Abstract

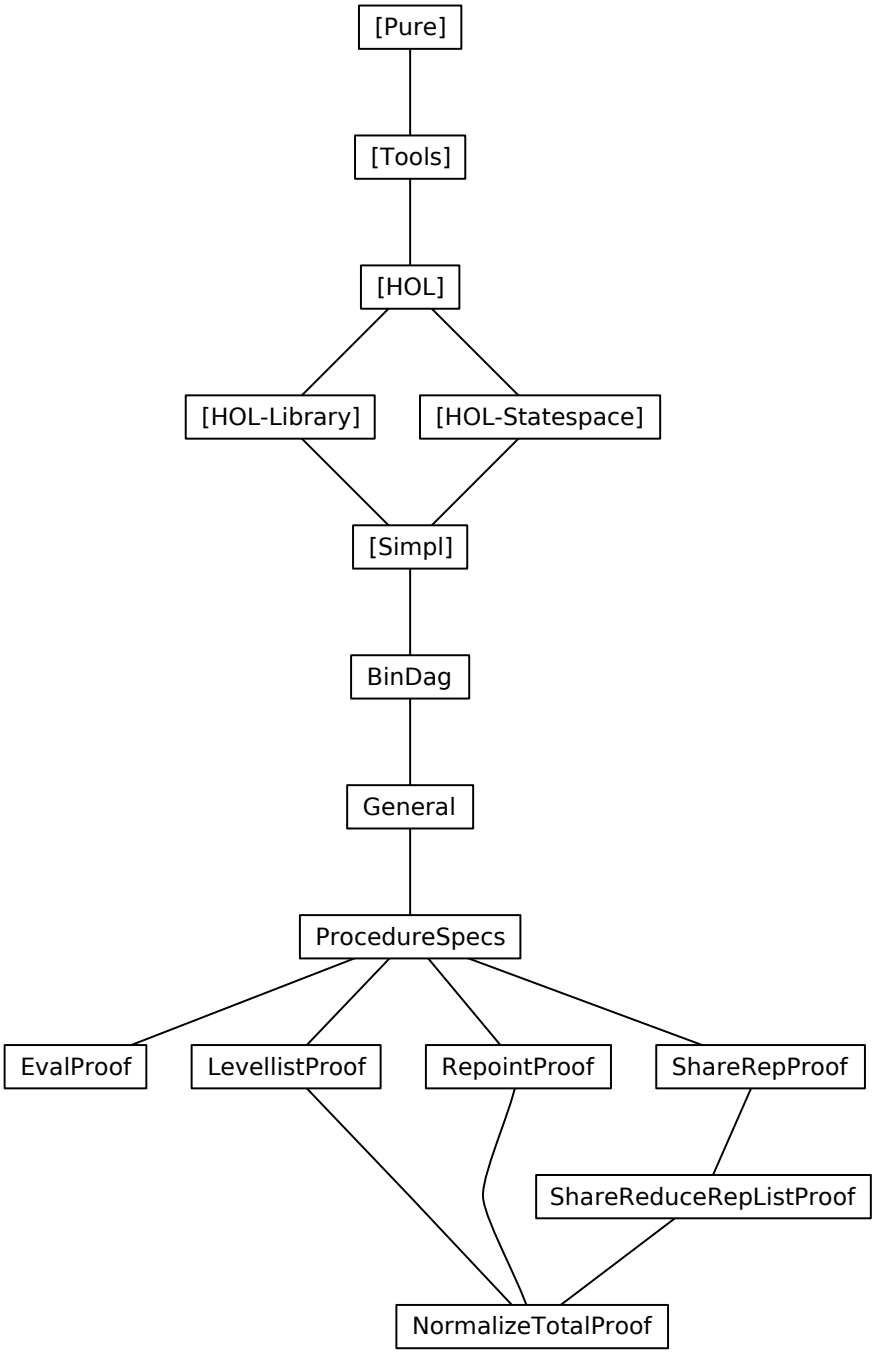
We present the verification of the normalisation of a binary decision diagram (BDD). The normalisation follows the original algorithm presented by Bryant in 1986 and transforms an ordered BDD in a reduced, ordered and shared BDD. The verification is based on Hoare logics.

BDD-Normalisation

Veronika Ortner and Norbert Schirmer

October 13, 2025

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
[HOL-Statespace]
[Simpl]
BinDag
General
ProcedureSpecs
EvalProof
LevellistProof
RepointProof
ShareRepProof
ShareReduceRepListProof
NormalizeTotalProof

1 Introduction

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

2 BDD Abstractions

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

datatype *dag* = *Tip* | *Node dag ref dag*

lemma [*simp*]: *Node lt a rt* ≠ *lt*
⟨*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* ⇒ *ref set* **where**
set-of-Tip: *set-of Tip* = {}
| *set-of-Node*: *set-of* (*Node lt a rt*) = {*a*} ∪ *set-of lt* ∪ *set-of rt*

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

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

lemma *subdag-size*: *subdag t s* ⟹ *size s* < *size t*
⟨*proof*⟩

lemma *subdag-neq*: *subdag t s* ⟹ *t*≠*s*
⟨*proof*⟩

lemma *subdag-trans* [*trans*]: *subdag t s* ⟹ *subdag s r* ⟹ *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*: $\bigwedge t.\ [\![subdag\ s\ t;\ subdag\ t\ s]\!] \Longrightarrow P$
⟨*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*: $(Node\ l\ a\ r) < x \Longrightarrow l < x \wedge r < x$
⟨*proof*⟩

lemma *less-dag-set-of*: $x < y \Longrightarrow set\text{-}of\ x \subseteq set\text{-}of\ y$
⟨*proof*⟩

lemma *le-dag-set-of*: $x \leq y \Longrightarrow set\text{-}of\ x \subseteq set\text{-}of\ y$
⟨*proof*⟩

lemma *DAG-less*: $DAG\ y \Longrightarrow x < y \Longrightarrow DAG\ x$
⟨*proof*⟩

lemma *less-DAG-set-of*:
assumes *x-less-y*: $x < y$
assumes *DAG-y*: $DAG\ y$
shows $set\text{-}of\ x \subset set\text{-}of\ y$
⟨*proof*⟩

lemma *in-set-of-decomp*:
$p \in set\text{-}of\ t = (\exists\ l\ r.\ t{=}(Node\ l\ p\ r) \vee subdag\ t\ (Node\ l\ p\ r))$
(**is** $?A = ?B$)
⟨*proof*⟩

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

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

lemma *Dag-Ref* [*simp*]:
$p{\neq}Null \Longrightarrow Dag\ p\ l\ r\ \ t = (\exists\ lt\ rt.\ t{=}Node\ lt\ p\ rt \wedge$
$Dag\ (l\ p)\ l\ r\ lt \wedge Dag\ (r\ p)\ l\ r\ rt)$
⟨*proof*⟩

lemma *Null-notin-Dag* [*simp*, *intro*]: $\bigwedge p\ l\ r.\ Dag\ p\ l\ r\ t \Longrightarrow Null \notin set\text{-}of\ t$
⟨*proof*⟩

theorem *notin-Dag-update-l* [*simp*]:
$\bigwedge p.\ q \notin set\text{-}of\ t \Longrightarrow Dag\ p\ (l(q := y))\ r\ \ t = Dag\ p\ l\ r\ t$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *Dag-upd-same-l-lemma*: $\bigwedge p.\ p{\neq}Null \Longrightarrow \neg\ Dag\ p\ (l(p{:=}p))\ r\ t$
⟨*proof*⟩

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

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

lemma *Dag-upd-same-r-lemma*: $\bigwedge p.\ p{\neq}Null \Longrightarrow \neg\ Dag\ p\ l\ (r(p{:=}p))\ t$
⟨*proof*⟩

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

lemma *Dag-update-l-new* [*simp*]: $[\![set\text{-}of\ t \subseteq set\ alloc]\!]$
$\Longrightarrow Dag\ p\ (l(new\ (set\ alloc) := x))\ r\ t = Dag\ p\ l\ r\ t$
⟨*proof*⟩

lemma *Dag-update-r-new* [*simp*]: $[\![set\text{-}of\ t \subseteq set\ alloc]\!]$
$\Longrightarrow Dag\ p\ l\ (r(new\ (set\ alloc) := x))\ t = Dag\ p\ l\ r\ t$
⟨*proof*⟩

lemma *Dag-update-lI* [*intro*]:
$[\![Dag\ p\ l\ r\ t;\ q \notin set\text{-}of\ t]\!] \Longrightarrow Dag\ p\ (l(q := y))\ r\ t$
⟨*proof*⟩

lemma *Dag-update-rI* [*intro*]:
$[\![Dag\ p\ l\ r\ t;\ q \notin set\text{-}of\ t]\!] \Longrightarrow Dag\ p\ l\ (r(q := y))\ t$
⟨*proof*⟩

lemma *Dag-unique*: $\bigwedge\ p\ t2.\ Dag\ p\ l\ r\ t1 \Longrightarrow Dag\ p\ l\ r\ t2 \Longrightarrow t1{=}t2$
⟨*proof*⟩

lemma *Dag-unique1*: $Dag\ p\ l\ r\ t \Longrightarrow \exists!t.\ Dag\ p\ l\ r\ t$
⟨*proof*⟩

lemma *Dag-subdag*: $\bigwedge\ p.\ Dag\ p\ l\ r\ t \Longrightarrow subdag\ t\ s \Longrightarrow \exists\ q.\ Dag\ q\ l\ r\ s$
⟨*proof*⟩

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

lemma *Dag-root-not-in-subdag-r* [*simp, intro*]:
assumes $Dag\ (r\ p)\ l\ r\ t$

shows $p \notin$ *set-of* t
⟨*proof*⟩

lemma *Dag-is-DAG*: $\bigwedge p\ l\ r.\ Dag\ p\ l\ r\ t \Longrightarrow DAG\ t$
⟨*proof*⟩

lemma *heaps-eq-Dag-eq*:
$\bigwedge p.\ \forall x \in$ *set-of* $t.\ l\ x = l'\ x \wedge r\ x = r'\ x$
$\Longrightarrow Dag\ p\ l\ r\ t = Dag\ p\ l'\ r'\ t$
⟨*proof*⟩

lemma *heaps-eq-DagI1*:
$[\![Dag\ p\ l\ r\ t;\ \forall x{\in}$*set-of* $t.\ l\ x = l'\ x \wedge r\ x = r'\ x]\!]$
$\Longrightarrow Dag\ p\ l'\ r'\ t$
⟨*proof*⟩

lemma *heaps-eq-DagI2*:
$[\![Dag\ p\ l'\ r'\ t;\ \forall x{\in}$*set-of* $t.\ l\ x = l'\ x \wedge r\ x = r'\ x]\!]$
$\Longrightarrow Dag\ p\ l\ r\ t$
⟨*proof*⟩

lemma *Dag-unique-all-impl-simp* [*simp*]:
$Dag\ p\ l\ r\ t \Longrightarrow (\forall t.\ Dag\ p\ l\ r\ t \longrightarrow P\ t) = P\ t$
⟨*proof*⟩

lemma *Dag-unique-ex-conj-simp* [*simp*]:
$Dag\ p\ l\ r\ t \Longrightarrow (\exists t.\ Dag\ p\ l\ r\ t \wedge P\ t) = P\ t$
⟨*proof*⟩

lemma *Dags-eq-hp-eq*:
$\bigwedge p\ p'.\ [\![Dag\ p\ l\ r\ t;\ Dag\ p'\ l'\ r'\ t]\!] \Longrightarrow$
$p'{=}p \wedge (\forall no \in$ *set-of* $t.\ l'\ no = l\ no \wedge r'\ no = r\ no)$
⟨*proof*⟩

definition *isDag* :: *ref* ⇒ (*ref* ⇒ *ref*) ⇒ (*ref* ⇒ *ref*) ⇒ *bool*
where $isDag\ p\ l\ r = (\exists t.\ Dag\ p\ l\ r\ t)$

definition *dag* :: *ref* ⇒ (*ref* ⇒ *ref*) ⇒ (*ref* ⇒ *ref*) ⇒ *dag*
where $dag\ p\ l\ r = (THE\ t.\ Dag\ p\ l\ r\ t)$

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

lemma *Dag-dag*: $Dag\ p\ l\ r\ t \Longrightarrow dag\ p\ l\ r = t$
⟨*proof*⟩

end

3 General Lemmas on BDD Abstractions

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

definition *subdag-eq*:: *dag* ⇒ *dag* ⇒ *bool* **where**
subdag-eq t_1 t_2 = (t_1 = t_2 ∨ *subdag* t_1 t_2)

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

fun *isLeaf* :: *dag* ⇒ *bool* **where**
isLeaf Tip = *False*
| *isLeaf* (*Node Tip v Tip*) = *True*
| *isLeaf* (*Node* (*Node l* 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* ⇒ (*ref* ⇒ *nat*) ⇒ *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 l1 vref1 r1*) *vref* (*Node l2 vref2 r2*)) =
(λ*bdtvar* .
(*if* (*bdtvar vref* = *0* ∨ *bdtvar vref* = *1*)
then None
else
(*case* (*bdt-fn* (*Node l1 vref1 r1*) *bdtvar*) *of*
None ⇒ *None*
|(*Some b1*) ⇒
(*case* (*bdt-fn* (*Node l2 vref2 r2*) *bdtvar*) *of*
None ⇒ *None*
|(*Some b2*) ⇒ *Some* (*Bdt-Node b1* (*bdtvar vref*) *b2*)))))

abbreviation *bdt* == *bdt-fn*

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

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


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


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



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

lemma eval-zero: ⟦bdt (Node l v r) var = Some x;
var (root (Node l v r)) = (0::nat)⟧ ⟹ x = Zero
⟨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*. ⟦ *Dag p low high t*; *bdt t var* = *Some bdt1*; *no* ∈ *set-of t*⟧
⟹ (*low no* = *Null*) = (*high no* = *Null*)
⟨*proof*⟩

primrec *dag-map* :: (*ref* ⇒ *ref*) ⇒ *dag* ⇒ *dag*
where
dag-map f Tip = *Tip* |
dag-map f (*Node l a r*) = (*Node* (*dag-map f l*) (*f a*) (*dag-map f r*))

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

definition *dag-in-levellist*:: *dag* ⇒ (*ref list list*) ⇒ (*ref* ⇒ *nat*) ⇒ *bool*
where *dag-in-levellist t levellist var* = (*t* ≠ *Tip* ⟶
(∀ *st*. *subdag-eq t st* ⟶ *root st* ∈ *set* (*levellist* ! (*var* (*root st*)))))

lemma *set-of-nn*: ⟦ *Dag p low high t*; *n* ∈ *set-of t*⟧ ⟹ *n* ≠ *Null*
⟨*proof*⟩

lemma *subnodes-ordered* [*rule-format*]:
∀ *p*. *n* ∈ *set-of t* ⟶ *Dag p low high t* ⟶ *ordered t var* ⟶ *var n* <= *var p*
⟨*proof*⟩

lemma *ordered-set-of*:
⋀ *x*. ⟦*ordered t var*; *x* ∈ *set-of t*⟧ ⟹ *var x* <= *var* (*root t*)
⟨*proof*⟩

lemma *dag-setofD*: ⋀ *p low high n*. ⟦ *Dag p low high t*; *n* ∈ *set-of t* ⟧ ⟹
(∃ *nt*. *Dag n low high nt*) ∧ (∀ *nt*. *Dag n low high nt* ⟶ *set-of nt* ⊆ *set-of t*)
⟨*proof*⟩

lemma *dag-setof-exD*: ⟦*Dag p low high t*; *n* ∈ *set-of t*⟧ ⟹ ∃ *nt*. *Dag n low high nt*
⟨*proof*⟩

lemma *dag-setof-subsetD*: $[\![Dag\ p\ low\ high\ t;\ n \in set\text{-}of\ t;\ Dag\ n\ low\ high\ nt]\!] \Longrightarrow$
$set\text{-}of\ nt \subseteq set\text{-}of\ t$
⟨*proof*⟩

lemma *subdag-parentdag-low*: $not <= lt \Longrightarrow not <= (Node\ lt\ p\ rt)$ **for** *not*
⟨*proof*⟩

lemma *subdag-parentdag-high*: $not <= rt \Longrightarrow not <= (Node\ lt\ p\ rt)$ **for** *not*
⟨*proof*⟩

lemma *set-of-subdag*: $\bigwedge p\ not\ no.$
$[\![Dag\ p\ low\ high\ t;\ Dag\ no\ low\ high\ not;\ no \in set\text{-}of\ t]\!] \Longrightarrow not <= t$
⟨*proof*⟩

lemma *children-ordered*: $[\![ordered\ (Node\ lt\ p\ rt)\ var]\!] \Longrightarrow$
$ordered\ lt\ var \wedge ordered\ rt\ var$
⟨*proof*⟩

lemma *ordered-subdag*: $[\![ordered\ t\ var;\ not <= t]\!] \Longrightarrow ordered\ not\ var$ **for** *not*
⟨*proof*⟩

lemma *subdag-ordered*:
$\bigwedge not\ no\ p.\ [\![ordered\ t\ var;\ Dag\ p\ low\ high\ t;\ Dag\ no\ low\ high\ not;$
$no \in set\text{-}of\ t]\!] \Longrightarrow ordered\ not\ var$
⟨*proof*⟩

lemma *elem-set-of*: $\bigwedge x\ st.\ [\![x \in set\text{-}of\ st;\ set\text{-}of\ st \subseteq set\text{-}of\ t]\!] \Longrightarrow x \in set\text{-}of\ t$
⟨*proof*⟩

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

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

lemma *cong-eval-sym*: $l \sim r = r \sim l$
⟨*proof*⟩

lemma *cong-eval-trans*: $[\![\ l \sim r;\ r \sim t]\!] \Longrightarrow l \sim t$
⟨*proof*⟩

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

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

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

lemma *null-comp-not-Null* [*simp*]: $h\ q \neq Null \Longrightarrow (g \propto h)\ q = g\ (h\ q)$
⟨*proof*⟩

lemma *id-trans*: $(a \propto id)\ (b\ (c\ p)) = (a \propto b)\ (c\ p)$
⟨*proof*⟩

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

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

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

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

definition *repNodes-eq* :: $ref \Rightarrow ref \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow (ref \Rightarrow ref) \Rightarrow bool$
where
$repNodes\text{-}eq\ p\ q\ low\ high\ rep ==$
$(rep \propto high)\ p = (rep \propto high)\ q \wedge (rep \propto low)\ p = (rep \propto low)\ q$

definition *isomorphic-dags-eq* :: $dag \Rightarrow dag \Rightarrow (ref \Rightarrow nat) \Rightarrow bool$
where
$isomorphic\text{-}dags\text{-}eq\ st_1\ st_2\ var =$
$(\forall\ bdt_1\ bdt_2.\ (bdt\ st_1\ var = Some\ bdt_1 \wedge bdt\ st_2\ var = Some\ bdt_2 \wedge (bdt_1 = bdt_2))$
$\longrightarrow st_1 = st_2)$

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

⟨*proof*⟩

definition *shared* :: *dag* ⇒ (*ref* ⇒ *nat*) ⇒ *bool*
where *shared t var* = (∀ *st*$_1$ *st*$_2$. (*st*$_1$ <= *t* ∧ *st*$_2$ <= *t*) ⟶ *isomorphic-dags-eq st*$_1$ *st*$_2$ *var*)

fun *reduced* :: *dag* ⇒ *bool* **where**
reduced Tip = *True*
| *reduced* (*Node Tip v Tip*) = *True*
| *reduced* (*Node l v r*) = (*l* ≠ *r* ∧ *reduced l* ∧ *reduced r*)

primrec *reduced-bdt* :: *bdt* ⇒ *bool*
where
reduced-bdt Zero = *True*
| *reduced-bdt One* = *True*
| *reduced-bdt* (*Bdt-Node lbdt v rbdt*) =
(*if lbdt* = *rbdt then False*
else (*reduced-bdt lbdt* ∧ *reduced-bdt rbdt*))

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

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*)
⟨*proof*⟩

lemma *nodes-in-wf-ll*:
⟦*wf-ll pret levellista var*; *i* < *length levellista*; *no* ∈ *set* (*levellista* ! *i*)⟧
⟹ *var no* = *i* ∧ *no* ∈ *set-of pret*
⟨*proof*⟩

lemma *subelem-set-of-low*:
⋀ *p*. ⟦ *x* ∈ *set-of t*; *x* ≠ *Null*; *low x* ≠ *Null*; *Dag p low high t* ⟧
⟹ (*low x*) ∈ *set-of t*
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *Nodes-levellist-subset-t*:
$[\![wf\text{-}ll\ t\ levellist\ var;\ i<=\ length\ levellist]\!] \Longrightarrow Nodes\ i\ levellist \subseteq set\text{-}of\ t$
⟨*proof*⟩

lemma *bdt-child*:
$[\![\ bdt\ (Node\ (Node\ llt\ l\ rlt)\ p\ (Node\ lrt\ r\ rrt))\ var = Some\ bdt1]\!]$
$\Longrightarrow \exists\ lbdt\ rbdt.\ \ bdt\ (Node\ llt\ l\ rlt)\ var = Some\ lbdt \wedge$
$bdt\ (Node\ lrt\ r\ rrt)\ var = Some\ rbdt$
⟨*proof*⟩

lemma *subbdt-ex-dag-def*:
$\bigwedge\ bdt1\ p.\ [\![Dag\ p\ low\ high\ t;\ bdt\ t\ var = Some\ bdt1;\ Dag\ no\ low\ high\ not;$
$no \in set\text{-}of\ t]\!] \Longrightarrow \exists\ bdt2.\ \ bdt\ not\ var = Some\ bdt2$ **for** *not*
⟨*proof*⟩

lemma *subbdt-ex*:
$\bigwedge\ bdt1.\ [\![\ (Node\ lst\ stp\ rst) <=\ t;\ bdt\ t\ var = Some\ bdt1]\!]$
$\Longrightarrow \exists\ bdt2.\ \ bdt\ (Node\ lst\ stp\ rst)\ var = Some\ bdt2$
⟨*proof*⟩

lemma *var-ordered-children*:
$\bigwedge\ p.\ [\![\ Dag\ p\ low\ high\ t;\ ordered\ t\ var;\ no \in set\text{-}of\ t;$
$low\ no \neq Null;\ high\ no \neq Null]\!]$
$\Longrightarrow var\ (low\ no) < var\ no \wedge var\ (high\ no) < var\ no$
⟨*proof*⟩

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

lemma *wf-ll-Nodes-pret*:
$[\![wf\text{-}ll\ pret\ levellista\ var;\ nb < length\ levellista;\ x \in Nodes\ nb\ levellista]\!]$
$\Longrightarrow x \in set\text{-}of\ pret \wedge var\ x < nb$

⟨*proof*⟩

lemma *bdt-Some-var1-One*:
$\bigwedge x.\ [\![\ bdt\ t\ var = Some\ x;\ var\ (root\ t) = 1]\!]$
$\Longrightarrow x = One \wedge t = (Node\ Tip\ (root\ t)\ Tip)$
⟨*proof*⟩

lemma *bdt-Some-var0-Zero*:
$\bigwedge x.\ [\![\ bdt\ t\ var = Some\ x;\ var\ (root\ t) = 0]\!]$
$\Longrightarrow x = Zero \wedge t = (Node\ Tip\ (root\ t)\ Tip)$
⟨*proof*⟩

lemma *reduced-children-parent*:
$[\![\ reduced\ l;\ l{=}\ (Node\ llt\ lp\ rlt);\ reduced\ r;\ r{=}(Node\ lrt\ rp\ rrt);$
$lp \neq rp\ ]\!]$
$\Longrightarrow reduced\ (Node\ l\ p\ r)$
⟨*proof*⟩

lemma *Nodes-subset*: $Nodes\ i\ levellista \subseteq Nodes\ (Suc\ i)\ levellista$
⟨*proof*⟩

lemma *Nodes-levellist*:
$[\![\ wf\text{-}ll\ pret\ levellista\ var;\ nb < length\ levellista;\ p \in Nodes\ nb\ levellista]\!]$
$\Longrightarrow p \notin set\ (levellista\ !\ nb)$
⟨*proof*⟩

lemma *Nodes-var-pret*:
$[\![wf\text{-}ll\ pret\ levellista\ var;\ nb < length\ levellista;\ p \in Nodes\ nb\ levellista]\!]$
$\Longrightarrow var\ p < nb \wedge p \in set\text{-}of\ pret$
⟨*proof*⟩

lemma *Dags-root-in-Nodes*:
assumes *t-in-DagsSucnb*: $t \in Dags\ (Nodes\ (Suc\ nb)\ levellista)\ low\ high$
shows $\exists\ p\ .\ Dag\ p\ low\ high\ t\ \wedge p \in Nodes\ (Suc\ nb)\ levellista$
⟨*proof*⟩

lemma *subdag-dag*:
$\bigwedge p.\ [\![Dag\ p\ low\ high\ t;\ st <= t]\!] \Longrightarrow \exists\ stp.\ Dag\ stp\ low\ high\ st$
⟨*proof*⟩

lemma *Dags-subdags*:
assumes *t-in-Dags*: $t \in Dags\ nodes\ low\ high$ **and**
st-t: $st <= t$ **and**
st-nTip: $st \neq Tip$
shows $st \in Dags\ nodes\ low\ high$

⟨*proof*⟩

lemma *Dags-Nodes-cases*:
assumes *P-sym*: $\bigwedge$ *t1 t2*. *P t1 t2 var* = *P t2 t1 var* **and**
dags-in-lower-levels:
$\bigwedge$ *t1 t2*. ⟦*t1* ∈ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*;
t2 ∈ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*⟧
⟹ *P t1 t2 var* **and**
dags-in-mixed-levels:
$\bigwedge$ *t1 t2*. ⟦*t1* ∈ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*;
t2 ∈ *Dags* (*fnct* '(*Nodes* (*Suc n*) *levellista*)) *low high*;
t2 ∉ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*⟧
⟹ *P t1 t2 var* **and**
dags-in-high-level:
$\bigwedge$ *t1 t2*. ⟦*t1* ∈ *Dags* (*fnct* '(*Nodes* (*Suc n*) *levellista*)) *low high*;
t1 ∉ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*;
t2 ∈ *Dags* (*fnct* '(*Nodes* (*Suc n*) *levellista*)) *low high*;
t2 ∉ *Dags* (*fnct* '(*Nodes n levellista*)) *low high*⟧
⟹ *P t1 t2 var*
shows ∀ *t1 t2*. *t1* ∈ *Dags* (*fnct* '(*Nodes* (*Suc n*) *levellista*)) *low high* ∧
t2 ∈ *Dags* (*fnct* '(*Nodes* (*Suc n*) *levellista*)) *low high*
⟶ *P t1 t2 var*
⟨*proof*⟩

lemma *Null-notin-Nodes*: ⟦*Dag p low high t*; *nb* <= *length levellista*; *wf-ll t levellista var*⟧ ⟹ *Null* ∉ *Nodes nb levellista*
⟨*proof*⟩

lemma *Nodes-in-pret*: ⟦*wf-ll t levellista var*; *nb* <= *length levellista*⟧ ⟹ *Nodes nb levellista* ⊆ *set-of t*
⟨*proof*⟩

lemma *restrict-root-Node*:
⟦*t* ∈ *Dags* (*repc* '*Nodes* (*Suc nb*) *levellista*) (*repc* ∝ *low*) (*repc* ∝ *high*); *t* ∉ *Dags* (*repc* '*Nodes nb levellista*) (*repc* ∝ *low*) (*repc* ∝ *high*);
ordered t var; ∀ *no* ∈ *Nodes* (*Suc nb*) *levellista*. *var* (*repc no*) <= *var no* ∧ *repc* (*repc no*) = *repc no*; *wf-ll pret levellista var*; *nb* < *length levellista*;*repc* '*Nodes* (*Suc nb*) *levellista* ⊆ *Nodes* (*Suc nb*) *levellista*⟧
⟹ ∃ *q*. *Dag* (*repc q*) (*repc* ∝ *low*) (*repc* ∝ *high*) *t* ∧ *q* ∈ *set* (*levellista* ! *nb*)
⟨*proof*⟩

lemma *same-bdt-var*: ⟦*bdt* (*Node lt1 p1 rt1*) *var* = *Some bdt1*; *bdt* (*Node lt2 p2 rt2*) *var* = *Some bdt1*⟧
⟹ *var p1* = *var p2*
⟨*proof*⟩

lemma *bdt-Some-Leaf-var-le-1*:
⟦*Dag p low high t*; *bdt t var* = *Some x*; *isLeaf-pt p low high*⟧
⟹ *var p* <= *1*
⟨*proof*⟩

lemma *subnode-dag-cons*:
⋀ *p*. ⟦*Dag p low high t*; *no* ∈ *set-of t*⟧ ⟹ ∃ *not*. *Dag no low high not*
⟨*proof*⟩

lemma *nodes-in-taken-in-takeSucn*: *no* ∈ *set* (*take n nodeslist*) ⟹ *no* ∈ *set* (*take* (*Suc n*) *nodeslist*)
⟨*proof*⟩

lemma *ind-in-higher-take*: ⋀*n k*. ⟦*n* < *k*; *n* < *length xs*⟧
⟹ *xs* ! *n* ∈ *set* (*take k xs*)
⟨*proof*⟩

lemma *take-length-set*: ⋀*n*. *n*=*length xs* ⟹ *set* (*take n xs*) = *set xs*
⟨*proof*⟩

lemma *repNodes-eq-ext-rep*: ⟦*low no* ≠ *nodeslist*! *n*; *high no* ≠ *nodeslist* ! *n*;
low sn ≠ *nodeslist* ! *n*; *high sn* ≠ *nodeslist* ! *n*⟧
⟹ *repNodes-eq sn no low high repa* = *repNodes-eq sn no low high* (*repa*(*nodeslist* ! *n* := *repa* (*low* (*nodeslist* ! *n*))))
⟨*proof*⟩

lemma *filter-not-empty*: ⟦*x* ∈ *set xs*; *P x*⟧ ⟹ *filter P xs* ≠ []
⟨*proof*⟩

lemma *x* ∈ *set* (*filter P xs*) ⟹ *P x*
⟨*proof*⟩

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

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

lemma *hd-filter-prop*:
assumes *non-empty*: *filter P xs* $\neq$ []
shows *P* (*hd* (*filter P xs*))
⟨*proof*⟩

lemma *index-elem*: *x* $\in$ *set xs* $\Longrightarrow$ $\exists$ *i*<*length xs*. *x* = *xs* ! *i*
⟨*proof*⟩

lemma *filter-hd-P-rep-indep*:
⟦$\forall$ *x*. *P x x*; $\forall$ *a b*. *P x a* $\longrightarrow$ *P a b* $\longrightarrow$ *P x b*; *filter* (*P x*) *xs* $\neq$ []⟧ $\Longrightarrow$
hd (*filter* (*P* (*hd* (*filter* (*P x*) *xs*))) *xs*) = *hd* (*filter* (*P x*) *xs*)
⟨*proof*⟩

lemma *take-Suc-not-last*:
$\bigwedge$*n*. ⟦*x* $\in$ *set* (*take* (*Suc n*) *xs*); *x*$\neq$*xs*!*n*; *n* < *length xs*⟧ $\Longrightarrow$ *x* $\in$ *set* (*take n xs*)
⟨*proof*⟩

lemma *P-eq-list-filter*: $\forall$ *x* $\in$ *set xs*. *P x* = *Q x* $\Longrightarrow$ *filter P xs* = *filter Q xs*
⟨*proof*⟩

lemma *hd-filter-take-more*: $\bigwedge$*n m*.⟦*filter P* (*take n xs*) $\neq$ []; *n* $\leq$ *m*⟧ $\Longrightarrow$
hd (*filter P* (*take n xs*)) = *hd* (*filter P* (*take m xs*))
⟨*proof*⟩

end

4 Definitions of Procedures

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

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

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

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

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

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

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

procedures

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

procedures

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

procedures

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

end

5 Proof of Procedure Eval

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

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

lemma (**in** *Eval-impl*) *Eval-spec*:
  shows $\forall\sigma\ t$ *bdt1*. $\Gamma\vdash$
  $\{\!|\sigma.$ *Dag* ´*p* ´*low* ´*high* $t \wedge$ *bdt t* ´*var* = *Some bdt1*$|\!\}$
   ´*R* :== *PROC Eval*(´*p*, ´*varval*)
  $\{\!|$´*R* = *eval bdt1* $^\sigma$*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 $\forall\sigma.\ \Gamma\vdash\{\sigma\}$ ´*levellist* :== *PROC Levellist* (´*p*, ´*m*, ´*levellist*)
          $\{t.\ t$ *may-only-modify-globals* σ *in* [*mark*,*next*]$\}$
  ⟨*proof*⟩

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

lemma *Dag-RefD*:
  $[\![$*Dag p l r t*; *p*≠*Null*$]\!] \Longrightarrow$
   $\exists$ *lt rt*. *t*=*Node lt p rt* $\wedge$ *Dag* (*l p*) *l r lt* $\wedge$ *Dag* (*r p*) *l r rt*
  ⟨*proof*⟩

lemma *Dag-unique-ex-conjI*:
  $[\![$*Dag p l r t*; *P t*$]\!] \Longrightarrow (\exists t.$ *Dag p l r t* $\wedge$ *P t*$)$
  ⟨*proof*⟩

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

⟨*proof*⟩

definition *first*:: *ref list* ⇒ *ref* **where**
first ps = (*case ps of* [] ⇒ *Null* | (*p*#*rs*) ⇒ *p*)

lemma *first-simps* [*simp*]:
first [] = *Null*
first (*r*#*rs*) = *r*
⟨*proof*⟩

definition *Levellist*:: *ref list* ⇒ (*ref* ⇒ *ref*) ⇒ (*ref list list*) ⇒ *bool* **where**
Levellist hds next ll ⟷ (*map first ll* = *hds*) ∧
(∀ *i* < *length hds*. *List* (*hds* ! *i*) *next* (*ll*!*i*))

lemma *Levellist-unique*:
assumes *ll*: *Levellist hds next ll*
assumes *ll'*: *Levellist hds next ll'*
shows *ll*=*ll'*
⟨*proof*⟩

lemma *Levellist-unique-ex-conj-simp* [*simp*]:
Levellist hds next ll ⟹ (∃ *ll*. *Levellist hds next ll* ∧ *P ll*) = *P ll*
⟨*proof*⟩

lemma *in-set-concat-idx*:
x ∈ *set* (*concat xss*) ⟹ ∃ *i* < *length xss*. *x* ∈ *set* (*xss*!*i*)
⟨*proof*⟩

definition *wf-levellist* :: *dag* ⇒ *ref list list* ⇒ *ref list list* ⇒
(*ref* ⇒ *nat*) ⇒ *bool* **where**
wf-levellist t levellist-old levellist-new var =
(*case t of Tip* ⇒ *levellist-old* = *levellist-new*
| (*Node lt p rt*) ⇒
(∀ *q*. *q* ∈ *set-of t* ⟶ *q* ∈ *set* (*levellist-new* ! (*var q*))) ∧
(∀ *i* ≤ *var p*. (∃ *prx*. (*levellist-new* ! *i*) = *prx*@(*levellist-old* ! *i*)
∧ (∀ *pt* ∈ *set prx*. *pt* ∈ *set-of t* ∧ *var pt* = *i*))) ∧
(∀ *i*. (*var p*) < *i* ⟶ (*levellist-new* ! *i*) = (*levellist-old* ! *i*)) ∧
(*length levellist-new* = *length levellist-old*))

lemma *wf-levellist-subset*:
assumes *wf-ll*: *wf-levellist t ll ll' var*
shows *set* (*concat ll'*) ⊆ *set* (*concat ll*) ∪ *set-of t*
⟨*proof*⟩

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

lemma *Levellist-length*: *Levellist hds p ll* $\Longrightarrow$ *length ll* $=$ *length hds*
⟨*proof*⟩

lemma *map-update*:
$\bigwedge i.\ i < length\ xss \Longrightarrow map\ f\ (xss[i := xs]) = (map\ f\ xss)\ [i := f\ xs]$
⟨*proof*⟩

lemma (**in** *Levellist-impl*) *Levellist-spec-total′*:
shows $\forall\, ll\ \sigma\ t.\ \Gamma,\Theta\vdash_t$
$\{\!|\sigma.\ Dag\ ´p\ ´low\ ´high\ t \wedge (´p \neq Null \longrightarrow (´p \rightarrow ´var) < length\ ´levellist) \wedge$
$ordered\ t\ ´var \wedge Levellist\ ´levellist\ ´next\ ll \wedge$
$(\forall\, n \in set\text{-}of\ t.$
$(if\ ´mark\ n = ´m$
$then\ n \in set\ (ll\ !\ ´var\ n) \wedge$
$(\forall\, nt\ p.\ Dag\ n\ ´low\ ´high\ nt \wedge p \in set\text{-}of\ nt$
$\longrightarrow ´mark\ p = ´m)$
$else\ n \notin set\ (concat\ ll)))|\!\}$
$´levellist :== PROC\ Levellist\ (´p,\ ´m,\ ´levellist)$
$\{\!|\exists\, ll'.\ Levellist\ ´levellist\ ´next\ ll' \wedge wf\text{-}levellist\ t\ ll\ ll'\ ^\sigma var \wedge$
$wf\text{-}marking\ t\ ^\sigma mark\ ´mark\ ^\sigma m \wedge$
$(\forall\, p.\ p \notin set\text{-}of\ t \longrightarrow {}^\sigma next\ p = ´next\ p)$
$|\!\}$
⟨*proof*⟩

lemma *allD*: $\forall\, ll.\ P\ ll \Longrightarrow P\ ll$
⟨*proof*⟩

lemma *replicate-spec*: $[\![\forall\, i < n.\ xs\ !\ i = x;\ n{=}length\ xs]\!]$
$\Longrightarrow replicate\ (length\ xs)\ x = xs$
⟨*proof*⟩

lemma (**in** *Levellist-impl*) *Levellist-spec-total*:
shows $\forall\, \sigma\ t.\ \Gamma,\Theta\vdash_t$
$\{\!|\sigma.\ Dag\ ´p\ ´low\ ´high\ t \wedge (\forall\, i < length\ ´levellist.\ ´levellist\ !\ i = Null) \wedge$
$length\ ´levellist\ = ´p \rightarrow ´var + 1 \wedge$
$ordered\ t\ ´var \wedge (\forall\, n \in set\text{-}of\ t.\ ´mark\ n = (\neg\ ´m)\ )|\!\}$
$´levellist :== PROC\ Levellist\ (´p,\ ´m,\ ´levellist)$
$\{\!|\exists\, ll.\ Levellist\ ´levellist\ ´next\ ll \wedge wf\text{-}ll\ t\ ll\ ^\sigma var\ \wedge$
$length\ ´levellist = {}^\sigma p \rightarrow {}^\sigma var + 1 \wedge$
$wf\text{-}marking\ t\ ^\sigma mark\ ´mark\ ^\sigma m \wedge$
$(\forall\, p.\ p \notin set\text{-}of\ t \longrightarrow {}^\sigma next\ p = ´next\ p)|\!\}$
⟨*proof*⟩

end

7 Proof of Procedure ShareRep

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

lemma (**in** *ShareRep-impl*) *ShareRep-modifies*:
shows $\forall \sigma.\ \Gamma\vdash\{\sigma\}$ *PROC ShareRep* (´*nodeslist*, ´*p*)
$\{t.\ t$ *may-only-modify-globals* σ *in* [*rep*]$\}$
⟨*proof*⟩

lemma *hd-filter-cons*:
$\bigwedge i.\ [\![\ P\ (xs\ !\ i)\ p;\ i < length\ xs;\ \forall\ no \in set\ (take\ i\ xs).\ \neg\ P\ no\ p;\ \forall\ a\ b.\ P\ a\ b = P\ b\ a]\!]$
$\Longrightarrow xs\ !\ i = hd\ (filter\ (P\ p)\ xs)$
⟨*proof*⟩

lemma (**in** *ShareRep-impl*) *ShareRep-spec-total*:
shows
$\forall \sigma\ ns.\ \Gamma,\Theta\vdash_t$
$\{\!|\sigma.\ List\ ´nodeslist\ ´next\ ns \wedge$
$(\forall no \in set\ ns.\ no \neq Null\ \wedge$
$((no\rightarrow´low = Null) = (no\rightarrow´high = Null)) \wedge$
$(isLeaf\text{-}pt\ ´p\ ´low\ ´high \longrightarrow isLeaf\text{-}pt\ no\ ´low\ ´high) \wedge$
$no\rightarrow´var = ´p\rightarrow´var) \wedge$
$´p \in set\ ns|\!\}$
PROC ShareRep (´*nodeslist*, ´*p*)
$\{\!|\ (^{\sigma}p \rightarrow ´rep = hd\ (filter\ (\lambda\ sn.\ repNodes\text{-}eq\ sn\ ^{\sigma}p\ ^{\sigma}low\ ^{\sigma}high\ ^{\sigma}rep)\ ns)) \wedge$
$(\forall pt.\ pt \neq {}^{\sigma}p \longrightarrow pt\rightarrow^{\sigma}rep = pt\rightarrow´rep) \wedge$
$(^{\sigma}p\rightarrow´rep\rightarrow^{\sigma}var = {}^{\sigma}p \rightarrow {}^{\sigma}var)|\!\}$
⟨*proof*⟩

end

8 Proof of Procedure ShareReduceRepList

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

lemma (**in** *ShareReduceRepList-impl*) *ShareReduceRepList-modifies*:
shows $\forall \sigma.\ \Gamma\vdash\{\sigma\}$ *PROC ShareReduceRepList* (´*nodeslist*)
$\{t.\ t$ *may-only-modify-globals* σ *in* [*rep*]$\}$
⟨*proof*⟩

lemma *hd-filter-app*: $[\![filter\ P\ xs \neq []; zs{=}xs@ys]\!] \Longrightarrow$
$hd\ (filter\ P\ zs) = hd\ (filter\ P\ xs)$
⟨*proof*⟩

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

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

lemma *low-high-exchange-dag*:
assumes *pt-same*: $\forall\, pt.\ pt \notin set\text{-}of\ lt \longrightarrow low\ pt = lowa\ pt \wedge high\ pt = higha\ pt$
assumes *pt-changed*: $\forall\, pt \in set\text{-}of\ lt.\ lowa\ pt = (rep \propto low)\ pt \wedge$
$higha\ pt = (rep \propto high)\ pt$
assumes *rep-pt*: $\forall\, pt \in set\text{-}of\ rt.\ rep\ pt = pt$
shows $\bigwedge q.\ \ Dag\ q\ (rep \propto low)\ (rep \propto high)\ rt \Longrightarrow$
$Dag\ q\ (rep \propto lowa)\ (rep \propto higha)\ rt$
⟨*proof*⟩

lemma (**in** *Repoint-impl*) *Repoint-spec'*:
shows
$\forall \sigma.\ \Gamma\vdash \{\sigma\}$
$'p :== PROC\ Repoint\ ('p)$
$\{\!|\forall\ rept.\ ((Dag\ ((^{\sigma}rep \propto id)\ ^{\sigma}p)\ (^{\sigma}rep \propto {}^{\sigma}low)\ (^{\sigma}rep \propto {}^{\sigma}high)\ rept)$
$\wedge\ (\forall\ no \in set\text{-}of\ rept.\ ^{\sigma}rep\ no = no))$
$\longrightarrow Dag\ 'p\ 'low\ 'high\ rept \wedge$
$(\forall pt.\ pt \notin set\text{-}of\ rept \longrightarrow {}^{\sigma}low\ pt = 'low\ pt \wedge {}^{\sigma}high\ pt = 'high\ pt)|\!\}$
⟨*proof*⟩

lemma (**in** *Repoint-impl*) *Repoint-spec*:
shows
$\forall \sigma\ rept.\ \Gamma\vdash \{\!|\sigma.\ Dag\ (('rep \propto id)\ 'p)\ ('rep \propto 'low)\ ('rep \propto 'high)\ rept$
$\wedge\ (\forall\ no \in set\text{-}of\ rept.\ 'rep\ no = no)\ |\!\}$
$'p :== PROC\ Repoint\ ('p)$
$\{\!|Dag\ 'p\ 'low\ 'high\ rept \wedge$
$(\forall pt.\ pt \notin set\text{-}of\ rept \longrightarrow {}^{\sigma}low\ pt = 'low\ pt \wedge {}^{\sigma}high\ pt = 'high\ pt)|\!\}$
⟨*proof*⟩

lemma (**in** *Repoint-impl*) *Repoint-spec-total*:
shows
$\forall \sigma\ rept.\ \Gamma\vdash_t \{\!|\sigma.\ Dag\ (('rep \propto id)\ 'p)\ ('rep \propto 'low)\ ('rep \propto 'high)\ rept$
$\wedge\ (\forall\ no \in set\text{-}of\ rept.\ 'rep\ no = no)\ |\!\}$
$'p :== PROC\ Repoint\ ('p)$
$\{\!|Dag\ 'p\ 'low\ 'high\ rept \wedge$
$(\forall pt.\ pt \notin set\text{-}of\ rept \longrightarrow {}^{\sigma}low\ pt = 'low\ pt \wedge {}^{\sigma}high\ pt = 'high\ pt)|\!\}$

⟨*proof*⟩

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

lemma (**in** *Normalize-impl*) *Normalize-spec*:
shows $\forall \sigma\ pret\ prebdt.\ \Gamma\vdash_t$
$\{\!|\sigma.\ Dag\ 'p\ 'low\ 'high\ pret \wedge ordered\ pret\ 'var \wedge$

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

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.