{high}) \text{no} \\
\land (\text{repb} \propto \text{low}) \text{no}1 = (\text{repb} \propto \text{low}) \text{no} = (\text{repb no} = \text{repb no}1))
\end{align*}\]

**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

\[= \text{hd [sn←(prx@\{node\}). repNodes-eq sn no low high repb]}\]

by (auto simp add: isLeaf-pt-def)

from isLeaf-no nodes-balanced-ordered [rule-format, OF no-in-nl]

have var-no: var no \leq 1

by auto

have all-nodes-nl-var-l-1: \(\forall x \in \text{set (prx@\{node\#sfx\}). var x \leq 1}\)

proof (intro ballI)

fix x

assume x-in-nl: \(x \in \text{set (prx@\{node\#sfx\})}\)

from all-nodes-same-var [rule-format, OF x-in-nl no-in-nl] var-no

show var x \leq 1

by auto

qed

have all-nodes-nl-Leafs: \(\forall x \in \text{set (prx@\{node\#sfx\}). isLeaf-pt x low high}\)

proof (intro ballI)

fix x

assume x-in-nl: \(x \in \text{set (prx@\{node\#sfx\})}\)

with all-nodes-nl-var-l-1 have var x \leq 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 =

\[= \text{hd [sn←(prx@\{node\}). repNodes-eq sn no low high repb]}\]

by auto

with hd-filter-Sucn-in-Sucn

have repb-no-in-take-Sucn: repb no \in \text{set (prx@\{node\})}

by simp

hence repb-no-in-nl: repb no \in \text{set (prx@\{node\})}

by auto

with all-nodes-nl-Leafs

have repb-no-Leaf: isLeaf-pt (repb no) low high

by auto

with repb-no-in-take-Sucn repb-no-share-def

have repb-repb-no-def: repb (repb no) =

122
hd [sn←(prx@node)]. repNodes-eq sn (repb no) low high repb
by (auto simp add: isLeaf-pt-def)
from filter-take-Sucn-not-empty
show thesis
apply (simp only: rebp-repb-no-def )
apply (simp only: rebp-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))
proof (intro ballI)
fix no1
assume no1-in-take-Sucn: no1 ∈ set (prx@node)
from no1-in-take-Sucn
have no1 ∈ set (prx@node#sfx)
by auto
with all-nodes-nl-Leafs
have isLeaf-no1: isLeaf-pt no1 low high
by auto
with rebp-no-share-def no1-in-take-Sucn
have rebp-no1-def: rebp no1 =
hd [sn←(prx@node)]. repNodes-eq sn no1 low high repb
by (auto simp add: isLeaf-pt-def)
show ((repb ∝ high) no1 = (repb ∝ high) no ∧
(repb ∝ low) no1 = (repb ∝ low) no) = (repb no = repb no1)
proof
assume rebpchildren-eq-no1-no: (repb ∝ high) no1 = (repb ∝ high) no ∧
(repb ∝ low) no1 = (repb ∝ low) no
have [sn←(prx@node)]. repNodes-eq sn no1 low high repb
= [sn←(prx@node)]. repNodes-eq sn no low high repb
proof =
have ∀ x ∈ 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 ∈ set (prx@node)
with rebpchildren-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 rebp-no1-def rebp-no-repb-def
show rebp no = repb no1
by simp

next
assume repb-no-no1: repb no = repb no1
with isLeaf-no isLeaf-no1
show (repb ∝ (high) no1 = (repb ∝ high) no
∧ (repb ∝ (low) no1 = (repb ∝ 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: ¬ isLeaf-pt no low high
with balanced-no obtain
lno-nNull: low no ≠ Null and
hno-nNull: high no ≠ Null
by (simp add: isLeaf-pt-def)
from no-nLeaf nodes-balanced-ordered [rule-format, OF no-in-nl]
have var-no: I < var no
by auto
have all-nodes-nl-var-l-1: ∀ x ∈ set (prx@node#sfx). I < var x
proof (intro ballI)
fix x
assume x-in-nl: x ∈ set (prx@node#sfx)
with all-nodes-same-var [rule-format, OF x-in-nl no-in-nl] var-no
show 1 < var x
by simp
qed

have all-nodes-nl-nLeafs: ∀ 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 1 < var x
by auto

with nodes-balanced-ordered [rule-format, OF x-in-nl] show ¬ isLeaf-pt x low high
by auto
qed

from no-nLeaf share-cond-or
have repachildren-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 repachildren-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)
proof
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 ∈ set (prx#node])
  hence no1-in-nodeslist: no1 ∈ set (prx#node#sfx)
    by auto
with all-nodes-nl-nLeafs
have no1-nLeaf: ¬ isLeaf-pt no1 low high
  by auto
show ?P no1
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
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 ∀ x ∈ 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 ∈ 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 ∈ 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#sf)
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 \((\text{repb} \propto \text{high}) \, \text{no1} = (\text{repb} \propto \text{high}) \, \text{no})\) \\
\& \((\text{repb} \propto \text{low}) \, \text{no1} = (\text{repb} \propto \text{low}) \, \text{no})\)

proof (cases \((\text{repb} \propto \text{low}) \, \text{no1} = (\text{repb} \propto \text{high}) \, \text{no1})\)

<table>
<thead>
<tr>
<th>case</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>note</td>
<td>\text{red-cond} = this</td>
</tr>
<tr>
<td>from</td>
<td>\text{no1-in-nodeslist all-nodes-nl-nLeafs}</td>
</tr>
<tr>
<td>have</td>
<td>\text{no1-nLeaf}: \neg \text{isLeaf-pt no1 low high}</td>
</tr>
<tr>
<td>by</td>
<td>auto</td>
</tr>
<tr>
<td>from</td>
<td>\text{nodes-balanced-ordered [rule-format, OF no1-in-nodeslist]}</td>
</tr>
</tbody>
</table>
| have | \text{no1-props}:  \((\text{low no1} \notin \text{set } (\text{prx}@\text{node}#sfx))\) \\
| | \& \((\text{high no1} \notin \text{set } (\text{prx}@\text{node}#sfx))\) \& (\text{low no1} = \text{Null}) = (\text{high} \\
| | no1 = \text{Null}) |
| by | auto |
| with | \text{red-cond no1-nLeaf no1-in-take-Sucn repb-no-red-def} |
| have | \text{repb-no1-def}: \text{repb no1} = (\text{repb} \propto \text{low}) \, \text{no1} |
| by | (auto simp add: isLeaf-pt-def) |
| with | \text{no1-nLeaf no1-props} have \text{repb no1 = repb (low no1)} |
| by | (simp add: null-comp-def isLeaf-pt-def) |
| from | \text{no1-props no1-nLeaf} have \text{rep (low no1) \notin \text{set } (\text{prx}@\text{node}#sfx)} |
| by | (auto simp add: isLeaf-pt-def null-comp-def) |
| with | \text{repb-repb-nc no1-props} |
| have | \text{repb (low no1) \notin \text{set } (\text{prx}@\text{node}#sfx)} |
| by | auto |
| with | \text{repb-no1-def repb-no-no1 no1-props no1-nLeaf} |
| have | \text{repb no \notin \text{set } (\text{prx}@\text{node}#sfx)} |
| by | (simp add: isLeaf-pt-def null-comp-def) |
| with | \text{repb-no-in-nl show } \text{thesis} |
| by | simp |

next

| assume | \((\text{repb} \propto \text{low}) \, \text{no1} \neq (\text{repb} \propto \text{high}) \, \text{no1})\) |
| with | \text{repb-no-share-def no1-in-take-Sucn} |
| have | \text{repb-no1-def} : \text{repb no1} = \text{hd } [\text{sn}←(\text{prx}@[\text{node}]). \text{repNodes-eq sn no1 low high repb}] |
| by | auto |
| from | \text{no1-in-take-Sucn} |
| have | \text{[sn←(prx@[node]). repNodes-eq sn no1 low high repb] \neq []} |
| apply | – |
| apply | (rule filter-not-empty) |
| apply | (auto simp add: repNodes-eq-def) |
| done |

then

| have | \text{repb-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 | \text{repb-repb-no repb-no-no1} |
| have | \text{repb-repb-eq no1 no low high repb} |
by (simp add: repNodes-eq-def)
then show ?thesis
  by (simp add: repNodes-eq-def)
qed
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


lemma (in Repoint-impl) Repoint-modifies:
  shows ∀ σ. Γ ⊢ {σ} ’p ::= PROC Repoint (’p)
           { t. t may-only-modify-globals σ in [low,high]}
  apply (hoare-rule HoarePartial.ProcRec1)
  apply (vcg spec=modifies)
  done

lemma low-high-exchange-dag:
  assumes pt-same: ∀ pt. pt ∉ set-of lt → low pt = lowa pt ∧ high pt = higha pt
  assumes pt-changed: ∀ pt ∈ set-of lt. lowa pt = (rep ∞ low) pt ∧
                        higha pt = (rep ∞ high) pt
  assumes rep-pt: ∀ pt ∈ set-of rt. rep pt = pt
  shows ∀ q. Dag q (rep ∞ low) (rep ∞ high) rt ⟷
            Dag q (rep ∞ lowa) (rep ∞ higha) rt
  using rep-pt
proof (induct rt)
case Tip thus ?case by simp
next
case (Node lrt q’ rrt)
have Dag q (rep ∞ low) (rep ∞ high) (Node lrt q’ rrt) by fact
then obtain
\( q' : q = q' \) and
\( q \text{-notNull} : q \neq \text{Null} \) and
\( \text{lr} : \text{Dag} \ ((\text{rep} \propto \text{low}) q) (\text{rep} \propto \text{low}) (\text{rep} \propto \text{high}) \text{lr} \) and
\( \text{rr} : \text{Dag} \ ((\text{rep} \propto \text{high}) q) (\text{rep} \propto \text{low}) (\text{rep} \propto \text{high}) \text{rr} \)
by auto
have rlowa-rlow: \(((\text{rep} \propto \text{lowa}) q) = ((\text{rep} \propto \text{low}) q)\)
proof (cases \( q \in \text{set-of} \text{lt} \))
    case True
    note q-in-lt=this
    with pt-changed have lowa-q: lowa \( q = (\text{rep} \propto \text{low}) q \)
    by simp
    thus (\( \text{rep} \propto \text{lowa} \) q) = (\( \text{rep} \propto \text{low} \) q)
proof (cases low q = Null)
    case True
    with lowa-q have lowa q = Null
    by simp
    with True show ?thesis
    by (simp add: null-comp-def)
next
assume lq-nNull: low q \( \neq \text{Null} \)
show ?thesis
proof (cases \( \text{rep} \propto \text{low} \) q = Null)
    case True
    with lowa-q have lowa q = Null
    by simp
    with True show ?thesis
    by (simp add: null-comp-def)
next
assume rlq-nNull: \( \text{rep} \propto \text{low} \) q \( \neq \text{Null} \)
with lrt lowa-q have lowa q \( \in \text{set-of} \text{lrt} \)
by auto
with Node.prems Node have lowa q \( \in \text{set-of} \) (Node lrt q' rrt)
by simp
with Node.prems have rep (lowa q) = lowa q
by auto
with lowa-q rlq-nNull show ?thesis
by (simp add: null-comp-def)
qed
qed
next
assume q-notin-lt: \( q \notin \text{set-of} \text{lt} \)
with pt-same have low q = lowa q
by auto
thus ?thesis
by (simp add: null-comp-def)
qed
have rhigha-rhigh: \(((\text{rep} \propto \text{higha}) q) = ((\text{rep} \propto \text{high}) q)\)
proof (cases \( q \in \text{set-of} \text{lt} \))
    case True

note  q-in-lt=this
with  pt-changed  have  higha-q: higha q = (rep ∞ high) q
by  simp
thus  ?thesis
proof  (cases high q = Null)
  case  True
  with  higha-q  have  higha q = Null
  by  (simp add: null-comp-def)
  with  True  show  ?thesis
  by  (simp add: null-comp-def)
next
assume  hq-nNull: high q ≠ Null
show  ?thesis
proof  (cases (rep ∞ high) q = Null)
  case  True
  with  higha-q  have  higha q = Null
  by  simp
  with  True  show  ?thesis
  by  (simp add: null-comp-def)
next
assume  rhq-nNull: (rep ∞ high) q ≠ Null
with  rrt  higha-q  have  higha q ∈ set-of rrt
by  auto
with  Node.prems  Node  have  higha q ∈ set-of (Node lrt q’ rrt)
by  simp
with  Node.prems  have  rep (higha q) = higha q
by  auto
with  higha-q  rhq-nNull  show  ?thesis
by  (simp add: null-comp-def)
qed
qed
next
assume  q-notin-lt:  q ∉ set-of lt
with  pt-same  have  high q = higha q
by  auto
thus  ?thesis
by  (simp add: null-comp-def)
qed
with  rrt  have  rhigha-mixed-dag:
  Dag ((rep ∞ higha) q) (rep ∞ low) (rep ∞ high) rrt
by  simp
from  lrt  rlowa-rlow  have  rlowa-mixed-dag:
  Dag ((rep ∞ lowa) q) (rep ∞ low) (rep ∞ high) lrt
by  simp
from  Node.prems  have  lrt-rep-eq:  ∀pt∈set-of lrt.  rep pt = pt
by  simp
from  Node.prems  have  rrt-rep-eq: ∀pt∈set-of rrt.  rep pt = pt
by  simp
from  rlowa-mixed-dag  lrt-rep-eq  have  lowa-lrt:
lemma (in Repoint-impl) Repoint-spec':

shows
\forall \sigma. \Gamma\vdash \{ \sigma \}
\begin{align*}
\vdash p & \triangleq \text{PROC Repoint } (p) \\
\{ \forall \text{rept. } (\text{Dag } ((\text{rep } \propto \text{id}) \sigma p) (\text{rep } \propto \sigma \text{low}) (\text{rep } \propto \sigma \text{high}) \text{rept}) \\
& \land (\forall \text{no } \in \text{set-of rept. } \sigma \text{rep no } = \text{no} ) \\
& \rightarrow \text{Dag } p \text{low } \text{high rept } \land \\
(\forall \text{pt. } \text{pt } \notin \text{set-of rept } \rightarrow \sigma \text{low pt } = \text{low pt } \land \sigma \text{high pt } = \text{high pt}) \}
\end{align*}

apply (hoare-rule HoarePartial.ProcRec1)
apply vcg
apply (rule conjI)
apply clarify
prefer 2
apply (intro impl 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 \neq \text{Null}
assume rp-nNull: rep p \neq \text{Null}
assume rec-spec-lrept:
\forall \text{rept. } \text{Dag } ((\text{rep } \propto \text{id}) \text{low } (\text{rep p})) (\text{rep } \propto \text{low}) (\text{rep } \propto \text{high}) \text{rept} \\
\land (\forall \text{no } \in \text{set-of rept. } \text{rep no } = \text{no} ) \\
\rightarrow \text{Dag } \text{pa lowa higha rept } \land \\
(\forall \text{pt. } \text{pt } \notin \text{set-of rept } \rightarrow \text{low pt } = \text{lowa pt } \land \text{high pt } = \text{higha pt}) \\
assume rec-spec-rrept:
∀rept. Dag (((rep χ id) (higha (rep p))) (rep χ lowa(rep p := pa))) (rep χ higha) rept
∧ (∀no∈set-of rept. rep no = no)
→ Dag pb lowb highb rept ∧

assume rept-dag: Dag (((rep χ id) p) (rep χ low) (rep χ high) rept
assume rno-rept: ∀no∈set-of rept. rep no = no
show Dag (rep p) lowb (highb(rep p := pb)) rept ∧

proof −
from rp-nNull rept-dag p-nNull obtain lrept rrept where
    rep-def: rept = Node lrept (rep p) rrept
    by auto
with rept-dag p-nNull have lrept-dag:
    Dag (((rep χ low) (rep p))) (rep χ low) (rep χ high) lrept
    by simp
from rep-def rept-dag p-nNull have rrept-dag:
    Dag (((rep χ high) (rep p))) (rep χ low) (rep χ 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 χ id) (low (rep p))) (rep χ low) (rep χ 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


by simp
from lrept-dag repoint-post-low obtain
    pa-def: pa = (rep χ low) (rep p) and
    lowa-higha-def: (∀no∈set-of lrept. lowa no = (rep χ low) no ∧ higha no = (rep χ 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 Dag ((\(\text{rep} \Join \text{id}\)) (\text{higha}\ (\text{rep}\ p))) (\text{rep} \Join \text{lowa}(\text{rep}\ p := \text{pa})) (\text{rep} \Join \text{higha})

proof
  from low-lowa-nc rp-notin-lrept have (\text{rep} \Join \text{high}) (\text{rep} p) = (\text{rep} \Join \text{higha}) (\text{rep} p)
  by (auto simp add: null-comp-def)
with rrept-dag have higha-mixed-rrept:
  Dag ((\text{rep} \Join \text{id}) (\text{higha}\ (\text{rep}\ p))) (\text{rep} \Join \text{low}) (\text{rep} \Join \text{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 ((\text{rep} \Join \text{id}) (\text{higha}\ (\text{rep}\ p))) (\text{rep} \Join \text{lowa}) (\text{rep} \Join \text{higha}) rrept
apply –
apply (rule low-high-exchange-dag)
apply auto
done

have Dag ((\text{rep} \Join \text{id}) (\text{higha}\ (\text{rep}\ p))) (\text{rep} \Join \text{lowa}(\text{rep}\ p := \text{pa})) (\text{rep} \Join \text{higha}) rrept

proof –
  have \(\forall\ no \in\ \text{set-of}\ rrept.\ (\text{rep} \Join \text{lowa}) no = (\text{rep} \Join \text{lowa}(\text{rep}\ p := \text{pa})) no\)
^ (\text{rep} \Join \text{higha}) no = (\text{rep} \Join \text{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 (\text{rep} \Join \text{lowa}) no = (\text{rep} \Join \text{lowa}(\text{rep}\ p := \text{pa})) no \land
      (\text{rep} \Join \text{higha}) no = (\text{rep} \Join \text{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 \land
  (\(\forall\ pt.\ pt \notin set-of\ rrept\ \rightarrow\ (\text{lowa}(\text{rep}\ p := \text{pa})) pt = \text{lowb}\ pt \land \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: (∀ pt. pt ∉ set-of rrept →
(lowa(rep p := pa)) pt = lowb pt ∧ higha pt = highb pt)
by simp
from repoint-rrept rrept-dag obtain
pb-def: pb = ((rep ∞ high) (rep p)) and
lowb-highb-def: (∀ no ∈ set-of rrept. lowb no = (rep ∞ low) no ∧ highb no =
(rep ∞ 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 ∀ no ∈ set-of rept. lowb no = (rep ∞ low) no ∧ (highb(rep p := pb))
no = (rep ∞ high) no
proof
fix no
assume no-in-rept: no ∈ set-of rept
show lowb no = (rep ∞ low) no ∧ (highb(rep p := pb)) no = (rep ∞ high)
proof (cases no ∈ set-of rrept)
case True
with lowb-highb-def pb-def show ?thesis
by simp
next
assume no-notin-rept: no ∉ set-of rrept
show ?thesis
proof (cases no ∈ set-of lrept)
case True
with no-notin-rept rp-notin-lrept ab-nc
have ab-nc-no: lowa no = lowb no ∧ 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 ∞ low) no ∧ higha no = (rep ∞ high) no
by auto
with ab-nc-no have lowb no = (rep ∞ low) no ∧ highb no = (rep ∞ high) no
high) no
by simp
with rp-notin-lrept True show ?thesis
apply (subgoal-tac no \neq rep p)
apply simp
apply blast
done
next
assume no-notin-lrept: no \notin 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 \land high no = higha no
by auto
from rp-notin-rrept no-rp ab-nc have
(lowa(rep p := pa)) no = lowb no \land higha no = highb no
by auto
with a-nc pa-def no-rp have (rep \propto low) no = lowb no \land high no =
highb no
by auto
with pb-def no-rp show ?thesis
by simp
qed
qed
with rept-dag have Dag (rep p) lowb (highb(rep p := pb)) rept =
Dag (rep p) (rep \propto low) (rep \propto 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 (\forall pt. pt \notin set-of rept \rightarrow low pt = lowb pt \land high pt = (highb(rep p :=
pb)) pt)
proof (intro allI impI)
fix pt
assume pt-notin-rept: pt \notin set-of rept
with rept-def obtain
pt-notin-lrept: pt \notin set-of lrept and
pt-notin-rrept: pt \notin set-of rrept and
pt-neq-rp: pt \neq rep p
by simp
with low-lowa-nc ab-nc show low pt = lowb pt \land 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
  ∀ σ. Γ ⊢ { σ. Dag ((rep ∞ id) p) (rep ∞ low) (rep ∞ high) rept ∧ (∀ no ∈ set-of rept. rep no = no) } p := PROC Repoint (p)
{ Dag 'p 'low 'high rept ∧ (∀ pt. pt ∈ set-of rept --> σlow pt = 'low pt ∧ σhigh 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 clarlsimp
apply clarify
proof −
  fix rept low high rep p
assume rept-dag: Dag ((rep ∞ id) p) (rep ∞ low) (rep ∞ high) rept
assume rno-rept: ∃ no ∈ set-of rept. rep no = no
assume p-nNull: p ≠ Null
assume rp-nNull: rep p ≠ Null
show ∃ lrept.
  Dag ((rep ∞ id) (low (rep p))) (rep ∞ low) (rep ∞ high) lrept ∧
  (∀ no ∈ set-of lrept. rep no = no) ∧
  (∀ lowa higha pa.
    Dag pa lowa higha lrept ∧
    (∀ pt. pt ∈ set-of lrept --> low pt = lowa pt ∧ high pt = higha pt) -->
    (∃ lrept.
      Dag ((rep ∞ id) (higha (rep p))) (rep ∞ lowa(rep p := pa))
      (rep ∞ higha) lrept ∧
      (∀ no ∈ set-of lrept. rep no = no) ∧
      (∀ lowb highb pb.
        Dag pb lowb highb lrept ∧
        (∀ pt. pt ∈ set-of lrept -->
          (lowa(rep p := pa)) pt = lowb pt ∧
          higha pt = highb pt) -->
        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 rept-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 lrept-dag:
  ∀ no ∈ set-of lrept. rep no = no
  by auto
from rno-rept rept-def have rrept-dag:
  ∀ no ∈ set-of rrept. 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
proof -
case (goal1 lowa higha pa)
  have lrepta:
    Dag pa lowa higha lrept
  by fact
  have low-lowa-nc:
    ∀ pt. pt /∈ set-of lrept /
    low pt = lowa pt ∧ high pt = higha pt
  by fact
from rept-dag lrepta obtain
  pa-def: pa = (rep ∞ low) (rep p) and
  lowa-higha-def: ∀ no ∈ set-of lrept.
  lowa no = (rep ∞ low) no ∧ higha no = (rep ∞ 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 /∈ set-of lrept
  by simp
from rept-DAG rept-def have rp-notin-rrept: rep p /∈ set-of rrept
  by simp
have rrepta:
  Dag ((rep ∞ id) (higha (rep p)))
  (rep ∞ lowa(rep p := pa)) (rep ∞ higha) rrept
proof -
from low-lowa-nc rp-notin-lrept
  have (rep ∞ high) (rep p) = (rep ∞ higha) (rep p)
    by (auto simp add: null-comp-def)
with rrept-dag have higha-mixed-rrept:
  Dag ((rep ∞ id) (higha (rep p))) (rep ∞ low) (rep ∞ 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 \otimes id) (higha (rep p))) (rep \otimes lowa) (rep \otimes higha) rrept
  apply –
  apply (rule low-high-exchange-dag)
  apply auto
  done
have Dag ((rep \otimes id) (higha (rep p))) (rep \otimes lowa) (rep \otimes higha) rrept = Dag ((rep \otimes id) (higha (rep p))) (rep \otimes lowa(rep p := pa)) (rep \otimes higha) rrept
proof –
  have \forall no \in \text{set-of rrept}.
    (rep \otimes lowa) no = (rep \otimes lowa(rep p := pa)) no \land
    (rep \otimes higha) no = (rep \otimes higha) no
  proof
    fix no
    assume no-in-rrept: no \in \text{set-of rrept}
    with rp-notin-rrept have no \neq rep p
    by blast
    thus (rep \otimes lowa) no = (rep \otimes lowa(rep p := pa)) no \land
      (rep \otimes higha) no = (rep \otimes 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 \?case
  apply (rule-tac x=rrept in exI)
  apply (rule conjI)
  apply (rule rrepta)
  apply (rule conjI)
  apply (rule rno-rrept)
  apply clarify
proof –
case (goal1 lowb highb pb)
  have rrepth: Dag pb lowb highb rrept by fact
  have ab-nc: \forall pt. pt \notin \text{set-of rrept} \rightarrow
    (lowa(rep p := pa)) pt = lowb pt \land higha pt = highb pt by fact
  from rrepth rrept-dag obtain
    pb-def: pb = ((rep \otimes high) (rep p)) and
    lowb-highb-def: \forall no \in \text{set-of rrept}.
      lowb no = (rep \otimes low) no \land highb no = (rep \otimes high) no
  apply –
  apply (drule Dags-eq-hp-eq)
  apply auto
  done

138
have \textit{rept-end-dag}: \textit{Dag \textbf{(rep p)} lowb \textbf{(highb\textbf{(rep p := pb)})) \textbf{rept}}

\textbf{proof} —

\textbf{have} \forall \textit{no} \in \textit{set-of \textbf{rept}}:\
\begin{align*}
\textit{lowb \textbf{no}} &= \textbf{(rep} \times \textbf{low\textbf{)}} \textit{no} \land \textbf{(highb\textbf{(rep p := pb}) \textbf{no}} = \textbf{(rep} \times \textbf{high\textbf{)}} \textbf{no}
\end{align*}

\textbf{proof} (\textbf{cases} \textit{no} \in \textit{set-of \textit{rrrept}})

\textbf{case} \textbf{True}

\textbf{with} \textit{lowb-highb-def pb-def} \textbf{show} \textbf{?thesis}

\textbf{by simp}

\textbf{next}

\textbf{assume} \textit{no-notin-rrrept}: \textit{no} \notin \textit{set-of \textit{rrrept}}

\textbf{show} \textbf{?thesis}

\textbf{proof} (\textbf{cases} \textit{no} \in \textit{set-of \textit{rrrept}})

\textbf{case} \textbf{True}

\textbf{with} \textit{no-notin-rrrept \textit{rp-notin-rrrept ab-nc}}

\textbf{have} \textit{ab-nc-no}: \textit{lowa \textbf{no}} = \textbf{lowb \textbf{no}} \land \textbf{higha \textbf{no}} = \textbf{highb \textbf{no}}

\textbf{apply} —

\textbf{apply} (erule-tac \textit{x=no in allE})

\textbf{apply} (erule impE)

\textbf{apply simp}

\textbf{apply} (subgoal-tac \textit{no} \neq \textit{rep p})

\textbf{apply simp}

\textbf{apply blast}

\textbf{done}

\textbf{from \textit{lowa-higha-def True have}}

\begin{align*}
\textit{lowa \textbf{no}} &= \textbf{(rep} \times \textbf{low\textbf{)}} \textit{no} \land \textbf{higha \textbf{no}} = \textbf{(rep} \times \textbf{high\textbf{)}} \textbf{no}
\end{align*}

\textbf{by auto}

\textbf{with} \textit{ab-nc-no}

\textbf{have} \textit{lowb \textbf{no}} = \textbf{(rep} \times \textbf{low\textbf{)}} \textit{no} \land \textbf{highb \textbf{no}} = \textbf{(rep} \times \textbf{high\textbf{)}} \textbf{no}

\textbf{by simp}

\textbf{with \textit{rp-notin-rrrept True show \textbf{?thesis}}}

\textbf{apply} (subgoal-tac \textit{no} \neq \textit{rep p})

\textbf{apply simp}

\textbf{apply blast}

\textbf{done}

\textbf{next}

\textbf{assume} \textit{no-notin-rrrept}: \textit{no} \notin \textit{set-of \textit{rrrept}}

\textbf{with} \textit{no-in-rrrept \textit{rept-def no-notin-rrrept have \textit{no-rp}: \textit{no} = \textit{rep p}}}

\textbf{by simp}

\textbf{with \textit{rp-notin-rrrept low-lowa-nc}}

\textbf{have a-nc}: \textit{low \textbf{no}} = \textbf{lowa \textbf{no}} \land \textbf{high \textbf{no}} = \textbf{higha \textbf{no}}

\textbf{by auto}

\textbf{from \textit{rp-notin-rrrept no-rp ab-nc}}

\textbf{have} (\textit{lowa(rep \textit{p} := \textit{pa}) \textbf{no}} = \textbf{lowb \textbf{no}} \land \textbf{higha \textbf{no}} = \textbf{highb \textbf{no}}

139
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
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-rept: pt /∈ set-of lrept and
    pt-notin-rept: 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 ?case
by simp
qed
qed
qed
qed

lemma (in Repoint-impl) Repoint-spec-total:
shows
∀ σ φ, Γ, l. σ. Dag (′rep ∞ id) `p) (′rep ∞ low) (′rep ∞ high) rept
∧ (∀ no ∈ set-of rept. ′rep no = no) l
′p := PROC Repoint (′p)
{Dag `p low high rept ∧
(∀ pt. pt /∈ set-of rept → σlow pt = low pt ∧ σhigh pt = high pt)}`
apply (hoare-rule HoareTotal.ProcRec1

140
where \( r = \text{measure} (\lambda (s, p), \text{size} (\text{dag} (((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) p) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%))) \)

apply vcg
apply (rule conjI)
prefer 2
apply (clarsimp simp add: null-comp-def)
apply clarify
apply (rule conjI)
prefer 2
apply clarsimp
proof
  fix rept low high rep p
assume rept-dag: \( \text{Dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) p) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \)
assume rno-rept: \( \forall no \in \text{set-of} \text{rept}. \text{rep no} = no \)
assume p-nNull: \( p \neq \text{Null} \)
assume rp-nNull: \( \text{rep p} \neq \text{Null} \)
show \( \exists \text{lrept}. \text{Dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%)) lrept \land (\forall no \in \text{set-of} \text{lrept}. \text{rep no} = no) \land 
\text{size} (\text{dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) p) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \land 
(\forall lowa higha pa. 
  \text{Dag} pa lowa higha trept \land 
  (\forall pt. pt \notin \text{set-of} trept \rightarrow 
    \text{low pt} = lowa pt \land \text{high pt} = higha pt) \rightarrow 
  (\exists rrept. 
    \text{Dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) rrept \land 
    (\forall no \in \text{set-of} \text{rrept}. \text{rep no} = no) \land 
    \text{size} (\text{dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \text{higha} (\text{rep p})) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \land 
    (\forall lowb highb pb. 
      \text{Dag} pb lowb highb \text{rrept} \land 
      (\forall pt. pt \notin \text{set-of} \text{rrept} \rightarrow 
        \text{lowa}(\text{rep p} := pa)) pt = lowb pt \land 
        \text{higha} pt = highb pt) \rightarrow 
      \text{Dag} (\text{rep p}) lowb (\text{highb}(\text{rep p} := pb)) \text{rrept} \land 
      (\forall pt. pt \notin \text{set-of} \text{rept} \rightarrow 
        \text{low pt} = lowb pt \land 
        \text{high pt} = (\text{highb}(\text{rep p} := pb)) pt)))")
)

proof
  from rp-nNull rept-dag p-nNull obtain trept rrept where
  rept-def: \( \text{rept} = \text{Node} \text{trept} (\text{rep p}) \text{rrept} \)
by auto
with rept-dag p-nNull have trept-dag: \( \text{Dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \text{trept} \land 
    (\forall no \in \text{set-of \text{trept}}. \text{rep no} = no) \land 
    \text{size} (\text{dag} ((\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) p) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) (\% \% \% \% \% \% \% \% \% \% \% \% \% \% \% \%) \land 
    (\forall lowb highb pb. 
      \text{Dag} pb lowb highb rrept \land 
      (\forall pt. pt \notin \text{set-of \text{rrept}} \rightarrow 
        \text{lowa}(\text{rep p} := pa)) pt = lowb pt \land 
        \text{higha} pt = highb pt) \rightarrow 
      \text{Dag} (\text{rep p}) lowb (\text{highb}(\text{rep p} := pb)) \text{rrept} \land 
      (\forall pt. pt \notin \text{set-of \text{rept}} \rightarrow 
        \text{low pt} = lowb pt \land 
        \text{high pt} = (\text{highb}(\text{rep p} := pb)) pt)))")

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

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

proof −
  case (goal1 lowa higha pa)
  have lrepta: Dag pa lowa higha lrept by fact
  have low-lowa-nc:
    ∀ pt. pt /∈ set-of lrept → low pt = lowa pt ∧ high pt = higha pt by fact
  from lrept-dag lrepta obtain
    pa-def: pa = (rep ⊳ low) (rep p) and
    lowa-higha-def: ∀ no ∈ set-of lrept.
    lowa no = (rep ⊳ low) no ∧ higha no = (rep ⊳ 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 /∈ set-of lrept
by simp

from rept-DAG rept-def have rp-notin-rrept: rep p /∈ set-of rrept
by simp

have rrepta: Dag ((rep ⊳ id) (higha (rep p)))
(   (rep ⊳ lowa(rep p := pa)) (rep ⊳ higha) ) rrept

proof −
  from low-lowa-nc rp-notin-lrept
  have (rep ⊳ high) (rep p) = (rep ⊳ higha) (rep p)
  by (auto simp add: null-comp-def)

with rrept-dag have higha-mixed-rrept:
  Dag ((rep ⊳ id) (higha (rep p))) (rep ⊳ low) (rep ⊳ 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:
\[\text{Dag (}(\text{rep } \times \text{id}) (\text{higha (rep p)})) (\text{rep } \times \text{lowa}) (\text{rep } \times \text{higha}) \text{ rrept}\]

apply –
apply (rule low-high-exchange-dag)
apply auto
done

have \[\text{Dag (}(\text{rep } \times \text{id}) (\text{higha (rep p)})) (\text{rep } \times \text{lowa}) (\text{rep } \times \text{higha}) \text{ rrept} =
\text{Dag (}(\text{rep } \times \text{id}) (\text{higha (rep p)})) \]
(\text{rep } \times \text{lowa(rep p := pa)}) (\text{rep } \times \text{higha}) \text{ rrept}

proof –
have \(\forall \text{ no } \in \text{ set-of rrept.}\)
(\text{rep } \times \text{lowa}) \text{ no} = (\text{rep } \times \text{lowa(rep p := pa)}) \text{ no} \land
(\text{rep } \times \text{higha}) \text{ no} = (\text{rep } \times \text{higha}) \text{ 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 (\text{rep } \times \text{lowa}) \text{ no} = (\text{rep } \times \text{lowa(rep p := pa)}) \text{ no} \land
(\text{rep } \times \text{higha}) \text{ no} = (\text{rep } \times \text{higha}) \text{ 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 \?case
apply (rule-tac x=rrept in ex1)
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
proof –
case (goal1 lowb highb pb)
have rrepb: Dag pb lowb highb rrept by fact
have ab-noc: \(\forall \text{ pt. pt } \notin\) set-of rrept \(\rightarrow\)
(\text{lowa(rep p := pa)}) \text{ pt} = lowb \text{ pt} \land higha \text{ pt} = highb \text{ pt} by fact
from rrepb rrept-day obtain
pb-def: pb = (\text{((rep } \times \text{high) (rep p))}) and
lowb-highb-def: \(\forall \text{ no } \in\) set-of rrept.
lowb \text{ no} = (\text{rep } \times \text{low}) \text{ no} \land highb \text{ 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 \( \forall no \in \text{set-of} \text{rept.} \)
    \[ \text{lowb } no = (\text{rep } \& \text{low}) \text{ no } \land (\text{highb(rep p := pb)}) \text{ no } = (\text{rep } \& \text{high}) \text{ no} \]
no
proof
  fix no
assume no-in-rept:  no \in \text{set-of} \text{rept}
show lowb no = (\text{rep } \& \text{low}) \text{ no } \land 
  (\text{highb(rep p := pb)}) \text{ no } = (\text{rep } \& \text{high}) \text{ no}
proof (cases no \in \text{set-of} \text{rrept})
case True
  with lowb-highb-def pb-def show \text{thesis}
  by simp
next
assume no-notin-rrept:  no \notin \text{set-of} \text{rrept}
show \text{thesis}
proof (cases no \in \text{set-of} \text{lrept})
case True
  with no-notin-rrept rp-notin-lrept ab-nc
  have ab-nc-no:  lowa no = lowb no \land higha no = highb no
  apply —
  apply (erule-tac x = no in allE)
  apply (erule impE)
  apply simp
  apply (subgoal-tac no \neq rep p)
  apply simp
  apply blast
  done
from lowa-higha-def True have
  lowa no = (\text{rep } \& \text{low}) \text{ no } \land 
  higha no = (\text{rep } \& \text{high}) \text{ no}
  by auto
with ab-nc-no
  have lowb no = (\text{rep } \& \text{low}) \text{ no } \land highb no = (\text{rep } \& \text{high}) \text{ no}
  by simp
with rp-notin-lrept True show \text{thesis}
  apply (subgoal-tac no \neq rep p)
  apply simp
  apply blast
  done
next
assume no-notin-lrept:  no \notin \text{set-of} \text{lrept}
with no-in-rept rept-def no-notin-rept have no-rp:  no = rep p
  by simp
with rp-notin-lrept low-lowa-nc
  have a-nc:  low no = lowa no \land high no = higha no
  by auto
from rp-notin-rept 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 ?case
   by simp
qed
qed
qed
qed
end

10 Proof of Procedure Normalize

theory NormalizeTotalProof imports LevellistProof ShareReduceRepListProof
   RepointProof begin

   145


hide-const (open) \text{DistinctTreeProver.\textit{set-of tree}.Node tree. Tip}

lemma \text{(in Normalize-impl) Normalize-modifies:}
\begin{align*}
\forall \sigma. \Gamma \vdash \sigma \ {p} := \text{PROC Normalize (}'p') \\
\{ t. t \text{ may-only-modify-globals } \sigma \text{ in [rep,mark,low,high,next]}\}
\end{align*}
apply \text{(hoare-rule HoarePartial.ProcRec1)}
apply \text{(veg spec=modifies)}
done

lemma \text{(in Normalize-impl) Normalize-spec:}
\begin{align*}
\forall \sigma \ \text{pret prebdt. } \Gamma \vdash_t \\
\{\sigma. \text{Dag } 'p '\text{low 'high pret }\land \text{ordered pret 'var }\land \\
 'p \neq \text{Null }\land (\forall n. n \in \text{set-of pret }\rightarrow \text{mark } n = \text{mark 'p}) \land \\
\text{bdt pret 'var }= \text{Some prebdt}\} \\
 'p := \text{PROC Normalize(}'p') \\
\{ (\forall pt. pt \notin \text{set-of pret } \\
 \rightarrow \sigma_{\text{rep}} pt = \text{rep pt }\land \sigma_{\text{low}} pt = \text{low pt }\land \sigma_{\text{high}} pt = \text{high pt }\land \\
 \sigma_{\text{mark}} pt = \text{mark pt }\land \sigma_{\text{next pt}} = \text{'next pt}) \land \\
\text{(\exists postt. Dag } 'p '\text{low 'high postt }\land \text{reduced postt }\land \\
\text{shared postt 'var }\land \text{ordered postt 'var }\land \\
\text{set-of postt }\subseteq \text{set-of pret }\land \\
(\exists \text{postbdt. bdt postt 'var }= \text{Some postbdt }\land \text{prebdt }\sim \text{postbdt})) \land \\
(\forall no. no \in \text{set-of pret }\rightarrow \text{mark no }= (\neg \sigma_{\text{mark 'p}})) \} \}
\end{align*}
apply \text{(hoare-rule HoareTotal.ProcNoRec1)}

apply \text{(hoare-rule anno=)
\begin{align*}
\text{levellist} &:= \text{replicate (}'p\rightarrow '\text{var }+ 1') \text{ Null};; \\
\text{levellist} &:= \text{CALL Levellist (}'p, ('\neg 'p\rightarrow '\text{mark})', 'levellist');; \\
(\text{ANNO} (\tau, lll). \{ r. \text{Levellist 'levellist 'next ll } \land \\
\text{Dag } \sigma_{\tau} 'p '\text{low 'high pret }\land \text{ordered pret 'var }\land \sigma_{\tau} p \neq \text{Null }\land \\
(\text{bdt pret 'var }= \text{Some prebdt}) \land \text{wf-ll pret ll 'var }\land \\
\text{length 'levellist }= ((\sigma_{\tau} p \rightarrow \sigma_{\tau} \text{var }) + 1) \land \\
\text{wf-marking pret 'mark 'mark (}\neg \sigma_{\text{mark 'p}}) \land \\
(\forall pt. pt \notin \text{set-of pret }\rightarrow \sigma_{\text{next pt}} = \text{'next pt}) \land \\
\text{low }= \sigma_{\text{low}} \land \text{high }= \sigma_{\text{high}} \land 'p = \sigma_{\tau} p \land '\text{rep }= \sigma_{\text{rep}} \land 'var = \sigma_{\text{var}} \\
'n := 0;; \\
\text{WHILE ('n < length 'levellist) }
\text{INV } \{\text{levellist 'levellist 'next ll } \land \\
\text{Dag } \sigma_{\tau} 'p '\text{low 'high pret }\land \text{ordered pret 'var }\land \sigma_{\tau} p \neq \text{Null }\land \\
(\text{bdt pret 'var }= \text{Some prebdt}) \land \text{wf-ll pret ll 'var }\land \\
\text{length 'levellist }= ((\sigma_{\tau} p \rightarrow \sigma_{\tau} \text{var }) + 1) \land \\
\text{wf-marking pret 'mark 'mark (}\neg \sigma_{\text{mark 'p}}) \land \\
\text{low }= \sigma_{\text{low}} \land \text{high }= \sigma_{\text{high}} \land 'p = \sigma_{\tau} p \land '\text{rep }= \sigma_{\text{rep}} \land \text{\tau var }= \sigma_{\tau} var \land 'n \leq \text{length 'levellist }\land \\
(\forall pt i. (pt \notin \text{set-of pret }\lor ('n <= i \land pt \in \text{set (ll ! i)}) \land \\
i <\text{length 'levellist }) \\
\rightarrow \sigma_{\text{rep}} pt = \text{rep pt}) \} \land
\end{align*}

146
apply simp \(\sigma\) var <\(\leq\) \(\sigma\) var ∧

\((\exists\ \text{nort}.\ \text{Dag}\ (\text{rep no})\ (\text{rep } \sigma_{\text{low}})\ (\text{rep } \sigma_{\text{high}})\ \text{nort ∧} \text{Dag no } \sigma_{\text{low}} \sigma_{\text{high}} \text{ not} \∧ \text{reduced nort} \∧ \text{ordered nort} \sigma_{\text{var}} \∧ \text{set-of nort} \subseteq \text{rep ' Nodes } \text{n ll} \∧ \text{(∀ no } \in \text{ set-of nort}.\ \text{rep no } = \text{no}) \∧ \text{(∃ nobdt norbdlt. bdt not } \sigma_{\text{var}} \text{ Some nobdt} \∧ \text{bdt nort } \sigma_{\text{var}} = \text{Some norbdlt } \∧ \text{nobdt } \sim \text{ norbdlt}) \)) \∧ 

\((\forall t1 t2.\ t1 \in \text{Dags} (\text{rep ' (Nodes } \text{n ll})) (\text{rep } \sigma_{\text{low}}) (\text{rep } \sigma_{\text{high}})\∧ t2 \in \text{Dags} (\text{rep ' (Nodes } \text{n ll})) (\text{rep } \sigma_{\text{low}}) (\text{rep } \sigma_{\text{high}})\rightarrow \text{isomorphic-dags-eq t1 t2 } \sigma_{\text{var}}) \∧ \text{levellist } = \text{levellist } \∧ \text{next } = \text{next } \∧ \text{mark } = \text{mark } \∧ \text{low } = \sigma_{\text{low}} \land \text{high } = \sigma_{\text{high}} \∧ \text{p } = \sigma_{\text{p}} \land \text{var } = \sigma_{\text{var}} \}) \}

\text{VAR MEASURE } (\text{length } \text{levellist } - \text{'}n\text{')}

\text{DO}
\text{CALL ShareReduceRepList(levellist ! 'n)\text{;}}$
\text{'}n : = \text{'}n + 1$
\text{OD}
\text{\{((∃ postnormt. \text{Dag} (\text{rep } \sigma_{\text{p}}) (\text{rep } \sigma_{\text{low}}) (\text{rep } \sigma_{\text{high}}) \text{ postnormt} \∧ \text{reduced postnormt} \∧ \text{shared postnormt } \sigma_{\text{var}} \∧ \text{ordered postnormt } \sigma_{\text{var}} \∧ \text{set-of postnormt} \subseteq \text{set-of pret} \∧ (∃ postnormbdlt. \text{bdt postnormbdlt } \∧ \text{prebdt } \sim \text{ postnormbdlt}) \∧ (\forall no \in \text{ set-of postnormt}.\ (\text{rep no } = \text{no})) \∧ \text{ordered pret } \sigma_{\text{var}} \∧ \sigma_{\text{p}} \neq \text{Null} \∧ (\forall pt.\ pt \notin \text{ set-of pret }\rightarrow \text{rep pt } = \text{rep pt}) \∧ \text{levellist } = \text{levellist } \∧ \text{next } = \text{next } \∧ \text{mark } = \text{mark } \land \text{low } = \sigma_{\text{low}} \land \text{high } = \sigma_{\text{high}} \land \text{p } = \sigma_{\text{p}} \land (\forall no. no \in \text{ set-of pret }\rightarrow \text{mark no } = (\neg \text{mark } \sigma_{\text{p}})) \}) \}\}

\text{\texttt{;}}$
\text{'}p : = \text{CALL Repoint ('p) in HoareTotal.annotateI)}$
\text{apply (eq spec=spec-total)}$
\text{prefer 2}$

\text{apply (simp add: Nodes-def null-comp-def)
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:
  ∀ pt i. pt ∉ set-of pret ∨ nb ≤ i ∧ pt ∈ set (ll ! i) ∧
  i < length levellista
  → rep pt = repb pt

assume wf-marking-prop: wf-marking pret mark marka (∼ mark p)
assume pret-dag: Dag p low high pret
assume prebdt: bdt pret var = Some prebdt
assume not-nbslla: ∼ nb < length levellista

148
assume \( nb \leq \) length levellista

assume normalize-prop: \( \forall \) no \( \in \) Nodes nb ll.
\[ \text{var (repb no)} \leq \text{var no} \wedge \]
\( (\exists \) not nort. Dag (repb no) (repb \( \propto \) low) (repb \( \propto \) 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} \cdot \text{Nodes nb ll} \wedge \]
\( (\forall \) no \( \in \) set-of nort. repb no = no) \( \wedge \)
\( (\exists \) nobdt norbdt. bdtd not var = Some nobdt \( \wedge \)
bdt not var = Some norbdt \( \wedge \) nobdt \( \sim \) norbdt))

assume repbNodes-in-Nodes: \( \text{repb} \cdot \text{Nodes nb ll} \subseteq \text{Nodes nb ll} \)

assume shared-mult-dags:
\[ \forall t1 t2. t1 \in \text{Dags} (\text{repb} \cdot \text{Nodes nb ll}) (\text{repb} \propto \text{low}) (\text{repb} \propto \text{high}) \wedge \]
t2 \( \in \) Dags (\text{repb} \cdot \text{Nodes nb ll}) (\text{repb} \propto \text{low}) (\text{repb} \propto \text{high})
\( \rightarrow \) isomorphic-dags-eq t1 t2 var

show \( \exists \) postnormt. Dag (repb p) (repb \propto \text{low}) (repb \propto \text{high}) postnormt \wedge
\( \text{reduced postnormt} \wedge \text{shared postnormt var} \wedge \]
\( \text{ordered postnormt var} \wedge \text{set-of postnormt} \subseteq \text{set-of pret} \wedge \]
(\exists postnormbdt.
\( \text{bdt postnormt var} = \text{Some postnormbdt} \wedge \)
\( \text{prebdt} \sim \text{postnormbdt} \)) \( \wedge \)
\( (\forall \) pt. pt \( \notin \) set-of pret \( \rightarrow \) rep pt = repb pt) \( \wedge \)
\( (\forall \) no. no \( \in \) set-of pret \( \rightarrow \) marka no = (\text{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 \( \notin \) set-of pret \( \rightarrow \) 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 \( \in \) set-of pret \( \rightarrow \) marka no = (\text{mark} p))
by (simp add: wf-marking-def)
from not-nbslla nb-le-lla have nb-length-lla: nb = length levellista
by simp
with length-lla have varp-s-nb: var p < nb
by simp
from pret-def have p-in-pret: p \( \in \) set-of pret
by simp
with wf-lla have p \( \in \) set (ll ! (var p))
by (simp add: wf-lla-def)
with varp-s-nb have p-in-Nodes: p \( \in \) Nodes nb ll
by (auto simp add: Nodes-def)
with normalize-prop obtain not nort where
\( \text{varrepmno-varn} \cdot \text{var (repb p)} \leq \text{var p} \)
\( \text{and nort-dag: Dag (repb p) (repb} \propto \text{low}) (repb} \propto \text{high}) \text{nort and} \)
\( \text{not-dag: Dag p low high not and} \)
\textbf{red-nort:} reduced nort \textbf{and}
\textbf{ord-nort:} ordered nort var \textbf{and}
\textbf{nort-in-repNodes:} set-of nort \subseteq \text{repb · Nodes nb ll} \textbf{and}
\textbf{nort-repb:} (\forall no \in \text{set-of nort}. \text{repb no} = no) \textbf{and}
\textbf{bdt-prop:} \exists \text{norbdt norbdt}. \text{bdt not var} = \text{Some norbdt} \land \text{bdt nort var} = \text{Some norbdt} \land 
\text{norbdt} \land 
\text{nobdt} \sim \text{norbdt}
by auto

from \text{wf-lla nb-length-lla} have \text{Nodes-in-pret:} \text{Nodes nb ll} \subseteq \text{set-of pret}
apply –
apply (rule \text{Nodes-in-pret})
apply (auto simp add: \text{length-ll-eq})
done
from \text{pret-dag wf-lla nb-length-lla} have \text{Null} \notin \text{Nodes nb ll}
apply –
apply (rule \text{Null-notin-Nodes})
apply (auto simp add: \text{length-ll-eq})
done
with \text{p-in-Nodes repbNodes-in-Nodes} have \text{rp-nNull:} \text{repb p} \neq \text{Null}
by auto
with \text{nort-dag} have \text{nort-nTip:} \text{nort} \neq \text{Tip}
by auto
have \exists \text{postnormt}. \text{Dag (repb p) (repb \propto low) (repb \propto high) postnormt} \land 
\text{reduced postnormt} \land \text{shared postnormt var} \land 
\text{ordered postnormt var} \land \text{set-of postnormt} \subseteq \text{set-of pret} \land 
(\exists \text{postnormbdt}.
\text{bdt postnormt var} = \text{Some postnormbdt} \land \text{prebdt} \sim \text{postnormbdt}) \land 
(\forall no \in \text{set-of postnormt}. \text{repb no} = no)
proof (rule-tac \text{exI})
from \text{nort-in-repNodes repbNodes-in-Nodes Nodes-in-pret} have \text{nort-in-pret:} \text{set-of nort} \subseteq \text{set-of pret}
by blast
from \text{not-dag pret-dag} have \text{not-pret:} \text{not} = \text{pret}
by (simp add: \text{Dag-unique})
with \text{bdt-prop prebdt} have \text{pret-bdt-prop:}
\exists \text{postnormbdt}.
\text{bdt nort var} = \text{Some postnormbdt} \land \text{prebdt} \sim \text{postnormbdt}
by auto
from \text{shared-mult-dags} have \text{shared nort var}
proof (auto simp add: \text{shared-def isomorphic-dags-eq-def})
fix \text{st1 st2 bdt1}
assume \text{shared-imp:}
\forall t1 t2. t1 \in \text{Dags (repb · Nodes nb ll) (repb \propto low) (repb \propto high)} \land 
t2 \in \text{Dags (repb · Nodes nb ll) (repb \propto low) (repb \propto high)} 
\longrightarrow 
(\exists \text{bdt1}. \text{bdt t1 var} = \text{Some bdt1} \land \text{bdt t2 var} = \text{Some bdt1}) 
\longrightarrow t1 = t2
assume \text{st1-nort:} \text{st1} \leq \text{nort}
assume \text{st2-nort:} \text{st2} \leq \text{nort}

150
assume $bdt-st1$: $bdt st1 \text{ var } = \text{ Some } bdt1$
assume $bdt-st2$: $bdt st2 \text{ var } = \text{ Some } bdt1$
from $\text{nort-in-repNodes nort-dag nort-nTip}$
have $\text{nort-in-DagsrNodes}$:
$nort \in \text{ Dags } (\text{ repb } \cdot (\text{ Nodes nb ll})) (\text{ repb } \propto \text{ low}) (\text{ repb } \propto \text{ high})$
apply ~
apply (rule $\text{DagsI}$)
apply auto
done
show $st1 = st2$
proof (cases $st1$
  case $\text{ Tip}$
  note $st1-\text{Tip}=\text{this}$
  with $bdt-st1$ $bdt-st2$ show $\text{thesis}$
  by auto
next
  case $(\text{Node lst1 st1p rst1})$
  note $st1-\text{Node}=\text{this}$
  then have $st1-n\text{Tip}: st1 \neq \text{ Tip}$
  by simp
  show $\text{thesis}$
  proof (cases $st2$
    case $\text{ Tip}$
    with $bdt-st1$ $bdt-st2$ show $\text{thesis}$
    by auto
next
    case $(\text{Node lst2 st2p rst2})$
    note $st2-\text{Node}=\text{this}$
    then have $st2-n\text{Tip}: st2 \neq \text{ Tip}$
    by simp
from $\text{nort-in-DagsrNodes st1-nort ord-nort wf-lla st1-nTip}$
have $st1-\text{in-Dags}$:
$st1 \in \text{ Dags } (\text{ repb } \cdot (\text{ Nodes nb ll})) (\text{ repb } \propto \text{ low}) (\text{ repb } \propto \text{ high})$
apply ~
apply (rule $\text{Dags-subdags}$)
apply auto
done
from $\text{nort-in-DagsrNodes st2-nort ord-nort wf-lla st2-nTip}$
have $st2-\text{in-Dags}$:
$st2 \in \text{ Dags } (\text{ repb } \cdot (\text{ Nodes nb ll})) (\text{ repb } \propto \text{ low}) (\text{ repb } \propto \text{ high})$
apply ~
apply (rule $\text{Dags-subdags}$)
apply auto
done
from $st1-\text{in-Dags}$ $st2-\text{in-Dags}$ $bdt-st1$ $bdt-st2$ 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 $\propto$ low) (repb $\propto$ high) nort $\land$
reduced nort $\land$ shared nort var $\land$ ordered nort var $\land$
set-of nort $\subseteq$ set-of pret $\land$
(\exists postnormbdtt. bdt nort var = Some postnormbdtt $\land$ prebdt $\sim$ postnormbdtt) $\land$
($\forall$ no $\in$ 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$\Rightarrow$bool and postnormt postnormbdtt
assume ll: Levellist levellista nexta ll
assume repoint-spec:
  Dag pb lowb highb postnormt
  $\forall$ pt. pt $\notin$ set-of postnormt $\rightarrow$ low pt = lowb pt $\land$ high pt = highb pt
assume pret-dag: Dag p low high pret
assume ord-pret: ordered pret var
assume pnN: p $\neq$ Null
assume onemark-pret:
  $\forall$ n. n $\in$ set-of pret $\rightarrow$ mark n = mark p (is $\forall$ n. ?in-pret n $\rightarrow$ ?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 $\propto$ low) (repa $\propto$ 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 postnormbdtt prebdt $\sim$ postnormbdtt and
postdag-repa: $\forall$ no $\in$ set-of postnormt. repa no = no and
rep-eq: $\forall$ pt. pt $\notin$ set-of pret $\rightarrow$ rep pt = repa pt and
pret-marked: $\forall$ no. no $\in$ set-of pret $\rightarrow$ marka no = ($\neg$ mark p)
assume unmodif-next: $\forall$ p. p $\notin$ set-of pret $\rightarrow$ next p = nexta p
show ($\forall$ pt. pt $\notin$ set-of pret
  $\rightarrow$ low pt = lowb pt $\land$ high pt = highb pt $\land$
\textit{mark pt} = \textit{marka pt} \\

\textbf{proof} –

\textbf{from ll have length-ll-eq: length levellista} = \textit{length ll} \\
\textbf{by (simp add: Levellist-length)}

\textbf{from repoint-spec pnN subset-pret} \\
\textbf{have repoint-nc: (\forall pt. pt \notin set-of pret} \\
\textbf{\quad \quad \quad \quad \rightarrow low pt = lowb pt \land high pt = highb pt) \land Dag pb lowb highb postnormt} \\
\textbf{by auto}

\textbf{then have lowhigh-b-eq: \forall pt. pt \notin set-of pret} \\
\textbf{\quad \quad \quad \quad \rightarrow low pt = lowb pt \land high pt = highb pt} \\
\textbf{by fastforce}

\textbf{from wf-marking-ll pret-dag pnN} \\
\textbf{have mark-b-eq: \forall pt. pt \notin set-of pret \rightarrow mark pt} = \textit{marka pt} \\
\textbf{apply –} \\
\textbf{apply (simp add: wf-marking-def) } \\
\textbf{apply (split dag.splits) } \\
\textbf{apply simp } \\
\textbf{apply (rule allI) } \\
\textbf{apply (rule impI) } \\
\textbf{apply (elim conjE) } \\
\textbf{apply (erule tac \_ \_ in allE) } \\
\textbf{apply fastforce } \\
\textbf{done}

\textbf{with lowhigh-b-eq rep-eq unmodif-next} \\
\textbf{have pret-nc: \forall pt. pt \notin set-of pret} \\
\textbf{\quad \quad \quad \quad \rightarrow rep pt} = \textit{repa pt} \land low pt = lowb pt \land high pt = highb pt \land \\
\textbf{mark pt} = \textit{marka pt} \land next pt = nexta pt \\
\textbf{by blast}

\textbf{from pret-nc} \\
\textbf{show \_\_thesis} \\
\textbf{by fastforce}

\textbf{qed}

\textbf{next} – invariant to invariant

\textbf{fix var low high p rep mark pret prebdt levellist ll next marka n repc} \\
\textbf{and repb :: ref \Rightarrow ref} \\
\textbf{assume ll: Levellist levellist next ll} \\
\textbf{assume pret-dag: Dag p low high pret} \\
\textbf{assume ord-pret: ordered pret var} \\
\textbf{assume pnN: p \neq Null} \\
\textbf{assume prebdt-pret: bdt pret var = Some prebdt} \\
\textbf{assume wf-ll: wf-ll pret ll var} \\
\textbf{assume lll: length levellist} = var p + 1 \\
\textbf{assume n-Suc-var-p: n < var p + 1} \\
\textbf{assume wf-marking-in-ma: wf-marking pret mark marka (\neg mark p)}
**assume** rep-nc: \( \forall pt \ i. \)
\[ pt \notin \text{set-of pret} \lor n \leq i \land pt \in \text{set} (ll ! i) \land i < \text{var} + 1 \rightarrow \]
\[ \text{rep pt} = \text{repa pt} \]
**assume** repbNodes-in-Nodes: \( \text{repa} ' \ Nodes \ n \ ll \subseteq \ Nodes \ n \ ll \)
**assume**
normalize-prop: \( \forall no \in \text{Nodes} \ n \ ll. \)
\[ \text{var} (\text{repa no}) \leq \text{var no} \land \]
\( (\exists \text{nort}. \ Dag (\text{repa no}) (\text{repa} \prec \text{low}) (\text{repa} \prec \text{high}) \text{nort} \land \)
\[ \text{Dag no} \ \text{low} \ \text{not} \land \text{reduced} \ \text{nort} \land \text{ordered} \ \text{nort} \ \text{var} \land \]
\[ \text{set-of nort} \subseteq \text{repa} ' \ Nodes \ n \ ll \land \]
\( (\forall no \in \text{set-of nort.} \ \text{repa no} = \text{no}) \land \]
\( (\exists \text{nobdt. bdlt no} \ \text{var} = \text{Some nobdt} \land \)
\( (\exists \text{nordt. bdlt nort var} = \text{Some nordt} \land \)
\[ \text{nordt} \sim \text{nordt})) \)
**assume**
isomorphic-dags-eq:
\( \forall t1 \ t2. \ t1 \in \text{Dags} (\text{repa} ' \ Nodes \ n \ ll) (\text{repa} \prec \text{low}) (\text{repa} \prec \text{high}) \land \)
\( t2 \in \text{Dags} (\text{repa} ' \ Nodes \ n \ ll) (\text{repa} \prec \text{low}) (\text{repa} \prec \text{high}) \)
\( \rightarrow \text{isomorphic-dags-eq t1 t2 var} \)
**show** \( (\forall no \in \text{set} (ll ! n). \)
\[ \text{no} \neq \text{Null} \land \]
\( (\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \land \]
\[ \text{low no} \notin \text{set} (ll ! n) \land \]
\[ \text{high no} \notin \text{set} (ll ! n) \land \]
\[ \text{isLeaf-pt no} \ \text{low} \ \text{high} = (\text{var no} \leq 1) \land \]
\[ (\text{low no} \neq \text{Null} \rightarrow \text{repa} (\text{low no}) \neq \text{Null}) \land (\text{repa} \prec \text{low no} \notin \text{set} (ll ! n)) \land \]
\( (\forall no1 \in \text{set} (ll ! n). \forall no2 \in \text{set} (ll ! n). \ \text{var no1} = \text{var no2} \land \)
\( (\forall \text{repa.} (\forall no. \ \text{no} \notin \text{set} (ll ! n) \rightarrow \text{repa no} = \text{repa no}) \land \)
\( (\forall no \in \text{set} (ll ! n). \)
\[ \text{repa no} \neq \text{Null} \land \]
\( (\text{if} (\text{repa} \prec \text{low no}) = (\text{repa} \prec \text{high no}) \ \text{no} \land \text{low no} \neq \text{Null} \land \)
\[ \text{then repa no} = (\text{repa} \prec \text{low no}) \land \]
\[ \text{else repa no} \in \text{set} (ll ! n) \land \]
\[ \text{repa} (\text{repa no}) = \text{repa no} \land \]
\( (\forall no1 \in \text{set} (ll ! n). \)
\[ ((\text{repa} \prec \text{high no}) = (\text{repa} \prec \text{high no}) \land \]
\[ (\text{repa} \prec \text{low no}) = (\text{repa} \prec \text{low no}) \land \]
\[ (\text{repa no} = \text{repa no1}) \lor \rightarrow \]
\[ \text{var} p + 1 - (n + 1) < \text{var} p + 1 - n \land \]
\[ n + 1 \leq \text{var} p + 1 \land \]
\( (\forall pt i. \ pt \notin \text{set-of pret} \lor (n + 1 \leq i \land pt \in \text{set} (ll ! i) \land i < \text{var} \]
\[ p + 1 \rightarrow \]
\[ \text{repa pt} = \text{repa pt}) \land \]
\[ \text{repa} ' \ Nodes (n + 1) ll \subseteq \text{Nodes} (n + 1) ll \land \]
\( (\forall no \in \text{Nodes} (n + 1) ll. \)
\[ \text{var} (\text{repa no}) \leq \text{var no} \land \]
\( (\exists \text{nort}. \)
\[ \text{Dag (repa no) (repa} \prec \text{low) (repa} \prec \text{high) nort} \land \]

154
\[\begin{align*}
\text{Dag no low high not} & \land \\
\text{reduced nort} & \land \\
\text{ordered nort var} & \land \\
\text{set-of nort} & \subseteq \text{repa} \ N\text{odes} \ (n + 1) \ \text{ll} & \land \\
(\forall \text{no} \in \text{set-of nort}. \ \text{repa no} = \text{no}) & \land \\
(\exists \text{nobdt}. \ \text{bdt not var} = \text{Some nobdt} & \land \\
(\exists \text{norbdt}. \ \text{bdt nort var} = \text{Some norbdt} & \land \ \text{nobdt} \sim \text{norbdt})))
\end{align*}\]

\[\land (\forall t_1, t_2. \ t_1 \in \text{Dags} \ (\text{repa} \ N\text{odes} \ (n + 1) \ \text{ll}) \ (\text{repa} \propto \text{low}) \ (\text{repa} \propto \text{high}) \rightarrow \text{isomorphic-dags-eq} \ t_1 \ t_2)\]

\text{proof –}

\text{from} \ \text{ll} \ \text{have} \ \text{length-ll-eq: length levellist} = \text{length ll} \ \text{by} \ \text{(simp add: Levellist-length)}

\text{from} \ n\text{-Suc-var-p lll} \ \text{have nsl:} \ n < \text{length levellist by simp}

\text{hence nseqll:} \ n \leq \text{length levellist by simp}

\text{have srrl-precond:} \ (\forall \text{no} \in (\text{ll}! n). \ \text{no} \neq \text{Null} \land \ (\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \land \ \text{low no} \notin (\text{ll}! n) \land \ \text{high no} \notin (\text{ll}! n) \land \ \text{isLeaf-pt no low high} = (\text{var no} \leq 1) \land \ (\text{low no} \neq \text{Null} \rightarrow \text{repb (low no) \neq Null}) \land \ (\text{repb \propto low}) \ 

\text{proof (intro ballI)}

\text{fix no}

\text{assume no-in-lln:} \ \text{no} \in (\text{ll}! n)

\text{with} \ \text{uf-ll nsl} \ \text{have no-in-pret-var:} \ \text{no} \in \text{set-of pret} \land \ \text{var no} = n \ \text{by} \ \text{(simp add: uf-ll-def length-ll-eq)}

\text{with} \ \text{pret-dag} \ \text{have no-nNull:} \ \text{no} \neq \text{Null}

\text{apply –}

\text{apply (rule set-of-nn)}

\text{apply auto}

\text{done}

\text{from} \ \text{pret-dag prebdlt-pret no-in-pret-var}

\text{have balanced-no:} \ (\text{low no} = \text{Null}) = (\text{high no} = \text{Null})

\text{apply –}

\text{apply (erule conjE)}

\text{apply (rule-tac p=p and low=low in balanced-bdt)}

\text{apply auto}

\text{done}

\text{have low-no-notin-lln:} \ \text{low no} \notin (\text{ll}! n)

\text{proof (cases low no = Null)}

\text{case True}

\text{note lno-Null=this}

155
with balanced-no have hno-Null: high no = Null
  by simp
show ?thesis
proof (cases low no ∈ set (ll ! n))
  case True
  with wf-ll nsl ll have low no ∈ set-of pret ∧ var (low no) = n
    by (auto simp add: wf-ll-def length-ll-eq)
  with pret-dag have low no ≠ Null
    apply –
    apply (rule set-of-nn)
    apply auto
    done
  with lno-Null show ?thesis
    by simp
next
  assume lno-notin-lln: low no /∈ set (ll ! n)
  then show ?thesis
    by simp
qed
next
  assume lno-nNull: low no ≠ Null
  with balanced-no have hno-nNull: high no ≠ Null
  by simp
  with lno-nNull pret-dag ord-pret no-in-pret-var
  have var-children-smaller: var (low no) < var no ∧ var (high no) < var no
    apply –
    apply (rule var-ordered-children)
    apply auto
    done
  show ?thesis
proof (cases low no ∈ set (ll ! n))
  case True
  with wf-ll nsl ll have low no ∈ set-of pret ∧ 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 /∈ set (ll ! n)
  thus ?thesis
    by simp
qed
qed
have high-no-notin-lln: high no /∈ 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 ∈ set (ll ! n))
  case True
  with wf-ll nsll have high no ∈ set-of pret ∧ var (high no) = n
    by (auto simp add: wf-ll-def length-ll-eq)
  with pret-dag have high no ≠ Null
    apply –
    apply (rule set-of-nn)
    apply auto
    done
  with hno-Null show ?thesis
    by simp
next
  assume hno-notin-lln: high no /∈ set (ll ! n)
  then show ?thesis
    by simp
qed

next
  assume hno-nNull: high no ≠ Null
  with balanced-no have lno-nNull: low no ≠ Null
    by simp
  with hno-nNull pret-dag ord-pret no-in-pret-var
  have var-children-smaller: var (low no) < var no ∧ var (high no) < var no
    apply –
    apply (rule var-ordered-children)
    apply auto
    done
  show ?thesis
  proof (cases high no ∈ set (ll ! n))
    case True
    with wf-ll nsll have high no ∈ set-of pret ∧ 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 /∈ 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 \leq 1)
proof
assume no-isLeaf: isLeaf-pt no low high
from nobdt-ex no-dag-ex no-isLeaf show var no \leq 1
  apply −
  apply (rule bdt-Some-Leaf-var-le-1)
  apply auto
done
next
assume varno-le-1: var no \leq 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 \neq 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 \rightarrow 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-n Null pret-dag ord-pret no-in-pret-var
have var-children-smaller: var (low no) < var no ∧ 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 ∈ 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 ∈ Nodes n ll
apply (simp add: Nodes-def)
apply (rule-tac x=var (low no) in exI)
apply simp
done
hence repb (low no) ∈ repb ‘ Nodes n ll
by simp
with repbNodes-in-Nodes have repb-lno-in-Nodes: repb (low no) ∈ Nodes n ll
by blast
from pret-dag wf-ll nsll have Null ∉ 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) ≠ Null
by auto
qed
have Null-notin-lln: Null ∉ set (ll ! n)
proof (cases Null ∈ set (ll ! n))
case True
with wf-ll nsll have Null ∈ set-of pret ∧ var (Null) = n
by (simp add: wf-ll-def length-ll-eq)
with pret-dag have Null ≠ Null
apply –
apply (rule set-of-nn)
apply auto
done
thus ?thesis
by auto
next
assume Null ∉ set (ll ! n)
thus ?thesis
by simp
qed
have (repb ∝ low) no ∉ set (ll ! n)
proof (cases low no = Null)
case True

with Null-notin-lln show ?thesis
  by (simp add: null-comp-def)

next
  assume lno-nNull: low no ≠ Null
  with no-nNull no-in-pret-var pret-day have lno-in-pret: low no ∈ set-of pret
    apply –
    apply (rule-tac low=low in subelem-set-of-low)
    apply auto
    done
  from lno-nNull have propto-eq-comp: (repb ∝ low) no = repb (low no)
    by (simp add: null-comp-def)
  from lno-nNull balanced-no have hno-nNull: high no ≠ Null
    by simp
  with lno-nNull pret-dag ord-pret no-in-pret-var have var-children-smaller: var (low no) < var no ∧ 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 nll have low no ∈ 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 ∈ Nodes n ll
    apply (simp add: Nodes-def)
    apply (rule-tac x=var (low no) in exI)
    apply simp
    done
  hence repb (low no) ∈ repb ' Nodes n ll
    by simp
  with repbNodes-in-Nodes have repb-lno-in-Nodes: repb (low no) ∈ Nodes n ll
    by blast
  with lno-in-Nodes-n normalize-prop have var (repb (low no)) ≤ 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 ∃ k < n. repb (low no) ∈ set (ll ! k)
    by (auto simp add: Nodes-def)
  with wf-ll propto-eq-comp nssl show (repb ∝ low) 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
qed
with no-nNull balanced-no low-no-notin-lln high-no-notin-lln isLeaf-var repb-low-nNull
show  no \neq \text{Null} \land \\
(low no = \text{Null}) = (high no = \text{Null}) \land \\
low no \notin \text{set} (ll ! n) \land high no \notin \text{set} (ll ! n) \land \\
\text{isLeaf-pt no low high} = (\text{var no} \leq 1) \land \\
(low no \neq \text{Null} \rightarrow \text{repb} (low no) \neq \text{Null}) \land \\
(repb \propto \text{low} no \notin \text{set} (ll ! n) \\
by \text{auto} \\
\text{qed} \\
\text{have all-nodes-same-var: } \forall no1 \in \text{set} (ll ! n). \forall no2 \in \text{set} (ll ! n). \text{var no1} = \text{var no2} \\
\text{proof (intro ballI impI)} \\
\text{fix no1 no2} \\
\text{assume no1} \in \text{set} (ll ! n) \\
\text{with wf-ll nsll have var-lln-i: } \text{var no1} = n \\
\text{by (simp add: wf-ll-def length-ll-eq)} \\
\text{assume no2} \in \text{set} (ll ! n) \\
\text{with wf-ll nsll have var no2} = n \\
\text{by (simp add: wf-ll-def length-ll-eq)} \\
\text{with var-lln-i show var no1} = \text{var no2} \\
\text{by simp} \\
\text{qed} \\
\text{have} \left( \forall \text{repa}. \left( \forall \text{no}. \text{no} \notin \text{set} (ll ! n) \rightarrow \text{repb no} = \text{repa no} \right) \land \\
\left( \forall \text{no} \in \text{set} (ll ! n) \\
\text{repa no} \neq \text{Null} \land \\
\left( \text{if} \left( \text{repa} \propto \text{low} \right) \text{no} = (\text{repa} \propto \text{high}) \text{no} \land \text{low no} \neq \text{Null} \right) \land \\
\text{else repa no} \in \text{set} (ll ! n) \land \\
\text{repa} (\text{repa no}) = \text{repa no} \land \\
\left( \forall \text{no} \in \text{set} (ll ! n). \\
\left( (\text{repa} \propto \text{high}) \text{no1} = (\text{repa} \propto \text{high}) \text{no} \land \\
\text{repa} (\text{repa low}) \text{no1} = (\text{repa} \propto \text{low}) \text{no} \right) = \\
\left( \text{repa no} = \text{repa no1} \right) \right) \rightarrow \\
\text{var } p + 1 - (n + 1) < \text{var } p + 1 - n \land \\
n + 1 \leq \text{var } p + 1 \land \\
\left( \forall \text{pt i. pt} \notin \text{set-of pret} \lor (n + 1 \leq i \land \text{pt} \in \text{set} (ll ! i) \land i < \text{var } p + 1) \rightarrow \\
\text{repa pt} = \text{repa pt} \right) \land \\
\text{repa ' Nodes} (n + 1) \text{ ll} \subseteq \text{Nodes} (n + 1) \text{ ll} \land \\
(\forall \text{no} \in \text{Nodes} (n + 1) \text{ ll}. \\
\text{var} (\text{repa no}) \leq \text{var no} \land \\
\exists \text{not nort.} \\
\text{Dag} (\text{repa no}) (\text{repa} \propto \text{low}) (\text{repa} \propto \text{high}) \text{ nort} \land \\
\text{Dag no low high nort} \land \\
\text{reduced nort} \land \\
\text{ordered nort} \text{ var} \land \\
\text{set-of nort} \subseteq \text{repa ' Nodes} (n + 1) \text{ ll} \land \\
(\forall \text{no} \in \text{set-of nort}. \text{repa no} = \text{no}) \land \\
(\exists \text{nobdt}. \\
\text{bdt} \text{ not var} = \text{Some} \text{ nobdt} \land \\
161
\[(\exists \text{norbdt. bdt nort var} = \text{Some norbd}t \land \text{nobdt} \sim \text{norbdt}))\]

\[
(\forall t1 t2. \\
\quad t1 \in \text{Dags (repa} \ ' \text{Nodes (n + 1) ll) (repa} \in \text{low) (repa} \in \text{high) }) \\
\quad t2 \in \text{Dags (repa} \ ' \text{Nodes (n + 1) ll) (repa} \in \text{low) (repa} \in \text{high) }) \\
\quad \rightarrow \\
\quad \text{isomorphic-dags-eq t1 t2 var})
\]

(is (\(\forall \text{repc. } \text{srfl-post repc} \rightarrow ?\text{norm-inv repc} \))

proof (intro allI impI , elim conjE)

fix repc

assume repbc-nc: \(\forall \text{no. no} \notin \text{set (ll} ! n) \rightarrow \text{repb no} = \text{repc no}\)

assume rep-prop: \(\forall \text{no} \in \text{set (ll} ! n)\),

\(\text{repc no} \neq \text{Null} \land \)

\((\text{if (repc} \in \text{low) no} = (\text{repc} \in \text{high) no} \land \text{low no} \neq \text{Null} \land \)

then \(\text{repc no} = (\text{repc} \in \text{low) no} \land \)

else \(\text{repc no} \in \text{set (ll} ! n) \land \)

\(\text{repc (repc no)} = \text{repc no} \land \)

\((\forall \text{no1} \in \text{set (ll} ! n). \)

\((\text{repc} \in \text{high) no1} = (\text{repc} \in \text{high) no} \land \)

\(\text{repc low no1} = (\text{repc} \in \text{low) no} = \)

\(\text{repc no} = \text{repc no1}))\)

show ?norm-inv repc

proof −

from n-Suc-var-p have termi: \(\text{var} p + 1 - (n + 1) < \text{var} p + 1 - n\)

by arith

from wf-ll repbc-ne nsll

have Nodes-n-rep-nc: \(\forall p. p \in \text{Nodes n ll} \rightarrow \text{repb p} = \text{repc p}\)

apply −

apply (rule allI)

apply (rule implI)

apply (simp add: Nodes-def)

apply (erule exE)

apply (erule-tac x=p in allE)

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

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

done

from repbNodes-in-Nodes

have Nodes-n-rep-in-Nodesn:

\(\forall p. p \in \text{Nodes n ll} \rightarrow \text{repb p} \in \text{Nodes n ll}\)

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: \(\forall p. p \in \text{Nodes n ll} \rightarrow \text{repb p} \in \text{set-of pret}\)

apply −
apply (intro allI impI)
apply blast
done

have Nodes-repbc-Dags-eq: \( \forall p \ t. \ p \in \text{Nodes } n \ ll \)
\[ \rightarrow \text{Dag (repb } p\text{) (repb } \propto \text{ low}) (\text{repb } \propto \text{ high}) \; t = \text{Dag (repc } p\text{) (repc } \propto \text{ low}) (\text{repc } \propto \text{ high}) \; t \]

proof (intro allI impI)
fix \( p \ t \)
assume \( p\)-in-Nodes: \( p \in \text{Nodes } n \ ll \)
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: \( \text{Dag (repb } p\text{) (repb } \propto \text{ low}) (\text{repb } \propto \text{ high}) \; nort \) and
nort-in-repbNodes: set-of nort \( \subseteq \text{repb } ' \text{Nodes } n \ ll \)
apply
apply (erule-tac \( x=p \) in ballE)
prefer 2
apply auto
done

from nort-in-repbNodes repbNodes-in-Nodes
have nort-in-Nodesn: set-of nort \( \subseteq \text{Nodes } n \ ll \)
by blast

from pret-dag wf-ll nsll have Null \( \notin \text{Nodes } n \ ll \)
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-rebpc-dag: \( \text{Dag (repc } p\text{) (repc } \propto \text{ low}) (\text{repc } \propto \text{ high}) \; nort \)
by simp

from nort-in-Nodesn have \( \forall x \in \text{set-of nort} \; x \in \text{Nodes } n \ ll \)
apply
apply (rule ballI)
apply blast
done

with wf-ll nsll have \( \forall x \in \text{set-of nort} \; x \in \text{set-of pret } \land \; \text{var } x < n \)
apply
apply (rule ballI)
apply (rule wf-ll-Nodes-pret)
apply (auto simp add: length-ll-eq)
done

with pret-dag prebdt-pret nort-rebpc-dag ord-pret wf-ll nsll repbc-nc
have
\( \forall x \in \text{set-of nort} \; (\text{repc } \propto \text{ low}) \; x = (\text{repb } \propto \text{ low}) \; x \land \)
(\( \text{repc } \propto \text{ high}) \; x = (\text{repb } \propto \text{ high}) \; x \)
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 ∞ low) (repb ∞ high) nort =
  Dag (repc p) (repe ∞ low) (repc ∞ 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 ∝ low) (repb ∝ high) t =
  Dag (repc p) (repe ∝ low) (repc ∝ 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: ∀ no∈set (ll ! n).
  repc no ≠ Null ∧
  (if (repc ∝ low) no = (repc ∝ high) no ∧ low no ≠ Null
    then repc no = (repc ∝ low) no
    else repc no ∈ set (ll ! n) ∧ repc (repc no) = repc no ∧
    (∀ no1∈set (ll ! n). ((repc ∝ high) no1 = (repc ∝ high) no ∧
    (repc ∝ low) no1 = (repc ∝ low) no) = (repc no = repc no1)))
by blast

from nsll lll have n-var-prop: n + 1 <= var p + 1
by simp

from rep-nc have Sucn-repb-nc: (∀ pt. pt ≠ set-of pret ∨
  (∃ i. n + 1 ≤ i ∧ pt ∈ set (ll ! i) ∧ i < var p + 1)
  → rep pt = repb pt)
apply -
apply (intro allI implI)
apply (erule tac x = pt in allE)
apply auto
apply (rule tac x = i in extI)
apply auto
done

have repc-nc:
\(\forall pt. \, pt \notin \text{set-of \ pret} \lor\)
\((\exists i. \, n + 1 \leq i \land pt \in \text{set } (ll!i) \land i < \text{var p + 1})\)
\(\rightarrow \, \text{rep pt} = \text{repc pt}\)

**proof (intro allI impI)**

**fix pt**

**assume pt-notin-lower-ll: pt \notin \text{set-of \ pret} \lor\)
\((\exists i. \, n + 1 \leq i \land pt \in \text{set } (ll!i) \land i < \text{var p + 1})\)

**show rep pt = repc pt**

**proof (cases pt \notin \text{set-of \ pret}**

**case True**

**with \text{wf-ll nsll} have pt \notin \text{set } (ll!n)**

**apply (simp add: \text{wf-ll-def length-ll-eq})**

**apply (case-tac pt \in \text{set } (ll!n))**

**apply (subgoal-tac pt \in \text{set-of \ pret})**

**apply (auto)**

**done**

**with repbc-nc have repb pt = repc pt**

**by auto**

**with Sucn-repb-nc True show \text{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 \land pt \in \text{set } (ll!i) \land i < \text{var p + 1}**

**by simp**

**with nsll \text{wf-ll lll} have pt-notin-lln: pt \notin \text{set } (ll!n)**

**apply \neg**

**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 repbc-nc have repb pt = repc pt**

**by auto**

**with Sucn-repb-nc pt-in-higher-ll show \text{thesis}**

**by auto**

**qed**

**qed from \text{wf-ll nsll}**

**have \text{Nodes-notin-lln: } \forall no \in \text{Nodes n ll. no \notin \text{set } (ll!n)}**

**apply (simp add: \text{Nodes-def})**

**apply clarify**

**apply (drule \text{no-in-one-ll})**

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

**done**

**with repbc-nc**

**have \text{Nodes-repnc: } \forall no \in \text{Nodes n ll. repb no = repc no}**

**apply \neg**

**apply (rule ballI)**

**apply (erule-tac x=no in allE)**

165
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 ⊆ Nodes (n + 1) ll
proof
fix x
assume x-in-repcNodesSucn: x ∈ repc ' Nodes (n + 1) ll
show x ∈ Nodes (n + 1) ll
proof (cases x ∈ repc 'Nodes n ll)
case True
  with repbNodes-repcNodes repbNodes-in-Nodes have x ∈ Nodes n ll
    by auto
  with Nodes-subset show ?thesis
    by auto
next
assume x /∈ repc 'Nodes n ll
with x-in-repcNodesSucn have x-in-repclln: x ∈ 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 = repc y
assume y-in-repclln: y ∈ set (ll ! n)
from rep-prop y-in-repclln obtain
  repc-nNull: repc y ≠ Null and
  red-prop: (repc ∞ low) y = (repc ∞ high) y ∧
  low y ≠ Null → repc y = (repc ∞ high) y and
  share-prop: ((repc ∞ low) y = (repc ∞ high) y → low y = Null)
  → repc y ∈ set (ll ! n) ∧ repc (repc y) = repc y ∧
  (∀ no1∈set (ll ! n)).
\[(\text{repc } \propto \text{ high}) \text{ no1} = (\text{repc } \propto \text{ high}) \text{ y} \land
(\text{repc } \propto \text{ low}) \text{ no1} = (\text{repc } \propto \text{ low}) \text{ y} = (\text{repc y} = \text{repc no1})\]

using \[[\text{simp-depth-limit} = 4]\]

by auto

from \(\text{wf-ll nsl} \ y-in-repeln\) obtain

\(y-in-pret\): \(y \in \setof{pret}\) and

\(\text{vary-n}\): \(\text{var y} = n\)

by (auto simp add: \(\text{wf-ll-def length-ll-eq}\))

from \(y-in-pret\) \(\text{pret-dag}\) have \(y-nNull\): \(y \neq \text{Null}\)

apply –

apply (rule \(\text{set-of-nn}\))

apply auto

done

show \(x \in \text{Nodes (n + 1) ll}\)

proof (cases low \(y = \text{Null}\))

case True

from \(\text{pret-dag prebdt-pret True y-in-pret}\)

have \(\text{highy-Null}\): \(\text{high y} = \text{Null}\)

apply –

apply (drule \(\text{balanced-bdt}\))

apply auto

done

with \(\text{share-prop True obtain}\)

\(\text{repcy-in-llb}: \text{repc y} \in \setof{ll ! n}\) and

\(\text{rry-ry}: \text{repc (repc y)} = \text{repc y and}\)

\(\text{y-other-node-prop}: \forall \text{ no1} \in \setof{ll ! n}.\)

\((\text{repc } \propto \text{ high}) \text{ no1} = (\text{repc } \propto \text{ high}) \text{ y} \land
(\text{repc } \propto \text{ low}) \text{ no1} = (\text{repc } \propto \text{ low}) \text{ y} = (\text{repc y} = \text{repc no1})\)

by simp

from \(\text{repcy-in-llb}\) \(x-repcy\) show \(\text{thesis}\)

by (auto simp add: \(\text{Nodes-def}\))

next

assume lowy-nNull: \(\text{low y} \neq \text{Null}\)

with \(\text{pret-dag prebdt-pret y-in-pret}\)

have \(\text{highy-nNull}\): \(\text{high y} \neq \text{Null}\)

apply –

apply (drule \(\text{balanced-bdt}\))

apply auto

done

show \(\text{thesis}\)

proof (cases \(\text{repc } \propto \text{ low} \ y = (\text{repc } \propto \text{ high}) \ y\))

case True

with \(\text{red-prop lowy-nNull}\) have \(\text{repc y} = (\text{repc } \propto \text{ high}) \ y\)

by auto

with \(\text{highy-nNull}\) have \(\text{red-repc-y}: \text{repc y} = \text{repc (high y)}\)

by (simp add: \(\text{null-comp-def}\))

from \(\text{pret-dag ord-pret y-in-pret lowy-nNull}\) \(\text{highy-nNull}\)

have \(\text{var (low y)} < \text{var y} \land \text{var (high y)} < \text{var y}\)

167
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 high-nNull pret-dag
have high y ∈ set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
done
with wf-ll varhighy have high y ∈ Nodes n ll
  by (auto simp add: wf-ll-def Nodes-def)
with red-repc-y have repc y ∈ repc +Nodes n ll
  by simp
with x-repcy have x ∈ repc +Nodes n ll
  by simp
with rephNodes-repcNodes repbNodes-in-Nodes
have x ∈ Nodes n ll
  by auto
with Nodes-subset show ?thesis
  by auto
next
assume (repc ∝ low) y ≠ (repc ∝ high) y
with share-prop obtain
  repcy-in-llbn: repc y ∈ set (ll ! n) and
  rry-ry: repc (repc y) = repc y and
  y-other-node-share: ∀ no1∈set (ll ! n).
  ((repc ∝ high) no1 = (repc ∝ high) y ∧
   (repc ∝ low) no1 = (repc ∝ low) y) = (repc y = repc no1)
  by auto
with repcy-in-llbn x-repcy have x ∈ set (ll ! n)
  by auto
then show ?thesis
  by (auto simp add: Nodes-def)
qed
qed
qed
qed
have (∀ no∈Nodes (n + 1) ll.
  var (repc no) ≤ var no ∧
  (∃ not nort. Dag (repc no) (repc ∝ low) (repc ∝ high) nort ∧
   Dag no low high not ∧
   reduced nort ∧ ordered nort var ∧
   set-of nort ⊆ repc +Nodes (n + 1) ll ∧
   (∀ no∈set-of nort. repc no = no) ∧
   (∃ nobdt. bdt not var = Some nobdt ∧
∃\text{norbdt. bdt nort var = Some norbdt ∧ nobdt ∼ norbdt)})
(is ∀\text{n0∈Nodes (n + 1) ll. ?Q i no)

\text{proof (intro ballI)}

\text{fix no}
\text{assume no-in-Nodes: no ∈ Nodes (n + 1) ll}
\text{from wf-ll no-in-Nodes nsll have no-in-pret: no ∈ set-of pret}
\text{apply (simp add: wf-ll-def Nodes-def length-ll-eq)}
\text{apply (erule conjE)}
\text{apply (thin-tac ∀ q. q ∈ set-of pret ---→ q ∈ set (ll ! var q))}
\text{apply (erule exE)}
\text{apply (erule-tac x \text{= k in allE})}
\text{apply arith}
\text{apply (erule-tac x\text{= no in ballE})}
\text{apply auto}
\text{done}

\text{from pret-dag no-in-pret have nonNull: no\neq Null}
\text{apply –}
\text{apply (rule set-of-nn [rule-format])}
\text{apply auto}
\text{done}

\text{show ?Q i no}
\text{proof (cases no ∈ Nodes n ll)}
\text{case True}
\text{note no-in-Nodesn\text{=this}}
\text{with wf-ll nssl \text{ na-in-Nodes}}
\text{have no-notin-llbn: no \notin set (ll ! n)}
\text{apply –}
\text{apply (simp add: Nodes-def length-ll-eq)}
\text{apply (elim exE)}
\text{apply (drule-tac ?i\text{=ka and ?j=n in no-in-one-ll})}
\text{apply arith}
\text{apply simp}
\text{apply auto}
\text{done}
\text{with repbc-nc have repb-no-eq-repc-no: repb no = repc no}
\text{by simp}
\text{from repbc-nc no-in-Nodes no-notin-llbn normalize-prop True}
\text{have varrep-eq-var: var (repc no) \leq var no}
\text{apply –}
\text{apply (erule-tac x\text{=no in ballE})}
\text{prefer 2}
\text{apply simp}
\text{apply (erule-tac x\text{=no in allE})}
\text{apply simp}
\text{done}

\text{moreover}
\text{from True normalize-prop no-in-Nodes obtain not nort where}
\text{nort-dag: Dag (repb no) (repb ∝ low) (repb ∝ high) nort and}
ord-nort: ordered nort var and
subset-nort-not: set-of nort \subseteq repb \langle(Nodes n ll) and
not-dag: Dag no low high not and
red-nort: reduced nort and
nort-repb: (\forall no \in set-of nort. repb no = no) and
bdlt-prop: \exists nobdt norbdlt. bdlt not var = Some nobdt \land
bdlt nort var = Some norbdlt \land nobdt \sim norbdlt
by blast

moreover
from Nodsn-notin-lIn rephc-nc nort-repb subset-nort-not repbNodes-in-Nodes

have nort-repc:
(\forall no \in 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 nortnodesn: (\forall no. no \in set-of nort \rightarrow no \notin Null)
apply –
apply (rule allI)
apply (rule impI)
apply (rule set-of-nn)
apply auto
done

moreover
from no-notin-llbn rephc-nc have repbc-no: repc no = repb no
by fastforce

with nort-dag
have nortrephc-dag: Dag (repc no) (repc \propto low) (repc \propto high) nort
by simp

from wf-ll nseqll have Nodes n ll \subseteq set-of pret
apply –
apply (rule Nodes-levelist-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 xsn-in-pret: \forall x \in set-of nort. var x < n \land x \in set-of pret
proof (rule ballI)
fix x
assume x-in-nort: x \in set-of nort
from x-in-nort subset-nort-not repbNodes-in-Nodes

170
have \( x \in \text{Nodes } n \text{ ll} \)
   by blast
with \( \text{wf-ll nsll} \) have \( xsn: \text{var } x < n \)
   apply (simp add: \( \text{wf-ll-def Nodes-def length-ll-eq} \))
   apply (erule conjE)
   apply (thin-tac \( \forall q. q \in \text{set-of pret} \to q \in \text{set } (\text{ll ! var } q) \))
   apply (erule \( \text{exE} \))
   apply (erule \( \text{impE} \))
   apply arith
   apply (erule-tac \( x = k \) in allE)
   apply (erule \( \text{impE} \))
   apply arith
   apply (erule-tac \( x = x \) in allE)
   apply auto
   done
from \( x\text{-in-nort subset-nort-pret} \) have \( x\text{-in-pret}: x \in \text{set-of pret} \)
   by blast
with \( xsn \) show \( \text{var } x < n \land x \in \text{set-of pret} \) by simp
qed
with \( \text{pret-dag prebdlt-pret nortrepbc-dag ord-pret wf-ll nsll} \)
repbc-nc
have \( \forall x \in \text{set-of nort} \cdot ((\text{repc }\propto \text{low}) x = (\text{repb }\propto \text{low}) x \land \)
   \( (\text{repc }\propto \text{high}) x = (\text{repb }\propto \text{high}) x \) \)
   apply –
   apply (rule \( \text{nort-null-comp} \))
   apply (auto simp add: \( \text{length-ll-eq} \))
   done
with \( \text{nort-dag} \)
have \( \text{Dag} (\text{repc }\propto \text{low}) (\text{repc }\propto \text{high}) \text{nort} = \)
   \( \text{Dag} (\text{repb }\propto \text{low}) (\text{repb }\propto \text{high}) \text{nort} \)
   apply –
   apply (rule heaps-eq-Dag-eq)
   apply simp
   done
with \( \text{nortrepbc-dag} \) show \( ?\text{thesis} \)
   by simp
qed
moreover
have \( \text{set-of nort} \subseteq \text{repc }'(\text{Nodes } (n + 1) \text{ ll}) \)
proof –
  have \( \text{Nodesn-in-NodesSucn}: \text{Nodes } n \text{ ll } \subseteq \text{Nodes } (n + 1) \text{ ll} \)
    by (simp add: \( \text{Nodes-def set-split} \))
  then have \( \text{rephNodesn-in-rephNodesSucn}: \)
    \( \text{reph }'(\text{Nodes } n \text{ ll}) \subseteq \text{reph }'(\text{Nodes } (n + 1) \text{ ll}) \)
    by blast
from \( \text{wf-ll nsll} \)
have \( \text{Nodess-n-notin-lln}: \forall no \in \text{Nodes } n \text{ ll}. no \notin \text{set } (\text{ll ! n}) \)
   apply (simp add: \( \text{Nodes-def length-ll-eq} \))
   apply clarify
   apply (drule no-in-one-ll)
   apply auto
done
with repbc-nc have \( \forall \ no \in \text{Nodes } n \ ld. \ repb \ no = \ repc \ no \)
apply
apply (rule ballI)
apply (erule-tac \( z = no \in \text{allE} \))
apply simp
done
then have repbNodes-repcNodes:
\( \text{repb } (\text{Nodes } n \ ld) = \text{repc } (\text{Nodes } n \ ld) \)
apply
apply rule
apply blast
apply rule
apply (erule imageE)
apply (erule-tac \( x=x a \in \text{ballE} \))
prefer 2
apply simp
apply rule
apply auto
done
from Nodesn-in-NodesSucn
have \( \text{repc } (\text{Nodes } n \ ld) \subseteq \text{repc } (\text{Nodes } (n+1) \ ld) \)
by blast
with repbNodes-repcNodes subset-nort-not repbNodesn-in-repbNodesSucn

show ?thesis by simp
qed
ultimately show ?thesis
  by blast
next
assume \( no \notin \text{Nodes } n \ ld \)
with no-in-Nodes have no-in-llbn: \( no \in \text{set } (\ld ! n) \)
apply (simp add: Nodes-def)
apply (erule exE)
apply (erule-tac \( x=k \in \text{allE} \))
apply (case-tac \( k<n \))
apply simp
apply simp
apply (elim conjE)
apply (case-tac \( k=n \))
apply simp
apply arith
done
with wf-ll nsll have varno: \( \text{var } no = n \)
by (simp add: wf-ll-def length-ll-eq)
from repbc-changes no-in-llbn
have repbcno-changes: \( \text{repc } no \neq \text{Null } \land \)
\((\text{repc } \propto \text{low } no = (\text{repc } \propto \text{high } no) \land \text{low } no \neq \text{Null } \)
\( \rightarrow \text{repc } no = (\text{repc } \propto \text{high } no) \) \land

172
(((repc $\propto$ low) no = (repc $\propto$ high) no $\rightarrow$ low no = Null)
$\rightarrow$ repc no $\in$ set (ll ! n) $\land$ repc (repc no) = repc no $\land$
($\forall$ no1$\in$set (ll ! n). ((repc $\propto$ high) no1 = (repc $\propto$ high) no $\land$
(repc $\propto$ low) no1 = (repc $\propto$ low) no) = (repc no = repc no1))
(is ?rnonN $\land$ ?repreduce $\land$ ?repshare)
using [[simp-depth-limit=4]]
by (simp split: split-if)
then obtain
rnonN: ?rnonN and
repreduce: ?repreduce and
repshare: ?repshare
by blast
have repcn-normalize: var (repc no) $\leq$ var no $\land$
($\exists$ not nort. Dag (repc no) (repc $\propto$ low) (repc $\propto$ high) nort $\land$
Dag no low high not $\land$ reduced nort $\land$ ordered nort var $\land$
set-of nort $\subseteq$ repc $\cdot$ Nodes (n + 1) ll $\land$
($\forall$ no$\in$set-of nort. repc no = no) $\land$
($\exists$ nobdt. bdt not var = Some nobdt $\land$
($\exists$ norbdt. bdt nort var = Some norbdt $\land$ nobdt $\sim$ norbdt))
(is ?varrep $\land$ ?repcn-prop
is ?varrep $\land$
($\exists$ not nort. ?nort-dag nort $\land$ ?not-dag not $\land$ ?red nort $\land$
?ord nort $\land$ ?nort-in-Nodes nort $\land$ ?repcno-no-n nort $\land$ ?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 nssl have varrno-n: var (repc no) = n
  by (simp add: wf-ll-def length-ll-eq)
with varrno have varrep: ?varrep
  by simp
from wf-ll nssl 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 nssl repcnoinln
have nno-in-pret: repc no $\in$ set-of pret
  by (simp add: wf-ll-def length-ll-eq)
from repcnoinln repshare lownoNull
have reprep-eq-rep: repc (repc no) = repc no
  by simp

173
with \text{repnoinln \ repshare lowNull}

\textbf{have repchildreneq:}
\[(\text{repc} \propto \text{high}) (\text{repc no}) = (\text{repc} \propto \text{high}) \text{ no} \land
(\text{repc} \propto \text{low}) (\text{repc no}) = (\text{repc} \propto \text{low}) \text{ no})\]
by simp

\textbf{have repcn-prop: repcn-prop}
apply –
apply (rule-tac \(x=(\text{Node Tip no Tip}) \text{ in exI}\))
apply (rule-tac \(x=(\text{Node Tip (repc no) Tip}) \text{ in exI}\))
apply (intro conjI)
apply simp
prefer 3
apply simp
prefer 3
apply simp

\textbf{proof –}
from pret-dag \(pnN \ rno-in-pret \text{ have rnonN: repc no \(\neq\) Null}
apply (case-tac repc no = Null)
apply auto
done

from highnoNull repchildreneq
have rhighNull: (repc \(\propto\) high) (repc no) = Null
by (simp add: null-comp-def)
from lownoNull repchildreneq
have rlowNull: (repc \(\propto\) low) (repc no) = Null
by (simp add: null-comp-def)
with rhighNull rnonN
show repc no \(\neq\) Null \(\land\) (repc \(\propto\) low) (repc no) = Null \(\land\)
(repc \(\propto\) 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) \(\subseteq\) repc ‘ Nodes (n + 1) ll
by simp

next
show \(\forall\) no \(\in\) set-of (Node Tip (repc no) Tip). repc no = no
proof
fix pt
assume pt-in-repLeaf: pt \(\in\) 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 varrno: 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 varrno 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 varrno-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 hno-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)

175
from pret-dag ord-pret no-in-pret hno-nNull hno-nNull

have varnos-varno: var (high no) < var no
by (metis var-ordered-children)
with varno have varnos-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-varno have high normal: high no /
∈ set (ll ! n)
apply −
apply (rule no-in-one-ll)
apply (auto simp add: length-ll-eq)
done
with repbc-nc have repb-repc-high: repb (high no) = repc (high no) by simp
with normalize-prop hno-in-Nodesa 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) ∧
(∀ nort ∈ set-of nort. repb no = no) ∧
(∃ nobdt norbdt. bd t not var = Some nobdt ∧ bd t 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 ∧
?rephno-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 varrepno-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

176
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=k 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. recp (high no) ∈ set (ll ! k)
  by (auto simp add: wf-ll-def)
with wf-ll nsll have recp (high 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
with repbc-nc
have repbhigh-idem: repb (repc (high no)) = recp (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 knot where high-dag: ?high-dag knot
  by auto
from wf-ll nsll
have ∀ no ∈ Nodes n ll. no \notin 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 = recp no
  apply –
  apply (rule ballI)
  apply (erule-tac x=no in allE)
  apply simp
done
then
have repbNodes-repcNodes:
  repb ‘(Nodes n ll) = recp ‘(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 nsll wf-ll
have null-notin-Nodesn: Null ∉ 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) ∈ repc '(Nodes n ll)
  by blast
with repbNodes-in-Nodes repcNodes-repbNodes
have repc (high no) ∈ Nodes n ll
  apply simp
  apply blast
  done
with null-notin-Nodesn have rhn-nNull: repc (high no) ≠ Null
  by auto

from no-in-pret nonNull hno-nNull pret-dag
have lno-in-pret: low no ∈ set-of pret
  by (rule subelem-set-of-low)
with wf-ll
have lno-in-ll: low no ∈ set (ll ! (var (low no)))
  by (simp add: wf-ll-def)
from pret-dag ord-pret no-in-pret hno-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 ∈ 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 nsll lno-in-ll
have low no ∉ set (ll ! n)
  apply –
  apply (rule no-in-one-ll)
  apply (auto simp add: length-ll-eq)
  done
with repbc-nc have repb-repe-low: repb (low no) = repc (low no) by simp

with normalize-prop lno-in-Nodesn varlnos-varno varno have low-normalize: var (repe (low no)) ≤ var (low no) ∧
(∃ not nort. Dag (repe (low no)) (repe low) (repe high) nort ∧
Dag (low no) low high not ∧ reduced nort ∧ ordered nort var ∧
set-of nort ⊆ repb ‘((Nodes n ll) ∧
(∀ no∈set-of nort. repb no = no) ∧
(∃ nobdt norbd. bdt not var = Some nobdt ∧ bdt nort var = Some
norbd ∧
nobdt ∼ norbd))
(is ?varrep-low ∧
(∃ not nort. ?repbelow-dag nort ∧ ?low-dag not ∧ ?redhigh nort ∧
?ordhigh nort ∧ ?replow-in-Nodes nort ∧ ?low-repno-no nort ∧
?lowadd-prop not nort)
is ?varrep-low ∧ ?not-nort-prop-low)
apply simp
apply (erule-tac x=low no in ballE)
apply simp
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 repe (low no) ∈ repe ‘((Nodes n ll)
by blast
with repbNodes-in-Nodes haveNodes-repbNodes have repe (low no) ∈ Nodes n ll
apply simp
apply blast
done
with null-notin-Nodesn have rln-nNull: repe (low no) ≠ Null
by auto

show ?thesis
proof (cases repe (low no) = repe (high no))
case True
  note red-case=this
with repreduce lno-nNull hno-nNull
have rno-eq-hrno: repe no = repe (high no)
by (simp add: null-comp-def)
  from varlnos-varno rno-eq-hrno varrep-high 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 rnot lnot)
  apply (rule-tac x=(Node lnot no rnot) in exI)
apply (rule-tac \( x=\text{rnort} \) in \( \exists I \))
apply (elim conjE)
apply (intro conjI)
prefer 7
apply (elim \( \exists E \))
apply (rename-tac rnot lnot rnort lnort rnobdt lnobdt rnorbdt lnorbdt)
apply (elim conjE)
apply (case-tac Suc 0 < var no)
apply (rule-tac \( x=(\text{Bdt-Node} \text{lnobdt} \text{var no} \text{rnobdt}) \) in \( \exists I \))
apply (rule conjI)
prefer 2
apply (rule-tac \( x=\text{rnorbdt} \) in \( \exists I \))
apply (rule conjI)
proof –
 fix \( \text{rnot lnot rnort lnort} \)
assume highnort-dag:
\( \text{Dag (repc (\text{high no})) (repb \propto \text{low}) (repb \propto \text{high}) \text{rnort}} \)
assume ord-nort: \( \text{ordered rnot var} \)
assume rnot-in-repNodes: \( \text{set-of rnot} \subseteq \text{repb ' Nodes n ll} \)
from rnot-in-repNodes repbNodes-in-Nodes
have nort-in-Nodes: \( \text{set-of rnot} \subseteq \text{Nodes n ll} \)
 by blast
from varhnos-n varrep-high
have vrhnos-n: \( \text{var (repc (\text{high no})) < n by simp} \)
from rhn-nNull highnort-dag
have \( \exists \text{lno rno. rnot = Node lno (repc (\text{high no})) rno by fastforce} \)
with highnort-dag-rhn-nNull have root rnot = repc (high no) by auto
 with ord-nort have \( \forall x \in \text{set-of rnot}. \text{var x} \leq \text{var (repc (high no))} \)
apply –
apply (rule ballI)
apply (drule ordered-set-of)
apply auto
done
with vrhnos-n have vxsn: \( \forall x \in \text{set-of rnot}. \text{var x} < n \)
 by fastforce
from nort-in-Nodes have \( \forall x \in \text{set-of rnot}. x \in \text{Nodes n ll} \)
 by auto
with wf-ll nsll
have x-in-pret: \( \forall x \in \text{set-of rnot}. x \in \text{set-of pret} \)
 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: \( \forall x \in \text{set-of rnot}. \text{var x} < n \wedge x \in \text{set-of pret} \)
 by auto
with pret-dag prebdt-pret highnort-dag ord-pret wf-ll nsll
  repbc-nc
have \( \forall x \in \text{set-of } \text{rnort}. (\text{repc } x) \land (\text{repb } x) \)
   apply –
   apply (rule nort-null-comp)
   apply (auto simp add: length-ll-eq)
   done
with rno-eq-hrno
have Dag (\text{repc no}) (\text{repc } low) (\text{repc } high) \text{rnort} =
  Dag (\text{repc no}) (\text{repb } low) (\text{repb } high) \text{rnort}
  apply –
  apply (rule heaps-eq-Dag-eq)
  apply simp
  done
with highnort-dag rno-eq-hrno
show Dag (\text{repc no}) (\text{repc } low) (\text{repc } high) \text{rnort} by simp
next
fix rnot lnot rnort lnort
assume lnot-dag: Dag \text{low no} \text{low high lnot}
assume rnot-dag: Dag \text{high no} \text{low high rnot}
with lnot-dag nonNull
show Dag no low high (Node lnot no rnot) by simp
next
fix rnot lnot rnort lnort
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: set-of \text{rnort} \subseteq \text{repb} \text{Nodes} \text{n ll}
have Nodes n ll \subseteq Nodes (n + 1) ll
  by (simp add: Nodes-def set-split)
then have recp \text{Nodes} n ll \subseteq recp \text{Nodes} (n + 1) ll
  by blast
with rnort-in-Nodes repbNodes-repNodes
show set-of \text{rnort} \subseteq recp \text{Nodes} (n + 1) ll
  by (simp add: Nodes-def)
next
fix rnot rnorbdt
assume bdt rnot var = Some rnorbdt
then show bdt rnot var = Some rnorbdt by simp
next
fix rnot lnot rnort lnort rnorbdt lnobdt rmorbdt lnorbdt
assume rnot-dag: rnorbdt \sim rnorbdt
assume lnot-dag:
\[ \text{Dag} \left( \text{repc} \left( \text{low no} \right) \right) \left( \text{repb} \left( \text{low} \right) \right) \left( \text{repb} \left( \text{high} \right) \right) \]

\text{assume \ rnor\-dag:}
\[ \text{Dag} \left( \text{repc} \left( \text{high no} \right) \right) \left( \text{repb} \left( \text{low} \right) \right) \left( \text{repb} \left( \text{high} \right) \right) \]

\text{assume \ lnor\-def: \ bdt} \ \text{lnort} \ \text{var} = \text{Some} \ \text{lnorbd}
\text{assume \ rnor\-def: \ bdt} \ \text{rnort} \ \text{var} = \text{Some} \ \text{rnorbd}

\text{assume \ lcongeval:lnobdt \sim \ lnorbd}

\text{from \ red-case \ lnort\-dag \ rnor\-dag}
\text{have \ lnort\-rnort: \ lnort = \ rnort}
\text{by \ simp add: \ Dag-unique \ del: \ Dag-Ref}

\text{with \ lnorbd\-def \ lcongeval \ rnorbd\-def}
\text{have \ lnobdt\-rnobdt: \ lnobdt \sim \ rnobdt \ by \ simp}

\text{with \ rcongeval \ have \ lnobdt \sim \ rnobdt}
\text{apply} -
\text{apply \ (rule \ cong-\-eval-trans)}
\text{apply \ (auto \ simp \ add: \ cong-\-eval-sym)}
\text{done}

\text{then have \ Bdt-Node \ lnobdt \ (var \ no) \ rnobdt \sim \ rnobdt}
\text{apply} -
\text{apply \ (simp \ add: \ cong-\-eval-sym \ [rule-format])}
\text{apply \ (rule \ cong-\-eval-child-high)}
\text{apply \ assumption}
\text{done}

\text{with \ rcongeval \ show \ Bdt-Node \ lnobdt \ (var \ no) \ rnobdt \sim \ rnobdt}
\text{apply} -
\text{apply \ (rotate-tac 1)}
\text{apply \ (rule \ cong-\-eval-trans)}
\text{apply \ auto}
\text{done}

\text{next}
\text{fix \ lnort \ rnot \ lnobdt \ rnobdt}
\text{assume \ lnort\-dag: \ Dag \ (low \ no) \ low \ high \ lnort}
\text{assume \ rnot\-dag: \ Dag \ (high \ no) \ low \ high \ rnot}
\text{assume \ lnobdt\-def: \ bdt} \ \text{lnort} \ \text{var} = \text{Some} \ \text{lnobdt}
\text{assume \ rnorbd\-def: \ bdt} \ \text{rnort} \ \text{var} = \text{Some} \ \text{rnorbd}
\text{assume \ onesvarno: \ Suc 0 < \ var \ no}

\text{with \ rnorbd\-def \ lnort\-dag \ rnorbd\-dag \ lnobdt\-def}
\text{show \ bdt \ (Node \ lnort \ no \ rnot) \ var =}
\text{Some} \ \text{(Bdt-Node \ lnobdt \ (var \ no) \ rnobdt)}
\text{by \ simp}

\text{next}
\text{fix \ rnot \ lnort \ rnot \ lnobdt \ lnobdt \ rnorbd \ rnorbd \ lnobdt \ lnorbd}
\text{assume \ lnobdt\-def: \ bdt} \ \text{lnort} \ \text{var} = \text{Some} \ \text{lnobdt}
\text{assume \ rnorbd\-def: \ bdt} \ \text{rnort} \ \text{var} = \text{Some} \ \text{rnorbd}
\text{assume \ cong-rno-rnor: \ rnorbd \sim \ rnorbd}
\text{assume \ lnort\-dag: \ Dag \ (low \ no) \ low \ high \ lnort}
\text{assume \ rnot\-dag: \ Dag \ (high \ no) \ low \ high \ rnot}
\text{assume \ \neg \ Suc 0 < \ var \ no}

\text{then have \ var\-seq\-1: \ var \ no = 0 \lor \ var \ no = 1 \ by \ auto}
show $\exists \text{nobdt}, \text{bdt} \ (\text{Node lnot no rnot} \ \text{var} = \text{Some nobdt} \land
(\exists \text{norbdt}, \text{bdt rnot var} = \text{Some norbdt} \land \text{nobdt} \sim \text{norbdt}))$

proof (cases \text{var no} = 0)

case True

note \text{vnoNull} = \text{this}

with \text{pret-dag ord-pret no-in-pret lno-nNull hno-nNull}

show ?thesis

apply —

apply (drule \text{var-ordered-children})

apply auto

done

next

assume \text{var no} \neq 0

with \text{varnoseq1} have \text{vnoOne}: \text{var no} = 1 by simp

from \text{pret-dag ord-pret no-in-pret lno-nNull hno-nNull}

have \text{vlorNull}: \text{var} (\text{low no}) = 0 \land \text{var} (\text{high no}) = 0

apply —

apply (drule \text{var-ordered-children})

apply auto

done

then have \text{vlNull}: \text{var} (\text{low no}) = 0 by simp

from \text{vlorNull} have \text{vrNull}: \text{var} (\text{high no}) = 0 by simp

from \text{lnobdt-def lnot-dag vlNull lno-nNull}

have \text{lnobdt-Zero}: \text{lnobdt} = \text{Zero}

apply —

apply (drule \text{bdt-Some-var0-Zero})

apply auto

done

from \text{rnobdt-def rnot-dag vrNull hno-nNull}

have \text{rnobdt-Zero}: \text{rnobdt} = \text{Zero}

apply —

apply (drule \text{bdt-Some-var0-Zero})

apply auto

done

from \text{lnobdt-Zero lnobdt-def} have \text{bdt lnot var} = \text{Some Zero} by simp

with \text{lnot-dag vlNull}

have \text{lnot-Node}: \text{lnot} = (\text{Node \ Tip} (\text{low no}) \ \text{Tip})

by auto

from \text{rnobdt-Zero rnot-dag vrNull}

have \text{rnot-Node}: \text{rnot} = (\text{Node \ Tip} (\text{high no}) \ \text{Tip})

by auto

from \text{pret-dag no-in-pret} obtain not where

not-ex: Dag no low high not

apply —

apply (drule \text{dag-setof-exD})

apply auto

done

183
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 lnort
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 N\odes n ll
  by blast
show \forall no \in set-of rnort. repec no = no
proof
fix pt
assume pt-in-rnort: pt \in set-of rnort
with rnort-in-Nodesn have pt \in N\odes n ll
  by blast
with Nodesn-notin-lln have pt \notin (ll ! n)
  by auto
with repbc-nc have repb pt = repec pt
  by auto
with rnort-repb-no pt-in-rnort show repec pt = pt
  by auto
qed
qed
next
assume share-case-cond: repec (low no) \neq repec (high no)
with \( \text{hno-nNull} \) \( \text{hno-nNull} \)

have \( \text{share-case-cond-propto}: (\text{repc} \propto \text{low}) \ \text{no} \neq (\text{repc} \propto \text{high}) \ \text{no} \)

by \( (\text{simp add: null-comp-def}) \)

with \( \text{repshare} \) obtain

\( \text{rno-in-llbn}: \text{repc} \ \text{no} \in \text{set (ll ! n) and} \)

\( (\text{rno-eq-rno}: (\text{repc (repc no)} = (\text{repc no}) \ \text{and}) \)

\( (\text{twonodes-in-llbn-prop}: (\forall \text{no1} \in \text{set (ll ! n)}. \)

\( ((\text{repc} \propto \text{high}) \ \text{no1} = (\text{repc} \propto \text{high}) \ \text{no} \ \wedge \)

\( (\text{repc} \propto \text{low}) \ \text{no1} = (\text{repc} \propto \text{low}) \ \text{no} = (\text{repc no} = (\text{repc no1})) \)

by \( \text{auto} \)

from \( \text{wf-ll} \ \text{rno-in-llbn} \ \text{nsll} \)

have \( \text{varrepno-n}: \text{var} (\text{repc no}) = n \)

by \( (\text{simp add: wf-ll-def length-ll-eq}) \)

with \( \text{varno} \) have \( \text{varrep} \)

by \( \text{simp} \)

from \( \text{not-nort-prop-high} \ \text{not-nort-prop-low} \) have \( \text{repcn-prop: ?repcn-prop} \)

apply –

apply \( (\text{elim exE}) \)

apply \( (\text{rename-tac rnot lnot rnort lnort}) \)

apply \( (\text{rule-tac x=Node lnot no rnot in exI}) \)

apply \( (\text{rule-tac x=Node lnort (repc no) rnort in exI}) \)

apply \( (\text{elim conjE}) \)

apply \( (\text{intro conjI}) \)

prefer 7

apply \( (\text{elim exE}) \)

apply \( (\text{rename-tac rnot lnot rnort lnort rnobdt lnobdt rnorbdt lnorbdt}) \)

apply \( (\text{elim conjE}) \)

apply \( (\text{case-tac Suc 0 < var no}) \)

apply \( (\text{rule-tac x=(Bdt-Node lnobdt (var no) rnobdt) in exI}) \)

apply \( (\text{rule conjI}) \)

prefer 2

apply \( (\text{rule-tac x=(Bdt-Node lnorbdt (var (repc no)) rnorbdt) in exI}) \)

apply \( (\text{rule conjI}) \)

proof –

fix \( \text{rnot lnot rnort lnort} \)

assume \( \text{rnort-dag}: \)

\( \text{Dag (repc (high no)) (repb \propto \text{low}) (repb \propto \text{high}) rnort} \)

assume \( \text{lnort-dag}: \)

\( \text{Dag (repc (low no)) (repb \propto \text{low}) (repb \propto \text{high}) lnort} \)

assume \( \text{rnort-in-repNodes: set-of rnort} \subseteq \text{repb ' Nodes n ll} \)

assume \( \text{lnort-in-repNodes: set-of lnort} \subseteq \text{repb ' Nodes n ll} \)

from \( \text{rnort-in-repNodes} \ \text{repbNodes-in-Nodes} \)

have \( \text{rnort-in-Nodes: set-of rnort} \subseteq \text{Nodes n ll} \)

by \( \text{simp} \)

from \( \text{lnort-in-repNodes} \ \text{repbNodes-in-Nodes} \)

have \( \text{lnort-in-Nodes: set-of lnort} \subseteq \text{Nodes n ll} \)

by \( \text{simp} \)

from \( \text{rnort-in-Nodes} \)

185
have \text{nortx-in-Nodes}: \forall \ x \in \text{set-of nort}. \ x \in \text{Nodes n ll} 
\text{by blast}
\text{with wf-ll nsl}

have \forall \ x \in \text{set-of nort}. \ x \in \text{set-of pret} \land \text{var} \ x < n
\text{apply --}
\text{apply (rule ballI)}
\text{apply (rule wf-ll-Nodes-pret)}
\text{apply (auto simp add: length-ll-eq)}
\text{done}
\text{with pret-dag prebdt-pret nort-dag ord-pret wf-ll nsl}
\text{repbc-nc}

have \forall \ x \in \text{set-of nort}. \ (\text{repc } \bowtie \text{low}) \ x = (\text{repb } \bowtie \text{low}) \ x \land 
(\text{repc } \bowtie \text{high}) \ x = (\text{repb } \bowtie \text{high}) \ x
\text{apply --}
\text{apply (rule nort-null-comp)}
\text{apply (auto simp add: length-ll-eq)}
\text{done}

\text{then have Dag (repc \ (\text{high no}) \ (repc } \bowtie \text{low}) \ (repc } \bowtie \text{high) nort =}
\text{Dag (repc \ (\text{high no}) \ (repc } \bowtie \text{low}) \ (repc } \bowtie \text{high) nort} 
\text{apply --}
\text{apply (rule heaps-eq-Dag-eq)}
\text{apply assumption}
\text{done}
\text{with nort-dag}

\text{have nort-dag-repc:}
\text{Dag (repc \ (\text{high no}) \ (repc } \bowtie \text{low}) \ (repc } \bowtie \text{high) nort}
\text{by simp}
\text{from lnort-in-Nodes}

\text{have lnortx-in-Nodes: \forall \ x \in \text{set-of lnort}. \ x \in \text{Nodes n ll} 
\text{by blast}
\text{with wf-ll nsl}

have \forall \ x \in \text{set-of lnort}. \ x \in \text{set-of pret} \land \text{var} \ x < n
\text{apply --}
\text{apply (rule ballI)}
\text{apply (rule wf-ll-Nodes-pret)}
\text{apply (auto simp add: length-ll-eq)}
\text{done}
\text{with pret-dag prebdt-pret lnort-dag ord-pret wf-ll nsl}
\text{repbc-nc}

have \forall \ x \in \text{set-of lnort}. \ (\text{repc } \bowtie \text{low}) \ x = (\text{repb } \bowtie \text{low}) \ x \land 
(\text{repc } \bowtie \text{high}) \ x = (\text{repb } \bowtie \text{high}) \ x
\text{apply --}
\text{apply (rule nort-null-comp)}
\text{apply (auto simp add: length-ll-eq)}
\text{done}

\text{then have}
\text{Dag (repc \ (\text{low no}) \ (repc } \bowtie \text{low}) \ (repc } \bowtie \text{high) lnort =}
\text{Dag (repc \ (\text{low no}) \ (repc } \bowtie \text{low}) \ (repc } \bowtie \text{high) lnort}

186
apply —
apply (rule heaps-eq-Dag-eq)
apply assumption
done
with lnort-dag
have lnort-dag-repc:
  Dag (repc (low no)) (repc ∝ low) (repc ∝ high) lnort
  by simp
from lno-nNull hno-nNull
have propo-comp: (repc ∝ low) no = recp (low no) ∧
  (repc ∝ high) no = recp (high no)
  by (simp add: null-comp-def)
from rno-in-llbn twonodes-in-llbn-prop rrno-eq-rno
have (repc ∝ high) (repc no) = (repc ∝ high) no ∧
  (repc ∝ low) (repc no) = (repc ∝ low) no
  by simp
with propo-comp lnort-dag-repc rnort-dag-repc lno-nNull hno-nNull
  rnonN
show Dag(repc no)(repc ∝ low)(repc ∝ high)(Node lnort (repc no)
  rnort)
  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 lnot
assume rnort-dag:
  Dag (repc (high no)) (repb ∝ low) (repb ∝ high) rnort
assume lnort-dag:
  Dag (repc (low no)) (repb ∝ low) (repb ∝ 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 lrnort (repc (high no)) rrnort)
  by auto
from rhn-nNull lnot-dag obtain lnort rlnot where
  lnort-Node: lnort = (Node lnort (repc (low no)) rlnot)
  by auto
from twonodes-in-llbn-prop rrno-eq-rno rno-in-llbn hno-nNull
lno-nNull
have ((repc ∝ high) (repc no)) = recp (high no) ∧
  ((repc ∝ low) (repc no)) = recp (low no)
apply —
apply (erule-tac x=repc no in ballE)
apply (auto simp add: null-comp-def)
done
with share-case-cond
have \(((\text{repc} \propto \text{high}) \ (\text{repc} \propto \text{low})) \neq ((\text{repc} \propto \text{low}) \ (\text{repc} \propto \text{no}))\)
  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 \(lp=\text{repc} \ (\text{low} \ no)\) and \(rp=\text{repc} \ (\text{high} \ no)\) and
    \(\text{llt}=\text{lnort} \ and \ \text{rlt}=\text{rnort} \ and \ \text{lrt}=\text{lnort} \ and \ \text{rrt}=\text{rnort}\)
  in reduced-children-parent)
  apply auto
done
next
fix lnort rnort
assume lnort-dag:
  \(\text{Dag} \ ((\text{repc} \ (\text{low} \ no)) \ (\text{repb} \propto \text{low}) \ (\text{repb} \propto \text{high})) \ \text{lnort}\)
assume ord-lnort: ordered lnort var
assume rnort-dag:
  \(\text{Dag} \ ((\text{repc} \ (\text{high} \ no)) \ (\text{repb} \propto \text{low}) \ (\text{repb} \propto \text{high})) \ \text{rnort}\)
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 lnnort-in-Nodes: set-of lnort \(\subseteq \) Nodes n ll
  by simp
from lnort-in-repNodes repbNodes-in-Nodes
have rnort-in-Nodes: set-of rnort \(\subseteq \) Nodes n ll
  by simp
from rhn-nNull rnort-dag obtain lnnort rnort where
  lnort-Node: rnort = (Node lnnort \ (repc (high no)) \ rnort)
  by auto
from rhn-nNull lnort-dag obtain llnort lnort where
  lnort-Node: lnort = (Node llnort \ (repc \ (low \ no)) \ lnort)
  by auto
from lnort-dag rhn-nNull lnort-in-Nodes
have repc (low no) \(\in\) set-of lnort
  by auto
with lnort-in-Nodes
have repc (low no) \(\in\) Nodes n ll
  by blast
with wf-ll nsll
have vrlno-sn: var \((\text{repc} \ (\text{low} \ no))\) < \(n\)
  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) \(\in\) set-of rnort

188
by auto

with rnot-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 varrepmu-n vrhno-sn lnort-dag ord-lnort rnot-dag rnot-Node
lnort-Node ord-rnot

show ordered (Node lnort (repc no) rnot) var
  by auto

next

fix lnort rnot
assume lnort-in-Nodes: set-of lnort ⊆ repb ‘Nodes n ll
assume rnot-in-Nodes: set-of rnot ⊆ repb ‘Nodes n ll
from lnort-in-Nodes repbNodes-repcNodes
have lnort-in-repcNodes: set-of lnort ⊆ repb ‘Nodes n ll
  by simp
from rnot-in-Nodes repbNodes-repcNodes
have rnot-in-repcNodes: set-of rnot ⊆ repb ‘Nodes n ll
  by simp
have nNodessubset: Nodes n ll ⊆ Nodes (n+1) ll
  by (simp add: Nodes-subset)
then have rnot-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 rnot-in-repcNodes
show set-of (Node lnort (repc no) rnot) ⊆
  repc ‘Nodes (n + 1) ll
  apply simp
  apply blast
  done

next

fix rnot lnor lnort lnodl lnobdt lnorbdt lnorbdt
assume lnobdt-def: bdt lnor var = Some lnobdt
assume rnotbd-def: bdt rnot var = Some rnotbdt
assume rnorbd-def: bdt rnot var = Some rnorbdt
assume cong-rno-rnor: rnodl ~ rnorbdt
assume lnor-dag: Dag (low no) low high lnor
assume rnot-dag: Dag (high no) low high rnot
assume ¬ Suc 0 < var no
then have varnoseq1: var no = 0 ∨ var no = 1 by auto
show ∃ nobdt. bdt (Node lnor no rnot) var = Some nobdt ∧
  (∃ norbdt. bdt (Node lnort (repc no) rnot) var = Some norbdt ∧
  nobdt ~ 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 ≠ 0
  with varnoseq1 have vnoOne: var no = 1 by simp
  from pret-dag ord-pret no-in-pret lno-nNull hno-nNull
  have vlvrNull: var (low no) = 0 ∧ 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: ∃ 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 lnobdt-dag rnot-dag lnobdt-def
show bdt (Node lnot no rnot) var =
   Some (Bdt-Node lnobdt (var no) rnobdt) by simp
next
fix rnot lnobdt rnot lnot lnot rnot lnobdt rnobdt lnobdt rnorbdlt lnorbdlt
assume rnot-dag:
   Dag (repc (high no)) (repb ∝ low) (repb ∝ high) rnot
assume lnot-dag:
   Dag (repe (low no)) (repb ∝ low) (repb ∝ high) lnot
assume rnorbdlt-def: bdt rnot var = Some rnorbdlt
assume lnorbdlt-def: bdt lnot var = Some lnorbdlt
assume varno-bOne: Suc 0 < var no
with varno have Suc 0 < n by simp
with varnorm-n have Suc 0 < var (repe no) by simp
with rnorbdlt-def lnorbdlt-def
show bdt (Node lnot (repe no) rnot) var =
   Some (Bdt-Node lnorbdlt (var (repe no)) rnorbdlt)
   by simp
next
  fix rnorbd lnorbd rnorbd lnorbd
  assume lcong-eval: lnorbd ∼ lnorbd
  assume rcong-eval: rnorbd ∼ rnorbd
  from varno varrepno-n have var (repc no) = var no by simp
  with lcong-eval rcong-eval
  show Bdt-Node lnorbd (var no) rnorbd
    apply (unfold cong-eval-def)
    apply (rule ext)
    by simp

next
  fix rnot lnot rnort lnort
  assume lnort-repb: ∀ no ∈ set-of lnort. repb no = no
  assume rnort-repb: ∀ no ∈ set-of rnort. repb no = no
  assume rnort-in-repb-Nodesn: set-of rnort ⊆ repb ' Nodes n ll
  assume lnort-in-repb-Nodesn: set-of lnort ⊆ repb ' Nodes n ll
  from repbNodes-in-Nodes rnort-in-repb-Nodesn
  have rnort-in-Nodesn: set-of rnort ⊆ Nodes n ll
    by blast
  from repbNodes-in-Nodes lnort-in-repb-Nodesn
  have lnort-in-Nodesn: set-of lnort ⊆ Nodes n ll
    by blast
  have rnort-repc: ∀ no ∈ set-of rnort. repc no = no
  proof
    fix pt
    assume pt-in-rnort: pt ∈ set-of rnort
    with rnort-in-Nodesn have pt ∈ Nodes n ll
      by blast
    with Nodesn-notin-lln have pt ∈ set (ll ! n)
      by auto
    with repbc-nc have repb pt = repc pt
      by auto
    with rnort-repb pt-in-rnort show repc pt = pt
      by auto
    qed
  have lnort-repc: ∀ no ∈ set-of lnort. repc no = no
  proof
    fix pt
    assume pt-in-lnort: pt ∈ set-of lnort
    with lnort-in-Nodesn have pt ∈ Nodes n ll
      by blast
    with Nodesn-notin-lln have pt ∈ set (ll ! n)
      by auto
    with repbc-nc have repb pt = repc pt
      by auto
    with lnort-repb pt-in-lnort show repc pt = pt
      by auto
    qed

192
∀ no ∈ set-of (Node lnort (repc no) rnort). repc no = no

proof
  fix pt
  assume pt-in-rept: pt ∈ set-of (Node lnort (repc no) rnort)
  show repc pt = pt
  proof (cases pt ∈ set-of lnort)
    case True
    with lnort-repc show ?thesis
    by auto
  next
  assume pt-notin-lnort: pt /∈ set-of lnort
  show ?thesis
  proof (cases pt ∈ set-of rnort)
    case True
    with rnort-repc show ?thesis
    by auto
  next
  assume pt-notin-rnort: pt /∈ set-of rnort
  with pt-notin-lnort pt-in-rept
  have pt = repc no
  by simp
  with rrno-eq-rno show repc pt = pt
  by simp
  qed
  qed
  qed
  qed

  with varrep rrno-eq-rno show ?thesis by simp
  qed
  qed
  with rrnonN show ?thesis by simp
  qed
  qed

note while-while-prop="this"
from wf-ll nsll
have ∀ no ∈ Nodes n ll. no /∈ set (ll ! n)
  apply (simp add: Nodes-def length-ll-eq)
  apply clarify
  apply (erule-tac x = no in allE)
  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=x 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 repbc-dags-eq:
  Dags (repc ' Nodes n ll) (repc ∞ low) (repc ∞ high) =
  Dags (repb ' Nodes n ll) (repb ∞ low) (repb ∞ 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 ⊆ repc ' Nodes n ll
  assume x-t: x=t
  with t-in-repcNodes repcNodes-repbNodes
  show set-of x ⊆ repb ' Nodes n ll
    by simp

next
  fix x p t
  assume t-in-repcNodes: set-of t ⊆ repc ' Nodes n ll
  assume t-dag: Dag p (repc ∞ low) (repc ∞ high) t
  assume t-nTip: t ≠ Tip
  assume x-t: x=t
  from t-nTip t-dag have p ≠ 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-repcNodes t-dag t-nTip t-Node obtain q where
  rq-p: repc q = p and q-in-Nodes: q ∈ Nodes n ll
  apply simp

194
apply (elim conjE)
apply (erule imageE)
apply auto
done
from q-in-Nodes have repb q = repec q
  by (rule Nodes-n-rep-nc [rule-format])
with rq-p have repbq-p: repb q = p by simp
from q-in-Nodes
  have Dag (repb q) (repb ∞ low) (repb ∞ high) t =
    Dag (repc q) (repc ∞ low) (repc ∞ high) t
    by (rule Nodes-repbc-Dags-eq [rule-format])
with t-dag rq-p have Dag (repb q) (repb ∞ low) (repb ∞ high) t by simp
with repbq-p x-t show Dag p (repb ∞ low) (repb ∞ high) x
  by simp
next
fix x p t
assume t-in-repcNodes: set-of t ⊆ repb ' Nodes n ll
assume x-t: x=t
with t-in-repcNodes repbNodes-repcNodes
  show set-of x ⊆ repc ' Nodes n ll
    by simp
next
fix x p t
assume t-in-repcNodes: set-of t ⊆ repb ' Nodes n ll
assume t-dag: Dag p (repb ∞ low) (repb ∞ high) t
assume t-nTip: t ≠ Tip
assume x-t: x=t
from t-nTip t-dag have p ≠ 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-repcNodes t-dag t-nTip t-Node obtain q where
  rq-p: repb q = p and q-in-Nodes: q ∈ Nodes n ll
  apply simp
  apply (elim conjE)
  apply (erule imageE)
  apply auto
  done
from q-in-Nodes have repb q = repec q
  by (rule Nodes-n-rep-nc [rule-format])
with rq-p have repbq-p: repb q = p by simp
from q-in-Nodes
  have Dag (repb q) (repb ∞ low) (repb ∞ high) t =
    Dag (repc q) (repc ∞ low) (repc ∞ high) t
    by (rule Nodes-repbc-Dags-eq [rule-format])
with t-dag rq-p have Dag (repb q) (repb ∞ low) (repb ∞ high) t by simp
with repbq-p x-t show Dag p (repc ∝ low) (repc ∝ high) x
by simp

next
fix x p and t :: dag
assume x-t: x = t
assume t-nTip: t ≠ Tip
with x-t show x ≠ Tip by simp

next
fix x p and t :: dag
assume x-t: x = t
assume t-nTip: t ≠ Tip
with x-t show x ≠ Tip by simp

qed

from pret-dag wf-ll nsll
have null-notin-Nodes-Suc-n:
null∉Nodes (Suc n ll)
by − (rule Null-notin-Nodes, auto simp add: length-ll-eq)

{ fix t1 t2
assume t1-in-DagsNodesn:
t1 ∈ Dags (repc ' Nodes n ll) (repc ∝ low) (repc ∝ high)
assume t2-notin-DagsNodesn:
t2 ∉ Dags (repc ' Nodes n ll) (repc ∝ low) (repc ∝ high)
assume t2-in-DagsNodesSucn:
t2 ∈ Dags (repc ' Nodes (Suc n) ll) (repc ∝ low) (repc ∝ high)
assume isomorphic-dags-eq-asm:
∀ t1 t2. t1 ∈ Dags (repc ' Nodes n ll) (repc ∝ low) (repb ∝ high)
∧ t2 ∈ Dags (repc ' Nodes n ll) (repb ∝ low) (repb ∝ high)
→ isomorphic-dags-eq t1 t2 var
assume repb-Dags:
Dags (repc ' Nodes n ll) (repc ∝ low) (repc ∝ high) =
Dags (repb ' Nodes n ll) (repc ∝ low) (repb ∝ high)
from t1-in-DagsNodesn repb-Dags
have t1-repb-subnode:
t1 ∈ Dags (repb ' Nodes n ll) (repb ∝ low) (repb ∝ high)
by simp
from t2-in-DagsNodesSucn
have t2-in-DagsNodesSucn:
t2 ∈ Dags (repc ' Nodes (Suc n) ll) (repc ∝ low) (repc ∝ high)
by simp
from repbNodes-in-Nodes repbNodes-repcNodes
have repcNodesn-in-Nodesn: repc 'Nodes n ll ⊆ Nodes n ll
by simp
from t1-in-DagsNodesn obtain q where
Dag-q-Nodes-n:
Dag (repc q) (repc ∝ low) (repc ∝ high) t1 ∧ q ∈ Nodes n ll
proof (elim Dags.cases)
fix p t
assume t1-t: t1 = t
assume t-in-repcNodesn: set-of t ⊆ repc ' Nodes n ll
assume t-dag: Dag p (repc ∝ low) (repc ∝ high) t
assume $t \sim Tip$: $t \neq Tip$

assume obtain-prop: $\forall q. \text{Dag (repc } q) \ (\text{repc } \propto \text{ low}) \ (\text{repc } \propto \text{ high}) \ t1 \land q \in \text{Nodes n ll} \implies \text{thesis}$

from $t \sim Tip$ t-dag have $p \neq \text{ Null}$

apply 
apply (case-tac $p = \text{ Null}$)
apply auto

done

with $t \sim Tip$ t-dag obtain $lt \ rt$ where $t \sim Node: t = \text{Node } lt \ p \ rt$
by auto

from $t \sim \text{ in-repcNodesn}$ t-dag t-Node obtain $k$ where
rk-p: $\text{repc } k = p$ and k-in-Nodes: $k \in \text{Nodes n ll}$

apply simp
apply (elim conjE)
apply (erule imageE)
apply auto

done

with $t1 \sim t$ t-dag obtain-prop rk-p k-in-Nodes show $\text{thesis}$
by auto

qed

with wf-ll nsll have varq-sn: $(\text{var } q < n)$

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 $\forall q. q \in \text{ set-of } \text{pret} \implies q \in \text{ set } (\text{ll } ! \text{ var } q))$
apply (erule-tac $x = k$ in allE)
apply auto

done

from $\text{ Dag-q-Nodes-n}$ have q-in-Nodesn: $q \in \text{Nodes n ll}$
by simp
then have $\exists k < n. q \in \text{ set } (\text{ll } ! k)$
by (simp add: Nodes-def)
with wf-ll nsll have $q \notin \text{ set } (\text{ll } ! n)$

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: $\text{repc } q = \text{ repb } q$

apply 
apply (erule-tac $x = q$ in allE)
apply auto
done

from normalize-prop q-in-Nodesn have var (repb $q$) $\leq$ var $q$

apply 
apply (erule-tac $x = q$ in ballE)
apply auto
done

with repbc-q have var-repc-q: var (repc q) <= var q
  by simp

with varq-sm have var-repc-q-n: var (repc q) < n
  by simp

from Nodes-subset Dag-q-Nodes-n while-while-prop
have ord-t1-var-rep: ordered t1 var \land var (repc q) <= 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 (repc q) <= 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 recp ' Nodes (Suc n) ll
  assume t-nTip: t \neq Tip
  assume t2t: t2 = t

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

from t-in-repcNodes have x-in-repcNodesSucn:
  \forall x. x \in set-of t \rightarrow x \in recp ' 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 set-of t
  by auto

with x-in-repcNodesSucn have p \in recp ' Nodes (Suc n) ll
  by simp

then obtain a where repca-p: p=repc a and
  a-in-NodesSucn: a \in Nodes (Suc n) 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:
   \( \forall \, no \in \text{Nodes} \, (\text{Suc} \, n) \, \text{ll} \).
   \( \text{var} \, (\text{repc} \, no) \leq \text{var} \, no \)
   by auto

from while-while-prop have rep-rep-nort:
   \( \forall \, no \in \text{Nodes} \, (n + 1) \, \text{ll} \).
   \( (\exists \, nort. \, \text{Dag} \, (\text{repc} \, no) \, (\text{repc} \propto \text{low}) \, (\text{repc} \propto \text{high}) \, \text{nort}) \)
   by auto

from replNodes-in-Nodes null-notin-Nodes-Suc-n
have \( \forall \, no \in \text{Nodes} \, (n+1) \, \text{ll} \). \( \text{repc} \, no \neq \text{Null} \)
   by auto

with repl-rep-nort have \( \forall \, no \in \text{Nodes} \, (n+1) \, \text{ll} \).
   \( \text{repc} \, (\text{repc} \, no) = (\text{repc} \, no) \)

apply –
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 \( \forall \, no \in \text{set-of nort}. \, \text{repc} \, no = no) \)
apply auto
done

with t2-in-DagsNodesSucn t2-notin-DagsNodesn ord-t2 while-prop-part
wf-ll nsll replNodes-in-Nodes obtain a where
   t2-repc-dag: Dag (repc a) (repc \propto \text{low}) (repc \propto \text{high}) t2 and
   a-in-lln: a \in (ll ! n)
   apply –
   apply (erule 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 nsll 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-null: \( \text{repc} \, a \neq \text{Null} \)
   repp-reduce: (repc \propto \text{low}) \( a = (\text{repc} \propto \text{high}) \) a \land \text{low} a \neq \text{Null}
   \( \rightarrow \) \( \text{repc} \, a = (\text{repc} \propto \text{high}) \) a \land \text{low} a \neq \text{Null}
   repp-share: ((repc \propto \text{low}) a = (repc \propto \text{high}) a \rightarrow \text{low} a = \text{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) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rhigha-in-rNodesn: repc (high a) ∈ repc ' Nodes n ll
  by simp
from higha-in-Nodesn normalize-prop obtain rt where 
  rt-dag: Dag (repb (high a)) (repb ∞ low) (repb ∞ high) rt and
\textit{rt-in-repbNort}: set-of rt \subseteq \text{repb}\ '\text{Nodes} n \text{ ll}

apply 
apply \ (erule-tac \( x = \text{high} a \) \text{ in ballE})
apply auto
done
from \textit{rt-in-repbNort} \text{ repbNodes-repcNodes}
have \textit{rt-in-repcNodesn}: set-of rt \subseteq \text{repc}\ '\text{Nodes} n \text{ ll}
by blast
from \textit{rt-dag higha-in-Nodesn}
have \textit{report-dag}: \text{Dag} (\text{repc}\ (\text{high} a)) (\text{repc} \propto \text{low}) (\text{repc} \propto \text{high}) rt
apply 
apply (drule \text{Nodes-repbe-Dags-eq} [\text{rule-format}])
apply auto
done
have \textit{rt-nTip}: rt \neq \text{Tip}
proof 
have \text{repc} (\text{high} a) \neq \text{Null}
proof 
note rhgha-in-rNodesn
also have \text{repc}\ '\text{Nodes} n \text{ ll} \subseteq \text{repc}\ '\text{Nodes} (\text{Suc} n) \text{ ll}
using \text{Nodes-subset}
by blast
also have \ldots \subseteq \text{Nodes} (\text{Suc} n) \text{ ll}
using repcNodes-in-Nodes
by simp
finally show \?thesis
using null-notin-Nodes-Suc-n
by auto
qed
with \text{report-dag} show \?thesis by auto
qed
from \textit{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 wi-ll have low a \in set (\text{ll}! (\text{var} (low a)))
by (simp add: wi-ll-def)
with a-in-ln \( l2\)-repc-dag var-children-smaller \text{vara-n}
have \( \exists k < n. \ (\text{low} a) \in \text{set} \ (\text{ll}! k) \)
by auto
then have (low a) \in \text{Nodes} n \text{ ll}
by (simp add: \text{Nodes-def})
then have \textit{rlow-in-rNodesn}: \text{repc} (low a) \in \text{repc} '\text{Nodes} n \text{ ll}
by simp
show \?thesis
proof 
from \text{repp-reduce low-nNull highp-nNull \text{ red-case}}
have \text{repc-p-def}: \text{repc} a = \text{repc} (\text{high} a)
by (simp add: null-comp-def)
with rt-in-repcNodesn repcr-dag rhigha-in-rNodesn a-in-lhn t2-repc-dag
repc-p-def rt-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=rt)
  apply (auto simp add: Dag-unique)
  done
from t1-in-DagsNodesn t2-in-Dags-Nodesn repbc-dags-eq isomorphic-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: ¬ ((repc ∞ low) a = (repc ∞ high) a ∧ low a ≠ Null)
then
have share: (repc ∞ low) a ≠ (repc ∞ high) a ∨ low a = Null
  using [[simp-depth-limit=1]]
  by simp
with repp-share obtain
  ra-in-llbn: repc a ∈ set (ll ! n) and
  rra-ra: repc (repc a) = repc a and
  two-nodes-share: ∀ no1∈set (ll ! n).
    ((repc ∞ high) no1 = (repc ∞ high) a ∧
     (repc ∞ low) no1 = (repc ∞ low) a) = (repc a = repc no1))
  using [[simp-depth-limit=3]]
  by simp
from wf-ll ra-in-llbn nsll have var-repc-a-n: var (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 = Some bdt1
  assume bdt-t2: bdt t2 var = 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
note mixed-Dags-case = this
from repbc-dags-eq isomorphic-dags-eq
have dags-shared:
  \forall t1 t2. t1 \in Dags (repc ' Nodes (Suc n) ll) (repc \propto low) (repc \propto high) \land
  t2 \in Dags (repc ' Nodes (Suc n) ll) (repc \propto low) (repc \propto high)
  \rightarrow 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 Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)
  assume t2-in-Dagsn:
  t2 \in Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)
  assume isomorphic-dags-eq-asm:
  \forall t1 t2. t1 \in Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high) \land
  t2 \in Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)
  \rightarrow isomorphic-dags-eq t1 t2 var
  assume repb-repc-Dags:
    Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high) =
    Dags (repb ' Nodes n ll) (repb \propto low) (repb \propto 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 Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)
  assume t2-notin-DagsNodesn:
  t2 \notin Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)
  assume t2-in-DagsNodesSucn:
  t2 \in Dags (repc ' Nodes (Suc n) ll) (repc \propto low) (repc \propto high)
  assume isomorphic-dags-eq-asm:
  \forall t1 t2. t1 \in Dags (repb ' Nodes n ll) (repb \propto low) (repb \propto high) \land
  t2 \in Dags (repb ' Nodes n ll) (repb \propto low) (repb \propto high)
  \rightarrow isomorphic-dags-eq t1 t2 var
  assume repbc-Dags:
\begin{verbatim}
from (t1-in-DagsNodesn t2-notin-DagsNodesn) 
from (t1-in-DagsNodesn t2-in-DagsNodesSucn) 
\end{verbatim}
with repea-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: \( \forall no \in \text{Nodes} \ (\text{Suc } n) \ \ll \).
  var (repc no) \leq var no
by auto
from while-while-prop have rep-rep-nort:
  \( \forall no \in \text{Nodes} \ (n + 1) \ \ll \).
  (\exists nort. \text{Dag} (\text{repc } no) \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}) \ nort \land
  (\forall no \in \text{set-of } nort. \text{repc } no = no))
by auto
from repcNodes-in-Nodes null-notin-Nodes-Suc-n
have \( \forall no \in \text{Nodes} \ (n+1) \ \ll. \ \text{repc } no \neq \text{Null} \)
by auto
with rep-rep-nort
have rep-rep-no:
  \( \forall no \in \text{Nodes} \ (n+1) \ \ll. \ \text{repc } (\text{repc } no) = (\text{repc } no) \)
  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 \text{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 \( \forall no \in \text{set-of } nort. \ \text{repc } no = no \))
  apply auto
  done
with t1-in-DagsNodesSucn t1-notin-DagsNodesn ord-t1 while-prop-part

wf-ll
nsl\( \ Whatever \ a1 \ obtain \ a1 \ where \)
t1-repc-dag: \text{Dag} (\text{repc } a1) \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}) \ t1 \ and \ a1-in-lln: \ a1 \in \text{set} (\ll! n)
  apply −
  apply (drule restrict-root-Node)
  apply (auto simp add: length-ll-eq)

205
done

with wf-ll nsll have a1-in-pret: a1 ∈ set-of pret
by (simp add: wf-ll-def length-ll-eq)

from wf-ll nsll 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 ≠ Null and
repa1-reduce: (repc ∞ low) a1 = (repc ∞ high) a1 ∧ low a1 ≠ Null
repa1-share: ((repc ∞ low) a1 = (repc ∞ high) a1 → low a1 = Null)
(∀ no1∈set (ll ! n). ((repc ∞ high) no1 = (repc ∞ high) a1 ∧
(repc ∞ low) no1 = (repc ∞ 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 ⊆ repc ‘ Nodes (Suc n) ll
  assume t-nTip: t ≠ Tip
  assume t2t: t2 = t
  assume t-dag: Dag p (repc ∞ low) (repc ∞ high) t
  from t-in-repcNodes have x-in-repcNodesSucn:
    ∃ x. x ∈ set-of t → x ∈ repc ‘ Nodes (Suc n) ll
    apply −
    apply auto
    done
  from t-nTip t-dag have p ≠ 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 ∈ set-of t
by auto
with x-in-repcNodesSucn have p ∈ repc ‘ Nodes (Suc n) ll
by simp
then obtain a where
  repca-p: p = repc a and a-in-NodesSucn: a ∈ Nodes (Suc n) ll
by auto
with reca-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 rep-rep no t2-in-DagsNodesSucn t2-notin-DagsNodesn ord-t2 while-prop-part

wf-ll

nsll recaNodes-in-llNodes obtain a2 where
  t2-repc-dag: Dag (repc a2) (repc ∞ low) (repc ∞ high) t2 and
  a2-in-lln: a2 ∈ set (ll ! n)
  apply ¬
  apply (drule restrict-root-Node)
  apply (auto simp add: length-ll-eq)
  done
with wf-ll nsll have a2-in-pret: a2 ∈ set-of pret
  by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsll a2-in-lln have vara2-n: var a2 = n
  by (simp add: wf-ll-def length-ll-eq)
from a2-in-lln rep-prop obtain
  repa2-nNull: repc a2 ≠ Null and
  repa2-reduce: (repc ∞ low) a2 = (repc ∞ high) a2 ∧ low a2 ≠ Null
  → repc a2 = (repc ∞ high) a2 and
  repa2-share: ((repc ∞ low) a2 = (repc ∞ high) a2 → low a2 = Null)
  → repc a2 ∈ set (ll ! n) ∧ repc (repc a2) = repc a2 ∧
  (∀no1∈set (ll ! n). ((repc ∞ high) no1 = (repc ∞ high) a2 ∧
  (repc ∞ low) no1 = (repc ∞ low) a2) = (repc a2 = repc no1))
  using [[simp-depth-limit = 4]]
  by auto
from t2-repc-dag a2-in-lln repa2-nNull obtain lt2 rt2 where
  t2-Node: t2 = (Node lt2 (repc a2) rt2)
  by auto
show isomorphic-dags-eq t1 t2 var
proof (cases (repc ∞ low) a1 = (repc ∞ high) a1 ∧ low a1 ≠ Null)
case True
  note t1-red-cond=this
  with t1-red-cond have t1-red-case: (repc ∞ low) a1 = (repc ∞ high) a1
    by simp
from t1-red-cond have lowa1-nNull: low a1 ≠ Null
  by simp
with pretdag prebd-t-pret a1-in-pret have higha1-nNull: high a1 ≠ Null
  apply ¬
  apply (drule balanced-bdt)
  apply auto
  done

207
from pret-dag ord-pret a1-in-pret lowa1-nNull higha1-nNull
have var-children-smaller-a1: var (low a1) < var a1 ∧ var (high a1) < var a1
  apply –
  apply (rule var-ordered-children)
  apply auto
  done
from pret-dag a1-in-pret have a1-nNull: a1 ≠ Null
  apply –
  apply (rule set-of-nn [rule-format])
  apply auto
  done
with a1-in-pret higha1-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 rhigha1-in-rNodesn: repc (high a1) ∈ repc ‘ Nodes n ll
  by simp
from higha1-in-Nodesn normalize-prop obtain rt1 where
  rt1-dag: Dag (repb (high a1)) (repb ∝ low) (repb ∝ high) rt1 and
  rt1-in-repbNort: set-of rt1 ⊆ 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-repeNodesn: set-of rt1 ⊆ repc ‘Nodes n ll
  by blast
from rt1-dag higha1-in-Nodesn
have repcrt1-dag: Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) rt1
  apply –
  apply (drule Nodes-repc-Dags-eq [rule-format])
  apply auto
  done
have rt1-nTip: rt1 ≠ Tip
proof –
  have repc (high a1) ≠ Null
  proof –
    note rhigha1-in-rNodesn
    also have repc ‘Nodes n ll ⊆ repc ‘Nodes (Suc n) ll
    using Nodes-subset

208
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 repc1-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 repc1-dag rhigha1-in-rNodesn a1-in-lhn
t1-repc-dag repc-a1-def rt1-nTip
have t1-in-Dags-Nodesn:
  t1 ∈ Dags (repc ' Nodes n ll) (repc ∞ low) (repc ∞ high)
apply −
apply (rule DagsI)
apply simp
apply (subgoal-tac t1 = rt1)
apply (auto simp add: Dag-unique)
done
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-pre 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

209
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 ≠ Tip
proof –
  have repc (high a2) ≠ Null
  proof –
    note rhiga2-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-Node:n:
  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-Node:n t2-in-Dags-Node:n

show ?thesis
  by auto
next
assume t2-share-cond:
  ∼((repc ∝ low) a2 = (repc ∝ high) a2 ∧ low a2 ≠ Null)
from t1-in-Dags-Node:n t2-notin-DagsNode:n t2-in-DagsNode:Sucn
isomorphic-dags-eq repc-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
repa1-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)

211
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 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 reprt2-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 \( \text{repc 'Nodes n \ll \subseteq \text{repc 'Nodes (Suc n) \ll} \) using Nodes-subset
by blast
also have \( \ldots \subseteq \text{Nodes (Suc n) \ll} \) using repcNodes-in-Nodes
by simp
finally show \( \text{?thesis} \) using null-notin-Nodes-Suc-n
by auto
qed
with repcrt2-dag show \( \text{?thesis} \) by auto
qed
from repa2-reduce lowa2-nNull higha2-nNull t2-red-case
have \( \text{t2-a2-def: repc a2 = repc (high a2)} \) by (simp add: null-comp-def)
with \( \text{rt2-in-repcNodesn repcrt2-dag rhigha2-in-rNodesn a2-in-lln} \) t2-repc-dag repc-a2-def rt2-nTip
have \( \text{t2-in-Dags-Nodesn:} \)
\( t2 \in Dags (\text{repc ' Nodes n \ll}) (\text{repc \( \preceq \) low}) (\text{repc \( \preceq \) high}) \)
apply –
apply (rule DagsI)
apply simp
apply (subgoal-tac t2=rt2)
apply (auto simp add: Dag-unique)
done
from \( \text{t2-in-Dags-Nodesn t1-notin-DagsNodesn t1-in-DagsNodesSucn} \)
isomorphic-dags-eq repbc-dags-eq
have \( \text{isomorphic-dags-eq t2 t1 var} \)
apply –
apply (rule mixed-Dags-case)
apply auto
done
then show \( \text{?thesis} \) by (simp add: isomorphic-dags-eq-sym)
next
assume \( \text{t2-shared-cond:} \)
\( \neg ((\text{repc \( \preceq \) low}) a2 = (\text{repc \( \preceq \) high}) a2 \land \text{low a2} \neq \text{Null}) \)
with repa2-share obtain
repea2-in-lbnb: repc a2 \( \in \) set \( \text{(ll ! n)} \) and
repprea2: repc \( (\text{repc a2}) = \text{repc a2} \) and
twonodes-lbnb-a2: \( \forall \text{no1} \in \text{set (ll ! n)} \).
\((\text{repc \( \preceq \) high}) \text{no1} = (\text{repc \( \preceq \) high}) a2 \land \)
\((\text{repc \( \preceq \) low}) \text{no1} = (\text{repc \( \preceq \) low}) a2) = (\text{repc a2 = repc no1}) \)
using [[simp-depth-limit=2]]
by auto
from \( \text{twonodes-lbnb-a2 a1-in-lln} \)
have \( \text{share-a1-a2:} \)
\((\text{repc \( \preceq \) high}) a1 = (\text{repc \( \preceq \) high}) a2 \land \)
\[(\text{repc } \propto \text{ low}) \ a_1 = (\text{repc } \propto \text{ low}) \ a_2) = (\text{repc } a_2 = \text{repc } a_1)\]
by auto
from twonodes-llbn-a1 repca1-in-llbn reprepa1
have children-repc-eq-a1: (\text{repc } \propto \text{ high}) (\text{repc } a_1) = (\text{repc } \propto \text{ high}) a_1
\wedge
\[(\text{repc } \propto \text{ low}) (\text{repc } a_1) = (\text{repc } \propto \text{ low}) a_1\]
by auto
from twonodes-llbn-a2 repca2-in-llbn reprepa2
have children-repc-eq-a2: (\text{repc } \propto \text{ high}) (\text{repc } a_2) = (\text{repc } \propto \text{ high}) a_2
\wedge
\[(\text{repc } \propto \text{ low}) (\text{repc } a_2) = (\text{repc } \propto \text{ low}) a_2\]
by auto
from t1-Node t2-Node show ?thesis
proof (clarsimp simp add: isomorphic-dags-eq-def)
fix bdt1
assume t1-bdt: \text{bdt} (\text{Node lt1} (\text{repc } a_1) \ \text{rt1}) \ \text{var} = \text{Some \ bdt1}
assume t2-bdt: \text{bdt} (\text{Node lt2} (\text{repc } a_2) \ \text{rt2}) \ \text{var} = \text{Some \ bdt1}
show lt1 = lt2 \land \text{repc } a_1 = \text{repc } a_2 \land rt1 = rt2
proof (cases bdt1)
case Zero
with t1-bdt t1-Node obtain
lt1-Tip: \text{lt1} = \text{Tip} \ \text{and}
rt1-Tip: \text{rt1} = \text{Tip}
by simp
from Zero t2-bdt t2-Node obtain
lt2-Tip: \text{lt2} = \text{Tip} \ \text{and}
rt2-Tip: \text{rt2} = \text{Tip}
by simp
from t1-repc-dag t1-Node lt1-Tip have (\text{repc } \propto \text{ low}) (\text{repc } a_1) = Null
by simp
with children-repc-eq-a1
have repc-low-a1-Null: (\text{repc } \propto \text{ low}) a_1 = Null
by simp
from t1-repc-dag t1-Node rt1-Tip
have (\text{repc } \propto \text{ high}) (\text{repc } a_1) = Null
by simp
with children-repc-eq-a1
have repc-high-a1-Null: (\text{repc } \propto \text{ high}) a_1 = Null
by simp
from t2-repc-dag t2-Node lt2-Tip have (\text{repc } \propto \text{ low}) (\text{repc } a_2) = Null
by simp
with children-repc-eq-a2
have repc-low-a2-Null: (\text{repc } \propto \text{ low}) a_2 = Null
by simp
from t2-repc-dag t2-Node rt2-Tip
have (\text{repc } \propto \text{ high}) (\text{repc } a_2) = Null
by simp

214
with children-repc-eq-a2
have repc-high-a2-Null: (repc ∝ 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 ∝ low) (repc a1) = Null
  by simp
with children-repc-eq-a1
have repc-low-a1-Null: (repc ∝ low) a1 = Null
  by simp
from t1-repc-dag t1-Node rt1-Tip have (repc ∝ high) (repc a1) = Null
  by simp
with children-repc-eq-a1
have repc-high-a1-Null: (repc ∝ high) a1 = Null
  by simp
from t2-repc-dag t2-Node lt2-Tip have (repc ∝ low) (repc a2) = Null
  by simp
with children-repc-eq-a2
have repc-low-a2-Null: (repc ∝ low) a2 = Null
  by simp
from t2-repc-dag t2-Node rt2-Tip have (repc ∝ high) (repc a2) = Null
  by simp
with children-repc-eq-a2
have repc-high-a2-Null: (repc ∝ 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 lbdt v rbdt)
ote bdt-Node-case = this
with t1-bdt t1-Node obtain
  lbdt-def-lt1: bdt lt1 var = Some lbdt and
  rbdt-def-rt1: bdt rt1 var = Some rbdt
  by auto
from t2-bdt bdt-Node-case t2-Node obtain
  lbdt-def-lt2: bdt lt2 var = Some lbdt and
  rbdt-def-rt2: bdt rt2 var = Some rbdt
  by auto
from lbdt-def-lt1 t1-Node t1-repc-dag children-repc-eq-a1 have (repc ∝ low) a1 ≠ Null
  by auto
  then have low-a1-nNull: (low a1) ≠ Null
    by (auto simp: null-comp-def)
from rbdt-def-rt1 t1-Node t1-repc-dag children-repc-eq-a1 have (repc ∝ high) a1 ≠ Null
  by auto
  then have high-a1-nNull: (high a1) ≠ Null
    by (auto simp: null-comp-def)
from lbdt-def-lt2 t2-Node t2-repc-dag children-repc-eq-a2 have (repc ∝ low) a2 ≠ Null
  by auto
  then have low-a2-nNull: (low a2) ≠ Null
    by (auto simp: null-comp-def)
from rbdt-def-rt2 t2-Node t2-repc-dag children-repc-eq-a2 have (repc ∝ high) a2 ≠ Null
  by auto
  then have high-a2-nNull: (high a2) ≠ 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 ∧ var (high a1) < var a1
      apply
      apply (rule var-ordered-children)
      apply auto
      done
from pret-dag a1-in-pret have a1-nNull: a1 ≠ 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 var-a1-n
have ∃ k<n. (high a1) ∈ set (ll ! k)
  by auto
then have high1-in-Nodesn: (high a1) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have rhigh1-in-rNodesn:
  repc (high a1) ∈ repc `Nodes n ll
  by simp
from high1-in-Nodesn normalize-prop obtain r1lh where
  r1l-dag: Dag (repb (high a1)) (repb ∝ low) (repb ∝ high) r1lh
and
  r1l-repbNort: set-of r1lh ⊆ repb `Nodes n ll
  apply –
  apply (erule-tac x=high a1 in ballE)
  apply auto
  done
from r1l-repbNort repbNodes-repcNodes
have r1l-repcNodesn: set-of r1lh ⊆ repc `Nodes n ll
  by blast
from r1l-dag high1-in-Nodesn
have repcrt1-dag:
  Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) r1lh
  apply –
  apply (drule Nodes-repc-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) r1
  by (auto simp add: null-comp-def)
with repcrt1-dag have r1lh-r1l: r1lh = r1l by (simp add: Dag-unique)
have r1l-nTip: r1l ≠ Tip
proof –
  have repc (high a1) ≠ Null
  proof –
    note rhigh1-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

217
using null-notin-Nodes-Suc-n

by auto

qed

with repcrt1-dag rt1h-rt1 show ?thesis by auto

qed

with rt1-in-repcNodren repcrt1-dag rhigha1-in-rNodren a1-in-ln
lt1-repc-dag rt1h-rt1

have rt1-in-Dags-Nodren:
  rt1 \in Dags (repc ' Nodren n ll) (repc \propto low) (repc \propto 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-ln ln-repc-dag var-children-smaller-a1 vara1-n

have \exists k < n. (low a1) \in set (ll ! k)
  by auto

then have lowa1-in-Nodren: (low a1) \in Nodren n ll
  by (simp add: Nodren-def)

then have rlowa1-in-rNodren: repc (low a1) \in repc ' Nodren n ll
  by simp

from lowa1-in-Nodren normalize-prop obtain lt1h where
lt1-dag: Dag (repc (low a1)) (repc \propto low) (repc \propto high) lt1h and
lt1-in-repbNort: set-of lt1h \subseteq repc 'Nodren n ll
  apply --
  apply (erule-tac x=low a1 in ballE)
  apply auto
  done

from lt1-in-repbNort repbNodren-repcNodren

have lt1-in-repcNodren: set-of lt1h \subseteq repc 'Nodren n ll
  by blast

from lt1-dag lowa1-in-Nodren

have repclt1-dag: Dag (repc (low a1)) (repc \propto low) (repc \propto high)
lth

  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

218
have \( \text{Dag} \left( \text{repc} \left( \text{low a1} \right) \right) \left( \text{repc} \propto \text{low} \right) \left( \text{repc} \propto \text{high} \right) \lt_1 \)
by (auto simp add: null-comp-def)

with \( \text{repclt1-dag} \) have \( \lt_1h \lt_1 = \lt_1 \) by (simp add: Dag-unique)

have \( \lt_1h \neq \text{Tip} \)

proof

have \( \text{repc} \left( \text{low a1} \right) \neq \text{Null} \)

proof

note rlowa1-in-rNodesn
also have
\( \text{repc} ' \text{Nodes} n \ll \subseteq \text{repc} ' \text{Nodes} \ (\text{Suc} \ n) \ll \)
using Nodes-subset
by blast
also have \( \ldots \subseteq \text{Nodes} \ (\text{Suc} \ n) \ll \)
using repcNodes-in-Nodes
by simp

finally show \( \text{thesis} \)
using null-notin-Nodes-Suc-n
by auto

qed

with \( \text{repclt1-dag} \) \( \lt_1h = \lt_1 \) show \( \text{thesis} \) by auto

qed

with \( \lt_1h = \lt_1 \) \( \text{repclt1-dag} \) \( \text{rlowa1-in-rNodesn} \) \( \text{a1-in-lln} \)
\( \text{lt1-repc-dag} \) \( \lt_1h = \lt_1 \)

have \( \lt_1 \in \text{Dags} \left( \text{repc} ' \text{Nodes} n \ll \right) \left( \text{repc} \propto \text{low} \right) \left( \text{repc} \propto \text{high} \right) \)
apply –
apply (rule DagsI)
apply auto
done

from \( \text{pret-dag} \) \( \text{ord-pret} \) \( \text{a2-in-pret} \) \( \text{low-a2-nNull} \) \( \text{high-a2-nNull} \)

have \( \text{var-children-smaller-a2} : \)
\( \text{var} \ (\text{low a2}) < \text{var} \ (\text{a2}) \land \text{var} \ (\text{high a2}) < \text{var} \ (\text{a2}) \)
apply –
apply (rule var-ordered-children)
apply auto
done

from \( \text{pret-dag} \) \( \text{a2-in-pret} \) have \( \text{a2-nNull} : \text{a2} \neq \text{Null} \)
apply –
apply (rule set-of-nn [rule-format])
apply auto
done

with \( \text{a2-in-pret} \) \( \text{high-a2-nNull} \) \( \text{pret-dag} \) have high a2 \( \in \) set-of pret
apply –
apply (erule subelem-set-of-high)

219
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 (repb (high a2)) (repb ∝ low) (repb ∝ 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 (erule 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

220
with \texttt{repcrt2-dag} \texttt{rt2h-rt2} \texttt{show} \texttt{thesis} \texttt{by} \texttt{auto}

\begin{verbatim}
from a2-nNull a2-in-pret low-a2-nNull pret-dag
have low a2 \in \text{set-of pret}
  apply –
  apply (drule subelem-set-of-low)
  apply auto
done

with \texttt{wf-ll}
 have low a2 \in \text{set} (ll ! (\text{var} (low a2))]
  by (simp add: \text{wf-ll-def})

with \texttt{a2-in-ltn} t2-repc-dag var-children-smaller-a2 vara2-n
have \( \exists k<n. (\text{low a2}) \in \text{set} (ll ! k) \)
  by auto
then have \texttt{lowa2-in-Nodesn}: (low a2) \in \text{Nodes n ll}
  by (simp add: Nodes-def)

then have \texttt{riowa2-in-rNodesn}: repc (low a2) \in repc ' \text{Nodes n ll}
  by simp
from \texttt{lowa2-in-Nodesn normalize-prop obtain lt2h where}
\texttt{lt2-dag}: Dag (repb (low a2)) (repb \in low) (repb \in high) lt2h and
\texttt{lt2-in-repbNort}: set-of lt2h \subseteq repb 'Nodes n ll
  apply –
  apply (erule-tac x=\text{low a2} in ballE)
  apply auto
done
from \texttt{lt2-in-repbNort repbNodes-repcNodes}
have \texttt{lt2-in-repctNord}: set-of lt2h \subseteq repc 'Nodes n ll
  by blast
from \texttt{lt2-dag lowa2-in-Nodesn}
have \texttt{repc\texttt{lt2-dag}: Dag (repc (low a2)) (repc \in low) (repc \in high)}
  apply –
  apply (drule Nodes-repbc-Dags-eq [rule-format])
  apply auto
done
from \texttt{t2-Node t2-repc-dag low-a2-nNull children-repceq-a2}
have Dag (repc (low a2)) (repc \in low) (repc \in high) lt2h
  by (auto simp add: null-comp-def)
\end{verbatim}
with repc2-dag have \( lt2h \cdot lt2 \): \( lt2h = lt2 \) by (simp add: Dag-unique)

have \( lt2 \cdot Tip \): \( lt2 \neq Tip \)
proof −
  have \( repc \text{ (low a2)} \neq \text{Null} \)
proof −
  note rlowa2-in-rNodesn
also have
    \( repc \text{ 'Nodes n ll} \subseteq \text{repc 'Nodes (Suc n) ll} \)
using Nodes-subset
by blast
also have \( \ldots \subseteq \text{Nodes (Suc n) ll} \)
using repcNodes-in-Nodes
by simp
finally show \(?thesis \)
using null-notin-Nodes-Suc-n
by auto
qed
with repc2-dag \( lt2h \cdot lt2 \) show \(?thesis \) by auto
qed

with \( lt2 \cdot \text{in-repcNodesn} \) repc2-dag rlowa2-in-rNodesn a2-in-lln
\( t2 \cdot \text{repc-dag} \) \( lt2h \cdot lt2 \)
have \( lt2 \cdot \text{in-Dags-Nodesn} \):
  \( lt2 \in \text{Dags (repc ' Nodes n ll) (repc \varnothing \text{ low}) (repc \varnothing \text{ high})} \)
apply −
apply (rule DagsI)
apply auto
done

\[ \text{repbc-dags-eq} \]

have \( \text{shared-lt1-lt2}: \text{isomorphic-dags-eq} \text{ lt1 \ lt2 \ var} \)
by auto
from isomorphic-dags-eq \( rt1 \cdot \text{in-Dags-Nodesn} \) \( rt2 \cdot \text{in-Dags-Nodesn} \)

\[ \text{repbc-dags-eq} \]

have \( \text{shared-rt1-rt2}: \text{isomorphic-dags-eq} \text{ rt1 \ rt2 \ var} \)
by auto

from \( \text{shared-lt1-lt2} \) lbdt-def-lt1 lbdt-def-lt2 have \( lt1 \cdot lt2 \): \( lt1 = lt2 \)
by (auto simp add: isomorphic-dags-eq-def)
then have \( \text{root-lt1-lt2}: \text{root lt1 = root lt2} \)
by auto
from \( lt1 \cdot \text{nTip} \) \( t1 \cdot \text{repc-dag} \) \( t1 \cdot \text{Node} \) have \( \text{repc \varnothing \text{ low}) (repc a\varnothing) \neq \text{Null} \)
by auto
with \( lt1 \cdot \text{nTip} \) \( t1 \cdot \text{repc-dag} \) \( t1 \cdot \text{Node} \) obtain \( llt1 \) \( lt1p \) \( rlt1 \) where
  \( t1 \cdot \text{Node}: \text{lt1 = Node llt1 lt1p rlt1} \)
by auto
with \( t1 \cdot \text{repc-dag} \) \( t1 \cdot \text{Node} \) children-repc-eq-a\varnothing \( lt1 \cdot \text{nTip} \)

222
have root-lt1: root lt1 = (repc ∝ low) a1
  by auto
from lt2-nTip t2-repc-dag t2-Node have (repc ∝ low) (repc a2) ≠ Null
  by auto
with lt2-nTip t2-repc-dag t2-Node obtain llt2 llt2p rlt2 where
  llt2-Node: llt2 = Node llt2 llt2p rlt2
  by auto
with t2-repc-dag t2-Node children-repc-eq-a2 lt2-nTip
have root-lt2: root lt2 = (repc ∝ low) a2
  by auto
from root-lt1-lt2 root-lt2 root-lt1
have low-a1-a2: (repc ∝ low) a1 = (repc ∝ low) a2
  by auto
from shared-rt1-rt2 rbdt-def-rt1 rbdt-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 ∝ high) (repc a1) ≠ Null
  by auto
with rt1-nTip t1-repc-dag t1-Node obtain lrt1 lrt1p rrt1 where
  rt1-Node: rt1 = Node lrt1 lrt1p rrt1
  by auto
with t1-repc-dag t1-Node children-repc-eq-a1 rt1-nTip
have root-rt1: root rt1 = (repc ∝ high) a1
  by auto
from rt2-nTip t2-repc-dag t2-Node
have (repc ∝ high) (repc a2) ≠ Null
  by auto
with rt2-nTip t2-repc-dag t2-Node obtain lrt2 lrt2p rrt2 where
  rt2-Node: rt2 = Node lrt2 lrt2p rrt2
  by auto
with t2-repc-dag t2-Node children-repc-eq-a2 rt2-nTip
have root-rt2: root rt2 = (repc ∝ high) a2
  by auto
from root-rt1-rt2 root-rt2 root-rt1
have high-a1-a2: (repc ∝ high) a1 = (repc ∝ high) a2
  by auto
from low-a1-a2 high-a1-a2 share-a1-a2
have repc a1 = repc a2
  by auto
with lt1-lt2 rt1-rt2 show ?thesis
  by auto
qed
qed
qed
qed
qed

223
\begin{verbatim}
from termi days-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
\end{verbatim}

References