

Abstract

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

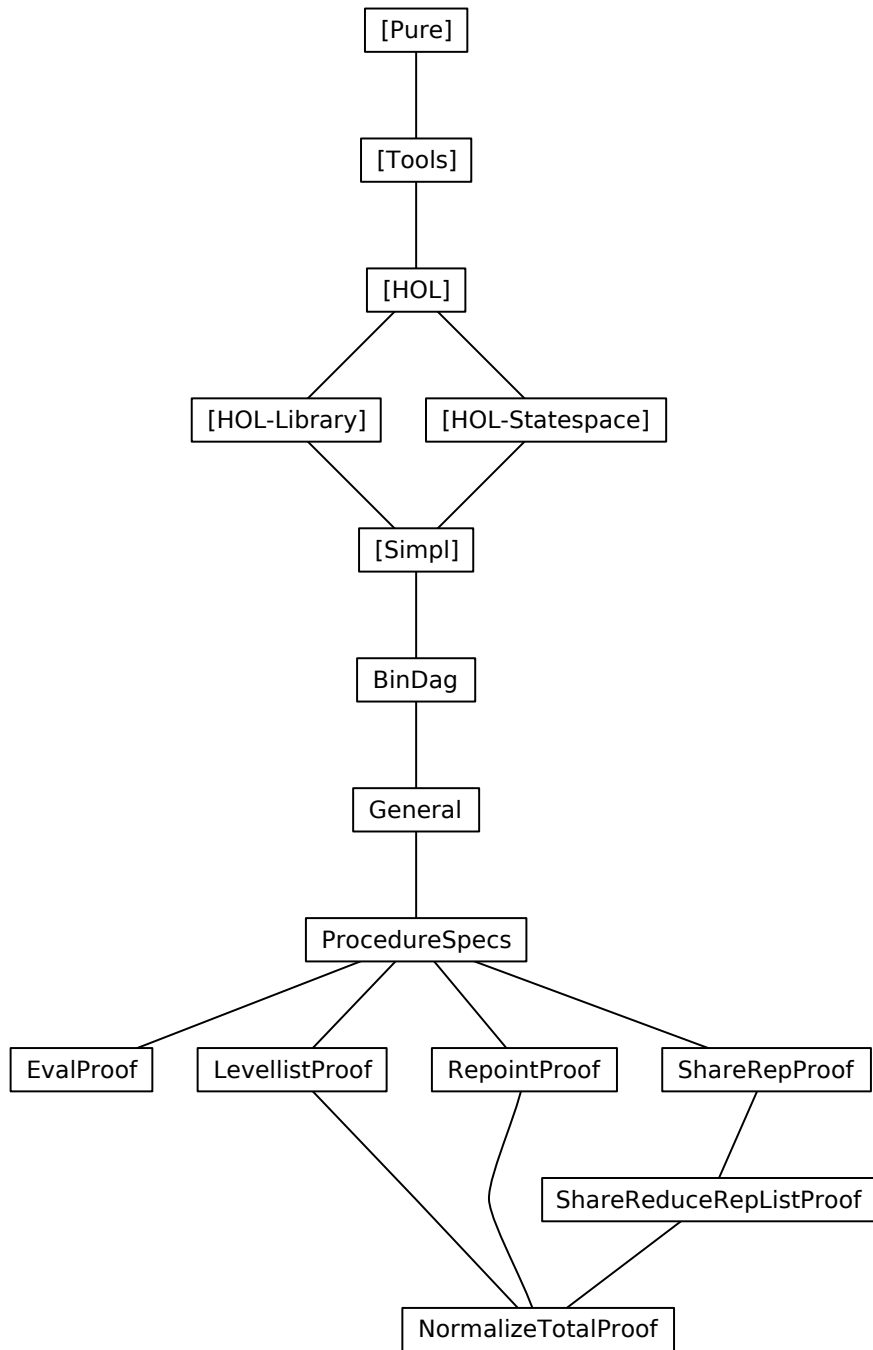
BDD-Normalisation

Veronika Ortner and Norbert Schirmer

May 26, 2024

Contents

1	Introduction	4
2	BDD Abstractions	4
3	General Lemmas on BDD Abstractions	9
4	Definitions of Procedures	19
5	Proof of Procedure Eval	22
6	Proof of Procedure Levellist	22
7	Proof of Procedure ShareRep	25
8	Proof of Procedure ShareReduceRepList	25
9	Proof of Procedure Repoint	26
10	Proof of Procedure Normalize	27



1 Introduction

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

2 BDD Abstractions

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

datatype dag = Tip | Node dag ref dag

lemma [simp]: Node lt a rt  $\neq$  lt
  <proof>

lemma [simp]: lt  $\neq$  Node lt a rt
  <proof>

lemma [simp]: Node lt a rt  $\neq$  rt
  <proof>

lemma [simp]: rt  $\neq$  Node lt a rt
  <proof>

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

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

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

lemma subdag-size: subdag t s  $\Longrightarrow$  size s < size t
  <proof>

lemma subdag-neq: subdag t s  $\Longrightarrow$  t $\neq$ s
  <proof>

lemma subdag-trans [trans]: subdag t s  $\Longrightarrow$  subdag s r  $\Longrightarrow$  subdag t r
  <proof>

lemma subdag-NodeD:
```

$subdag\ t\ (Node\ lt\ a\ rt) \implies subdag\ t\ lt \wedge subdag\ t\ rt$
 $\langle proof \rangle$

lemma *subdag-not-sym*: $\bigwedge t. \llbracket subdag\ s\ t; subdag\ t\ s \rrbracket \implies P$
 $\langle proof \rangle$

instantiation *dag* :: *order*
begin

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

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

lemma *le-dag-refl*: $(x::dag) \leq x$
 $\langle proof \rangle$

lemma *le-dag-trans*:
fixes $x::dag$ **and** y **and** z
assumes $x-y: x \leq y$ **and** $y-z: y \leq z$
shows $x \leq z$
 $\langle proof \rangle$

lemma *le-dag-antisym*:
fixes $x::dag$ **and** y
assumes $x-y: x \leq y$ **and** $y-x: y \leq x$
shows $x = y$
 $\langle proof \rangle$

lemma *dag-less-le*:
fixes $x::dag$ **and** y
shows $(x < y) = (x \leq y \wedge x \neq y)$
 $\langle proof \rangle$

instance
 $\langle proof \rangle$

end

lemma *less-dag-Tip* [*simp*]: $\neg (x < Tip)$
 $\langle proof \rangle$

lemma *less-dag-Node*: $x < (Node\ l\ a\ r) =$
 $(x \leq l \vee x \leq r)$
 $\langle proof \rangle$

lemma *less-dag-Node'*: $x < (Node\ l\ a\ r) =$
 $(x=l \vee x=r \vee x < l \vee x < r)$

<proof>

lemma *less-Node-dag*: $(Node\ l\ a\ r) < x \implies l < x \wedge r < x$
<proof>

lemma *less-dag-set-of*: $x < y \implies set-of\ x \subseteq set-of\ y$
<proof>

lemma *le-dag-set-of*: $x \leq y \implies set-of\ x \subseteq set-of\ y$
<proof>

lemma *DAG-less*: $DAG\ y \implies x < y \implies DAG\ x$
<proof>

lemma *less-DAG-set-of*:
 assumes *x-less-y*: $x < y$
 assumes *DAG-y*: $DAG\ y$
 shows $set-of\ x \subset set-of\ y$
<proof>

lemma *in-set-of-decomp*:
 $p \in set-of\ t = (\exists\ l\ r. t = (Node\ l\ p\ r) \vee subdag\ t\ (Node\ l\ p\ r))$
 (**is** $?A = ?B$)
<proof>

primrec *Dag*:: $ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow dag \Rightarrow bool$
where
 $Dag\ p\ l\ r\ Tip = (p = Null) \mid$
 $Dag\ p\ l\ r\ (Node\ lt\ a\ rt) = (p = a \wedge p \neq Null \wedge$
 $Dag\ (l\ p)\ l\ r\ lt \wedge Dag\ (r\ p)\ l\ r\ rt)$

lemma *Dag-Null [simp]*: $Dag\ Null\ l\ r\ t = (t = Tip)$
<proof>

lemma *Dag-Ref [simp]*:
 $p \neq Null \implies Dag\ p\ l\ r\ t = (\exists\ lt\ rt. t = Node\ lt\ p\ rt \wedge$
 $Dag\ (l\ p)\ l\ r\ lt \wedge Dag\ (r\ p)\ l\ r\ rt)$
<proof>

lemma *Null-notin-Dag [simp, intro]*: $\bigwedge p\ l\ r. Dag\ p\ l\ r\ t \implies Null \notin set-of\ t$
<proof>

theorem *notin-Dag-update-l [simp]*:
 $\bigwedge p. q \notin set-of\ t \implies Dag\ p\ (l(q := y))\ r\ t = Dag\ p\ l\ r\ t$
<proof>

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

<proof>

lemma *Dag-upd-same-l-lemma*: $\bigwedge p. p \neq \text{Null} \implies \neg \text{Dag } p \ (l(p:=p)) \ r \ t$
<proof>

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

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

lemma *Dag-upd-same-r-lemma*: $\bigwedge p. p \neq \text{Null} \implies \neg \text{Dag } p \ l \ (r(p:=p)) \ t$
<proof>

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

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

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

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

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

lemma *Dag-unique*: $\bigwedge p \ t2. \text{Dag } p \ l \ r \ t1 \implies \text{Dag } p \ l \ r \ t2 \implies t1 = t2$
<proof>

lemma *Dag-unique1*: $\text{Dag } p \ l \ r \ t \implies \exists ! t. \text{Dag } p \ l \ r \ t$
<proof>

lemma *Dag-subdag*: $\bigwedge p. \text{Dag } p \ l \ r \ t \implies \text{subdag } t \ s \implies \exists q. \text{Dag } q \ l \ r \ s$
<proof>

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>

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>

lemma *Dag-is-DAG*: $\bigwedge p \ l \ r. \text{Dag } p \ l \ r \ t \implies \text{DAG } t$
<proof>

lemma *heaps-eq-Dag-eq*:
 $\bigwedge p. \forall x \in \text{set-of } t. l \ x = l' \ x \wedge r \ x = r' \ x$
 $\implies \text{Dag } p \ l \ r \ t = \text{Dag } p \ l' \ r' \ t$
<proof>

lemma *heaps-eq-DagI1*:
 $\llbracket \text{Dag } p \ l \ r \ t; \forall x \in \text{set-of } t. l \ x = l' \ x \wedge r \ x = r' \ x \rrbracket$
 $\implies \text{Dag } p \ l' \ r' \ t$
<proof>

lemma *heaps-eq-DagI2*:
 $\llbracket \text{Dag } p \ l' \ r' \ t; \forall x \in \text{set-of } t. l \ x = l' \ x \wedge r \ x = r' \ x \rrbracket$
 $\implies \text{Dag } p \ l \ r \ t$
<proof>

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

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

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

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

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

lemma *Dag-conv-isDag-dag*: $\text{Dag } p \ l \ r \ t = (\text{isDag } p \ l \ r \wedge t = \text{dag } p \ l \ r)$
<proof>

lemma *Dag-dag*: $\text{Dag } p \ l \ r \ t \implies \text{dag } p \ l \ r = t$
<proof>

end

3 General Lemmas on BDD Abstractions

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

definition *subdag-eq*:: *dag* \Rightarrow *dag* \Rightarrow *bool* **where**
subdag-eq *t*₁ *t*₂ = (*t*₁ = *t*₂ \vee *subdag* *t*₁ *t*₂)

primrec *root* :: *dag* \Rightarrow *ref*

where

root *Tip* = *Null* |

root (*Node* *l* *a* *r*) = *a*

fun *isLeaf* :: *dag* \Rightarrow *bool* **where**

isLeaf *Tip* = *False*

| *isLeaf* (*Node* *Tip* *v* *Tip*) = *True*

| *isLeaf* (*Node* (*Node* *l* *v*₁ *r*) *v*₂ *Tip*) = *False*

| *isLeaf* (*Node* *Tip* *v*₁ (*Node* *l* *v*₂ *r*)) = *False*

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

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

bdt-fn *Tip* = (λ *bdtvar* . *None*)

| *bdt-fn* (*Node* *Tip* *vref* *Tip*) =

(λ *bdtvar* .

(*if* (*bdtvar* *vref* = 0)

then *Some* *Zero*

else (*if* (*bdtvar* *vref* = 1)

then *Some* *One*

else *None*)))

| *bdt-fn* (*Node* *Tip* *vref* (*Node* *l* *vref1* *r*)) = (λ *bdtvar* . *None*)

| *bdt-fn* (*Node* (*Node* *l* *vref1* *r*) *vref* *Tip*) = (λ *bdtvar* . *None*)

| *bdt-fn* (*Node* (*Node* *l*₁ *vref1* *r*₁) *vref* (*Node* *l*₂ *vref2* *r*₂)) =

(λ *bdtvar* .

(*if* (*bdtvar* *vref* = 0 \vee *bdtvar* *vref* = 1)

then *None*

else

(*case* (*bdt-fn* (*Node* *l*₁ *vref1* *r*₁) *bdtvar*) *of*

None \Rightarrow *None*

| (*Some* *b1*) \Rightarrow

(*case* (*bdt-fn* (*Node* *l*₂ *vref2* *r*₂) *bdtvar*) *of*

None \Rightarrow *None*

| (*Some* *b2*) \Rightarrow *Some* (*Bdt-Node* *b1* (*bdtvar* *vref*) *b2*))))))

abbreviation *bdt* == *bdt-fn*

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

where

```

eval Zero env = False |
eval One env = True |
eval (Bdt-Node l v r) env = (if (env ! v) then eval r env else eval l env)

```

```

fun ordered-bdt:: bdt  $\Rightarrow$  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)  $\wedge$  (v2 < v)  $\wedge$ 
   (ordered-bdt (Bdt-Node l1 v1 r1))  $\wedge$  (ordered-bdt (Bdt-Node l2 v2 r2)))
| ordered-bdt (Bdt-Node (Bdt-Node l1 v1 r1) v r) =
  ((v1 < v)  $\wedge$  (ordered-bdt (Bdt-Node l1 v1 r1)))
| ordered-bdt (Bdt-Node l v (Bdt-Node l2 v2 r2)) =
  ((v2 < v)  $\wedge$  (ordered-bdt (Bdt-Node l2 v2 r2)))
| ordered-bdt (Bdt-Node l v r) = True

```

```

fun ordered:: dag  $\Rightarrow$  (ref $\Rightarrow$ nat)  $\Rightarrow$  bool where
ordered Tip = ( $\lambda$  var. True)
| ordered (Node (Node l1 v1 r1) v (Node l2 v2 r2)) =
  ( $\lambda$  var. (var v1 < var v  $\wedge$  var v2 < var v)  $\wedge$ 
   (ordered (Node l1 v1 r1) var)  $\wedge$  (ordered (Node l2 v2 r2) var))
| ordered (Node Tip v Tip) = ( $\lambda$  var. (True))
| ordered (Node Tip v r) =
  ( $\lambda$  var. (var (root r) < var v)  $\wedge$  (ordered r var))
| ordered (Node l v Tip) =
  ( $\lambda$  var. (var (root l) < var v)  $\wedge$  (ordered l var))

```

```

primrec max-var :: bdt  $\Rightarrow$  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: \llbracket bdt (Node l v r) var = Some x;
var (root (Node l v r)) = (0::nat) $\rrbracket \Longrightarrow x = \text{Zero}$
<proof>

lemma bdt-Some-One-iff [simp]:
(bdt t var = Some One) = (\exists p. t = Node Tip p Tip \wedge var p = 1)
<proof>

lemma bdt-Some-Zero-iff [simp]:
(bdt t var = Some Zero) = (\exists p. t = Node Tip p Tip \wedge var p = 0)
<proof>

lemma *bdt-Some-Node-iff* [simp]:

$$(bdt\ t\ var = Some\ (Bdt-Node\ bdt1\ v\ bdt2)) = \\ (\exists\ p\ l\ r.\ t = Node\ l\ p\ r \wedge bdt\ l\ var = Some\ bdt1 \wedge bdt\ r\ var = Some\ bdt2 \wedge \\ 1 < v \wedge var\ p = v)$$

$\langle proof \rangle$

lemma *balanced-bdt*:

$$\bigwedge p\ bdt1.\ \llbracket Dag\ p\ low\ high\ t;\ bdt\ t\ var = Some\ bdt1;\ no \in set-of\ t \rrbracket \\ \implies (low\ no = Null) = (high\ no = Null)$$

$\langle proof \rangle$

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

where

$$dag-map\ f\ Tip = Tip\ |$$

$$dag-map\ f\ (Node\ l\ a\ r) = (Node\ (dag-map\ f\ l)\ (f\ a)\ (dag-map\ f\ r))$$

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

where

$$wf-marking\ t\ mark-old\ mark-new\ marked =$$

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

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

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

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

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

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

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

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

$\langle proof \rangle$

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

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

$\langle proof \rangle$

lemma *ordered-set-of*:

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

$\langle proof \rangle$

lemma *dag-setofD*: $\bigwedge p\ low\ high\ n.\ \llbracket Dag\ p\ low\ high\ t;\ n \in set-of\ t \rrbracket \implies$

$$(\exists\ nt.\ Dag\ n\ low\ high\ nt) \wedge (\forall\ nt.\ Dag\ n\ low\ high\ nt \longrightarrow set-of\ nt \subseteq set-of\ t)$$

$\langle proof \rangle$

lemma *dag-setof-exD*: $\llbracket Dag\ p\ low\ high\ t;\ n \in set-of\ t \rrbracket \implies \exists\ nt.\ Dag\ n\ low\ high\ nt$

$\langle proof \rangle$

lemma *dag-setof-subsetD*: $\llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t; \text{Dag } n \text{ low high } nt \rrbracket \implies$
 $\text{set-of } nt \subseteq \text{set-of } t$
 $\langle \text{proof} \rangle$

lemma *subdag-parentdag-low*: $\text{not } \leq lt \implies \text{not } \leq (\text{Node } lt \ p \ rt)$ **for** *not*
 $\langle \text{proof} \rangle$

lemma *subdag-parentdag-high*: $\text{not } \leq rt \implies \text{not } \leq (\text{Node } lt \ p \ rt)$ **for** *not*
 $\langle \text{proof} \rangle$

lemma *set-of-subdag*: $\bigwedge p \ \text{not } no.$
 $\llbracket \text{Dag } p \text{ low high } t; \text{Dag } no \text{ low high } not; no \in \text{set-of } t \rrbracket \implies \text{not } \leq t$
 $\langle \text{proof} \rangle$

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

lemma *ordered-subdag*: $\llbracket \text{ordered } t \ \text{var}; \text{not } \leq t \rrbracket \implies \text{ordered } not \ \text{var}$ **for** *not*
 $\langle \text{proof} \rangle$

lemma *subdag-ordered*:
 $\bigwedge \ \text{not } no \ p. \llbracket \text{ordered } t \ \text{var}; \text{Dag } p \text{ low high } t; \text{Dag } no \text{ low high } not;$
 $\text{no} \in \text{set-of } t \rrbracket \implies \text{ordered } not \ \text{var}$
 $\langle \text{proof} \rangle$

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

definition *wf-ll* :: $\text{dag} \Rightarrow \text{ref list list} \Rightarrow (\text{ref} \Rightarrow \text{nat}) \Rightarrow \text{bool}$

where

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

definition *cong-eval* :: $\text{bdt} \Rightarrow \text{bdt} \Rightarrow \text{bool}$ (**infix** ~ 60)

where $\text{cong-eval } bdt_1 \ bdt_2 = (\text{eval } bdt_1 = \text{eval } bdt_2)$

lemma *cong-eval-sym*: $l \sim r = r \sim l$
 $\langle \text{proof} \rangle$

lemma *cong-eval-trans*: $\llbracket l \sim r; r \sim t \rrbracket \implies l \sim t$
 $\langle \text{proof} \rangle$

lemma *cong-eval-child-high*: $l \sim r \implies r \sim (\text{Bdt-Node } l \ v \ r)$
 ⟨proof⟩

lemma *cong-eval-child-low*: $l \sim r \implies l \sim (\text{Bdt-Node } l \ v \ r)$
 ⟨proof⟩

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

lemma *null-comp-not-Null* [*simp*]: $h \ q \neq \text{Null} \implies (g \times h) \ q = g \ (h \ q)$
 ⟨proof⟩

lemma *id-trans*: $(a \times \text{id}) \ (b \ (c \ p)) = (a \times b) \ (c \ p)$
 ⟨proof⟩

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

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

lemma *empty-Dags* [*simp*]: $\text{Dags } \{\} \ \text{low high} = \{\}$
 ⟨proof⟩

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

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

where
repNodes-eq $p \ q \ \text{low high rep} ==$
 $(\text{rep} \times \text{high}) \ p = (\text{rep} \times \text{high}) \ q \wedge (\text{rep} \times \text{low}) \ p = (\text{rep} \times \text{low}) \ q$

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

where
isomorphic-dags-eq $st_1 \ st_2 \ \text{var} =$
 $(\forall \text{bdt}_1 \ \text{bdt}_2. (\text{bdt } st_1 \ \text{var} = \text{Some } \text{bdt}_1 \wedge \text{bdt } st_2 \ \text{var} = \text{Some } \text{bdt}_2 \wedge (\text{bdt}_1 = \text{bdt}_2))$
 $\longrightarrow st_1 = st_2)$

lemma *isomorphic-dags-eq-sym*: $\text{isomorphic-dags-eq } st_1 \ st_2 \ \text{var} = \text{isomorphic-dags-eq } st_2 \ st_1 \ \text{var}$

<proof>

definition *shared* :: *dag* \Rightarrow (*ref* \Rightarrow *nat*) \Rightarrow *bool*
 where *shared* *t* *var* = (\forall *st*₁ *st*₂. (*st*₁ \leq *t* \wedge *st*₂ \leq *t*) \longrightarrow *isomorphic-dags-eq*
*st*₁ *st*₂ *var*)

fun *reduced* :: *dag* \Rightarrow *bool* **where**
 reduced *Tip* = *True*
 | *reduced* (*Node* *Tip* *v* *Tip*) = *True*
 | *reduced* (*Node* *l* *v* *r*) = (*l* \neq *r* \wedge *reduced* *l* \wedge *reduced* *r*)

primrec *reduced-bdt* :: *bdt* \Rightarrow *bool*
where
 reduced-bdt *Zero* = *True*
 | *reduced-bdt* *One* = *True*
 | *reduced-bdt* (*Bdt-Node* *l* *bdt* *v* *r* *bdt*) =
 (*if* *l* *bdt* = *r* *bdt* *then* *False*
 else (*reduced-bdt* *l* \wedge *reduced-bdt* *r* *bdt*))

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

lemma *no-in-one-ll*:
 [[*wf-ll* *pret* *levellista* *var*; *i* < *length* *levellista*; *j* < *length* *levellista*;
 no \in *set* (*levellista* ! *i*); *i* \neq *j*]]
 \implies *no* \notin *set* (*levellista* ! *j*)
<proof>

lemma *nodes-in-wf-ll*:
 [[*wf-ll* *pret* *levellista* *var*; *i* < *length* *levellista*; *no* \in *set* (*levellista* ! *i*)]]
 \implies *var* *no* = *i* \wedge *no* \in *set-of* *pret*
<proof>

lemma *subelem-set-of-low*:
 \bigwedge *p*. [[*x* \in *set-of* *t*; *x* \neq *Null*; *low* *x* \neq *Null*; *Dag* *p* *low* *high* *t*]]
 \implies (*low* *x*) \in *set-of* *t*
<proof>

lemma *subelem-set-of-high*:
 \bigwedge *p*. [[*x* \in *set-of* *t*; *x* \neq *Null*; *high* *x* \neq *Null*; *Dag* *p* *low* *high* *t*]]
 \implies (*high* *x*) \in *set-of* *t*
<proof>

lemma *set-split*: $\{k. k < (\text{Suc } n)\} = \{k. k < n\} \cup \{n\}$
 <proof>

lemma *Nodes-levellist-subset-t*:
 $\llbracket \text{wf-ll } t \text{ levellist } \text{var}; i \leq \text{length levellist} \rrbracket \implies \text{Nodes } i \text{ levellist} \subseteq \text{set-of } t$
 <proof>

lemma *bdt-child*:
 $\llbracket \text{bdt } (\text{Node } (\text{Node } \text{llt } l \text{ rlt}) p \text{ (Node } \text{lrt } r \text{ rrt)}) \text{ var} = \text{Some } \text{bdt1} \rrbracket$
 $\implies \exists \text{lbdt } \text{rbdt}. \text{ bdt } (\text{Node } \text{llt } l \text{ rlt}) \text{ var} = \text{Some } \text{lbdt} \wedge$
 $\text{ bdt } (\text{Node } \text{lrt } r \text{ rrt}) \text{ var} = \text{Some } \text{rbdt}$
 <proof>

lemma *subbdt-ex-dag-def*:
 $\bigwedge \text{bdt1 } p. \llbracket \text{Dag } p \text{ low } \text{high } t; \text{ bdt } t \text{ var} = \text{Some } \text{bdt1}; \text{ Dag } \text{no } \text{low } \text{high } \text{not};$
 $\text{ no } \in \text{set-of } t \rrbracket \implies \exists \text{bdt2}. \text{ bdt } \text{not } \text{var} = \text{Some } \text{bdt2} \text{ for } \text{not}$
 <proof>

lemma *subbdt-ex*:
 $\bigwedge \text{bdt1}. \llbracket (\text{Node } \text{lst } \text{stp } \text{rst}) \leq t; \text{ bdt } t \text{ var} = \text{Some } \text{bdt1} \rrbracket$
 $\implies \exists \text{bdt2}. \text{ bdt } (\text{Node } \text{lst } \text{stp } \text{rst}) \text{ var} = \text{Some } \text{bdt2}$
 <proof>

lemma *var-ordered-children*:
 $\bigwedge p. \llbracket \text{Dag } p \text{ low } \text{high } t; \text{ ordered } t \text{ var}; \text{ no } \in \text{set-of } t;$
 $\text{ low } \text{no} \neq \text{Null}; \text{ high } \text{no} \neq \text{Null} \rrbracket$
 $\implies \text{var } (\text{low } \text{no}) < \text{var } \text{no} \wedge \text{var } (\text{high } \text{no}) < \text{var } \text{no}$
 <proof>

lemma *nort-null-comp*:
assumes *pret-dag*: $\text{Dag } p \text{ low } \text{high } \text{pret}$ **and**
prebdt-pret: $\text{bdt } \text{pret } \text{var} = \text{Some } \text{prebdt}$ **and**
nort-dag: $\text{Dag } (\text{repc } \text{no}) (\text{repb } \times \text{low}) (\text{repb } \times \text{high}) \text{ nort}$ **and**
ord-pret: $\text{ordered } \text{pret } \text{var}$ **and**
wf-llb: $\text{wf-ll } \text{pret } \text{levellistb } \text{var}$ **and**
nbsll: $\text{nb} < \text{length } \text{levellistb}$ **and**
repcb-nc: $\forall \text{nt}. \text{nt} \notin \text{set } (\text{levellistb } ! \text{nb}) \longrightarrow \text{repb } \text{nt} = \text{repc } \text{nt}$ **and**
xsnb-in-pret: $\forall x \in \text{set-of } \text{nort}. \text{var } x < \text{nb} \wedge x \in \text{set-of } \text{pret}$
shows $\forall x \in \text{set-of } \text{nort}. ((\text{repc } \times \text{low}) x = (\text{repb } \times \text{low}) x \wedge$
 $(\text{repc } \times \text{high}) x = (\text{repb } \times \text{high}) x)$
 <proof>

lemma *wf-ll-Nodes-pret*:
 $\llbracket \text{wf-ll } \text{pret } \text{levellista } \text{var}; \text{nb} < \text{length } \text{levellista}; x \in \text{Nodes } \text{nb } \text{levellista} \rrbracket$
 $\implies x \in \text{set-of } \text{pret} \wedge \text{var } x < \text{nb}$

$\langle \text{proof} \rangle$

lemma *bdt-Some-var1-One*:

$\bigwedge x. \llbracket \text{bdt } t \text{ var} = \text{Some } x; \text{var } (\text{root } t) = 1 \rrbracket$
 $\implies x = \text{One} \wedge t = (\text{Node } \text{Tip } (\text{root } t) \text{Tip})$
 $\langle \text{proof} \rangle$

lemma *bdt-Some-var0-Zero*:

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

lemma *reduced-children-parent*:

$\llbracket \text{reduced } l; l = (\text{Node } \text{llt } lp \text{ rlt}); \text{reduced } r; r = (\text{Node } \text{lrt } rp \text{ rrt});$
 $lp \neq rp \rrbracket$
 $\implies \text{reduced } (\text{Node } l \text{ } p \text{ } r)$
 $\langle \text{proof} \rangle$

lemma *Nodes-subset*: $\text{Nodes } i \text{ levellista} \subseteq \text{Nodes } (\text{Suc } i) \text{ levellista}$

$\langle \text{proof} \rangle$

lemma *Nodes-levellist*:

$\llbracket \text{wf-ll pret levellista var}; nb < \text{length levellista}; p \in \text{Nodes } nb \text{ levellista} \rrbracket$
 $\implies p \notin \text{set } (\text{levellista } ! nb)$
 $\langle \text{proof} \rangle$

lemma *Nodes-var-pret*:

$\llbracket \text{wf-ll pret levellista var}; nb < \text{length levellista}; p \in \text{Nodes } nb \text{ levellista} \rrbracket$
 $\implies \text{var } p < nb \wedge p \in \text{set-of pret}$
 $\langle \text{proof} \rangle$

lemma *Dags-root-in-Nodes*:

assumes *t-in-DagsSucnb*: $t \in \text{Dags } (\text{Nodes } (\text{Suc } nb) \text{ levellista}) \text{ low high}$
shows $\exists p. \text{Dag } p \text{ low high } t \wedge p \in \text{Nodes } (\text{Suc } nb) \text{ levellista}$
 $\langle \text{proof} \rangle$

lemma *subdag-dag*:

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

lemma *Dags-subdags*:

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

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

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

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

$\langle \text{proof} \rangle$

lemma *Dags-Nodes-cases:*

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

dags-in-lower-levels:

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

dags-in-mixed-levels:

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

dags-in-high-level:

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

shows $\forall t1\ t2. t1 \in \text{Dags} (\text{fnct } \langle \text{Nodes } (\text{Suc } n)\ \text{levellista} \rangle)\ \text{low}\ \text{high} \wedge$
 $t2 \in \text{Dags} (\text{fnct } \langle \text{Nodes } (\text{Suc } n)\ \text{levellista} \rangle)\ \text{low}\ \text{high}$
 $\longrightarrow P\ t1\ t2\ \text{var}$

$\langle \text{proof} \rangle$

lemma *Null-notin-Nodes:* $\llbracket \text{Dag } p\ \text{low}\ \text{high}\ t; nb \leq \text{length } \text{levellista}; \text{wf-ll } t\ \text{levellista}\ \text{var} \rrbracket \implies \text{Null} \notin \text{Nodes } nb\ \text{levellista}$

$\langle \text{proof} \rangle$

lemma *Nodes-in-pret:* $\llbracket \text{wf-ll } t\ \text{levellista}\ \text{var}; nb \leq \text{length } \text{levellista} \rrbracket \implies \text{Nodes } nb\ \text{levellista} \subseteq \text{set-of } t$

$\langle \text{proof} \rangle$

lemma *restrict-root-Node:*

$\llbracket t \in \text{Dags} (\text{repc } \langle \text{Nodes } (\text{Suc } nb)\ \text{levellista} \rangle)\ (\text{repc } \alpha\ \text{low})\ (\text{repc } \alpha\ \text{high}); t \notin \text{Dags}$
 $(\text{repc } \langle \text{Nodes } nb\ \text{levellista} \rangle)\ (\text{repc } \alpha\ \text{low})\ (\text{repc } \alpha\ \text{high});$
 $\text{ordered } t\ \text{var}; \forall no \in \text{Nodes } (\text{Suc } nb)\ \text{levellista}. \text{var } (\text{repc } no) \leq \text{var } no \wedge \text{repc}$
 $(\text{repc } no) = \text{repc } no; \text{wf-ll } \text{pret } \text{levellista}\ \text{var}; nb < \text{length } \text{levellista}; \text{repc } \langle \text{Nodes } (\text{Suc } nb)$
 $\text{levellista} \subseteq \text{Nodes } (\text{Suc } nb)\ \text{levellista} \rrbracket$

$\implies \exists q. \text{Dag } (\text{repc } q)\ (\text{repc } \alpha\ \text{low})\ (\text{repc } \alpha\ \text{high})\ t \wedge q \in \text{set } (\text{levellista } !\ nb)$

$\langle \text{proof} \rangle$

lemma *same-bdt-var*: $\llbracket \text{bdt} (\text{Node } lt1 \ p1 \ rt1) \ \text{var} = \text{Some } bdt1; \text{bdt} (\text{Node } lt2 \ p2 \ rt2) \ \text{var} = \text{Some } bdt1 \rrbracket$
 $\implies \text{var } p1 = \text{var } p2$
 $\langle \text{proof} \rangle$

lemma *bdt-Some-Leaf-var-le-1*:
 $\llbracket \text{Dag } p \ \text{low } \text{high } t; \text{bdt } t \ \text{var} = \text{Some } x; \text{isLeaf-pt } p \ \text{low } \text{high} \rrbracket$
 $\implies \text{var } p \leq 1$
 $\langle \text{proof} \rangle$

lemma *subnode-dag-cons*:
 $\bigwedge p. \llbracket \text{Dag } p \ \text{low } \text{high } t; no \in \text{set-of } t \rrbracket \implies \exists \text{ not. Dag } no \ \text{low } \text{high } \text{not}$
 $\langle \text{proof} \rangle$

lemma *nodes-in-taken-in-takeSucn*: $no \in \text{set} (\text{take } n \ \text{nodeslist}) \implies no \in \text{set} (\text{take} (\text{Suc } n) \ \text{nodeslist})$
 $\langle \text{proof} \rangle$

lemma *ind-in-higher-take*: $\bigwedge n \ k. \llbracket n < k; n < \text{length } xs \rrbracket$
 $\implies xs ! n \in \text{set} (\text{take } k \ xs)$
 $\langle \text{proof} \rangle$

lemma *take-length-set*: $\bigwedge n. n = \text{length } xs \implies \text{set} (\text{take } n \ xs) = \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *repNodes-eq-ext-rep*: $\llbracket \text{low } no \neq \text{nodeslist} ! n; \text{high } no \neq \text{nodeslist} ! n; \text{low } sn \neq \text{nodeslist} ! n; \text{high } sn \neq \text{nodeslist} ! n \rrbracket$
 $\implies \text{repNodes-eq } sn \ no \ \text{low } \text{high } \text{repa} = \text{repNodes-eq } sn \ no \ \text{low } \text{high} (\text{repa}(\text{nodeslist} ! n := \text{repa} (\text{low} (\text{nodeslist} ! n))))$
 $\langle \text{proof} \rangle$

lemma *filter-not-empty*: $\llbracket x \in \text{set } xs; P \ x \rrbracket \implies \text{filter } P \ xs \neq []$
 $\langle \text{proof} \rangle$

lemma $x \in \text{set} (\text{filter } P \ xs) \implies P \ x$
 $\langle \text{proof} \rangle$

lemma *hd-filter-in-list*: $\text{filter } P \text{ } xs \neq [] \implies \text{hd } (\text{filter } P \text{ } xs) \in \text{set } xs$
 <proof>

lemma *hd-filter-in-filter*: $\text{filter } P \text{ } xs \neq [] \implies \text{hd } (\text{filter } P \text{ } xs) \in \text{set } (\text{filter } P \text{ } xs)$
 <proof>

lemma *hd-filter-prop*:
assumes *non-empty*: $\text{filter } P \text{ } xs \neq []$
shows $P (\text{hd } (\text{filter } P \text{ } xs))$
 <proof>

lemma *index-elem*: $x \in \text{set } xs \implies \exists i < \text{length } xs. x = xs ! i$
 <proof>

lemma *filter-hd-P-rep-indep*:
 $\llbracket \forall x. P \text{ } x \text{ } x; \forall a \text{ } b. P \text{ } a \text{ } a \longrightarrow P \text{ } a \text{ } b \longrightarrow P \text{ } x \text{ } b; \text{filter } (P \text{ } x) \text{ } xs \neq [] \rrbracket \implies$
 $\text{hd } (\text{filter } (P \text{ } (\text{hd } (\text{filter } (P \text{ } x) \text{ } xs))) \text{ } xs) = \text{hd } (\text{filter } (P \text{ } x) \text{ } xs)$
 <proof>

lemma *take-Suc-not-last*:
 $\bigwedge n. \llbracket x \in \text{set } (\text{take } (\text{Suc } n) \text{ } xs); x \neq xs ! n; n < \text{length } xs \rrbracket \implies x \in \text{set } (\text{take } n \text{ } xs)$
 <proof>

lemma *P-eq-list-filter*: $\forall x \in \text{set } xs. P \text{ } x = Q \text{ } x \implies \text{filter } P \text{ } xs = \text{filter } Q \text{ } xs$
 <proof>

lemma *hd-filter-take-more*: $\bigwedge n \text{ } m. \llbracket \text{filter } P \text{ } (\text{take } n \text{ } xs) \neq []; n \leq m \rrbracket \implies$
 $\text{hd } (\text{filter } P \text{ } (\text{take } n \text{ } xs)) = \text{hd } (\text{filter } P \text{ } (\text{take } m \text{ } xs))$
 <proof>

end

4 Definitions of Procedures

theory *ProcedureSpecs*
imports *General Simpl.Vcg*
begin

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

```

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

```

procedures

```

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

```

procedures

```

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

```

procedures

```

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

```

```

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

```

procedures

```

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

```

procedures

```

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

```

procedures

```

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

```

end

5 Proof of Procedure Eval

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

lemma (**in** *Eval-impl*) *Eval-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC Eval } (\prime p, \prime \text{varval}, \prime R)$
 $\{t. t \text{ may-not-modify-globals } \sigma\}$
<proof>

lemma (**in** *Eval-impl*) *Eval-spec*:
shows $\forall \sigma t \text{ bdt1}. \Gamma \vdash$
 $\{\sigma. \text{Dag } \prime p \prime \text{low } \prime \text{high } t \wedge \text{bdt } t \prime \text{var} = \text{Some } \text{bdt1}\}$
 $\prime R := \text{PROC Eval}(\prime p, \prime \text{varval})$
 $\{\prime R = \text{eval } \text{bdt1 } \sigma \text{varval}\}$
<proof>

end

6 Proof of Procedure Levellist

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

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

lemma (**in** *Levellist-impl*) *Levellist-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \prime \text{levellist} := \text{PROC Levellist } (\prime p, \prime m, \prime \text{levellist})$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{mark}, \text{next}]\}$
<proof>

lemma *all-stop-cong*: $(\forall x. P x) = (\forall x. P x)$
<proof>

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

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

lemma *dag-Null [simp]*: $\text{dag Null l r} = \text{Tip}$

$\langle \text{proof} \rangle$

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

lemma *first-simps* [*simp*]:
 $\text{first } [] = \text{Null}$
 $\text{first } (r\#rs) = r$
 $\langle \text{proof} \rangle$

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

lemma *Levellist-unique*:
assumes ll : *Levellist hds next ll*
assumes ll' : *Levellist hds next ll'*
shows $ll=ll'$
 $\langle \text{proof} \rangle$

lemma *Levellist-unique-ex-conj-simp* [*simp*]:
 $\text{Levellist } hds \text{ next } ll \Longrightarrow (\exists ll. \text{Levellist } hds \text{ next } ll \wedge P ll) = P ll$
 $\langle \text{proof} \rangle$

lemma *in-set-concat-idx*:
 $x \in \text{set } (\text{concat } xss) \Longrightarrow \exists i < \text{length } xss. x \in \text{set } (xss!i)$
 $\langle \text{proof} \rangle$

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

lemma *wf-levellist-subset*:
assumes wf-ll : *wf-levellist t ll ll' var*
shows $\text{set } (\text{concat } ll') \subseteq \text{set } (\text{concat } ll) \cup \text{set-of } t$
 $\langle \text{proof} \rangle$

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

lemma *Levellist-length*: $\text{Levellist hds } p ll \implies \text{length } ll = \text{length hds}$
 $\langle \text{proof} \rangle$

lemma *map-update*:
 $\bigwedge i. i < \text{length } xss \implies \text{map } f (xss[i := xs]) = (\text{map } f xss) [i := f xs]$
 $\langle \text{proof} \rangle$

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

shows $\forall ll \sigma t. \Gamma, \Theta \vdash_t$
 $\{\sigma. \text{Dag } 'p \text{ 'low 'high } t \wedge ('p \neq \text{Null} \longrightarrow ('p \rightarrow 'var) < \text{length } 'levellist) \wedge$
 $\text{ordered } t \text{ 'var} \wedge \text{Levellist } 'levellist \text{ 'next } ll \wedge$
 $(\forall n \in \text{set-of } t.$
 $\text{if } 'mark \ n = 'm$
 $\text{then } n \in \text{set } (ll ! 'var \ n) \wedge$
 $(\forall nt \ p. \text{Dag } n \text{ 'low 'high } nt \wedge p \in \text{set-of } nt$
 $\longrightarrow 'mark \ p = 'm)$
 $\text{else } n \notin \text{set } (\text{concat } ll))\}$
 $'levellist := \text{PROC } \text{Levellist } ('p, 'm, 'levellist)$
 $\{\exists ll'. \text{Levellist } 'levellist \text{ 'next } ll' \wedge \text{wf-levellist } t \ ll \ ll' \sigma_{var} \wedge$
 $\text{wf-marking } t \ \sigma_{mark} \ 'mark \ \sigma_m \wedge$
 $(\forall p. p \notin \text{set-of } t \longrightarrow \sigma_{next} \ p = 'next \ p)$
 $\}$
 $\langle \text{proof} \rangle$

lemma *allD*: $\forall ll. P ll \implies P ll$
 $\langle \text{proof} \rangle$

lemma *replicate-spec*: $\llbracket \forall i < n. xs ! i = x; n = \text{length } xs \rrbracket$
 $\implies \text{replicate } (\text{length } xs) \ x = xs$
 $\langle \text{proof} \rangle$

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

shows $\forall \sigma t. \Gamma, \Theta \vdash_t$
 $\{\sigma. \text{Dag } 'p \text{ 'low 'high } t \wedge (\forall i < \text{length } 'levellist. 'levellist ! i = \text{Null}) \wedge$
 $\text{length } 'levellist = 'p \rightarrow 'var + 1 \wedge$
 $\text{ordered } t \text{ 'var} \wedge (\forall n \in \text{set-of } t. 'mark \ n = (\neg 'm))\}$
 $'levellist := \text{PROC } \text{Levellist } ('p, 'm, 'levellist)$
 $\{\exists ll. \text{Levellist } 'levellist \text{ 'next } ll \wedge \text{wf-ll } t \ ll \ \sigma_{var} \wedge$
 $\text{length } 'levellist = \sigma_p \rightarrow \sigma_{var} + 1 \wedge$
 $\text{wf-marking } t \ \sigma_{mark} \ 'mark \ \sigma_m \wedge$
 $(\forall p. p \notin \text{set-of } t \longrightarrow \sigma_{next} \ p = 'next \ p)\}$
 $\langle \text{proof} \rangle$

end

7 Proof of Procedure ShareRep

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

lemma (in *ShareRep-impl*) *ShareRep-modifies*:
shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC ShareRep } ('nodeslist, 'p)$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [rep]\}$
 $\langle proof \rangle$

lemma *hd-filter-cons*:

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

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

shows

$\forall \sigma \text{ ns. } \Gamma, \Theta \vdash_t$
 $\{\sigma. \text{List } 'nodeslist \text{ } 'next \text{ ns} \wedge$
 $(\forall no \in \text{set } ns. no \neq \text{Null} \wedge$
 $((no \rightarrow 'low = \text{Null}) = (no \rightarrow 'high = \text{Null})) \wedge$
 $(\text{isLeaf-pt } 'p \text{ } 'low \text{ } 'high \longrightarrow \text{isLeaf-pt } no \text{ } 'low \text{ } 'high) \wedge$
 $no \rightarrow 'var = 'p \rightarrow 'var) \wedge$
 $'p \in \text{set } ns\}$
 $\text{PROC ShareRep } ('nodeslist, 'p)$
 $\{\{\sigma_p \rightarrow 'rep = \text{hd } (\text{filter } (\lambda sn. \text{repNodes-eq } sn \text{ } \sigma_p \text{ } \sigma_{low} \text{ } \sigma_{high} \text{ } \sigma_{rep}) \text{ } ns)\} \wedge$
 $(\forall pt. pt \neq \sigma_p \longrightarrow pt \rightarrow \sigma_{rep} = pt \rightarrow 'rep) \wedge$
 $(\sigma_p \rightarrow 'rep \rightarrow \sigma_{var} = \sigma_p \rightarrow \sigma_{var})\}$
 $\langle proof \rangle$

end

8 Proof of Procedure ShareReduceRepList

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

lemma (in *ShareReduceRepList-impl*) *ShareReduceRepList-modifies*:

shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC ShareReduceRepList } ('nodeslist)$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [rep]\}$
 $\langle proof \rangle$

lemma *hd-filter-app*: $\llbracket \text{filter } P \text{ } xs \neq []; zs = xs @ ys \rrbracket \implies$

$\text{hd } (\text{filter } P \text{ } zs) = \text{hd } (\text{filter } P \text{ } xs)$

$\langle proof \rangle$

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

$\forall \sigma\ ns. \Gamma \vdash_t$
 $\{\sigma. List\ 'nodeslist\ 'next\ ns \wedge$
 $(\forall no \in set\ ns.$
 $no \neq Null \wedge ((no \rightarrow 'low = Null) = (no \rightarrow 'high = Null)) \wedge$
 $no \rightarrow 'low \notin set\ ns \wedge no \rightarrow 'high \notin set\ ns \wedge$
 $(isLeaf-pt\ no\ 'low\ 'high = (no \rightarrow 'var \leq 1)) \wedge$
 $(no \rightarrow 'low \neq Null \longrightarrow (no \rightarrow 'low) \rightarrow 'rep \neq Null) \wedge$
 $(('rep \times 'low)\ no \notin set\ ns)) \wedge$
 $var-eq\ ns\ 'var\}$
PROC *ShareReduceRepList* ('nodeslist)
 $\{\{(\forall no. no \notin set\ ns \longrightarrow no \rightarrow^\sigma rep = no \rightarrow 'rep) \wedge$
 $(\forall no \in set\ ns. no \rightarrow 'rep \neq Null \wedge$
 $(if\ (('rep \times^{\sigma low})\ no = ('rep \times^{\sigma high})\ no \wedge no \rightarrow^{\sigma low} \neq Null)$
 $then\ (no \rightarrow 'rep = ('rep \times^{\sigma low})\ no)$
 $else\ ((no \rightarrow 'rep) \in set\ ns \wedge no \rightarrow 'rep \rightarrow 'rep = no \rightarrow 'rep \wedge$
 $(\forall no1 \in set\ ns.$
 $(('rep \times^{\sigma high})\ no1 = ('rep \times^{\sigma high})\ no \wedge$
 $('rep \times^{\sigma low})\ no1 = ('rep \times^{\sigma low})\ no) = (no \rightarrow 'rep = no1 \rightarrow 'rep))))\}\}$
 $\langle proof \rangle$

end

9 Proof of Procedure Repoint

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

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

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

shows $\forall \sigma. \Gamma \vdash \{\sigma\} 'p ::= PROC\ Repoint\ ('p)$
 $\{t. t\ may-only-modify-globals\ \sigma\ in\ [low,high]\}$
 $\langle proof \rangle$

lemma *low-high-exchange-dag*:

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

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

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

shows $\bigwedge q. Dag\ q\ (rep \times low)\ (rep \times high)\ rt \implies$
 $Dag\ q\ (rep \times lowa)\ (rep \times higha)\ rt$

$\langle proof \rangle$

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

shows

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

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

shows

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

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

shows

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

$\langle \text{proof} \rangle$

end

10 Proof of Procedure Normalize

theory *NormalizeTotalProof* **imports** *LevellistProof ShareReduceRepListProof*
RepointProof **begin**

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

lemma (in *Normalize-impl*) *Normalize-modifies*:

shows

$\forall \sigma. \Gamma \vdash \{\sigma\}$ $\acute{p} ::= \text{PROC Normalize } (\acute{p})$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{rep,mark,low,high,next}]\}$
 $\langle \text{proof} \rangle$

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

shows $\forall \sigma \text{ pret prebdt. } \Gamma \vdash_t$

$\{\!\{ \sigma. \text{Dag } \acute{p} \text{ 'low 'high pret } \wedge \text{ordered pret 'var } \wedge$

$\acute{p} \neq \text{Null} \wedge (\forall n. n \in \text{set-of pret} \longrightarrow \text{'mark } n = \text{'mark } \acute{p}) \wedge$
 $\text{bdt pret } \acute{var} = \text{Some prebdt} \}$
 $\acute{p} ::= \text{PROC Normalize}(\acute{p})$
 $\{ (\forall pt. pt \notin \text{set-of pret}$
 $\longrightarrow \sigma_{rep} pt = \text{'rep } pt \wedge \sigma_{low} pt = \text{'low } pt \wedge \sigma_{high} pt = \text{'high } pt \wedge$
 $\sigma_{mark} pt = \text{'mark } pt \wedge \sigma_{next} pt = \text{'next } pt) \wedge$
 $(\exists \text{postt. Dag } \acute{p} \text{'low } \text{'high } \text{postt} \wedge \text{reduced } \text{postt} \wedge$
 $\text{shared } \text{postt } \sigma_{var} \wedge \text{ordered } \text{postt } \sigma_{var} \wedge$
 $\text{set-of } \text{postt} \subseteq \text{set-of } \text{pret} \wedge$
 $(\exists \text{postbdt. bdt } \text{postt } \sigma_{var} = \text{Some } \text{postbdt} \wedge \text{prebdt} \sim \text{postbdt})) \wedge$
 $(\forall \text{no. no} \in \text{set-of } \text{pret} \longrightarrow \text{'mark } \text{no} = (\neg \sigma_{mark} \sigma_p)) \}$
 $\langle \text{proof} \rangle$

end

References

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