

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.

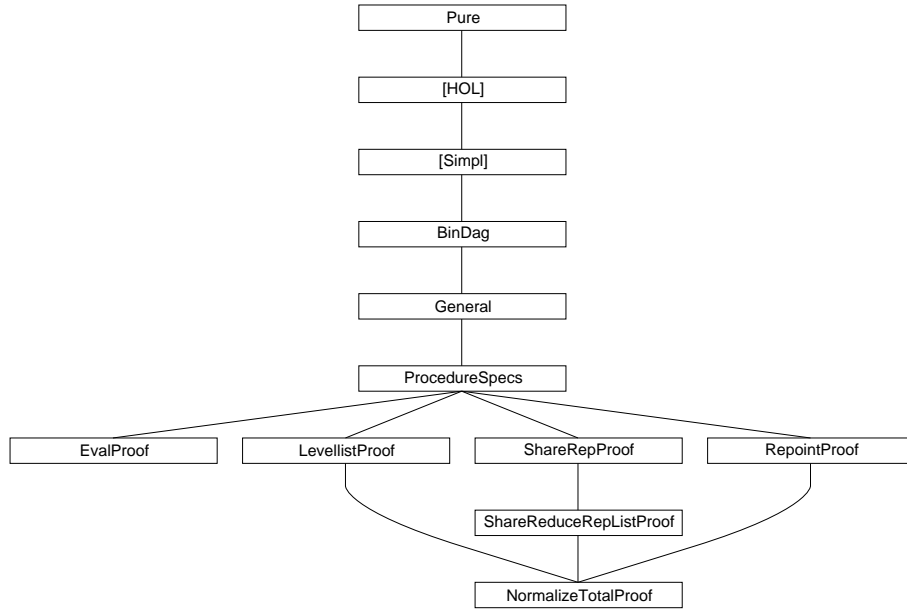
BDD-Normalisation

Veronika Ortner and Norbert Schirmer

December 12, 2009

Contents

1	Introduction	3
2	BDD Abstractions	3
3	General Lemmas on BDD Abstractions	8
4	Definitions of Procedures	19
5	Proof of Procedure Eval	21
6	Proof of Procedure Levellist	22
7	Proof of Procedure ShareRep	24
8	Proof of Procedure ShareReduceRepList	25
9	Proof of Procedure Reprint	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/Heap
begin
  
```

```

datatype dag = Tip | Node dag ref dag
  
```

```

lemma [simp]: Node lt a rt ≠ lt
  <proof>
  
```

```

lemma [simp]: lt ≠ Node lt a rt
  <proof>
  
```

```

lemma [simp]: Node lt a rt ≠ rt
  <proof>
  
```

```

lemma [simp]: rt ≠ 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) \Longrightarrow *subdag t lt* \wedge *subdag t rt*

<proof>

lemma *subdag-not-sym*: $\bigwedge t. \llbracket \text{subdag } s \ t; \text{subdag } t \ s \rrbracket \Longrightarrow P$

<proof>

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*

<proof>

lemma *le-dag-trans*:

fixes *x::dag* **and** *y* **and** *z*

assumes *x-y*: *x* \leq *y* **and** *y-z*: *y* \leq *z*

shows *x* \leq *z*

<proof>

lemma *le-dag-antisym*:

fixes *x::dag* **and** *y*

assumes *x-y*: *x* \leq *y* **and** *y-x*: *y* \leq *x*

```

shows  $x = y$ 
  ⟨proof⟩

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

instance ⟨proof⟩

end

lemma less-dag-Tip [simp]:  $\neg (x < Tip)$ 
  ⟨proof⟩

lemma less-dag-Node:  $x < (Node\ l\ a\ r) =$ 
   $(x \leq l \vee x \leq r)$ 
  ⟨proof⟩

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:  $\llbracket DAG\ y; x < y \rrbracket \implies DAG\ x$ 
  ⟨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⟩

consts Dag::  $ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow dag \Rightarrow bool$ 
primrec

```

$$\begin{aligned}
Dag\ p\ l\ r\ Tip &= (p = Null) \\
Dag\ p\ l\ r\ (Node\ lt\ a\ rt) &= (p = a \wedge p \neq Null \wedge \\
&\quad Dag\ (l\ p)\ l\ r\ lt \wedge Dag\ (r\ p)\ l\ r\ rt)
\end{aligned}$$

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

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

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

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

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

lemma *Dag-upd-same-l-lemma*: $\bigwedge p. p \neq Null \implies \neg Dag\ p\ (l(p:=p))\ r\ t$
 $\langle proof \rangle$

lemma *Dag-upd-same-l* [*simp*]: $Dag\ p\ (l(p:=p))\ r\ t = (p=Null \wedge t=Tip)$
 $\langle proof \rangle$

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

lemma *Dag-upd-same-r-lemma*: $\bigwedge p. p \neq Null \implies \neg Dag\ p\ l\ (r(p:=p))\ t$
 $\langle proof \rangle$

lemma *Dag-upd-same-r* [*simp*]: $Dag\ p\ l\ (r(p:=p))\ t = (p=Null \wedge t=Tip)$
 $\langle proof \rangle$

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

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

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

lemma *Dag-update-rI* [*intro*]:

$\llbracket \text{Dag } p \text{ l } r \text{ t}; q \notin \text{set-of } t \rrbracket \implies \text{Dag } p \text{ l } (r(q := y)) \text{ t}$
 $\langle \text{proof} \rangle$

lemma *Dag-unique*: $\bigwedge p \text{ t}2. \text{Dag } p \text{ l } r \text{ t}1 \implies \text{Dag } p \text{ l } r \text{ t}2 \implies t1=t2$
 $\langle \text{proof} \rangle$

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

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

lemma *Dag-root-not-in-subdag-l*[*simp,intro*]:
 $\text{Dag } (l \ p) \text{ l } r \text{ t} \implies p \notin \text{set-of } t$
 $\langle \text{proof} \rangle$

lemma *Dag-root-not-in-subdag-r*[*simp,intro*]:
 $\text{Dag } (r \ p) \text{ l } r \text{ t} \implies p \notin \text{set-of } t$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

constdefs
 $isDag::ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow bool$
 $isDag \ p \ l \ r \equiv (\exists t. \text{Dag } p \ l \ r \ t)$
 $dag::ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow dag$
 $dag \ p \ l \ r \equiv \text{THE } t. \text{Dag } p \ l \ r \ t$

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

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

end

3 General Lemmas on BDD Abstractions

theory *General* imports *BinDag* begin

consts *subdag-eq*:: $dag \Rightarrow dag \Rightarrow bool$
defs *subdag-eq-def*:
 $subdag-eq \ t_1 \ t_2 == (t_1 = t_2 \vee subdag \ t_1 \ t_2)$

consts *root* :: $dag \Rightarrow ref$

primrec

$root \ Tip = \text{Null}$

$root \ (\text{Node } l \ a \ r) = a$

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

$isLeaf \ Tip = \text{False}$

$| \ isLeaf \ (\text{Node } Tip \ v \ Tip) = \text{True}$

$| \ isLeaf \ (\text{Node } (\text{Node } l \ v_1 \ r) \ v_2 \ Tip) = \text{False}$

$| \ isLeaf \ (\text{Node } Tip \ v_1 \ (\text{Node } l \ v_2 \ r)) = \text{False}$

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

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

$bdt-fn \ Tip = (\lambda bdtvar . \text{None})$

$| \ bdt-fn \ (\text{Node } Tip \ vref \ Tip) =$

$(\lambda bdtvar .$

$\text{if } (bdtvar \ vref = 0)$

```

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

```

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

consts $eval :: bdt \Rightarrow bool \text{ list} \Rightarrow bool$

primrec

$eval\ Zero\ env = False$

$eval\ One\ env = True$

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

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

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

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

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

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

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

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

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

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

$| ordered\ (Node\ (Node\ l1\ v1\ r1)\ v\ (Node\ l2\ v2\ r2)) =$
 $(\lambda\ var.\ (var\ v1 < var\ v \wedge var\ v2 < var\ v) \wedge$
 $(ordered\ (Node\ l1\ v1\ r1)\ var) \wedge (ordered\ (Node\ l2\ v2\ r2)\ var))$

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

$| ordered\ (Node\ Tip\ v\ r) =$
 $(\lambda\ var.\ (var\ (root\ r) < var\ v) \wedge (ordered\ r\ var))$

| *ordered* (*Node l v Tip*) =
 (λ *var*. (*var* (*root l*) < *var v*) ∧ (*ordered l var*))

consts *max-var* :: *bdt* ⇒ *nat*

primrec

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*: $\llbracket \text{bdt } (\text{Node } l \ v \ r) \ \text{var} = \text{Some } x; \text{var } (\text{root } (\text{Node } l \ v \ r)) = (0 :: \text{nat}) \rrbracket \implies x = \text{Zero}$
 ⟨*proof*⟩

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

(*bdt t var* = *Some One*) = (∃ *p*. *t* = *Node Tip p Tip* ∧ *var p* = 1)
 ⟨*proof*⟩

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

(*bdt t var* = *Some Zero*) = (∃ *p*. *t* = *Node Tip p Tip* ∧ *var p* = 0)
 ⟨*proof*⟩

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

(*bdt t var* = *Some* (*Bdt-Node bdt1 v bdt2*)) =
 (∃ *p l r*. *t* = *Node l p r* ∧ *bdt l var* = *Some bdt1* ∧ *bdt r var* = *Some bdt2* ∧
 1 < *v* ∧ *var p* = *v*)
 ⟨*proof*⟩

lemma *balanced-bdt*:

∧ *p bdt1*. $\llbracket \text{Dag } p \ \text{low } \text{high } t; \text{bdt } t \ \text{var} = \text{Some } \text{bdt1}; \text{no} \in \text{set-of } t \rrbracket$
 ⇒ (*low no* = *Null*) = (*high no* = *Null*)
 ⟨*proof*⟩

consts *dag-map* :: (*ref* ⇒ *ref*) ⇒ *dag* ⇒ *dag*

primrec

dag-map f Tip = *Tip*

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

consts *wf-marking* :: *dag* ⇒ (*ref* ⇒ *bool*) ⇒ (*ref* ⇒ *bool*) ⇒ *bool* ⇒ *bool*

defs *wf-marking-def*: *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*)

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

defs *dag-in-levellist-def* : *dag-in-levellist t levellist var == t ≠ Tip* \longrightarrow
 $(\forall st. \text{subdag-eq } t \text{ st} \longrightarrow \text{root st} \in \text{set } (\text{levellist ! } (\text{var } (\text{root st}))))$

lemma *set-of-nn*: $\llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t \rrbracket \implies n \neq \text{Null}$
 $\langle \text{proof} \rangle$

lemma *subnodes-ordered* [rule-format]:
 $\forall p. n \in \text{set-of } t \longrightarrow \text{Dag } p \text{ low high } t \longrightarrow \text{ordered } t \text{ var} \longrightarrow \text{var } n \leq \text{var } p$
 $\langle \text{proof} \rangle$

lemma *ordered-set-of*:
 $\bigwedge x. \llbracket \text{ordered } t \text{ var}; x \in \text{set-of } t \rrbracket \implies \text{var } x \leq \text{var } (\text{root } t)$
 $\langle \text{proof} \rangle$

lemma *dag-setofD*: $\bigwedge p \text{ low high } n. \llbracket \text{Dag } p \text{ low high } t; n \in \text{set-of } t \rrbracket \implies$
 $(\exists nt. \text{Dag } n \text{ low high } nt) \wedge (\forall nt. \text{Dag } n \text{ low high } nt \longrightarrow \text{set-of } nt \subseteq \text{set-of } t)$
 $\langle \text{proof} \rangle$

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

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*: $\text{not } \leq lt \implies \text{not } \leq (\text{Node } lt \text{ } p \text{ } rt)$
 $\langle \text{proof} \rangle$

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

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

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

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

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

constdefs *wf-ll* :: $\text{dag} \Rightarrow \text{ref list list} \Rightarrow (\text{ref} \Rightarrow \text{nat}) \Rightarrow \text{bool}$
wf-ll *t levellist var* ==
 ($\forall p. p \in \text{set-of } t \longrightarrow p \in \text{set } (\text{levellist } ! \text{ var } p)$) \wedge
 ($\forall k < \text{length levellist}. \forall p \in \text{set } (\text{levellist } ! k). p \in \text{set-of } t \wedge \text{var } p = k$)

constdefs *cong-eval* :: $\text{bdt} \Rightarrow \text{bdt} \Rightarrow \text{bool}$ (**infix** ~ 60)
cong-eval *bdt₁ bdt₂* $\equiv \text{eval } bdt_1 = \text{eval } bdt_2$

lemma *cong-eval-sym*: $l \sim r = r \sim l$
 <proof>

lemma *cong-eval-trans*: $\llbracket l \sim r; r \sim t \rrbracket \implies l \sim t$
 <proof>

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>

constdefs *null-comp* :: $(\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow (\text{ref} \Rightarrow \text{ref})$ (**infix** $\times 60$)
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>

constdefs *Nodes* :: $\text{nat} \Rightarrow \text{ref list list} \Rightarrow \text{ref set}$
Nodes *i 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
Dags! : $\llbracket \text{set-of } t \subseteq \text{nodes}; \text{Dag } p \text{ low high } t; t \neq \text{Tip} \rrbracket$
 $\implies t \in \text{Dags nodes low high}$

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

consts *isLeaf-pt* :: $ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow bool$
defs *isLeaf-pt-def* : $isLeaf-pt p low high == (low p = Null \wedge high p = Null)$

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

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

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

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

lemma *isomorphic-dags-eq-sym*: $isomorphic-dags-eq st_1 st_2 var = isomorphic-dags-eq st_2 st_1 var$
 ⟨*proof*⟩

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

defs *shared-def*:
 $shared t var == \forall st_1 st_2. (st_1 <= t \wedge st_2 <= t) \longrightarrow isomorphic-dags-eq st_1 st_2 var$

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

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

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

primrec

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

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

$\langle proof \rangle$

lemma *replicate-length [simp]* : $length (replicate\ n\ x) = n$
 $\langle proof \rangle$

lemma *no-in-one-ll*:
[[*wf-ll pret levellista var; i < length levellista; j < length levellista;*
no ∈ set (levellista ! i); i ≠ j]]
⇒ *no ∉ set (levellista ! j)*
 $\langle proof \rangle$

lemma *nodes-in-wf-ll*:
[[*wf-ll pret levellista var; i < length levellista; no ∈ set (levellista ! i)*]]
⇒ *var no = i ∧ no ∈ set-of pret*
 $\langle proof \rangle$

lemma *subelem-set-of-low*:
∧ *p. [[x ∈ set-of t; x ≠ Null; low x ≠ Null; Dag p low high t]]*
⇒ *(low x) ∈ set-of t*
 $\langle proof \rangle$

lemma *subelem-set-of-high*:
∧ *p. [[x ∈ set-of t; x ≠ Null; high x ≠ Null; Dag p low high t]]*
⇒ *(high x) ∈ set-of t*
 $\langle proof \rangle$

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

lemma *Nodes-levellist-subset-t*:
[[*wf-ll t levellist var; i ≤ length levellist*]] ⇒ *Nodes i levellist ⊆ set-of t*
 $\langle proof \rangle$

lemma *bdt-child*:
[[*bdt (Node (Node llt l rlt) p (Node lrt r rrt)) var = Some bdt1*]]
⇒ ∃ *lbd rbd. bdt (Node llt l rlt) var = Some lbd ∧*
bdt (Node lrt r rrt) var = Some rbd
 $\langle proof \rangle$

lemma *subbdt-ex-dag-def*:
∧ *bdt1 p. [[Dag p low high t; bdt t var = Some bdt1; Dag no low high not;*
no ∈ set-of t]] ⇒ ∃ *bdt2. bdt not var = Some bdt2*
 $\langle proof \rangle$

lemma *subbdt-ex*:
∧ *bdt1. [[(Node lst stp rst) ≤ t; bdt t var = Some bdt1]]*
⇒ ∃ *bdt2. bdt (Node lst stp rst) var = Some bdt2*

$\langle \text{proof} \rangle$

lemma *var-ordered-children*:

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

$\langle \text{proof} \rangle$

lemma *nort-null-comp*:

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

$\langle \text{proof} \rangle$

lemma *wf-ll-Nodes-pret*:

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

lemma *bdt-Some-var1-One*:

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

lemma *bdt-Some-var0-Zero*:

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

lemma *reduced-children-parent*:

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

lemma *Nodes-subset*: *Nodes* *i levellista* \subseteq *Nodes (Suc i) levellista*

$\langle \text{proof} \rangle$

lemma *Nodes-levellist*:

$\llbracket \text{wf-ll pret levellista var}; \text{nb} < \text{length levellista}; p \in \text{Nodes nb levellista} \rrbracket$

$\implies p \notin \text{set } (\text{levellista } ! \text{ nb})$
 <proof>

lemma *Nodes-var-pret:*

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

lemma *Dags-root-in-Nodes:*

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

lemma *subdag-dag:*

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

lemma *Dags-subdags:*

assumes *t-in-Dags:* $t \in \text{Dags nodes low high}$ **and**
st-t: $st \leq t$ **and**
st-nTip: $st \neq \text{Tip}$
shows $st \in \text{Dags nodes low high}$
 <proof>

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:

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

dags-in-mixed-levels:

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

dags-in-high-level:

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

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

<proof>

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

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

lemma *restrict-root-Node*:
 $\llbracket t \in \text{Dags } (\text{repc } \text{'Nodes (Suc nb) levellista}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}); t \notin \text{Dags } (\text{repc } \text{'Nodes nb levellista}) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high});$
 $\text{ordered } t \text{ var}; \forall \text{ no} \in \text{Nodes (Suc nb) levellista. var } (\text{repc no}) \leq \text{var no} \wedge \text{repc } (\text{repc no}) = \text{repc no}; \text{ wf-ll pret levellista } \text{var}; \text{ nb} < \text{length levellista}; \text{repc } \text{'Nodes (Suc nb) levellista} \subseteq \text{Nodes (Suc nb) levellista} \rrbracket$
 $\implies \exists q. \text{Dag } (\text{repc } q) (\text{repc } \propto \text{low}) (\text{repc } \propto \text{high}) t \wedge q \in \text{set (levellista ! nb)}$
 ⟨proof⟩

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

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

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

lemma *set-take-le*: $\bigwedge i j. i \leq j \implies \text{set (take } i \text{ xs)} \subseteq \text{set (take } j \text{ xs)}$
 ⟨proof⟩

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

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

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

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

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

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

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

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

lemma *hd-filter-prop*:
assumes *non-empty*: $\text{filter } P \ xs \neq []$
shows $P \ (\text{hd } (\text{filter } P \ xs))$
 ⟨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*:
 $\llbracket \forall x. P \ x \ x; \forall a \ b. P \ x \ a \longrightarrow P \ a \ b \longrightarrow P \ x \ b; \text{filter } (P \ x) \ xs \neq [] \rrbracket \implies \text{hd } (\text{filter } (P \ (\text{hd } (\text{filter } (P \ x) \ xs))) \ xs) = \text{hd } (\text{filter } (P \ x) \ xs)$
 ⟨proof⟩

lemma *take-Suc-not-last*:

$\bigwedge n. \llbracket x \in \text{set } (\text{take } (\text{Suc } n) \text{ } xs); x \neq \text{xs}!n; n < \text{length } xs \rrbracket \implies x \in \text{set } (\text{take } n \text{ } xs)$

<proof>

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

<proof>

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

<proof>

end

4 Definitions of Procedures

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

record *globals* =

var-' :: ref \Rightarrow nat

low-' :: ref \Rightarrow ref

high-' :: ref \Rightarrow ref

rep-' :: ref \Rightarrow ref

mark-' :: ref \Rightarrow bool

next-' :: ref \Rightarrow ref

record *'g bdd-state* = *'g state* +

varval-' :: bool list

p-' :: ref

R-' :: bool

levellist-' :: ref list

nodeslist-' :: ref

node-': ref

m-' :: bool

n-' :: nat

procedures

Eval (*p*, *varval* | *R*) =

IF (*p* \rightarrow *'var* = 0) *THEN* *'R* ::= *False*

ELSE IF (*p* \rightarrow *'var* = 1) *THEN* *'R* ::= *True*

ELSE IF (*'varval* ! (*p* \rightarrow *'var*)) *THEN CALL* *Eval* (*p* \rightarrow *'high*, *'varval*, *'R*)

ELSE CALL *Eval* (*p* \rightarrow *'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
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* beginlemma (in *Eval-impl*) *Eval-modifies*:

```

shows ∀σ. Γ ⊢ {σ} PROC Eval ( 'p, 'varval, 'R )
      { t. t may-not-modify-globals σ }
⟨proof⟩

```

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

```

shows ∀σ t bdt1. Γ ⊢
  {σ. Dag 'p 'low 'high t ∧ bdt t 'var = Some bdt1 }
  'R ::= PROC Eval ( 'p, 'varval )
  { 'R = eval bdt1 σ varval }
⟨proof⟩

```

end

6 Proof of Procedure Levellist

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

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

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

shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ 'levellist } ::= \text{PROC Levellist } (\text{'p}, \text{'m}, \text{'levellist})$
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{mark}, \text{next}]\}$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *Dag-RefD*:

$\llbracket \text{Dag } p \text{ l r t}; p \neq \text{Null} \rrbracket \implies$
 $\exists lt \text{ rt}. t = \text{Node } lt \text{ p } rt \wedge \text{Dag } (l \text{ p}) \text{ l r } lt \wedge \text{Dag } (r \text{ p}) \text{ l r } rt$
 $\langle \text{proof} \rangle$

lemma *Dag-unique-ex-conjI*:

$\llbracket \text{Dag } p \text{ l r t}; P t \rrbracket \implies (\exists t. \text{Dag } p \text{ l r } t \wedge P t)$
 $\langle \text{proof} \rangle$

lemma *dag-Null [simp]*: *dag Null l r = Tip*

$\langle \text{proof} \rangle$

lemma *list-ext*:

$\bigwedge ys. \llbracket \text{length } xs = \text{length } ys; \forall i < \text{length } xs. xs!i = ys!i \rrbracket \implies xs = ys$
 $\langle \text{proof} \rangle$

constdefs *first*:: *ref list* \Rightarrow *ref*

first ps \equiv *case ps of* $\llbracket \rrbracket \Rightarrow \text{Null} \mid (p \# rs) \Rightarrow p$

lemma *first-simps [simp]*:

first $\llbracket \rrbracket = \text{Null}$
first $(r \# rs) = r$
 $\langle \text{proof} \rangle$

constdefs *Levellist*:: *ref list* \Rightarrow (*ref* \Rightarrow *ref*) \Rightarrow (*ref list list*) \Rightarrow *bool*

Levellist hds next ll \equiv (*map 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 $ll=ll'$
 $\langle proof \rangle$

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

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

consts *wf-levellist* :: $dag \Rightarrow ref\ list\ list \Rightarrow ref\ list\ list \Rightarrow$
 $(ref \Rightarrow nat) \Rightarrow bool$

defs *wf-levellist-def*: $wf-levellist\ t\ levellist-old\ levellist-new\ var \equiv$
 $case\ t\ of\ Tip \Rightarrow levellist-old = levellist-new$
 $| (Node\ lt\ p\ rt) \Rightarrow$
 $(\forall q. q \in set-of\ t \longrightarrow q \in set\ (levellist-new\ !\ (var\ q))) \wedge$
 $(\forall i \leq var\ p. (\exists prx. (levellist-new\ !\ i) = prx@(levellist-old\ !\ i)$
 $\wedge (\forall pt \in set\ prx. pt \in set-of\ t \wedge var\ pt = i))) \wedge$
 $(\forall i. (var\ p) < i \longrightarrow (levellist-new\ !\ i) = (levellist-old\ !\ i)) \wedge$
 $(length\ levellist-new = length\ levellist-old)$

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

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

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

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

lemma (**in** *Levellist-impl*) *Levellist-spec-total'*:
shows $\forall ll\ \sigma\ t. \Gamma, \Theta \vdash_t$
 $\{\sigma. Dag\ 'p\ low\ 'high\ t \wedge ('p \neq Null \longrightarrow ('p \rightarrow 'var) < length\ 'levellist) \wedge$

$$\text{ordered } t \text{ 'var} \wedge \text{Levellist 'levellist 'next ll} \wedge$$

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

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

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

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

$$\text{ } \longrightarrow \text{'mark } p = \text{'m)}$$

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

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

$$\{\exists ll'. \text{Levellist 'levellist 'next ll}' \wedge \text{wf-levellist } t \text{ ll ll}' \sigma\text{var} \wedge$$

$$\text{wf-marking } t \sigma\text{mark 'mark} \sigma\text{m} \wedge$$

$$(\forall p. p \notin \text{set-of } t \longrightarrow \sigma\text{next } p = \text{'next } p)$$

$$\}$$

$$\langle \text{proof} \rangle$$

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

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

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

shows $\forall \sigma t. \Gamma, \Theta \vdash_t$

$$\{\sigma. \text{Dag 'p 'low 'high } t \wedge (\forall i < \text{length 'levellist. 'levellist ! } i = \text{Null}) \wedge$$

$$\text{length 'levellist} = \text{'p} \rightarrow \text{'var} + 1 \wedge$$

$$\text{ordered } t \text{ 'var} \wedge (\forall n \in \text{set-of } t. \text{'mark } n = (\neg \text{'m}))\}$$

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

$$\{\exists ll. \text{Levellist 'levellist 'next ll} \wedge \text{wf-ll } t \text{ ll} \sigma\text{var} \wedge$$

$$\text{length 'levellist} = \sigma\text{p} \rightarrow \sigma\text{var} + 1 \wedge$$

$$\text{wf-marking } t \sigma\text{mark 'mark} \sigma\text{m} \wedge$$

$$(\forall p. p \notin \text{set-of } t \longrightarrow \sigma\text{next } p = \text{'next } p)\}$$

$$\langle \text{proof} \rangle$$

end

7 Proof of Procedure ShareRep

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

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

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

lemma *hd-filter-cons*:

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

<proof>

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

shows

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

<proof>

end

8 Proof of Procedure ShareReduceRepList

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

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

shows $\forall \sigma. \Gamma \vdash \{\sigma\} \text{ PROC } \text{ShareReduceRepList } ('nodeslist)$

$\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [rep]\}$

<proof>

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

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

<proof>

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

defines *var-eq* $\equiv (\lambda ns \ var. (\forall no1 \in \text{set ns. } \forall no2 \in \text{set ns. } no1 \rightarrow var = no2 \rightarrow var))$

shows

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

```

then (no→'rep = ('rep ∘ σlow) no )
else ((no→'rep) ∈ set ns ∧ no→'rep→'rep = no→'rep ∧
      (∀ no1 ∈ set ns.
        (('rep ∘ σhigh) no1 = ('rep ∘ σhigh) no ∧
         ('rep ∘ σlow) no1 = ('rep ∘ σlow) no) = (no→'rep = no1→'
rep))))))}
⟨proof⟩

end

```

9 Proof of Procedure Reprint

theory *ReprintProof* **imports** *ProcedureSpecs* **begin**

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

lemma (**in** *Reprint-impl*) *Reprint-modifies*:
shows $\forall \sigma. \Gamma \vdash \{ \sigma \} \text{'} p ::= \text{PROC Reprint ('} p)$
 $\{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [low, high] \}$
⟨proof⟩

lemma *low-high-exchange-dag*:

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

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

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

shows $\bigwedge q. \text{Dag } q (rep \circ low) (rep \circ high) rt \implies$
 $\text{Dag } q (rep \circ lowa) (rep \circ higha) rt$

⟨proof⟩

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

shows

$\forall \sigma. \Gamma \vdash \{ \sigma \}$

$\text{'} p ::= \text{PROC Reprint ('} p)$

$\{\! \{ \forall \text{rept. } ((\text{Dag } ((\sigma_{rep} \circ id) \sigma_p) (\sigma_{rep} \circ \sigma_{low}) (\sigma_{rep} \circ \sigma_{high}) \text{rept})$

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

$\longrightarrow \text{Dag '} p \text{' } low \text{' } high \text{rept} \wedge$

$(\forall pt. pt \notin \text{set-of } \text{rept} \longrightarrow \sigma_{low} pt = \text{' } low \text{ } pt \wedge \sigma_{high} pt = \text{' } high \text{ } pt) \}$

⟨proof⟩

thm *low-high-exchange-dag*

<proof>
thm *heaps-eq-Dag-eq*
 <proof>

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

shows

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

thm *low-high-exchange-dag*
 <proof>

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

shows

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

<proof>

thm *low-high-exchange-dag*
 <proof>

end

10 Proof of Procedure Normalize

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

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

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

shows

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

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

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

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

$$\begin{aligned}
&\longrightarrow \sigma_{rep} pt = 'rep\ pt \wedge \sigma_{low} pt = 'low\ pt \wedge \sigma_{high} pt = 'high\ pt \wedge \\
&\quad \sigma_{mark} pt = 'mark\ pt \wedge \sigma_{next} pt = 'next\ pt) \wedge \\
&(\exists postt. \text{Dag } 'p\ 'low\ 'high\ postt \wedge \text{reduced postt} \wedge \\
&\text{shared postt } \sigma_{var} \wedge \text{ordered postt } \sigma_{var} \wedge \\
&\text{set-of postt} \subseteq \text{set-of pret} \wedge \\
&(\exists postbdt. \text{bdt postt } \sigma_{var} = \text{Some postbdt} \wedge \text{prebdt} \sim \text{postbdt})) \wedge \\
&(\forall no. no \in \text{set-of pret} \longrightarrow 'mark\ no = (\neg \sigma_{mark} \sigma_p)) \} \\
&\langle proof \rangle
\end{aligned}$$

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.