Abstract

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

Veronika Ortner and Norbert Schirmer

May 27, 2015

Contents

1 Introduction 3
2 BDD Abstractions 3
3 General Lemmas on BDD Abstractions 11
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: \( \forall t. [\text{subdag s t; subdag t s}] \Rightarrow P \)
  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 ≤ (t::dag) ⟷ s=t ∨ s < t

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

dag-less-le:

lemma le-dag-antisym:
  fixes x::dag and y
  assumes x-y: x ≤ y and y-x: y ≤ x
  shows x = y
proof (cases x=y)
  case True thus thesis .
next
  case False
  with x-y y-x show thesis
  by (auto simp add: le-dag-def dest: subdag-neq)
qed

instance
  by default (auto simp add: dag-less-le le-dag-def intro: subdag-not-sym)
end

lemma less-dag-Tip [simp]: ¬ (x < Tip)
  by (simp add: less-dag-def)
lemma less-dag-Node: \(x < (\text{Node } l a r) = (x \leq l \lor x \leq r)\)
by (auto simp add: order-le-less less-dag-def)

lemma less-dag-Node': \(x < (\text{Node } l a r) = (x=\text{l} \lor x=\text{r} \lor x < l \lor x < r)\)
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 set-of x \(\subset\) 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 \(\notin\) set-of l a \(\notin\) set-of r
by simp
from Node obtain l-less-y: \(l < y\) and r-less-y: \(r < y\)
by (simp add: less-dag-Node)
from Node a obtain
l-subset-y: set-of l \(\subseteq\) set-of y and
r-subset-y: set-of r \(\subseteq\) set-of y
by auto
from Node x-less-y have x=l \(\lor\) x=r \(\lor\) x < l \(\lor\) 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 $\text{set-of } x \subseteq \text{set-of } l$
    by (rule less-dag-set-of)
  also note $l$-subset-$y$
  finally show $?thesis$.
next
  assume $x < r$
  hence $\text{set-of } x \subseteq \text{set-of } r$
    by (rule less-dag-set-of)
  also note $r$-subset-$y$
  finally show $?thesis$.
qed

lemma in-set-of-decomp:
  $p \in \text{set-of } t = (\exists l r. \ t = (\text{Node } l \ p \ r) \lor \text{subdag } t (\text{Node } l \ p \ r))$
(is $?A = ?B$)
proof
  assume $?A$ thus $?B$
    by (induct $t$) auto
next
  assume $?B$ thus $?A$
    by (induct $t$) auto
qed

primrec Dag:: $\text{ref} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{dag} \Rightarrow \text{bool}$
where
Dag $p$ $l$ $r$ Tip = ($p$ = Null) |
Dag $p$ $l$ $r$ (Node $l t$ $a$ $r t$) = ($p$ = $a$ \land $p$ \neq Null \land
Dag ($l$ $p$) $l$ $r$ $l t$ \land Dag ($r$ $p$) $l$ $r$ $r t$)

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

lemma Dag-Ref [simp]:
  $p \neq \text{Null} \Longrightarrow \text{Dag } p \ l \ r \ t = (\exists l t r t = \text{Node } l t p r t \land
Dag (l p) \ l \ r \ l t \land Dag (r p) \ l \ r \ r t)$
  by (cases $t$) auto

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

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

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

by (induct t) auto

**lemma** Dag-upd-same-l-lemma: \( \forall p. p \neq \text{Null} \implies \neg \text{Dag } p \ l \ (r(p := p)) \ r \ t \)

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]: \( \text{Dag } p \ l \ (r(p := p)) \ r \ t = (p = \text{Null} \land t = \text{Tip}) \)

apply (cases p = Null)
apply simp
apply (fast dest: Dag-upd-same-l-lemma)
done

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

**lemma** Dag-upd-same-r-lemma: \( \forall p. p \neq \text{Null} \implies \neg \text{Dag } p \ l \ r \ (r(p := p)) \ t \)

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]: \( \text{Dag } p \ l \ r \ (r(p := p)) \ t = (p = \text{Null} \land t = \text{Tip}) \)

apply (cases p = Null)
apply simp
apply (fast dest: Dag-upd-same-r-lemma)
done

**lemma** Dag-update-l-new [simp]: \( \text{set-of } t \subseteq \text{set alloc} \) \( \implies \text{Dag } p \ l \ (l(new (set alloc) := x)) \ r \ t = \text{Dag } p \ l \ r \ t \)

by (rule notin-Dag-update-l) fastforce

**lemma** Dag-update-r-new [simp]: \( \text{set-of } t \subseteq \text{set alloc} \) \( \implies \text{Dag } p \ l \ (r(new (set alloc) := x)) \ t = \text{Dag } p \ l \ r \ t \)

by (rule notin-Dag-update-r) fastforce

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

by simp

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

by simp

8
lemma Dag-unique: $\Lambda p t2. \text{Dag} p l r t1 \implies \text{Dag} p l r t2 \implies t1 = t2$
   by (induct t1) auto

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

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

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

lemma Dag-root-not-in-subdag-r [simp, intro]:

assumes \( \text{Dag} (r \ p) \ l \ r \ t \)
shows \( p \notin \text{set-of} \ t \)
proof – 

\{ 
fix \( l t \ rt \)
assume \( t: t = \text{Node} \ l t \ p \ rt \)
from \( \text{assms} \) have \( \text{Dag} (r \ p) \ l \ r \ rt \)
  by (clarsimp simp only: \( t \text{ Dag.simps} \))
with \( \text{assms} \) have \( t=rt \)
  by (rule \text{Dag-unique})
with \( t \) have \( \text{False} \)
  by simp
\}
moreover 
\{ 
fix \( l t \ rt \)
assume \( \text{subdag} \): \( \text{subdag} \ t \ (\text{Node} \ l t \ p \ rt) \)
with \( \text{assms} \) obtain \( q \) where \( \text{Dag} \ q \ l \ r \ (\text{Node} \ l t \ p \ rt) \)
  by (rule \text{Dag-subdag} [elim-format]) improver
hence \( \text{Dag} (r \ p) \ l \ r \ rt \)
  by auto
with \( \text{assms} \) have \( t=rt \)
  by (rule \text{Dag-unique})
moreover 
have \( \text{subdag} \ t \ rt \)
proof – 
  note \( \text{subdag} \)
  also have \( \text{subdag} \ (\text{Node} \ l t \ p \ rt) \ rt \)
  by simp
finally show \( ?\text{thesis} \).
qed
ultimately have \( \text{False} \)
  by (simp add: \text{subdag-neq})
\}
ultimately show \( ?\text{thesis} \)
  by (auto simp add: in-set-of-decomp)
qed

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

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

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

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

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

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

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

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

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

lemma Dag-conv-isDag-dag: \[ \text{Dag } p \ l \ r \ t = (\text{isDag } p \ l \ r \ t \implies \text{dag } p \ l \ r \ t) \]
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)

end

3 General Lemmas on BDD Abstractions
**definition** subdag-eq:: dag ⇒ dag ⇒ bool where

\[
\text{subdag-eq } t_1, t_2 = (t_1 = t_2 \lor \text{subdag } t_1, t_2)
\]

**primrec** root :: dag ⇒ ref where

\[
\text{root Tip } = \text{Null} \\
\text{root } (\text{Node } l \ a \ r) = a
\]

**fun** isLeaf :: dag ⇒ bool where

\[
\text{isLeaf Tip } = \text{False} \\
\text{isLeaf } (\text{Node } \text{Tip } v \text{Tip}) = \text{True} \\
\text{isLeaf } (\text{Node } l \ v_1 \ r \ v_2 \text{Tip}) = \text{False} \\
\text{isLeaf } (\text{Node } \text{Tip } v_1 \ (\text{Node } l \ v_2 \ r)) = \text{False}
\]

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

**fun** bdt-fn :: dag ⇒ (ref ⇒ nat) ⇒ bdt option where

\[
\text{bdt-fn Tip } = (\lambda \text{bdtvar}. \text{None}) \\
\text{bdt-fn } (\text{Node } \text{Tip } vref \text{Tip}) = (\lambda \text{bdtvar}. \\
\quad \begin{cases} 
\text{if } (\text{bdtvar } vref = 0) & \text{then Some } \text{Zero} \\
\text{else if } (\text{bdtvar } vref = 1) & \text{then Some } \text{One} \\
\text{else } \text{None} 
\end{cases}) \\
\text{bdt-fn } (\text{Node } \text{Tip } vref \text{Tip}) = (\lambda \text{bdtvar}. \\
\quad \begin{cases} 
\text{if } (\text{bdtvar } vref = 0 \lor \text{bdtvar } vref = 1) & \text{then None} \\
\text{else } \text{None} 
\end{cases})
\]

| bdt-fn (Node Tip vref Tip) = (λbdtvar . None) \\
| bdt-fn (Node (Node l vref1 r) vref Tip) = (λbdtvar . None) \\
| bdt-fn (Node (Node l vref1 r1) vref (Node l2 vref2 r2)) = (λbdtvar . \\
\quad \begin{cases} 
\text{if } (\text{bdtvar } vref = 0 \lor \text{bdtvar } vref = 1) & \text{then None} \\
\text{else } \text{None} 
\end{cases})
\]

**abbreviation** bdt == bdt-fn

**primrec** eval :: bdt ⇒ bool list ⇒ bool where

\[
\text{eval Zero } env = \text{False} |
\text{eval One } env = \text{True} |
\text{eval } (\text{Bdt-Node } l \ v \ r) \ env = (\text{if } (env ! v) \text{ then eval } r \text{ env else eval } l \text{ env})
\]

12
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
done

lemma bdt-Some-One-iff [simp]:
(bdt t var = Some One) = (∃ p. t = Node Tip p Tip ∧ var p = t)
apply (induct t rule: bdt-fn.induct)
apply (auto split: option.splits)
done

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

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

lemma \texttt{balanced-bdt}:
\[
\forall \ p \ bdt1. \; [ \; \texttt{Dag \ p \ low \ high \ t}; \; \texttt{bdt \ t \ \var} = \texttt{Some \ bdt1}; \; \texttt{no} \in \texttt{set-of \ t} \; ]
\Rightarrow (\texttt{low \ no} = \texttt{Null}) = (\texttt{high \ no} = \texttt{Null})
\]
proof (induct \ t)
then show ?case by simp
next
case \texttt{(Node \ lt \ a \ rt)}
note \texttt{NN= \ this}
have \texttt{bdt1: \ bdt (Node \ lt \ a \ rt) \ \var = \texttt{Some \ bdt1} \ by \ fact}
have \texttt{no-in-t: \ no \in \texttt{set-of \ (Node \ lt \ a \ rt)} \ by \ fact}
have \texttt{p-tree: \ Dag \ p \ low \ high \ (Node \ lt \ a \ rt) \ by \ fact}
from \texttt{Node.prems obtain}
lit: \texttt{Dag \ (low \ p)} \ low \ high \ \texttt{lt} \ and
rt: \texttt{Dag \ (high \ p)} \ low \ high \ \texttt{rt}
by auto
show ?case
proof (cases \texttt{lt})
case \texttt{(Node \ llt \ l \ rlt)}
note \texttt{Nlt= \ this}
show ?thesis
proof (cases \texttt{rt})
case \texttt{(Node \ lrt \ r \ rrt)}
note \texttt{Nrt= \ this}
from \texttt{Nlt \ Nrt \ bdt1 \ obtain \ lbdt \ rbdt where}
\texttt{lbdt-def: \ bdt \ lt \ \var} = \texttt{Some \ lbdt \ and}
\texttt{rbdt-def: \ bdt \ rt \ \var} = \texttt{Some \ rbdt \ and}
\texttt{bdt1-def: \ bdt1 = Bdt-Node \ lbdt \ (var \ a) \ rbdt}
by (auto split: \texttt{split-if-asm option.splits})
from \texttt{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 \( \text{dag-in-level} \) t levellist var = (t ≠ \text{Tip} ⨸ \\
(∀ st. \text{subdag-eq} t st ⨸ \text{root} st ∈ \text{set} (levellist ! (\text{var} (\text{root} st)))))

**Lemma:** set-of-nn: [ \[ \text{Dag} p \ \text{low} \ \text{high} \ t; \ n ∈ \text{set-of} t \] ] ⨸ n ≠ \text{Null}
apply (induct t)
apply simp
apply auto
done

**Lemma:** subnodes-ordered [rule-format]:
\( ∀ p. \ n ∈ \text{set-of} t ⨸ \text{Dag} p \ \text{low} \ \text{high} \ t ⨸ \text{ordered} t \ \text{var} ⨸ \text{var} n ⨸ \text{var} p \)
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 simp
apply (case-tac t2)
apply simp
apply fastforce
apply (case-tac t2)
apply fastforce
done

**Lemma:** ordered-set-of:
\( ∀ x. \ [ \text{ordered} t \ \text{var}; \ x ∈ \text{set-of} t \] ⨸ \text{var} x ⨸ \text{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 (case-tac t2)
apply fastforce
apply fastforce
done

**Lemma:** dag-setofD: \( ∀ \ p \ \text{low} \ \text{high} \ n. \ [ \text{Dag} p \ \text{low} \ \text{high} \ t; \ n ∈ \text{set-of} t \] ⨸ (∃ nt. \text{Dag} n \ \text{low} \ \text{high} \ nt ⨸ (\forall nt. \text{Dag} n \ \text{low} \ \text{high} \ nt ⨸ \text{set-of} nt ⊆ \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: \[Dag p low high t; n \in set-of t\] \implies \exists nt. Dag n low high nt
apply (simp add: dag-setofD)
done

lemma dag-setof-subsetD: \[Dag p low high t; n \in set-of t; Dag n low high nt\] \implies set-of nt \subseteq 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: \(\prod p\) not no.
\[Dag p low high t; Dag no low high not; no \in set-of t\] \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-cq-def)
next
  assume no ≠ p
  with Node.prems have no-in-ltorrt: no ∈ set-of lt ∨ no ∈ set-of rt
  by simp
  show ?thesis
  proof (cases no ∈ 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 ∉ set-of lt
  with no-in-ltorrt have no-in-rt: no ∈ 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: \[ \text{ordered (Node } \text{lt } \text{p } \text{rt} \text{)} \text{ var} \] \implies \text{ordered } \text{lt} \text{ var } \land \text{ ordered } \text{rt} \text{ var}

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

lemma ordered-subdag: \[ \text{ordered } \text{t} \text{ var; not } <= \text{ t} \] \implies \text{ordered } \text{not} \text{ var}

proof (induct \text{t})
  case \text{Tip}
  then show ?thesis by (simp add: less-dag-def le-dag-def)
next
  case (Node \text{lt } \text{p } \text{rt})
  show ?case
  proof (cases \text{not = Node } \text{lt } \text{p } \text{rt})
    case \text{True}
    with \text{Node.prems} show ?thesis by simp
  next
    assume \text{notnt: not } \neq \text{ Node } \text{lt } \text{p } \text{rt}
    with \text{Node.prems} have \text{notstltorrt: not } <= \text{ lt } \lor \text{ not } <= \text{ rt}
      apply
      apply (simp add: less-dag-def le-dag-def)
      apply fastforce
      19
done
from Node.prems have ord-lt: ordered lt var
  apply –
  apply (drule children-ordered)
  apply simp
  done
from Node.prems have ord-rt: ordered rt var
  apply –
  apply (drule children-ordered)
  apply simp
  done
show ?thesis
proof (cases not <= lt)
  case True
  with ord-lt show ?thesis
  apply –
  apply (rule Node.hyps)
  apply assumption+
  done
next
  assume ¬ not ≤ lt
  with notstltorrt have notinrt: not <= rt
  by simp
  from Node.hyps have hyprt: ordered rt var; not ≤ rt] ⇒ ordered not var
  by simp
  from notinrt ord-rt show ?thesis
  apply –
  apply (rule hyprt)
  apply assumption+
  done
  qed
qed
qed

lemma subdag-ordered:
\( \forall \text{not no } p. \left[ \text{ordered } t \text{ var; Dag } p \text{ low high } t; \text{Dag } no \text{ low high } not; \right. \no \in \text{set-of } t] \implies \text{ordered not var} \)
proof (induct t)
  case Tip
  from Tip.prems show ?case by simp
next
  case (Node lt po rt)
  note nN = this
  show ?case
  proof (cases lt)
    case Tip
    note ltTip = this
    show ?thesis
proof (cases rt)
case Tip
  from Node.prems have ppo: p=po
    by simp
  from Tip ltTip Node.prems have no=p
    by simp
  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 (Node lrnot rn rrnot)
  from Node.prems ltTip Node have ord-rt: ordered rt var
    by simp
  from Node.prems have ppo: p=po
    by simp
  from Node.prems have ponN: po ≠ Null
    by auto
  with ppo ponN ltTip Node.prems have *: Dag (high po) low high rt
    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 ltTip have no ∈ set-of rt
    by simp
  with ord-rt * (Dag no low high not) show ?thesis
    by (rule Node.hyps)
qed
qed
next
case (Node llt l rlt)
  note ltNode=this
  show ?thesis
proof (cases rt)
case Tip
  from Node.prems Tip ltNode have ord-lt: ordered lt var
    by simp
  from Node.prems have ppo: p=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 lt Node have ord-lt: ordered lt var
    by simp
  from Node.prems Node lt Node have ord-rt: ordered rt var
    by simp
  from Node.prems have ponN: po \neq Null
    by auto
    with ppo ponN Node Node.prems lt Node have lt-Dag: Dag (low po) low high
      by auto
    with ppo ponN Node Node.prems lt Node have rt-Dag: Dag (high po) low high
      by auto
  show ?thesis
proof (cases no \in set-of lt)
  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
  assume no \neq po
  with Node.prems have no-in-ll-or-rt: no \in set-of lt \vee no \in set-of rt
    by simp
  show ?thesis
proof (cases no \in 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 \notin set-of lt
  with no-in-ll-or-rt have no-in-rt: no \in set-of rt
by simp
from Node.hyps have hyp2:
  \( \forall p \) no not. [ordered rt var; Dag p low high rt; Dag no low high not; no
  \( \in \) set-of rt]
  \( \Rightarrow \) 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

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

definition wf-ll :: dag \( \Rightarrow \) ref list list \( \Rightarrow \) (ref \( \Rightarrow \) nat) \( \Rightarrow \) bool
where
wf-ll t levellist var =
  \( (\forall p. \) p \( \in \) set-of t \( \Rightarrow \) (p \( \in \) set (levellist ! var p)) \&
    (\forall k < \) length levellist. \( \forall p \) \( \in \) set (levellist ! k). p \( \in \) set-of t \& var p = k)\)

definition cong-eval :: bdt \( \Rightarrow \) bdt \( \Rightarrow \) bool (infix \( \sim \) 60)
where cong-eval bdt_1 bdt_2 = (eval bdt_1 = eval bdt_2)

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

lemma cong-eval-trans: [\( l \sim r ; r \sim t \)] \( \Rightarrow \) \( l \sim t \)
  by (simp add: cong-eval-def)

lemma cong-eval-child-high: \( l \sim r \Rightarrow r \sim (Bdt-Node l v r) \)
  apply (simp add: cong-eval-def )
  apply (rule ext)
  apply auto
  done

lemma cong-eval-child-low: \( l \sim r \Rightarrow l \sim (Bdt-Node l v r) \)
  apply (simp add: cong-eval-def )
  apply (rule ext)

apply auto
done

definition null-comp :: (ref ⇒ ref) ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref) (infix α 60)
where null-comp a b = (λ p. (if (b p) = Null then Null else ((a ∘ b) p)))

lemma null-comp-not-Null [simp]: h q ≠ Null ⇒ (g ∝ h) q = g (h q)
by (simp add: null-comp-def)

lemma id-trans: (a ∝ id) (b (c p)) = (a ∝ b) (c p)
by (simp add: null-comp-def)

definition Nodes :: nat ⇒ ref list list ⇒ ref set
where Nodes i levellist = (∪ k ∈ {k. k < i} . set (levellist ! k))

inductive-set Dags :: ref set ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref) ⇒ dag set
for nodes low high
where
DagsI: [ set-of t ⊆ nodes; Dag p low high t; t ≠ Tip] 
⇒ t ∈ 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 rule
done

definition isLeaf-pt :: ref ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref) ⇒ bool
where isLeaf-pt p low high = (low p = Null ∧ high p = Null)

definition repNodes-eq :: ref ⇒ ref ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref) ⇒ (ref ⇒ ref)
⇒ bool
where
repNodes-eq p q low high rep =
( (rep ∝ high) p = (rep ∝ high) q ∧ (rep ∝ low) p = (rep ∝ low) q

definition isomorphic-dags-eq :: dag ⇒ dag ⇒ (ref ⇒ nat) ⇒ bool
where
isomorphic-dags-eq st1 st2 var =
( ∨ bdt1 bdt2. (bdt st1 var = Some bdt1 ∧ bdt st2 var = Some bdt2 ∧ (bdt1 = bdt2)))

24
\[ \rightarrow st_1 = st_2 \]

**lemma** isomorphic-dags-eq-sym: isomorphic-dags-eq \( st_1 \) \( st_2 \) \( \forall \) \( var \) = isomorphic-dags-eq \( st_2 \) \( st_1 \) \( \forall \) \( var \)

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

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

where

\[
\begin{align*}
\text{shared} \ t \ \text{var} &= (\forall \ st_1 \ st_2. \ (st_1 < t \land st_2 <= t) \rightarrow \text{isomorphic-dags-eq} \ st_1 \ st_2 \ \text{var})
\end{align*}
\]

**fun** reduced :: dag \( \Rightarrow \) bool where

\[
\begin{align*}
\text{reduced} \ \text{Tip} &= True \\
| \ \text{reduced} \ \text{(Node} \ \text{Tip} \ v \ \text{Tip}) &= True \\
| \ \text{reduced} \ \text{(Node} \ l \ v \ r) &= (l \neq r \land \text{reduced} \ l \land \text{reduced} \ r)
\end{align*}
\]

**primrec** reduced-bdt :: bdt \( \Rightarrow \) bool where

\[
\begin{align*}
\text{reduced-bdt} \ \text{Zero} &= True \\
| \ \text{reduced-bdt} \ \text{One} &= True \\
| \ \text{reduced-bdt} \ \text{(Bdt-Node} \ \text{lbdt} \ v \ \text{rbdt}) &= \\
& \quad \text{(if} \ \text{lbdt} = \ \text{rbdt} \ \text{then} \ \text{False}} \\
& \quad \quad \text{else} \ (\text{reduced-bdt} \ \text{lbdt} \land \text{reduced-bdt} \ \text{rbdt}))
\end{align*}
\]

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

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

**done**

**lemma** no-in-one-ll:

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

apply (unfold \( \text{wf-ll-def} \))
apply (erule conjE)
apply (rotate-tac 5)
apply (frule-tac \( x = i \) \( \text{and} \ ?R= \text{no} \in \text{set-of pret} \land \text{var} \text{no} = i \ \text{in} \ \text{allE} \))
apply (erule \( \text{impE} \))
apply simp
apply (rotate-tac 6)
apply (erule-tac \( x=\text{no in ballE} \))

25
apply assumption
apply simp
apply (cases no / set (levellista ! j))
apply assumption
apply (erule-tac x=j in allE)
apply (erule impE)
apply assumption
apply (rotate-tac 7)
apply (erule-tac x=no in ballE)
prefer 2
apply assumption
apply (elim conjE)
apply (thin-tac \( \forall q. q \in \text{set-of} \text{pret} \rightarrow q \in \text{set (levellista} \text{ ! var q)})
apply fastforce
done

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

lemma subelem-set-of-low:
\[
\begin{align*}
\bigwedge \text{p.} \begin{cases}
\text{x } \in \text{ set-of } t; \text{ x } \neq \text{ Null; low } x \neq \text{ Null; Dag p low high t}
\end{cases}
\implies \text{ low } x \in \text{ set-of } t
\end{align*}
\]
proof (induct t)
case \text{ Tip}
then show \text{ ?case by simp}
next
case (Node lt po rt)
note \text{ tNode= this}
then have \text{ppo: p=po by simp}
show \text{ ?case}
proof (cases x=p)
case True
with \text{Node.prems have txrootlt: low } x = \text{ root lt}
proof (cases lt)
case Tip
with True \text{Node.prems show ?thesis by auto}
next
case (Node llt l rlt)
with Node.prems True show ?thesis by auto
qed
with True Node.prems have \text{low } x \in \text{ set-of } (\text{Node lt p rt})
proof (cases lt)
case Tip
with \text{txrootlt Node.prems show ?thesis}
by simp

next

  case (Node llt l rlt)
    with lxrootlt Node.prems show ?thesis
      by simp

  qed

  with ppo show ?thesis
    by simp

next

  assume xnp: \( x \neq p \)
  with Node.prems have \( x \in \text{set-of } lt \vee x \in \text{set-of } rt \)
    by simp

  show ?thesis
  proof (cases \( x \in \text{set-of } lt \))
    case True
      note xincl=th \this
      from Node.prems have Dag (low p) low high lt
        by fastforce

      with Node.prems True have low x \in \text{set-of } lt
        apply 
          apply (rule Node.hyps)
          apply assumption+
        done

      with Node.prems show ?thesis
        by auto

  next

    assume xnotincl: \( x \notin \text{set-of } lt \)
    with xnp Node.prems have xincl: \( x \in \text{set-of } rt \)
      by simp

    from Node.prems have Dag (high p) low high rt
      by fastforce

    with Node.prems xincl have low x \in \text{set-of } rt
      apply 
        apply (rule Node.hyps)
        apply assumption+
      done

    with Node.prems show ?thesis
      by auto

  qed

  qed

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

proof (induct \( t \))

  case Tip

  then show ?case by simp

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

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

lemma Nodes-levellist-subset-t:
  [wf-ll t levellist var; i<= length levellist] ⇒ Nodes i levellist ⊆ set-of t
proof (induct i)
  case 0
  show ?case by (simp add: Nodes-def)
next
  case (Suc n)
  from Suc.prems Suc.hyps have Nodesn-in-t: Nodes n levellist ⊆ set-of t
    by simp
  from Suc.prems have ∀ x ∈ set (levellist ! n), x ∈ set-of t
    apply −
    apply (rule ballI)
    apply (simp add: wf-ll-def)
    apply (erule conjE)
    apply (thin-tac ∀ q. q ∈ set-of t −→ q ∈ 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) ⊆ set-of t
  apply blast
  done
with Suc.prems Nodesn-in-t show ?case
  apply (simp add: Nodes-def)
  apply (simp add: set-split)
  done
qed

lemma bdt-child:
  [ bdt (Node (Node llt l rlt) p (Node lrt r rrt)) var = Some bdt1]
\[ \exists \ bdt \ rbdt. \ bdt (\text{Node} \ llt \ l rt) \var = \text{Some} \ lbd \ \land \ bdt (\text{Node} \ lrt \ r rrt) \var = \text{Some} \ rbdt \]

by \ (simp \ split: \ \text{option.splits})

**lemma** \(\text{subbdt-ex-dag-def:} \)

\(\forall \ bdt1 \ p. \ [\text{Dag} \ p \ \text{low} \ \text{high} \ t; \ bdt \ t \var = \text{Some} \ bdt1; \ \text{Dag} \ \text{no} \ \text{low} \ \text{high} \ \text{not}; \ \text{no} \in \ \text{set-of} \ t] \implies \exists \ bdt2. \ bdt \ \text{not} \var = \text{Some} \ bdt2\)

**proof** (induct \(t\))

\text{case} Tip

\text{then show} \ ?\text{case} \ by \ simp

next

\text{case} (\text{Node} \ \text{l}t \ \text{po} \ \text{r}t)

\text{note} \ p\text{Node}=\this

\text{with} \ \text{Node.prems} \ \text{have} \ \text{p-po:} \ p=\text{po} \ by \ simp

\text{show} \ ?\text{case}

\text{proof} (\text{cases} \ \text{no} = \ \text{po})

\text{case} True

\text{note} \ no-\text{eq-po}=\this

\text{from} \ p-\text{po} \ \text{Node.prems} \ \text{no-eq-po} \ \text{have} \ \text{not} = (\text{Node} \ \text{l}t \ \text{po} \ \text{r}t) \ by \ (simp \ add: \ \text{Dag-unique \ del:} \ \text{Dag-Ref})

\text{with} \ \text{Node.prems} \ \text{have} \ bdt \ \text{not} \var = \text{Some} \ bdt1 \ by \ (simp \ add; \ \text{le-dag-def})

\text{then show} \ ?\text{thesis} \ by \ simp

next

\text{assume} \ \text{no} \neq \ \text{po}

\text{with} \ \text{Node.prems} \ \text{have} \ \text{no-in-lt-or-rt:} \ \text{no} \in \ \text{set-of} \ \text{l}t \ \lor \ \text{no} \in \ \text{set-of} \ \text{r}t \ by \ simp

\text{show} \ ?\text{thesis}

\text{proof} (\text{cases} \ \text{no} \in \ \text{set-of} \ \text{l}t)

\text{case} True

\text{note} \ \text{no-in-lt}=\this

\text{from} \ \text{Node.prems} \ p-\text{po} \ \text{have} \ \text{lt-dag}: \ \text{Dag} (\text{low} \ \text{po}) \ \text{low} \ \text{high} \ \text{lt} \ by \ simp

\text{from} \ \text{Node.prems} \ \text{have} \ \text{lbd-def:} \ \exists \ lbd. \ bdt \ \text{lt} \var = \text{Some} \ lbd

\text{proof} (\text{cases} \ \text{l})

\text{case} Tip

\text{with} \ \text{Node.prems} \ \text{no-in-lt} \ \text{show} \ ?\text{thesis} \ by \ (simp \ add; \ \text{le-dag-def})

next

\text{case} (\text{Node} \ \text{l}lt \ \text{l} \ \text{r}lt)

\text{note} \ \text{lNode}=\this

\text{show} \ ?\text{thesis}

\text{proof} (\text{cases} \ \text{rt})

\text{case} Tip

\text{note} \ r\text{Node}=\this

\text{with} \ \text{lNode} \ \text{Node.prems} \ \text{show} \ ?\text{thesis} \ by \ simp

next

\text{case} (\text{Node} \ \text{l}rt \ \text{r} \ \text{rrt})

\text{note} \ r\text{Node}=\this

\text{with} \ \text{lNode} \ \text{Node.prems} \ \text{show} \ ?\text{thesis} \ by \ (simp \ split: \ \text{option.splits})

\text{qed}

\text{qed}
then obtain \( lbdt \) where \( bdt \ t \ var = Some \ lbdt \)

with \( \text{Node.prems lt-dag no-in-lt} \) show ?thesis
  apply –
  apply (rule \text{Node.hyps})
  apply assumption+
  done

next
  assume \( \text{no} \notin \text{set-of \( \text{lt} \)} \)
  with \( \text{no-in-lt-or-rt} \) have \( \text{no-in-rt} \): \( \text{no} \in \text{set-of \( \text{rt} \)} \) by simp
  from \( \text{Node.prems p-po} \) have \( \text{rt-dag} \): \( \text{Dag (high \( \text{po} \)} \) low high \( \text{rt} \) by simp
  from \( \text{Node.hyps} \) have \( \text{hyp2} \): \( \exists \ rbdt. \ [\text{Dag (high \( \text{po} \)} \) low high \( \text{rt} \); \( bdt \ rt \ var = Some \ rbdt \) \]
    \( \text{by simp} \)
  from \( \text{Node.prems} \) have \( \text{lbdt-def} \): \( \exists \ rbdt. \ bdt \ rt \ var = Some \ rbdt \)

proof (cases \( \text{rt} \))
  case \( \text{Tip} \)
    note \( \text{rt} \text{Tip} = \text{this} \)
    with \( \text{rNode Node.prems} \) show ?thesis by (simp add: le-dag-def)
  next
  case (\( \text{Node \( \text{lrt} \ l \ rrt} \))
    note \( \text{lNode = this} \)
    with \( \text{rNode Node.prems} \) show ?thesis by (simp split: option.splits)

qed

qed

then obtain \( rbdt \) where \( bdt \ rt \ var = Some \ rbdt \)

with \( \text{Node.prems rt-dag no-in-rt} \) show ?thesis
  apply –
  apply (rule \text{hyp2})
  apply assumption+
  done

qed

qed


lemma \( \text{subbdt-ex} \):
\[ \land \ bdt1. \ [\ (Node \ lst \ stp \ rst) \ <\ t ; \ bdt \ t \ var = Some \ bdt1 \] \]
\[ \Longrightarrow \exists \ bdt2. \ bdt \ (Node \ lst \ stp \ rst) \ var = Some \ bdt2 \]

proof (induct \( \text{t} \))
  case \( \text{Tip} \)
  then show ?case by (simp add: le-dag-def)
next
\[\begin{align*}
\text{case} & \ (\text{Node } lt \ p \ rt) \\
\text{note} & \ p\text{Node} = \text{this} \\
\text{show} & \ \text{?case} \\
\text{proof} & \ (\text{cases } \text{Node } \text{lst} \ \text{stp} \ \text{rst} = \text{Node } lt \ p \ rt) \\
\text{case} & \ True \\
\text{with} & \ \text{Node.prems} \ \text{show} \ \text{?thesis by simp} \\
\text{next} & \\
\text{assume} & \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} \neq \text{Node } lt \ p \ rt \\
\text{with} & \ \text{Node.prems} \ \text{have} \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} < \text{Node } lt \ p \ rt \ \text{apply} \ (\text{simp add: le-dag-def}) \ \text{apply auto done} \\
\text{then have} & \ \text{in-ltrt}: \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{lt} \lor \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{rt} \\
\text{by} & \ (\text{simp add: less-dag-Node}) \\
\text{show} & \ \text{?thesis} \\
\text{proof} & \ (\text{cases } \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{lt}) \\
\text{case} & \ True \\
\text{note} & \ \text{in-lt} = \text{this} \\
\text{from} & \ \text{Node.prems} \ \text{have} \ \exists \ bdt. \ bdt \ \text{lt} \ \text{var} = \text{Some } bdt \\
\text{proof} & \ (\text{cases } \text{lt}) \\
\text{case} & \ \text{Tip} \\
\text{with} & \ \text{Node.prems} \ \text{in-lt} \ \text{show} \ \text{?thesis by simp} \\
\text{next} & \\
\text{case} & \ \text{(Node } \text{lt} \ l \ rlt) \\
\text{note} & \ l\text{Node} = \text{this} \\
\text{show} & \ \text{?thesis} \\
\text{proof} & \ (\text{cases } \text{rt}) \\
\text{case} & \ \text{Tip} \\
\text{note} & \ r\text{Node} = \text{this} \\
\text{with} & \ l\text{Node} \ \text{Node.prems} \ \text{show} \ \text{?thesis by simp} \\
\text{next} & \\
\text{case} & \ \text{(Node } \text{rt} \ r \ rrt) \\
\text{note} & \ r\text{Node} = \text{this} \\
\text{with} & \ l\text{Node} \ \text{Node.prems} \ \text{show} \ \text{?thesis by simp split: option.splits} \\
\text{qed} \\
\text{qed} \\
\text{then obtain} & \ \exists \ bdt \ \text{where} \ bdt \ \text{lt} \ \text{var} = \text{Some } bdt. \\
\text{with} & \ \text{Node.prems} \ \text{in-lt} \ \text{show} \ \text{?thesis} \\
\text{apply} & \ − \\
\text{apply} & \ (\text{rule Node.hyps}) \\
\text{apply} & \ \text{assumption+} \\
\text{done} \\
\text{next} & \\
\text{assume} & \ \neg \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{lt} \\
\text{with} & \ \text{in-ltrt} \ \text{have} \ \text{in-rt}: \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{rt} \ \text{by simp} \\
\text{from} & \ \text{Node.hyps} \ \text{have} \ \text{hyp2}: \ \exists \ \text{rbdt}. \ \text{Node } \text{lst} \ \text{stp} \ \text{rst} \leq \text{rt}; \ bdt \ \text{rt} \ \text{var} = \text{Some } \text{rbdt} \\
\text{by} & \ \text{simp} \\
\text{from} & \ \text{Node.prems} \ \text{have} \ \text{rbdt-def}: \ \exists \ \text{rbdt}. \ bdt \ \text{rt} \ \text{var} = \text{Some } \text{rbdt} \\
\text{proof} & \ (\text{cases } \text{rt}) \\
\text{case} & \ \text{Tip} \\
\end{align*}\]
with Node.prems in-rt show ?thesis by (simp add: le-dag-def)

next
  case (Node lrt l rrt)
  note rNode = this
  show ?thesis
  proof (cases lt)
    case Tip
    note lNode = this
    with rNode Node.prems show ?thesis by simp
  next
  case (Node lrt r rrt)
  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 in-rt show ?thesis
  apply
  apply (rule hyp2)
  apply assumption+
  done
  qed
  qed

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

34
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
nbsll: nb < length levellistb and
repbc-nc: ∀ nt. nt /∈ set (levellistb ! nb) → repb nt = repc nt and
xsnb-in-pret: ∀ 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 highN)
  done
with xsnb have lowxsnb: var (low x) < nb
  by arith
from children-var-smaller xsnb have highxsnb: var (high x) < nb

qed
qed
qed
by arith
from x-in-pret xnN True pret-dag have lowxpret: (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 highxpret: (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 ⊢ low) x = Null
  by (simp add: null-comp-def)
from lowxNull have repblowNull: (repb ⊢ low) x = Null
  by (simp add: null-comp-def)
with `repclowNull` have `louzrepbc`: $(\text{repc} \propto \text{low}) \cdot x = (\text{repb} \propto \text{low}) \cdot x$
  by simp

from `highxNull` have `repchighNull`: $(\text{repc} \propto \text{high}) \cdot x = \text{Null}$
  by (simp add: `null-comp-def`)

from `highxNull` have $(\text{repb} \propto \text{high}) \cdot x = \text{Null}$
  by (simp add: `null-comp-def`)

with `repchighNull` have `highxrepbc`: $(\text{repc} \propto \text{high}) \cdot x = (\text{repb} \propto \text{high}) \cdot x$
  by simp

with `louzrepbc` show `thesis`
  by simp

qed

lemma `uf-ll-Nodes-pret`:
$\begin{array}{c}
\{ \text{wf-ll pret levellista var; nb < length levellista; } x \in \text{Nodes nb levellista} \} \\
\implies x \in \text{set-of pret} \land \text{var} x < \text{nb}
\end{array}$
apply (simp add: `wf-ll-def Nodes-def`)
apply (erule conjE)
apply (thin-tac `\forall` q. q \in \text{set-of pret} \rightarrow q \in \text{set (levellista \char`\{\text{\textbackslash!} var q\})}
apply (erule `exE`)
apply (elim conjE)
apply (erule-tac `x` = k in `allE`)
apply (erule `impE`)
apply arith
apply auto
done

lemma `bdt-Some-var1-One`:
$\begin{array}{c}
\{ \text{bdt t var = Some x; var (root t) = 1} \} \\
\implies x = \text{One} \land t = (\text{Node Tip (root t) Tip})
\end{array}$
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 lTip 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


lemma bdt-Some-var0-Zero:
\[
\forall x. \left[ \text{bdt} \; \text{t} \; \text{var} = \text{Some} \; x; \; \text{var} \; (\text{root} \; \text{t}) = 0 \right] \implies x = \text{Zero} \land \text{t} = (\text{Node} \; \text{Tip} \; (\text{root} \; \text{t}) \; \text{Tip})
\]

proof (induct t)

  case Tip
  then show ?case by simp

next

  case (Node llt p rlt)
  note tNode = this
  show ?case
  proof (cases lt)
    case Tip
    note lTip = this
    show ?thesis
    proof (cases rt)
      case Tip
      note rtTip = this
      with lTip Node.prems show ?thesis by auto
    next
    case (Node lrt r rrt)
    note rtNode = this
    with Node.prems lTip 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 reduced-children-parent:
[ reduced l; l=(Node llt lp llt); reduced r; r=(Node lrt rp rrt);
lp ≠ rp ]
⇒ reduced (Node l p r)
by simp

lemma Nodes-subset: Nodes i levellista ⊆ Nodes (Suc i) levellista
apply (simp add: Nodes-def)
apply (simp add: set-split)
done

lemma Nodes-levellist:
[ wf-ll pret levellista var; nb < length levellista; p ∈ Nodes nb levellista]
⇒ p ≠ set (levellista \ nb)
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:
[ wf-ll pret levellista var; nb < length levellista; p ∈ Nodes nb levellista]
⇒ var p < nb ∧ p ∈ set-of pret
apply (simp add: Nodes-def wf-ll-def)
apply (erule conjE)
apply (thin-tac \ q. q ∈ set-of pret →→ q ∈ set (levellista \ var q))
apply (erule exE)
apply (erule-tac \ x. 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-DagsSucnb: t ∈ Dags (Nodes (Suc nb) levellista) low high
shows ∃ p . Dag p low high t ∧ p ∈ Nodes (Suc nb) levellista
proof –
  from t-in-DagsSucnb obtain p where t-day: Dag p low high t and t-subset-Nodes:
  set-of t ⊆ Nodes (Suc nb) levellista and t-nTip: t≠ Tip
  by (fastforce elim: Dags.cases)
from t-dag t-nTip have p≠Null by (cases t) auto
with t-subset-Nodes t-dag have p ∈ Nodes (Suc nb) levellista
  by (cases t) auto
with t-dag show ?thesis
  by auto
qed

lemma subdag-dag:
\( \forall p. \forall [Dag p low high t; st \leq t] \implies \exists stp. Dag stp low high st \)
proof (induct t)
case Tip
  then show ?case
    by (simp add: le-dag-def less-dag-def)
next
case (Node lt po rt)
  note t-Node=this
  with Node.prems have p-po: p=po
    by simp
  show ?case
    proof (cases st = Node lt po rt)
      case True
        note st-t=this
        with Node.prems show ?thesis
          by auto
    next
      assume st-nt: st ≠ Node lt po rt
      with Node.prems p-po have st-subdag-lt-rt: st\leq lt ∨ st \leq rt
        by (auto simp add: le-dag-def less-dag-def)
      from Node.prems p-po obtain lp rp where lt-dag: Dag lp low high lt and
                        rt-dag: Dag rp low high rt
        by auto
      show ?thesis
        proof (cases st\leq lt)
          case True
            note st-lt=this
            with lt-dag show ?thesis
              apply
                apply (rule Node.hyps)
                apply auto
              done
          next
            assume ¬ st \leq lt
            with st-subdag-lt-rt have st-rt: st \leq rt
              by simp
            from Node.hyps have rhyp: [Dag rp low high rt; st \leq rt] \implies \exists stp. Dag stp low high st
        qed
    qed
by simp
definition simp from st-rt rt-dag show \( \asthesis \)
apply 
apply (rule rhyp)
apply auto
done
qed
qed
qed

lemma Dags-subdags:
assumes in-Dags: \( t \in \text{Dags nodes low high} \) and
st-t: \( \text{st} \leq t \) and
st-nTip: \( \text{st} \neq \text{Tip} \)
shows \( \text{st} \in \text{Dags nodes low high} \)
proof
from in-Dags obtain \( p \) where t-dag: \( \text{Dag} p \text{ low high } t \) and t-subset-Nodes: set-of \( t \subseteq \text{nodes} \) and t-nTip: \( t \neq \text{Tip} \)
by (fastforce elim: Dags.cases)
from st-t have set-of \( \text{st} \subseteq \text{set-of } t \)
by (simp add: le-dag-set-of)
with t-subset-Nodes have st-subset-fnctNodes: set-of \( \text{st} \subseteq \text{nodes} \)
by blast
from st-t t-dag obtain \( s \) where Dag \( \text{stp} \) low high st
apply 
apply (drule subdag-dag)
apply auto
done
with st-subset-fnctNodes st-nTip show \( \asthesis \)
apply 
apply (rule DagsI)
apply auto
done
qed

lemma Dags-Nodes-cases:
assumes P-sym: \( \bigwedge t1 t2. P t1 t2 \text{ var} = P t2 t1 \text{ var} \) and
dags-in-lower-levels:
\( \bigwedge t1 t2. [t1 \in \text{Dags } (\text{fnct} \text{ '(Nodes n levellista)) low high};
\text{t2 } \in \text{Dags } (\text{fnct} \text{ '(Nodes n levellista)) low high}] \)
\( \implies P t1 t2 \text{ var} \) and
dags-in-mixed-levels:
\( \bigwedge t1 t2. [t1 \in \text{Dags } (\text{fnct} \text{ '(Nodes n levellista)) low high};
\text{t2 } \in \text{Dags } (\text{fnct} \text{ '(Nodes Suc n levellista)) low high};
\text{t2 } \notin \text{Dags } (\text{fnct} \text{ '(Nodes n levellista)) low high}] \)
\( \implies P t1 t2 \text{ var} \) and
dags-in-high-level:
\( \bigwedge t1 t2. [t1 \in \text{Dags } (\text{fnct} \text{ '(Nodes Suc n levellista)) low high}] \)
shows $\forall t_1 t_2. \; t_1 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} (\text{Suc} n) \text{ levellista})) \text{ low high} \land t_2 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} (\text{Suc} n) \text{ levellista})) \text{ low high} \implies P t_1 t_2 \; \text{var}$

proof (intro allI impI , elim conjE)
fix $t_1 t_2$
assume $t_1$-in-higher-levels: $t_1 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} (\text{Suc} n) \text{ levellista})) \text{ low high}$
assume $t_2$-in-higher-levels: $t_2 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} (\text{Suc} n) \text{ levellista})) \text{ low high}$
show $P t_1 t_2 \; \text{var}$
proof (cases $t_1 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$)
case True
note $t_1$-in-ll = this
show $? \text{thesis}$
proof (cases $t_2 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$)
case True
note $t_2$-in-ll = this
with $t_1$-in-ll dags-in-lower-levels show $? \text{thesis}$
by simp
next
assume $t_2$-notin-ll: $t_2 \notin \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$
with $t_1$-in-ll $t_2$-in-higher-levels dags-in-mixed-levels show $? \text{thesis}$
by simp
qed
next
assume $t_1$-notin-ll: $t_1 \notin \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$
show $? \text{thesis}$
proof (cases $t_2 \in \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$)
case True
note $t_2$-in-ll = this
with dags-in-mixed-levels $t_1$-in-higher-levels $t_1$-notin-ll P-sym show $? \text{thesis}$
by auto
next
assume $t_2$-notin-ll: $t_2 \notin \text{Dags} (\text{fnct}{'} (\text{Nodes} n \text{ levellista})) \text{ low high}$
with $t_1$-notin-ll $t_1$-in-higher-levels $t_2$-in-higher-levels dags-in-high-level show $? \text{thesis}$
by simp
qed
qed

lemma Null-notin-Nodes: $Dag p \; \text{low high} t; \; \text{nb} \leq \text{length levellista}; \; \text{wf-ll} t \text{ level-lista} \; \text{var} \implies \text{Null} \notin \text{Nodes} \; \text{nb} \text{ levellista}$
apply (simp add: Nodes-def wf-ll-def del: Dag-Ref)
apply (rule allI)
apply (rule impI)
apply (elim conjE)

42
apply (thin-tac ∀ q. P q for P)
apply (erule-tac x=k in allE)
apply (erule impE)
simp
apply (erule-tac x=Null in ballE)
apply (erule conjE)
apply (drule set-of-nn [rule-format])
apply auto
done

lemma Nodes-in-pret: \[ \text[wf-ll t levellista var; nb} \leq length levellista] \Rightarrow \text{Nodes nb levellista} \subseteq \text{set-of t} \]
apply –
apply rule
apply (simp add: wf-ll-def Nodes-def)
apply (erule exE)
apply (elim conjE)
apply (thin-tac ∀ q. q ∈ set-of t \rightarrow q ∈ set (levellista ! var q))
apply (erule-tac x=k in allE)
apply (erule impE)
simp
apply (erule-tac x=x in ballE)
apply auto
done

lemma restrict-root-Node:
\[ t \in \text{Dags (repc 'Nodes (Suc nb) levellista) (repc} \propto \text{low) (repc} \propto \text{high); t} \notin \text{Dags (repc 'Nodes nb levellista) (repc} \propto \text{low) (repc} \propto \text{high);} ordered t var; \forall \text{ no} \in \text{Nodes (Suc nb) leviellista. var (repc no)} \leq \text{var no} \land \text{repc (repc no) = repc no; wf-ll pret levellista var; nb} \leq \text{length levellista}; \text{repc 'Nodes (Suc nb) levellista} \subseteq \text{Nodes (Suc nb) levellista]} \]
\Rightarrow \exists q. \text{Dag (repc q) (repc} \propto \text{low) (repc} \propto \text{high) t} \land q \in \text{set (levellista} ! nb)\]
proof (elim Dags.cases)
fix p and ta :: dag
assume t-notin-DagsNodesnb: t \notin \text{Dags (repc} \propto \text{Nodes nb levellista) (repc} \propto \text{low) (repc} \propto \text{high)\)
assume t-ta: t = ta
assume ta-in-repc-NodesSucnb: set-of ta \subseteq \text{repc 'Nodes (Suc nb) levellista}
assume ta-dag: Dag p (repc} \propto \text{low) (repc} \propto \text{high) ta}
assume ta-nTip: ta \neq \text{Tip}
assume ord-t: ordered t var
assume varrep-prop: \forall \text{ no} \in \text{Nodes (Suc nb) levellista. var (repc no)} \leq \text{var no} \land \text{repc (repc no) = repc no\)
assume wf-lla: wf-ll pret levellista var
assume nbslla: nb \leq \text{length levellista}\)
assume repcNodes-in-Nodes: \text{repc 'Nodes (Suc nb) levellista} \subseteq \text{Nodes (Suc nb)}

43
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 ∈ set-of ta
by auto
with ta-in-repc-NodesSucnb have p-in-repcNodes-Sucnb: p ∈ repc ‘Nodes (Suc nb)’ levellista
by auto
show ?thesis
proof (cases p ∈ repc ‘(set (levellista ! nb)))
case True
then obtain q where
p-repc: p = repc q and
a-in-llanb: q ∈ set (levellista ! nb)
by auto
with ta-dag t-ta show ?thesis
apply –
apply (rule-tac x=q in exI)
apply simp
done
next
assume p-notin-repc-llanb: p /∈ repc ‘ set (levellista ! nb)
with p-in-repcNodes-Sucnb have p-in-repc-Nodesnb: p ∈ repc ‘Nodes nb levellista
apply –
apply (erule imageE)
apply rule
apply (simp add: Nodes-def)
apply (simp add: Nodes-def)
apply (erule exE conjE)
apply (case-tac k=nb)
apply simp
apply (rule-tac x=k in exI)
apply auto
done
have t ∈ Dags (repc ‘Nodes nb levellista) (repc ∝ low) (repc ∝ high)
proof –
have set-of t ⊆ repc ‘Nodes nb levellista
proof (rule)
fix x :: ref
assume x-in-t: x ∈ set-of t
with ord-t have var x <= var (root t)
apply –
apply (rule ordered-set-of)
apply auto
done
with t-ta ta-Node have varx-varp: var x <= var p

44
by auto

from p-in-repc-Nodesnb obtain k where ksnb: k < nb and p-in-repc-llak: p ∈ repc (set (levellista ! k))
  by (auto simp add: Nodes-def ImageE)
then obtain q where p-repq: p=reprp q and q-in-llak: q ∈ set (levellista ! k)
  by auto
from q-in-llak wf-lla nbslla have varqk: var q = k
  by (simp add: wf-ll-def)
have Nodese-nLodese-nodese-nSuccnb: Nodes nb levellista ⊆ Nodes (Suc nb) levellista
  by (rule Nodes-subset)
from q-in-lak ksnb have q ∈ Nodes nb levellista
  by (auto simp add: Nodes-def)
with varrep-prop Nodese-nLodese-nodese-nSuccnb have var (reprp q) † var q
  by auto
with varqk ksnb p-repq have var p < nb
  by auto
with varx-varp have varx-sn: var x < nb
  by auto
from x-in-t t-ta ta-in-repc-Nodese-nodese-nSuccnb obtain a where
  x-repca: x=reprp a and
  a-in-Nodese-nodese-nSuccnb: a ∈ Nodes (Suc nb) levellista
  by auto
with varrep-prop have rx-x: reprp x = x
  by auto
have x ∈ set-of pret
proof –
  from wf-lila nbsilla 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-Nodese-nodese-nSuccnb repcNodese-nodese-nodese-nSuccnb show ?thesis
  by auto
qed
with wf-lila have x ∈ set (levellista ! (var x))
  by (auto simp add: wf-lila-def)
with varx-sn 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=reprp 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: \[ \text{[bdt \ (Node \ lt1 \ p1 \ rt1) \ var = \ Some \ bdt1; bdt \ (Node \ lt2 \ p2 \ rt2) \ var = \ Some \ bdt1]} \]
\[ \implies \text{var \ p1 = var \ p2} \]
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:
\[ \text{[Dag \ p \ low \ high \ t; bdt \ t \ var = \ Some \ x; isLeaf-pt \ p \ low \ high]} \]
\[ \implies \text{var \ p \ \leq \ 1} \]
proof (induct t)
  case Tip
  thus ?case by simp
next
  case (Node lt p rt)
  note tNode=this
  from Node.prems tNode show ?case
    apply (simp add: isLeaf-pt-def)
    apply (case_tac var \ p = 0)
    apply simp
    apply (case-tac var \ p = Suc 0)
  qed
apply simp
apply simp
done
qed

lemma subnode-dag-cons:
\( p. [\text{Dag} p \text{ low high } t; \text{no } \in \text{ set-of } t] \implies \exists \text{ not. Dag no low high not} \)

proof (induct \( t \))
  case Tip
  thus ?case by simp
next
case (Node lt q rt)
  with Node.prems have q-p: \( p = q \)
  by simp
  from Node.prems have lt-dag: \( \text{Dag} (\text{low } p) \text{ low high } \text{lt} \)
  by auto
  from Node.prems have rt-dag: \( \text{Dag} (\text{high } p) \text{ low high } \text{rt} \)
  by auto
  show ?case
  proof (cases no \in \text{ set-of } \text{lt})
    case True
    with Node.hyps lt-dag show ?thesis
    by simp
  next
  assume no-notin-lt: \( \text{no } \notin \text{ set-of } \text{lt} \)
  show ?thesis
  proof (cases \( \text{no}=p \))
    case True
    with Node.prems q-p show ?thesis
    by auto
  next
  assume no-neq-p: \( \text{no } \neq \text{p} \)
  with Node.prems no-notin-lt have no-in-rt: \( \text{no } \in \text{ set-of } \text{rt} \)
  by simp
  with rt-dag Node.hyps show ?thesis
  by auto
qed
qed
lemma nodes-in-taken-in-takeSucn: $\text{no} \in \text{set} \ (\text{take} \ n \ \text{nodelist}) \implies \text{no} \in \text{set} \ (\text{take} \ (\text{Suc} \ n) \ \text{nodelist})$

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

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

lemma repNodes-eq-ext-rep: $[ \text{low} \text{no} \neq \text{nodelist} \ n; \ \text{high} \text{no} \neq \text{nodelist} \ n; \ \text{low} \text{sn} \neq \text{nodelist} \ n; \ \text{high} \text{sn} \neq \text{nodelist} \ n ]$ $\implies \text{repNodes-eq sn no low high repa = repNodes-eq sn no low high (repa(nodelist ! n))}$
  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) auto

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

48
lemma hd-filter-in-list: \( \text{filter} P \; xs \neq [] \implies \text{hd} (\text{filter} P \; xs) \in \text{set} \; xs \)
by (induct \( xs \)) auto

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

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

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

lemma filter-hd-P-rep-indep:
\( \forall x. \; P x \; x; \forall a \; b. \; P \; x \; a \implies P \; a \; b \implies P \; x \; b; \; \text{filter} (P \; x) \; xs \neq [] \implies \text{hd} (\text{filter} (P \; (\text{hd} (\text{filter} (P \; x) \; xs))) \; xs) = \text{hd} (\text{filter} (P \; x) \; xs) \)
apply (induct \( xs \))
apply simp
apply (case-tac \( P \; x \; a \))
using [[simp-depth-limit=2]]
apply ( simp )
apply clarsimp
apply (fastforce dest: hd-filter-prop)
done

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

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

lemma hd-filter-take-more: \( \forall n \, m. [\text{filter } P (\text{take } n \, xs) \neq [] ; \, n \leq m] \implies hd (\text{filter } P (\text{take } n \, xs)) = hd (\text{filter } P (\text{take } m \, xs)) \)

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

procedures
Eval (p, varval \mid R) =
IF (\texttt{\textasciitilde}p \rightarrow \texttt{var} = 0) THEN \texttt{\textasciitilde}R := \texttt{False}
ELSE IF (\texttt{\textasciitilde}p \rightarrow \texttt{var} = 1) THEN \texttt{\textasciitilde}R := \texttt{True}
ELSE IF (\texttt{\textasciitilde}varval ! (\texttt{\textasciitilde}p \rightarrow \texttt{var})) THEN CALL Eval (\texttt{\textasciitilde}p \rightarrow \texttt{high}, \texttt{varval}, \texttt{\textasciitilde}R)
ELSE CALL Eval (\texttt{\textasciitilde}p \rightarrow \texttt{low}, \texttt{\textasciitilde}varval, \texttt{\textasciitilde}R)
FI
FI
FI
procedures
Levellist (p, m, levellist | levellist) =
IF (p \neq \texttt{Null}) THEN
IF (p \rightarrow \texttt{\textasciitilde}mark \neq \texttt{\textasciitilde}m) THEN
levellist ::= CALL Levellist (p \rightarrow \texttt{\textasciitilde}low, m, levellist);
levellist ::= CALL Levellist (p \rightarrow \texttt{\textasciitilde}high, m, levellist);
p \rightarrow \texttt{\textasciitilde}next ::= levellist ! (p \rightarrow \texttt{\textasciitilde}var);
p \rightarrow \texttt{\textasciitilde}mark ::= \texttt{\textasciitilde}m
FI
FI
procedures
ShareRep (nodeslist, p) =
IF (isLeaf-pt p \texttt{\textasciitilde}\texttt{\textasciitilde}low \texttt{\textasciitilde}\texttt{\textasciitilde}high)
THEN p \rightarrow \texttt{\textasciitilde}rep ::= nodeslist
ELSE
WHILE (nodeslist \neq \texttt{Null}) DO
IF (repNodes-eq nodeslist p \texttt{\textasciitilde}\texttt{\textasciitilde}low \texttt{\textasciitilde}\texttt{\textasciitilde}high rep)
THEN p \rightarrow \texttt{\textasciitilde}rep ::= nodeslist; nodeslist ::= \texttt{Null}
ELSE nodeslist ::= nodeslist \rightarrow \texttt{\textasciitilde}next
FI
OD
FI
procedures
ShareReduceRepList (nodeslist | ) =
node ::= \texttt{\textasciitilde}nodeslist;\
\texttt{\textasciitilde}node = \texttt{\textasciitilde}\texttt{\textasciitilde}isLeaf-pt \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}low \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}high \rightarrow
\texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}node \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}low \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}rep = \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}node \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}high \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}rep
THEN node \rightarrow \texttt{\textasciitilde}rep ::= node \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}\texttt{\textasciitilde}low \rightarrow \texttt{\textasciitilde}\texttt{\textasciitilde}rep
ELSE CALL ShareRep (nodeslist, node)
\[FI::
\]
\[\text{node} := '\text{node} \rightarrow '\text{next}\\
\]
\[OD\]

procedures

\[\text{Repoint}(p|p) =
\]
\[\text{IF } (p \neq \text{Null})
\]
\[\text{THEN}
\]
\[p := p \rightarrow 'rhop;
\]
\[\text{IF } (p \neq \text{Null})
\]
\[\text{THEN } p \rightarrow '\text{low} := \text{CALL Repoint}(p \rightarrow '\text{low});
\]
\[p \rightarrow '\text{high} := \text{CALL Repoint}(p \rightarrow '\text{high})\\
\]
\[FI\\
\]
\[FI\]

procedures

\[\text{Normalize}(p|p) =
\]
\[\text{levellist} := \text{replicate}(p \rightarrow '\text{var} + 1) \text{Null};
\]
\[\text{levellist} := \text{CALL Levellist}(p, (\neg p \rightarrow '\text{mark}) , \text{levellist});
\]
\[n := 0;
\]
\[\text{WHILE } (n < \text{length } \text{levellist}) \text{ DO}
\]
\[\text{CALL ShareReduceRepList}('\text{levellist} ! n);
\]
\[n := n + 1\\
\]
\[OD);\\
\]
\[p := \text{CALL Repoint}(p)\\
\]
end

5 Proof of Procedure Eval

theory EvalProof imports ProcedureSpecs begin

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

lemma (in Eval-impl) Eval-spec:
\[\text{shows } \forall \sigma t \text{ bdt1}. \Gamma \vdash
\]
\[\{\sigma. \text{Dag } p '\text{low} '\text{high} t \land \text{bdt } t '\text{var} = \text{Some bdt1}\}
\]
\[R := \text{PROC Eval}(p, '\text{varval})
\]
\[\{R = \text{eval bdt1 }\sigma'\text{varval}\}\]
apply (hoare-rule HoarePartial.ProcRec1)
apply vcg

52
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 (case-tac bdt1)
apply fastforce
apply fastforce
apply fastforce
apply (case-tac bdt1)
apply fastforce
apply fastforce
apply (case-tac bdt1)
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 ∀σ. Γσ {σ} 'levellist ::= PROC Levellist ('p, 'm, 'levellist)
  {t. t may-only-modify-globals σ in [mark,next]}
  apply (hoare-rule HoarePartial.ProcRec1)
  apply (vcg spec=modifies)
done

lemma all-stop-cong: (∀x. P x) = (∀x. P x)
  by simp

lemma Dag-RefD:
  [Dag p l r t; p≠Null] ⊢
  ∃lt rt, t=Node lt rt Dag (l p) l r lt ∧ Dag (r p) l r rt
  by simp

53
lemma *Dag-unique-ex-conjI*: \[ [\text{Dag } p \ l \ r \ t; \ P t] \implies (\exists t. \text{Dag } p \ l \ r \wedge P t) \]
by simp

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

lemma *list-ext*: \[ \forall ys. [\text{length } xs = \text{length } ys; \forall i < \text{length } xs. xs![i]=ys![i]] \implies xs=ys \]
apply (induct xs)
apply simp
apply (case-tac ys)
apply simp
apply force

done

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

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

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

lemma *Levellist-unique*:
assumes \ll: *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 \( \text{leq: } \text{length } ll = \text{length } ll' \) by simp
show ?thesis
proof (rule list-ext [OF leq, rule-format!])
fix \( i \)
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 \textit{Levellist-unique-ex-conj-simp} [simp]:
\begin{align*}
\text{Levellist } \text{hds } \text{next } \text{ll } & \Longrightarrow (\exists \text{ll}. \text{Levellist } \text{hds } \text{next } \text{ll } \land \ P \ \text{ll}) = P \ \text{ll} \\
\end{align*}
by (auto dest: \textit{Levellist-unique})

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

definition \textit{wf-levellist} :: \text{dag } \Rightarrow \text{ref list list } \Rightarrow \text{ref list list } \Rightarrow \text{nat } \Rightarrow \text{bool } where
\begin{align*}
\text{wf-levellist } \text{t } \text{levellist-old } \text{levellist-new } \text{var } &= \text{\begin{align*}
| (\text{Node } \text{lt } p \ \text{rt}) & \Rightarrow \\
(\forall \ q. \ q \in \text{set-of } t & \longrightarrow 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 \ pt \in \text{set } \text{prx}. \ pt \in \text{set-of } t & \land \ \text{var } pt = i)) \land \\
(\forall \ i. \ (\text{var } p) < i & \longrightarrow (\text{levellist-new } ! i) = (\text{levellist-old } ! i)) \land \\
(\text{length } \text{levellist-new } & = \text{length } \text{levellist-old})) \\
\end{align*}}

lemma \textit{wf-levellist-subset}:
\begin{align*}
\text{assumes } \text{wf-ll: } \text{wf-levellist } \text{t } \text{ll } \text{ll'} \ \text{var } & \Rightarrow \\
\text{shows } \text{set } (\text{concat } \text{ll'}) & \subseteq \text{set } (\text{concat } \text{ll}) \cup \text{set-of } t \\
\text{proof } (\text{cases } \text{t}) \\
\end{align*}
next
case \text{Tip } with \text{wf-ll } show ?thesis by (simp add: \textit{wf-levellist-def})

proof –
{
fix \ n
assume \ n \in \text{set } (\text{concat } \text{ll'})
from \textit{in-set-concat-idx} [OF this]
obtain \ i \ where \ \text{i-bound: } i < \text{length } \text{ll'} \ \text{and } n \in \text{set } (\text{ll'} ! i)
by blast
have \ n \in \text{set } (\text{concat } \text{ll}) \cup \text{set-of } t \\
proof (cases \ i \leq \text{var } p)

55
case True
with wf-ll obtain prx where
\(ll' = ll \setminus \{i\} = prx \cup ll \setminus \{i\}\) and
prx: \(\forall pt \in \text{set prx}, pt \in \text{set-of } t\) and
leq: \(\text{length } ll' = \text{length } ll\)
apply (clarsimp simp add: wf-levellist-def Node)
apply (erule_tac x = \(i\) in allE)
apply clarsimp
done

show \(?thesis\)
proof (cases \(n \in \text{set prx}\))
  case True
  with prx have \(n \in \text{set-of } t\)
  by simp
  thus \(?thesis\) by simp
next
  case False
  with \(n\)-in \(ll'\)
  have \(n \in \text{set } (ll \setminus \{i\})\)
  by simp
  with \(i\)-bound leq
  have \(n \in \text{set } (\text{concat } ll)\)
  by auto
  thus \(?thesis\) by simp
qed

next
  case False
  with wf-ll obtain \(ll' \setminus \{i\} = ll \setminus \{i\}\) length \(ll' = \text{length } ll\)
  by (auto simp add: wf-levellist-def Node)
  with \(n\)-in \(i\)-bound
  have \(n \in \text{set } (\text{concat } ll)\)
  by auto
  thus \(?thesis\) by simp
qed

thus \(?thesis\) by auto
qed

lemma Levellist-ext-to-all: \((\exists \(ll\). \text{Levellist hds next } ll \land P ll) \rightarrow Q) = \((\forall \(ll\). \text{Levellist hds next } ll \land P ll \rightarrow Q)\)
apply blast
done
lemma \textbf{Levellist-length}: \textbf{Levellist} \( \text{hds} \ p \ \text{ll} \rightarrow \text{length} \ \text{ll} = \text{length} \ \text{hds} \)

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

lemma \textbf{map-update}:
\( \forall i. \ i < \text{length} \ \text{xss} \rightarrow \text{map} \ f \ ((\text{xss}[i := \text{xs}])) = (\text{map} \ f \ \text{xss}) \ [i := f \ \text{xs}] \)

apply (induct \text{zss})

apply simp

apply (case-tac \ i)

apply simp

apply simp

done

lemma (in \text{Levellist-impl}) \text{Levellist-spec-total}:
\( \forall \text{ll} \ \text{t} \ \text{Γ}, \text{Θ} \vdash \text{t} \{ |\text{σ}. \ \text{Dag} \ \text{´p} \ \text{low} \ \text{´high} \ \text{t} \land (\text{´p} \neq \text{Null} \rightarrow (\text{´p} \rightarrow \text{´var}) < \text{length} \ \text{´levellist}) \land \ \text{ordered} \ \text{´var} \ \land \ \text{Levellist} \ \text{´levellist} \ \text{´next} \ \text{ll} \land \ (\forall \text{n} \in \text{set-of} \ \text{t}. \ (\text{if} \ \text{´mark} \ \text{n} = \text{´m} \ \text{then} \ \text{n} \in \text{set} \ ((\text{ll} \ \rightarrow \text{´var} \ \text{n}) \land \ (\forall \text{nt} \ \text{p}. \ \text{Dag} \ \text{n} \ \text{´low} \ \text{´high} \ \text{nt} \land \text{p} \in \text{set-of} \ \text{nt} \ \rightarrow \ \text{´mark} \ \text{p} = \text{´m}) \ \text{else} \ \text{n} \notin \text{set} \ ((\text{concat} \ \text{ll}))))\})

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

\( (\exists \text{ll}'). \ \text{Levellist} \ \text{´levellist} \ \text{´next} \ \text{ll} \land \ \text{wf-levellist} \ \text{t} \ \text{ll} \ \text{ll} \ \text{´var} \ \land \ \text{wf-marking} \ \text{t} \ \text{´mark} \ \text{´mark} \ \text{´mark} \ \text{´m} \ \land \ (\forall \text{p} \ \text{p} \notin \text{set-of} \ \text{t} \rightarrow \ \text{´next} \ \text{p} = \text{´next} \ \text{p}) \)

\)

apply (hoare-rule \text{HoareTotal.ProcRec1}

[where \text{r}=\text{measure} (\lambda(s,p). \text{size} (\text{dag} \ \text{´low} \ \text{´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 ifI)

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

defer

apply (clarsimp simp add: \text{wf-levellist-def} \text{wf-marking-def})

apply (simp only: \text{Levellist-ext-to-all})

proof

fix \text{ll} \ \text{var} \ \text{low} \ \text{high} \ \text{mark} \ \text{nexta} \ \text{p} \ \text{levellist} \ \text{m} \ \text{lt} \ \text{rt}

assume pnN: \text{p} \neq \text{Null}

assume mark-p: \text{mark} \ \text{p} = (\neg \text{m})

assume lt: \text{Dag} \ (\text{low} \ \text{p}) \ \text{low} \ \text{high} \ \text{lt}

assume rt: \text{Dag} \ (\text{high} \ \text{p}) \ \text{low} \ \text{high} \ \text{rt}

57
from \textit{pnN lt rt} have \textit{Dag-p}: Dag p low high (Node lt p rt) by simp

from \textit{Dag-p rt} have \textit{size-rt-dec}: size (dag (high p) low high) < size (dag p low high)
  by (simp only: Dag-dag) simp

from \textit{Dag-p lt} have \textit{size-lt-dec}: size (dag (low p) low high) < size (dag p low high)
  by (simp only: Dag-dag) simp

assume \textit{ll}: 

\textbf{Levellist levellist next ll}

assume \textit{marked-child-ll}:

\(\forall n \in \text{set-of (Node lt p rt)}.\)
  if mark \(n = m\)
    then \(n \in \text{set (ll ! var n)}\) \&
      \((\forall nt p. \text{Dag n low high nt} \& p \in \text{set-of nt} \rightarrow \text{mark p} = m)\)
  else \(n \notin \text{set (concat ll)}\)

with \textit{mark-p} have \textit{p-notin-ll}: \(p \notin \text{set (concat ll)}\)
  by auto

assume \textit{varsll}': \(\text{var p} < \text{length levellist}\)

with \textit{ll} have \textit{varsll}: \(\text{var p} < \text{length ll}\)
  by (simp add: \textit{Levellist-length})

assume \textit{orderedt}: ordered (Node lt p rt) var

show (low p \neq \text{Null} \rightarrow \text{var (low p) < length levellist}) \&

ordered lt var \&
  \((\forall n \in \text{set-of lt}.\)
    if mark \(n = m\)
      then \(n \in \text{set (ll ! var n)}\) \&
        \((\forall nt p. \text{Dag n low high nt} \& p \in \text{set-of nt} \rightarrow \text{mark p} = m)\)
      else \(n \notin \text{set (concat ll)}\)) \&

size (dag (low p) low high) < size (dag p low high) \&

\((\forall \text{marka neztax levellist lla}.\)
  
\textbf{Levellist levellist neztax lla} \&

\textit{wf-levellist lt ll lla var} \& \textit{wf-marking lt mark marka m} \&

\((\forall p. p \notin \text{set-of lt} \rightarrow \text{next p} = \text{nexta p})\)\rightarrow

(high p \neq \text{Null} \rightarrow \text{var (high p) < length levellist}) \&

ordered rt var \&

\((\exists lla. \text{Levellist levellist neztax lla})\)

\((\forall n \in \text{set-of rt}.\)
  if marka \(n = m\)
    then \(n \in \text{set (lla ! var n)}\) \&
      \((\forall nt p. \text{Dag n low high nt} \& p \in \text{set-of nt} \rightarrow \text{marka p} = m)\)
    else \(n \notin \text{set (concat lla)}\)) \&

size (dag (high p) low high) < size (dag p low high) \&

\((\forall \text{markb neztb levellist llb}.\)
  
\textbf{Levellist levellist neztb llb} \&

\textit{wf-levellist rt llb lla var} \& \textit{wf-marking rt marka markb m} \&

\((\forall p. p \notin \text{set-of rt} \rightarrow \text{nexta p} = \text{nextb p})\)\rightarrow

\((\exists ll'! \text{ Levellist (levellist[\text{var p := p}])})\)
(nextb(p := levellist ! var p)) ll’ ∧
wf-levellist (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 wfll-lt: wf-levellist Tip ll lla var
  wf-marking Tip mark marka m by fact+
then have ll-lla: ll = lla
  by (simp add: wf-levellist-def)
moreover
with wfll-lt lt-Tip lt have marka = mark
  by (simp add: wf-marking-def)
moreover
have wfll-rt: wf-levellist Tip lla llb var
  wf-marking Tip marka markb m by fact+
then have lla-llb: lla = llb
  by (simp add: wf-levellist-def)
moreover
with wfll-rt Tip rt have markb = marka
  by (simp add: wf-marking-def)
moreover
from varsll llb llla lla-llb
obtain var p < length levellistb var p < length llb
  by (simp add: Levellist-length)
with llb pnN
have llc: Levellist (levellistb[var p := p]) (nextb(p := levellistb ! var p))
  (llb[var p := p ≠ llb ! var p])
  apply (clarsimp simp add: Levellist-def map-update)
  apply (erule-tac x=i in allE)
  apply clarsimp
  apply (subgoal-tac p /∈ set (llb ! i )
prefer 2
using p-notin-ll ll-lla lla-llb

59
apply  simp
apply (case-tac i=var p)
apply  simp
apply  simp
done
ultimately
show  ?case
  using  lt-Tip  Tip  varsll
  apply (clarsimp simp add: wf-levellist-def wf-marking-def)
proof  -
  fix  i
  assume  varsllb:  var  p < length  llb
  assume  i  ≤  var  p
  show  ∃ prx.  llb[var  p :=  p#llb!var  p]!i  =  prx  @  llb!i  ∧
               (∀ pt∈set  prx.  pt  =  p  ∧  var  pt  =  i)
  proof  (cases  i  =  var  p)
    case  True
    with  pnN  lt  rt  varsllb  lt-Tip  Tip  show  ?thesis
      apply  -
      apply  (rule-tac  x=[p]  in  exI)
      apply  (simp  add:  subdag-eq-def)
      done
    next
    assume  i  ≠  var  p
    with  varsllb  show  ?thesis
      apply  -
      apply  (rule-tac  x=[]  in  exI)
      apply  (simp  add:  subdag-eq-def)
      done
    qed
    qed
  qed
next
  case  (Node  dag1  a  dag2)
  have  rt-node:  rt = Node  dag1  a  dag2  by  fact
  with  rt  have  high-p:  high  p  =  a
            by  simp
  have  s:  ⋀  nexta.  (∀  p.  next  p  =  nexta  p)  =  (next  =  nexta)
            by  auto
  show  ?thesis
    using  size-rt-dec  size-lt-dec  rt-node  lt-Tip  Tip  lt  rt
    apply (clarsimp simp del: set-of-Node  split  del:  split-if  simp  add:  s)
  proof  -
    case  (goal1  marka  levellista  lla)
    have  lla:  Levellist  levellista  next  lla  by  fact
    have  wfl-ltl: wf-levellist  Tip  ltl  var
              wf-marking  Tip  mark  marka  m  by  fact+
    from  this  have  ltl-lla:  ll = lla
              by  (simp  add:  wf-levellist-def)
moreover
from wfll-lt lt-Tip lt have marklrec: marka = mark
  by (simp add: wfl-marking-def)
from orderedt varsll lla ll-lia 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-lia
have mark-rt: (\(\forall n \in \text{set-of} \ (\text{Node dag1}) \ (\text{high p}) \text{ dag2}\).
  if marka n = m
  then n \in \text{set} (lla ! var n) \land
    (\(\forall nt \cdot \text{Dag n low high nt} \land p \in \text{set-of} nt \rightarrow \text{marka p} = m\)
  else n \notin \text{set} (concat lla))
apply (simp only: BinDag.set-of.simps)
apply clarify
apply (drule-tac x = n in bspec)
apply blast
apply assumption
done
show \?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 wflll-rt: wfl-levellist (Node dag1 (high p) dag2) lla llb var by fact
have wflmarking-rt: wfl-marking (Node dag1 (high p) dag2) marka markb
  m by fact
from wflll-rt varsll lla ll-lia
obtain var-p-bounds: var p < length levellistb var p < length llb
  by (simp add: Levellist-length wfl-levellist-def)
with p-notin-ll ll-lia wflll-rt
have p-notin-llb: \(\forall i < \text{length llb} \cdot p \notin \text{set} (\text{llb} ! i)\)
apply –
apply (intro allI impl)
apply (clarsimp simp add: wfl-levellist-def)
apply (case-tac i \leq \text{var} (\text{high p}))
apply (drule-tac x = i in spec)
using orderedt rt-node lt-Tip high-p
applyclarsimp
apply (drule-tac x = i in spec)
apply (drule-tac \(x = i\) in \(\text{spec}\))
apply clarsimp
done

with \(llb\) \(pnN\) \(\text{var-p-bounds}\)

have \(llc\):

\[
\begin{align*}
\text{Levellist} & \left(\text{levellistb}[\text{var } p := p]\right) \\
\text{nextb} & \left(p := \text{levellistb} ! \text{var } p\right) \\
\text{llb}[\text{var } p := p \neq \text{llb} ! \text{var } p]\end{align*}
\]

apply (clarsimp simp add: Levellist-def map-update)
apply (erule-tac \(x = i\) in \(\text{allE}\))
apply (erule-tac \(x = i\) in \(\text{allE}\))
apply clarsimp
apply (case-tac \(i = \text{var } p\))
apply simp
apply simp
done

then show ?case
apply simp
using \(\text{wfll-rt}\) \(\text{wfmarking-rt}\)

\[
\begin{align*}
\text{lt-Tip} & \text{rt-node} \text{varsl} \text{ordert} \text{lt} \text{rt} \text{pnN} \text{ll-lla} \text{marklrec}\end{align*}
\]
apply (clarsimp simp add: \(\text{wf-levellist-def} \ \text{wf-marking-def}\))
apply (intro conjI)
apply (rule allI)
apply (rule conjI)
apply (erule-tac \(x = q\) in \(\text{allE}\))
apply (case-tac \(\text{var } p = \text{var } q\))
apply fastforce
apply fastforce
apply (case-tac \(\text{var } p = \text{var } q\))
apply hypsubst-thin
apply fastforce
apply fastforce
apply (rule allI)
apply (rotate-tac 4)
apply (erule-tac \(x = i\) in \(\text{allE}\))
apply (case-tac \(i = \text{var } p\))
apply simp
apply (case-tac \(\text{var } (\text{high } p) < i\))
apply simp
apply simp
apply (erule \(\text{exE}\))
apply (rule-tac \(x = \text{prx}\) in \(\text{exI}\))
apply (intro conjI)
apply simp
apply clarify
apply (rotate-tac 15)
apply (erule-tac \(x = pt\) in \(\text{ballE}\))
apply fastforce
apply fastforce
done
qed

next
case (Node llt l rlt)

have llt-Node: l = Node llt l rlt by fact

from ordered lt varsll' lt-Node

obtain ordered lt:
ordered lt var (low p ≠ Null → var (low p) < length levellist)
by (cases rt) auto

from lt lt-Node marked-child-ll

have mark-lt: ∀ n∈set-of lt.
if mark n = m
then n ∈ set (ll ! var n) ∧
(∀ nt. Dag n low high nt ∧ p ∈ set-of nt → mark p = m)
else n /∈ set (concat ll)
apply (simp only: BinDag.set-of.simps)
apply clarify
apply (drule_tac x=n in bspec)
apply blast
apply assumption
done

show ?thesis
apply (intro conjI ordered lt mark-lt size-lt-dec)
apply (clarify)
apply (simp add: size-rt-dec split del: 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 wfll-lt: wf-levellist lt ll lla var by fact

have wflmarking-lt:wf-marking lt mark marka m by fact

from wfll-lt lt-Node

have lla-eq-ll: length lla = length ll
by (simp add: wf-levellist-def)

with l ll have lla-eq-ll': length levellista = length levellist
by (simp add: Levellist-length)

with ordered rt lt-Node lt varsll'

obtain ordered rt:
ordered rt var (high p ≠ Null → var (high p) < length levellista)
by (cases rt) auto

from wfll-lt lt-Node

have nodes-in-lla: ∀ q. q ∈ set-of lt → q ∈ set (lla ! (q→var))
by (simp add: wf-levellist-def)

from wfll-lt lt-Node lt

have lla-st: ∀ i ≤ (low p)→var.
(∃ prx. (lla ! i) = prx@(ll ! i) ∧
(∀ pt ∈ set prx. pt ∈ set-of lt ∧ pt→var = i)))

by (simp add: wf-levellist-def)
\textbf{From} \textit{wfll-lt lt-Node lt} \\
\textbf{Have} \textit{lla-nc:} \( \forall i. ((\text{low } p) \rightarrow \text{var} < i \rightarrow (\text{lla } ! i) = (\text{ll } ! i) \) \\
\textbf{By} (simp add: \textit{wf-levellist-def}) \\
\textbf{From} \textit{wfmarking-lt lt-Node lt} \\
\textbf{Have} \textit{mot-nc:} \( \forall n. n \notin \text{set-of lt} \rightarrow \text{mark } n = \text{marka } n \) \\
\textbf{By} (simp add: \textit{wf-marking-def}) \\
\textbf{From} \textit{wfmarking-lt lt-Node lt} \\
\textbf{Have} \textit{mit-marked:} \( \forall n. n \in \text{set-of lt} \rightarrow \text{marka } n = m \) \\
\textbf{By} (simp add: \textit{wf-marking-def}) \\
\textbf{From} \textit{marked-child-ll nodes-in-lla mot-nc mit-marked lla-st} \\
\textbf{Have} \textit{mark-rt:} \( \forall n \in \text{set-of rt} \) \\
\hspace{1em} \text{if marka } n = m \\
\hspace{2em} \text{then } n \in \text{set } (\text{lla } ! \text{ var } n) \land \\
\hspace{3em} \left( \forall nt p. \text{Dag } n \text{ low high } nt \land p \in \text{set-of } nt \rightarrow \text{marka } p = m \right) \\
\hspace{2em} \text{else } n \notin \text{set } (\text{concat lla}) \\
\textbf{Apply} \texttt{--} \\
\textbf{Apply} (\texttt{rule ballI}) \\
\textbf{Apply} (\texttt{drule-tac } x=n \texttt{ in bspec}) \\
\textbf{Apply} (simp) \\
\textbf{Proof} \texttt{--} \\
\textbf{Fix} \( n \) \\
\textbf{Assume} \textit{nodes-in-lla:} \( \forall q. q \in \text{set-of lt} \rightarrow q \in \text{set } (\text{lla } ! \text{ var } q) \) \\
\textbf{Assume} \textit{mot-nc:} \( \forall n. n \notin \text{set-of lt} \rightarrow \text{mark } n = \text{marka } n \) \\
\textbf{Assume} \textit{mit-marked:} \( \forall n. n \in \text{set-of lt} \rightarrow \text{marka } n = m \) \\
\textbf{Assume} \textit{marked-child-ll:} \( \text{if mark } n = m \) \\
\hspace{1em} \text{then } n \in \text{set } (\text{lt } ! \text{ var } n) \land \\
\hspace{2em} \left( \forall nt p. \text{Dag } n \text{ low high } nt \land p \in \text{set-of } nt \rightarrow \text{marka } p = m \right) \\
\hspace{2em} \text{else } n \notin \text{set } (\text{concat ll}) \\
\textbf{Assume} \textit{lla-st:} \( \forall i \leq \text{var } (\text{low } p) \). \\
\hspace{1em} \exists \text{prx}. \text{lla } ! i = \text{prx } @ \text{ll } ! i \land \\
\hspace{2em} \left( \forall pt \in \text{set } \text{prx}. \text{pt } \in \text{set-of lt } \land \text{var } \text{pt } = i \right) \\
\textbf{Assume} \textit{n-in-rt:} \( n \in \text{set-of rt} \) \\
\textbf{Show} \textit{n-in-lla-marked:} \( \text{if marka } n = m \) \\
\hspace{1em} \text{then } n \in \text{set } (\text{lla } ! \text{ var } n) \land \\
\hspace{2em} \left( \forall nt p. \text{Dag } n \text{ low high } nt \land p \in \text{set-of } nt \rightarrow \text{marka } p = m \right) \\
\hspace{2em} \text{else } n \notin \text{set } (\text{concat lla}) \\
\textbf{Proof} (\texttt{cases } n \in \text{set-of lt}) \\
\textbf{Case} \texttt{True} \\
\textbf{From} \textit{True nodes-in-lla have} \textit{n-in-ll:} \( n \in \text{set } (\text{lla } ! \text{ var } n) \) \\
\textbf{By} simp \\
\textbf{Moreover} \\
\textbf{From} \textit{True wfmarking-lt} \\
\textbf{Have} \textit{marka } n = m \\
\textbf{Apply} (\texttt{cases lt}) \\
\textbf{Apply} (\texttt{auto simp add: wf-marking-def}) \\
\textbf{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-l: 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-l 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
    with n-in-lla show ?thesis
      by fastforce
  next
  assume vars: ~ var (low p) < var n
  with lla-st
  have ll-in-lla: ∃ prx. lla ! (var n) = prx @ ll ! (var n)
apply (erule-tac x=\text{var} \ n \ \text{in} \ allE)
apply fastforce
done
with \text{n-in-ll} show \ ?\text{thesis}
by fastforce
qed

\{ 
  \begin{enumerate}
  \item \text{fix} \ nt \ \text{pt}
  \item assume \text{nt-Dag}: \text{Dag} \ n \ \text{low} \ \text{high} \ nt
  \item assume \text{pt-in-nt}: \text{pt} \ \in \ \text{set-of} \ nt
  \end{enumerate}
proof (cases \text{pt} \ \in \ \text{set-of} \ lt)
  \begin{enumerate}
  \item case True
    with \text{mit-marked} show \ ?\text{thesis}
    by fastforce
  \item case False
    \begin{enumerate}
    \item note \text{n-not-marka} = \text{this}
    \item with \text{wfmarking-lt} \text{n-notin-lt}
    \item have \text{mark} \ n \neq \ m
      by (simp add: wf-marking-def lt-Node)
    \item with \text{marked-child-lt}
    \item have \text{n-notin-lt}: \text{n} \ \notin \ \text{set} \ (\text{concat} \ ll)
      by simp
    \item show \ ?\text{thesis}
    proof (cases \text{n} \ \in \ \text{set} \ (\text{concat} \ lla))
      \begin{enumerate}
      \item case False with \text{n-not-marka} show \ ?\text{thesis} by simp
      \item case True
        with \text{wf-levelist-subset} \text{[OF wfll-lt]} \text{n-notin-lt}
        \item have \n \ \in \ \text{set-of} \ lt
          by blast
        \item with \text{n-notin-lt} have False by simp
        \item thus \ ?\text{thesis} ..
      \end{enumerate}
    \end{enumerate}
  qed
show \textit{case}
apply (intro conjI ordered-rt mark-rt)
apply clarify
proof
  \textit{case} (goal1 markb nextb levellistb llb)
have llb: Levellist levellistb nextb llb by fact
have wfll-rt: wf-levellist rt lla llb var by fact
have wfmarking-rt: wf-marking rt marka markb m by fact
show \textit{case}
proof (cases rt)
  \textit{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: \( \forall i < \text{length llb}. \ p \notin \text{set (llb} \! \! \text{! i}) \)
    apply
      apply (intro allI impl)
      apply (clarsimp simp add: wf-levellist-def lt-Node)
      apply (case-tac i \leq \text{var l})
      apply (drule-tac x=i in spec)
      using orderedt Tip lt-Node
      apply clarsimp
      apply (drule-tac x=i in spec)
      apply (drule-tac x=i in spec)
      apply clarsimp
    done
  with llb pnN var-p-bounds
  have llc: Levellist (levellistb[\text{var p := p}])
    \( (\text{nextb}(p := \text{levellistb} \! \! \text{! var p)}) \)
    \( (\text{llb}[\text{var p := p \# llb} \! \! \text{! var p}]) \)
    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=\text{var p})
    apply simp
    done
ultimately show \textit{thesis}
using Tip lt-Node varsll orderedt lt rt pnN wfll-lt wfmarking-lt
apply (clarsimp simp add: wf-levellist-def wfm-marking-def)
apply (intro conjI)
apply (rule allI)
apply (rule conjI)
apply (erule-tac x=q in allE)
apply (case-tac var p = var q)
apply fastforce
apply fastforce
apply (case-tac var p = var q)
apply hypsubst-thin
apply fastforce
apply fastforce
apply (rule allI)
apply (rotate-tac 4)
apply (erule-tac x=i in allE)
apply (case-tac i=var p)
apply simp
apply (case-tac var (low p) < i)
apply simp
apply simp
apply (erule exE)
apply (rule-tac x=prx in exI)
apply (intro conjI)
apply simp
apply clarify
apply (rotate-tac 15)
apply (erule-tac x=pt in ballE)
apply fastforce
apply fastforce
done

next
case (Node lrt r rrt)
  have rt-Node: rt = Node lrt r rrt by fact
from wfll-rt rt-Node
  have llb-eq-lla: length llb = length lla
    by (simp add: wf-levellist-def)
  with llb lla
    have llb-eq-lla': length levellistb = length levellista
      by (simp add: Levellist-length)
from wfll-rt rt-Node
  have nodes-in-llb: ∀ q. q ∈ set-of rt → q ∈ set (llb ! (q→var))
    by (simp add: wf-levellist-def)
from wfll-rt rt-Node
  have llb-st: (∀ i ≤ (high p)→var.
    (∃ prx. (llb ! i) = prx@(lla ! i) ∧
      (∀ pt ∈ set prx. pt ∈ set-of rt ∧ pt→var = i))
  by (simp add: wf-levellist-def)

68
from wfll-rt rt-Node rt
have llb-nc:
\( \forall i. ((\text{high } p) \rightarrow \text{var }) < i \longrightarrow (\text{llb } i) = (\text{lla } i) \)
by (simp add: wf-levellist-def)
from wfnmarking-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 wfnmarking-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 wtll-lt
have p-notin-llb:
\( \forall i < \text{length } \text{llb}. p \notin (\text{llb } i) \)
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 (clarsimp simp add: Levellist-def map-update)
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
apply clarsimp
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
apply clarsimp
done
from wtll-lt wtll-rt varsll lla llb
obtain var-p-bounds: \( \text{var } p < \text{length } \text{levellistb } \text{var } p < \text{length } \text{llb} \)
by (simp add: Levellist-length wf-levellist-def rt-Node rt-Node)
with p-notin-llb llb pnN var-p-bounds
have ic: Levellist (levellistb[\text{var } p := p])
\( \text{(nextb}(p := \text{levellistb } ! \text{var } p)) \)
\( (\text{lb}[\text{var } p := p \neq \text{llb } ! \text{var } p]) \)
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=\text{var } p)
apply simp
apply simp

69
then show \(\text{?thesis}\)

**proof (clarsimp)**

show \(\text{wf-levelist (Node } l t \ p r t) \ ll (\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p]) \ \text{var} \land \)

\(\text{wf-marking (Node } l t \ p r t) \ \text{mark} (\text{markb} (p := m)) \ m\)

**proof**

have \(\text{nodes-in-upllb} : \forall \ q. \ q \in \text{set-of } (\text{Node } l t \ p r t)\)

\(\rightarrow q \in \text{set } (\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p] ! (\text{var } q))\)

apply 

apply (\text{rule allI})

apply (\text{rule impI})

**proof**

fix \(q\)

assume \(q\text{-in-\text{t}} : q \in \text{set-of } (\text{Node } l t \ p r t)\)

show \(q\text{-in-upllb} :\)

\(q \in \text{set } (\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p] ! (\text{var } q))\)

**proof (cases } q \in \text{set-of } rt\)

case True

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

by fastforce

from \(\text{orderedt } rt\text{-Node } l t\text{-Node } l t\ \text{rt}\)

have \(\text{ordered-rt} : \text{ordered } rt \ \text{var}\)

by fastforce

from \(\text{True } rt \text{ ordered-rt } rt\text{-Node } l t\text{-Node } l t\ \text{have } q \leq \text{var } r\)

apply 

apply (\text{drule subnodes-ordered})

apply fastforce

apply fastforce

apply fastforce

done

with \(\text{orderedrt } rt \text{ rt-Node } l t\text{-Node } l t\ \text{have } q < \text{var } p\)

by fastforce

then have

\(\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p] ! \ \text{var } q = \)

\(\text{llb} ! \ \text{var } q\)

by fastforce

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

by fastforce

next

assume \(q\text{-notin-rt} : q \notin \text{set-of } rt\)

show \(q \in \text{set } (\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p] ! \ \text{var } q)\)

**proof (cases } q \in \text{set-of } lt\)

case True

assume \(q\text{-in-lt} : q \in \text{set-of } lt\)

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

by fastforce

from \(\text{orderedt } rt\text{-Node } l t\text{-Node } l t\ \text{rt}\)

have \(\text{ordered-lt} : \text{ordered } lt \ \text{var}\)

by fastforce

next

assume \(q\text{-notin-lt} : q \notin \text{set-of } lt\)

show \(q \in \text{set } (\text{llb} [\text{var } p := p \# \text{llb} \ ! \ \text{var } p] ! \ \text{var } q)\)

**proof (cases } q \in \text{set-of } ll\)

case True

assume \(q\text{-in-ll} : q \in \text{set-of } ll\)

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

by fastforce

from \(\text{orderedt } rt\text{-Node } l t\text{-Node } l t\ \text{rt}\)

have \(\text{ordered-lt} : \text{ordered } lt \ \text{var}\)

by fastforce

70
from q-in-lt lt ordered-lt rt-Node rt lt-Node
have var q ≤ var l
  apply −
  apply (drule subnodes-ordered)
  apply fastforce
  apply fastforce
  apply fastforce
  done
with orderedlt rt lt rt-Node lt-Node have qsp: var q < var p
  by fastforce
then show ?thesis
proof (cases var q ≤ var (high p))
  case True
  with llb-st
  have ∃ prx. (llb ! (var q)) = prx@[lla ! (var q)]
      by fastforce
  with nodes-in-lla q-in-lla
  have q-in-lla: q ∈ set (llb ! (var q))
      by fastforce
  from qsp
  have llb[var p := p # llb ! (var p)] ! var q =
      llb ! (var q)
      by fastforce
  with q-in-lla show ?thesis
      by fastforce
next
  assume ¬ var q ≤ var (high p)
  with llb-nc have llb ! (var q) = lla ! (var q)
      by fastforce
  with q-in-lla have q-in-lla: q ∈ set (llb ! (var q))
      by fastforce
  from qsp have
      llb[var p := p # llb ! (var p)] ! var q = llb ! (var q)
      by fastforce
  with q-in-lla show ?thesis
      by fastforce
qed
next
  assume q-notin-lt: q /∈ set-of lt
  with q-notin-rt rt lt rt-Node lt-Node q-in-t have qp: q = p
      by fastforce
  with varll lla-eq-ll lla-eq-lla have var p < length llb
      by fastforce
  with qp show ?thesis
      by simp
qed
qed
have prx-ll-st: ∀ i ≤ var p.
(\exists \text{prx}. \text{llb}[\text{var } p := \text{p} \# \text{llb}[\text{var } p]!] i = \text{prx}@[\text{llb}]i) \\
(\forall pt \in \text{set prx}. pt \in \text{set-of} (\text{Node } lt p rt) \land \text{var } pt = i))

\text{apply} \\
\text{apply (rule allI)} \\
\text{apply (rule impI)}

\text{proof} \\
\text{fix } i \\
\text{assume } i\text{sep}: i \leq \text{var } p \\
\text{show } \exists \text{prx}. \text{llb}[\text{var } p := \text{p} \# \text{llb}[\text{var } p]!] i = \text{prx}@[\text{llb}]i \land \\
(\forall pt \in \text{set prx}. pt \in \text{set-of} (\text{Node } lt p rt) \land \text{var } pt = i)

\text{proof} (\text{cases } i = \text{var } p)

\text{case True}
\text{with orderedlt } lt \text{-Node rt rt-Node}
\text{have } lpsp: \text{var } (\text{low } p) < \text{var } p \\
\text{by fastforce}
\text{with orderedlt } lt \text{-Node rt rt-Node}
\text{have } hpsp: \text{var } (\text{high } p) < \text{var } p \\
\text{by fastforce}
\text{with } lpsp lla-nc
\text{have } llall: lla ! \text{var } p = ll ! \text{var } p \\
\text{by fastforce}
\text{with } hpsp llb-nc \text{have } llb ! \text{var } p = ll ! \text{var } p \\
\text{by fastforce}
\text{with } llb-eq-lla lla-eq-ll \text{isep } True \text{ varsll lt rt show } \text{thesis}
\text{apply} \\
\text{apply (rule-tac } x=[\text{p}] \text{ in } exI) \\
\text{apply (rule conjI)} \\
\text{apply simp} \\
\text{apply (rule ballI)} \\
\text{apply fastforce} \\
\text{done}

\text{next}
\text{assume } inp: i \neq \text{var } p \\
\text{show } \text{thesis}
\text{proof} (\text{cases } \text{var } (\text{low } p) < i)

\text{case True}
\text{with } lla-nc \text{have } llall: lla ! i = ll ! i \\
\text{by fastforce}
\text{assume } vpsi: \text{var } (\text{low } p) < i \\
\text{show } \text{thesis}
\text{proof} (\text{cases } \text{var } (\text{high } p) < i)

\text{case True}
\text{with } llall llb-nc \text{have } llb ! i = ll ! i \\
\text{by fastforce}
\text{with } inp \text{ True vpsi varsll lt rt show } \text{thesis}
\text{apply} \\
\text{apply (rule-tac } x=[] \text{ in } exI) \\
\text{apply (rule conjI)} \\
\text{apply simp}
apply $(\text{rule ballI})$
apply fastforce
done

next
assume isehp: $\neg \text{var (high p)} < i$
with vpsi lla-nc have lla-li: lla ! i = ll ! i
  by fastforce
with isehp llb-st
have prx-lia: $\exists \text{prx. llb } i = \text{prx } @ lla ! i$
  $\forall pt \in \text{set prx. pt } \in \text{set-of rt } \land \text{var pt } = i$
  apply --
  apply (erule-tac $x=i$ in allE)
  apply simp
done
with lla-li inp rt show ?thesis
  apply --
  apply (erule exE)
  apply (rule-tac $x=\text{prx}$ in exI)
  apply simp
done
qed

next
assume iselp: $\neg \text{var (low p)} < i$
show ?thesis
proof (cases $\text{var (high p)} < i$)
case True
with llb-nc have llb-li: llb ! i = lla ! i
  by fastforce
with iselp lla-st
have prx-lia: $\exists \text{prx. lla } i = \text{prx } @ ll ! i$
  $\forall pt \in \text{set prx. pt } \in \text{set-of lt } \land \text{var pt } = i$
  apply --
  apply (erule-tac $x=i$ in allE)
  apply simp
done
with llb-li inp lt show ?thesis
  apply --
  apply (erule exE)
  apply (rule-tac $x=\text{prx}$ in exI)
  apply simp
done
next
assume isehp: $\neg \text{var (high p)} < i$
from iselp lla-st
have prxh: $\exists \text{prx. llb } i = \text{prx } @ lla ! i$
  $\forall pt \in \text{set prx. pt } \in \text{set-of lt } \land \text{var pt } = i$
  by fastforce
from isehp llb-st
have prxh: $\exists \text{prx. llb } i = \text{prx } @ lla ! i$
\[(\forall pt \in \text{set prx}, pt \in \text{set-of rt} \land \text{var pt} = i)\]

by \text{fastforce}

with \text{prxI inp lt pnN rt show ?thesis}

apply \text{--}
apply \text{(elim exE)}
apply \text{(rule-tac \text{x=}prxa \& prx in exI)}
apply \text{simp}
apply \text{(elim conjE)}
apply \text{fastforce}

done

qed

qed

qed

have \text{big-Nodes-nc:} \forall i. (p \rightarrow \text{var}) < i

\rightarrow (\text{llb}[\text{var p} := p \# \text{llb ! var p}]) ! i = \text{ll} ! i

apply \text{--}
apply \text{(rule allI)}
apply \text{(rule impI)}

proof --

fix i

assume psi: \text{var p} < i

with \text{ordertd lt rt lt-Node rt-Node have} lpsi: \text{var (low p) < i}

by \text{fastforce}

with \text{lla-nc have} lla-ll: lla ! i = ll ! i

by \text{fastforce}

from psi ordertd lt rt lt-Node rt-Node have hpsi: \text{var (high p) < i}

by \text{fastforce}

with \text{llb-nc have} llb-lla: llb ! i = lla ! i

by \text{fastforce}

from psi

have upllb-llb: \text{llb}[\text{var p} := p \# \text{llb ! var p}] ! i = \text{llb} ! i

by \text{fastforce}

from upllb-llb llb-lla lla-ll

show \text{llb}[\text{var p} := p \# \text{llb ! var p}] ! i = \text{ll} ! i

by \text{fastforce}

qed

from lla-eq-ll llb-eq-lla

have \text{length-eq: length (llb[\text{var p} := p \# \text{llb ! var p}])} = \text{length ll}

by \text{fastforce}

from \text{length-eq big-Nodes-nc prx-ll-st nodes-in-upllb}

have \text{wf-ll-upllb:}

\text{wf-levellist (Node lt p rt) ll (llb[\text{var p} := p \# \text{llb ! var p}]}) \text{ var}

by \text{(simp add: wf-levellist-def)}

have \text{mark-nc:}

\forall n. n \notin \text{set-of (Node lt p rt)} \rightarrow (\text{markb(p=}m)) n = \text{mark n}

apply \text{--}
apply \text{(rule allI)}
apply \text{(rule impI)}
proof –
  fix n
  assume nnit: n ≠ set-of (Node lt p rt)
  with lt rt have nnilt: n ≠ set-of lt
      by fastforce
from nnit lt rt have nnirt: n ∉ set-of rt
      by fastforce
with nnilt mot-nc mort-nc have mb-eq-m: markb n = mark n
      by fastforce
from nnit have n ≠ p
      by fastforce
then have upmarkb-markb: (markb(p := m)) n = markb n
      by fastforce
with mb-eq-m show (markb(p := m)) n = m
      by fastforce
qed

have mark-c: ∀ n. n ∈ set-of (Node lt p rt) → (markb(p := m)) n = m

apply –
apply (intro allI)
apply (rule impI)

proof –
  fix n
  assume nint: n ∈ set-of (Node lt p rt)
  show (markb(p := m)) n = m
  proof (cases n = p)
    case True
    then show ?thesis
      by fastforce
next
  assume nnp: n ≠ p
  show ?thesis
  proof (cases n ∈ set-of rt)
    case True
    with mirt-marked have markb n = m
      by fastforce
    with nnp show ?thesis
      by fastforce
next
  assume nninrt: n ∉ set-of rt
  with nint nnp have nnilt: n ∈ set-of 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 nnp show ?thesis
by fastforce
qed
qed
qed
from mark-c mark-nc
have wf-mark: wf-marking (Node lt p rt) mark (markb(p := m)) m
by (simp add: wf-marking-def)
with wf-ll-upllb show ?thesis
by fastforce
qed
qed
qed
qed
qed
qed

next
fix var low high p lt rt and levellist and
ll::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 ordered: 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 ∈ set (ll ! var p)
apply −
apply (drule-tac x=p in bspec)
apply simp
apply fastforce
done

from marked-st marked-child-ll lt rt show ?thesis
  apply –
  apply (rule allI)
  apply (erule-tac x=q in allE)
  apply (rule impI)
  apply (erule impE)
  apply assumption
  apply (erule-tac x=q in bspec)
  apply simp
  apply fastforce
  done

have levellist-nc: ∀ i ≤ var p. (∃ prx. ll ! i = prx@(ll ! i) ∧
  (∀ pt ∈ set prx. pt ∈ set-of (Node lt p rt) ∧ var pt = i))
  apply –
  apply (rule allI)
  apply (rule impI)
  apply (rule-tac x=∅ in exI)
  apply fastforce
  done

have ll-nc: ∀ i. (var p) < i −→ ll ! i = ll ! i
  by fastforce

have length-ll: length ll = length ll
  by fastforce

with ll-nc have levellist-nc nodest-in-ll

have wf: wf-levellist (Node lt p rt) ll ll var
  by (simp add: wf-levellist-def)

have m-nc: ∀ n. n ∈ set-of (Node lt p rt) −→ mark n = mark n
  by fastforce

from marked-st have ∀ n. n ∈ set-of (Node lt p rt) −→ mark n = mark p
  by fastforce

with m-nc have wf-marking (Node lt p rt) mark mark (mark p)
  by (simp add: wf-marking-def)

with wf show ?thesis
  by fastforce

qed

lemma allD: ∀ ll. P ll ⇒ P ll
  by blast

lemma replicate-spec: [∀ i < n. xs ! i = x; n=length 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 \( \forall \sigma \ t \. \Gamma, \Theta \vdash t \ldots \)

\[
\begin{align*}
\hline
\sigma. \text{Dag } \check{p}. \text{'low} \ ' \text{high } t & \wedge (\forall i < \text{length } \text{levellist}. \text{levellist} ! i = \text{Null}) \wedge \\
\text{length } \text{levellist} & = \check{p} \rightarrow \check{\text{var}} + 1 \wedge \\
\text{ordered } t & \check{\text{var}} \wedge (\forall n \in \text{set-of } t. \text{mark } n = (~ \check{m}) )
\end{align*}
\]

\[
\begin{align*}
\text{levellist} & ::= \text{PROC Levellist} (\check{p}, \check{m}, \text{levellist}) \\
\text{allD} & \text{[OF Levellist-spec-total]} \\
\end{align*}
\]
apply (simp add: Levellist-length)
apply (rule allI)
apply (rule implI)
apply (rule ballI)
apply (rotate-tac 11)
apply (erule-tac x=k in allE)
apply (rename-tac dag1 ref dag2 k pa)
apply (subgoal-tac k <= var ref)
prefer 2
apply (subgoal-tac ref = p)
apply simp
apply clarify
apply (erule-tac ?P = Dag p low high (Node dag1 ref dag2) in rev-mp)
apply (simp (no-asms))
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
apply simp
done

end

7 Proof of Procedure ShareRep

theory ShareRepProof imports ProcedureSpecs ../Simpl/HeapList begin

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

lemma hd-filter-cons:
  \forall i. \[ P \ \{xs \! i\ p\}; i < \text{length } xs; \forall no \in \text{set } (\text{take } i \ xs). \neg P \ \{no\ p\}; \forall a\ b. \ P\ a\ b = P\ b\ a\]
  \implies \[xs\! i = \text{hd } (\text{filter } P\ p)\ \xs\]
  apply (induct xs)
  apply simp
  apply (case-tac P a p)
  apply simp
  apply (case-tac i)
  apply simp
  apply simp
  done
apply (case-tac i)
apply simp
apply auto
done

lemma (in ShareRep-impl) ShareRep-spec-total:
shows
\( \forall \sigma \, \text{List 'nodeslist 'next ns} \land \)
\( (\forall \text{no } \in \text{set ns. no } \neq \text{Null} \land \)
\( ((\text{no} \rightarrow \text{low} = \text{Null}) = (\text{no} \rightarrow \text{high} = \text{Null})) \land \)
\( (\text{isLeaf-pt 'p low 'high } \rightarrow \text{isLeaf-pt no 'low 'high}) \land \)
\( \text{no} \rightarrow \text{'var = 'p} \rightarrow \text{'var}) \land \)
\( 'p \in \text{set ns} \}
PROC ShareRep ('nodeslist, 'p)
\( (\forall \text{pt. } \text{pt} \neq 'p \rightarrow \text{pt} \rightarrow 'p \rightarrow \text{rep} = \text{pt} \rightarrow 'p \rightarrow \text{rep}) \land \)
\( ('p \rightarrow \text{rep} \rightarrow \text{'var = 'p} \rightarrow \text{'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 \text{Null})
INV \( \exists \text{prx sfx. List 'nodeslist 'next sfx} \land \text{ns} = \text{prx}\@\text{sfx} \land \)
\( \neg \text{isLeaf-pt 'p 'low 'high} \land \)
\( (\forall \text{no } \in \text{set ns. no } \neq \text{Null} \land \)
\( ((\text{no} \rightarrow \text{low} = \text{Null}) = (\text{no} \rightarrow \text{high} = \text{Null})) \land \)
\( \text{isLeaf-pt 'p 'low 'high } \rightarrow \text{isLeaf-pt no 'low 'high}) \land \)
\( \text{no} \rightarrow \text{'var = 'p} \rightarrow \text{'var}) \land \)
\( 'p \in \text{set ns} \land \)
\( (((\exists \text{pt } \in \text{set prx. } \text{repNodes-eq pt 'p 'low 'high 'rep}) \land \\
\rightarrow 'p \rightarrow \text{p}) = \text{hd} (\text{filter } (\lambda \text{sn. } \text{repNodes-eq sn 'p 'low 'high 'rep} \text{ prx}) \land \\
(\forall \text{pt. } \text{pt} \neq 'p \rightarrow \text{pt} \rightarrow 'p \rightarrow \text{rep} = \text{pt} \rightarrow 'p \rightarrow \text{rep}) \land \\
((\forall \text{pt } \in \text{set prx. } \neg \text{repNodes-eq pt 'p 'low 'high 'rep} \rightarrow 'p \rightarrow 'p \land \\
(\neg 'nodeslist \neq \text{Null} \land \\
(\forall \text{pt } \in \text{set prx. } \neg \text{repNodes-eq pt 'p 'low 'high 'rep} \rightarrow 'p \rightarrow 'p \land \\
(\text{p} = 'p \land 'high = 'high \land 'low = 'low})) \}
VAR MEASURE (\text{length } (\text{list 'nodeslist 'next}))
DO
IF (\text{repNodes-eq 'nodeslist 'p 'low 'high 'rep})
THEN 'p \rightarrow 'rep ::= 'nodeslist;; 'nodeslist ::= \text{Null}
ELSE 'nodeslist ::= 'nodeslist\rightarrow 'next
FI
OD
FI in HoareTotal.annotateI)
apply vcg
using [[\text{simpl-depth-limit = 2]}]
apply (rule conjI)

80
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 sfz)
apply (rule-tac x=prx@[nodeslist] in exI)
apply (rule-tac x=sfx in exI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply simp
prefer 4
apply (elim exE conjE)
apply (simp (no-asm-use))
apply hypsubst
using [[simp-depth-limit = 100]]
proof –

fix ns var low high rep next p nodeslist
assume ns: List nodeslist next ns
assume no-prop: \( \forall no \in \text{set } ns. \) 
\( no \neq \text{Null} \land \)
\( (\text{low } no = \text{Null}) = (\text{high } no = \text{Null}) \land \)
\( (\text{isLeaf-pt } p \text{ low high } \rightarrow \text{isLeaf-pt } no \text{ low high}) \land \text{var } no = \text{var } p \)
assume p-in-ns: p \( \in \) set ns
assume p-Leaf: \text{isLeaf-pt } p \text{ low high}
show nodeslist = hd [sn←ns . repNodes-eq sn p low high rep] \land 
\text{var } nodeslist = \text{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 . 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

82
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
ultimately show ?thesis
using no-props
by (intro conjI)
qed

next

83
fix var low high p repa next nodeslist prx sfx
assume nodeslist-not-Null: nodeslist ≠ Null
assume sfx: List nodeslist next sfx
assume p-not-Leaf: ¬ isLeaf-pt p low high
assume no-props: ∀ no∈set prx ∪ set sfx.
   no ≠ Null ∧
   (low no = Null) = (high no = Null) ∧
   (isLeaf-pt p low high → isLeaf-pt no low high) ∧ var no = var p
assume p-in-ns: p ∈ set prx ∨ p ∈ set sfx
assume match-prx: (∃ pt∈set prx. repNodes-eq pt p low high repa) →
   repa p = hd [sn←prx . repNodes-eq sn p low high repa]
assume nomatch-prx: ∀ pt∈set prx. ¬ repNodes-eq pt p low high repa
assume match: repNodes-eq nodeslist p low high repa
show (∀ no∈set prx ∪ set sfx.
   no ≠ Null ∧
   (low no = Null) = (high no = Null) ∧
   (isLeaf-pt p low high → isLeaf-pt no low high) ∧ var no = var p) ∧
   (p ∈ set prx ∨ p ∈ set sfx) ∧
   ((∃ pt∈set prx ∪ set sfx. repNodes-eq pt p low high repa) →
    nodeslist =
    hd ([sn←prx . repNodes-eq sn p low high repa] @
     [sn←sfx . repNodes-eq sn p low high repa])) ∧
   ((∀ pt∈set prx ∪ set sfx. ¬ repNodes-eq pt p low high repa) →
    repa = repa(p := nodeslist))
proof
  from nodeslist-not-Null sfx
  obtain sfx' where sfx'=nodeslist#sfx'
    by (cases nodeslist=Null) auto
  from nomatch-prx match sfx'
  have hd: hd ([sn←prx . repNodes-eq sn p low high repa] @
    [sn←sfx . repNodes-eq sn p low high repa]) = nodeslist
    by simp
  from match sfx'
  have triv: (∀ pt∈set prx ∪ set sfx. ¬ repNodes-eq pt p low high repa) →
    repa = repa(p := nodeslist))
    by simp
  show ?thesis
    apply (rule conjI)
    apply (rule no-props)
    apply (intro conjI)
    apply (rule p-in-ns)
    apply (simp add: hd)
    apply (rule triv)
  done
qed
qed
end
8 Proof of Procedure ShareReduceRepList

theory ShareReduceRepListProof imports ShareRepProof begin

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

lemma hd-filter-app: \( [\text{filter} \ P \ [x::y]] \quad \Rightarrow \)
\( h d \ (\text{filter} \ P \ [x]) \quad = \quad h d \ (\text{filter} \ P \ [x]) \)
by (induct \( x \) arbitrary; n m) auto

lemma (in ShareReduceRepList-impl) ShareReduceRepList-spec-total:
defines var-eq \( \equiv (\lambda ns \ \var. \ (\forall \ no2 \in set \ ns. \ \forall \ no1 \in set \ ns. \ \no1 \rightarrow \var = \no2 \rightarrow \var)) \)
shows
\( \forall \sigma \ ns. \ \Gamma \vdash \)
\( \{ \var \} \). List 'nodeslist 'next ns \&
\( (\forall \ no \in set \ ns. \)
\( \no \neq \text{null} \ \& \ ((\no \rightarrow \text{low} = \text{null}) = (\no \rightarrow \text{high} = \text{null})) \ \&
\( \no \rightarrow \text{low} \notin \text{set} \ ns \ \& \no \rightarrow \text{high} \notin \text{set} \ ns \ \&
\( (\text{isLeaf-pt} \ \no \ \text{low} \ '\text{high} = (\no \rightarrow \var \leq 1)) \ \&
\( (\no \rightarrow \text{low} \neq \text{null} \quad \rightarrow \quad (\no \rightarrow \text{low}) \rightarrow \text{rep} \neq \text{null}) \ \&
\( (\text{rep} \ \text{low} \no \notin \text{set} \ ns) \ \&
\)\( \forall \ no1 \in set \ ns.
\)\( ((\text{rep} \ \text{low} \no1 = (\text{rep} \ \text{high} \no) \ \&
\( (\text{rep} \ \text{low} \no1 = (\text{rep} \ \text{high} \no) = (\no \rightarrow \text{rep} = no1 \rightarrow \text{rep})))))\}
apply (hoare-rule HoareTotal.ProcNoRec1)
apply (hoare-rule anno=
'tnode :==' 'nodeslist;;
WHILE (\'node \neq \text{null} )
INV \{\exists \prx \ sfx. \ \text{List} 'node 'next sfx \&
\text{List} 'nodeslist 'next ns \& ns=\prx@sfx \&
(\forall \ no \in set \ ns. \)
\( \no \neq \text{null} \ \& \ ((\no \rightarrow \text{low} = \text{null}) = (\no \rightarrow \text{high} = \text{null})) \ \&
\( \no \rightarrow \text{low} \notin \text{set} \ ns \ \& \no \rightarrow \text{high} \notin \text{set} \ ns \ \&
\( (\text{isLeaf-pt} \ \no \ \text{low} \ '\text{high} = (\no \rightarrow \var \leq 1)) \ \&
\( \no \rightarrow \text{low} \neq \text{null} \rightarrow (\no \rightarrow \text{low}) \rightarrow \text{rep} \neq \text{null}) \ \&
\( (\text{rep} \ \text{low} \no1 = (\text{rep} \ \text{high} \no) = (\no \rightarrow \text{rep} = no1 \rightarrow \text{rep})))))\}

\[ ((\sigma_{\text{rep}} \propto \sigma_{\text{low}}) \text{ no } \notin \text{ set } \text{ns}) \land \\
\text{var-eq ns } \text{var} \land \\
(\forall \text{no} . \text{ no } \notin \text{ set } \text{prx} \rightarrow \text{no} \rightarrow \sigma_{\text{rep}} = \text{no} \rightarrow \sigma_{\text{rep}}) \land \\
(\forall \text{no} \in \text{ set } \text{prx} . \text{ no} \rightarrow \sigma_{\text{rep}} \neq \text{Null} \land \\
(\text{if } ((\sigma_{\text{rep}} \propto \sigma_{\text{low}}) \text{ no } = (\sigma_{\text{rep}} \propto \sigma_{\text{high}}) \text{ no} \land \text{ no} \rightarrow \sigma_{\text{low}} \neq \text{Null}) \\
\text{then } (\text{no} \rightarrow \sigma_{\text{rep}} = (\sigma_{\text{rep}} \propto \sigma_{\text{low}}) \text{ no}) \\
\text{else } ((\text{no} \rightarrow \sigma_{\text{rep}}) = \text{hd} (\lambda \text{sn} . \text{repNodes-eq sn no} \sigma_{\text{low}} \sigma_{\text{high}} \sigma_{\text{rep}}) \\
\text{prx}) \land \\
((\text{no} \rightarrow \sigma_{\text{rep}}) \rightarrow \sigma_{\text{rep}}) = \text{no} \rightarrow \sigma_{\text{rep}} \land \\
(\forall \text{no1} \in \text{ set } \text{prx} . \\
((\sigma_{\text{rep}} \propto \sigma_{\text{high}}) \text{ no1 } = (\sigma_{\text{rep}} \propto \sigma_{\text{high}}) \text{ no} \land \\
(\sigma_{\text{rep}} \propto \sigma_{\text{low}}) \text{ no1 } = (\sigma_{\text{rep}} \propto \sigma_{\text{low}}) \text{ no}) = \\
(\text{no} \rightarrow \sigma_{\text{rep}} = \text{no1} \rightarrow \sigma_{\text{rep}}))) \land \\
\text{nodeslist} = \sigma_{\text{nodeslist}} \land \text{high} = \sigma_{\text{high}} \land \text{low} = \sigma_{\text{low}} \land \text{var} = \sigma_{\text{var}} \} \]

\text{VAR MEASURE } (\text{length } (\text{list } \text{node } \text{next})) 

\text{DO} 
\text{IF } (\neg \text{isLeaf-pt } \text{node } \text{low} \text{high} \land \\
\text{node} \rightarrow \text{low} \rightarrow \sigma_{\text{rep}} = \text{node} \rightarrow \text{high} \rightarrow \sigma_{\text{rep}}) 
\text{THEN } \text{node} \rightarrow \sigma_{\text{rep}} ::= \text{node} \rightarrow \text{low} \rightarrow \sigma_{\text{rep}} 
\text{ELSE \ CALL \ ShareRep } (\text{nodeslist} , \text{node}) 
\text{FI} ;; 
\text{node} ::= \text{node} \rightarrow \text{next} 
\text{OD \ in \ HoareTotal.annotate1} 

\text{apply } (\text{veg spec=} \text{spec-total}) 
\text{apply } (\text{rule-tac } x=[] \text{ in } \text{exI}) 
\text{apply } (\text{rule-tac } x=\text{ns in } \text{exI}) 
\text{using } [[\text{simp-depth-limit} = 2]] 
\text{apply } (\text{simp (no-asm-use)}) 
\text{prefer } 2 
\text{using } [[\text{simp-depth-limit} = 4]] 
\text{apply } (\text{clar simp}) 
\text{prefer } 2 
\text{apply } (\text{rule conjI}) 
\text{apply } \text{clarify} 
\text{apply } (\text{rule conjI}) 
\text{apply } (\text{clar simp simp add: List-list}) 
\text{apply } (\text{simp only: List-not-Null simp-thms triv-forall-equality}) 
\text{apply } \text{clarify} 
\text{apply } (\text{simp only: triv-forall-equality}) 
\text{apply } (\text{rename-tac sfx}) 
\text{apply } (\text{rule-tac } x=\text{prx}\text{[node] in } \text{exI}) 
\text{apply } (\text{rule-tac } x=\text{sfx in } \text{exI}) 
\text{apply } (\text{rule conjI}) 
\text{apply } \text{assumption} 
\text{apply } (\text{rule conjI}) 
\text{apply } (\text{simp (no-asm)}) 
\text{apply } (\text{rule conjI}) 
\text{apply } (\text{assumption}) 
\text{prefer } 2
apply clarify
apply (simp only: List-not-Null simp-thms' triv-forall-equality)
apply clarify
apply (simp only: triv-forall-equality)
apply (rename-tac sfx)
apply (rule-tac x=prx@node#sfx in exI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (rule ballI)
apply (frule-tac x=no in bspec, assumption)
apply (erule-tac x=no in bspec)
apply (simp (no-asm-use))
apply (elim conjE)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (rule conjI)
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (rule conjI)
apply assumption
apply (unfold var-eq-def)
apply (drule-tac x=no 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-asms))
apply (simp (no-asms))
apply (rule conjI)
apply (rule conjI)
apply (simp (no-asm))
apply (rule conjI)
apply (rule conjI)
apply (assumption)
using [[simp-depth-limit = 100]]

proof –

fix var low high rep nodeslist ns repa next no
assume ns: List nodeslist next ns
assume no-in-ns: no ∈ set ns
assume while-inv: ∀ no∈set ns.
  repa no ≠ Null ∧
  (if (repa ∞ low) no = (repa ∞ high) no ∧ high no ≠ Null
  then repa no = (repa ∞ low) no
  else repa no = hd [sn←ns . repNodes-eq sn no low high repa] ∧
  repa (repa no) = repa no ∧
\[
(\forall \text{no1} \in \text{set ns}. \\
(\text{repa} \propto \text{high} \ \text{no1} = (\text{repa} \propto \text{high} \ \text{no}) \land \\
(\text{repa} \propto \text{low} \ \text{no1} = (\text{repa} \propto \text{low} \ \text{no}) = \\
(\text{repa no} = (\text{repa no1})))
\]

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

**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 no} = (\text{repa} \propto \text{high no}) \rightarrow \text{high no} = \text{Null}
\]

**assume unmodif:** \(\forall \text{no}. \text{no} \notin \text{set ns} \rightarrow \text{rep no} = \text{repa no}
\]

**show** \(\text{hd (sn←ns . repNodes-eq sn no low high repa) \in set ns} \land \\
\text{rep (hd (sn←ns . repNodes-eq sn no low high repa)}) = \\
\text{hd (sn←ns . repNodes-eq sn no low high repa)}
\]

**proof**

- **from no-in-ns pre obtain**
  - 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})
  
  by blast

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

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

  from no-in-ns repNodes-eq-same-node
  
  **have** \(\text{repNodes-not-empty: [sn←ns . repNodes-eq sn no low high repa]} \neq []\)
  
  by (rule filter-not-empty)

  **then have** \(\text{hd-term-in-ns: hd (sn←ns . repNodes-eq sn no low high repa) \in set ns}\)
  
  **by** (rule hd-filter-in-list)

  **with** while-inv obtain
  
  repa-hd-nNull: \(\text{rep (hd (sn←ns . repNodes-eq sn no low high repa)}) \neq \text{Null}\)
  
  by auto

  let \(\text{?hd} = \text{hd (sn←ns . repNodes-eq sn no low high repa)}\)

  from hd-term-in-ns pre obtain
  
  hd-nNull: \(\text{?hd} \neq \text{Null} \land \text{hd-balanced} \) \(\text{low (hd (sn←ns . repNodes-eq sn no low high repa)}) = \text{Null}\) = \\
  \(\text{high (hd (sn←ns . repNodes-eq sn no low high repa)}) = \text{Null}\) \land \text{isLeaf-var} \text{isLeaf-pt (hd (sn←ns . repNodes-eq sn no low high repa)}) \text{low high} = \\
  \text{(var (hd (sn←ns . repNodes-eq sn no low high repa)}) \leq \text{Suc 0)}
  
  by blast

  **have** \(\text{rep (hd (sn←ns . repNodes-eq sn no low high repa)}) = \\
  \text{hd (sn←ns . repNodes-eq sn no low high repa)}\)

  **proof** (cases high no = Null)
case True
with no-balanced have low no = Null
  by simp
with True have no-Leaf: isLeaf-pt no low high
  by (simp add: isLeaf-pt-def)
with isLeaf-var have varno: var no <= 1
  by simp
from same-var [rule-format, OF no-in-ns hd-term-in-ns] varno
have var (hd [sn←ns . repNodes-eq sn no low high repa]) \leq 1
  by simp
with hd-isLeaf-var have
  isLeaf-pt (hd [sn←ns . repNodes-eq sn no low high repa]) low high
  by simp
with while-inv hd-term-in-ns repNodes-not-empty show thesis
  apply (simp add: isLeaf-pt-def)
  apply (erule-tac x= hd [sn←ns . repNodes-eq sn no low high repa] in ballE)
  prefer 2
  apply simp
  apply (simp (no-asm-use) add: repNodes-eq-def)
  apply (rule filter-hd-P-rep-indep)
  apply (simp (no-asm-simp))
  apply (simp (no-asm-simp))
  apply assumption
  done
next
assume hno-nNull: high no \neq Null
with share-case have repchildren-neq: (repa \propto low) no \neq (repa \propto high) no
  by simp
from repNodes-not-empty have
  repNodes-eq (hd [sn←ns . repNodes-eq sn no low high repa]) no low high
  repa
  by (rule hd-filter-prop)
then
have (repa \propto low) (hd [sn←ns . repNodes-eq sn no low high repa]) =
  (repa \propto low) no \land
  (repa \propto high) (hd [sn←ns . repNodes-eq sn no low high repa]) =
  (repa \propto high) no
  by (simp add: repNodes-eq-def)
with repchildren-neq have
  (repa \propto low) (hd [sn←ns . repNodes-eq sn no low high repa]) \neq
  (repa \propto high) (hd [sn←ns . repNodes-eq sn no low high repa])
  by simp
with while-inv hd-term-in-ns repNodes-not-empty show thesis
  apply (simp add: isLeaf-pt-def)
  apply (erule-tac x= hd [sn←ns . repNodes-eq sn no low high repa] in ballE)
  prefer 2
  apply simp
apply (simp (no asm use) add: repNodes-eq-def)
apply (rule filter-hd-P-rep-indep)
apply simp
apply fastforce
apply fastforce
done
qed

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

next

fix var low high rep nodeslist repa next node prx sfx
assume ns: List nodeslist next (prx @ node # sfx)
assume sfx: List (next node) next sfx
assume node-not-Null: node ≠ 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) ∧
  (rep ∝ 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-nln: repa (low node) = repa (high node)
show (∀ no. no ∉ set (prx @ [node]) →
  rep no = (repa(node := repa (high node))) no) ∧
(∀ no ∈ set (prx @ [node]).
  (repa(node := repa (high node))) no ≠ Null ∧
  (if (repa(node := repa (high node)) ∝ low) no =
   (repa(node := repa (high node)) ∝ high) no ∧
   low no ≠ Null
   then (repa(node := repa (high node))) no =
   (repa(node := repa (high node)) ∝ low) no

90
else (repa(node := repa(high node))) no = 

    hd [sn←prx @ [node] . 
    repNodes-eq sn no low high 
    (repa(node := repa(high node)))] \land 
    (repa(node := repa(high node)))

((repa(node := repa(high node))) no) = 
(repa(node := repa(high node))) no \land 
(∀no1∈set (prx @ [node]). 

( (repa(node := repa(high node)) ∝ high)) no1 = 
(repa(node := repa(high node)) ∝ high) no ∨ 
(repa(node := repa(high node)) ∝ low) no1 = 
(repa(node := repa(high node)) ∝ low) no) = 
((repa(node := repa(high node))) no) = 
(repa(node := repa(high node)) no1))

(is ?NodesUnmodif ∧ ?NodesModif)

proof – 

— This proof was originally conducted without the substitution repa(low node) = repa(high node) preformed. So don’t be confused if we show everythin for repa(low node).

from rep-repa-nc

have nodes-unmodif: ?NodesUnmodif

by auto

hence rep-Sucna-nc:

(∀no. no /∉ set (prx @ [node])
    → rep no = (repa(node := repa(low node))) no)

by auto

have nodes-modif: ?NodesModif (is ∀no∈set (prx @ [node]). ?P no ∧ ?Q no)

proof (rule ballI)

fix no
assume no-in-take-Sucna: no ∈ set (prx @ [node])

show ?P no ∧ ?Q no

proof (cases no = node)

    case False
    note no-noteq-nln=this

with no-in-take-Sucna

have no-in-take-n: no ∈ set prx

    by auto

with no-in-take-n while-inv obtain

repa-no-nNull: repa no /≠ Null and
repa-cases: (if (repa ∝ low) no = (repa ∝ high) no ∧ low no /≠ Null
    then repa no = (repa ∝ low) no
else repa no = hd [sn←prx . repNodes-eq sn no low high repa]

∧ repa (repa no) = repa no ∧ 
(∀no1∈set prx. ((repa ∝ high) no1 = (repa ∝ high) no
∧ (repa ∝ low) no1 = (repa ∝ low) no
= (repa no = repa no1)))

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

by auto

from no-in-take-n
have no-in-nodeslist: \( \text{no} \in \text{set}(\text{prx} @ \text{node} \neq \text{sfx}) \)
  by auto
from repa-no-nNull no-noteq-nln have ext-repa-nNull: \(?P \text{ no} \)
  by auto
from no-in-nodeslist nodes-balanced-ordered obtain
  nln-nNull: \( \text{node} \neq \text{Null} \) and
  nln-balanced-children: \((\text{low} \text{ node} = \text{Null}) = (\text{high} \text{ node} = \text{Null})\) and
  lnln-notin-nodeslist: \((\text{low} \text{ node} \notin \text{set}(\text{prx} @ \text{node} \neq \text{sfx})\) and
  isLeaf-var-nln: isLeaf-pt \text{ node low high} = (\text{var} \text{ node} \leq 1) and
  node-nNull-rep-null-nln: \((\text{low} \text{ node} \neq \text{Null}) \rightarrow \text{rep}(\text{low} \text{ node}) \neq \text{Null}\) and
  nln-varrep-le-var: \((\text{rep} \propto \text{low} \text{ node}) \notin \text{set}(\text{prx} @ \text{node} \neq \text{sfx})\)
apply (erule-tac \text{ x=no in ballE})
apply auto
done
from no-in-nodeslist nodes-balanced-ordered no-in-take-Sucna
obtain
  no-nNull: \( \text{no} \neq \text{Null} \) and
  balanced-children: \((\text{low} \text{ no} = \text{Null}) = (\text{high} \text{ no} = \text{Null})\) and
  lno-notin-nodeslist: \((\text{low} \text{ no} \notin \text{set}(\text{prx} @ \text{node} \neq \text{sfx})\) and
  hno-notin-nodeslist: \((\text{high} \text{ no} \notin \text{set}(\text{prx} @ \text{node} \neq \text{sfx})\) and
  isLeaf-var-no: isLeaf-pt \text{ no low high} = (\text{var} \text{ no} \leq 1) and
  node-nNull-rep-null-nln: \((\text{low} \text{ no} \neq \text{Null}) \rightarrow \text{rep}(\text{low} \text{ no}) \neq \text{Null})\) and
  varrep-le-var: \((\text{rep} \propto \text{low} \text{ no}) \notin \text{set}(\text{prx} @ \text{node} \neq \text{sfx})\)
apply –
apply (erule-tac \text{ x=no in ballE})
apply auto
done
from lno-notin-nodeslist
have ext-rep-null-comp-low:
  \((\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{low}) \text{ no} = (\text{repa} \propto \text{low} \text{ no}) \) no
  by (auto simp add: null-comp-def)
from hno-notin-nodeslist
have ext-rep-null-comp-high:
  \((\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{high}) \text{ no} = (\text{repa} \propto \text{high} \text{ no}) \) no
  by (auto simp add: null-comp-def)
have share-reduce-if: \(?Q \text{ no}\)
proof (cases \(\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{low}) \text{ no} = \)
  \((\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{high}) \text{ no} \land \text{low} \text{ no} \neq \text{Null})\)
case True
  then obtain
    \(\text{red-case}: (\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{low}) \text{ no} = \)
    \((\text{repa}(\text{node} := \text{repa}(\text{low} \text{ node})) \propto \text{high}) \text{ no} \) and
    hno-nNull: \text{low} \text{ no} \neq \text{Null}
    by simp
from lno-nNull balanced-children have hno-nNull: \text{high} \text{ no} \neq \text{Null}
  by simp
from True ext-rep-null-comp-low ext-rep-null-comp-high

92
have repchildren-eq-no: (repa ∞ low) no = (repa ∞ high) no
  by simp
with repa-cases lno-nNull have repa no = (repa ∞ 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)) ∝ low) no
  by simp
with True repchildren-eq-nln show ?thesis
  by auto
next
  assume share-case-ext:
  ¬ ((repa (node := repa (low node)) ∝ low) no = (repa (node := repa (low node)) ∝ high) no ∧ low no ≠ Null)
from not-Leaf isLeaf-var-nln
have 1 < var node
  by simp
with all-nodes-same-var
have all-nodes-nl-Suc0-l-var: ∀ x ∈ set (prx @ node # sfx). 1 < var x
  using [[simp-depth-limit=1]]
  by auto
with nodes-balanced-ordered
have all-nodes-nl-noLeaf:
  ∀ x ∈ set (prx @ node # sfx). ¬ isLeaf-pt x low high
  apply ¬
  apply rule
  apply (drule-tac x=x in bspec, assumption)
  apply (drule-tac x=x in bspec, assumption)
  apply auto
  done
from nodes-balanced-ordered
have all-nodes-nl-balanced:
  ∀ x ∈ set (prx @ node # sfx). (low x = Null) = (high x = Null)
  apply ¬
  apply rule
  apply (drule-tac x=x in bspec, assumption)
  apply auto
  done
from all-nodes-nl-Suc0-l-var no-in-nodeslist
have Suc0-l-var-no: 1 < var no
  by auto
with isLeaf-var-no have no-nLeaf: ¬ 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

93
have \textit{repchildren-neq-no}: \((\text{repa } \& \text{low}) \text{no} \neq (\text{repa } \& \text{high}) \text{no}\)

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

with \textit{repa-cases}

have \textit{share-case-inv}:

\begin{align*}
\text{repa} \text{no} &= \text{hd } [\text{sn}\leftarrow \text{prx} \cdot \text{repNodes-eq sn no low high repa}] \land \\
(\forall \text{no1} \in \text{set prx}. ((\text{repa } \& \text{high}) \text{no1} = (\text{repa } \& \text{high}) \text{no}) \land \\
(\text{repa } \& \text{low}) \text{no1} = (\text{repa } \& \text{low}) \text{no}) = (\text{repa} \text{no} = \text{repa} \text{no1}))
\end{align*}

by auto

then have \textit{repa-no}:

\begin{align*}
\text{repa} \text{no} &= \text{hd } [\text{sn}\leftarrow \text{prx} \cdot \text{repNodes-eq sn no low high repa}]
\end{align*}

by \((\text{simp})\)

from \textit{Suc0-l-var-no} have \(\forall x \in \text{set } (\text{prx @} \text{node} \# \text{sfx}). I < \text{var no}\)

by auto

from \textit{no-in-take-n} have \([\text{sn}\leftarrow \text{prx} \cdot \text{repNodes-eq sn no low high repa}] \neq []\)

apply \-

apply \((\text{rule filter-not-empty})\)

apply \((\text{auto simp add: repNodes-eq-def})\)

done

then have \textit{repNodes-eq}:

\begin{align*}
(\text{hd } [\text{sn}\leftarrow \text{prx} \cdot \text{repNodes-eq sn no low high repa}]) \text{no low high repa}
\end{align*}

by \((\text{rule hd-filter-prop})\)

with \textit{repa-no}

have \textit{rep-children-eq-no-repa-no}:

\begin{align*}
(\text{repa } \& \text{low}) (\text{repa} \text{no}) = (\text{repa } \& \text{low}) \text{no} \land \\
(\text{repa } \& \text{high}) (\text{repa} \text{no}) = (\text{repa } \& \text{high}) \text{no}
\end{align*}

by \((\text{simp add: repNodes-eq-def})\)

from \textit{mo-notin-nodeslist rep-repa-nc}

have \textit{rep-repa-nc-low-no}:

\begin{align*}
\text{rep (low no)} = \text{repa (low no)}
\end{align*}

apply \-

apply \((\text{erule-tac x}= \text{low no in allE})\)

apply auto

done

have \(\forall x \in \text{set } (\text{prx @} \text{node}). \text{repNodes-eq x no low high (repa(node := repa (low node)))} = \text{repNodes-eq x no low high repa}\)

proof \((\text{rule ballI}, \text{unfold repNodes-eq-def})\)

fix \textit{x}

assume \textit{x-in-take-Sucn}: \(x \in \text{set } (\text{prx @} \text{node})\)

hence \textit{x-in-nodeslist}: \(x \in \text{set } (\text{prx @} \text{node} \# \text{sfx})\)

by auto

with \textit{all-nodes-nl-noLeaf nodes-balanced-ordered}

have \textit{children-nNull-x}:

\begin{align*}
\text{low x} \neq \text{Null } \land \text{high x} \neq \text{Null}
\end{align*}

apply \-

apply \((\text{erule-tac } x=x \text{ in bspec,assumption})\)

apply \((\text{erule-tac } x=x \text{ in bspec,assumption})\)

apply \((\text{auto simp add: isLeaf-pt-def})\)

done

from \textit{x-in-nodeslist nodes-balanced-ordered}

94
have \( \text{low} \ x \notin \text{set (prx @ node # sfx)} \land \text{high} \ x \notin \text{set (prx @ node # sfx)} \)

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 \( ((\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node})) \propto \text{high}) \ x = \\
(\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node})) \propto \text{high}) \ no \land \\
(\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node})) \propto \text{low}) \ x = \\
(\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node})) \propto \text{low}) \ no) = \\
((\text{rep} \propto \text{high}) \ x = (\text{rep} \propto \text{high}) \ no \land \\
(\text{rep} \propto \text{low}) \ x = (\text{rep} \propto \text{low}) \ no) \)

by (simp add: null-comp-def)

qed

then have filter-extrep-rep:
\[
[sn\gets (prx @[node]). \text{repNodes-eq sn no low high} \\
(\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node})))] = \\
[sn\gets (prx @[node]). \text{repNodes-eq sn no low high repa}] \\
\]

by (rule P-eq-list-filter)

from no-in-take-n
have filter-n-notempty: \([sn\gets prx. \text{repNodes-eq sn no low high repa}] \neq []\)
apply (rule filter-not-empty)
apply (simp add: repNodes-eq-def)
done

then have \( \text{hd} \ [sn\gets prx. \text{repNodes-eq sn no low high repa}] = \\
\text{hd} \ [sn\gets prx@[node]. \text{repNodes-eq sn no low high repa}] \)

by auto

with no-noteq-nln filter-extrep-rep repa-no
have ext-repa-no: \((\text{rep}(\text{node}:= \text{rep} (\text{low} \ \text{node}))) \ no = \\
\text{hd} \ [sn\gets prx@[node]. \text{repNodes-eq sn no low high} \ (\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node}))))] \)

by simp

have \((\text{rep}(\text{node} := \text{rep} (\text{low} \ \text{node}))) \ (\text{repa} \ no) = \text{repa} \ no \)

proof \((\text{cases repa} \ no = \text{node})\)

\begin{itemize}
  \item 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 \ (\text{repa} \ no) \neq \text{Null}
  
  apply (simp add: null-comp-def)
  
  apply auto
done

  with nodes-balanced-ordered rno-nln

  have high-rep-no-nNull: high \ (\text{repa} \ no) \neq \text{Null}
  
  apply 
  
  apply (drule-tac \(x=\text{repa} \ no\) in bspec)
  
  apply auto
done
\end{itemize}
with \texttt{low-rep-no-nNull} rno-nln \texttt{rep-children-eq-no-repa-no} \\
\textbf{have} repa (low node) = (repa \propto low) no \land \\
\quad repa (high node) = (repa \propto high) no \\
\quad \text{by (simp add: null-comp-def)} \\
\textbf{with} \texttt{repchildren-eq-nln} \textbf{have} (repa \propto low) no = (repa \propto high) no \\
\quad \text{by simp} \\
\textbf{with} \texttt{repchildren-neq-no} \textbf{show} \ ?thesis \\
\quad \text{by simp} \\
\textbf{next} \\
\textbf{assume} rno-not-nln: repa no \neq node \\
\textbf{from} share-case-inv \textbf{have} repa (repa no) = repa no \\
\quad \text{by auto} \\
\textbf{with} rno-not-nln \textbf{show} \ ?thesis \\
\quad \text{by simp} \\
\textbf{qed} \\
\textbf{with} \texttt{no-noteq-nln} \textbf{have} ext-repa-ext-repa: \\
\quad (repa(node := repa (low node))) \\
\quad ((repa(node := repa (low node))) \propto high) no1 = \\
\quad (repa(node := repa (low node)) \propto high) no \land \\
\quad (repa(node := repa (low node)) \propto low) no1 = \\
\quad (repa(node := repa (low node)) \propto low) no = \\
\quad (repa(node := repa (low node))) no = \\
\quad (repa(node := repa (low node))) no1) \\
\textbf{proof} (\texttt{rule ballI}) \\
\textbf{fix} no1 \\
\textbf{assume} no1-in-take-Sucn: no1 \in \texttt{set (prx@ \langle node \rangle)} \\
\textbf{hence} no1-in-nodeslist: no1 \in \texttt{set (prx @ node \# sfz)} \\
\quad \text{by auto} \\
\textbf{show} ((repa(node := repa (low node)) \propto high) no1 = \\
\quad (repa(node := repa (low node)) \propto high) no \land \\
\quad (repa(node := repa (low node)) \propto low) no1 = \\
\quad (repa(node := repa (low node)) \propto low) no = \\
\quad (repa(node := repa (low node))) no = \\
\quad (repa(node := repa (low node))) no1) \\
\textbf{proof} (\texttt{cases no1 = node}) \\
\textbf{case} True \\
\textbf{show} \ ?thesis \\
\textbf{proof} (\texttt{rule, elim conjE}) \\
\textbf{assume} ext-repa-no-no1: \\
\quad (repa(node := repa (low node))) no = \\
\quad (repa(node := repa (low node))) no1 \\
\textbf{with} True \texttt{no-noteq-nln} \\
\textbf{have} repa-no-repa-low-nln: repa no = repa (low node) \\
\quad \text{by simp} \\
\textbf{from} \texttt{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 lln-notin-nodeslist rep-repa-nc
have rep-repa-low-nln: rep (low node) = repa (low node)
    by auto
from all-nodes-nl-noLeaf nln-balanced-children
have low node ≠ Null
    by (auto simp add: isLeaf-pt-def)
with rep-repa-low-nln lln-notin-nodeslist lno-nNull
    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 repchildren-eq-no1: repa (low no1) = repa (high no1)
    by simp
from not-Leaf True nln-balanced-children
have children-nNull-no1: (low no1) ≠ Null ∧ high no1 ≠ Null
    by (simp add: isLeaf-pt-def)
with repchildren-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
    lln-notin-nodeslist lno-notin-nodeslist True
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
    lln-notin-nodeslist hno-notin-nodeslist True
have rep-high-eq-no-no1: (repa ∝ high) no1 = (repa ∝ high) no

97
by (simp add: null-comp-def)
with rep-low-eq-no-no1 repchildren-eq-no1
have (repa \times low) no = (repa \times high) no
  by simp
with repchildren-neq-no
show (repa(node := repa (low node))) no =
  (repa(node := repa (low node))) no1
  by simp
qed

next
assume no1-neq-nln: no1 \neq node
from no1-in-nodeslist nodes-balanced-ordered
have children-notin-nl-no1:
  low no1 \notin set (prx @ node # sfx) \land high no1 \notin set (prx @ node # sfx)
apply ~
apply (drule-tac x=no1 in bspec,assumption)
by auto
from no1-neq-nln no1-in-take-Sucn
have no1-in-take-n: no1 \in set prx
  by auto
from no1-in-nodeslist all-nodes-nl-noLeaf all-nodes-nl-balanced
have children-nNull-no1: (low no1) \neq Null \land high no1 \neq Null
  by (auto simp add: isLeaf-pt-def)
show ?thesis
proof (rule, elim conjE)
  assume ext-repa-high-no1-no:
    (repa(node := repa (low node)) \times high) no1
    = (repa(node := repa (low node)) \times high) no1
  assume ext-repa-low-no1-no:
    (repa(node := repa (low node)) \times low) no1
    = (repa(node := repa (low node)) \times low) no1
  from children-nNull-no1 hno-nNull ext-repa-high-no1-no
    children-notin-nl-no1
    hno-notin-nodeslist
  have repa-high-no1-no: (repa \times high) no1 = (repa \times high) no1
    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 \times low) no1 = (repa \times low) no1
    by (simp add: null-comp-def)
  from repchildren-neq-no repa-high-no1-no repa-low-no1-no
  have (repa \times low) no1 \neq (repa \times high) no1
    by simp
  from no1-in-take-n share-case-inv repa-high-no1-no repa-low-no1-no
  have repa no = repa no1
    by auto
  with no-noteq-nln no1-neq-nln
  show (repa(node := repa (low node))) no =
(repa(node := repa (low node))) no1
by simp
next
assume (repa(node := repa (low node))) no =
(repa(node := repa (low node))) no1
with no-noteq-nln no1-neq-nln
have repa no = repa no1
by simp
with share-case-inv no1-in-take-n
have repa no = repa no1
by (auto simp add: null-comp-def)
qed
next
from ext-repa-ext-repa ext-repa-no share-case-ext repchildren-eq-nln this
show ?thesis
using [[simp-depth-limit=4]]
by auto
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 not-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
done
then have low no /∈ set prx
by auto
with rep-repa-nc no-nln rep-low-no-nNull
have (repa(node := repa (low node))) no ≠ Null
by simp
moreover

have (if (repa(node := repa (low node)) ∗ low) no =
  (repa(node := repa (low node)) ∗ high) no ∧ low no ≠ Null
then (repa(node := repa (low node))) no =
  (repa(node := repa (low node)) ∗ low) no
else (repa(node := repa (low node))) no =
  hd [sn←prxM[node]. repNodes-eq sn no low high
  (repa(node := repa (low node)))] ∧
  (repa(node := repa (low node)))
  ((repa(node := repa (low node))) ∗ low) no =
  (repa(node := repa (low node))) no ∧
  (∀ no1∈set (prx0[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))) no1))

proof (cases (repa(node := repa (low node)) ∗ low) no =
  (repa(node := repa (low node)) ∗ high) no ∧ low no ≠ Null)

  case True
note red-case=this

  with children-nNull-no lno-notin-nl hno-notin-nl
  have (repa ∗ low) no = (repa ∗ high) no
  by (auto simp add: null-comp-def)

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

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

  from no-nln children-nNull-no
  have repa (low node) = (repa ∗ 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: ¬ ((repa(node := repa (low node)) ∗ low) no =
  (repa(node := repa (low node)) ∗ high) no ∧ low no ≠ Null)

with low-no-nNull have (repa(node := repa (low node)) ∗ low) no ≠
  (repa(node := repa (low node)) ∗ high) no
  by simp

with children-nNull-no lno-notin-nl hno-notin-nl
have (repa ∗ low) no ≠ (repa ∗ high) no
by (auto simp add: null-comp-def)
with children-nNull-no have repa (low no) ≠ repa (high no)
by (simp add: null-comp-def)
with repchildren-eq-nln no-nln show ?thesis
by simp
qed
ultimately show ?thesis
using repchildren-eq-nln
apply -
apply (simp only:)
apply (simp (no-asm))
done
qed
qed
from nodes-unmodif nodes-modif
show ?thesis
by iprover
qed
next
fix var low high rep nodeslist repa next node prx sfx repb
assume ns: List nodeslist next (prx @ node # sfx)
assume sfx: List (next node) next sfx
assume nodes-balanced-ordered: ∀ 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) ∧
  (rep ∝ low) no ∉ set (prx @ node # sfx)
assume all-nodes-same-var: ∀ no1∈set (prx @ node # sfx).
  ∀ no2∈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 (rep ∝ low) no = (rep ∝ high) no ∧ low no ≠ Null
   then repa no = (rep ∝ low) no
   else repa no = hd [sn←prx . repNodes-eq sn no low high repa] ∧
   repa (repa no) = repa no ∧
  (∀ no1∈set prx.
   ((rep ∝ high) no1 = (rep ∝ high) no ∧
   (rep ∝ low) no1 = (rep ∝ low) no) =
   (repa no = repa no1))))
assume share-cond:
  ¬ (∼ isLeaf-pt node low high ∧ repa (low node) = repa (high node))
assume repb-node:
  repb node = hd [sn←prx @ node # sfx . repNodes-eq sn node low high repa]
assume repa-repb-nc: ∀ pt. pt ≠ node → repa pt = repb pt
assume var-repb-node: var (repb node) = var node
show $(\forall \textno. \textno \notin \text{set} (\textprx @ \text{node})) \rightarrow \textrep \textno = \textrepb \textno) \land
(\forall \textno \in \text{set} (\textprx @ \text{node})).$

\text{repb} \textno \neq \text{Null} \land
(if \ (\text{repb} \propto \text{low}) \textno = (\text{repb} \propto \text{high}) \textno \land \textno \neq \text{Null}
then \textrepb \textno = (\text{repb} \propto \text{low}) \textno
else \textrepb \textno =
\text{hd} [\textsn \leftarrow \textprx @ \text{node}]. \text{repNodes-eq \textsn \textno \text{low} \text{high} \textrepb] \land
\text{repb} (\text{repb} \textno) = \text{repb} \textno \land
(\forall \textno1 \in \text{set} (\textprx @ \text{node})).
((\text{repb} \propto \text{high}) \textno1 = (\text{repb} \propto \text{high}) \textno \land
(\text{repb} \propto \text{low}) \textno1 = (\text{repb} \propto \text{low}) \textno) =
(\text{repb} \textno = (\text{repb} \textno1)))

proof –

have \textrepb-nc: $(\forall \textno. \textno \notin \text{set} (\textprx @ \text{node})) \rightarrow \textrep \textno = \textrepb \textno$
proof (intro allI impI)
fix \textno
assume \textno-notin-Sucn: \textno \notin \text{set} (\textprx @ \text{node})
with \textrepb-nc
have \textrepb-nc-Sucn: \textrep \textno = \textrepa \textno
by auto
from \textno-notin-Sucn have \textno \neq \text{node}
by auto
with \textrepa-\textrepb-nc have \textrepa \textno = \textrepb \textno
by auto
with \textrepb-nc-Sucn show \textrep \textno = \textrepb \textno
by simp
qed
moreover
have \textrepb-no-share-def:
$(\forall \textno \in \text{set} (\textprx @ \text{node})).
\neg ((\text{repb} \propto \text{low}) \textno = (\text{repb} \propto \text{high}) \textno \land \textno \neq \text{Null}) \rightarrow
\text{repb} \textno = \text{hd} [\textsn \leftarrow (\textprx @ \text{node}]. \text{repNodes-eq \textsn \textno \text{low} \text{high} \textrepb)]$
proof (intro ballI impI)
fix \textno
assume \textno-in-Sucn: \textno \in \text{set} (\textprx @ \text{node})
assume \textshare-prop: $\neg ((\text{repb} \propto \text{low}) \textno = (\text{repb} \propto \text{high}) \textno \land \textno \neq \text{Null})$
from \textshare-prop have \textshare-or:
$(\text{repb} \propto \text{low}) \textno \neq (\text{repb} \propto \text{high}) \textno \lor \textno = \text{Null}$
using [[simp-depth-limit=2]]
by simp
from \textno-in-Sucn have \textno-in-nl: \textno \in \text{set} (\textprx @ \text{node} \# \textsfz)
by auto
from \textnodes-balanced-ordered [rule-format, OF this] obtain
\textno-nNull: \textno \neq \text{Null} and
\textbalanced-no: (\textno = \text{Null}) = (\text{high no} = \text{Null}) and
\textno-notin-nl: \textno \notin \text{set} (\textprx @ \text{node} \# \textsfz) and
\texthno-notin-nl: \texthigh no \notin \text{set} (\textprx @ \text{node} \# \textsfz) and
\textisLeaf-var-no: \textisLeaf-\textpt no \textlow high = (\str \textno \leq 1)
by auto

102
have nodes-notin-nl-neq-nln: \( \forall p. p \notin \text{set} \ (\text{prx} \ @ \ \text{node} \ # \ sfx) \rightarrow p \neq \text{node} \)
by auto

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

proof (cases no = node)
case False
note no-notin-nl=this
with no-in-take-Sucn have no-in-take-n: no \in \text{set} \ prx
by auto
from False repa-repb-nc have repb-repa-no: repb no = repa no
by auto
with while-inv [rule-format, OF no-in-take-n] no-in-take-n obtain repa-no-nNull: repa no \neq \text{Null} \ and
while-share-red-exp:
(if (repa \propto \text{low}) no = (repa \propto \text{high}) no \wedge \text{low} no \neq \text{Null}
then repa no = (repa \propto \text{low}) no
else repa no = hd \([sn←\text{prx}. \text{repNodes-eq} \ sn \ \text{no} \ \text{low} \ \text{high} \ repa]\) \wedge
repa (repa no) = repa no \wedge
(\forall no1 \in \text{set} \ prx. ((repa \propto \text{high}) no1 = (repa \propto \text{high}) no \wedge
(repa \propto \text{low}) no1 = (repa \propto \text{low}) no) = (repa no = \text{rep no1}))
using [[simp-depth-limit = 2]]
by auto
from no-in-take-n have filter-take-n-notempty: \([sn←\text{prx}. \text{repNodes-eq} \ sn \ \text{no} \ \text{low} \ \text{high} \ repa]\) \neq []
apply –
apply (rule filter-not-empty)
apply (auto simp add: repNodes-eq-def)
done
then have hd-term-n-Sucn:
hd \([sn←\text{prx}. \text{repNodes-eq} \ sn \ \text{no} \ \text{low} \ \text{high} \ repa]\)
= hd \([sn←\text{prx}@\text{node}. \text{repNodes-eq} \ sn \ \text{no} \ \text{low} \ \text{high} \ repa]\)
by auto
thus ?thesis

proof (cases low no = \text{Null})
case True
note lno-Null=this
with balanced-no have hno-Null: hno-Null: high no = \text{Null}
by simp
from hno-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 \([sn←\text{prx}. \text{repNodes-eq} \ sn \ \text{no} \ \text{low} \ \text{high} \ repa]\) \wedge
repa (repa no) = repa no \wedge
(\forall no1 \in \text{set} \ prx. ((repa \propto \text{high}) no1 = (repa \propto \text{high}) no
\wedge (repa \propto \text{low}) no1 = (repa \propto \text{low}) no) = (repa no = \text{rep no1}))
by auto
have all-nodes-in-nl-Leafs:
\( \forall x \in \text{set} \ (\text{prx} \ @ \ \text{node} \ # \ sfx). \text{isLeaf-pt} \ x \ \text{low} \ \text{high} \)

103
proof (intro ballI)
fix x
assume x-in-nodeslist: x ∈ set (prx @ node # sfx)
from isLeaf-no isLeaf-var-no have var no ≤ 1
  by simp
with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
have var x ≤ 1
  by simp
with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
show isLeaf-pt x low high
  by (auto simp add: isLeaf-pt-def)
qed
have ∀ x ∈ set (prx@[node]). repNodes-eq x no low high repb
  = repNodes-eq x no low high repa
proof (rule ballI)
fix x
assume x-in-take-Sucn: x ∈ set (prx@[node])
hence x-in-nodeslist: x ∈ set (prx @ node # sfx)
  by auto
with all-nodes-in-nl-Leufs have isLeaf-pt x low high
  by auto
with isLeaf-no repa-repb-nc show repNodes-eq x no low high repb
  = repNodes-eq x no low high repa
  by (simp add: repNodes-eq-def null-comp-def isLeaf-pt-def)
qed
then have [sn←(prx@[node]). repNodes-eq sn no low high repa]
  = [sn←(prx@[node]). repNodes-eq sn no low high repb]
  apply −
  apply (rule P-eq-list-filter)
  apply simp
  done
with hd-term-n-Sucn while-low-Null repb-repa-no show ?thesis
  by auto
next
assume bno-nNull: low no ≠ Null
with balanced-no have bno-nNull: high no ≠ Null
  by simp
with bno-nNull have no-nLeaf: ¬ isLeaf-pt no low high
  by (simp add: isLeaf-pt-def)
with isLeaf-var-no have Sucn-s-varno: 1 < var no
  by auto
with no-in-nl all-nodes-same-var
have all-nodes-nl-var: ∀ x ∈ set (prx @ node # sfx). 1 < var x
  apply −
  apply (rule ballI)
  apply (drule-tac x=no in bspec.assumption)
  apply (drule-tac x=z in bspec.assumption)
  apply auto
  done

104
with nodes-balanced-ordered
have all-nodes-nl-nLeaf:
\( \forall x \in \text{set} (\text{prx} @ \text{node} \# \text{sfx}). \neg \text{isLeaf-pt} x \text{ low high} \)
apply –
apply (rule ballU)
apply (drule-tac \( x=x \) in bspec, assumption)
apply auto
done
from lno-nNull share-or
have repbchildren-eq-no: (repb \( \propto \) low) \( \neq \) (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 repbchildren-eq-no: (repa \( \propto \) low) \( \neq \) (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 \([\text{sn} ← \text{prx}]. \text{repNodes-eq sn no low high repa}]\)
by auto
with no-notin-nl repa-repb-nc
have repb no = hd \([\text{sn} ← \text{prx}]. \text{repNodes-eq sn no low high repa}]\)
by simp
with hd-term-n-Sucn
have repb-no-hd-term-repa:
repa no = hd \([\text{sn} ← \text{prx}@\text{node}]. \text{repNodes-eq sn no low high repa}]\)
by simp
have \( \forall x \in \text{set} (\text{prx}@\text{node}]\).
repNodes-eq x no low high repa = repNodes-eq x no low high repb
proof (intro ballU)
fix x
assume x-in-take-Sucn: \( x \in \text{set} (\text{prx}@\text{node}]\)
hence x-in-nodeslist: \( x \in \text{set} (\text{prx} \@ \text{node} \# \text{sfx}]\)
by auto
with all-nodes-nl-nLeaf have x-nLeaf: \( \neg \text{isLeaf-pt} x \text{ low high} \)
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-null
hno-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 \([\text{sn} ← (\text{prx}@\text{node}]]. \text{repNodes-eq sn no low high repa}] =
[\text{sn} ← (\text{prx}@\text{node}]]. \text{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 ≠ sfx). repNodes-eq sn node low high repa]
by simp
show ?thesis
proof (cases isLeaf-pt no low high)
case True
note isLeaf-no=this
have ∀x ∈ set (prx @ node ≠ sfx). repNodes-eq x no low high repb = repNodes-eq x no low high repa
proof (rule ballI)
fix x
assume x-in-nodeslist: x ∈ set (prx @ node ≠ sfx)
have all-nodes-in-nd-Leafs:
∀x ∈ set (prx @ node ≠ sfx). isLeaf-pt x low high
proof (intro ballI)
fix x
assume x-in-nodeslist: x ∈ set (prx @ node ≠ sfx)
from isLeaf-no isLeaf-var-no have var no ≤ 1
by simp
with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
have var x ≤ 1
by simp
with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
show isLeaf-pt x low high
by (auto simp add: isLeaf-pt-def)
qed
with x-in-nodeslist have isLeaf-pt x low high
by auto
with isLeaf-no repb-repb-nc
show repNodes-eq x no low high repb = repNodes-eq x no low high repa
by (simp add: repNodes-eq-def null-comp-def isLeaf-pt-def)
qed
with repb-no-def no-nln have repb-no-whole-nl: repb no = hd [sn← (prx @ node ≠ sfx). repNodes-eq sn node low high repb]
apply −
apply (subgoal-tac
[sn← (prx @ node ≠ sfx). repNodes-eq sn node low high repa]
= [sn← (prx @ node ≠ sfx) . repNodes-eq sn node low high repb])
apply simp
apply (rule P-eq-list-filter)
apply auto
done
from no-in-take-Suc n 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
  apply (unfold isLeaf-pt-def)
  apply fastforce
  done
qed
with x-in-nodeslist have x-nLeaf: ¬ isLeaf-pt x low high
  by auto

107
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 hno-notin-nl kno-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
have repb-no-red-def: (∀ no∈ set (prx@[node]).(repb ∞ low) no = (repb ∞ high) no)
  ∧ low no ≠ Null → repb no = (repb ∞ low) no)
proof (intro ballI impI)
  fix no
  assume no-in-take-Sucn: no ∈ set (prx@[node])
  assume red-cond-no: (repb ∞ low) no = (repb ∞ high) no ∧ low no ≠ Null
from no-in-take-Sucn have no-in-nl: no ∈ set (prx@node#sfx)
  by auto
from nodes-balanced-ordered [rule-format, OF this]obtain
no-nNull: \( \text{no} \neq \text{Null} \) and
balanced-no: \((\text{low no} = \text{Null}) = (\text{high no} = \text{Null}) \) and
lno-notin-nl: \( \text{low no} \notin \text{set} (\text{prx@node#sfz}) \) and
hno-notin-nl: \( \text{high no} \notin \text{set} (\text{prx@node#sfz}) \) and
isLeaf-var-no: isLeaf-pt no low high = (\( \text{var no} \leq 1 \))

by auto
have nodes-notin-nl-neq-nln: \( \forall p. \ p \notin \text{set} (\text{prx@node#sfz}) \rightarrow p \neq \text{node} \)
by auto
show repb no = (repb \( \propto \) low) no
proof (cases no = node)
case False
note no-notin-nl=this
with no-in-take-Sucn have no-in-take-n: \( \text{no} \in \text{set prx} \)
by auto
from False repa-repb-nc have repb-repa-no: repb no = repa no
by auto
with while-inv [rule-format, \( \text{OF no-in-take-n} \)] obtain
repa-no-nNull: repa no \( \neq \) Null and
while-share-red-exp:
(if (repa \( \propto \) low) no = (repa \( \propto \) low) no \( \land \) low no \( \neq \) Null
then repa no = (repa \( \propto \) low) no
else repa no = hd [sne–prx. repNodes-eq sn no low high repa] \( \land \)
repa (repa no) = repa no \( \land \)
(\( \forall \text{no1} \in \text{set prx}. (\text{repa} \( \propto \) high) \text{no1} = (\text{repa} \( \propto \) high) \text{no} \( \land \)
(repa \( \propto \) low) no1 = (repa \( \propto \) 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 \( \propto \) 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 \( \propto \) low) no = (repa \( \propto \) 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 \( \propto \) 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 \( \lor \) repa (low no) \( \neq \) repa (high no)
by simp
show ?thesis
proof (cases isLeaf-pt no low high)
case True
with red-cond-no show ?thesis

109
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 lno-notin-nl hno-notin-nl nodes-notin-nl-neq-nln
balanced-no repa-repb-nc
have repb (low no) ≠ repb (high no)
using [[simp-depth-limit=2]]
by (auto simp add: isLeaf-pt-def)
with no-nLeaf balanced-no
have (repb ∝ low no) ≠ (repb ∝ high no)
by simp add: null-comp-def isLeaf-pt-def
with red-cond-no
show ?thesis
by simp
qed
qed

have while-while: (∀ no∈set (prx@node)).
repb no ≠ Null ∧
(if (repb ∝ low no) no = (repb ∝ high no) ∧ low no ≠ Null
then repb no = (repb ∝ low no)
else repb no = hd [sn←(prx@node)]. repNodes-eq sn no low high repb] ∧
repb (repb no) = repb no ∧
(∀ no1∈set ((prx@node))). ((repb ∝ high no) no1 = (repb ∝ high no)
∧ (repb ∝ low) no1 = (repb ∝ low) no) = (repb 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: \( \forall p. p \notin \text{set } (\text{prx}@\text{node#sfx}) \rightarrow p \neq \text{node} \)
  by auto
show \(?P \land \ ?Q\) no
proof (cases \(\text{no = node}\))
  case False
  note notin-nl = this
  with no-in-take-Sucn
  have no-in-take-n: \(\text{no} \in \text{set prx}\)
    by auto
  from False repa-repb-nc have repb-repa-no: \(\text{repb no} = \text{repa no}\)
    by auto
  with while-inv [rule-format, OF no-in-take-n] obtain
  repa-no-nNull: \(\text{repa no} \neq \text{Null and}\)
  while-share-red-exp:
    (if (\(\text{repa} \times \text{low}\)) no = (\(\text{repa} \times \text{high}\)) no \& \text{low no} \neq \text{Null}
      then repa no = (\(\text{repa} \times \text{low}\)) no
      else repa no = \text{hd }[\text{sn}\leftarrow \text{prx. repNodes-eq sn no low high repa}] \land
      repa (repa no) = repa no \land
      (\(\forall \text{no1} \in \text{set prx. (\(\text{repa} \times \text{high}\)) no1} = (\(\text{repa} \times \text{high}\)) no \land
      (\(\text{repa} \times \text{low}\)) no1 = (\(\text{repa} \times \text{low}\)) no) = (\(\text{repa no} = \text{repa no1}\)))
    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 (\(\text{repb} \times \text{low}\)) no = (\(\text{repb} \times \text{high}\)) no \& \text{low no} \neq \text{Null})
  case True
  with no-in-take-Sucn repb-no-red-def show \(?\text{thesis}\)
    by auto
next
  assume share-case-repb:
    \(\neg ((\(\text{repa} \times \text{low}\)) no = (\(\text{repa} \times \text{high}\)) no \& \text{low no} \neq \text{Null})\)
  with repb-no-share-def no-in-take-Sucn
  have repb-no-def: \(\text{repb no} = \text{hd }[\text{sn}\leftarrow (\text{prx}[\text{node}])]. \text{repNodes-eq sn no low high repa}]\)
    by auto
  with share-case-repb
  have (\(\text{repb} \times \text{low}\)) no \neq (\(\text{repb} \times \text{high}\)) no \lor \text{low no} = \text{Null}
    using [[simp-depth-limit=2]]
    by simp
  thus \(?\text{thesis}\)
proof (cases \(\text{low no} = \text{Null}\))
  case True
  note lno-Null = this
  with balanced-no have hno-Null: \(\text{high no} = \text{Null}\)
    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 \([sn \leftarrow \text{prx}. \text{repNodes-eq sn no low high repa}] \) \land
  repa (repa no) = repa no \land
  (\forall no1 \in \text{set prx}. ((\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))
by auto
from no-in-take-n
have \([sn \leftarrow \text{prx}. \text{repNodes-eq sn no low high repa}] \neq []\)
  apply
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
done
then have hd-term-n-Suc-
  \[\text{hd} [\text{sn} \leftarrow \text{prx}. \text{repNodes-eq sn no low high repa}] = \text{hd} [\text{sn} \leftarrow (\text{prx}[\text{node}]) . \text{repNodes-eq sn no low high repa}]\]
apply
apply (rule hd-filter-app [symmetric])
apply auto
done
have all-nodes-in-
proof (intro ballI)
fix x
assume x-in-nodeslist: \(x \in \text{set (prx[node]#sfx)}\)
from isLeaf-no isLeaf-var-no have var no \leq 1
by simp
with all-nodes-same-var [rule-format, OF x-in-nodeslist no-in-nl]
have var x \leq 1
by simp
with nodes-balanced-ordered [rule-format, OF x-in-nodeslist]
show isLeaf-pt x low high
by (auto simp add: isLeaf-pt-def)
qed
from no-in-take-Suc
have
filter-Sucn-no-notempty:
  \([\text{sn} \leftarrow (\text{prx}[\text{node}]). \text{repNodes-eq sn no low high repb}] \neq []\]
apply
apply (rule filter-not-empty)
apply (auto simp add: repNodes-eq-def)
done
then have hd-term-in-
then have hd-term-in-
by auto
112
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 repb-repb-no-repb: repb (repb no) = repb no
  by auto
proof (cases repb no = node)
case False
  with repb-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 repb =
  hd [sn← (prx@[node]#sfx)]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)
  low high repb
  by (simp add: isLeaf-pt-def repNodes-eq-def null-comp-def)
from hd-term-in-take-Sucn
have [sn←(prx@[node])]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)
  low high repb ≠ []
  apply –
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then have hd [sn←(prx@[node])]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)
  low high repb =
  hd [sn← (prx@[node]#sfx)]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)
  low high repb
  apply –
  apply (rule hd-filter-app [symmetric])
  apply auto
  done
then have hd-term-nodeslist-Sucn:
  hd [sn←(prx@[node]#sfx)]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)
  low high repb =
  hd [sn← (prx@[node])]. repNodes-eq sn
  (hd [sn← (prx@[node])]. repNodes-eq sn no low high repb)

113
low high repb

by simp
from no-in-take-Sucn filter-Sucn-no-notempty
have filter-filter: hd [sn←(prx[node])]. repNodes-eq sn
  (hd [sn←(prx[node])]. repNodes-eq sn no low high repb)
  low high repb =
  hd [sn←(prx[node])]. repNodes-eq sn no low high repb
apply –
apply (rule filter-hd-P-rep-indep)
apply (auto simp add: repNodes-eq-def)
done
from repb-no-def repb-no-nln repb-node
have repb (repb no) = hd [sn←(prx[node]). repNodes-eq sn
  (hd [sn←(prx[node]). repNodes-eq sn no low high repb])
  low high repb]
by simp
with nested-hd-repa-repb
have repb (repb no) = hd [sn←(prx[node]). repNodes-eq sn
  (hd [sn←(prx[node]). repNodes-eq sn no low high repb])
  low high repb]
by simp
with hd-term-nodeslist-Sucn
have repb (repb no) = hd [sn←(prx[node]). repNodes-eq sn
  (hd [sn←(prx[node]). repNodes-eq sn no low high repb])
  low high repb]
by simp
with filter-filter
have repb (repb no) = hd [sn←(prx[node]). repNodes-eq sn
  repNodes-eq sn no low high repb]
by simp
with repb-no-def show thesis
by simp
qed
have two-nodes-repb: (∀ no1∈set (prx[node]).
  (repa × high) no1 = (repa × high) no
  ∧ (repa × low) no1 = (repa × low) no)
  = (repa no = repa no1))
proof (intro ballI)
fix no1
assume no1-in-take-Sucn: no1 ∈ set (prx[node])
then have no1 ∈ set (prx[node]#sfx) by auto
with all-nodes-in-nl-Leafs
have isLeaf-no1: isLeaf-pt no1 low high
by auto
with isLeaf-no
have repachildren-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 isLeaf-no1 isLeaf-no
have repachildren-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 ((repb ∞ high) no1 = (repb ∞ high) no ∧
  (repb ∞ low) no1 = (repb ∞ low) no) = (repb no = repb no1)
proof (cases no1 = node)
  case False
  with no1-in-take-Sucn have no1 ∈ set prx
    by auto
  with while-low-same-rep repachildren-eq-no-no1
  have repa no = repa no1
    by auto
  with repb-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]#sfz)
    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 repbchildren-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 repbchildren-eq-no1-no show repNodes-eq x no low high repb
      = repNodes-eq x no1 low high repb
      by (simp add: repNodes-eq-def)
  qed
then have [sn←(prx[node]). repNodes-eq sn no low high repb]
  = [sn←(prx[node]). repNodes-eq sn no1 low high repb]
  by (rule P-eq-list-filter)
with repb-no-def repb-no1 have repb-no-no1: repb no = repb no1
by simp
with repbchildren-eq-no1-no show ?thesis
by simp
qed
qed
with repb-repb-no repb-no-share-def no-in-take-Sucn share-case-repb
show ?thesis
using [[simp-depth-limit=4]]
by auto

next
assume lno-nNull: low no ≠ Null
with share-case-repb
have repbchildren-neq-no: (repb ∞ low) no ≠ (repb ∞ high) no
by auto
from balanced-no lno-nNull
have hno-nNull: high no ≠ 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 ∞ low) no ≠ (repa ∞ 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
∧ (∀ no1∈ set prx. ((repa ∞ high) no1 = (repa ∞ high) no
∧ (repa ∞ low) no1 = (repa ∞ low) no) = (repa no = repa no1))
by auto
from lno-nNull hno-nNull
have no-nLeaf: ¬ isLeaf-pt no low high
by (simp add: isLeaf-pt-def)
have all-nodes-in-nl-nLeafs:
∀ x ∈ set (prx@node#sfx). ¬ isLeaf-pt x low high
proof (intro ballI)
fix x
assume x-in-nodeslist: x ∈ set (prx@node#sfx)
from 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 ¬ 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 (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 repb-no-def have repNodes-eq (repb no) no low high repb
  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 ∧
  (repb ∝ low) no1 = (repb ∝ low) no) = (repb no = repb no1))
proof (intro ballI)
  fix no1
assume no1-in-take-Sucn: no1 ∈ set (prx@[node])
hence no1-in-nl: no1 ∈ set (prx@node#sfx)
  by auto
from all-nodes-in-nl-nLeafs no1-in-nl
have no1-nLeaf: ¬ isLeaf-pt no1 low high
by auto

from nodes-balanced-ordered [rule-format, OF no1-in-nl]
have no1-props: (low no1 = Null) = (high no1 = Null)
∧ low no1 ∉ set (prx@node#sfx) ∧ high no1 ∉ set (prx@node#sfx)
by auto

show ((repb × high) no1 = (repb × high) no
∧ (repb × low) no1 = (repb × low) no) = (repb no = repb no1)

proof (cases no1 = node)
case False
note no1-neq-nln=this
with no1-in-take-Sucn
have no1-in-take-n: no1 ∈ set prx
by auto

with repb-while-inv have ((repb × high) no1 = (repb × high) no
∧ (repb × low) no1 = (repb × low) no) = (repb no = repb no1)
by fastforce

with no1-props no1-nLeaf balanced-no hno-notin-nl
hno-notin-ml nodes-notin-ml-neq-nln no-notin-nl
no1-neq-nln repa-repb-nc

show ?thesis

using [[simp-depth-limit=1]]
by (auto simp add: null-comp-def isLeaf-pt-def)

next
assume no1-nln: no1 = node
show ?thesis

proof
assume repbchildren-eq-no1-no:
(repb × high) no1 = (repb × high) no
∧ (repb × low) no1 = (repb × low) no
with repbchildren-neq-no

have (repb × high) no1 ≠ (repb × low) no1
by auto

with repb-no-share-def no1-in-take-Sucn
have repb-no1-def: repb no1 = hd [sn←(prx@[node])].
repnodes-eq sn no1 low high repb
by auto

have filter-no1-eq-filter-no: [sn←(prx@[node])].
repnodes-eq sn no1 low high repb =
[sn←(prx@[node])]. repnodes-eq sn no low high repb

proof ~

have ∀ 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-no1-def repb-no-def show repb no = repb no1
by simp

next
assume repb-na-no1-eq: repb no = repb no1
from no1-nln repb-node repb-no-def have repb-no1-def:
    repb no1 =
    hd [sn←(prx@node#sfz). 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#sfz). repNodes-eq sn no low high repb] =
    hd [sn←(prx@node#sfz). repNodes-eq sn no1 low high repa]
by simp
from filter-take-Sucn-not-empty have hd-Sucn-hd-whole-list:
    hd [sn←(prx@node#sfz). repNodes-eq sn no low high repb] =
    hd [sn←(prx@node#sfz). repNodes-eq sn no low high repa]
apply −
apply (rule hd-filter-app [symmetric])
apply auto
done
then have hd-Sucn-hd-whole-list:
    hd [sn←(prx@node#sfz). repNodes-eq sn no low high repb] =
    hd [sn←(prx@node#sfz). repNodes-eq sn no low high repa]
by simp
have hd-nl-repb-repa:
    [sn← (prx@node#sfz). repNodes-eq sn no low high repb] =
    [sn← (prx@node#sfz). repNodes-eq sn no low high repa]
proof −
have ∀x ∈ set (prx@node#sfz).
    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#sfz)
from all-nodes-in-nl-nLeafs x-in-nl 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#sfz) ∧ high x /∈ set (prx@node#sfz)
    by auto

119
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#sfz). repNodes-eq sn no low high repa] =
  hd [sn←(prx@node#sfz). repNodes-eq sn no1 low high repa]
by simp
from no-in-nl have filter-no-not-empty:
  [sn←(prx@node#sfz). 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#sfz). 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#sfz). 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#sfz). 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 ∞ high) no1 =
  (repa ∞ high) no ∧ (repa ∞ low) no1 = (repa ∞ low) no
  by (simp add: repNodes-eq-def)
with hno-nNull no1-props no1-nLeaf hno-nNull hno-notin-nl hno-notin-nl nodes-notin-nl-neq-nln repa-repb-nc
show (repa ∞ high) no1 =
\[ \text{(repb \propto high) no} \land \text{(repb \propto low) no1} = \text{(repb \propto low) no} \]

using \([\text{simp-depth-limit=1]}\]
by (auto simp add: isLeaf-plt-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 \([\text{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 \([\text{sn} \leftarrow (prx@node#sfx)]. \text{repNodes-eq sn no low high repa}]\)
by simp
from no-nln have no \in set (prx@node#sfx)
by auto
then have filter-nl-repa-not-empty:
  \([\text{sn} \leftarrow (prx@node#sfx)]. \text{repNodes-eq sn no low high repa}] \neq []\)
apply –
apply (rule filter-not-empty)
apply (auto simp add: repNodes-eq-def)
done

then have hd-filter-nl-in-nl:
  \(\text{hd} [\text{sn} \leftarrow (prx@node#sfx)]. \text{repNodes-eq sn no low high repa}] \in \text{set (prx@node#sfx)}\)
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-plt no high \lor repa (low no) \neq repa (high no)
by auto
have share-reduce-if: (if (repb \propto low) no = (repb \propto high) no \land low no \neq
Null
  then repb no = (repb \propto low) no
  else repb no = hd \([\text{sn} \leftarrow (prx@\text{node}]. \text{repNodes-eq sn no low high repb}]\)
\land
repb (repb no) = repb no
\land \(\forall no1 \in \text{set (prx}@\text{node}]. ((\text{repb \propto high} no1 = (\text{repb \propto high} no
\land (\text{repb \propto low} no1 = (\text{repb \propto low} no) = (\text{repb no} = \text{repb no1})))\)
proof (cases isLeaf-plt no low high)
case True

121
note isLeaf-no=this
then have lno-Null: low no = Null by (simp add: isLeaf-pt-def)
from isLeaf-no in-take-Sucn repb-no-share-def
have repb-no-repb-def: repb no = hd [sn←(prx[node]). repNodes-eq sn no low high repb]
  by (auto simp add: isLeaf-pt-def)
from isLeaf-no nodes-balanced-ordered [rule-format, OF no-in-nl]
have var-no: var no ≤ 1
  by auto
have all-nodes-nl-var-l-1: ∀ x ∈ set (prx[node#sfx]). var x ≤ 1
proof (intro ballI)
  fix x
  assume x-in-nl: x ∈ set (prx[node#sfx])
  with all-nodes-nl-var-l-1 have var x ≤ 1
    by auto
qed
have all-nodes-nl-Leafs: ∀ x ∈ set (prx[node#sfx]). isLeaf-pt x low high
proof (intro ballI)
  fix x
  assume x-in-nl: x ∈ set (prx[node#sfx])
  with all-nodes-nl-var-l-1 have var x ≤ 1
    by auto
  with nodes-balanced-ordered [rule-format, OF x-in-nl]
  show isLeaf-pt x low high
    by auto
qed
have repb-repb-no: repb (repb no) = repb no
proof –
  from repb-no-share-def no-in-take-Sucn lno-Null
  have repb-no-def: repb no =
    hd [sn←(prx[node]). repNodes-eq sn no low high repb]
    by auto
  with hd-filter-Sucn-in-Sucn
  have repb-no-in-take-Sucn: repb no ∈ set (prx[node])
    by simp
  hence repb-no-in-nil: repb no ∈ 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) =
    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: 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))
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 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 simp add: isLeaf-pt-def)
  show ((repb ∝ high) no1 = (repb ∝ high) no
  ∧ (repb ∝ low) no1 = (repb ∝ low) no) = (repb no = repb no1)
  proof
    assume repbchildren-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 repbchildren-eq-no1-no
        show repNodes-eq x no1 low high repb = repNodes-eq x no low high repb
      proof
        by (simp add: repNodes-eq-def)
      qed
    qed
  then show ?thesis
  by (rule P-eq-list-filter)
  qed
  with repb-no1-def repb-no-repb-def
  show repb no = repb no1
  by simp
next
assume repb-no-no1: repb no = repb no1
with isLeaf-no isLeaf-no1
show (repb ∝ high) no1 = (repb ∝ high) no
∧ (repb ∝ low) no1 = (repb ∝ low) no

123
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: 1 < var no
by auto
have all-nodes-nl-var-l-1: \( \forall x \in \text{set} \ (\text{prx}@\text{node}#\text{sfx}) \). 1 < var x
proof (intro ballI)
  fix x
  assume x-in-nl: \( x \in \text{set} \ (\text{prx}@\text{node}#\text{sfx}) \)
  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: \( \forall x \in \text{set} \ (\text{prx}@\text{node}#\text{sfx}) \). ¬ isLeaf-pt x low high
proof (intro ballI)
  fix x
  assume x-in-nl: \( x \in \text{set} \ (\text{prx}@\text{node}#\text{sfx}) \)
  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 repachildren-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)
  simp
  done
with repbchildren-neq-no
have repbchildren-neq-repb-no: (repb ∞ low) (repb no) ≠ (repb ∞ high)
  (repb no)
    by (simp add: repNodes-eq-def)
from filter-take-Sucn-not-empty
have repb no ∈ set (prx@[node])
  apply (simp only: repb-no-def )
  apply (rule hd-filter-in-list)
  simp
  done
with repbchildren-neq-repb-no repb-no-share-def
have repb-repb-no-def:
  repb (repb no) = hd [sn←(prx@[node]) . repNodes-eq sn (repb no) low high repb]
  by auto
from filter-take-Sucn-not-empty show ?thesis
  apply (simp only: repb-repb-no-def )
  apply (simp only: repb-no-def)
  apply (rule filter-hd-P-rep-indep)
  simp
  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#sfx)
  by auto
from filter-take-Sucn-not-empty
have repNodes-repb-no: repNodes-eq (repb no) no low high repb
  apply (simp only: repb-no-def)
  apply (rule hd-filter-prop)
  apply simp
  done
show (repb ∞ high) no1 = (repb ∞ high) no
  ∧ (repb ∞ low) no1 = (repb ∞ low) no
proof (cases (repb ∞ low) no1 = (repb ∞ high) no1)
  case True
  note red-cond=this
from no1-in-nodeslist all-nodes-nl-nLeafs
have no1-nLeaf: ¬ isLeaf-pt no1 low high
  by auto
from nodes-balanced-ordered [rule-format, OF no1-in-nodeslist]
have no1-props: (low no1 ∉ set (prx@node#sfx))
  ∧ (high no1 ∉ set (prx@node#sfx)) ∧ (low no1 = Null) = (high no1 = Null)
  ∧ ((rep ∖ low) no1 ∉ set (prx@node#sfx))
  by auto
with red-cond no1-nLeaf no1-in-take-Sucn repb-no-red-def
have repb-no1-def: repb no1 = (repb ∖ low) no1
  by (auto simp add: isLeaf-pt-def)
with no1-nLeaf no1-props have repb no1 = repb (low no1)
  by (simp add: null-comp-def isLeaf-pt-def)
from no1-props no1-nLeaf have rep (low no1) ∉ set (prx@node#sfx)
  by (auto simp add: isLeaf-pt-def null-comp-def)
with rep-repb-nc no1-props
have repb (low no1) ∉ set (prx@node#sfx)
  by auto
with repb-no1-def repb-no-no1 no1-props no1-nLeaf
have repb no ∉ set (prx@node#sfx)
  by (simp add: isLeaf-pt-def null-comp-def)
with repb-no-in-nl show ![thesis]
  by simp
next
assume (repb ∖ low) no1 ≠ (repb ∖ high) no1
with repb-no-share-def no1-in-take-Sucn
have repb-no1-def: repb no1 = hd [sn←(prx@node)]. repNodes-eq sn no1 low high repb]
  by auto
from no1-in-take-Sucn
have [sn←(prx@node)]. repNodes-eq sn no1 low high repb] ≠ []
  apply –
  apply (rule filter-not-empty)
  apply (auto simp add: repNodes-eq-def)
  done
then
have repNodes-repb-no1: repNodes-eq (repb no1) no1 low high repb
  apply (simp only: repb-no1-def )
  apply (rule hd-filter-prop)
  apply simp
  done
with repNodes-repb-no repb-no-no1
have repNodes-eq no1 no low high repb
  by (simp add: repNodes-eq-def)
then show ![thesis]
  by (simp add: repNodes-eq-def)
qed
qed
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 qed end

9 Proof of Procedure Repoint

theory RepointProof imports ProcedureSpecs begin

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

lemma (in Repoint-impl) Repoint-modifies:
  shows ∀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-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-notNull: q ≠ Null and
  lrt: Dag ((rep ≡ low) q) (rep ≡ low) (rep ≡ high) lrt and
  rrt: Dag ((rep ≡ high) q) (rep ≡ low) (rep ≡ high) rrt
  by auto
have rlowa-row: ((rep ≡ lowa) q) = ((rep ≡ low) q)
proof (cases q ∈ set-of lt)
  case True
  note q-in-lt=this
  with pt-changed have lowa-q: lowa q = (rep ∞ low) q
    by simp
  thus (rep ∞ lowa) q = (rep ∞ low) q
proof (cases low q = Null)
  case True
  with lowa-q have lowa q = Null
    by (simp add: null-comp-def)
  with True show ?thesis
    by (simp add: null-comp-def)
next
assume lq-nNull: low q ≠ Null
show ?thesis
proof (cases (rep ∞ 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: (rep ∞ low) q ≠ Null
with lrt lowa-q have lowa q ∈ set-of lrt
  by auto
with Node.prems Node have lowa q ∈ 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 ∉ set-of lt
with pt-same have low q = lowa q
  by auto
thus ?thesis
  by (simp add: null-comp-def)
qed
have rhigha-rhigh: ((rep ∞ higha) q) = ((rep ∞ high) q)
proof (cases q ∈ set-of 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:
  Dag ((rep ∝ lowa) q) (rep ∝ lowa) (rep ∝ higha) lrt
  apply −
  apply (rule Node.hyps)
  apply auto
done
from rhigha-mixed-dag rrt-rep-eq have higha-rrt:
\textit{Dag} \((\text{rep} \propto \text{higha}) \ q \) \((\text{rep} \propto \text{lowa}) \) \((\text{rep} \propto \text{higha}) \) \text{rrt}
apply –
apply (rule Node.hyps)
apply auto
done
with \text{lowa-lrt} q’ q-notNull
show \text{Dag} q \((\text{rep} \propto \text{lowa}) \) \((\text{rep} \propto \text{higha}) \) \((\text{Node lrt} q’ \) \text{rrt})
by simp
qed

\textbf{lemma (in Repoint-impl) Repoint-spec'}:
\textbf{shows}
\begin{align*}
& \forall \sigma, \Gamma \vdash \{ \sigma \} \\
& \quad \forall \text{p} : \equiv \text{PROC Repoint (p)} \\
& \quad \forall \text{rept}. \ (\forall \text{p}. (\forall \sigma. (\forall \text{rept}. (\forall \text{no} \in \text{set-of rept.} \ \tau) \ \sigma_{\text{low}} \ \text{rept}) \\
& \quad \land (\forall \text{no} \in \text{set-of rept.} \ \sigma_{\text{no}} = \text{no})) \\
& \quad \land (\forall \text{pt} / \in \text{set-of rept.} \ \sigma_{\text{low pt}} = \sigma_{\text{low pt}} \land \sigma_{\text{high pt}} = \sigma_{\text{high pt}})) \\
& \quad \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
& \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
& \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
\end{align*}
apply (hoare-rule HoarePartial.ProcRecI)
apply vcg
apply (rule conjI)
apply clarify
prefer 2
apply (intro impI allI )
apply (simp add: null-comp-def)
apply (rule conjI)
prefer 2
apply (clarsimp)
apply clarify
proof –
fix \text{low} \text{high p rep lowa higha pa lowb highb pb rept}
assume p-nNull: \text{p} \neq \text{Null}
assume rp-nNull: \text{rep p} \neq \text{Null}
assume rec-spec-rept:
\begin{align*}
& \forall \text{rept}. \ (\forall \sigma. (\forall \text{rept}. (\forall \text{no} \in \text{set-of rept.} \ \sigma_{\text{no}} = \text{no})) \\
& \quad \land (\forall \text{pt} / \in \text{set-of rept.} \ \sigma_{\text{low pt}} = \sigma_{\text{low pt}} \land \sigma_{\text{high pt}} = \sigma_{\text{high pt}}) \\
& \quad \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
& \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
\end{align*}
assume rec-spec-rrept:
\begin{align*}
& \forall \text{rept}. \ (\forall \sigma. \ (\forall \text{rept}. (\forall \text{no} \in \text{set-of rept.} \ \sigma_{\text{no}} = \text{no})) \\
& \quad \land (\forall \text{pt} / \in \text{set-of rept.} \ \sigma_{\text{low pt}} = \sigma_{\text{low pt}} \land \sigma_{\text{high pt}} = \sigma_{\text{high pt}})) \\
& \quad \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
& \quad \land (\forall \text{pt}. \ p = \tau \text{low pt} \land \tau \text{high pt}) \\
\end{align*}
assume repl-dag: Dag (\( \text{rep} \propto \text{id} \)) \( \text{p} \) (\( \text{rep} \propto \text{low} \)) (\( \text{rep} \propto \text{high} \)) \( \text{rept} \)
assume rno-rept: \( \forall \text{no} \in \text{set-of} \ \text{rept} \). \( \text{rep} \ \text{no} = \text{no} \)
show Dag (\( \text{rep} \ p \) \( \text{lowb} \) (highb(\( \text{rep} \ p := \text{pb} \))) \( \text{rept} \) ∧
\( (\forall \text{pt}. \ \text{pt} \notin \text{set-of} \ \text{rept} \rightarrow \text{low pt} = \text{lowb pt} \land \text{high pt} = \text{highb}(\text{rep} \ p := \text{pb})) \) pt)

proof −
from rp-nNull repl-dag p-nNull obtain lrept rrept where
rept-def: \( \text{rept} = \text{Node} \ lrept \ (\text{rep} \ p) \) rrept
by auto
with repl-dag p-nNull have lrept-dag:
Dag (\( \text{rep} \propto \text{low} \)) (\( \text{rep} \propto \text{low} \)) (\( \text{rep} \propto \text{high} \)) lrept
by simp
from rept-def repl-dag p-nNull have rrept-dag:
Dag (\( \text{rep} \propto \text{high} \)) (\( \text{rep} \propto \text{low} \)) (\( \text{rep} \propto \text{high} \)) rrept
by simp
from rno-rept rept-def have rno-lrept:
\( \forall \text{no} \in \text{set-of} \ \text{lrept} \). \( \text{rep} \ \text{no} = \text{no} \)
by auto
from rno-rept rept-def have rno-rrept:
\( \forall \text{no} \in \text{set-of} \ \text{rrept} \). \( \text{rep} \ \text{no} = \text{no} \)
by auto
have repoint-post-low:
Dag \( \text{pa lowa higha} \ \text{lrept} \) ∧
\( (\forall \text{pt}. \ \text{pt} \notin \text{set-of} \ \text{lrept} \rightarrow \text{low pt} = \text{lowa pt} \land \text{high pt} = \text{higha pt}) \)
proof −
from lrept-dag have Dag ((\( \text{rep} \propto \text{id} \)) (\( \text{low} \ (\text{rep} \ p)\))) (\( \text{rep} \propto \text{low} \)) (\( \text{rep} \propto \text{high} \)) lrept
by simp add: id-trans
with rec-spec-lrept rno-lrept show \( \text{?thesis} \)
apply −
apply (erule-tac \( \text{x} = \text{lrept} \) in allE)
apply (erule impE)
apply simp
apply assumption
done
qed
hence lown-lowa-nc: (\( \forall \text{pt}. \ \text{pt} \notin \text{set-of} \ \text{lrept} \rightarrow \text{low pt} = \text{lowa pt} \land \text{high pt} = \text{higha pt} \))
by simp
from lrept-dag repoint-post-low obtain
pa-def: \( \text{pa} = (\text{rep} \propto \text{low}) \) (\( \text{rep} \propto \text{low} \)) and
lowa-higha-def: (\( \forall \text{no} \in \text{set-of} \ \text{lrept}. \ \text{lowa no} = (\text{rep} \propto \text{low}) \ \text{no} \land \text{higha no} = (\text{rep} \propto \text{high}) \) no)
apply −
apply (drule Dags-eq-hp-eq)
apply auto
done
from repl-dag have repl-DAG: DAG \( \text{rept} \)
by (rule Dag-is-DAG)
with repl-def have rp-notin-lrept: \( \text{rep} \ \text{p} \notin \text{set-of} \ \text{lrept} \)
by simp
from rept-DAG rept-def have rp-notin-rrept: rep p \notin set-of rrept 
   by simp 
   have Dag ((rep \Join id) (higha (rep p))) (rep \Join lowa(rep p := pa)) (rep \Join higha) rrept 
   proof - 
   from low-lowa-nc rp-notin-lrept have (rep \Join high) (rep p) = (rep \Join higha) (rep p) 
   by (auto simp add: null-comp-def) 
   with rrept-dag have higha-mixed-rrept: Dag ((rep \Join id) (higha (rep p))) (rep \Join low) (rep \Join higha) 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 \Join id) (higha (rep p))) (rep \Join lowa) (rep \Join higha) rrept 
   apply - 
   apply (rule low-high-exchange-dag) 
   apply auto 
   done 
   have Dag ((rep \Join id) (higha (rep p))) (rep \Join lowa) (rep \Join higha) rrept = Dag ((rep \Join id) (higha (rep p))) (rep \Join lowa(rep p := pa)) (rep \Join higha) rrept 
   proof - 
   have \(\forall\) no \in set-of rrept. (rep \Join lowa) no = (rep \Join lowa(rep p := pa)) no 
   \wedge 
   (rep \Join higha) no = (rep \Join higha) no 
   proof 
   fix no 
   assume no-in-rrept: no \in set-of rrept 
   with rp-notin-rrept have no \neq rep p 
   by blast 
   thus (rep \Join lowa) no = (rep \Join lowa(rep p := pa)) no \wedge 
   (rep \Join higha) no = (rep \Join higha) no 
   by (simp add: null-comp-def) 
   qed 
   thus \?thesis 
   by (rule heaps-eq-Dag-eq) 
   qed 
   with lowa-higha-rrept show \?thesis 
   by simp 
   qed 
   with rec-spec-rrept rno-rrept have repoint-rrept: Dag pb lowb highb rrept \wedge 
   (\forall pt. pt \notin set-of rrept \rightarrow 
    (lowa(rep p := pa)) pt = lowb pt \wedge higha pt = highb pt) 
   apply - 
   apply (erule-tac x=rrept in allE) 
   apply (erule impE) 
   apply simp 
   apply assumption 
   done 

133
then have \textit{ab-nc}: (\(\forall\) pt. pt \(\notin\) set-of \(r\)rept \(\rightarrow\) \\
\(l\)owa(rep p := pa)) pt = lowb pt \(\land\) higha pt = highb pt)
\textit{by simp}

\textit{from repoint-rrept rrept-dag obtain}
\(pb\)-\textit{def}: \(pb = ((rep \propto high) (rep p))\) \textit{and}
\(lowb\)-\(highb\)-\textit{def}: (\(\forall\) no \(\in\) set-of \(r\)rept. lowb no = (rep \(\propto\) low) no \(\land\) highb no = \\
(rep \(\propto\) high) no)
\textit{apply} –
\textit{apply (drule Dags-eq-hp-eq)}
\textit{apply auto}
\textit{done}

\textit{have rept-end-dag: Dag (rep p) lowb (highb(rep p := pb)) rept}
\textit{proof} –
\textit{have \(\forall\) no \(\in\) set-of rept. lowb no = (rep \(\propto\) low) no \(\land\) (highb(rep p := pb)) no = (rep \(\propto\) high) no}
\textit{proof}
\textit{fix no}
\textit{assume no-in-rept: no \(\notin\) set-of rept}
\textit{show \(\forall\) no \(\in\) set-of lrept. lowb no = (rep \(\propto\) low) no \(\land\) (highb(rep p := pb)) no = (rep \(\propto\) high) no}
\textit{proof (cases no \(\in\) set-of \(r\)rept)}
\textit{case True}
\textit{with lowb-highb-def pb-def show \?thesis}
\textit{by simp}
\textit{next}
\textit{assume no-notin-rrept: no \(\notin\) set-of \(r\)rept}
\textit{show \?thesis}
\textit{proof (cases no \(\notin\) set-of lrept)}
\textit{case True}
\textit{with no-notin-rrept rp-notin-lrept ab-nc}
\textit{have \textit{ab-nc-no}: l\}owa no = lowb no \(\land\) higha no = highb no
\textit{apply} –
\textit{apply (erule-tac x=no in allE)}
\textit{apply (erule impE)}
\textit{apply simp}
\textit{apply (subgoal-tac no \(\neq\) rep p)}
\textit{apply simp}
\textit{apply blast}
\textit{done}

\textit{from l\}owa-higha-def True have}
\textit{l\}owa no = (rep \(\propto\) low) no \(\land\) higha no = (rep \(\propto\) high) no
\textit{by auto}
\textit{with ab-nc-no have lowb no = (rep \(\propto\) low) no \(\land\) highb no = (rep \(\propto\) high) no}
\textit{by simp}
\textit{with rp-notin-lrept True show \?thesis}
\textit{apply (subgoal-tac no \(\neq\) rep p)}
\textit{apply simp}
\textit{apply blast}
done

next

assume no-notin-lrept: no /∈ set-of lrept
with no-in-rept rept-def no-notin-rrept have no-rp: no = rep p
    by simp
with rp-notin-lrept low-lowa-nc have a-nc:
    low no = lowa no ∧ high no = higha no
    by auto
from rp-notin-rrept no-rp ab-nc have
    (lowa(rep p := pa)) no = lowb no ∧ higha no = highb no
    by auto
with a-nc pa-def no-rp have (rep ∞ low) no = lowb no ∧ high no =
    highb no
    by auto
with pb-def no-rp show ?thesis
    by simp
qed

with rept-dag have Dag (rep p) low (highb(rep p := pb)) rept =
    Dag (rep p) (rep ∞ low) (rep ∞ high) rept
    apply thm heaps-eq-Dag-eq
    apply (rule heaps-eq-Dag-eq)
    apply auto
    done
with rept-dag p-nNull show ?thesis
    by simp
qed
have (∀ pt. pt /∈ set-of rept → low pt = lowb pt ∧ high pt = (highb(rep p := pb)) pt)
    proof (intro allI impl)
    fix pt
    assume pt-notin-rept: pt /∈ set-of rept
    with rept-def obtain
        pt-notin-lrept: pt /∈ set-of lrept and
        pt-notin-rrept: pt /∈ set-of rrept and
        pt-neq-rp: pt ≠ rep p
        by simp
    with low-lowa-nc ab-nc show low pt = lowb pt ∧ high pt = (highb(rep p := pb)) pt
        by auto
    qed
    with rept-end-dag show ?thesis
        by simp
    qed

lemma (in Repoint-impl) Repoint-spec:
shows
\[\forall \sigma. \Gamma \vdash \{ | \sigma. \text{Dag}((\text{rep }\propto \text{id}) \text{ p}) (\text{rep }\propto \text{low}) (\text{rep }\propto \text{high}) \text{ rept} \wedge (\forall \text{ no } \in \text{ set-of rept}. \text{ rep no = no}) \} \]
\[\text{p := PROC Repoint}(\text{ p})\]
\[\{ | \text{Dag p low high rept} \wedge (\forall \text{ pt. pt }\notin \text{ set-of rept }\rightarrow \sigma \text{low pt = low pt }\wedge \sigma \text{high pt = high pt}) \}\]
apply (hoare-rule HoarePartial.ProcRec1)
apply vcg
apply (rule conjI)
prefer 2
apply (clarsimp simp add: null-comp-def)
apply clarify
apply (rule conjI)
prefer 2
apply clarsimp
apply clarify
proof
- fix rept low high rep p
assume rept-dag: Dag ((\text{rep }\propto \text{id}) \text{ p}) (\text{rep }\propto \text{low}) (\text{rep }\propto \text{high}) \text{ rept}
assume rno-rept: \forall \text{ no } \in \text{ set-of rept}. \text{ rep no = no}
assume p-nNull: \text{ p }\neq \text{ Null}
assume rp-nNull: \text{ rep p }\neq \text{ Null}
show \exists lrept.
\[\text{Dag ((\text{rep }\propto \text{id}) (\text{low (rep p)))) (\text{rep }\propto \text{low}) (\text{rep }\propto \text{high}) \text{ lrept }\wedge (\forall \text{ no } \in \text{ set-of lrept}. \text{ rep no = no}) \wedge (\forall \text{ lowa higha pa}. \]
\[\text{Dag pa lowa higha lrept }\wedge (\forall \text{ pt. pt }\notin \text{ set-of lrept }\rightarrow \text{low pt = lowa pt }\wedge \text{high pt = higha pt}) \rightarrow (\exists lrept.) \]
\[\text{Dag ((\text{rep }\propto \text{id}) (\text{higha (rep p)))) (\text{rep }\propto \text{lowa(rep p := pa)}) (\text{rep }\propto \text{higha}) \text{ rrept }\wedge (\forall \text{ no } \in \text{ set-of rrept}. \text{ rep no = no}) \wedge (\forall \text{ lowb highb pb}. \]
\[\text{Dag pb lowb highb rrept }\wedge (\forall \text{ pt. pt }\notin \text{ set-of rrept }\rightarrow (\text{lowa(rep p := pa)}) \text{ pt = lowb pt }\wedge \text{higha pt = highb pt}) \rightarrow \]
\[\text{Dag (rep p) lowb (highb(rep p := pb)) rrept }\wedge (\forall \text{ pt. pt }\notin \text{ set-of rrept }\rightarrow \text{low pt = lowb pt }\wedge \text{high pt = (highb(rep p := pb)) pt}))\]

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:
\[\text{Dag ((\text{rep }\propto \text{low}) (\text{rep p})) (\text{rep }\propto \text{low}) (\text{rep }\propto \text{high}) \text{ lrept}\]
by simp

136
from rept-def rept-dag p-nNull have rrept-dag:
\[ \text{Dag} \left( \left( \text{rep} \propto \text{high} \right) \left( \text{rep} \propto \text{low} \right) \left( \text{rep} \propto \text{high} \right) \right) \text{rrept} \]
by simp

from rno-rept rept-def have rno-lrept: \( \forall \, \text{no} \in \text{set-of lrept} \). \( \text{rep no} = \text{no} \)
by auto

from rno-rept rept-def have rno-rrept: \( \forall \, \text{no} \in \text{set-of rrept} \). \( \text{rep no} = \text{no} \)
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: \( \text{Dag pa lowa higha lrept} \)
by fact
have low-lowa-nc:
\( \forall \, \text{pt} \, . \, \text{pt} \notin \text{set-of lrept} \rightarrow \text{low pt} = \text{lowa pt} \land \text{high pt} = \text{higha pt} \)
by fact
from lrept-dag lrepta obtain
pa-def: \( \text{pa} = \left( \text{rep} \propto \text{id} \right) \left( \text{higha} \left( \text{rep p} \right) \right) \)
lowa-higha-def: \( \forall \, \text{no} \in \text{set-of lrept} \).
\( \text{lowa no} = \left( \text{rep} \propto \text{low} \right) \text{no} \land \text{higha no} = \left( \text{rep} \propto \text{high} \right) \text{no} \)
apply –
apply (drule Dags-eq-hp-eq)
apply auto
done

from rept-dag have rept-DAG: \( \text{DAG} \text{ rept} \)
by (rule Dag-is-DAG)
with rept-def have rp-notin-lrept: \( \text{rep p} \notin \text{set-of lrept} \)
by simp
from rept-DAG rept-def have rp-notin-rrept: \( \text{rep p} \notin \text{set-of rrept} \)
by simp

have rrepta: \( \text{Dag} \left( \left( \text{rep} \propto \text{id} \right) \left( \text{higha} \left( \text{rep p} \right) \right) \right) \)
\( \left( \text{rep} \propto \text{lowa} \left( \text{rep p} := \text{pa} \right) \right) \left( \text{rep} \propto \text{higha} \right) \text{rrept} \)
proof –
from low-lowa-nc rp-notin-lrept have \( \left( \text{rep} \propto \text{high} \right) \left( \text{rep} \propto \text{high} \right) \left( \text{rep} \propto \text{high} \right) \)
by (auto simp add: null-comp-def)
with rrept-dag have higha-mixed-rrept:
\( \text{Dag} \left( \left( \text{rep} \propto \text{id} \right) \left( \text{higha} \left( \text{rep p} \right) \right) \right) \left( \text{rep} \propto \text{low} \right) \left( \text{rep} \propto \text{high} \right) \text{rrept} \)
by (simp add: id-trans)
thm low-high-exchange-dag
with low-lowa-nc lowa-higha-def rno-rrept
have lowa-higha-rrept:
\( \text{Dag} \left( \left( \text{rep} \propto \text{id} \right) \left( \text{higha} \left( \text{rep p} \right) \right) \right) \left( \text{rep} \propto \text{low} \right) \left( \text{rep} \propto \text{high} \right) \text{rrept} \)
apply –
apply (rule low-high-exchange-dag)
apply auto

137
have $\text{Dag} ((\text{rep} \propto \text{id}) (\text{higha} (\text{rep} p))) (\text{rep} \propto \text{lowa}) (\text{rep} \propto \text{higha}) \text{rrept} = \text{Dag} ((\text{rep} \propto \text{id}) (\text{higha} (\text{rep} p))) (\text{rep} \propto \text{lowa}(\text{rep} p := \text{pa})) (\text{rep} \propto \text{higha}) \text{rrept}$

proof 

have $\forall \text{no} \in \text{set-of} \text{rrept}.$

$(\text{rep} \propto \text{lowa}) \text{no} = (\text{rep} \propto \text{lowa}(\text{rep} p := \text{pa})) \text{no} \land$

$(\text{rep} \propto \text{higha}) \text{no} = (\text{rep} \propto \text{higha}) \text{no}$

proof 

fix no 

assume no-in-rrept: \(\text{no} \in \text{set-of} \text{rrept}\)

with rp-notin-rrept have no $\neq$ rep p

by blast 

thus $(\text{rep} \propto \text{lowa}) \text{no} = (\text{rep} \propto \text{lowa}(\text{rep} p := \text{pa})) \text{no} \land$

$(\text{rep} \propto \text{higha}) \text{no} = (\text{rep} \propto \text{higha}) \text{no}$

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 exI)

apply (rule conjI)

apply (rule rrepta)

apply (rule conjI)

apply (rule rno-rrept)

apply clarify

proof 

case (goal1 lowb highb pb)

have rreptb: $\text{Dag} \text{pb} \text{lowlb highbh} \text{rrept}$ by fact 

have ab-nc: $\forall \text{pt}. \text{pt} \notin \text{set-of} \text{rrept} \rightarrow$

$(\text{lowl}(\text{rep} p := \text{pa})) \text{pt} = \text{lowlb} \text{pt} \land \text{highha} \text{pt} = \text{highbh} \text{pt}$ by fact

from rreptb rrept-dag obtain

pb-def: $\text{pb} = ((\text{rep} \propto \text{high}) (\text{rep} p))$ and

lowb-highh-def: $\forall \text{no} \in \text{set-of} \text{rrept}.$

lowb no = (rep $\propto$ low) no $\land$ highh no = (rep $\propto$ high) no

apply 

apply (drule Dags-eq-hp-eq)

apply auto

done 

have rept-end-dag: $\text{Dag} (\text{rep} p) \text{lowlb} (\text{highb}(\text{rep} p := \text{pb})) \text{rrept}$ 

proof 

have $\forall \text{no} \in \text{set-of} \text{rrept}.$

lowb no = (rep $\propto$ low) no $\land$ (highb(rep p := pb)) no = (rep $\propto$ 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) no
proof (cases no ∈ set-of rept)
  case True
  with lowb-highb-def pb-def show thesis
    by simp
next
assume no-notin-rept: no /∈ set-of rept
show thesis
proof (cases no ∈ set-of rept)
  case True
  with no-notin-rept rp-notin-rept 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
    by simp
with rp-notin-rept True show thesis
  apply (subgoal-tac no ≠ rep p)
  apply simp
  apply blast
done
next
assume no-notin-rept: no /∈ set-of rept
with no-in-rept rept-def no-notin-rept have no-rp: no = rep p
  by simp
with rp-notin-rept low-lowa-nc
  have a-nc: low no = lowa no ∧ 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
done

with rept-dag
have Dag (rep p) lowb (highb(rep p := pb)) rept =
   Dag (rep p) (rep \propto low) (rep \propto high) rept
   apply -
   apply (rule heaps-eq-Dag-eq)
   apply auto
   done

with rept-dag p-nNull show ?thesis
   by simp
qed

have (\forall pt. pt \notin 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 ?case
   by simp
qed
qed

lemma (in Repoint-impl) Repoint-spec-total:
shows
  \forall \sigma \text{rept}. \Gamma \vdash \{\sigma. Dag ((\text{\`{r}ep} \propto \text{id}) \text{\`{r}ep}) (\text{\`{r}ep} \propto \text{low}) (\text{\`{r}ep} \propto \text{high}) \text{rept} \\
    \land (\forall no \in set-of rept. \text{\`{r}ep no = no}) \} \\
  \text{\`{p} := PROC Repoint (\text{\`{p}})} \\
  \{\text{Dag \text{\`{p}} low \text{high rept} \land} \\
  (\forall pt. pt \notin set-of rept \rightarrow \sigma low pt = \text{low pt} \land \sigma high pt = \text{high pt})\}
apply (hoare-rule HoareTotal.ProcRec1
   [where r=measure (\lambda(s,p). size
   (dag ((\text{\`{r}ep} \propto \text{id}) \text{\`{r}ep}) (\text{\`{r}ep} \propto \text{low}) (\text{\`{r}ep} \propto \text{high}))))]
apply vcg
apply (rule conjI)
prefer 2
apply (clarsimp simp add: null-comp-def)
apply clarify
apply (rule conjI)
prefer 2
apply clarsimp
apply clarify

proof –

fix rept low high rep p
assume rept-dag: Dag ((rep ⊡ 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) ∧
size (dag ((rep ⊡ id) (low (rep p))) (rep ⊡ low) (rep ⊡ high))
< size (dag ((rep ⊡ id) p) (rep ⊡ low) (rep ⊡ high)) ∧
(∀ lowa higha pa.
Dag pa lowa higha lrept ∧
(∀ pt. pt /∈ set-of lrept → 
low pt = lowa pt ∧ high pt = higha pt) →
(∃ rrept.
Dag ((rep ⊡ id) (higha (rep p))) (rep ⊡ lowa(rep p := pa))
(rep ⊡ higha) rrept ∧
(∀ no ∈ set-of rrept. rep no = no) ∧
size (dag ((rep ⊡ id) (higha (rep p))))
< size (dag ((rep ⊡ id) p) (rep ⊡ low) (rep ⊡ higha)) ∧
(∀ lowb highb pb.
Dag pb lowb highb rrept ∧
(∀ pt. pt /∈ set-of rrept → 
(lowa(rep p := pa)) pt = lowb pt ∧
higha pt = highb pt) →
Dag (rep p) lowb (highb(rep p := pb)) rrept ∧
(∀ pt. pt /∈ set-of rrept →
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 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 (goalI lowa higha pa)

have lrept-a: 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 lrept-a 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

proof −

from lrept-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 rrept-a: Dag ((rep ⊪ id) (higna (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) (higna (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 ⊪ id) (higna (rep p))) (rep ⊪ lowa) (rep ⊪ higha) rrept

apply −
apply (rule low-high-exchange-dag)
apply auto

done

have Dag ((rep ⊪ id) (higna (rep p))) (rep ⊪ lowa) (rep ⊪ higha) rrept =


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

**proof**

**have** $\forall \text{no } \in \text{set-of rrept}.$

\[
\quad (\text{rep } \propto \text{lowa}) \text{ no } = (\text{rep } \propto \text{lowa}(\text{rep } p := pa)) \text{ no } \land
\quad (\text{rep } \propto \text{higha}) \text{ no } = (\text{rep } \propto \text{higha}) \text{ no}
\]

**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 } \propto \text{lowa}) \text{ no } = (\text{rep } \propto \text{lowa}(\text{rep } p := pa)) \text{ no } \land

\[
\quad (\text{rep } \propto \text{higha}) \text{ no } = (\text{rep } \propto \text{higha}) \text{ no}
\]

**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 exI)

**apply** (rule conjI)

**apply** (rule rrepta)

**apply** (rule conjI)

**apply** (rule rno-rrept)

**apply** (rule conjI)

**using** rept-dag rept-def rrepta

**apply** (simp only: Dag-dag)

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

**apply** clarify

**proof**

**case** (goal1 lowb highb pb)

**have** rreptb: Dag pb lowb highb rrept **by** fact

**have** ab-nc: $\forall \text{pt } . \text{ pt } \notin \text{set-of rrept } \rightarrow$

\[
\quad (\text{lowa}(\text{rep } p := pa)) \text{ pt } = \text{lowb pt } \land \text{higha pt } = \text{highb pt } \text{ by } \text{fact}
\]

**from** rreptb rrept-dag **obtain**

\[
\quad \text{pb-def}: pb = ((\text{rep } \propto \text{high}) (\text{rep } p)) \quad \text{and}
\]

\[
\quad \text{lowb-highb-def}: \forall \text{no } \in \text{set-of rrept}.
\quad \quad \text{lowb no } = (\text{rep } \propto \text{low}) \text{ no } \land \text{highb no } = (\text{rep } \propto \text{high}) \text{ no}
\]

**apply**

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

**apply** auto

**done**

**have** rrept-end-dag: Dag (rep p) lowb (highb(rep p := pb)) rept

**proof**

**have** $\forall \text{no } \in \text{set-of rept}.$

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

10 Proof of Procedure Normalize

theory NormalizeTotalProof imports LevellistProof ShareReduceRepListProof
  RepointProof begin


lemma (in Normalize-impl) Normalize-modifies:
shows
  ∀ σ. Γ |- {σ} p := PROC Normalize {p} {t. t may-only-modify-globals σ in [rep,mark,low,high,next]}

145
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

lemma (in Normalize-impl) Normalize-spec:
shows ∀ σ pret prebdt. Γ ⊢ σ
{[σ]. Dag 'p 'low 'high pret ∧ ordered pret 'var ∧
'p ≠ Null ∧ (∀ n. n ∈ set-of pret → 'mark n = 'mark 'p) ∧
bdt pret 'var = Some prebdt} 
'p ::= PROC Normalize( 'p)

{∀ pt. pt ∉ set-of pret
  → σrep pt = 'rep pt ∧ σlow pt = 'low pt ∧ σhigh pt = 'high pt ∧
  σmark pt = 'mark pt ∧ σnext pt = 'next pt) ∧
(∃ postt. Dag 'p 'low 'high postt ∧ reduced postt ∧
shared postt 'var ∧ ordered postt 'var ∧
set-of postt ⊆ set-of pret ∧
(∃ postbdt. bdt postt σvar = Some postbdt ∧ prebdt ~ postbd))} ∧
(∀ no. no ∈ set-of pret → 'mark no = (¬ σmark σp))} 
apply (hoare-rule HoareTotal.ProcNoRec1)
apply (hoare-rule anno=)
  levellist ::= replicate ( 'p→ 'var + 1) Null;;
  levellist ::= CALL Levellist ( 'p, (¬ 'p→ 'mark) , levellist);;
  (ANNO (τ, ll). } τ. Levellist levellist 'next ll ∧
    Dag σp σlow σhigh pret ∧ ordered pret σvar ∧ σp ≠ Null ∧
    (bdt pret σvar = Some prebdt) ∧
    wf-ll pret ll σvar ∧
    length levellist = ((σp → σvar) + 1) ∧
    wf-marking pret σmark τmark (¬ σmark σp) ∧
    (∀ pt. pt ∉ set-of pret → σnext pt = 'next pt) ∧
    'low = σlow ∧ 'high = σhigh ∧ 'p = σp ∧ 'rep = σrep ∧
    'var = σvar }í
'n ::= 0;;
WHILE ('n < length levellist)
INV {∀ levellist 'next ll ∧
  Dag σp σlow σhigh pret ∧ ordered pret σvar ∧ σp ≠ Null ∧
  (bdt pret σvar = Some prebdt) ∧ wf-ll pret ll σvar ∧
  length levellist = ((σp → σvar) + 1) ∧
  wf-marking pret σmark τmark (¬ σmark σp) ∧
  'low = σlow ∧ 'high = σhigh ∧ 'p = σp ∧ 'rep = σrep ∧
  'var = σvar }í
'n ≤ length levellist ∧
(∀ pt i. (pt ∉ set-of pret ∀ ('n <= i ∧ pt ∈ set (ll ! i) ∧
  i < length levellist) 
  → σrep pt = 'rep pt)) ∧
'rep · Nodes 'n ll ⊆ Nodes 'n ll ∧
(∀ no ∈ Nodes 'n ll.
  no→'rep→σvar <= no→σvar ∧
  (∃ not nort. Dag ('rep no) ('rep ∞ σlow) ('rep ∞ σhigh) nort ∧
    Dag no σlow σhigh not ∧ reduced nort ∧
    ordered nort σvar ∧ set-of nort ⊆ 'rep · Nodes 'n ll ∧

146
\[ \forall \text{no} \in \text{set-of nort.} \ (\text{rep no} = \text{no}) \land \]
\[ (\exists \text{nordt norbdt.} \ \text{bdt not } \sigma \text{var} = \text{Some norbdt} \land \]
\[ \text{bdt nort } \sigma \text{var} = \text{Some norbdt} \land \text{nordt not norbdt}) \land \]
\[ (\forall \text{t1 t2,} \]
\[ \text{t1} \in \text{Dags ('}\text{rep '}(\text{Nodes '} n ll))('\text{rep } \sigma \text{low })('\text{rep } \sigma \text{high})\land \]
\[ \text{t2} \in \text{Dags ('}\text{rep '}(\text{Nodes '} n ll))('\text{rep } \sigma \text{low })('\text{rep } \sigma \text{high}) \]
\[ \rightarrow \]
\[ \text{isomorphic-dags-eq t1 t2 } \sigma \text{var} \land \]
\[ \text{levellist } = \tau \text{levellist} \land \text{'next } = \tau \text{'next} \land \text{'mark } = \tau \text{'mark} \land \text{'low } = \sigma \text{'low} \land \]
\[ \text{'high } = \sigma \text{'high} \land \text{'p } = \sigma \text{'p } \land \text{'var } = \sigma \text{'var } \}
\]
\[ \text{VAR MEASURE (length 'levellist } - 'n) \]
\[ \text{DO} \]
\[ \text{CALL ShareReduceRepList('levellist } ! 'n);} \]
\[ 'n :== 'n + 1 \]
\[ \text{OD} \]
\[ \{\exists \text{postnormt.} \ 	ext{Dag ('rep } \sigma \text{p') ('rep } \sigma \text{low } ('\text{rep } \sigma \text{high}) \text{ postnormt } \land \]
\[ \text{reduced postnormt } \land \text{shared postnormt } \sigma \text{var } \land \]
\[ \text{ordered postnormt } \sigma \text{var } \land \text{set-of postnormt } \subseteq \text{set-of pret } \land \]
\[ (\exists \text{postnormbdt.} \ 	ext{bdt postnormt } \sigma \text{var} = \text{Some postnormbdt } \land \text{prebdt } \sim \text{postnormbdt}) \land \]
\[ (\forall \text{no } \in \text{set-of postnormt.} \ (\text{rep no} = \text{no})) \land \]
\[ \text{ordered pret } \sigma \text{var } \land \sigma \text{p } \neq \text{Null } \land \]
\[ (\forall \text{pt. pt } \notin \text{set-of pret } \rightarrow \text{rep pt } = \text{rep pt}) \land \]
\[ \text{levellist } = \tau \text{levellist} \land \text{'next } = \tau \text{'next} \land \text{'mark } = \tau \text{'mark} \land \text{'low } = \sigma \text{'low} \land \]
\[ \text{'high } = \sigma \text{'high} \land \]
\[ \text{'p } = \sigma \text{'p } \land (\forall \text{no. no } \in \text{set-of pret } \rightarrow \text{mark no } = (\neg \sigma \text{'mark } \sigma \text{'p})) \} \}
\]
\[ ;; \]
\[ 'p :== \text{CALL Repoint ('p) in HoareTotal.annotateI} \]
\[ \text{apply (vcg spec } = \text{spec-total) prefer 2} \]
\[ \text{apply (simp add: Nodes-def null-comp-def)} \]
\[ \text{apply (rule-tac x=pret in ezl)} \]
\[ \text{apply clarify} \]
\[ \text{apply (rule conjI)} \]
\[ \text{apply clarsimp} \]
\[ \text{apply (case-tac i)} \]
\[ \text{apply simp} \]
\[ \text{apply simp} \]
\[ \text{apply (rule conjI)} \]
\[ \text{apply simp} \]
\[ \text{apply (rule conjI)} \]
\[ \text{apply simp} \]
\[ \text{apply clarify} \]
\[ \text{apply (simp (no-asm-use) only: simp-thms)} \]
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-nblla: ∼ nb < length levellista
assume nb-le-lla: nb ≤ length levellista

assume normalize-prop: ∀ no∈Nodes nb ll.
  var (repb no) ≤ var no ∧
  (∃ not nort. Dag (repb no) (repb ∝ low) (repb ∝ high) nort ∧
  Dag no low high not ∧ reduced nort ∧ ordered nort var ∧
set-of nort ⊆ repb ' Nodes nb ll ∧
(∀ no∈set-of nort. repb no = no) ∧
(∃ nobdt norbdt. bdt not var = Some nobdt ∧
bdt nort var = Some norbdt ∧ nobdt ∼ norbdt))

**assume** repbNodes-in-Nodes: repb ' Nodes nb ll ⊆ Nodes nb ll

**assume** shared-mult-dags:

∀ t1 t2. t1 ∈ Dags (repb ' Nodes nb ll) (repb ∞ low) (repb ∞ high) ∧
t2 ∈ Dags (repb ' Nodes nb ll) (repb ∞ low) (repb ∞ high) → isomorphic-dags-eq t1 t2 var

**show** (∃ postnormt. Dag (repb p) (repb ∞ low) (repb ∞ high) postnormt ∧
reduced postnormt ∧ shared postnormt var ∧
ordered postnormt var ∧ set-of postnormt ⊆ set-of pret ∧
(∃ postnormbdt. bdt postnormt var = Some postnormbdt ∧
prebdt ∼ postnormbdt) ∧
(∀ no ∈ set-of postnormt. repb no = no) ∧
ordered pret var ∧ p ≠ Null ∧
(∀ pt. pt /∈ set-of pret → rep pt = repb pt) ∧
(∀ no. no ∈ set-of pret → marka no = (¬ 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: ∀ pt. pt /∈ set-of pret → 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: (∀ no. no ∈ set-of pret → marka no = (¬ 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 ∈ set-of pret
 by simp

**with** wf-lla have p ∈ set (ll ! (var p))
 by (simp add: wf-ll-def)

**with** varp-s-nb have p-in-Nodes: p ∈ Nodes nb ll
 by (auto simp add: Nodes-def)

**with** normalize-prop obtain not nort where

varrepno-varno: var (repb p) ≤ var p and

nort-dag: Dag (repb p) (repb ∞ low) (repb ∞ high) nort and

not-dag: Dag p low high not and

red-nort: reduced nort and

ord-nort: ordered nort var and

nort-in-repNodes: set-of nort ⊆ repb ' Nodes nb ll and

nort-repb: (∀ no∈set-of nort. repb no = no) and

bdt-prop: ∃ nobdt norbdt. bdt not var = Some nobdt ∧ bdt nort var = Some norbdt ∧
nobdt ~ norbdt
by auto

from wf-lla nb-length-lla have Nodes-in-pret: Nodes nb ll ⊆ set-of pret
apply −
apply (rule Nodes-in-pret)
apply (auto simp add: length-ll-eq)
done
from pret-dag wf-lla nb-length-lla have Null ∉ Nodes nb ll
apply −
apply (rule Null-notin-Nodes)
apply (auto simp add: length-ll-eq)
done
with p-in-Nodes repbNodes-in-Nodes have rp-nNull: repb p ≠ Null
by auto
with nort-dag have nort-nTip: nort ≠ Tip
by auto
have ∃ postnormt. Dag (repb p) (repb low) (repb high) postnormt ∧ reduced postnormt ∧ shared postnormt var ∧ ordered postnormt var ∧ set-of postnormt ⊆ set-of pret ∧ (∃ postnormbdt. bdt postnormt var = Some postnormbdt ∧ prebdt ~ postnormbdt) ∧ (∀ no ∈ set-of postnormt. repb no = no)
proof (rule-tac x = nort in exI)
from nort-in-repNodes repbNodes-in-Nodes Nodes-in-pret
have nort-in-pret: set-of nort ⊆ set-of pret
by blast
from not-dag pret-dag have not-pret: not = pret
by (simp add: Dag-unique)
with bdt-prop prebdt have pret-bdt-prop:
∃ postnormbdt. bdt nort var = Some postnormbdt ∧ prebdt ~ postnormbdt
by auto
from shared-mult-dags have shared nort var
proof (auto simp add: shared-def isomorphic-dags-eq-def)
fix st1 st2 bdt1
assume shared-imp:
∀ t1 t2. t1 ∈ Dags (repb ' Nodes nb ll) (repb low) (repb high) ∧ t2 ∈ Dags (repb ' Nodes nb ll) (repb low) (repb high)
→ (∃ bdt1. bdt t1 var = Some bdt1 ∧ bdt t2 var = Some bdt1) → t1 = t2
assume st1-nort: st1 ≤ nort
assume st2-nort: st2 ≤ nort
assume bdt-st1: bdt st1 var = Some bdt1
assume bdt-st2: bdt st2 var = Some bdt1
from nort-in-repNodes nort-dag nort-nTip
have nort-in-DagsrNodes:
nort ∈ Dags (repb ' (Nodes nb ll)) (repb low) (repb high)
apply −
apply (rule DagsI)
apply auto
done
show \( st1 = st2 \)
proof (cases \( st1 \))
  case Tip
  note \( st1\text{-}Tip=\text{this} \)
  with \( \text{bdt}\text{-}st1 \) \( \text{bdt}\text{-}st2 \) show \( ?\text{thesis} \)
    by auto
next
  case \( (\text{Node} \, \text{lst1} \, st1p \, rst1) \)
  note \( st1\text{-}Node=\text{this} \)
  then have \( st1\text{-}n\text{Tip} \): \( st1 \neq \text{Tip} \)
    by simp
  show \( ?\text{thesis} \)
  proof (cases \( st2 \))
    case Tip
    with \( \text{bdt}\text{-}st1 \) \( \text{bdt}\text{-}st2 \) show \( ?\text{thesis} \)
      by auto
  next
    case \( (\text{Node} \, \text{lst2} \, st2p \, rst2) \)
    note \( st2\text{-}Node=\text{this} \)
    then have \( st2\text{-}n\text{Tip} \): \( st2 \neq \text{Tip} \)
      by simp
      from \( \text{nort}\text{-}in\text{-}DagsrNodes \, st1\text{-}nort \) \( \text{ord}\text{-}nort \) \( \text{wf}\text{-}lla \) \( st1\text{-}n\text{Tip} \)
    have \( st1\text{-in}\text{-}Dags \):
      \( st1 \in \text{Dags} \, (\text{repb} \, \, \text{Nodes} \, \text{nb ll}) \, (\text{repb} \, \leadsto \, \text{low}) \, (\text{repb} \, \leadsto \, \text{high}) \)
      apply --
      apply (rule Dags-subdags)
      apply auto
    done
  from \( \text{nort}\text{-}in\text{-}DagsrNodes \, st2\text{-}nort \) \( \text{ord}\text{-}nort \) \( \text{wf}\text{-}lla \) \( st2\text{-}n\text{Tip} \)
  have \( st2\text{-in}\text{-}Dags \):
    \( st2 \in \text{Dags} \, (\text{repb} \, \, \text{Nodes} \, \text{nb ll}) \, (\text{repb} \, \leadsto \, \text{low}) \, (\text{repb} \, \leadsto \, \text{high}) \)
    apply --
    apply (rule Dags-subdags)
    apply auto
    done
  from \( \text{st1}\text{-in}\text{-}Dags \, \text{st2}\text{-in}\text{-}Dags \) \( \text{bdt}\text{-}st1 \) \( \text{bdt}\text{-}st2 \) \( \text{shared}\text{-}imp \) show \( st1=st2 \)
    by simp
qed
qed

with \( \text{nort}\text{-}dag \) \( \text{red}\text{-}nort \) \( \text{ord}\text{-}nort \) \( \text{nort}\text{-in}\text{-}pret \) \( \text{pret}\text{-}bdt\text{-}prop \) \( \text{nort}\text{-}repb \)
show \( \text{Dag} (\text{repb} \, p) \, (\text{repb} \, \leadsto \, \text{low}) \, (\text{repb} \, \leadsto \, \text{high}) \, \text{nort} \wedge \)
  \( \text{reduced} \, \text{nort} \wedge \text{shared} \, \text{nort \, var} \wedge \text{ordered} \, \text{nort \, var} \wedge \)
  \( \text{set}\text{-of} \, \text{nort} \subseteq \text{set}\text{-of} \, \text{pret} \wedge \)
  \( \exists \, \text{postnormbdt} \).
  \( \text{bdt} \, \text{nort \, var} = \text{Some} \, \text{postnormbdt} \wedge \text{prebdt} \, \sim \, \text{postnormbdt} \)
\((\forall \text{no} \in \text{set-of nort}. \ \text{repb no} = \text{no})\)

apply
apply \((\text{intro conjI})\)
apply assumption+
done

qed
with \(\text{wf-lla length-lla ord-pret pnN rep-nc-post marking-inverted}\)
show \(?\text{thesis}\)
by simp
qed

next
— From postcondition inner SPEC to final postcondition
fix \(\text{var low high p rep levellist marka next}\)
nexta \(\text{lowb highb pb levellista ll repa pret prebdt}\)
and \(\text{mark::ref} \Rightarrow \text{bool and postnormt postnormbd}\)
assume \(\text{ll: Levellist levellista nexta ll}\)
assume \(\text{repoint-spec:}\)
\(\text{Dag pb lowb highb postnormt}\)
\(\forall \text{pt}, \text{pt} \notin \text{set-of postnormt} \rightarrow \text{low pt} = \text{lowb pt} \land \text{high pt} = \text{highb pt}\)
assume \(\text{pret-dag: Dag p low high pret}\)
assume \(\text{ord-pret: ordered pret var}\)
assume \(\text{pnN: p} \neq \text{Null}\)
assume \(\text{onemark-pre:}\)
\(\forall \text{n}. \ \text{n} \in \text{set-of pret} \rightarrow \text{mark n} = \text{mark p} \ (\text{is} \ \forall \text{n}. \ ?\text{in-pret n} \rightarrow \ ?\text{eq-mark-p n})\)
assume \(\text{pret-bdt: bdt pret var} = \text{Some prebdt}\)

assume \(\text{wf-ll: wf-ll pret ll var and}\)
\(\text{length-ll:length levellist = var p + 1 and and}\)
\(\text{wf-marking-ll: wf-marking pret mark marka (}\neg \text{mark p})\)
assume \(\text{postnormt-dag: Dag (repa p) (repa } \propto \text{ low}) (\text{repa } \propto \text{ high}) \text{ postnormt and}\)
\(\text{reduced-postnormt: reduced postnormt and}\)
\(\text{shared-postnormt: shared postnormt var and}\)
\(\text{ordered-postnormt: ordered postnormt var and}\)
\(\text{subset-pret: set-of postnormt } \subseteq \text{ set-of pret and}\)
\(\text{sim-bdt: bdt postnormt var} = \text{Some postnormbd}\text{ prebdt } \sim \text{postnormbd}\text{ and}\)
\(\text{postdag-repa: } \forall \text{no} \in \text{set-of postnormt}. \ \text{repa no} = \text{no and}\)
\(\text{rep-eq: } \forall \text{pt}, \text{pt} \notin \text{set-of pret} \rightarrow \text{repa pt} = \text{repa pt and}\)
\(\text{pret-marked: } \forall \text{no}. \ \text{no} \in \text{set-of pret} \rightarrow \text{marka no} = (\neg \text{mark p})\)
assume \(\text{unmodif-next: } \forall \text{p}, \ \text{p} \notin \text{set-of pret} \rightarrow \text{next p} = \text{nexta p}\)
show \((\forall \text{pt}, \text{pt} \notin \text{set-of pret}\)
\(\rightarrow \text{low pt} = \text{lowb pt} \land \text{high pt} = \text{highb pt} \land\)
\(\text{mark pt} = \text{marka pt})\)

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

from pret-nc
show ?thesis
by fastforce
qed

— invariant to invariant

next

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

assume rep-nc: (∀ pt i.
  pt /∈ set-of pret ∨ n ≤ i ∧ pt ∈ set (ll ! i) ∧ i < var p + 1 →
  rep pt = repb pt
assume repbNodes-in-Nodes: repb ′ Nodes n ll ⊆ Nodes n ll
assume normalize-prop: ∀ no∈Nodes n ll.
\[\begin{align*}
\text{var (repa no)} & \leq \text{var no} \land \\
(\exists \text{not nort. Dag (repa no) (repa } \propto \text{ low) (repa } \propto \text{ high) nort } \land \\
\text{Dag no low high not } \land \text{ reduced nort } \land \text{ ordered nort var } \land \\
\text{set-of nort } \subseteq \text{ repa } \setminus \text{ Nodes n ll } \land \\
(\forall \text{no} \in \text{set-of nort. repa no } = \text{ no} ) \land \\
(\exists \text{nobdt. bdt not var } = \text{ Some nobdt } \land \\
(\exists \text{norbdt. bdt nort var } = \text{ Some norbdt } \land \\
\text{norbdt } \sim \text{ nobdt}))
\end{align*}\]

**assume**

**isomorphic-dags-eq:**

\[\begin{align*}
\forall t_1 t_2. t_1 \in \text{Dags (repa } \setminus \text{ Nodes n ll) (repa } \propto \text{ low) (repa } \propto \text{ high) } \land \\
t_2 \in \text{Dags (repa } \setminus \text{ Nodes n ll) (repa } \propto \text{ low) (repa } \propto \text{ high) } \rightarrow \text{isomorphic-dags-eq t1 t2 var}
\end{align*}\]

**show**

\[(\forall \text{no} \in \text{set } \{l! n\}).
\]

\[\begin{align*}
\text{no } \neq \text{ Null } \land \\
(\text{low no } = \text{ Null}) = (\text{high no } = \text{ Null}) \land \\
\text{low no } \notin \text{ set } \{l! n\} \land \\
\text{high no } \notin \text{ set } \{l! n\} \land \\
\text{isLeaf-pt no low high } = (\text{var no } \leq 1) \land \\
(\text{low no } \neq \text{ Null } \rightarrow \text{repa (low no) } \neq \text{ Null}) \land (\text{repa } \propto \text{ low) no } \notin \text{ set } \{l! n\}) \land \\
(\forall \text{no1} \in \text{set } \{l! n\}, \forall \text{no2} \in \text{set } \{l! n\}, \text{var no1 } = \text{ var no2} ) \land \\
(\forall \text{repa}. (\forall \text{no}. \text{no } \notin \text{ set } \{l! n\} \rightarrow \text{repa no } = \text{ repa no} ) \land \\
(\forall \text{no} \in \text{set } \{l! n\}).
\end{align*}\]

\[\begin{align*}
\text{repa no } \neq \text{ Null } \land \\
(\text{if (repa } \propto \text{ low) no } = (\text{repa } \propto \text{ high) no } \land \text{low no } \neq \text{ Null}
\text{then repa no } = (\text{repa } \propto \text{ low) no }
\text{else repa no } \in \text{ set } \{l! n\} \land \\
\text{repa (repa no) } = \text{ repa no } \land \\
(\forall \text{no1} \in \text{set } \{l! n\}.
\text{((repa } \propto \text{ high) no1 } = (\text{repa } \propto \text{ high) no } \land \\
(\text{repa } \propto \text{ low) no1 } = (\text{repa } \propto \text{ low) no }) = \\
(\text{repa no } = \text{ repa no1}))) \rightarrow \\
(\forall \text{p+i. pt } \notin \text{ set-of pret } \lor (\text{n+1 } \leq \text{i } \land \text{pt } \in \text{ set } \{l! i\} \land i < \text{var p+1}) \\
(\forall \text{pt i. pt } \notin \text{ set-of pret } \lor (\text{n+1 } \leq i \land pt \in set \{ll! i\} \land i < \text{var p+1}) \rightarrow \\
\text{rep pt } = \text{ repa pt } \land \\
\text{repa } \setminus \text{ Nodes (n+1) ll } \subseteq \text{ Nodes (n+1) ll } \land \\
(\forall \text{no } \in \text{Nodes (n+1) ll}.
\text{var (repa no) } \leq \text{ var no } \land \\
(\exists \text{not nort.}
\text{Dag (repa no) (repa } \propto \text{ low) (repa } \propto \text{ high) nort } \land \\
\text{Dag no low high not } \land \text{ reduced nort } \land \\
\text{ ordered nort var } \land \\
\text{set-of nort } \subseteq \text{ repa } \setminus \text{ Nodes (n+1) ll } \land \\
(\forall \text{no} \in \text{set-of nort. repa no } = \text{ no} ) \land \\
(\exists \text{nobdt}.
\end{align*}\]
\[ bt 
\land
(\exists n \; bt n \equiv \text{Some norbt} 
\land
(\forall t1 t2. 
\quad t1 \in \text{Dags (repा NNodes (n + 1) ll) (repा low) (repा high)}
\land
\quad t2 \in \text{Dags (repा NNodes (n + 1) ll) (repा low) (repा high)}
\rightarrow
\quad \text{isomorphic-dags-eq t1 t2 var})
\]

\text{proof} --
\text{from ll have length-ll-eq: length levellist = length ll}
\quad \text{by (simp add: Levellist-length)}
\text{from n-Suc-var-p lll have nsll: n < length levellist by simp}
\text{hence ns2ll: n \leq length levellist by simp}
\text{have srl-precond: (\forall no \in set (ll ! n).)
\quad no \neq \text{Null} 
\land
\quad (\text{low no = Null}) = (\text{high no = Null}) 
\land
\quad \text{low no} \notin \text{set (ll ! n)} 
\land
\quad \text{high no} \notin \text{set (ll ! n)} 
\land
\quad \text{isLeaf-pt no low high = (\text{var no} \leq 1)} 
\land
\quad (\text{low no} \neq \text{Null} \rightarrow \text{repb (low no)} \neq \text{Null}) 
\land
\quad (\text{repb low} no \notin \text{set (ll ! n)})
\quad \text{proof (intro ballI)}
\quad \text{fix no}
\quad \text{assume no-in-lln: no \in set (ll ! n)}
\quad \text{with wf-ll nsll have no-in-pret-var: no \in set-of pret \land var no = n}
\quad \text{by (simp add: wf-ll-def length-ll-eq)}
\quad \text{with pret-dag have no-nNull: no \neq \text{Null}}
\quad \text{apply –}
\quad \text{apply (rule set-of-nn)}
\quad \text{apply auto}
\quad \text{done}
\text{from pret-dag prebdt-pret no-in-pret-var}
\text{have balanced-no: (low no = Null) = (high no = Null)}
\quad \text{apply –}
\quad \text{apply (erule conjE)}
\quad \text{apply (rule-tac p=p and low=low in balanced-bdt)}
\quad \text{apply auto}
\quad \text{done}
\text{have low-no-notin-lln: low no \notin \text{set (ll ! n)}}
\text{proof (cases low no = Null)}
\quad \text{case True}
\quad \text{note lno-Null=this}
\quad \text{with balanced-no have lno-Null: high no = Null}
\quad \text{by simp}
\quad \text{show \#thesis}
\text{proof (cases low no \in set (ll ! n))}
\quad \text{case True}
\quad \text{with wf-ll nsll have low no \in set-of pret \land 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 nsll 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 hno-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 hno-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 (erule 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 (erule 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 ino-nNull: low no \neq Null
  with no-nNull no-in-pret-var pret-dag have ino-in-pret: low no \in set-of pret
  apply -
  apply (rule tac low=low in subelem-set-of-low)
  apply auto
  done
from ino-nNull balanced-no have hno-nNull: high no \neq Null
  by simp
with hno-nNull pret-dag ord-pret no-in-pret-var
have var-children-smaller: var (low no) < var no \land var (high no) < var no
  apply -
  apply (rule var-ordered-children)
  apply auto
done
with no-in-pret-var have \( \text{var-lno-l-n}: \text{var} \ (\text{low no}) < n \)
  by simp
with wf-ll lno-in-pret nsll have low no \( \in \) (ll ! (var (low no)))
  by (simp add: wf-ll-def length-ll-eq)
with lno-in-pret var-lno-l-n have low no \( \in \) Nodes n ll
  apply (simp add: Nodes-def)
  apply (rule-tac \( x=\text{var} \) (low no) \( \text{in} \) exI)
  apply simp
done
hence repb (low no) \( \in \) repb \( \cdot \) Nodes n ll
  by simp
with repbNodes-in-Nodes have repb-lno-in-Nodes:
  repb (low no) \( \in \) Nodes n ll
  by blast
from pret-dag wf-ll nsll have Null \( \notin \) Nodes n ll
  apply –
  apply (rule Null-notin-Nodes)
  apply (auto simp add: length-ll-eq)
done
with repb-lno-in-Nodes show repb (low no) \( \neq \) Null
  by auto
qed
have Null-notin-lln: Null \( \notin \) set (ll ! n)
proof (cases Null \( \in \) set (ll ! n))
  case True
  with wf-ll nsll have Null \( \in \) set-of pret \( \land \) var (Null) = n
    by (simp add: wf-ll-def length-ll-eq)
  with pret-dag have Null \( \neq \) Null
    apply –
    apply (rule set-of-nn)
    apply auto
done
  thus \(?thesis\)
    by auto
next
  assume Null \( \notin \) set (ll ! n)
  thus \(?thesis\)
    by simp
qed
have (repb \( \propto \) low) no \( \notin \) 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 \( \neq \) 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 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 nsll have low no ∈ set (ll ! (var (low no)))
  by (simp add: wf-ll-def length-ll-req)
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 nsll 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-req)
done

qed

with no-nNull balanced-no low-no-notin-lln high-no-notin-lln isLeaf-var repb-low-nNull
show no ≠ Null ∧
  (low no = Null) = (high no = Null) ∧
  low no /∈ set (ll ! n) ∧ high no /∈ set (ll ! n) ∧
  isLeaf-pt no low high = (var no ≤ 1) ∧
  (low no ≠ Null ⟹ repb (low no) ≠ Null) ∧
  (repb ∝ low) no /∈ set (ll ! n)
by auto

qed

have all-nodes-same-var: \( \forall \text{no1} \in \text{set}\ (ll \! \! n)\). \( \forall \text{no2} \in \text{set}\ (ll \! \! n)\). \( \text{var no1} = \text{var no2} \)

proof (intro ballI impI)
  fix no1 no2
  assume no1 \in \text{set}\ (ll \! \! n)
  with wf-ll nsll have var-lln-i: \( \text{var no1} = n \)
    by (simp add: wf-ll-def length-ll-eq)
  assume no2 \in \text{set}\ (ll \! \! n)
  with wf-ll nsll have var no2 = n
    by (simp add: wf-ll-def length-ll-eq)
  with var-lln-i show \( \text{var no1} = \text{var no2} \)
    by simp
  qed

have (\( \forall \text{repa}.\ (\forall \text{no}.\ \text{no} \not\in \text{set}\ (ll \! \! n) \rightarrow \text{repb no} = \text{repa no}) \) \land
  (\( \forall \text{no} \in \text{set}\ (ll \! \! n)\). \( \text{repa no} \not= \text{Null} \) \land
    (if \( \text{repa} \propto \text{low} \) then \( \text{repa no} = (\text{repa} \propto \text{high}) \) no \land \text{low no} \not= \text{Null} \) else \( \text{repa no} \in \text{set}\ (ll \! \! n) \) \land
    \( \text{repa (repa no)} = \text{repa no} \) \land
    (\( \forall \text{no1} \in \text{set}\ (ll \! \! n)\). \( ((\text{repa} \propto \text{high}) \text{no1} = (\text{repa} \propto \text{high}) \text{no} \land
      (\text{repa} \propto \text{low}) \text{no1} = (\text{repa} \propto \text{low}) \text{no}) =
      (\text{repa no} = \text{repa no1}))) \rightarrow
    \text{var p} + 1 - (n + 1) < \text{var p} + 1 - n \land
    n + 1 \leq \text{var p} + 1 \land
    (\forall \text{pt i}\. \text{pt} \notin \text{set-of pret} \land \text{var (repa no)} \leq \text{var no} \land
      (\exists \text{not nort}. \text{Dag (repa no) (repa \propto low) (repa \propto high) nort} \land
        \text{Dag no low high nort} \land
        \text{reduced nort} \land
        \text{ordered nort var} \land
        \text{set-of nort} \subseteq \text{repa \ propto \ Nodes (n + 1) ll} \land
        (\forall \text{no} \in \text{set-of nort}. \text{repa no} = \text{no}) \land
        (\exists \text{nobdt}. \text{bdt not var = Some nobdt} \land
          (\exists \text{nordt}. \text{bdt nort var = Some nordt} \land \text{nobdt} \sim \text{nordt})))\))

\land
  (\forall \text{tl t2}. \text{tl} \in \text{Dags (repa \ propto \ Nodes (n + 1) ll) (repa \ propto low) (repa \ propto high)}
  \land
  \text{tl} \in \text{Dags (repa \ propto \ Nodes (n + 1) ll) (repa \ propto low) (repa \ propto high)})
→

isomorphic-dags-eq t1 t2 var)
(is (∀ rec. ?srn-post recp → ?norm-inv recp )
proof (intro allI impl, elim conjE)
fix recp
assume repbc-nc: ∀ no. no /∈ set (ll ! n) → repb no = repc no
assume rep-prop: ∀ 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)))
show ?norm-inv recp
proof 
from n-Suc-var-p have termi: var p + 1 − (n + 1) < var p + 1 − n
by arith
from wf-ll repbc-nc nsll
have Nodes-n-rep-nc: ∀ p. p ∈ Nodes n ll → repb p = repc p
apply −
apply (rule allI)
apply (rule impl)
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:
∀ p. p ∈ Nodes n ll → repb p ∈ Nodes n ll
by auto
from wf-ll nsll have Nodes n ll ⊆ 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: ∀ p. p ∈ Nodes n ll → repb p ∈ set-of pret
apply −
apply (intro allI impl)
apply blast
done
have Nodes-repbc-Dags-eq: ∀ p t. p ∈ Nodes n ll
→ Dag (repb p) (repc ∞ low) (repc ∞ high) t =
Dag (repc p) (repc ∞ low) (repc ∞ high) t

162
proof (intro allI impI)
fix p t
assume p-in-Nodes: p ∈ Nodes n ll
then have repp-nc: repb p = repc p
by (rule Nodes-n-rep-nc [rule-format!])
from p-in-Nodes normalize-prop obtain nort where
nort-repb-dag: Dag (repb p) (repb ∞ low) (repb ∞ high) nort and
nort-in-repbNodes: set-of nort ⊆ repb ' 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 ⊆ 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 p-in-Nodes repbNodes-in-Nodes have repp-nNull: repb p ≠ Null
by auto
from nort-repb-dag repp-nc
have nort-repbc-dag: Dag (repc p) (repb ∞ low) (repb ∞ high) nort
by simp
from nort-in-Nodesn have ∀ x ∈ set-of nort. x ∈ Nodes n ll
apply –
apply (rule ballI)
apply blast
done
with wf-ll nsll have ∀ x ∈ set-of nort. x ∈ set-of pret ∧ var x < n
apply –
apply (rule ballI)
apply (rule wf-ll-Nodes-pret)
apply (auto simp add: length-ll-eq)
done
with pret-dag prebdt-pret nort-repbc-dag ord-pret wf-ll nsll repbc-nc
have ∀ x ∈ set-of nort. (repc ∞ low) x = (repb ∞ low) x ∧
(repc ∞ high) x = (repb ∞ 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) (repc ∞ low) (repc ∞ high) nort
apply –

163
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 (repc p) (repc ∝ low) (repc ∝ high) t =
Dag (repc p) (repc ∝ 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 impI)
apply (erule-tac x=pt in allE)
apply auto
apply (rule-tac x=i in exI)
apply auto
done

have repe-nc:
(∀ pt. pt /∈ set-of pret ∨
(∃ i. n + 1 ≤ i ∧ pt ∈ set (ll ! i) ∧ i < var p + 1)
→ rep pt = repc pt)
proof (intro allI impI)
fix pt
assume pt-notin-lower-ll: pt /∈ set-of pret ∨
\( \exists i. \, n + 1 \leq i \land pt \in \text{set} \,(ll ! i) \land i < var \, p + 1 \)

**show** \( \text{rep} \, pt = \text{repc} \, pt \)

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

**case** \( True \)

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

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

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

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

**apply** (auto)

**done**

**with** \( \text{repbe-nc} \) **have** \( \text{rep} \, pt = \text{repc} \, pt \)

**by** auto

**with** \( \text{Sucn-repb-nc} \, True \) **show** \( \text{thesis} \)

**by** auto

**next**

**assume** \( pt-in-pret: \, \neg \, pt \notin \text{set-of} \, \text{pret} \)

**with** \( \text{pt-notin-lower-ll} \) **have** \( \text{pt-in-higher-ll:} \)

\( \exists i. \, n + 1 \leq i \land pt \in \text{set} \,(ll ! i) \land i < var \, p + 1 \)

**by** simp

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

**apply** –

**apply** (erule exE)

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

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

**done**

**with** \( \text{repbe-nc} \) **have** \( \text{rep} \, pt = \text{repc} \, pt \)

**by** auto

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

**by** auto

**qed**

**from** \( \text{wf-ll} \, nsl \)

**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** (erule no-in-one-ll)

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

**done**

**with** \( \text{repbe-nc} \)

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

**apply** –

**apply** (rule ballI)

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

**apply** simp

**done**

**then** **have** \( \text{repbNodes-repncNodes:} \, \text{repb} \,'(\text{Nodes} \, n \, ll) = \text{repc} \,'(\text{Nodes} \, n \, ll) \)

**apply** –

**apply** rule

165
apply blast
apply rule
apply (erule imageE)
apply (erule-tac x=x in ballE)
pref 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)
  from rep-prop y-in-repclln obtain
    repc-y-Null: 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).
      (repc ∞ high) no1 = (repc ∞ high) y ∧
      (repc ∞ low) no1 = (repc ∞ low) y) = (repc y = repc no1))
  using [[simp-depth-limit = 4]]
  by auto
from wf-ll nsll y-in-repclln obtain
  y-in-pret: y ∈ set-of pret and
\text{vary-n: var } y = n \\
\text{by (auto simp add: wf-ll-def length-ll-eq)}

\text{from y-in-pret pret-dag have y-nNull: } y \neq \text{ Null}
\text{apply --}
\text{apply (rule set-of-nn)}
\text{apply auto}
\text{done}

\text{show } x \in \text{ Nodes } (n + 1) \text{ ll}
\text{proof (cases low } y = \text{ Null)}
\text{case True}
\text{from pret-dag prebd-t-prebdt-pret True y-in-pret}
\text{have highy-Null: high } y = \text{ Null}
\text{apply --}
\text{apply (drule balanced-bdt)}
\text{apply auto}
\text{done}

\text{with share-prop True obtain}
\text{repcy-in-llb: repc } y \in \text{ set } (ll ! n) \text{ and}
\text{rry-ry: repc } (\text{repc } y) = \text{ repc } y \text{ and}
\text{y-other-node-prop: } \forall \text{ no1 } \in \text{ set } (ll ! n).
\text{((repc } \propto \text{ high) no1 } = \text{ (repc } \propto \text{ high) y} \land
\text{ (repc } \propto \text{ low) no1 } = \text{ (repc } \propto \text{ low) y} = \text{ (repc } y = \text{ repc no1)}
\text{by simp}
\text{from repcy-in-llb x-repcy show ?thesis}
\text{by (auto simp add: Nodes-def)}
\text{next}
\text{assume lowy-nNull: low } y \neq \text{ Null}
\text{with pret-dag prebd-t-prebdt-pret y-in-pret}
\text{have highy-nNull: high } y \neq \text{ Null}
\text{apply --}
\text{apply (drule balanced-bdt)}
\text{apply auto}
\text{done}

\text{show ?thesis}
\text{proof (cases (repc } \propto \text{ low) } y = \text{ (repc } \propto \text{ high) y)}
\text{case True}
\text{with red-prop lowy-nNull have repc } y = \text{ (repc } \propto \text{ high) y}
\text{by auto}
\text{with highy-nNull have red-repc-y: repc } y = \text{ repc } (\text{high } y)
\text{by (simp add: null-comp-def)}
\text{from pret-dag ord-pret y-in-pret lowy-nNull highy-nNull}
\text{have var } (\text{low } y) < \text{ var } y \land \text{ var } (\text{high } y) < \text{ var } y
\text{apply --}
\text{apply (rule var-ordered-children)}
\text{apply auto}
\text{done}
\text{with vary-n have varhighy: var } (\text{high } y) < n
\text{by auto}
from y-in-pret y-nNull highy-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 repbNodes-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-lbbn: 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-lbbn 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 nort ∧
   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 ∧
    (∃ norbdt. bdt nort var = Some norbdt ∧ nobdt ∼ norbdt)))
  (is ∀ no ∈ Nodes (n + 1) ll. ?Q i no)
proof (intro ballI)
fix no
assume no-in-Nodes: no ∈ Nodes (n + 1) ll
from wf-ll no-in-Nodes nsll have no-in-pret: no ∈ set-of pret

168
apply (simp add: wf-ll-def Nodes-def length-ll-eq)
apply (erule conjE)
apply (thin-tac ∀ q. q ∈ set-of pret → q ∈ set (ll ! var q))
apply (erule exE)
apply (erule-tac x = k in allE)
apply arith
apply (erule-tac x = no in ballE)
apply auto
done
from pret-dag no-in-pret have nonNull: no ≠ Null
apply -
apply (rule set-of-nn [rule-format])
apply auto
done
show ?Q i no
proof (cases no ∈ Nodes n ll)
  case True
  note no-in-Nodesn=this
  with wf-ll nsll no-in-Nodes
  have no-notin-llbn: no /∈ set (ll ! n)
    apply -
    apply (simp add: Nodes-def length-ll-eq)
    apply (elim exE)
    apply (drule-tac ?i = ka and ?j = n in no-in-one-ll)
    apply arith
    apply simp
    apply auto
done
with repbc-nc have repb-no-eq-repe-no: repb no = repe no
  by simp
from repbc-nc no-in-Nodes no-notin-llbn normalize-prop True
have varrep-eq-var: var (repe no) ≤ var no
  apply -
  apply (erule-tac x = no in ballE)
  prefer 2
  apply simp
  apply (erule-tac x = no in allE)
  apply simp
done
moreover
from True normalize-prop no-in-Nodes obtain not nort where
  nort-dag: Dag (repe no) (repe ∞ low) (repe ∞ high) nort and
  ord-nort: ordered nort var and
  subset-nort-not: set-of nort ⊆ repb '(Nodes n ll) and
  not-dag: Dag no low high not and
  red-nort: reduced nort and
  nort-repb: (∀ no∈set-of nort. repb no = no) and
  bdt-prop: ∃ nobdt norbdt. bdt not var = Some nobdt ∧
\[ \text{bdt nort var} = \text{Some norbdt} \land \text{nobdt} \sim \text{norbdt} \]

by blast

moreover

from \text{Nodesn-notin-lln repbe-nc nort-repb subset-nort-not repbNodes-in-Nodes}

have nort-repc:

\[ (\forall \text{no} \in \text{set-of nort. repc no} = \text{no}) \]

apply auto

apply (subgoal-tac no \in Nodes n ll)

apply fastforce

apply blast

done

moreover

from nort-dag have nortnodesN: \((\forall \text{no. no} \in \text{set-of nort} \rightarrow \text{no} \neq \text{Null})\)

apply –

apply (rule allI)

apply (rule impI)

apply (rule set-of-nn)

apply auto

done

moreover

have Dag (repc no) (repc \prop low) (repc \prop high) nort

proof –

from no-notin-llbn repbe-nc have repbe-no: repc no = repb no

by fastforce

with nort-dag

have nortrepc-dag: Dag (repc no) (repb \prop low) (repb \prop high) nort

by simp

from wf-ll nseqll have Nodes n ll \subseteq set-of pret

apply –

apply (rule Nodes-levellist-subset-t)

apply assumption+

apply (simp add: length-ll-eq)

done

with repbNodes-in-Nodes subset-nort-not

have subset-nort-pret: set-of nort \subseteq set-of pret

by simp

have vxsn-in-pret: \(\forall \text{x} \in \text{set-of nort. var x} < \text{n} \land \text{x} \in \text{set-of pret}\)

proof (rule ballI)

fix x

assume x-in-nort: \(x \in \text{set-of nort}\)

from x-in-nort subset-nort-not repbNodes-in-Nodes

have x \in Nodes n ll

by blast

with wf-ll nseql have xsn: var x < n

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

apply (erule conjE)

apply (thin-tac \(\forall q. q \in \text{set-of pret} \rightarrow q \in \text{set (ll ! var q)}\))
apply (erule exE)
apply (erule tac x=k in allE)
apply (erule impE)
apply arith
apply (erule tac x=x in ballE)
apply auto
done
from x-in-nort subset-nort-pret have x-in-pret: x ∈ set-of pret
by blast
with xsn show var x < n ∧ x ∈ set-of pret by simp
qed
with pret-dag prebdlt-pret nortrepbc-dag ord-pret wf-ll nsll
repbc-nc
have ∀ x ∈ set-of nort. ((repc ∝ low) x = (repb ∝ low) x ∧
(repc ∝ high) x = (repb ∝ high) x)
apply –
apply (rule nort-null-comp)
apply (auto simp add: length-ll-eq)
done
with nort-dag
have Dag (repc no) (repc ∝ low) (repc ∝ high) nort =
Dag (repb no) (repb ∝ low) (repb ∝ high) nort
apply –
apply (rule heaps-eq-Dag-eq)
apply simp
done
with nortrepbc-dag show ?thesis
by simp
qed
moreover
have set-of nort ⊆ repc ‘(Nodes (n + 1) ll)
proof –
have Nodesn-in-NodesSucn: Nodes n ll ⊆ Nodes (n + 1) ll
by (simp add: Nodes-def set-split)
then have repbNodesn-in-repbNodesSucn:
repb ‘(Nodes n ll) ⊆ repb ‘(Nodes (n + 1) ll)
by blast
from wf-ll nsll
have Nodes-n-notin-lln: ∀ no ∈ Nodes n ll, no /∈ set (ll ! n)
apply (simp add: Nodes-def length-ll-eq)
apply clarify
apply (erule no-in-one-ll)
apply auto
done
with repbc-nc have ∀ no ∈ Nodes n ll. repb no = repc no
apply –
apply (rule ballI)
apply (erule-tac x=no in allE)
apply simp

171
done
then have repbNodes-repcNodes:
  repb \( '(\text{Nodes } n \text{ ll}) = \text{repc } '(\text{Nodes } n \text{ ll}) \)
  apply 
  apply rule
  apply blast
  apply rule
  apply (erule imageE)
  apply (erule-tac \( x=xa \) in ballE)
  prefer 2
  apply simp
  apply rule
  apply auto
  done
from Nodesn-in-NodesSucn
have repc \( '(\text{Nodes } n \text{ ll}) \subseteq \text{repc } '(\text{Nodes } (n + 1) \text{ ll}) \)
  by blast
with repbNodes-repeNodes subset-not-not repbNodesn-in-repbNodesSucn

  show \( ?\text{thesis} \) by simp
qed
ultimately show \( ?\text{thesis} \)
  by blast
next
assume \( \text{no} \notin \text{Nodes } n \text{ ll} \)
with no-in-Nodes have no-in-llbn: \( \text{no} \in \text{set} (\text{ll }! \text{ n}) \)
  apply (simp add: Nodes-def)
  apply (erule exE)
  apply (erule-tac \( x=k \) in 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 nsl have varno: \( \text{var no} = n \)
  by (simp add: wf-ll-def length-ll-eq)
from repbc-changes no-in-llbn
have repb-no-changes: \( \text{repc } \text{no} \neq \text{Null} \land \)
  \( (\text{repc } \otimes \text{low} ) \text{ no} = (\text{repc } \otimes \text{high} ) \text{ no} \land \text{low no} \neq \text{Null} \)
  \( \rightarrow \text{repc } \text{no} = (\text{repc } \otimes \text{high} ) \text{ no} \) \land
  \( ((\text{repc } \otimes \text{low} ) \text{ no} = (\text{repc } \otimes \text{high} ) \text{ no} \rightarrow \text{low no} = \text{Null} ) \)
  \( \rightarrow \text{repc } \text{no} \in \text{set} (\text{ll }! \text{ n}) \land \text{repc } (\text{repc } \text{no}) = \text{repc } \text{no} \land \)
  \( (\forall \text{no1} \in \text{set} (\text{ll }! \text{ n}). ((\text{repc } \otimes \text{high} ) \text{no1} = (\text{repc } \otimes \text{high} ) \text{no} \land \)
  \( (\text{repc } \otimes \text{low} ) \text{no1} = (\text{repc } \otimes \text{low} ) \text{no} ) = (\text{repc } \text{no} = \text{repc } \text{no1}) ) \)
  \( \land \text{is } ?\text{monN } \land ?\text{reproduce } \land ?\text{repshare} \)
using [[simp-depth-limit=4]]
by (simp split: split-if)
then obtain
\( rnonN \): \( rnonN \) and
repreduce: \( \text{repreduce} \) and
repshare: \( \text{repshare} \) by blast
have repcn-normalize: \( \text{var} (\text{repc no}) \leq \text{var no} \land \\
(\exists \text{not nort}. \text{Dag} (\text{repc no}) (\text{repc} \propto \text{low}) (\text{repc} \propto \text{high}) \text{ nort} \land \\
\text{Dag no low high not} \land \text{reduced nort} \land \text{ordered nort} \land \\
\text{set-of nort} \subseteq \text{repc} \propto \text{Nodes} (n + 1) l l \land \\
(\forall \text{no} \in \text{set-of} \ \text{nort}. \ \text{repc 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})))
(is \( \text{?varrep} \land \text{?repcn-prop} \) is \( \text{?varrep} \land \\
(\exists \text{not nort}. \ \text{?nort-dag nort} \land \text{?not-dag nort} \land \text{?red nort} \land \\
\text{?ord nort} \land \text{?nort-in-Nodes nort} \land \text{?repcno-no-n nort} \land \text{?bdt-equ not nort}))
proof (cases high no = Null)
case True
note highnoNull=this
with pret-dag prebdt-pret no-in-pret
have lownoNull: \( \text{low no} = \text{Null} \)
apply –
apply (drule balanced-bdt)
apply assumption+
apply simp
done
with repshare have repcnoinlln: \( \text{repc no} \in \text{set} (ll ! n) \)
by simp
with wf-ll nsll have varrno-n: \( \text{var} (\text{repc no}) = n \)
by (simp add: wf-ll-def length-ll-eq)
with varrno have varrepr: \( \text{?varrep} \)
by simp
from wf-ll nsll no-in-llbn varrno-n
have varrno-varno: \( \text{var} (\text{repc no}) = \text{var no} \)
by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsll repcnoinlln
have rno-in-pret: \( \text{repc no} \in \text{set-of} \ \text{pret} \)
by (simp add: wf-ll-def length-ll-eq)
from repcnoinlln repshare lownoNull
have reprep-eq-rep: \( \text{repc} (\text{repc no}) = \text{repc no} \)
by simp
with repcnoinlln repshare lownoNull
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
have repcn-prop: \( \text{?repcn-prop} \)

173
apply
apply (rule-tac x=(Node Tip no Tip) in exI)
apply (rule-tac x=(Node Tip (repc no) Tip) in exI)
apply (intro conjI)
apply simp
prefer 3
apply simp
prefer 3
apply simp

proof
from pret-dag pnN rno-in-pret have rnonN: repc no ≠ Null
apply (case-tac repc no = Null)
apply auto
done
from highnoNull repchildreneq have rhhighNull: (repc ∝ high) (repc no) = Null
  by (simp add: null-comp-def)
from lownoNull repchildreneq have rlowlNull: (repc ∝ low) (repc no) = Null
  by (simp add: null-comp-def)
with rhhighNull rnonN
show repc no ≠ Null ∧ (repc ∝ low) (repc no) = Null ∧
  (repc ∝ high) (repc no) = Null
  by simp
next
from nonNull lownoNull highnoNull
show ?not-dag (Node Tip no Tip)
  by simp
next
from no-in-Nodes
show set-of (Node Tip (repc no) Tip) ⊆ repc ' Nodes (n + 1) ll
  by simp
next
show ∀ no∈set-of (Node Tip (repc no) Tip). repc no = no
proof
  fix pt
  assume pt-in-repcLeaf: pt ∈ set-of (Node Tip (repc no) Tip)
  with reprep-eq-rep show repc pt = pt
    by simp
qed
next
show ?bdt-equ (Node Tip no Tip) (Node Tip (repc no) Tip)
proof (cases var no = 0)
  case True
  note vno-Null=this
  then have nobdt: bdt (Node Tip no Tip) var = Some Zero by simp
  from vartep vno-Null have vartno: var (repc no) = 0 by simp
  then have norbdt: bdt (Node Tip (repc no) Tip) var = Some Zero
    by simp

174
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 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) ≤
      by (metis set-of-subdag)
    with prebdt-pret have ∃ bdt2. bdt (Node Tip no Tip) var = Some
      by (metis balanced-bdt)
    qed
  qed
with varrep reprep-eq-rep show ?thesis by simp
next
  assume hno-nNull: high no ≠ Null
  with pret-dag prebdont-pret no-in-pret have hno-nNull:
    low no ≠ Null
    by (metis balanced-bdt)
  have hno-pret: high no ∈ set-of pret
    by (metis subelem-set-of-high)
  with wf-ll have hno-in-l: high no ∈ set (ll ! (var (high no)))
    by (simp add: wf-ll-def)
  from pret-dag ord-pret no-in-pret hno-nNull hno-nNull
    have varhnos-varno: var (high no) < var no
      by (metis var-ordered-children)
  with varno have varhnos-n: var (high no) < n
    by simp
  with hno-in-l have hno-in-Nodesn:
    high no ∈ Nodes n ll

175
apply (simp add: Nodes-def)
apply (rule-tac x=var (high no) in exI)
apply simp
done
from wf-ll nsll hno-in-ll varhnos-n
have high no ∉ set (ll ! n)
  apply –
  apply (rule no-in-one-ll)
  apply (auto simp add: length-ll-eq)
  done
with repbc-nc have repb-repc-high: repb (high no) = repc (high no) by simp

with normalize-prop hno-in-Nodesn varhnos-varno varno
have high-normalize: var (repc (high no)) ≤ var (high no) ∧
  (∃ not nort. Dag (repc (high no)) (repb ∞ low) (repb ∞ high) nort ∧
  Dag (high no) low high not ∧ reduced nort ∧
  ordered nort var ∧ set-of nort ⊆ repb ' (Nodes n ll) ∧
  (∀ no∈set-of nort.. repb no = no) ∧
  (∃ nobdt norbdt. bd not var = Some nobdt ∧ bd not nort var =
  Some norbdt ∧ nobdt ~ norbdt))
(is ?varrep-high ∧
  (∃ not nort.. ?rephigh-dag nort ∧ ?high-dag not ∧
  ?redhigh nort ∧ ?ordhigh nort ∧ ?rephigh-in-Nodes nort ∧
  ?rephno-no nort ∧ ?highdd-prop not nort)
  is ?varrep-high ∧ ?not-nort-prop)
apply simp
apply (erule-tac x=high no in ballE)
apply (simp del: Dag-Ref)
apply simp
done
then have varrep-high: ?varrep-high by simp
from varhnos-n varrep-high have varrephno-s-n:
  var (repc (high no)) < n
  by simp
from Nodes-subset
have Nodes n ll ⊆ Nodes (Suc n) ll
  by auto
with hno-in-Nodesn repcNodes-in-Nodes
have repc (high no) ∈ Nodes (Suc n) ll
  apply simp
  apply blast
  done
with wf-ll nsll have repc (high no) ∈ set-of pret
  apply (simp add: wf-ll-def Nodes-def length-ll-eq)
  apply (elim conjE exE)
  apply (thin-tac ∀ q. q ∈ set-of pret → q ∈ set (ll ! var q))
  apply (erule-tac x=k in allE)
  apply (erule impE)
  apply simp

176
apply (erule-tac \( x = \text{repc \{high \ no\} \ in \ \text{ballE} \))
apply auto
done

with \( \text{wf-ll varrephno-s-n} \)
have \( \exists k < n. \text{repc \{high \ no\} \in \ set \ (ll \! \! n) \) \)
by (auto simp add: \( \text{wf-ll-def} \))

with \( \text{wf-ll nsll} \) have \( \text{repc \{high \ no\} \notin \ set \ (ll \! \! n) \)
apply \-
apply (erule \( \text{exE} \))
apply (erule-tac \( \text{rule-tac \ i = k \ and \ j = n \ in \ no-in-one-ll} \))
apply (auto simp add: \( \text{length-ll-eq} \))
done

with \( \text{repbc-nc} \)
have \( \text{rephigh-idem: reph \{repc \{high \ no\}\} = repc \{repc \{high \ no\}\} \)
by auto

from \( \text{high-normalize} \)
have \( \text{not-nort-prop-high: \text{?not-nort-prop by (simp del: Dag-Ref) \) \}
from \( \text{not-nort-prop-high} \) obtain \( \text{hnot where high-dag: \text{?high-dag hnot} \)
by auto
from \( \text{wf-ll nsll} \)
have \( \forall no \in \text{Nodes n ll}. no \notin \set \ (ll \! \! n) \)
apply (simp add: \( \text{Nodes-def length-ll-eq} \))
apply clarify
apply (drule no-in-one-ll)
apply auto
done

with \( \text{repbc-nc} \) have \( \forall no \in \text{Nodes n ll}. \text{reph no = repc no} \)
apply \-
apply (rule \( \text{ballI} \))
apply (erule-tac \( x = no \ in \ \text{allE} \))
apply simp
done

then

have \( \text{rephNodes-repcNodes:} \)
\( \text{reph \{'(Nodes n ll)\} = repc \{'(Nodes n ll)\} \)
apply \-
apply rule
apply blast
apply rule
apply (erule imageE)
apply (erule-tac \( x = xa \ in \ \text{ballE} \))
prefer 2
apply simp
apply rule
apply auto
done

then have \( \text{repcNodes-rephNodes:} \)
\( \text{repc \{'(Nodes n ll)\} = repb \{'(Nodes n ll)\} \)
by simp
from pret-dag nsll wf-ll
have null-notin-Nodesn: Null $\notin$ Nodes n ll
  apply –
  apply (rule Null-notin-Nodes)
  apply (auto simp add: length-ll-eq)
done
from hno-in-Nodesn have repec (high no) $\in$ repec ‘(Nodes n ll)
  by blast
with repbNodes-in-Nodes repecNodes-repbNodes
have repec (high no) $\in$ Nodes n ll
  apply simp
  apply blast
done
with null-notin-Nodesn have rhn-nNull: repec (high no) $\neq$ Null
  by auto

from no-in-pret nonNull lno-nNull pret-dag
have lno-in-pret: low no $\in$ set-of pret
  by (rule subelem-set-of-low)
with wf-ll
have lno-in-ll: low no $\in$ set (ll ! (var (low no)))
  by (simp add: wf-ll-def)
from pret-dag ord-pret no-in-pret lno-nNull hno-nNull
have varlnos-varno: var (low no) < var no
  apply –
  apply (drule var-ordered-children)
  apply assumption+
  apply auto
done
with varno have varlnos-n: var (low no) < n by simp
with lno-in-ll have lno-in-Nodesn: low no $\in$ Nodes n ll
  apply (simp add: Nodes-def)
  apply (rule-tac x=var (low no) in exI)
  apply simp
done
from varlnos-n wf-ll nssl lno-in-ll
have low no $\notin$ set (ll ! n)
  apply –
  apply (rule no-in-one-ll)
  apply (auto simp add: length-ll-eq)
done
with repbc-nc have repbc-repc-low: repb (low no) = repec (low no) by
  simp
with normalize-prop lno-in-Nodesn varlnos-varno varno
have low-normalize: var (repec (low no)) $\leq$ var (low no) ∧
(∃ not nort. Dag (repec (low no)) (repb $\propto$ low) (repb $\propto$ high) nort ∧
Dag (low no) low high not ∧ reduced nort ∧ ordered nort var ∧
set-of nort ⊆ repb \( '(\text{Nodes } n ll) \land \)
\((\forall no \in \text{set-of nort} . \text{repb no } = \text{no}) \land \)
\( (\exists \text{nobdt nobdt}. \text{bdt not var } = \text{Some nobdt} \land \text{bdt nort var } = \text{Some} \)
\)
\n\( \text{nordt } \land \)
\( \text{nordt } \sim \text{nobdt} (\)\n\)
\( (\exists \text{not nort}. \text{repb\text{-}below-dag nort } \land \text{?low\text{-}dag not } \land \text{?red\text{-}high nort } \land \)
\( \text{?ord\text{-}high nort } \land \text{?replow\text{-}in\text{-}Nodes nort } \land \text{?low\text{-}repno\text{-}no nort } \land \)
\( \text{?low\text{-}dd\text{-}prop not nort} ) \)
\( (\exists \text{not\text{-}prop\text{-}low } \land \text{?not\text{-}nort\text{-}prop\text{-}low}) \)
\( \text{apply simp} \)
\( \text{apply (erule-tac x } = \text{low no in ballE)} \)
\( \text{apply (simp del: Dag\text{-}Ref)} \)
\( \text{apply simp} \)
\( \text{done} \)
\( \text{then have varrep\text{-}low: ?varrep\text{-}low by simp} \)
\( \text{from low\text{-}normalize have not\text{-}nort\text{-}prop\text{-}low: ?not\text{-}nort\text{-}prop\text{-}low} \)
\( \text{by (simp del: Dag\text{-}Ref)} \)
\( \text{from lno\text{-}in\text{-}Nodes have repc (low no) } \in \text{repc } '(\text{Nodes } n ll) \)
\( \text{by blast} \)
\( \text{with repb\text{\text{-}Nodes\text{-}in\text{-}Nodes repc\text{\text{-}Nodes\text{-}repb\text{\text{-}Nodes}}} \)
\( \text{have repc (low no) } \in \text{Nodes } n ll \)
\( \text{apply simp} \)
\( \text{apply blast} \)
\( \text{done} \)
\( \text{with null\text{-}notin\text{-}Nodes have rln\text{-}\text{-}nNull: repc (low no) } \neq \text{Null} \)
\( \text{by auto} \)

\text{show ?thesis} \)
\text{proof (cases repc (low no) } = \text{repc (high no))} \)
\text{case True} \)
\text{note red\text{-}case } = \text{this} \)
\text{with repreduce lno\text{-}nNull hno\text{-}nNull} \)
\text{have rno\text{-}eq\text{-}hrno: repc no } = \text{repc (high no)} \)
\text{by (simp add: null\text{-}comp\text{-}def)} \)
\text{from varhnos\text{-}varno rno\text{-}eq\text{-}hrno varrep\text{-}high have varrep: ?varrep by simp} \)
\text{from not\text{-}nort\text{-}prop\text{-}high not\text{-}nort\text{-}prop\text{-}low have repc\text{-}prop: ?repc\text{-}prop} \)
\text{apply –} \)
\text{apply (elim exE)} \)
\text{apply (rename-tac rnot lnot rnot lnot lnot rnot)} \)
\text{apply (rule\text{-}tac x } = (\text{Node lnot no rnot}) \text{ in exI)} \)
\text{apply (rule\text{-}tac x } = \text{rnot rnot}) \text{ in exI)} \)
\text{apply (elim conjE)} \)
\text{apply (intro conjI)} \)
\text{prefer 7} \)
\text{apply (elim exE)} \)
\text{apply (rename\text{-}tac rnot lnot rnot lnot rnot no rnot rnot rnot rnot)} \)

179
apply (elim conjE)
apply (case-tac Suc 0 < var no)
apply (rule-tac x=(Bdt-Node lnobdt (var no) rnobdt) in exI)
apply (rule conjI)
prefer 2
apply (rule-tac x=rnorbdt in exI)
apply (rule conjI)

proof –
fix rnot lnot rnort lnort
assume highnort-dag:
  Dag (repc (high no)) (repb ∝ low) (repb ∝ high) rnort
assume ord-nort: ordered rnort var
assume rnort-in-repNodes: set-of rnort ⊆ repb ' Nodes n ll
from rnort-in-repNodes repbNodes-in-Nodes
have nort-in-Nodes: set-of rnort ⊆ Nodes n ll
  by blast
from varhnos-n varrep-high
have vrhnos-n: var (repc (high no)) < n by simp
from rhn-nNull highnort-dag
have ∃ lno rno. rnort = Node lno (repc (high no)) rno by fastforce
with highnort-dag rhn-nNull have root rnort = repc (high no) by auto

with ord-nort have ∀ x ∈ set-of rnort. var x < var (repc (high no))

apply –
apply (rule ballI)
apply (drule ordered-set-of)
apply auto
done
with vrhnos-n have vxsn: ∀ x ∈ set-of rnort. var x < n
  by fastforce
from nort-in-Nodes have ∀ x ∈ set-of rnort. x ∈ Nodes n ll
  by auto
with wf-ll nsll
have x-in-pret: ∀ x ∈ set-of rnort. x ∈ set-of pret
  by auto
with vrhnos-n have vxsn: ∀ x ∈ set-of rnort. var x < n
  by fastforce
from x-in-pret have vxsn-in-pret: ∀ x ∈ set-of rnort. var x <n ∧ x ∈ set-of pret
  by auto
with pret-dag prebdt-pret highnort-dag ord-pret wf-ll nsll
  repbc-nc
have ∀ x ∈ set-of rnort. (repc ∝ low) x = (repb ∝ low) x ∧
  (repc ∝ high) x = (repb ∝ high) x
  by auto
with (repc ∝ low) x = (repb ∝ low) x ∧
  (repc ∝ high) x = (repb ∝ high) x
apply –
apply (rule nort-null-comp)
apply (auto simp add: length-ll-eq)
done
with rno-eq-hrno
have Dag (repc no) (repc low) (repc high) rnot =
  Dag (repc no) (repb low) (repb high) rnot
  apply
  apply (rule heaps-eq-Dag-eq)
  apply simp
done
with highnort-dag rno-eq-hrno
show Dag (repc no) (repc low) (repc high) rnot by simp
next
  fix rnot lnot rnot lnort
  assume lnot-dag: Dag (low no) low high lnot
  assume rnot-dag: Dag (high no) low high rnot
  with lnot-dag nonNull
  show Dag no low high (Node lnot no rnot) by simp
next
  fix rnot lnot rnot lnort
  assume reduced rnot
  then show reduced rnot by simp
next
  fix rnot
  assume ordered rnot var
  then show ordered rnot var by simp
next
  fix rnot
  assume rnot-in-Nodes: set-of rnot \subseteq repb ' Nodes n ll
  have Nodes n ll \subseteq Nodes (n + 1) ll
    by (simp add: Nodes-def set-split)
    then have repc ' Nodes n ll \subseteq repc ' Nodes (n + 1) ll
      by blast
      with rnot-in-Nodes repbNodes-repcNodes
      show set-of rnot \subseteq repc ' Nodes (n + 1) ll
        by (simp add: Nodes-def)
next
  fix rnot rnorbd t
  assume bdt rnorbd var = Some rnorbd t
  then show bdt rnorbd var = Some rnorbd t by simp
next
  fix rnot lnot rnorbd t lnorbd t inorbd t
  assume rcongeval: rnorbd t \sim inorbd t
  assume lnot-dag: Dag (repc (low no)) (repb low) (repb high) lnort
  assume rnorbd-def: bdt lnort var = Some inorbd t
  assume rnorbd-def: bdt rnorbd var = Some rnorbd t
  assume lcongeval: lnorbd t \sim inorbd t

181
from read-case lnort-dag rnot-dag
have lnort-rnort: lnort = rnot
  by (simp add: Dag-unique del: Dag-Ref)
with lnobdt-def lcongeval rnobdt-def
have lnobdt-rnorbdt: lnobdt ~ rnorbdt by simp
with rcongeval have lnobdt ~ rnobdt
  apply --
  apply (rule cong-eval-trans)
  apply (auto simp add: cong-eval-sym)
  done
then have Bdt-Node lnobdt (var no) rnobdt ~ rnobdt
  apply --
  apply (simp add: cong-eval-sym [rule-format])
  apply (rule cong-eval-child-high)
  apply assumption
  done
with rcongeval show Bdt-Node lnobdt (var no) rnobdt ~ rnorbdt
  apply --
  apply (rotate-tac 1)
  apply (rule cong-eval-trans)
  apply auto
  done
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 lnobdt-def lnot-dag rnot-dag lnobdt-def
show bdt (Node lnot no rnot) var =
  Some (Bdt-Node lnobdt (var no) rnobdt)
  by simp
next
fix rnot lnot lnort rnot rnor lnobdt rnobdt
assume lnobdt-def: bdt lnot var = Some lnobdt
assume rnot-rnor: rnor ~ rnobdt
assume lnobdt-def: bdt rnot var = Some rnor
assume lnot-dag: Dag (low no) low high lnot
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 lnot no rnot) var = Some nobdt ∧
  (∃ rnorbdt. bdt rnot var = Some rnorbdt ∧ nobdt ~ rnorbdt)
proof (cases var no = 0)
  case True
  note vnoNull=this
  with pret-dag ord-pret no-in-pret hno-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 lnno-nNull hno-nNull
  vnoOne
  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 lnno-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 noOne 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-repb-Nodesn: set-of rnort ⊆ repb ' Nodes n ll
assume rnort-repb-no: ∀no∈set-of rnort. repb no = no
from repbNodes-in-Nodes rnort-in-repb-Nodesn
have rnort-in-Nodesn: set-of rnort ⊆ Nodes n ll
  by blast
show ∀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-no pt-in-rnort show repc pt = pt
      by auto
  qed
  qed
with varrep show ?thesis by simp
next
assume share-case-cond: repc (low no) ≠ repc (high no)
with lno-nNull hno-nNull
have share-case-cond-propto: (repc ∝ low) no ≠ (repc ∝ high) no
  by (simp add: null-comp-def)
with repshare obtain
  rno-in-llbn: repc no ∈ set (ll ! n) and
  rno-eq-rno: repc (repc no) = repc no and

184
\begin{verbatim}
twonodes-in-llbn-prop: (∀ no1∈set (ll ! n)),
((repc ∞ high) no1 = (repc ∞ high) no ∧
(repc ∞ low) no1 = (repc ∞ low) no) = (repc no = recp no1)
by auto
from wf-ll rno-in-llbn nsll
have varrepno-n: var (repc no) = n
by (simp add: wf-ll-def length-ll-eq)
by simp
from not-nort-prop-high not-nort-prop-low have repcn-prop: ?repcn-prop
apply
apply (elim exE)
apply (rename-tac rnot lnot rnort lnort)
apply (rule-tac x=Node lnot no rnot in exI)
apply (rule-tac x=Node lnort (repc no) rnort in exI)
apply (elim conjE)
apply (intro conjI)
prefer 7
apply (elim exE)
apply (rename-tac rnot lnot rnort lnort)
apply (elim conjE)
apply (case-tac Suc 0 < var no)
apply (rule-tac x=(Bdt-Node inobdt (var no) rnobdt) in exI)
apply (rule conjI)
prefer 2
apply (rule-tac x=(Bdt-Node inorbdt (var (repc no)) rnorbdt) in exI)
apply (rule conjI)
proof –
fix rnot lnot rnort lnort
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 rnort-in-repNodes: set-of rnort ⊆ repb ' Nodes n ll
assume lnort-in-repNodes: set-of lnort ⊆ repb ' Nodes n ll
from rnort-in-repNodes repbNodes-in-Nodes
have rnort-in-Nodes: set-of rnort ⊆ Nodes n ll
  by simp
from lnort-in-repNodes repbNodes-in-Nodes
have lnort-in-Nodes: set-of lnort ⊆ Nodes n ll
  by simp
from rnort-in-Nodes
have rnortx-in-Nodes: ∀ x ∈ set-of rnort. x ∈ Nodes n ll
  by blast
with wf-ll nsll
have ∀ x ∈ set-of rnort. x ∈ set-of pret ∧ var x < n
  apply –
  apply (rule ballI)
\end{verbatim}
apply (rule wf-ll-Nodes-pret)
apply (auto simp add: length-ll-eq)
done
with pret-dag prebdt-pret rnort-dag ord-pret wf-ll nsll
  repbc-nc
have \( \forall x \in \text{set-of} \text{rnort}. (\text{repc} \propto \text{low}) x = (\text{repb} \propto \text{low}) x \land
  (\text{repc} \propto \text{high}) x = (\text{repb} \propto \text{high}) x \)
  apply -
  apply (rule nort-null-comp)
  apply (auto simp add: length-ll-eq)
done
then have \( \text{Dag} (\text{repc} (\text{high no})) (\text{repc} \propto \text{low}) (\text{repc} \propto \text{high}) \text{rnort} = \)
apply -
apply (rule heaps-eq-Dag-eq)
apply assumption
done
with rnort-dag
have rnort-dag-repc:
  \( \text{Dag} (\text{repc} (\text{high no})) (\text{repc} \propto \text{low}) (\text{repc} \propto \text{high}) \text{rnort} = \)
by simp
from lnort-in-Nodes
have lnortx-in-Nodes: \( \forall x \in \text{set-of} \text{lnort}. x \in \text{Nodes n ll} \)
  by blast
with wf-ll nsll
have \( \forall x \in \text{set-of} \text{lnort}. x \in \text{set-of} \text{pret} \land \text{var} x < n \)
  apply -
  apply (rule ballI)
  apply (rule wf-ll-Nodes-pret)
  apply (auto simp add: length-ll-eq)
done
with pret-dag prebdt-pret lnort-dag ord-pret wf-ll nsll
  repbc-nc
have \( \forall x \in \text{set-of} \text{lnort}. (\text{repc} \propto \text{low}) x = (\text{repb} \propto \text{low}) x \land
  (\text{repc} \propto \text{high}) x = (\text{repb} \propto \text{high}) x \)
  apply -
  apply (rule nort-null-comp)
  apply (auto simp add: length-ll-eq)
done
then have
\( \text{Dag} (\text{repc} (\text{low no})) (\text{repc} \propto \text{low}) (\text{repc} \propto \text{high}) \text{lnort} = \)
\( \text{Dag} (\text{repc} (\text{low no})) (\text{repb} \propto \text{low}) (\text{repb} \propto \text{high}) \text{lnort} = \)
apply -
apply (rule heaps-eq-Dag-eq)
apply assumption
done
with lnort-dag
have lnort-dag-repc:
Dag (repc (low no)) (repc \(\propto\) low) (repc \(\propto\) high) lnort
by simp
from lno-nNull hno-nNull
have propo-comp: (repc \(\propto\) low) no = repc (low no) \(\wedge\)
(repc \(\propto\) high) no = repc (high no)
by (simp add: null-comp-def)
from rno-in-lbn twonodes-in-lbn-prop rrno-eq-rno
have (repc \(\propto\) high) (repc no) = (repc \(\propto\) high) no \(\wedge\)
(repc \(\propto\) low) (repc no) = (repc \(\propto\) low) no
by simp
with propo-comp lnort-dag-repc rnort-dag-repc lno-nNull hno-nNull
rnort
show Dag(repc no)(repc \(\propto\) low)(repc \(\propto\) high)(Node lnort (repc no))
by auto
next
fix rnot lnot rnort 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 rnot lnort
assume rnot-dag:
Dag (repc (high no)) (repb \(\propto\) low) (repb \(\propto\) high) lnort
assume lnort-dag:
Dag (repc (low no)) (repb \(\propto\) low) (repb \(\propto\) high) lnort
assume red-rnort: reduced rnort
assume red-lnort: reduced lnort
from rhn-nNull rnort-dag obtain lnort rrnort where
rnort-Node: rnort = (Node lnort (repc (high no)) rrnort)
by auto
from rhn-nNull lnort-dag obtain llnort rlnort where
lnort-Node: lnort = (Node llnort (repc (low no)) rlnort)
by auto
from twonodes-in-lbn-prop rrno-eq-rno rno-in-lbn hno-nNull
lno-nNull
have ((repc \(\propto\) high) (repc no)) = repc (high no) \(\wedge\)
((repc \(\propto\) low) (repc no)) = repc (low no)
apply –
apply (erule-tac x=repc no in ballE)
apply (auto simp add: null-comp-def)
done
with share-case-cond
have ((repc \(\propto\) high) (repc no)) \(\neq\) ((repc \(\propto\) low) (repc no))
by auto
with red-lnort red-rnort rrnort-Node lnort-Node share-case-cond
show reduced (Node lnort (repc no) rnort)
apply −
apply (rule-tac lp=repc (low no) and rp=repc (high no) and
llt=lnort and rlt = rlort and lrt=lrnort and rrt=rrnort
in reduced-children-parent)
apply auto
done
next
fix inort 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 rnort-in-repNodes: set-of lnort \subseteq \text{repb} '\text{Nodes n ll}
assume rnort-in-repNodes: set-of rnort \subseteq \text{repb} '\text{Nodes n ll}
from lnort-in-repNodes repbNodes-in-Nodes
have lnort-in-Nodes: set-of lnort \subseteq \text{Nodes n ll}
  by simp
from rnort-in-repNodes repbNodes-in-Nodes
have rnort-in-Nodes: set-of rnort \subseteq \text{Nodes n ll}
  by simp

from rhn-nNull rnort-dag obtain lnort rrnort where
  rnort-Node: rnort = (Node lrnort (repc (high no)) rrnort)
  by auto
from rhn-nNull lnort-dag obtain llnort rlnort where
  lnort-Node: lnort = (Node llnort (repc (low no)) rlnort)
  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\ \text{Nodes n ll}
  by blast
with wf-ll nsll
have vrlnosn: \text{var} (\text{repc} (\text{low no})) < n
  apply −
  apply (drule wf-ll-Nodes-pre)
  apply (auto simp add: length-ll-eq)
  done
from rnort-dag rhn-nNull rnort-in-Nodes
have repc (high no) \in\ set-of rnort
  by auto
with rnort-in-Nodes
have repc (high no) \in\ \text{Nodes n ll}
  by blast
with wf-ll nsll have vrlnosn: \text{var} (\text{repc} (\text{high no})) < n
  apply −
apply (erule wf-ll-Nodes-pret)
apply (auto simp add: length-ll-eq)
done

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

next

fix rnot lnot rnort lnobdt lnobdt lnorbdt lnorbdt
assume lnobdt-def: bdt lnot var = Some lnobdt
assume rnobdt-def: bdt rnot var = Some rnobdt
assume rnorbdt-def: bdt rnort var = Some rnorbdt
assume cong-rno-rnor: rnobdt ∼ rnorbdt
assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume ¬ Suc 0 < var no
then have varnoseq1: var no = 0 ∨ var no = 1 by auto
show ∃ nobdt. bdt (Node lnort 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 varNull=this
with pret-dag ord-pret no-in-pret lno-null hno-null
show ?thesis
  apply –
apply (drule var-ordered-children)
apply auto
done

next
assume var no \neq 0
with varnoseq1 have vnoOne: var no = 1 by simp
from pret-dag ord-pret no-in-pret hno-nNull hno-nNull
vnoOne
have vlvrNull: var (low no) = 0 \land var (high no) = 0
apply -
apply (drule var-ordered-children)
apply auto
done
then have vlNull: var (low no) = 0 by simp
from vlnull have vrNull: var (high no) = 0 by simp
from lnobdt-def lnot-dag vlnull lno-nNull
have lnobdt-Zero: lnobdt = Zero
apply -
apply (drule bdt-Some-var0-Zero)
apply auto
done
from rnobdt-def rnot-dag vrnull hno-nNull
have rnobdt-Zero: rnobdt = Zero
apply -
apply (drule bdt-Some-var0-Zero)
apply auto
done
from lnobdt-Zero lnobdt-def
have bdt lnot var = Some Zero by simp
with lnot-dag vlnull
have lnot-Node: lnot = (Node Tip (low no) Tip)
by auto
from rnobdt-Zero rnobdt-def rnot-dag vrnull
have rnot-Node: rnot = (Node Tip (high no) Tip)
by auto
from pret-dag no-in-pret obtain not
where not-ex: Dag no low high not
apply -
apply (drule dag-setof-exD)
apply auto
done
with pret-dag no-in-pret have not-ex-in-pret: not <= pret
apply -
apply (rule set-of-subdag)
apply auto
done
from not-ex lnot-dag rnot-dag nonNull
have not-def: not = (Node lnot no rnot)
by (simp add: Dag-unique del: Dag-Ref)
with not-ex-in-pret prebdt-pret
have nobdt-ex: \( \exists \) nobdt. bdt (Node lnot no rnot) var = Some nobdt
  apply –
  apply (rule subbdt-ex)
  apply auto
  done
then obtain nobdt where
  nobdt-def: bdt (Node lnot no rnot) var = Some nobdt by auto
from not-def have root not = no by simp
with nobdt-def nmaOne not-def
have not = (Node Tip no Tip)
  apply –
  apply (drule bdt-Some-var1-One)
  apply auto
  done
with not-def have rnot = Tip by simp
with rnot-Node show ?thesis by simp
qed
next
fix lnot rnot lnobdt rnobdt
assume lnot-dag: Dag (low no) low high lnot
assume rnot-dag: Dag (high no) low high rnot
assume lnobdt-def: bdt lnot var = Some lnobdt
assume rnobdt-def: bdt rnot var = Some rnobdt
assume onesvarno: Suc 0 < var no
with rnobdt-def lnot-dag rnot-dag lnobdt-def
show bdt (Node lnot no rnot) var =
  Some (Bdt-Node lnobdt (var no) rnobdt) by simp
next
fix rnot lnot rnot lnobdt rnobdt lnobdt rnorbd lnorbdt
assume rnot-dag:
  Dag (repc (high no)) (repa \times low) (repa \times high) rnot
assume lnort-dag:
  Dag (repc (low no)) (repa \times low) (repa \times high) lnort
assume lnorbdt-def: bdt lnort var = Some lnorbdt
assume lnorbd-def: bdt lnort var = Some lnorbd
assume varno-bOne: Suc 0 < var no
with varno have Suc 0 < n by simp
with varrepno-bOne have Suc 0 < var (repc no) by simp
with rnorbd-def inorbdt-def
show bdt (Node lnort (repc no) rnort) var =
  Some (Bdt-Node lnorbd (var (repc no)) rnorbd)
  by simp
next
fix rnobdt lnobdt rnorbd lnorbdt
assume lcong-eval: lnobdt \sim lnorbdt
assume rcong-eval: rnobdt \sim rnorbd
from varno varrepno-bOne have var (repc no) = var no by simp
with lcong-eval rcong-eval
show Bdt-Node lnobdt (var no) rnobdt ~
Bdt-Node lnorbdt (var (repc no)) rnorbdt
apply (unfold cong-eval-def)
apply (rule ext)
by simp

next
fix rnnot lnnot rnort lnort
assume lnnot-repb: \( \forall \ no \in \text{set-of} \ lnort. \ \text{repb} \ no = \ no \)
assume rnor-not-b: \( \forall \ no \in \text{set-of} \ rnort. \ \text{repb} \ no = \ no \)
assume rnor-in-repb-Nodesn: set-of rnort \( \subseteq \ \text{repb} \ \text{Nodes} \ n \ \text{ll} \)
assume lnnot-in-repb-Nodesn: set-of lnort \( \subseteq \ \text{repb} \ \text{Nodes} \ n \ \text{ll} \)
from repbNodes-in-Nodes rnor-in-repb-Nodesn
have rnor-in-Nodesn: set-of rnort \( \subseteq \ \text{Nodes} \ n \ \text{ll} \)
by blast
from repbNodes-in-Nodes lnnot-in-repb-Nodesn
have lnnot-in-Nodesn: set-of lnort \( \subseteq \ \text{Nodes} \ n \ \text{ll} \)
by blast
have rnor-repb: \( \forall \ no \in \text{set-of} \ rnort. \ \text{repb} \ no = \ no \)
proof
fix pt
assume pt-in-rnort: \( pt \in \text{set-of} \ rnort \)
with rnor-in-Nodesn have pt \( \in \ \text{Nodes} \ n \ \text{ll} \)
by blast
with Nodesn-notin-lnl have pt \( \notin \ \text{set} \ (\text{ll} \! \n) \)
by auto
with repbc-nc have repb pt = repc pt
by auto
with rnor-repb pt-in-rnort show repc pt = pt
by auto
qed
have lnnot-repb: \( \forall \ no \in \text{set-of} \ lnort. \ \text{repc} \ no = \ no \)
proof
fix pt
assume pt-in-lnort: \( pt \in \text{set-of} \ lnort \)
with lnnot-in-Nodesn have pt \( \in \ \text{Nodes} \ n \ \text{ll} \)
by blast
with Nodesn-notin-lnl have pt \( \notin \ \text{set} \ (\text{ll} \! \n) \)
by auto
with repbc-nc have repb pt = repc pt
by auto
with lnnot-repb pt-in-lnort show repc pt = pt
by auto
qed
show \( \forall \ no \in \text{set-of} \ (\text{Node} \ lnort \ (\text{repc} \ no) \ \text{rnort}). \ \text{repc} \ no = \ no \)
proof
fix pt
assume pt-in-rept: \( pt \in \text{set-of} \ (\text{Node} \ lnort \ (\text{repc} \ no) \ \text{rnort}) \)
show repc pt = pt
proof (cases pt \( \in \text{set-of} \ lnort \))
case True
with lnort-repc show ?thesis
  by auto
next
assume pt-notin-lnort: pt $\notin$ set-of lnort
show ?thesis
proof (cases pt $\in$ set-of rnort)
case True
with rnort-repc show ?thesis
  by auto
next
assume pt-notin-rnort: pt $\notin$ 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 rnonN show ?thesis by simp
qed
note while-while-prop=this
from wf-ll nsll
have $\forall$ no $\in$ Nodes n ll. no $\notin$ set (ll ! n)
  apply (simp add: Nodes-def length-ll-eq)
  apply clarify
  apply (erule no-in-one-ll)
  apply auto
  done
with repbc-nc have $\forall$ no $\in$ Nodes n ll. repb no = repc no
  apply —
  apply (rule ballI)
  apply (erule-tac x=no in allE)
  apply simp
  done
then have repbNodes-repcNodes:
  repb ‘(Nodes n ll) = repc ‘(Nodes n ll)
  apply —
  apply rule
  apply blast
  apply rule
  apply (erule imageE)
  apply (erule-tac x=xa in ballE)
prefer 2
apply simp
apply rule
apply auto
done
then have repcNodes-repbNodes:
  repc ('(Nodes n ll) = repb ('(Nodes n ll)
by simp
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 rt 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
  apply (elim conjE)
  apply (erule imageE)
  apply auto
done
from q-in-Nodes have repb q = repc 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 \( \text{Dag} \ (\text{repb} \ q) \ (\text{repb} \succeq \text{low}) \ (\text{repb} \succeq \text{high}) \ t = \n\)
\( \text{Dag} \ (\text{repc} \ q) \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ t \)
by (rule Nodes-repbc-Dags-eq [rule-format])
with t-dag rq-p have \( \text{Dag} \ (\text{repb} \ q) \ (\text{repb} \succeq \text{low}) \ (\text{repb} \succeq \text{high}) \ t \) by simp
with repbq-p x-t show \( \text{Dag} \ p \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ x \)
by simp
next
fix x p t
assume t-in-repcNodes: set-of t \subseteq repb ' N ode s n ll
assume x-t: x=t
with t-in-repcNodes repbNodes-repcNodes
show set-of x \subseteq repc ' N ode s n ll
by simp
next
fix x p t
assume t-in-repcNodes: set-of t \subseteq repb ' N ode s n ll
assume t-dag: \( \text{Dag} \ p \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ t \)
assume t-nTip: t \neq Tip
assume x-t: x=t
from t-nTip t-dag have p \neq Null
apply 
apply (case-tac p=Null)
apply auto
done
with t-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 \in Nodes n ll
apply simp
apply (elim conjE)
apply (erule imageE)
apply auto
done
from q-in-Nodes have repb q = repc q
by (rule Nodes-n-rep-nc [rule-format])
with rq-p have repbq-p: repc q = p by simp
from q-in-Nodes
have \( \text{Dag} \ (\text{repc} \ q) \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ t = \n\)
\( \text{Dag} \ (\text{repc} \ q) \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ t \)
by (rule Nodes-repbc-Dags-eq [rule-format])
with t-dag rq-p have \( \text{Dag} \ (\text{repc} \ q) \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ t \) by simp
with repbq-p x-t show \( \text{Dag} \ p \ (\text{repc} \succeq \text{low}) \ (\text{repc} \succeq \text{high}) \ x \)
by simp
next
fix x p and t :: dag
assume x-t: x = t
assume t-nTip: t \neq Tip
with \( x-t \) show \( x \neq \text{Tip} \) by simp

next

fix \( x \) \( p \) and \( t :: \text{dag} \)
assumption \( x-t : x = t \)
assumption \( t-n\text{Tip} : t \neq \text{Tip} \)
with \( x-t \) show \( x \neq \text{Tip} \) by simp

qed

from \text{pret-dag uf-ll nsll}

have \text{null-notin-Nodes-Suc-n} : \text{Null} \notin \text{Nodes} (\text{Suc} n) \ll
  by (\text{rule Null-notin-Nodes, auto simp add: length-ll-eq})

{ fix \( t1 \) \( t2 \)
  assume \( t1\text{-in-DagsNodesn} \):
    \( t1 \in \text{Dags} (\text{repc ' Nodes n ll}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) \)
  assume \( t2\text{-notin-DagsNodesn} \):
    \( t2 \notin \text{Dags} (\text{repc ' Nodes n ll}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) \)
  assume \( t2\text{-in-DagsNodesSucn} \):
    \( t2 \in \text{Dags} (\text{repc ' Nodes (\text{Suc} n) ll}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) \)
  assume \text{isomorphic-dags-eq-asm}:
    \( \forall t1 \ t2. \ t1 \in \text{Dags} (\text{repb ' Nodes n ll}) (\text{repb } \propto \text{low}) (\text{repb } \propto \text{high}) \)
    \( \rightarrow \text{isomorphic-dags-eq} t1 \ t2 \ var \)
  assume \text{rephc-Dags}:
    \( \text{Dags} (\text{repc ' Nodes n ll}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) = \)
    \( \text{Dags} (\text{repb ' Nodes n ll}) (\text{repb } \propto \text{low}) (\text{repb } \propto \text{high}) \)
  from \( t1\text{-in-DagsNodesn} \ \text{rephc-Dags} \)
  have \( t1\text{-rephc-subnode} \):
    \( t1 \in \text{Dags} (\text{repb ' Nodes n ll}) (\text{repb } \propto \text{low}) (\text{repb } \propto \text{high}) \)
    by simp
  from \( t2\text{-in-DagsNodesSucn} \)
  have \( t2\text{-in-DagsNodesSucn} \):
    \( t2 \in \text{Dags} (\text{repc ' Nodes (\text{Suc} n) ll}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) \)
    by simp
  from \text{rephNodes-in-Nodes} \ \text{rephNodes-repcNodes} 
  have \text{rephNodes-in-Nodesn} : \text{repc 'Nodes n ll} \subseteq \text{Nodes n ll}
    by simp
  from \( t1\text{-in-DagsNodesn} \) obtain \( q \) where
    \( \text{Dag-q-Nodes-n} \):
    \( \text{Dag} (\text{repc q}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) t1 \land q \in \text{Nodes n ll} \)
  proof (elim \text{Dags.cases})
  fix \( p \) \( t \)
  assume \( t1-t : t1 = t \)
  assume \( t\text{-in-repcNodesn} \): set-of \( t \subseteq \text{repc ' Nodes n ll} \)
  assume \( t\text{-dag} : \text{Dag} p (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) t \)
  assume \( t\text{-nTip} : t \neq \text{Tip} \)
  assume \text{obtain-prop} : \( \forall q. \text{Dag} (\text{repc q}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) t1 \land q \in \text{Nodes n ll} \rightarrow \text{thesis} \)
  from \( t\text{-nTip} \) \( t\text{-dag} \) have \( p \neq \text{Null} \)
    apply
    apply (case_tac \( p=\text{Null} \))
apply auto
done
with t-nTip t-dag obtain lt rt where t-Node: t\(=\)Node lt p rt by auto
from t-in-repcNodesn t-dag t-nTip t-Node obtain k where
rk-p: repek = p and k-in-Nodes: k \(\in\) Nodes n ll
apply simp
apply (elim conjE)
apply (erule imageE)
apply auto
done
with tl-t t-dag obtain-prop rk-p k-in-Nodes show \(?\)thesis
by auto
qed
with wf-ll nsll haveq-sn: (var q < n)
apply (simp add: Nodes-def)
apply (elim conjE)
apply (erule exE)
apply (simp add: wf-ll-def length-ll-eq)
apply (elim conjE)
apply (thin-tac \(\forall\) q. q \(\in\) set-of pret \(\rightarrow\) q \(\in\) set (ll ! var q))
apply (erule-tac x=k in allE)
apply auto
done
from Dag-q-Nodes-n have q-in-Nodesn: q \(\in\) Nodes n ll
by simp
then have \(\exists\) k<\(\)n. q \(\in\) set (ll ! k)
by (simp add: Nodes-def)
with wf-ll nsll have q \(\notin\) set (ll ! n)
apply –
apply (erule exE)
apply (rule-tac i=k and j=n in no-in-one-ll)
apply (auto simp add: length-ll-eq)
done
with repbc-nc have repbc-q: repe q = 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-repe-q: var (repe q) \(\leq\) var q
by simp
with var-q-sn have var-repe-q-n: var (repe q) < n
by simp

197
from Nodes-subset Dag-q-Nodes-n while-while-prop
have ord-t1-var-rep: ordered t1 var ∧ 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 ⊆ 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 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:
  ∀ no ∈ Nodes (Suc n) ll.
  var (repc no) <= var no
  by auto
from while-while-prop have rep-rep-nort:
∀ no∈Nodes (n + 1) ll. (∃ nort. Dag (repc no) (repc ∋ low) (repc ∋ high) nort ∧
(∀ no∈set-of nort. repc no = no))
by auto
from repcNodes-in-Nodes null-notin-Nodes-Suc-n
have ∀ no ∈ Nodes (n+1) ll. repc no ≠ Null
by auto
with repc-rep-nort have ∀ no ∈ Nodes (n+1) ll. repc (repc no) = (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 ∈ 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 ∀ no∈set-of nort. repc no = no)
apply auto
done
with t2-in-DagsNodesSucn t2-notin-DagsNodesn ord-t2 while-prop-part
wf-ll nsll repcNodes-in-Nodes obtain a where
t2-repc-dag: Dag (repc a) (repc ∋ low) (repc ∋ high) t2 and
a-in-lnl: a ∈ set (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 ∈ set-of pret
by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsll a-in-lnl have vara-n: var a = n
by (simp add: wf-ll-def length-ll-eq)
from a-in-lnl rep-prop obtain
repp-nNull: repc a ≠ Null and
repp-reduce: (repc ∋ low) a = (repc ∋ high) a ∧ low a ≠ Null
→ repc a = (repc ∋ high) a and
repp-share: ((repc ∋ low) a = (repc ∋ high) a → low a = Null)
→ repc a ∈ set (ll ! n) ∧
repc (repc a) = repc a ∧
(∀ no1∈set (ll ! n). ((repc ∋ high) no1 = (repc ∋ high) a ∧
(repc ∋ low) no1 = (repc ∋ low) a) = (repc a = repc no1))
using [[simp-depth-limit=4]]
by auto
from t2-repc-dag a-in-lln repp-nNull obtain lt2 rt2 where
t2-Node: t2 = (Node lt2 (repc a) rt2)
by auto
have isomorphic-dags-eq t1 t2 var
proof (cases (repc ∞ low) a = (repc ∞ high) a ∧ low a ≠ Null)
case True
note red = this
then have red-case: (repc ∞ low) a = (repc ∞ high) a
by simp
from red have low-nNull: low a ≠ Null
by simp
with pret-dag prebdt-pret a-in-pret have highp-nNull: high a ≠ Null
apply –
apply (drule balanced-bdt)
apply auto
done
from pret-dag ord-pret a-in-pret low-nNull highp-nNull
have var-children-smaller: var (low a) < var a ∧ var (high a) < var a
apply –
apply (rule var-ordered-children)
apply auto
done
from pret-dag a-in-pret have a-nNull: a ≠ Null
apply –
apply (rule set-of-nn [rule-format])
apply auto
done
with a-in-pret highp-nNull pret-dag have high a ∈ set-of pret
apply –
apply (drule subelem-set-of-high)
apply auto
done
with wf-ll have high a ∈ set (ll ! (var (high a)))
by (simp add: wf-ll-def)
with a-in-lln t2-repc-dag var-children-smaller vara-n
have ∃ k<n. (high a) ∈ set (ll ! k)
by auto
then have higha-in-Nodesn: (high a) ∈ 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
rt-in-repbNort: set-of rt ⊆ repb ' Nodes n ll
apply –
apply (erule-tac x=high a in ballE)
apply auto
done
from rt-in-repbNort repbNodes-repcNodes
have \textit{rt-in-repcNodesn}: set-of rt \subseteq \textit{repc 'Nodes n ll} \\
by blast 
from \textit{rt-dag higha-in-Nodesn} 
have \textit{repcrt-dag}: \textit{Dag (repc (high a)) (repc \propto low) (repc \propto high) rt} 
apply – 
apply (drule Nodes-repbe-Dags-eq [rule-format]) 
apply auto 
done 
have \textit{rt-nTip}: rt \neq \textit{Tip} 
proof – 
have \textit{repc (high a)} \neq \textit{Null} 
proof – 
note \textit{rhiga-in-rNodesn} 
also have \textit{repc 'Nodes n ll} \subseteq \textit{repc 'Nodes (Suc n) ll} 
using Nodes-subset 
by blast 
also have \ldots \subseteq \textit{Nodes (Suc n) ll} 
using repcNodes-in-Nodes 
by simp 
finally show \textit{\textit{?thesis}} 
using null-notin-Nodes-Suc-n 
by auto 
qed 
with \textit{repcrt-dag} show \textit{\textit{?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 \textit{wf-ll} have low a \in set (\textit{ll} ! (\textit{var} (low a))) 
by (simp add: wf-ll-def) 
with \textit{a-in-lin} \textit{t2-repbe-dag var-children-smaller vara-n} 
have \exists k<n. (low a) \in set (\textit{ll} ! k) 
by auto 
then have (low a) \in \textit{Nodes n ll} 
by (simp add: Nodes-def) 
then have \textit{low-in-rNodesn}: \textit{repc (low a) \in repc ' Nodes n ll} 
by simp 
show \textit{\textit{?thesis}} 
proof – 
from \textit{repp-reduce low-nNull highp-nNull red-case} 
have \textit{repc-p-def}: \textit{repc a = repc (high a)} 
by (simp add: null-comp-def) 
with \textit{rt-in-repcNodesn repcrt-dag rhiga-in-rNodesn a-in-lin t2-repbe-dag} 
\textit{repc-p-def} \textit{rt-nTip} 
have \textit{t2-in-Dags- Nodesn}: 
\textit{t2} \in \textit{Dags (repc ' Nodes n ll) (repc \propto low) (repc \propto high)} 
apply – 

201
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-repe-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 bd1
  assume bd-t1: bd1 t1 var = Some bd1
  assume bd-t2: bd2 t2 var = Some bd1
  show t1 = t2
  proof (cases t1)
    case Tip
    with bd-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 bd-t1 bd-t2 t1-Node have var (repc q) = var (repc a)
      apply –
      apply (rule same-bdt-var)
      apply auto
  qed

202
done
  with var-repc-q-n var-repc-a-n show ?thesis
  by simp
qed
qed
qed
}

note mixed-Dags-case = this

from repbc-dags-eq isomorphic-dags-eq

have days-shared:
  \forall t1 t2. t1 \in Dags (repc \ Nodes (Suc n) ll) (repc \ low) (repc \ high) \land
t2 \in Dags (repc \ Nodes (Suc n) ll) (repc \ low) (repc \ 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-DagsNodesn:
    t1 \in Dags (repc \ Nodes n ll) (repc \ low) (repc \ high)
  assume t2-in-DagsNodesn:
    t2 \in Dags (repc \ Nodes n ll) (repc \ low) (repc \ high)
  assume isomorphic-dags-eq-asm:
    \forall t1 t2. t1 \in Dags (repb \ Nodes n ll) (repb \ low) (repb \ high) \land
t2 \in Dags (repb \ Nodes n ll) (repb \ low) (repb \ high)
    \rightarrow isomorphic-dags-eq t1 t2 var
  assume repbc-Dags:
    Dags (repc \ Nodes n ll) (repc \ low) (repc \ high) =
    Dags (repb \ Nodes n ll) (repb \ low) (repb \ high)
  with t1-in-DagsNodesn t2-in-DagsNodesn 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 \ low) (repc \ high)
  assume t2-notin-DagsNodesn:
    t2 \notin Dags (repc \ Nodes n ll) (repc \ low) (repc \ high)
  assume t2-in-DagsNodesSucn:
    t2 \in Dags (repc \ Nodes (Suc n) ll) (repc \ low) (repc \ high)
  assume isomorphic-dags-eq-asm:
    \forall t1 t2. t1 \in Dags (repb \ Nodes n ll) (repb \ low) (repb \ high) \land
t2 \in Dags (repb \ Nodes n ll) (repb \ low) (repb \ high)
    \rightarrow isomorphic-dags-eq t1 t2 var
  assume repbc-Dags:
    Dags (repc \ Nodes n ll) (repc \ low) (repc \ high) =
    Dags (repb \ Nodes n ll) (repb \ low) (repb \ high)
  from t1-in-DagsNodesn t2-notin-DagsNodesn t2-in-DagsNodesSucn
    isomorphic-dags-eq-asm repbc-Dags
  show isomorphic-dags-eq t1 t2 var
  apply –

203
apply (rule mixed-Dags-case)  
apply auto  
done

next

fix $t_1$ $t_2$

assume $t_1$-$in$-$Dags$-$Nodes$-$Sucn$:
$t_1 \in Dags \ (\text{repc ' Nodes } (\text{Suc } n) \ \text{ll} \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}))$

assume $t_1$-$notin$-$Dags$-$Nodes$-$n$:
$t_1 \notin Dags \ (\text{repc ' Nodes } n \ \text{ll} \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}))$

assume $t_2$-$in$-$Dags$-$Nodes$-$Sucn$:
$t_2 \in Dags \ (\text{repc ' Nodes } (\text{Suc } n) \ \text{ll} \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}))$

assume $t_2$-$notin$-$Dags$-$Nodes$-$n$:
$t_2 \notin Dags \ (\text{repc ' Nodes } n \ \text{ll} \ (\text{repc } \propto \text{low}) \ (\text{repc } \propto \text{high}))$

from $t_1$-$in$-$Dags$-$Nodes$-$Sucn$ have ord-$t_1$: ordered $t_1$ var

proof (elim Dags.cases)

fix $p$ $t$

assume t-in-repcNodes: set-of $t$ $\subseteq$ repc ' Nodes (Suc $n$) ll
assume t-nTip: $t \neq \text{Tip}$
assume t2t: $t_1 = t$

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

from t-in-repcNodes have x-in-repcNodesSucn:
$\forall x. \ x \in \text{set-of } t \longrightarrow x \in \text{repc ' Nodes } (\text{Suc } n) \ \text{ll}$

apply -
apply auto

then have $p \in \text{set-of } t$

with t-nTip t-dag obtain lt rt where t-Node: $t = \text{Node } lt \ p \ rt$

by auto

then have $p \in \text{set-of } t$

by auto

with x-in-repcNodesSucn have $p \in \text{repc ' Nodes } (\text{Suc } n) \ \text{ll}$

by simp

then obtain $a$ where
repa-p: $p = \text{repc } a$ and a-in-NodesSucn: $a \in \text{Nodes } (\text{Suc } n) \ \text{ll}$

by auto

with repca-p while-while-prop t-dag t2t show ?thesis

apply -
apply (erule-tac $x = a$ in ballE)
apply (elim conjE exE)
apply (subgoal-tac nort = $t$)
prefer 2
apply (simp add: Dag-unique)
apply auto
done

qed

from while-while-prop
have while-prop-part: \( \forall \, no \in Nodes \, (\text{Suc } n) \, ll. \)
  \( var \, (\text{repc } no) \leq var \, no \)
  by auto

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

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

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

nsll repcNodes-in-Nodes obtain a1 where
t1-repc-dag: \( \text{Dag } (\text{repc } a1) \, (\text{repc } \propto \text{low}) \, (\text{repc } \propto \text{high}) \, t1 \, \text{and} \)
a1-in-lhn: \( a1 \in (ll ! n) \)
  apply –
  apply (drule restrict-root-Node)
  apply (auto simp add: length-ll-eq)
done

with wf-ll nsll have a1-in-pret: \( a1 \in \text{set-of } \text{pret} \)
  by (simp add: wf-ll-def length-ll-eq)
from wf-ll nsll a1-in-lhn have var1-n: \( var \, a1 = n \)
  by (simp add: wf-ll-def length-ll-eq)
from a1-in-lhn rep-prop obtain
repa1-nNull: \(\text{repe} a1 \neq \text{Null} \) and
repa1-reduce: \((\text{repe} \propto \text{low}) \ a1 = (\text{repe} \propto \text{high}) \ a1 \wedge \text{low \ a1} \neq \text{Null} \)
\(\rightarrow \text{repe} a1 = (\text{repe} \propto \text{high}) \ a1 \) and
repa1-share: \((\text{repe} \propto \text{low}) \ a1 = (\text{repe} \propto \text{high}) \ a1 \rightarrow \text{low \ a1} = \text{Null} \)
\(\rightarrow \text{repe} a1 \in \text{set} (\text{ll} ! n) \wedge \text{repe} (\text{repe} a1) = \text{repe} a1 \wedge \exists \text{a1} \in \text{set} (\text{ll} ! n). ((\text{repe} \propto \text{high}) \ a1 = (\text{repe} \propto \text{high}) \ a1 \wedge (\text{repe} \propto \text{low}) \ a1 = (\text{repe} \propto \text{low}) \ a1) = (\text{repe} a1 = \text{repe} a1)\)
using \([\text{simp-depth-limit} = 4]\)
by \(\text{auto}\)
from \((1\text{-repc-dag} \ a1\text{-in-lln} \ \text{repa1-nNull} \ \text{obtain} \ lt1 \ rt1 \ \text{where}\)
\(\ t1\text{-Node} : t1 = (\text{Node} \ lt1 (\text{repe} a1) \ rt1)\)
by \(\text{auto}\)

from \((2\text{-in-DagsNodesSucn} \ \text{have} \ \text{ord-}t2 : \text{ordered} \ t2 \ \text{var} \)
proof (elim Dags.cases)
fix \(p\ t\)
assume \(t\text{-in-repeNodes} : \text{set-of} \ t \subseteq \text{repe} \ \text{\ 'Nodes} (\text{Suc} \ n) \ \text{ll} \)
assume \(t\text{-nTip} : t \neq \text{Tip} \)
assume \(t2t : t2 = t\)
assume \(t\text{-dag} : \text{Dag} p (\text{repe} \propto \text{low}) (\text{repe} \propto \text{high}) t\)
from \(t\text{-in-repeNodes} \)
have \(x\text{-in-repeNodesSucn} : \)
\(\forall \ x. \ x \in \text{set-of} \ t \rightarrow x \in \text{repe} \ \text{\ 'Nodes} (\text{Suc} \ n) \ \text{ll} \)
apply 
apply \(\text{auto}\)
done
from \(t\text{-nTip} \ t\text{-dag} \ \text{have} \ p \neq \text{Null} \)
apply 
apply \((\text{case-tac} \ p=\text{Null})\)
apply \(\text{auto}\)
done
with \(t\text{-nTip} \ t\text{-dag} \ \text{obtain} \ lt \ rt \ \text{where} \ t\text{-Node} : t=\text{Node} \ lt \ p \ rt \)
by \(\text{auto}\)
then have \(p \in \text{set-of} \ t\)
by \(\text{auto}\)
with \(x\text{-in-repeNodesSucn} \ \text{have} \ p \in \text{repe} \ \text{\ 'Nodes} (\text{Suc} \ n) \ \text{ll} \)
by \(\text{simp}\)
then obtain \(a\ \text{where}\)
\(\text{repe}\ a-p : p=\text{repe} a \ \text{and} \ a\text{-in-NodesSucn} : a \in \text{Nodes} (\text{Suc} \ n) \ \text{ll} \)
by \(\text{auto}\)
with \(\text{repe}\ a-p \ \text{while-while-prop} \ t\text{-dag} \ t2t \ \text{show} \ ?\text{thesis} \)
apply 
apply \((\text{erule-tac} \ x=a \ \text{in} \ \text{ballE})\)
apply \((\text{elim conjE} \ \text{exE})\)
apply \((\text{subgoal-tac} \ \text{nort} = t)\)
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 repcNodes-in-Nodes 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 pret-dag prebdt-pret a1-in-pret have higha1-nNull: high a1 ̸= Null
apply –
apply (drule balanced-bdt)
apply auto
done
from pret-dag ord-pret a1-in-pret lowa1-nNull higha1-nNull
have var-children-smaller-a1: var (low a1) < var a1 ∧ var (high a1) <
var a1
apply –
apply (rule var-ordered-children)
apply auto
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 (rule 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-ll r1-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 r1 where
  r1-dag: Dag (repb (high a1)) (repb ∝ low) (repb ∝ high) r1 and
  r1-in-repbNort: set-of r1 ⊆ repb ' Nodes n ll
apply —
apply (erule-tac x=high a1 in ballE)
apply auto
done
from r1-in-repbNort repbNodes-repcNodes
have r1-in-repcNodesn: set-of r1 ⊆ repc ' Nodes n ll
  by blast
from r1-dag higha1-in-Nodesn
have repc-r1-dag: Dag (repc (high a1)) (repc ∝ low) (repc ∝ high) r1
  apply —
  apply (drule Nodes-repbe-Dags-eq [rule-format])
  apply auto
done
have r1-nTip: r1 ≠ 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
      by blast
    also have ... ⊆ Nodes (Suc n) ll
      using repcNodes-in-Nodes
      by simp
  finally show ?thesis
    using null-notin-Nodes-Suc-n

208
by auto
qed
with repcr1-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 repcr1-dag rhiga1-in-rNodesn a1-in-lhn
t1-repc-dag repc-a1-def rt1-nTip
have t1-in-Dags-Nodessn:
  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 prebdtt-pret a2-in-pret have higha2-nNull: high a2 ≠ Null
  apply –
  apply (drule balanced-bdt)
  apply auto
done
from pret-dag ord-pret a2-in-pret lowa2-nNull higha2-nNull
have var-children-smaller-a2:
  var (low a2) < var a2 ∧ var (high a2) < var a2
  apply –
  apply (rule var-ordered-children)
  apply auto
done
from pret-dag a2-in-pret have a2-nNull: a2 ≠ Null
  apply –
  apply (rule set-of-nn [rule-format])
  apply auto
done
with a2-in-pret higha2-nNull pret-dag have high a2 ∈ set-of pret
  apply –
  apply (drule subelem-set-of-high)
  apply auto
done
with wf-ll have high a2 ∈ set (ll ! (var (high a2)))
by \((\text{simp add: } \text{wf-ll-def})\)

with \(\text{a2-in-lln } t2\)-\(\text{repc-dag var-children-smaller-a2 vara2-n}\)

have \(\exists \, k < n. \ (\text{high } a2) \in \text{set}(ll ! k)\)

by \(\text{auto}\)

then have \(\text{high2-in-Nodesn: } (\text{high } a2) \in \text{Nodes n ll}\)

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

then have \(\text{rhigh2-in-\text{rNodesn}: } \text{repc}(\text{high } a2) \in \text{repc }\ \text{Nodes n ll}\)

by \(\text{simp}\)

from \(\text{high2-in-Nodesn normalize-prop obtain } rt2\) where

\(rt2\)-\(\text{dag: } \text{Dag}(\text{repb (high } a2)) (\text{repb } \propto \text{ low}) (\text{repb } \propto \text{ high}) rt2 \text{ and}\)

\(rt2\)-\(\text{in-\text{rNodesNort: set-of } rt2 \subseteq \text{repc }\ \text{Nodes n ll}\)

apply –

apply \((\text{erule-tac } x=\text{high } a2 \text{ in ballE})\)

apply \(\text{auto}\)

done

from \(rt2\)-\(\text{dag high2-in-Nodesn}\)

have \(\text{repcrt2-dag: } \text{Dag}(\text{repc (high } a2)) (\text{repc } \propto \text{ low}) (\text{repc } \propto \text{ high}) rt2\)

apply –

apply \((\text{drule Nodes-repc-Dags-eq }[\text{rule-format}])\)

apply \(\text{auto}\)

done

have \(rt2\)-\(\text{nTip: } rt2 \neq \text{Tip}\)

proof –

have \(\text{repc (high } a2) \neq \text{Null}\)

proof –

note \(\text{rhigh2-in-\text{rNodesn}}\)

also have \(\text{repc }\ \text{Nodes n ll} \subseteq \text{repc }\ \text{Nodes (Suc n) ll}\)

using \(\text{Nodes-subset}\)

by \(\text{blast}\)

also have \(\ldots \subseteq \text{Nodes (Suc n) ll}\)

using \(\text{repcNodes-in-Nodes}\)

by \(\text{simp}\)

finally show \(\text{?thesis}\)

using \(\text{null-notin-Nodes-Suc-n}\)

by \(\text{auto}\)

qed

with \(\text{repcrt2-dag } \text{show } \text{?thesis by } \text{auto}\)

qed

from \(\text{repa2-reduce lowa2-nNull higha2-nNull } t2\)-\(\text{red-case}\)

have \(\text{repc-a2-def: } \text{repc a2} = \text{repc (high } a2)\)

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

with \(\text{t2-in-\text{rNodesn repcrt2-dag high2a-in-\text{rNodesn a2-in-lln}} t2\)-\(\text{repc-dag repc-a2-def } t2\)-\(\text{nTip}\)

have \(\text{t2-in-Dags-Nodesn:}\)

\(t2 \in \text{Dags}(\text{repc }\ \text{Nodes n ll}) (\text{repc } \propto \text{ low}) (\text{repc } \propto \text{ high})\)

apply –

210
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:
\neg ((\text{repc} \propto \text{low}) a2 = (\text{repc} \propto \text{high}) a2 \land \text{low} a2 \neq \text{Null})
from t1-in-Dags-Node n t2-notin-Dags-Node n t2-in-Dags-Node Suc n
isomorphic-dags-eq repbc-dags-eq
show thesis
apply –
apply (rule mixed-Dags-case)
apply auto
done

qed

next
assume t1-share-cond:
\neg ((\text{repc} \propto \text{low}) a1 = (\text{repc} \propto \text{high}) a1 \land \text{low} a1 \neq \text{Null})
with repa1-share obtain
\text{repa1-in-llbn}: \text{repc} a1 \in \text{set (ll } n) \text{ and}
\text{reprepa1}: \text{repc (repc} a1) = \text{repc} a1 \text{ and}
\text{two nodes-llbn-a1}:
(\forall \text{no1} \in \text{set (ll } n). ((\text{repc} \propto \text{high}) \text{no1} = (\text{repc} \propto \text{high}) a1 \land
(\text{repc} \propto \text{low}) \text{no1} = (\text{repc} \propto \text{low}) a1) = (\text{repc} a1 = \text{repc} \text{no1}))
using [[simp-depth-limit=2]]
by auto
show thesis
proof (cases (\text{repc} \propto \text{low}) a2 = (\text{repc} \propto \text{high}) a2 \land \text{low} a2 \neq \text{Null})
case True
note t2-red-cond=thi s
with t2-red-cond have t2-red-case: (\text{repc} \propto \text{low}) a2 = (\text{repc} \propto \text{high}) a2
by simp
from t2-red-cond have lowa2-nNull: low a2 \neq \text{Null}
by simp
with \text{pret-dag} \text{prebdt-pret a2-in-pret} have higha2-nNull: high a2 \neq \text{Null}
apply –
apply (drule balanced-bdt)
apply auto
done

from \text{pret-dag} \text{ord-pret a2-in-pret lowa2-nNull higha2-nNull}
have \text{var-children-smaller-a2}:
\text{var (low} a2) < \text{var} a2 \land \text{var (high} a2) < \text{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-ln ln 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 (repc (high a2)) (repc ∝ low) (repc ∝ high) rt2
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 report2-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 rt2-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
have t2-in-Dags-Nodesn:
  t2 ∈ Dags (repc ' Nodes a ll) (repc ∞ low) (repc ∞ high)
  apply –
  apply (rule DagsI)
  apply simp
  apply (subgoal-tac t2 = rt2)
  apply (auto simp add: Dag-unique)
  done
from t2-in-Dags-Nodesn t1-notin-DagsNodesn t1-in-DagsNodesSucn
have isomorphic-dags-eq repbc-dags-eq
apply –
apply (rule mixed-Dags-case)
apply auto
done
then show ?thesis
  by (simp add: isomorphic-dags-eq-sym)
next
assume t2-shared-cond:
  ¬ ((repc ∞ low) a2 = (repc ∞ high) a2 ∧ low a2 ≠ Null)
with repa2-share obtain
repcrt2-in-llbn: repc a2 ∈ set (ll ! n) and
reprepa2: repc (repc a2) = repc a2 and
twonodes-llbn-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=2]]
by auto
from twonodes-llbn-a2 a1-in-lln
have share-a1-a2:
  ((repc ∞ high) a1 = (repc ∞ high) a2 ∧
   (repc ∞ low) a1 = (repc ∞ low) a2) = (repc a2 = repc a1)
  by auto
from twonodes-llbn-a1 repea1-in-llbn reprepa1
have children-repc-eq-a1: (repc ∞ high) (repc a1) = (repc ∞ high) a1
  ∧
  (repc ∞ low) (repc a1) = (repc ∞ low) a1
by auto
from twonodes-llbn-a2 repea2-in-llbn reprea2
have children-repe-eq-a2: (repc \propto\ high) (repc a2) = (repc \propto\ high) a2
∧
(repc \propto\ low) (repc a2) = (repc \propto\ low) a2
by auto
from t1-Node t2-Node show thesis
proof (clarsimp simp add: isomorphic-dags-eq-def)
fix bdt1
assume t1-bdt: bdt (Node lt1 (repc a1) rt1) var = Some bdt1
assume t2-bdt: bdt (Node lt2 (repc a2) rt2) var = Some bdt1
show lt1 = lt2 ∧ repe a1 = repe a2 ∧ rt1 = rt2
proof (cases bdt1)
case Zero
with t1-bdt t1-Node obtain
lt1-Tip: lt1 = Tip and
rt1-Tip: rt1 = Tip
by simp
from Zero t2-bdt t2-Node obtain
lt2-Tip: lt2 = Tip and
rt2-Tip: rt2 = Tip
by simp
from t1-repe-dag t1-Node lt1-Tip have (repc \propto\ low) (repc a1) = Null
by simp
with children-repe-eq-a1
have repe-low-a1-Null: (repc \propto\ low) a1 = Null
by simp
from t1-repe-dag t1-Node rt1-Tip
have (repc \propto\ high) (repc a1) = Null
by simp
with children-repe-eq-a1
have repe-high-a1-Null: (repc \propto\ high) a1 = Null
by simp
from t2-repe-dag t2-Node lt2-Tip have (repc \propto\ low) (repc a2) = Null
by simp
with children-repe-eq-a2
have repe-low-a2-Null: (repc \propto\ low) a2 = Null
by simp
from t2-repe-dag t2-Node rt2-Tip
have (repc \propto\ high) (repc a2) = Null
by simp
with children-repe-eq-a2
have repe-high-a2-Null: (repc \propto\ high) a2 = Null
by simp
with share-a1-a2 repe-low-a1-Null repe-high-a1-Null
repe-low-a2-Null repe-high-a2-Null
have repe a2 = repe a1

214
by auto

with l1-Tip r1-Tip l2-Tip r2-Tip show \(?thesis
by auto

next

case One

with t1-bdt t1-Node obtain

l1-Tip: l1 = Tip and
r1-Tip: r1 = Tip
by simp

from One t2-bdt t2-Node obtain

l2-Tip: l2 = Tip and
r2-Tip: r2 = Tip
by simp

from t1-repc-dag t1-Node l1-Tip have (repc \propto low) (repc a1) = Null
by simp

with children-repc-eq-a1
have recp-low-a1-Null: (repc \propto low) a1 = Null
by simp

from t1-repc-dag t1-Node r1-Tip have (repc \propto high) (repc a1) = Null
by simp

with children-repc-eq-a1
have recp-high-a1-Null: (repc \propto high) a1 = Null
by simp

from t2-repc-dag t2-Node l2-Tip have (repc \propto low) (repc a2) = Null
by simp

with children-repc-eq-a2
have recp-low-a2-Null: (repc \propto low) a2 = Null
by simp

from t2-repc-dag t2-Node r2-Tip have (repc \propto high) (repc a2) = Null
by simp

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

with share-a1-a2 recp-low-a1-Null recp-high-a1-Null
repc-low-a2-Null recp-high-a2-Null
have recp a2 = recp a1
by auto

with l1-Tip r1-Tip l2-Tip r2-Tip show \(?thesis
by auto

next

case (Bdt-Node lbdt v rbdt)

note bdt-Node-case=\this

with t1-bdt t1-Node obtain

lbdt-def-lt1: bdt l1 var = Some lbdt and
rbdt-def-rt1: bdt r1 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 rbd
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 rbd-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 rbd-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 \ vara1-n \\
have \exists \ k < n. \ (\text{high } a1) \in \text{set } (ll \uplus k) \\
  \text{by auto} \\
then have \ higha1-in-Nodesn: \ (\text{high } a1) \in \text{Nodes } n \ ll \\
  \text{by (simp add: Nodes-def)} \\
then have \ rhigha1-in-rNodesn: \\
  \\
  \text{repc } (\text{high } a1) \in \text{repc } ' \text{Nodes } n \ ll \\
  \text{by simp} \\
from \ higha1-in-Nodesn \ normalize-prop \ obtain \ rt1h \ where \\
  rt1-dag: \ Dag (\text{repb } (\text{high } a1)) \ (\text{repb } \propto \text{low}) \ (\text{repb } \propto \text{high}) \ rt1h \\
and \\
  rt1-in-repbNort: \ \text{set-of } rt1h \subseteq \text{repb } ' \text{Nodes } n \ ll \\
  \text{apply –} \\
  \text{apply (erule-tac } x=\text{high } a1 \text{ in ballE)} \\
  \text{apply auto} \\
  \text{done} \\
from \ rt1-in-repbNort \ repbNort-repbNodes \\
have \ rt1-in-repbNort \ \text{set-of } rt1h \subseteq \text{repb } ' \text{Nodes } n \ ll \\
  \text{by blast} \\
from \ rt1-dag \ higha1-in-Nodesn \\
have \ repcrt1-dag: \\
  Dag (\text{repb } (\text{high } a1)) \ (\text{repb } \propto \text{low}) \ (\text{repb } \propto \text{high}) \ rt1h \\
  \text{apply –} \\
  \text{apply (drule Nodes-repbc-Dags-eq [rule-format])} \\
  \text{apply auto} \\
  \text{done} \\
from \ t1-Node \ t1-repc-dag \ high-a1-nNull \ children-repc-eq-a1 \\
have \ Dag (\text{repb } (\text{high } a1)) \ (\text{repb } \propto \text{low}) \ (\text{repb } \propto \text{high}) \ rt1 \\
  \text{by (auto simp add: null-comp-def)} \\
with \ repcrt1-dag \ have \ rt1h-rt1: \ rt1h = rt1 \text{ by (simp add: Dag-unique)} \\
have \ rt1-nTip: \ rt1 \neq \text{Tip} \\
proof – \\
  have \ \text{repc } (\text{high } a1) \neq \text{Null} \\
proof – \\
  note \ rhigha1-in-rNodesn \\
  also have \\
  \text{repc } ' \text{Nodes } n \ ll \subseteq \text{repc } ' \text{Nodes } (\text{Suc } n) \ ll \\
  \text{using Nodes-subset} \\
  \text{by blast} \\
  also have \ldots \subseteq \text{Nodes } (\text{Suc } n) \ ll \\
  \text{using repcNodes-in-Nodes} \\
  \text{by simp} \\
finally show \ ?thesis \\
  \text{using null-notin-Nodes-Suc-n} \\
  \text{by auto} \\
ategor {with \ repcrt1-dag \ rt1h-rt1 \ show \ ?thesis \ by \ auto} 
qed \\
with \ rt1-in-repbNort \ repcrt1-dag \ rhigha1-in-rNodesn \ a1-in-lln
tl-repc-dag  rt1h-rt1
have rt1-in-Dags-Nodesn:
  rt1 ∈ Dags (repc ' Nodes n ll) (repc ∝ low) (repc ∝ high)
  apply —
  apply (rule DagsI)
  apply auto
done

from a1-nNull a1-in-pret low-a1-nNull pret-dag
have low a1 ∈ set-of pretended
  apply —
  apply (drule subelem-set-of-low)
  apply auto
done

with wf-ll have
  low a1 ∈ set (ll ! (var (low a1))) by (simp add: wf-ll-def)
with a1-in-lln t1-repc-dag var-children-smaller-a1 vara1-n
have ∃ k < n. (low a1) ∈ set (ll ! k)
  by auto
then have lowa1-in-Nodesn: (low a1) ∈ Nodes n ll
  by (simp add: Nodes-def)
then have riowa1-in-rNodesn: repc (low a1) ∈ repc ' Nodes n ll
  by simp
from lowa1-in-Nodesn normalize-prop obtain lt1h where
lt1-dag: Dag (repb (low a1)) (repc ∝ low) (repc ∝ high) lt1h and
lt1-in-repbNort: set-of lt1h ⊆ repb 'Nodes n ll
  apply —
  apply (erule-tac x=low a1 in ballE)
  apply auto
done
from lt1-in-repbNort repbNodes-repcNodes
have lt1-in-repcNodesn: set-of lt1h ⊆ repc 'Nodes n ll
  by blast
from lt1-dag lowa1-in-Nodesn
have repclt1-dag: Dag (repc (low a1)) (repc ∝ low) (repc ∝ high)
lth
  apply —
  apply (drule Nodes-repbc-Dags-eq [rule-format])
  apply auto
done
from tl-Node tl-repc-dag low-a1-nNull children-repc-eq-a1
have Dag (repc (low a1)) (repc ∝ low) (repc ∝ high) lt1
  by (auto simp add: null-comp-def)
with repclt1-dag have lt1h-lt1: lt1h = lt1 by (simp add: Dag-unique)
have lt1-nTip: lt1 ≠ Tip
proof —
  have repc (low a1) ≠ Null

218
proof
  note rlowa1-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 repc1dag lt1h-lt1 show ?thesis by auto
qed
with lt1-in-repcNodesn repc1dag rlowa1-in-rNodesn a1-in-lln
  t1-repc-dag lt1h-lt1
have lt1-in-Dags-Nodesn:
  lt1 ∈ Dags (repc ' Nodes n ll) (repc α low) (repc α high)
apply —
apply (rule DagsI)
apply auto
done
from pret-dag ord-pret a2-in-pret low-a2-nNull high-a2-nNull
have var-children-smaller-a2:
  var (low a2) < var a2 ∧ 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 high-a2-nNull pret-dag have high a2 ∈ set-of pret
apply —
apply (drule subelem-set-of-high)
apply auto
done
with wf-ll have high a2 ∈ set (ll ! (var (high a2)))
  by (simp add: wf-ll-def)
with a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n
have ∃ k<n. (high a2) ∈ set (ll ! k)
by auto
then have \( \text{higha2-in-Nodesn} \): \((\text{high a2}) \in \text{Nodes n ll}\)
by \((\text{simp add: Nodes-def})\)
then have \( \text{rhiga2-in-rNodesn} \):
  \( \text{repc} (\text{high a2}) \in \text{repc ' Nodes n ll} \)
by simp
from \( \text{higha2-in-Nodesn} \) normalize-prop obtain \( \text{rt2h} \) where
  \( \text{rt2-dag: Dag (repb (high a2)) (repb \propto \text{low}) (repb \propto \text{high}) rt2h} \)
and
  \( \text{rt2-in-repbNort: set-of rt2h \subseteq \text{repb ' Nodes n ll}} \)
  apply –
  apply \((\text{erule-tac x=high a2 in ballE})\)
  apply auto
  done
from \( \text{rt2-in-repbNort} \) \( \text{repbNodes-repcNodes} \)
have \( \text{rt2-in-repcNosen: set-of rt2h \subseteq \text{repc ' Nodes n ll}} \)
by blast
from \( \text{rt2-dag} \) \( \text{higha2-in-Nodesn} \) have
  \( \text{repcrt2-dag: Dag (repc (high a2)) (repc \propto \text{low}) (repc \propto \text{high}) rt2h} \)
  apply –
  apply \((\text{drule Nodes-repbc-Dags-eq [rule-format]})\)
  apply auto
  done
from \( \ell2\)-Node \( \ell2\text{-repc-dag} \) \( \text{high-a2-nNull children-repc-eq-a2} \) have
  \( \text{Dag (repc (high a2)) (repc \propto \text{low}) (repc \propto \text{high}) rt2} \)
by \((\text{auto simp add: null-comp-def})\)
with \( \text{repcrt2-dag} \) have \( \text{rt2h-rt2: rt2h = rt2} \) by \((\text{simp add: Dag-unique})\)
have \( \text{rt2-nTip: rt2 \neq Tip} \)
proof –
  have \( \text{repc (high a2) \neq Null} \)
proof –
  note \( \text{rhiga2-in-rNodesn} \)
  also have
    \( \text{repc ' Nodes n ll \subseteq \text{repc ' Nodes (Suc n) ll}} \)
    using \( \text{Nodes-subset} \)
    by blast
  also have \( \ldots \subseteq \text{Nodes (Suc n) ll} \)
    using \( \text{repcNodes-in-Nodes} \)
    by simp
finally show \?thesis
  using \( \text{null-notin-Nodes-Suc-n} \)
  by auto
qed
with \( \text{repcrt2-dag} \) \( \text{rt2h-rt2} \) show \?thesis by auto
qed
with \( \text{rt2-in-repcNosen} \) \( \text{repcrt2-dag} \) \( \text{rhiga2-in-rNodesn} \) \( \text{a2-in-lln} \)
\( \ell2\text{-repc-dag} \) \( \text{rt2h-rt2} \) have \( \text{rt2-in-Dags-Nodesn} \):
\( r t 2 \in D a g s ( r e p c \ ' N o d e s n l l ) ( r e p c \propto l o w ) ( r e p c \propto h i g h ) \)
apply –
apply (rule DagsI)
apply auto
done

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

with \texttt{wf-l} have \texttt{low a2 \in set (ll ! (var (low a2)))}
by (simp add: wf-l-def)
with \texttt{a2-in-lln t2-repc-dag var-children-smaller-a2 vara2-n}
have \( \exists k < n. (\texttt{low a2}) \in \texttt{set (ll ! k)} \)
by auto
then have \texttt{lowa2-in-Nodesn: \ (low a2) \in Node}\nby (simp add: Nodes-def)
then have \texttt{rlowa2-in-rNodesn: repc (low a2) \in repc \ ' Node}\nby simp

from \texttt{lowa2-in-Nodesn normalize-prop obtain lt2h where}
\texttt{lt2-dag: Dag (repb (low a2)) (repb \propto low) (repb \propto high) lt2h and}
\texttt{lt2-in-repbNodes: set-of lt2h \subseteq repb \ ' Node}\napply –
apply (erule-tac \( \texttt{x} = \texttt{low a2} \) in \texttt{ballE})
apply auto
done
from \texttt{lt2-in-repbNodes repbNodes-repcNodes}
have \texttt{lt2-in-repcNodesn: set-of lt2h \subseteq repc \ ' Node}\nby blast
from \texttt{lt2-dag lowa2-in-Nodesn}
have \texttt{repclt2-dag: Dag (repc (low a2)) (repc \propto low) (repc \propto high) lt2h}
apply –
apply (drule Nodes-repbc-Dags-eq [rule-format])
apply auto
done
from \texttt{t2-Node t2-repc-dag low-a2-nNull children-repc-eq-a2}
have \texttt{Dag (repc (low a2)) (repc \propto low) (repc \propto high) lt2}
by (auto simp add: null-comp-def)
with \texttt{repclt2-dag have lt2h-lt2: lt2h = lt2 by (simp add: Dag-unique)}
have \texttt{lt2-nTip: lt2 \neq Tip}
proof –
have \texttt{repc (low a2) \neq Null}
proof –
note \texttt{rlowa2-in-rNodesn}
also have
\[ \text{repc } \text{Nodes } n \; \text{ll } \subseteq \text{repc } \text{Nodes } (\text{Suc } n) \; \text{ll} \]
using 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 repclt2-dag lt2h-lt2 show \(?\)thesis by auto
qed

with lt2-in-repcNodesn repcLt2-dag rlRowa2-in-rNodesn a2-in-lln
t2-repc-dag lt2h-lt2
have lt2-in-Dags-Nodesn:
\[ \text{lt2 } \in \text{Dags } (\text{repc } \text{Nodes } n \; \text{ll}) (\text{repc } \llow) (\text{repc } \lhight) \]
apply –
apply (rule DagsI)
apply auto
done

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

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

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

by auto
with lt2-nTip t2-repc-dag t2-Node obtain llt2 lt2p rlt2 where
lt2-Node: \( \text{l}t2 = \text{Node} \ llt2 \ llt2p \ rlt2 \)
by auto
with t2-repc-dag t2-Node children-repc-eq-a2 lt2-nTip
have root-lt2: root \( \text{l}t2 = (\text{repc} \propto \text{low}) \ a2 \)
by auto
from root-lt1-lt2 root-lt2 root-lt1
have low-a1-a2: (\text{repc} \propto \text{low}) \ a1 = (\text{repc} \propto \text{low}) \ a2
by auto
from shared-rt1-rt2 rbdt-def-rt1 rbdt-def-rt2 have rt1-rt2: \( \text{rt1} = \text{rt2} \)
by (auto simp add: isomorphic-dags-eq-def)
then have root-rt1-rt2: \( \text{root \ rt1} = \text{root \ rt2} \)
by auto
from rt1-nTip t1-repc-dag t1-Node have (\text{repc} \propto \text{high}) \ (\text{repc} \ a1) \neq \text{Null}
by auto
with rt1-nTip t1-repc-dag t1-Node obtain \( \text{lrt1} \ \text{lrt1p} \ \text{rrt1} \) where
rt1-Node: \( \text{rt1} = \text{Node} \ \text{lrt1} \ \text{lrt1p} \ \text{rrt1} \)
by auto
with t1-repc-dag t1-Node children-repc-eq-a1 rt1-nTip
have root-rt1: \( \text{root \ rt1} = (\text{repc} \propto \text{high}) \ a1 \)
by auto
from rt2-nTip t2-repc-dag t2-Node
have (\text{repc} \propto \text{high}) \ (\text{repc} \ a2) \neq \text{Null}
by auto
with rt2-nTip t2-repc-dag t2-Node obtain \( \text{lrt2} \ \text{lrt2p} \ \text{rrt2} \) where
rt2-Node: \( \text{rt2} = \text{Node} \ \text{lrt2} \ \text{lrt2p} \ \text{rrt2} \)
by auto
with t2-repc-dag t2-Node children-repc-eq-a2 rt2-nTip
have root-rt2: \( \text{root \ rt2} = (\text{repc} \propto \text{high}) \ a2 \)
by auto
from root-rt1-rt2 root-rt2 root-rt1
have high-a1-a2: (\text{repc} \propto \text{high}) \ a1 = (\text{repc} \propto \text{high}) \ a2
by auto
from low-a1-a2 high-a1-a2 share-a1-a2
have \( \text{repc} \ a1 = \text{repc} \ a2 \)
by auto
with \( \text{lt1-lt2} \ \text{rt1-rt2} \) show \( \text{?thesis} \)
by auto
qed
deq
deq
deq
deq
from termi dags-shared while-while-prop repcNodes-in-Nodes repc-nc n-var-prop

wf-marking-m-ma
show \( \text{?thesis} \)
by auto
qed
```text
qed
with srrl-precond all-nodes-same-var
show thesis
  apply
  apply (intro conjI)
  apply assumption+
  done
qed
qed
end

References

```