Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

theory *IsaSAT-Literals*
imports *Watched-Literals.WB-More-Refinement HOL−Word.More-Word*
Watched-Literals.Watched-Literals-Watch-List-Domain
Entailment-Definition.Partial-Herbrand-Interpretation
Watched-Literals.Bits-Natural Watched-Literals.WB-Word
begin

hide-const *Autoref-Fix-Rel.CONSTRAINT*

Refinement of the Watched Function

definition *map-fun-rel* :: ‹$(nat \times 'key)\ set \Rightarrow ('b \times 'a)\ set \Rightarrow ('b\ list \times ('key \Rightarrow 'a))\ set$› **where**
map-fun-rel-def-internal:
‹$map\text{-}fun\text{-}rel\ D\ R = \{(m, f).\ \forall (i, j){\in}D.\ i < length\ m \wedge (m\ !\ i, f\ j) \in R\}$›

lemma *map-fun-rel-def*:
‹$\langle R\rangle map\text{-}fun\text{-}rel\ D = \{(m, f).\ \forall (i, j){\in}D.\ i < length\ m \wedge (m\ !\ i, f\ j) \in R\}$›
⟨*proof*⟩

0.0.1 Literals as Natural Numbers

Definition

lemma *Pos-div2-iff*:
‹$Pos\ ((bb :: nat)\ div\ 2) = b \longleftrightarrow is\text{-}pos\ b \wedge (bb = 2 * atm\text{-}of\ b \vee bb = 2 * atm\text{-}of\ b + 1)$›
⟨*proof*⟩
lemma *Neg-div2-iff*:
‹$Neg\ ((bb :: nat)\ div\ 2) = b \longleftrightarrow is\text{-}neg\ b \wedge (bb = 2 * atm\text{-}of\ b \vee bb = 2 * atm\text{-}of\ b + 1)$›
⟨*proof*⟩

Modeling *nat literal* via the transformation associating $(2::'a) * n$ or $(2::'a) * n + (1::'a)$ has some advantages over the transformation to positive or negative integers: 0 is not an issue. It is also a bit faster according to Armin Biere.

fun *nat-of-lit* :: ‹$nat\ literal \Rightarrow nat$› **where**
‹$nat\text{-}of\text{-}lit\ (Pos\ L) = 2{*}L$›
| ‹$nat\text{-}of\text{-}lit\ (Neg\ L) = 2{*}L + 1$›

lemma *nat-of-lit-def*: ‹$nat\text{-}of\text{-}lit\ L = (if\ is\text{-}pos\ L\ then\ 2 * atm\text{-}of\ L\ else\ 2 * atm\text{-}of\ L + 1)$›
⟨*proof*⟩

fun *literal-of-nat* :: ‹$nat \Rightarrow nat\ literal$› **where**
‹$literal\text{-}of\text{-}nat\ n = (if\ even\ n\ then\ Pos\ (n\ div\ 2)\ else\ Neg\ (n\ div\ 2))$›

lemma *lit-of-nat-nat-of-lit*[*simp*]: ‹$literal\text{-}of\text{-}nat\ (nat\text{-}of\text{-}lit\ L) = L$›
⟨*proof*⟩

lemma *nat-of-lit-lit-of-nat*[*simp*]: ‹$nat\text{-}of\text{-}lit\ (literal\text{-}of\text{-}nat\ n) = n$›
⟨*proof*⟩

lemma *atm-of-lit-of-nat*: ‹$atm\text{-}of\ (literal\text{-}of\text{-}nat\ n) = n\ div\ 2$›
⟨*proof*⟩

There is probably a more "closed" form from the following theorem, but it is unclear if that is useful or not.

lemma *uminus-lit-of-nat*:
‹$-\ (literal\text{-}of\text{-}nat\ n) = (if\ even\ n\ then\ literal\text{-}of\text{-}nat\ (n{+}1)\ else\ literal\text{-}of\text{-}nat\ (n{-}1))$›
⟨*proof*⟩

lemma *literal-of-nat-literal-of-nat-eq*[*iff*]: ‹$literal\text{-}of\text{-}nat\ x = literal\text{-}of\text{-}nat\ xa \longleftrightarrow x = xa$›
⟨*proof*⟩

definition *nat-lit-rel* :: ‹$(nat \times nat\ literal)\ set$› **where**
‹$nat\text{-}lit\text{-}rel = \ br\ literal\text{-}of\text{-}nat\ (\lambda\text{-}.\ True)$›

definition *unat-lit-rel* :: ‹$(uint32 \times nat\ literal)\ set$› **where**

‹$unat\text{-}lit\text{-}rel \equiv uint32\text{-}nat\text{-}rel\ O\ nat\text{-}lit\text{-}rel$›

fun *pair-of-ann-lit* :: ‹$('a, 'b)\ ann\text{-}lit \Rightarrow 'a\ literal \times 'b\ option$› **where**
‹$pair\text{-}of\text{-}ann\text{-}lit\ (Propagated\ L\ D) = (L, Some\ D)$›
| ‹$pair\text{-}of\text{-}ann\text{-}lit\ (Decided\ L) = (L, None)$›

fun *ann-lit-of-pair* :: ‹$'a\ literal \times 'b\ option \Rightarrow ('a, 'b)\ ann\text{-}lit$› **where**
‹$ann\text{-}lit\text{-}of\text{-}pair\ (L, Some\ D) = Propagated\ L\ D$›
| ‹$ann\text{-}lit\text{-}of\text{-}pair\ (L, None) = Decided\ L$›

lemma *ann-lit-of-pair-alt-def*:
‹$ann\text{-}lit\text{-}of\text{-}pair\ (L, D) = (if\ D = None\ then\ Decided\ L\ else\ Propagated\ L\ (the\ D))$›
⟨*proof*⟩

lemma *ann-lit-of-pair-pair-of-ann-lit*: ‹$ann\text{-}lit\text{-}of\text{-}pair\ (pair\text{-}of\text{-}ann\text{-}lit\ L) = L$›
⟨*proof*⟩

lemma *pair-of-ann-lit-ann-lit-of-pair*: ‹$pair\text{-}of\text{-}ann\text{-}lit\ (ann\text{-}lit\text{-}of\text{-}pair\ L) = L$›
⟨*proof*⟩

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ‹$literal\text{-}of\text{-}nat\ b = L \longleftrightarrow b = nat\text{-}of\text{-}lit\ L$›
⟨*proof*⟩

lemma *nat-of-lit-eq-iff*[*iff*]: ‹$nat\text{-}of\text{-}lit\ xa = nat\text{-}of\text{-}lit\ x \longleftrightarrow x = xa$›
⟨*proof*⟩

definition *ann-lit-rel*:: ‹$('a \times nat)\ set \Rightarrow ('b \times nat\ option)\ set \Rightarrow$
$(('a \times 'b) \times (nat, nat)\ ann\text{-}lit)\ set$› **where**
ann-lit-rel-internal-def:
‹$ann\text{-}lit\text{-}rel\ R\ R' = \{(a, b).\ \exists\, c\ d.\ (fst\ a, c) \in R \wedge (snd\ a, d) \in R' \wedge$
$b = ann\text{-}lit\text{-}of\text{-}pair\ (literal\text{-}of\text{-}nat\ c, d)\}$›

type-synonym *ann-lit-wl* = ‹$uint32 \times nat\ option$›
type-synonym *ann-lits-wl* = ‹$ann\text{-}lit\text{-}wl\ list$›
type-synonym *ann-lit-wl-fast* = ‹$uint32 \times uint64\ option$›
type-synonym *ann-lits-wl-fast* = ‹$ann\text{-}lit\text{-}wl\text{-}fast\ list$›

definition *nat-ann-lit-rel* :: ‹$(ann\text{-}lit\text{-}wl \times (nat, nat)\ ann\text{-}lit)\ set$› **where**
nat-ann-lit-rel-internal-def: ‹$nat\text{-}ann\text{-}lit\text{-}rel = \langle uint32\text{-}nat\text{-}rel, \langle nat\text{-}rel\rangle option\text{-}rel\rangle ann\text{-}lit\text{-}rel$›

lemma *ann-lit-rel-def*:
‹$\langle R, R'\rangle ann\text{-}lit\text{-}rel = \{(a, b).\ \exists\, c\ d.\ (fst\ a, c) \in R \wedge (snd\ a, d) \in R' \wedge$
$b = ann\text{-}lit\text{-}of\text{-}pair\ (literal\text{-}of\text{-}nat\ c, d)\}$›
⟨*proof*⟩

lemma *nat-ann-lit-rel-def*:
‹$nat\text{-}ann\text{-}lit\text{-}rel = \{(a, b).\ b = ann\text{-}lit\text{-}of\text{-}pair\ ((\lambda(a,b).\ (literal\text{-}of\text{-}nat\ (nat\text{-}of\text{-}uint32\ a), b))\ a)\}$›
⟨*proof*⟩

definition *nat-ann-lits-rel* :: ‹$(ann\text{-}lits\text{-}wl \times (nat, nat)\ ann\text{-}lits)\ set$› **where**
‹$nat\text{-}ann\text{-}lits\text{-}rel = \langle nat\text{-}ann\text{-}lit\text{-}rel\rangle list\text{-}rel$›

lemma *nat-ann-lits-rel-Cons*[*iff*]:
‹$(x\ \#\ xs, y\ \#\ ys) \in nat\text{-}ann\text{-}lits\text{-}rel \longleftrightarrow (x, y) \in nat\text{-}ann\text{-}lit\text{-}rel \wedge (xs, ys) \in nat\text{-}ann\text{-}lits\text{-}rel$›

$\langle proof \rangle$

definition (**in** $-$)*the-is-empty* **where**
‹*the-is-empty* D = *Multiset.is-empty* (*the* D)›

0.0.2 Atoms with bound

abbreviation *uint-max* :: *nat* **where**
‹*uint-max* ≡ *uint32-max*›

lemmas *uint-max-def* = *uint32-max-def*

context
fixes $\mathcal{A}_{in}$ :: ‹*nat multiset*›
begin

abbreviation D_0 :: ‹(*nat* × *nat literal*) *set*› **where**
‹D_0 ≡ (λL. (*nat-of-lit* L, L)) ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›

definition *length-ll-f* **where**
‹*length-ll-f* W L = *length* (W L)›

lemma *length-ll-length-ll-f*:
‹(*uncurry* (*RETURN oo length-ll*), *uncurry* (*RETURN oo length-ll-f*)) ∈
[λ(W, L). L ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$]$_f$ (($\langle Id \rangle$*map-fun-rel* D_0) $\times_r$ *nat-lit-rel*) →
$\langle$*nat-rel*$\rangle$ *nres-rel*›
$\langle proof \rangle$

lemma *ex-list-watched*:
fixes W :: ‹*nat literal* ⇒ *'a list*›
shows ‹∃ *aa*. ∀ x∈#$\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit* x < *length aa* ∧ *aa* ! *nat-of-lit* x = W x›
(**is** ‹∃ *aa*. *?P aa*›)
$\langle proof \rangle$

definition *isasat-input-bounded* **where**
[*simp*]: ‹*isasat-input-bounded* = (∀ L ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit* L ≤ *uint-max*)›

definition *isasat-input-nempty* **where**
[*simp*]: ‹*isasat-input-nempty* = (*set-mset* $\mathcal{A}_{in}$ ≠ {})›

definition *isasat-input-bounded-nempty* **where**
‹*isasat-input-bounded-nempty* = (*isasat-input-bounded* ∧ *isasat-input-nempty*)›

context
assumes *in-*$\mathcal{L}_{all}$*-less-uint-max*: ‹*isasat-input-bounded*›
begin

lemma *in-*$\mathcal{L}_{all}$*-less-uint-max'*: ‹L ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$ ⟹ *nat-of-lit* L ≤ *uint-max*›
$\langle proof \rangle$

lemma *in-*$\mathcal{A}_{in}$*-less-than-uint-max-div-2*:
‹L ∈# $\mathcal{A}_{in}$ ⟹ L ≤ *uint-max div 2*›
$\langle proof \rangle$

lemma *simple-clss-size-upper-div2'*:
assumes

lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*› **and**
in-$\mathcal{L}_{all}$*-less-uint-max*: ‹$\forall$ *L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint-max* − *1*›
shows ‹*size C* ≤ *uint-max div 2*›
⟨*proof*⟩

lemma *simple-clss-size-upper-div2*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*›
shows ‹*size C* ≤ *1* + *uint-max div 2*›
⟨*proof*⟩

lemma *clss-size-uint-max*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*›
shows ‹*size C* ≤ *uint-max* + *2*›
⟨*proof*⟩

lemma *clss-size-uint64-max*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*›
shows ‹*size C* < *uint64-max*›
⟨*proof*⟩

lemma *clss-size-upper*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
in-$\mathcal{L}_{all}$*-less-uint-max*: ‹$\forall$ *L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint-max* − *1*›
shows ‹*size C* ≤ *uint-max*›
⟨*proof*⟩

lemma
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}_{in}$ *M*› **and**
n-d: ‹*no-dup M*›
shows
literals-are-in-$\mathcal{L}_{in}$*-trail-length-le-uint32-max*:
‹*length M* ≤ *Suc* (*uint-max div 2*)› **and**
literals-are-in-$\mathcal{L}_{in}$*-trail-count-decided-uint-max*:
‹*count-decided M* ≤ *Suc* (*uint-max div 2*)› **and**
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*:
‹*get-level M L* ≤ *Suc* (*uint-max div 2*)›
⟨*proof*⟩

lemma *length-trail-uint-max-div2*:
fixes *M* :: ‹(*nat*, *'b*) *ann-lits*›
assumes
M-$\mathcal{L}_{all}$: ‹$\forall$ *L*∈*set M*. *lit-of L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}_{in}$› **and**
n-d: ‹*no-dup M*›

shows ‹*length M* ≤ *uint-max div 2* + *1*›
⟨*proof*⟩

end

end

First we instantiate our types with sort heap and default, to have compatibility with code generation. The idea is simplify to create injections into the components of our datatypes.

instance *literal* :: (*heap*) *heap*
⟨*proof*⟩

instance *annotated-lit* :: (*heap*, *heap*, *heap*) *heap*
⟨*proof*⟩

instantiation *literal* :: (*default*) *default*
begin

definition *default-literal* **where**
‹*default-literal* = *Pos default*›
instance ⟨*proof*⟩

end

instantiation *fmap* :: (*type*, *type*) *default*
begin

definition *default-fmap* **where**
‹*default-fmap* = *fmempty*›
instance ⟨*proof*⟩

end

0.1 Code Generation

0.1.1 Literals as Natural Numbers

definition *propagated* **where**
‹*propagated L C* = (*L*, *Some C*)›

definition *decided* **where**
‹*decided L* = (*L*, *None*)›

definition *uminus-lit-imp* :: ‹*nat* ⇒ *nat*› **where**
‹*uminus-lit-imp L* = *bitXOR L 1*›

lemma *uminus-lit-imp-uminus*:
‹(*RETURN o uminus-lit-imp*, *RETURN o uminus*) ∈
nat-lit-rel $\rightarrow_f$ ⟨*nat-lit-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *uminus-code* :: ‹*uint32* ⇒ *uint32*› **where**
‹*uminus-code L* = *bitXOR L 1*›

0.1.2 State Conversion

Functions and Types:

type-synonym *nat-clauses-l* = ‹*nat list list*›

Refinement of the Watched Function

definition *watched-by-nth* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-nth* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) *L i*. *W L* ! *i*)›

definition *watched-app*
:: ‹(*nat literal* ⇒ (*nat watcher*) *list*) ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-app M L i* ≡ *M L* ! *i*›

lemma *watched-by-nth-watched-app*:
‹*watched-by S K* ! *w* = *watched-app* ((*snd o snd o snd o snd o snd o snd*) *S*) *K w*›
⟨*proof*⟩

More Operations

lemma *nat-of-uint32-shiftr*: ‹*nat-of-uint32* (*shiftr xi n*) = *shiftr* (*nat-of-uint32 xi*) *n*›
⟨*proof*⟩

definition *atm-of-code* :: ‹*uint32* ⇒ *uint32*› **where**
‹*atm-of-code L* = *shiftr L 1*›

0.1.3 Code Generation

More Operations

definition *literals-to-update-wl-empty* :: ‹*nat twl-st-wl* ⇒ *bool*› **where**
‹*literals-to-update-wl-empty* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *Q* = {#})›

lemma *in-nat-list-rel-list-all2-in-set-iff*:
‹(*a*, *aa*) ∈ *nat-lit-rel* ⟹
list-all2 (λ*x x*′. (*x*, *x*′) ∈ *nat-lit-rel*) *b ba* ⟹
a ∈ *set b* ⟷ *aa* ∈ *set ba*›
⟨*proof*⟩

definition *is-decided-wl* **where**
‹*is-decided-wl L* ⟷ *snd L* = *None*›

lemma *is-decided-wl-is-decided*:
‹(*RETURN o is-decided-wl*, *RETURN o is-decided*) ∈ *nat-ann-lit-rel* → ⟨*bool-rel*⟩ *nres-rel*›
⟨*proof*⟩

lemma *ann-lit-of-pair-if*:
‹*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))›
⟨*proof*⟩

definition *get-maximum-level-remove* **where**
‹*get-maximum-level-remove M D L* = *get-maximum-level M* (*remove1-mset L D*)›

lemma *in-list-all2-ex-in*: ‹*a* ∈ *set xs* ⟹ *list-all2 R xs ys* ⟹ ∃ *b* ∈ *set ys*. *R a b*›
⟨*proof*⟩

definition *find-decomp-wl-imp* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat literal* ⇒ (*nat*, *nat*) *ann-lits nres*› **where**
‹*find-decomp-wl-imp* = (λM_0 *D L*. *do* {
let lev = *get-maximum-level* M_0 (*remove1-mset* (−*L*) *D*);
let k = *count-decided* M_0;
(-, *M*) ←
$WHILE_T{}^{\lambda(j,\ M).\ j\ =\ count\text{-}decided\ M\ \wedge\ j\ \geq\ lev\ \wedge}$ $(M = [] \longrightarrow j = lev) \wedge$ $(\exists M'.\ M_0 = M'\ @\ M \wedge (j =$
(λ(*j*, *M*). *j* > *lev*)
(λ(*j*, *M*). *do* {
ASSERT(*M* ≠ []);
if is-decided (*hd M*)
then RETURN (*j*−*1*, *tl M*)
else RETURN (*j*, *tl M*)}
)
(*k*, M_0);
RETURN M
})›

lemma *ex-decomp-get-ann-decomposition-iff*:
‹(∃ *M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)) ⟷
(∃ *M2*. *M* = *M2* @ *Decided K* # *M1*)›
⟨*proof*⟩

lemma *count-decided-tl-if*:
‹*M* ≠ [] ⟹ *count-decided* (*tl M*) = (*if is-decided* (*hd M*) *then count-decided M* − *1 else count-decided M*)›
⟨*proof*⟩

lemma *count-decided-butlast*:
‹*count-decided* (*butlast xs*) = (*if is-decided* (*last xs*) *then count-decided xs* − *1 else count-decided xs*)›
⟨*proof*⟩

definition *find-decomp-wl′* **where**
‹*find-decomp-wl′* =
(λ(*M*::(*nat*, *nat*) *ann-lits*) (*D*::*nat clause*) (*L*::*nat literal*).
SPEC(λ*M1*. ∃ *K M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
get-level M K = *get-maximum-level M* (*D* − {#−*L*#}) + *1*))›

definition *get-conflict-wl-is-None* :: ‹*nat twl-st-wl* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *is-None D*)›

lemma *get-conflict-wl-is-None*: ‹*get-conflict-wl S* = *None* ⟷ *get-conflict-wl-is-None S*›
⟨*proof*⟩

lemma *watched-by-nth-watched-app′*:
‹*watched-by S K* = ((*snd o snd o snd o snd o snd o snd*) *S*) *K*›
⟨*proof*⟩

lemma (**in** −) *hd-decided-count-decided-ge-1*:
‹*x* ≠ [] ⟹ *is-decided* (*hd x*) ⟹ *Suc 0* ≤ *count-decided x*›
⟨*proof*⟩

definition (**in** −) *find-decomp-wl-imp′* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clause-l list* ⇒ *nat* ⇒

$nat\ clause \Rightarrow nat\ clauses \Rightarrow nat\ clauses \Rightarrow nat\ lit\text{-}queue\text{-}wl \Rightarrow$
$(nat\ literal \Rightarrow nat\ watched) \Rightarrow \text{-} \Rightarrow (nat, nat)\ ann\text{-}lits\ nres$› **where**
‹$find\text{-}decomp\text{-}wl\text{-}imp' = (\lambda M\ N\ U\ D\ NE\ UE\ W\ Q\ L.\ find\text{-}decomp\text{-}wl\text{-}imp\ M\ D\ L)$›

lemma *nth-ll-watched-app*:
‹$(uncurry2\ (RETURN\ ooo\ nth\text{-}rll), uncurry2\ (RETURN\ ooo\ watched\text{-}app)) \in$
$[\lambda((W, L), i).\ L \in\#\ (\mathcal{L}_{all}\ \mathcal{A})]_f\ ((\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A})) \times_r nat\text{-}lit\text{-}rel) \times_r nat\text{-}rel \rightarrow$
$\langle nat\text{-}rel \times_r Id\rangle\ nres\text{-}rel$›
⟨*proof*⟩

lemma *ex-literal-of-nat*: ‹$\exists bb.\ b = literal\text{-}of\text{-}nat\ bb$›
⟨*proof*⟩

definition (**in** −) *is-pos-code* :: ‹$uint32 \Rightarrow bool$› **where**
‹$is\text{-}pos\text{-}code\ L \longleftrightarrow bitAND\ L\ 1 = 0$›

Unit Propagation: Step

definition *delete-index-and-swap-update* :: ‹$('a \Rightarrow 'b\ list) \Rightarrow 'a \Rightarrow nat \Rightarrow 'a \Rightarrow 'b\ list$› **where**
‹$delete\text{-}index\text{-}and\text{-}swap\text{-}update\ W\ K\ w = W(K := delete\text{-}index\text{-}and\text{-}swap\ (W\ K)\ w)$›

The precondition is not necessary.

lemma *delete-index-and-swap-ll-delete-index-and-swap-update*:
‹$(uncurry2\ (RETURN\ ooo\ delete\text{-}index\text{-}and\text{-}swap\text{-}ll), uncurry2\ (RETURN\ ooo\ delete\text{-}index\text{-}and\text{-}swap\text{-}update))$
$\in[\lambda((W, L), i).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f\ (\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \times_r nat\text{-}lit\text{-}rel) \times_r nat\text{-}rel \rightarrow$
$\langle\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A})\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *append-update* :: ‹$('a \Rightarrow 'b\ list) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b\ list$› **where**
‹$append\text{-}update\ W\ L\ a = W(L{:=}\ W\ (L)\ @\ [a])$›

lemma *append-ll-append-update*:
‹$(uncurry2\ (RETURN\ ooo\ (\lambda xs\ i\ j.\ append\text{-}ll\ xs\ (nat\text{-}of\text{-}uint32\ i)\ j)), uncurry2\ (RETURN\ ooo\ append\text{-}update))$
$\in\ [\lambda((W, L), i).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f$
$\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \times_f unat\text{-}lit\text{-}rel \times_f Id \rightarrow \langle\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A})\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *is-decided-hd-trail-wl* **where**
‹$is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\ S = is\text{-}decided\ (hd\ (get\text{-}trail\text{-}wl\ S))$›

definition *is-decided-hd-trail-wll* :: ‹$nat\ twl\text{-}st\text{-}wl \Rightarrow bool\ nres$› **where**
‹$is\text{-}decided\text{-}hd\text{-}trail\text{-}wll = (\lambda(M, N, D, NE, UE, Q, W).$
$RETURN\ (is\text{-}decided\ (hd\ M))$
$)$›

lemma *Propagated-eq-ann-lit-of-pair-iff*:
‹$Propagated\ x21\ x22 = ann\text{-}lit\text{-}of\text{-}pair\ (a, b) \longleftrightarrow x21 = a \land b = Some\ x22$›
⟨*proof*⟩

definition *lit-and-ann-of-propagated-code* **where**
‹$lit\text{-}and\text{-}ann\text{-}of\text{-}propagated\text{-}code = (\lambda L{::}ann\text{-}lit\text{-}wl.\ (fst\ L, the\ (snd\ L)))$›

lemma *set-mset-all-lits-of-mm-atms-of-ms-iff*:

‹*set-mset* (*all-lits-of-mm* A) = *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\longleftrightarrow$ *atms-of-ms* (*set-mset* A) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
⟨*proof*⟩

definition *card-max-lvl* **where**
‹*card-max-lvl* M C ≡ *size* (*filter-mset* (λL. *get-level* M L = *count-decided* M) C)›

lemma *card-max-lvl-add-mset*: ‹*card-max-lvl* M (*add-mset* L C) =
(*if get-level* M L = *count-decided* M *then 1 else 0*) +
card-max-lvl M C›
⟨*proof*⟩

lemma *card-max-lvl-empty*[*simp*]: ‹*card-max-lvl* M {#} = *0*›
⟨*proof*⟩

lemma *card-max-lvl-all-poss*:
‹*card-max-lvl* M C = *card-max-lvl* M (*poss* (*atm-of* ‘# C))›
⟨*proof*⟩

lemma *card-max-lvl-distinct-cong*:
assumes
‹$\bigwedge L$. *get-level* M (*Pos* L) = *count-decided* M $\Longrightarrow$ ($L \in$ *atms-of* C) $\Longrightarrow$ ($L \in$ *atms-of* C')› **and**
‹$\bigwedge L$. *get-level* M (*Pos* L) = *count-decided* M $\Longrightarrow$ ($L \in$ *atms-of* C') $\Longrightarrow$ ($L \in$ *atms-of* C)› **and**
‹*distinct-mset* C› ‹¬*tautology* C› **and**
‹*distinct-mset* C'› ‹¬*tautology* C'›
shows ‹*card-max-lvl* M C = *card-max-lvl* M C'›
⟨*proof*⟩

end
theory *IsaSAT-Arena*
imports
Watched-Literals.WB-More-Refinement-List
Watched-Literals.WB-Word
IsaSAT-Literals
begin

0.1.4 The memory representation: Arenas

We implement an “arena” memory representation: This is a flat representation of clauses, where all clauses and their headers are put one after the other. A lot of the work done here could be done automatically by a C compiler (see paragraph on Cadical below).

While this has some advantages from a performance point of view compared to an array of arrays, it allows to emulate pointers to the middle of array with extra information put before the pointer. This is an optimisation that is considered as important (at least according to Armin Biere).

In Cadical, the representation is done that way although it is implicit by putting an array into a structure (and rely on UB behaviour to make sure that the array is “inlined” into the structure). Cadical also uses another trick: the array is but inside a union. This union contains either the clause or a pointer to the new position if it has been moved (during GC-ing). There is no way for us to do so in a type-safe manner that works both for *uint64* and *nat* (unless we know some details of the implementation). For *uint64*, we could use the space used by the headers. However, it is not clear if we want to do do, since the behaviour would change between the two types, making a comparison impossible. This means that half of the blocking literals will be

lost (if we iterate over the watch lists) or all (if we iterate over the clauses directly).
The order in memory is in the following order:

1. the saved position (is optional in cadical too);
2. the status;
3. the activity;
4. the LBD;
5. the size;
6. the clause.

Remark that the information can be compressed to reduce the size in memory:

1. the saved position can be skipped for short clauses;
2. the LBD will most of the time be much shorter than a 32-bit integer, so only an approximation can be kept and the remaining bits be reused;
3. the activity is not kept by cadical (to use instead a MTF-like scheme).

As we are already wasteful with memory, we implement the first optimisation. Point two can be implemented automatically by a (non-standard-compliant) C compiler.
In our case, the refinement is done in two steps:

1. First, we refine our clause-mapping to a big list. This list contains the original elements. For type safety, we introduce a datatype that enumerates all possible kind of elements.
2. Then, we refine all these elements to uint32 elements.

In our formalisation, we distinguish active clauses (clauses that are not marked to be deleted) from dead clauses (that have been marked to be deleted but can still be accessed). Any dead clause can be removed from the addressable clauses (*vdom* for virtual domain). Remark that we actually do not need the full virtual domain, just the list of all active position (TODO?).
Remark that in our formalisation, we don't (at least not yet) plan to reuse freed spaces (the predicate about dead clauses must be strengthened to do so). Due to the fact that an arena is very different from an array of clauses, we refine our data structure by hand to the long list instead of introducing refinement rules. This is mostly done because iteration is very different (and it does not change what we had before anyway).
Some technical details: due to the fact that we plan to refine the arena to uint32 and that our clauses can be tautologies, the size does not fit into uint32 (technically, we have the bound $\mathit{uint\text{-}max} + 1$). Therefore, we restrict the clauses to have at least length 2 and we keep $\mathit{length}\ C - 2$ instead of $\mathit{length}\ C$ (same for position saving). If we ever add a preprocessing path that removes tautologies, we could get rid of these two limitations.
To our own surprise, using an arena (without position saving) was exactly as fast as the our former resizable array of arrays. We did not expect this result since:

1. First, we cannot use *uint32* to iterate over clauses anymore (at least no without an additional trick like considering a slice).

2. Second, there is no reason why MLton would not already use the trick for array.

(We assume that there is no gain due the order in which we iterate over clauses, which seems a reasonnable assumption, even when considering than some clauses will subsume the previous one, and therefore, have a high chance to be in the same watch lists).

We can mark clause as used. This trick is used to implement a MTF-like scheme to keep clauses.

Status of a clause

datatype *clause-status* = *IRRED* | *LEARNED* | *DELETED*

instance *clause-status* :: *heap*
⟨*proof*⟩

instantiation *clause-status* :: *default*
begin

definition *default-clause-status* **where** ‹*default-clause-status* = *DELETED*›
instance ⟨*proof*⟩

end

Definition

The following definitions are the offset between the beginning of the clause and the specific headers before the beginning of the clause. Remark that the first offset is not always valid. Also remark that the fields are *before* the actual content of the clause.

definition *POS-SHIFT* :: *nat* **where**
‹*POS-SHIFT* = *5*›

definition *STATUS-SHIFT* :: *nat* **where**
‹*STATUS-SHIFT* = *4*›

definition *ACTIVITY-SHIFT* :: *nat* **where**
‹*ACTIVITY-SHIFT* = *3*›

definition *LBD-SHIFT* :: *nat* **where**
‹*LBD-SHIFT* = *2*›

definition *SIZE-SHIFT* :: *nat* **where**
‹*SIZE-SHIFT* = *1*›

definition *MAX-LENGTH-SHORT-CLAUSE* :: *nat* **where**
[*simp*]: ‹*MAX-LENGTH-SHORT-CLAUSE* = *4*›

definition *is-short-clause* **where**
[*simp*]: ‹*is-short-clause C* ⟷ *length C* ≤ *MAX-LENGTH-SHORT-CLAUSE*›

abbreviation *is-long-clause* **where**
‹*is-long-clause C* ≡ ¬*is-short-clause C*›

definition *header-size* :: ‹*nat clause-l* ⇒ *nat*› **where**
‹*header-size C* = (*if is-short-clause C then 4 else 5*)›

lemmas *SHIFTS-def* = *POS-SHIFT-def STATUS-SHIFT-def ACTIVITY-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def*

lemma *arena-shift-distinct*:
‹$i > 3 \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{LBD-SHIFT}$›
‹$i > 3 \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{ACTIVITY-SHIFT}$›
‹$i > 3 \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{STATUS-SHIFT}$›
‹$i > 3 \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{ACTIVITY-SHIFT}$›
‹$i > 3 \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{STATUS-SHIFT}$›
‹$i > 3 \Longrightarrow i - \textit{ACTIVITY-SHIFT} \neq i - \textit{STATUS-SHIFT}$›

‹$i > 4 \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i > 4 \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i > 4 \Longrightarrow i - \textit{ACTIVITY-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i > 4 \Longrightarrow i - \textit{STATUS-SHIFT} \neq i - \textit{POS-SHIFT}$›

‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i - \textit{SIZE-SHIFT} = j - \textit{SIZE-SHIFT} \longleftrightarrow i = j$›
‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i - \textit{LBD-SHIFT} = j - \textit{LBD-SHIFT} \longleftrightarrow i = j$›
‹$i > 4 \Longrightarrow j > 4 \Longrightarrow i - \textit{ACTIVITY-SHIFT} = j - \textit{ACTIVITY-SHIFT} \longleftrightarrow i = j$›
‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i - \textit{STATUS-SHIFT} = j - \textit{STATUS-SHIFT} \longleftrightarrow i = j$›
‹$i > 4 \Longrightarrow j > 4 \Longrightarrow i - \textit{POS-SHIFT} = j - \textit{POS-SHIFT} \longleftrightarrow i = j$›

‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{LBD-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{ACTIVITY-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{STATUS-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{ACTIVITY-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{STATUS-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow i - \textit{ACTIVITY-SHIFT} \neq i - \textit{STATUS-SHIFT}$›

‹$i \geq \textit{header-size } C \Longrightarrow \textit{is-long-clause } C \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow \textit{is-long-clause } C \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow \textit{is-long-clause } C \Longrightarrow i - \textit{ACTIVITY-SHIFT} \neq i - \textit{POS-SHIFT}$›
‹$i \geq \textit{header-size } C \Longrightarrow \textit{is-long-clause } C \Longrightarrow i - \textit{STATUS-SHIFT} \neq i - \textit{POS-SHIFT}$›

‹$i \geq \textit{header-size } C \Longrightarrow j \geq \textit{header-size } C' \Longrightarrow i - \textit{SIZE-SHIFT} = j - \textit{SIZE-SHIFT} \longleftrightarrow i = j$›
‹$i \geq \textit{header-size } C \Longrightarrow j \geq \textit{header-size } C' \Longrightarrow i - \textit{LBD-SHIFT} = j - \textit{LBD-SHIFT} \longleftrightarrow i = j$›
‹$i \geq \textit{header-size } C \Longrightarrow j \geq \textit{header-size } C' \Longrightarrow i - \textit{ACTIVITY-SHIFT} = j - \textit{ACTIVITY-SHIFT} \longleftrightarrow i = j$›
‹$i \geq \textit{header-size } C \Longrightarrow j \geq \textit{header-size } C' \Longrightarrow i - \textit{STATUS-SHIFT} = j - \textit{STATUS-SHIFT} \longleftrightarrow i = j$›
‹$i \geq \textit{header-size } C \Longrightarrow j \geq \textit{header-size } C' \Longrightarrow \textit{is-long-clause } C \Longrightarrow \textit{is-long-clause } C' \Longrightarrow$
$i - \textit{POS-SHIFT} = j - \textit{POS-SHIFT} \longleftrightarrow i = j$›
⟨*proof*⟩

lemma *header-size-ge0*[*simp*]: ‹$0 < \textit{header-size } x1$›
⟨*proof*⟩

datatype *arena-el* =
is-Lit: *ALit* (*xarena-lit*: ‹*nat literal*›) |
is-LBD: *ALBD* (*xarena-lbd*: *nat*) |
is-Act: *AActivity* (*xarena-act*: *nat*) |
is-Size: *ASize* (*xarena-length*: *nat*) |
is-Pos: *APos* (*xarena-pos*: *nat*) |
is-Status: *AStatus* (*xarena-status*: *clause-status*) (*xarena-used*: *bool*)

type-synonym *arena* = ‹*arena-el list*›

definition *xarena-active-clause* :: ‹*arena* ⇒ *nat clause-l* × *bool* ⇒ *bool*› **where**
‹*xarena-active-clause arena* = (λ(*C*, *red*).
$(\textit{length } C \geq 2 \land$
$\textit{header-size } C + \textit{length } C = \textit{length arena} \land$
$(\textit{is-long-clause } C \longrightarrow (\textit{is-Pos } (\textit{arena}!(\textit{header-size } C - \textit{POS-SHIFT})) \land$
$\textit{xarena-pos}(\textit{arena}!(\textit{header-size } C - \textit{POS-SHIFT})) \leq \textit{length } C - 2))) \land$
$\textit{is-Status}(\textit{arena}!(\textit{header-size } C - \textit{STATUS-SHIFT})) \land$
$(\textit{xarena-status}(\textit{arena}!(\textit{header-size } C - \textit{STATUS-SHIFT})) = \textit{IRRED} \longleftrightarrow \textit{red}) \land$
$(\textit{xarena-status}(\textit{arena}!(\textit{header-size } C - \textit{STATUS-SHIFT})) = \textit{LEARNED} \longleftrightarrow \neg\textit{red}) \land$
$\textit{is-LBD}(\textit{arena}!(\textit{header-size } C - \textit{LBD-SHIFT})) \land$
$\textit{is-Act}(\textit{arena}!(\textit{header-size } C - \textit{ACTIVITY-SHIFT})) \land$
$\textit{is-Size}(\textit{arena}!(\textit{header-size } C - \textit{SIZE-SHIFT})) \land$
$\textit{xarena-length}(\textit{arena}!(\textit{header-size } C - \textit{SIZE-SHIFT})) + 2 = \textit{length } C \land$
$\textit{drop } (\textit{header-size } C) \textit{ arena} = \textit{map ALit } C$
))›

As $(N \propto i, \textit{irred } N\ i)$ is automatically simplified to *the* (*fmlookup* N i), we provide an alternative definition that uses the result after the simplification.

lemma *xarena-active-clause-alt-def*:
‹*xarena-active-clause arena* (*the* (*fmlookup N i*)) ⟷ (
$(\textit{length } (N\propto i) \geq 2 \land$
$\textit{header-size } (N\propto i) + \textit{length } (N\propto i) = \textit{length arena} \land$
$(\textit{is-long-clause } (N\propto i) \longrightarrow (\textit{is-Pos } (\textit{arena}!(\textit{header-size } (N\propto i) - \textit{POS-SHIFT})) \land$
$\textit{xarena-pos}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{POS-SHIFT})) \leq \textit{length } (N\propto i) - 2)) \land$
$\textit{is-Status}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{STATUS-SHIFT})) \land$
$(\textit{xarena-status}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{STATUS-SHIFT})) = \textit{IRRED} \longleftrightarrow \textit{irred } N\ i) \land$
$(\textit{xarena-status}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{STATUS-SHIFT})) = \textit{LEARNED} \longleftrightarrow \neg\textit{irred } N\ i) \land$
$\textit{is-LBD}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{LBD-SHIFT})) \land$
$\textit{is-Act}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{ACTIVITY-SHIFT})) \land$
$\textit{is-Size}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{SIZE-SHIFT})) \land$
$\textit{xarena-length}(\textit{arena}!(\textit{header-size } (N\propto i) - \textit{SIZE-SHIFT})) + 2 = \textit{length } (N\propto i) \land$
$\textit{drop } (\textit{header-size } (N\propto i)) \textit{ arena} = \textit{map ALit } (N\propto i)$
))›
⟨*proof*⟩

The extra information is required to prove "separation" between active and dead clauses. And it is true anyway and does not require any extra work to prove. TODO generalise LBD to extract from every clause?

definition *arena-dead-clause* :: ‹*arena* ⇒ *bool*› **where**
‹*arena-dead-clause arena* ⟷
$\textit{is-Status}(\textit{arena}!(4 - \textit{STATUS-SHIFT})) \land \textit{xarena-status}(\textit{arena}!(4 - \textit{STATUS-SHIFT})) = \textit{DELETED}$
∧
$\textit{is-LBD}(\textit{arena}!(4 - \textit{LBD-SHIFT})) \land$
$\textit{is-Act}(\textit{arena}!(4 - \textit{ACTIVITY-SHIFT})) \land$
$\textit{is-Size}(\textit{arena}!(4 - \textit{SIZE-SHIFT}))$
›

When marking a clause as garbage, we do not care whether it was used or not.

definition *extra-information-mark-to-delete* **where**
‹*extra-information-mark-to-delete arena i* = *arena*[*i* − *STATUS-SHIFT* := *AStatus DELETED False*]›

This extracts a single clause from the complete arena.

abbreviation *clause-slice* **where**

⟨*clause-slice arena N i* ≡ *Misc.slice* (*i* − *header-size* (*N*∝*i*)) (*i* + *length*(*N*∝*i*)) *arena*⟩

abbreviation *dead-clause-slice* **where**
⟨*dead-clause-slice arena N i* ≡ *Misc.slice* (*i* − *4*) *i arena*⟩

We now can lift the validity of the active and dead clauses to the whole memory and link it the mapping to clauses and the addressable space.

In our first try, the predicated *xarena-active-clause* took the whole arena as parameter. This however turned out to make the proof about updates less modular, since the slicing already takes care to ignore all irrelevant changes.

definition *valid-arena* :: ⟨*arena* ⇒ *nat clauses-l* ⇒ *nat set* ⇒ *bool*⟩ **where**
⟨*valid-arena arena N vdom* ⟷
(∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))) ∧
(∀ *i* ∈ *vdom*. *i* ∉# *dom-m N* ⟶ (*i* < *length arena* ∧ *i* ≥ *4* ∧
arena-dead-clause (*dead-clause-slice arena N i*)))
⟩

lemma *valid-arena-empty*: ⟨*valid-arena* [] *fmempty* {}⟩
⟨*proof*⟩

definition *arena-status* **where**
⟨*arena-status arena i* = *xarena-status* (*arena*!(*i* − *STATUS-SHIFT*))⟩

definition *arena-used* **where**
⟨*arena-used arena i* = *xarena-used* (*arena*!(*i* − *STATUS-SHIFT*))⟩

definition *arena-length* **where**
⟨*arena-length arena i* = *2* + *xarena-length* (*arena*!(*i* − *SIZE-SHIFT*))⟩

definition *arena-lbd* **where**
⟨*arena-lbd arena i* = *xarena-lbd* (*arena*!(*i* − *LBD-SHIFT*))⟩

definition *arena-act* **where**
⟨*arena-act arena i* = *xarena-act* (*arena*!(*i* − *ACTIVITY-SHIFT*))⟩

definition *arena-pos* **where**
⟨*arena-pos arena i* = *2* + *xarena-pos* (*arena*!(*i* − *POS-SHIFT*))⟩

definition *arena-lit* **where**
⟨*arena-lit arena i* = *xarena-lit* (*arena*!*i*)⟩

Separation properties

The following two lemmas talk about the minimal distance between two clauses in memory. They are important for the proof of correctness of all update function.

lemma *minimal-difference-between-valid-index*:
assumes ⟨∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))⟩ **and**
⟨*i* ∈# *dom-m N*⟩ **and** ⟨*j* ∈# *dom-m N*⟩ **and** ⟨*j* > *i*⟩
shows ⟨*j* − *i* ≥ *length* (*N*∝*i*) + *header-size* (*N*∝*j*)⟩
⟨*proof*⟩

lemma *minimal-difference-between-invalid-index*:

assumes ‹*valid-arena arena N vdom*› **and**
‹$i \in\#$ *dom-m N*› **and** ‹$j \notin\#$ *dom-m N*› **and** ‹$j \geq i$› **and** ‹$j \in$ *vdom*›
shows ‹$j - i \geq$ *length* $(N \propto i) + 4$›
⟨*proof*⟩

At first we had the weaker $(1::'a) \leq i - j$ which we replaced by $(4::'a) \leq i - j$. The former however was able to solve many more goals due to different handling between $1::'a$ (which is simplified to *Suc 0*) and $4::'a$ (which is not). Therefore, we replaced $4::'a$ by *Suc* (*Suc* (*Suc* (*Suc 0*)))

lemma *minimal-difference-between-invalid-index2*:
assumes ‹*valid-arena arena N vdom*› **and**
‹$i \in\#$ *dom-m N*› **and** ‹$j \notin\#$ *dom-m N*› **and** ‹$j < i$› **and** ‹$j \in$ *vdom*›
shows ‹$i - j \geq$ *Suc* (*Suc* (*Suc* (*Suc 0*)))› **and**
‹*is-long-clause* $(N \propto i) \Longrightarrow i - j \geq$ *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))›
⟨*proof*⟩

lemma *valid-arena-in-vdom-le-arena*:
assumes ‹*valid-arena arena N vdom*› **and** ‹$j \in$ *vdom*›
shows ‹$j <$ *length arena*› **and** ‹$j \geq 4$›
⟨*proof*⟩

lemma *valid-minimal-difference-between-valid-index*:
assumes ‹*valid-arena arena N vdom*› **and**
‹$i \in\#$ *dom-m N*› **and** ‹$j \in\#$ *dom-m N*› **and** ‹$j > i$›
shows ‹$j - i \geq$ *length* $(N \propto i)$ + *header-size* $(N \propto j)$›
⟨*proof*⟩

Updates

Mark to delete

lemma *clause-slice-extra-information-mark-to-delete*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \in\#$ *dom-m N*› **and**
dom: ‹$\forall i \in\#$ *dom-m N*. $i <$ *length arena* $\wedge$ $i \geq$ *header-size* $(N \propto i)$ $\wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*extra-information-mark-to-delete arena i*) *N ia* =
(*if ia = i then extra-information-mark-to-delete* (*clause-slice arena N ia*) (*header-size* $(N \propto i)$)
else clause-slice arena N ia)›
⟨*proof*⟩

lemma *clause-slice-extra-information-mark-to-delete-dead*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \notin\#$ *dom-m N*› ‹$ia \in$ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*extra-information-mark-to-delete arena i*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
⟨*proof*⟩

lemma *length-extra-information-mark-to-delete*[*simp*]:
‹*length* (*extra-information-mark-to-delete arena i*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-mono*: ‹*valid-arena ab ar vdom1* $\Longrightarrow$ *vdom2* $\subseteq$ *vdom1* $\Longrightarrow$ *valid-arena ab ar vdom2*›
⟨*proof*⟩

lemma *valid-arena-extra-information-mark-to-delete*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) (*insert i vdom*)›
⟨*proof*⟩

lemma *valid-arena-extra-information-mark-to-delete'*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *vdom*›
⟨*proof*⟩

Removable from addressable space

lemma *valid-arena-remove-from-vdom*:
assumes ‹*valid-arena arena N* (*insert i vdom*)›
shows ‹*valid-arena arena N vdom*›
⟨*proof*⟩

Update activity

definition *update-act* **where**
‹*update-act C act arena* = *arena*[C − *ACTIVITY-SHIFT* := *AActivity act*]›

lemma *clause-slice-update-act*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \in\#$ *dom-m N*› **and**
dom: ‹$\forall i \in\#$ *dom-m N*. $i <$ *length arena* $\wedge$ $i \geq$ *header-size* ($N \propto i$) $\wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*update-act i act arena*) *N ia* =
(*if ia* = *i then update-act* (*header-size* ($N \propto i$)) *act* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
⟨*proof*⟩

lemma *length-update-act*[*simp*]:
‹*length* (*update-act i act arena*) = *length arena*›
⟨*proof*⟩

lemma *clause-slice-update-act-dead*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \notin\#$ *dom-m N*› ‹$ia \in$ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-act i act arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
⟨*proof*⟩

lemma *xarena-active-clause-update-act-same*:
assumes
‹$i \geq$ *header-size* ($N \propto i$)› **and**
‹$i <$ *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*update-act* (*header-size* ($N \propto i$)) *act* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›

⟨*proof*⟩

lemma *valid-arena-update-act*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*update-act i act arena*) *N vdom*›
⟨*proof*⟩

Update LBD **definition** *update-lbd* **where**
‹*update-lbd C lbd arena* = *arena*[*C* − *LBD-SHIFT* := *ALBD lbd*]›

lemma *clause-slice-update-lbd*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \in\#$ *dom-m N*› **and**
dom: ‹$\forall i \in\#$ *dom-m N*. $i <$ *length arena* $\wedge$ $i \geq$ *header-size* ($N \propto i$) $\wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*update-lbd i lbd arena*) *N ia* =
(*if ia* = *i then update-lbd* (*header-size* ($N \propto i$)) *lbd* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
⟨*proof*⟩

lemma *length-update-lbd*[*simp*]:
‹*length* (*update-lbd i lbd arena*) = *length arena*›
⟨*proof*⟩

lemma *clause-slice-update-lbd-dead*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \notin\#$ *dom-m N*› ‹$ia \in$ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-lbd i lbd arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
⟨*proof*⟩

lemma *xarena-active-clause-update-lbd-same*:
assumes
‹$i \geq$ *header-size* ($N \propto i$)› **and**
‹$i <$ *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*update-lbd* (*header-size* ($N \propto i$)) *lbd* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
⟨*proof*⟩

lemma *valid-arena-update-lbd*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*update-lbd i lbd arena*) *N vdom*›
⟨*proof*⟩

Update saved position **definition** *update-pos-direct* **where**

$\langle$*update-pos-direct C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos pos*]$\rangle$

lemma *clause-slice-update-pos*:
  assumes
    i: $\langle i \in\# \mathit{dom\text{-}m}\ N\rangle$ **and**
    ia: $\langle ia \in\# \mathit{dom\text{-}m}\ N\rangle$ **and**
    dom: $\langle\forall i \in\# \mathit{dom\text{-}m}\ N.\ i < \mathit{length}\ \mathit{arena} \wedge i \geq \mathit{header\text{-}size}\ (N{\propto}i) \wedge$
        $\mathit{xarena\text{-}active\text{-}clause}\ (\mathit{clause\text{-}slice}\ \mathit{arena}\ N\ i)\ (\mathit{the}\ (\mathit{fmlookup}\ N\ i))\rangle$ **and**
    long: $\langle\mathit{is\text{-}long\text{-}clause}\ (N \propto i)\rangle$
  shows
    $\langle\mathit{clause\text{-}slice}\ (\mathit{update\text{-}pos\text{-}direct}\ i\ \mathit{pos}\ \mathit{arena})\ N\ ia =$
      $(\mathit{if}\ ia = i\ \mathit{then}\ \mathit{update\text{-}pos\text{-}direct}\ (\mathit{header\text{-}size}\ (N{\propto}i))\ \mathit{pos}\ (\mathit{clause\text{-}slice}\ \mathit{arena}\ N\ ia)$
        $\mathit{else}\ \mathit{clause\text{-}slice}\ \mathit{arena}\ N\ ia)\rangle$
$\langle\mathit{proof}\rangle$

lemma *clause-slice-update-pos-dead*:
  assumes
    i: $\langle i \in\# \mathit{dom\text{-}m}\ N\rangle$ **and**
    ia: $\langle ia \notin\# \mathit{dom\text{-}m}\ N\rangle$ $\langle ia \in \mathit{vdom}\rangle$ **and**
    dom: $\langle\mathit{valid\text{-}arena}\ \mathit{arena}\ N\ \mathit{vdom}\rangle$ **and**
    long: $\langle\mathit{is\text{-}long\text{-}clause}\ (N \propto i)\rangle$
  shows
    $\langle\mathit{arena\text{-}dead\text{-}clause}\ (\mathit{dead\text{-}clause\text{-}slice}\ (\mathit{update\text{-}pos\text{-}direct}\ i\ \mathit{pos}\ \mathit{arena})\ N\ ia) =$
      $\mathit{arena\text{-}dead\text{-}clause}\ (\mathit{dead\text{-}clause\text{-}slice}\ \mathit{arena}\ N\ ia)\rangle$
$\langle\mathit{proof}\rangle$

lemma *xarena-active-clause-update-pos-same*:
  assumes
    $\langle i \geq \mathit{header\text{-}size}\ (N \propto i)\rangle$ **and**
    $\langle i < \mathit{length}\ \mathit{arena}\rangle$ **and**
    $\langle\mathit{xarena\text{-}active\text{-}clause}\ (\mathit{clause\text{-}slice}\ \mathit{arena}\ N\ i)$
     $(\mathit{the}\ (\mathit{fmlookup}\ N\ i))\rangle$ **and**
    long: $\langle\mathit{is\text{-}long\text{-}clause}\ (N \propto i)\rangle$ **and**
    $\langle\mathit{pos} \leq \mathit{length}\ (N \propto i) - 2\rangle$
  shows $\langle\mathit{xarena\text{-}active\text{-}clause}\ (\mathit{update\text{-}pos\text{-}direct}\ (\mathit{header\text{-}size}\ (N{\propto}i))\ \mathit{pos}\ (\mathit{clause\text{-}slice}\ \mathit{arena}\ N\ i))$
     $(\mathit{the}\ (\mathit{fmlookup}\ N\ i))\rangle$
  $\langle\mathit{proof}\rangle$

lemma *length-update-pos*[*simp*]:
  $\langle\mathit{length}\ (\mathit{update\text{-}pos\text{-}direct}\ i\ \mathit{pos}\ \mathit{arena}) = \mathit{length}\ \mathit{arena}\rangle$
  $\langle\mathit{proof}\rangle$

lemma *valid-arena-update-pos*:
  assumes *arena*: $\langle\mathit{valid\text{-}arena}\ \mathit{arena}\ N\ \mathit{vdom}\rangle$ **and** *i*: $\langle i \in\# \mathit{dom\text{-}m}\ N\rangle$ **and**
    long: $\langle\mathit{is\text{-}long\text{-}clause}\ (N \propto i)\rangle$**and**
    pos: $\langle\mathit{pos} \leq \mathit{length}\ (N \propto i) - 2\rangle$
  shows $\langle\mathit{valid\text{-}arena}\ (\mathit{update\text{-}pos\text{-}direct}\ i\ \mathit{pos}\ \mathit{arena})\ N\ \mathit{vdom}\rangle$
$\langle\mathit{proof}\rangle$

Swap literals **definition** *swap-lits* **where**
  $\langle\mathit{swap\text{-}lits}\ C\ i\ j\ \mathit{arena} = \mathit{swap}\ \mathit{arena}\ (C + i)\ (C + j)\rangle$

lemma *clause-slice-swap-lits*:
  assumes
    i: $\langle i \in\# \mathit{dom\text{-}m}\ N\rangle$ **and**

ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows
‹*clause-slice* (*swap-lits i k l arena*) *N ia* =
(*if ia* = *i then swap-lits* (*header-size* (*N*∝*i*)) *k l* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
⟨*proof*⟩

lemma *length-swap-lits*[*simp*]:
‹*length* (*swap-lits i k l arena*) = *length arena*›
⟨*proof*⟩

lemma *clause-slice-swap-lits-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›**and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*swap-lits i k l arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
⟨*proof*⟩

lemma *xarena-active-clause-swap-lits-same*:
assumes
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›**and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows ‹*xarena-active-clause* (*clause-slice* (*swap-lits i k l arena*) *N i*)
(*the* (*fmlookup* (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *i*))›
⟨*proof*⟩

lemma *is-short-clause-swap*[*simp*]: ‹*is-short-clause* (*swap* (*N* ∝ *i*) *k l*) = *is-short-clause* (*N* ∝ *i*)›
⟨*proof*⟩

lemma *header-size-swap*[*simp*]: ‹*header-size* (*swap* (*N* ∝ *i*) *k l*) = *header-size* (*N* ∝ *i*)›
⟨*proof*⟩

lemma *valid-arena-swap-lits*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows ‹*valid-arena* (*swap-lits i k l arena*) (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *vdom*›
⟨*proof*⟩

Learning a clause **definition** *append-clause-skeleton* **where**
‹*append-clause-skeleton pos st used act lbd C arena* =
(*if is-short-clause C then*
arena @ (*AStatus st used*) # *AActivity act* # *ALBD lbd* #

ASize (*length C* − *2*) # *map ALit C*
else arena @ *APos pos* # (*AStatus st used*) # *AActivity act* #
ALBD lbd # *ASize* (*length C* − *2*) # *map ALit C*)›

definition *append-clause* **where**
‹*append-clause b C arena* =
append-clause-skeleton 0 (*if b then IRRED else LEARNED*) *False 0* (*length C* − *2*) *C arena*›

lemma *arena-active-clause-append-clause*:
assumes
‹$i \geq$ *header-size* ($N \propto i$)› **and**
‹$i <$ *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N i*)
(*the* (*fmlookup N i*))›
⟨*proof*⟩

lemma *length-append-clause*[*simp*]:
‹*length* (*append-clause-skeleton pos st used act lbd C arena*) =
length arena + *length C* + *header-size C*›
‹*length* (*append-clause b C arena*) = *length arena* + *length C* + *header-size C*›
⟨*proof*⟩

lemma *arena-active-clause-append-clause-same*: ‹$2 \leq$ *length C* $\Longrightarrow$ *st* $\neq$ *DELETED* $\Longrightarrow$
pos $\leq$ *length C* − *2* $\Longrightarrow$
b $\longleftrightarrow$ (*st* = *IRRED*) $\Longrightarrow$
xarena-active-clause
(*Misc.slice* (*length arena*) (*length arena* + *header-size C* + *length C*)
(*append-clause-skeleton pos st used act lbd C arena*))
(*the* (*fmlookup* (*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*length arena* + *header-size C*)))›
⟨*proof*⟩

lemma *clause-slice-append-clause*:
assumes
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*› **and**
‹*arena-dead-clause* (*dead-clause-slice* (*arena*) *N ia*)›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N ia*)›
⟨*proof*⟩

lemma *valid-arena-append-clause-skeleton*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* $\geq$ *2*› **and**
b: ‹*b* $\longleftrightarrow$ (*st* = *IRRED*)› **and** *st*: ‹*st* $\neq$ *DELETED*› **and**
pos: ‹*pos* $\leq$ *length C* − *2*›
shows ‹*valid-arena* (*append-clause-skeleton pos st used act lbd C arena*)
(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*insert* (*length arena* + *header-size C*) *vdom*)›
⟨*proof*⟩

lemma *valid-arena-append-clause*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* $\geq$ *2*›
shows ‹*valid-arena* (*append-clause b C arena*)
(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)

(*insert* (*length arena* + *header-size C*) *vdom*)›
⟨*proof*⟩

Refinement Relation

definition *status-rel*:: (*nat* × *clause-status*) *set* **where**
‹*status-rel* = {(*0*, *IRRED*), (*1*, *LEARNED*), (*3*, *DELETED*)}›

definition *bitfield-rel* **where**
‹*bitfield-rel n* = {(*a*, *b*). *b* ⟷ *a AND* (*2* ^ *n*) > *0*}›

definition *arena-el-relation* **where**
‹*arena-el-relation x el* = (*case el of*
AStatus n b ⇒ (*x AND 0b11*, *n*) ∈ *status-rel* ∧ (*x*, *b*) ∈ *bitfield-rel 2*
| *APos n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ASize n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ALBD n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *AActivity n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ALit n* ⇒ (*x*, *n*) ∈ *nat-lit-rel*
)›

definition *arena-el-rel* **where**
arena-el-rel-interal-def: ‹*arena-el-rel* = {(*x*, *el*). *arena-el-relation x el*}›

lemmas *arena-el-rel-def* = *arena-el-rel-interal-def*[*unfolded arena-el-relation-def*]

Preconditions and Assertions for the refinement

The following lemma expresses the relation between the arena and the clauses and especially shows the preconditions to be able to generate code.

The conditions on *arena-status* are in the direction to simplify proofs: If we would try to go in the opposite direction, we could rewrite ¬ *irred N i* into *arena-status arena i* ≠ *LEARNED*, which is a weaker property.

The inequality on the length are here to enable simp to prove inequalities *Suc 0* < *arena-length arena C* automatically. Normally the arithmetic part can prove it from *2* ≤ *arena-length arena C*, but as this inequality is simplified away, it does not work.

lemma *arena-lifting*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹*i* ∈# *dom-m N*›
shows
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*›
‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
‹*length* (*N* ∝ *i*) = *arena-length arena i*›
‹*j* < *length* (*N* ∝ *i*) ⟹ *N* ∝ *i* ! *j* = *arena-lit arena* (*i* + *j*)› **and**
‹*j* < *length* (*N* ∝ *i*) ⟹ *is-Lit* (*arena* ! (*i*+*j*))› **and**
‹*j* < *length* (*N* ∝ *i*) ⟹ *i* + *j* < *length arena*› **and**
‹*N* ∝ *i* ! *0* = *arena-lit arena i*› **and**
‹*is-Lit* (*arena* ! *i*)› **and**
‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*› **and**
‹*is-long-clause* (*N* ∝ *i*) ⟹ *is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› **and**
‹*is-long-clause* (*N* ∝ *i*) ⟹ *arena-pos arena i* ≤ *arena-length arena i*› **and**
‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› **and**
‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› **and**

‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))› **and**
‹*SIZE-SHIFT* ≤ *i*› **and**
‹*LBD-SHIFT* ≤ *i*›
‹*ACTIVITY-SHIFT* ≤ *i*› **and**
‹*arena-length arena i* ≥ *2*› **and**
‹*arena-length arena i* ≥ *Suc 0*› **and**
‹*arena-length arena i* ≥ *0*› **and**
‹*arena-length arena i* > *Suc 0*› **and**
‹*arena-length arena i* > *0*› **and**
‹*arena-status arena i* = *LEARNED* ⟷ ¬*irred N i*› **and**
‹*arena-status arena i* = *IRRED* ⟷ *irred N i*› **and**
‹*arena-status arena i* ≠ *DELETED*› **and**
‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* (*N* ∝ *i*)›
⟨*proof*⟩

lemma *arena-dom-status-iff*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹*i* ∈ *vdom*›
shows
‹*i* ∈# *dom-m N* ⟷ *arena-status arena i* ≠ *DELETED*› (**is** ‹*?eq*› **is** ‹*?A* ⟷ *?B*›) **and**
‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› (**is** *?lbd*) **and**
‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› (**is** *?act*) **and**
‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))› (**is** *?stat*) **and**
‹*4* ≤ *i*› (**is** *?ge*)
⟨*proof*⟩

lemma *valid-arena-one-notin-vdomD*:
‹*valid-arena M N vdom* ⟹ *Suc 0* ∉ *vdom*›
⟨*proof*⟩

This is supposed to be used as for assertions. There might be a more "local" way to define it, without the need for an existentially quantified clause set. However, I did not find a definition which was really much more useful and more practical.

definition *arena-is-valid-clause-idx* :: ‹*arena* ⇒ *nat* ⇒ *bool*› **where**
‹*arena-is-valid-clause-idx arena i* ⟷
(∃ *N vdom*. *valid-arena arena N vdom* ∧ *i* ∈# *dom-m N*)›

This precondition has weaker preconditions is restricted to extracting the status (the other headers can be extracted but only garbage is returned).

definition *arena-is-valid-clause-vdom* :: ‹*arena* ⇒ *nat* ⇒ *bool*› **where**
‹*arena-is-valid-clause-vdom arena i* ⟷
(∃ *N vdom*. *valid-arena arena N vdom* ∧ *i* ∈ *vdom*)›

lemma *nat-of-uint32-div*:
‹*nat-of-uint32* (*a div b*) = *nat-of-uint32 a div nat-of-uint32 b*›
⟨*proof*⟩

lemma *SHIFTS-alt-def*:
‹*POS-SHIFT* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))›
‹*STATUS-SHIFT* = *Suc* (*Suc* (*Suc* (*Suc 0*)))›
‹*ACTIVITY-SHIFT* = *Suc* (*Suc* (*Suc 0*))›
‹*LBD-SHIFT* = *Suc* (*Suc 0*)›
‹*SIZE-SHIFT* = *Suc 0*›
⟨*proof*⟩

Code Generation

Length **definition** *isa-arena-length* **where**
‹*isa-arena-length arena i* = *do* {
    ASSERT($i \geq$ *SIZE-SHIFT* $\wedge$ $i <$ *length arena*);
    RETURN (*two-uint64* + *uint64-of-uint32* ((*arena* ! (*fast-minus i SIZE-SHIFT*))))
}›

lemma *arena-length-uint64-conv*:
assumes
a: ‹(*a*, *aa*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› **and**
ba: ‹*ba* ∈# *dom-m N*› **and**
valid: ‹*valid-arena aa N vdom*›
shows ‹*Suc* (*Suc* (*xarena-length* (*aa* ! (*ba* − *SIZE-SHIFT*)))) =
nat-of-uint64 (*2* + *uint64-of-uint32* (*a* ! (*ba* − *SIZE-SHIFT*)))›
⟨*proof*⟩

lemma *isa-arena-length-arena-length*:
‹(*uncurry* (*isa-arena-length*), *uncurry* (*RETURN oo arena-length*)) ∈
$[$*uncurry arena-is-valid-clause-idx*$]_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_r$ *nat-rel* → ⟨*uint64-nat-rel*⟩*nres-rel*›
⟨*proof*⟩

Literal at given position **definition** *isa-arena-lit* **where**
‹*isa-arena-lit arena i* = *do* {
    ASSERT($i <$ *length arena*);
    RETURN (*arena* ! *i*)
}›

lemma *arena-length-literal-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* ∈# *dom-m N*› **and**
ba-le: ‹*ba* − *j* < *arena-length arena j*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› **and**
ba-j: ‹*ba* ≥ *j*›
shows
‹*ba* < *length arena*› (**is** *?le*) **and**
‹(*a* ! *ba*, *xarena-lit* (*arena* ! *ba*)) ∈ *unat-lit-rel*› (**is** *?unat*)
⟨*proof*⟩

definition *arena-is-valid-clause-idx-and-access* :: ‹*arena* ⇒ *nat* ⇒ *nat* ⇒ *bool*› **where**
‹*arena-is-valid-clause-idx-and-access arena i j* ⟷
(∃ *N vdom*. *valid-arena arena N vdom* ∧ *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*))›

This is the precondition for direct memory access: N ! i where $i = j + (j - i)$ instead of $N \propto j$! $(i - j)$.

definition *arena-lit-pre* **where**
‹*arena-lit-pre arena i* ⟷
(∃ *j*. *i* ≥ *j* ∧ *arena-is-valid-clause-idx-and-access arena j* (*i* − *j*))›

lemma *isa-arena-lit-arena-lit*:
‹(*uncurry isa-arena-lit*, *uncurry* (*RETURN oo arena-lit*)) ∈
$[$*uncurry arena-lit-pre*$]_f$

$\langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel* $\times_r$ *nat-rel* $\rightarrow$ $\langle$*unat-lit-rel*$\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

Status of the clause **definition** *isa-arena-status* **where**
‹*isa-arena-status arena i* = *do* {
  ASSERT(*i* < *length arena*);
  ASSERT(*i* $\geq$ *STATUS-SHIFT*);
  RETURN (*arena* ! (*fast-minus i STATUS-SHIFT*) *AND 0b11*)
}›

lemma *arena-status-literal-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* $\in$ *x*› **and**
a: ‹(*a*, *arena*) $\in$ $\langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel*›
shows
‹*j* < *length arena*› (**is** *?le*) **and**
‹*4* $\leq$ *j*› **and**
‹*j* $\geq$ *STATUS-SHIFT*› **and**
‹ (*a* ! (*j* − *STATUS-SHIFT*) *AND 0b11*, *xarena-status* (*arena* ! (*j* − *STATUS-SHIFT*)))
  $\in$ *uint32-nat-rel O status-rel*› (**is** *?rel*)
$\langle$*proof*$\rangle$

lemma *isa-arena-status-arena-status*:
‹(*uncurry isa-arena-status*, *uncurry* (*RETURN oo arena-status*)) $\in$
  [*uncurry arena-is-valid-clause-vdom*]$_f$
  $\langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel* $\times_r$ *nat-rel* $\rightarrow$ $\langle$*uint32-nat-rel O status-rel*$\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

Swap literals **definition** *isa-arena-swap* **where**
‹*isa-arena-swap C i j arena* = *do* {
  ASSERT(*C* + *i* < *length arena* $\wedge$ *C* + *j* < *length arena*);
  RETURN (*swap arena* (*C*+*i*) (*C*+*j*))
}›

definition *swap-lits-pre* **where**
‹*swap-lits-pre C i j arena* $\longleftrightarrow$ *C* + *i* < *length arena* $\wedge$ *C* + *j* < *length arena*›

lemma *isa-arena-swap*:
‹(*uncurry3 isa-arena-swap*, *uncurry3* (*RETURN oooo swap-lits*)) $\in$
  [*uncurry3 swap-lits-pre*]$_f$
  nat-rel $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ $\langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel* $\rightarrow$
  $\langle\langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel*$\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

Update LBD **definition** *isa-update-lbd* :: ‹*nat* $\Rightarrow$ *uint32* $\Rightarrow$ *uint32 list* $\Rightarrow$ *uint32 list nres*› **where**
‹*isa-update-lbd C lbd arena* = *do* {
  ASSERT(*C* − *LBD-SHIFT* < *length arena* $\wedge$ *C* $\geq$ *LBD-SHIFT*);
  RETURN (*arena* [*C* − *LBD-SHIFT* := *lbd*])
}›

lemma *arena-lbd-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**

j: ‹$j \in\#$ *dom-m N*› **and**
a: ‹$(a, \ arena) \in \langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel*› **and**
b: ‹$(b, \ bb) \in$ *uint32-nat-rel*›
shows
‹j − *LBD-SHIFT* < *length arena*› (**is** *?le*) **and**
‹$(a[j - LBD\text{-}SHIFT := b]$, *update-lbd j bb arena*) $\in \langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel*›
(**is** *?unat*)
⟨*proof*⟩

definition *update-lbd-pre* **where**
‹*update-lbd-pre* = (λ((*C*, *lbd*), *arena*). *arena-is-valid-clause-idx arena C*)›

lemma *isa-update-lbd*:
‹(*uncurry2 isa-update-lbd*, *uncurry2* (*RETURN ooo update-lbd*)) $\in$
[*update-lbd-pre*]$_f$
nat-rel $\times_f$ *uint32-nat-rel* $\times_f$ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* →
⟨⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

Get LBD **definition** *get-clause-LBD* :: ‹*arena* ⇒ *nat* ⇒ *nat*› **where**
‹*get-clause-LBD arena C* = *xarena-lbd* (*arena* ! (*C* − *LBD-SHIFT*))›

definition *get-clause-LBD-pre* **where**
‹*get-clause-LBD-pre* = *arena-is-valid-clause-idx*›

definition *isa-get-clause-LBD* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint32 nres*› **where**
‹*isa-get-clause-LBD arena C* = *do* {
ASSERT(*C* − *LBD-SHIFT* < *length arena* ∧ *C* ≥ *LBD-SHIFT*);
RETURN (*arena* ! (*C* − *LBD-SHIFT*))
}›

lemma *arena-get-lbd-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹$j \in\#$ *dom-m N*› **and**
a: ‹$(a, \ arena) \in \langle$*uint32-nat-rel O arena-el-rel*$\rangle$*list-rel*›
shows
‹j − *LBD-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*LBD-SHIFT* ≤ j› (**is** *?ge*) **and**
‹(a ! (j − *LBD-SHIFT*),
xarena-lbd (*arena* ! (j − *LBD-SHIFT*)))
∈ *uint32-nat-rel*›
⟨*proof*⟩

lemma *isa-get-clause-LBD-get-clause-LBD*:
‹(*uncurry isa-get-clause-LBD*, *uncurry* (*RETURN oo get-clause-LBD*)) ∈
[*uncurry get-clause-LBD-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨*uint32-nat-rel*⟩*nres-rel*›
⟨*proof*⟩

Saved position **definition** *get-saved-pos-pre* **where**
‹*get-saved-pos-pre arena C* ⟷ *arena-is-valid-clause-idx arena C* ∧
arena-length arena C > *MAX-LENGTH-SHORT-CLAUSE*›

definition *isa-get-saved-pos* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint64 nres*› **where**
‹*isa-get-saved-pos arena C* = *do* {
$\quad$ *ASSERT*(*C* − *POS-SHIFT* < *length arena* ∧ *C* ≥ *POS-SHIFT*);
$\quad$ *RETURN* (*uint64-of-uint32* (*arena* ! (*C* − *POS-SHIFT*)) + *two-uint64*)
}›

lemma *arena-get-pos-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* ∈# *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› **and**
length: ‹*arena-length arena j* > *MAX-LENGTH-SHORT-CLAUSE*›
shows
‹*j* − *POS-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*POS-SHIFT* ≤ *j*› (**is** *?ge*) **and**
‹(*uint64-of-uint32* (*a* ! (*j* − *POS-SHIFT*)) + *two-uint64*,
arena-pos arena j)
∈ *uint64-nat-rel*› (**is** *?rel*) **and**
‹*nat-of-uint64*
(*uint64-of-uint32*
(*a* ! (*j* − *POS-SHIFT*)) +
two-uint64) =
Suc (*Suc* (*xarena-pos*
(*arena* ! (*j* − *POS-SHIFT*))))› (**is** *?eq′*)
⟨*proof*⟩

lemma *isa-get-saved-pos-get-saved-pos*:
‹(*uncurry isa-get-saved-pos*, *uncurry* (*RETURN oo arena-pos*)) ∈
$[$*uncurry get-saved-pos-pre*$]_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨*uint64-nat-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *isa-get-saved-pos′* :: ‹*uint32 list* ⇒ *nat* ⇒ *nat nres*› **where**
‹*isa-get-saved-pos′ arena C* = *do* {
$\quad$ *pos* ← *isa-get-saved-pos arena C*;
$\quad$ *RETURN* (*nat-of-uint64 pos*)
}›

lemma *isa-get-saved-pos-get-saved-pos′*:
‹(*uncurry isa-get-saved-pos′*, *uncurry* (*RETURN oo arena-pos*)) ∈
$[$*uncurry get-saved-pos-pre*$]_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

Update Saved Position **definition** *isa-update-pos* :: ‹*nat* ⇒ *nat* ⇒ *uint32 list* ⇒ *uint32 list nres*› **where**
‹*isa-update-pos C n arena* = *do* {
$\quad$ *ASSERT*(*C* − *POS-SHIFT* < *length arena* ∧ *C* ≥ *POS-SHIFT* ∧ *n* ≥ *2* ∧ *n* − *2* ≤ *uint32-max*);
$\quad$ *RETURN* (*arena* [*C* − *POS-SHIFT* := (*uint32-of-nat* (*n* − *2*))])
}›

definition *arena-update-pos* **where**
‹*arena-update-pos C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos* (*pos* − *2*)]›

lemma *arena-update-pos-alt-def*:
‹*arena-update-pos C i N = update-pos-direct C* (*i* − *2*) *N*›
⟨*proof*⟩

lemma *arena-update-pos-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* ∈# *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› **and**
length: ‹*arena-length arena j* > *MAX-LENGTH-SHORT-CLAUSE*› **and**
pos-le: ‹*pos* ≤ *arena-length arena j*› **and**
b': ‹*pos* ≥ *2*›
shows
‹*j* − *POS-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*j* ≥ *POS-SHIFT*› (**is** *?ge*)
‹(*a*[*j* − *POS-SHIFT* := *uint32-of-nat* (*pos* − *2*)], *arena-update-pos j pos arena*) ∈
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› (**is** *?unat*) **and**
‹*pos* − *2* ≤ *uint-max*›
⟨*proof*⟩

definition *isa-update-pos-pre* **where**
‹*isa-update-pos-pre* = (λ((*C*, *lbd*), *arena*). *arena-is-valid-clause-idx arena C* ∧ *lbd* ≥ *2* ∧
lbd ≤ *arena-length arena C* ∧ *arena-length arena C* > *MAX-LENGTH-SHORT-CLAUSE* ∧
lbd ≥ *2*)›

lemma *isa-update-pos*:
‹(*uncurry2 isa-update-pos*, *uncurry2* (*RETURN ooo arena-update-pos*)) ∈
[*isa-update-pos-pre*]$_f$
nat-rel $\times_f$ *nat-rel* $\times_f$ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* →
⟨⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

Mark clause as garbage **definition** *mark-garbage-pre* **where**
‹*mark-garbage-pre* = (λ(*arena*, *C*). *arena-is-valid-clause-idx arena C*)›

definition *mark-garbage* **where**
‹*mark-garbage arena C* = *do* {
ASSERT(*C* ≥ *STATUS-SHIFT* ∧ *C* − *STATUS-SHIFT* < *length arena*);
RETURN (*arena*[*C* − *STATUS-SHIFT* := (*3* :: *uint32*)])
}›

lemma *mark-garbage-pre*:
assumes
j: ‹*j* ∈# *dom-m N*› **and**
valid: ‹*valid-arena arena N x*› **and**
arena: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows
‹*STATUS-SHIFT* ≤ *j*› (**is** *?ge*) **and**
‹(*a*[*j* − *STATUS-SHIFT* := *3*], *arena*[*j* − *STATUS-SHIFT* := *AStatus DELETED False*])
∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*› (**is** *?rel*) **and**
‹*j* − *STATUS-SHIFT* < *length arena*› (**is** *?le*)
⟨*proof*⟩

lemma *isa-mark-garbage*:
‹(*uncurry mark-garbage*, *uncurry* (*RETURN oo extra-information-mark-to-delete*)) ∈

$[mark\text{-}garbage\text{-}pre]_f$
$\langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \rightarrow$
$\langle\langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel\rangle nres\text{-}rel\rangle$
$\langle proof\rangle$

Activity **definition** *arena-act-pre* **where**
$\langle arena\text{-}act\text{-}pre = arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\rangle$

definition *isa-arena-act* :: $\langle uint32\ list \Rightarrow nat \Rightarrow uint32\ nres\rangle$ **where**
$\langle isa\text{-}arena\text{-}act\ arena\ C = do\ \{$
$ASSERT(C - ACTIVITY\text{-}SHIFT < length\ arena \wedge C \geq ACTIVITY\text{-}SHIFT);$
$RETURN\ (arena\ !\ (C - ACTIVITY\text{-}SHIFT))$
$\}\rangle$

lemma *arena-act-conv*:
assumes
valid: $\langle valid\text{-}arena\ arena\ N\ x\rangle$ **and**
j: $\langle j \in\#\ dom\text{-}m\ N\rangle$ **and**
a: $\langle(a,\ arena) \in \langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel\rangle$
shows
$\langle j - ACTIVITY\text{-}SHIFT < length\ arena\rangle$ (**is** *?le*) **and**
$\langle ACTIVITY\text{-}SHIFT \leq j\rangle$ (**is** *?ge*) **and**
$\langle(a\ !\ (j - ACTIVITY\text{-}SHIFT),$
$xarena\text{-}act\ (arena\ !\ (j - ACTIVITY\text{-}SHIFT)))$
$\in uint32\text{-}nat\text{-}rel\rangle$
$\langle proof\rangle$

lemma *isa-arena-act-arena-act*:
$\langle(uncurry\ isa\text{-}arena\text{-}act,\ uncurry\ (RETURN\ oo\ arena\text{-}act)) \in$
$[uncurry\ arena\text{-}act\text{-}pre]_f$
$\langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \rightarrow$
$\langle uint32\text{-}nat\text{-}rel\rangle nres\text{-}rel\rangle$
$\langle proof\rangle$

Increment Activity **definition** *isa-arena-incr-act* :: $\langle uint32\ list \Rightarrow nat \Rightarrow uint32\ list\ nres\rangle$ **where**
$\langle isa\text{-}arena\text{-}incr\text{-}act\ arena\ C = do\ \{$
$ASSERT(C - ACTIVITY\text{-}SHIFT < length\ arena \wedge C \geq ACTIVITY\text{-}SHIFT);$
$let\ act = arena\ !\ (C - ACTIVITY\text{-}SHIFT);$
$RETURN\ (arena[C - ACTIVITY\text{-}SHIFT := act + 1])$
$\}\rangle$

definition *arena-incr-act* **where**
$\langle arena\text{-}incr\text{-}act\ arena\ i = arena[i - ACTIVITY\text{-}SHIFT := AActivity\ (sum\text{-}mod\text{-}uint32\text{-}max\ 1\ (xarena\text{-}act\ (arena!(i - ACTIVITY\text{-}SHIFT))))]\rangle$

lemma *arena-incr-act-conv*:
assumes
valid: $\langle valid\text{-}arena\ arena\ N\ x\rangle$ **and**
j: $\langle j \in\#\ dom\text{-}m\ N\rangle$ **and**
a: $\langle(a,\ arena) \in \langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel\rangle$
shows
$\langle j - ACTIVITY\text{-}SHIFT < length\ arena\rangle$ (**is** *?le*) **and**
$\langle ACTIVITY\text{-}SHIFT \leq j\rangle$ (**is** *?ge*) **and**
$\langle(a[j - ACTIVITY\text{-}SHIFT := a\ !\ (j - ACTIVITY\text{-}SHIFT) + 1],\ arena\text{-}incr\text{-}act\ arena\ j) \in \langle uint32\text{-}nat\text{-}rel\ O\ arena\text{-}el\text{-}rel\rangle list\text{-}rel\rangle$

⟨*proof*⟩

lemma *isa-arena-incr-act-arena-incr-act*:
‹(*uncurry isa-arena-incr-act*, *uncurry* (*RETURN oo arena-incr-act*)) ∈
[*uncurry arena-act-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *length-clause-slice-list-update*[*simp*]:
‹*length* (*clause-slice* (*arena*[*i* := *x*]) *a b*) = *length* (*clause-slice arena a b*)›
⟨*proof*⟩

lemma *length-arena-incr-act*[*simp*]:
‹*length* (*arena-incr-act arena C*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-arena-incr-act*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*arena-incr-act arena C*) *N vdom*›
⟨*proof*⟩

Divide activity by two **definition** *isa-arena-decr-act* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint32 list nres*›
where
‹*isa-arena-decr-act arena C* = *do* {
ASSERT(*C* − *ACTIVITY-SHIFT* < *length arena* ∧ *C* ≥ *ACTIVITY-SHIFT*);
let act = *arena* ! (*C* − *ACTIVITY-SHIFT*);
RETURN (*arena*[*C* − *ACTIVITY-SHIFT* := (*act* >> *1*)])
}›

definition *arena-decr-act* **where**
‹*arena-decr-act arena i* = *arena*[*i* − *ACTIVITY-SHIFT* :=
AActivity (*xarena-act* (*arena*!(*i* − *ACTIVITY-SHIFT*)) *div 2*)]›

lemma *arena-decr-act-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* ∈# *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows
‹*j* − *ACTIVITY-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*ACTIVITY-SHIFT* ≤ *j*› (**is** *?ge*) **and**
‹(*a*[*j* − *ACTIVITY-SHIFT* := *a* ! (*j* − *ACTIVITY-SHIFT*) >> *Suc 0*], *arena-decr-act arena j*)
∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
⟨*proof*⟩

lemma *isa-arena-decr-act-arena-decr-act*:
‹(*uncurry isa-arena-decr-act*, *uncurry* (*RETURN oo arena-decr-act*)) ∈
[*uncurry arena-act-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *length-arena-decr-act*[*simp*]:
‹*length* (*arena-decr-act arena C*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-arena-decr-act*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*arena-decr-act arena C*) *N vdom*›
⟨*proof*⟩

Mark used **definition** *isa-mark-used* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint32 list nres*› **where**
‹*isa-mark-used arena C* = *do* {
  ASSERT(*C* − *STATUS-SHIFT* < *length arena* ∧ *C* ≥ *STATUS-SHIFT*);
  let act = *arena* ! (*C* − *STATUS-SHIFT*);
  RETURN (*arena*[*C* − *STATUS-SHIFT* := *act OR 0b100*])
}›

definition *mark-used* **where**
‹*mark-used arena i* =
  arena[*i* − *STATUS-SHIFT* := *AStatus* (*xarena-status* (*arena*!(*i* − *STATUS-SHIFT*))) *True*]›

lemma *isa-mark-used-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* ∈# *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows
‹*j* − *STATUS-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*STATUS-SHIFT* ≤ *j*› (**is** *?ge*) **and**
‹(*a*[*j* − *STATUS-SHIFT* := *a* ! (*j* − *STATUS-SHIFT*) *OR 4*], *mark-used arena j*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
⟨*proof*⟩

lemma *isa-mark-used-mark-used*:
‹(*uncurry isa-mark-used*, *uncurry* (*RETURN oo mark-used*)) ∈
  [*uncurry arena-act-pre*]$_f$
  ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
  ⟨ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *length-mark-used*[*simp*]: ‹*length* (*mark-used arena C*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-mark-used*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*mark-used arena C*) *N vdom*›
⟨*proof*⟩

Mark unused **definition** *isa-mark-unused* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint32 list nres*› **where**
‹*isa-mark-unused arena C* = *do* {
  ASSERT(*C* − *STATUS-SHIFT* < *length arena* ∧ *C* ≥ *STATUS-SHIFT*);
  let act = *arena* ! (*C* − *STATUS-SHIFT*);
  RETURN (*arena*[*C* − *STATUS-SHIFT* := *act AND 0b11*])

}›

definition *mark-unused* **where**
‹*mark-unused arena i* =
arena[$i - STATUS\text{-}SHIFT$:= *AStatus* (*xarena-status* (*arena*!($i - STATUS\text{-}SHIFT$))) *False*]›

lemma *isa-mark-unused-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹$j \in\#$ *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows
‹$j - STATUS\text{-}SHIFT <$ *length arena*› (**is** *?le*) **and**
‹$STATUS\text{-}SHIFT \le j$› (**is** *?ge*) **and**
‹(*a*[$j - STATUS\text{-}SHIFT$:= *a* ! ($j - STATUS\text{-}SHIFT$) *AND 3*], *mark-unused arena j*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
⟨*proof*⟩

lemma *isa-mark-unused-mark-unused*:
‹(*uncurry isa-mark-unused*, *uncurry* (*RETURN oo mark-unused*)) ∈
[*uncurry arena-act-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *length-mark-unused*[*simp*]: ‹*length* (*mark-unused arena C*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-mark-unused*:
assumes *C*: ‹$C \in\#$ *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*mark-unused arena C*) *N vdom*›
⟨*proof*⟩

Marked as used?

definition *marked-as-used* :: ‹*arena* ⇒ *nat* ⇒ *bool*› **where**
‹*marked-as-used arena C* = *xarena-used* (*arena* ! ($C - STATUS\text{-}SHIFT$))›

definition *marked-as-used-pre* **where**
‹*marked-as-used-pre* = *arena-is-valid-clause-idx*›

definition *isa-marked-as-used* :: ‹*uint32 list* ⇒ *nat* ⇒ *bool nres*› **where**
‹*isa-marked-as-used arena C* = *do* {
ASSERT($C - STATUS\text{-}SHIFT <$ *length arena* $\wedge C \ge STATUS\text{-}SHIFT$);
RETURN (*arena* ! ($C - STATUS\text{-}SHIFT$) *AND 4* ≠ *0*)
}›

lemma *arena-marked-as-used-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹$j \in\#$ *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows

⟨$j - STATUS\text{-}SHIFT < length\ arena$⟩ (**is** *?le*) **and**
⟨$STATUS\text{-}SHIFT \le j$⟩ (**is** *?ge*) **and**
⟨$a\ !\ (j - STATUS\text{-}SHIFT)\ AND\ 4 \ne 0 \longleftrightarrow$
marked-as-used arena j⟩
⟨*proof*⟩

lemma *isa-marked-as-used-marked-as-used*:
⟨(*uncurry isa-marked-as-used*, *uncurry* (*RETURN oo marked-as-used*)) $\in$
[*uncurry marked-as-used-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* $\to$
⟨*bool-rel*⟩*nres-rel*⟩
⟨*proof*⟩

lemma *valid-arena-vdom-le*:
assumes ⟨*valid-arena arena N ovdm*⟩
shows ⟨*finite ovdm*⟩ **and** ⟨*card ovdm* $\le$ *length arena*⟩
⟨*proof*⟩

lemma *valid-arena-vdom-subset*:
assumes ⟨*valid-arena arena N* (*set vdom*)⟩ **and** ⟨*distinct vdom*⟩
shows ⟨*length vdom* $\le$ *length arena*⟩
⟨*proof*⟩

end
theory *IsaSAT-Literals-SML*
imports *Watched-Literals.WB-Word-Assn*
Watched-Literals.Array-UInt IsaSAT-Literals
begin

sepref-decl-op *atm-of*: ⟨*atm-of* :: *nat literal* $\Rightarrow$ *nat*⟩ ::
⟨(*Id* :: (*nat literal* $\times$ -) *set*) $\to$ (*Id* :: (*nat* $\times$ -) *set*)⟩ ⟨*proof*⟩

lemma [*def-pat-rules*]:
⟨*atm-of* $\equiv$ *op-atm-of*⟩
⟨*proof*⟩

sepref-decl-op *lit-of*: ⟨*lit-of* :: (*nat*, *nat*) *ann-lit* $\Rightarrow$ *nat literal*⟩ ::
⟨(*Id* :: ((*nat*, *nat*) *ann-lit* $\times$ -) *set*) $\to$ (*Id* :: (*nat literal* $\times$ -) *set*)⟩ ⟨*proof*⟩

lemma [*def-pat-rules*]:
⟨*lit-of* $\equiv$ *op-lit-of*⟩
⟨*proof*⟩

sepref-decl-op *watched-app*:
⟨*watched-app* ::(*nat literal* $\Rightarrow$ (*nat* $\times$ -) *list*) $\Rightarrow$ *nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat watcher*⟩
::
⟨(*Id* :: ((*nat literal* $\Rightarrow$ (*nat watcher*) *list*) $\times$ -) *set*) $\to$ (*Id* :: (*nat literal* $\times$ -) *set*) $\to$ *nat-rel* $\to$
nat-rel $\times_r$ (*Id* :: (*nat literal* $\times$ -) *set*) $\times_r$ *bool-rel*⟩
⟨*proof*⟩

lemma (**in** $-$) *safe-minus-nat-assn*:
⟨(*uncurry* (*return oo* ($-$)), *uncurry* (*RETURN oo fast-minus*)) $\in$
$[\lambda(m, n).\ m \ge n]_a$ *nat-assn*$^k *_a$ *nat-assn*$^k \to$ *nat-assn*⟩

⟨proof⟩

definition *map-fun-rel-assn*
:: ‹$(nat \times nat\ literal)\ set \Rightarrow ('a \Rightarrow 'b \Rightarrow assn) \Rightarrow (nat\ literal \Rightarrow 'a) \Rightarrow 'b\ list \Rightarrow assn$›
where
‹$map\text{-}fun\text{-}rel\text{-}assn\ D\ R = pure\ (\langle the\text{-}pure\ R\rangle map\text{-}fun\text{-}rel\ D)$›

lemma [*safe-constraint-rules*]: ‹*is-pure* (*map-fun-rel-assn D R*)›
⟨proof⟩
abbreviation *nat-lit-assn* :: ‹$nat\ literal \Rightarrow nat \Rightarrow assn$› **where**
‹$nat\text{-}lit\text{-}assn \equiv pure\ nat\text{-}lit\text{-}rel$›

abbreviation *unat-lit-assn* :: ‹$nat\ literal \Rightarrow uint32 \Rightarrow assn$› **where**
‹$unat\text{-}lit\text{-}assn \equiv pure\ unat\text{-}lit\text{-}rel$›

lemma *hr-comp-uint32-nat-assn-nat-lit-rel*[*simp*]:
‹$hr\text{-}comp\ uint32\text{-}nat\text{-}assn\ nat\text{-}lit\text{-}rel = unat\text{-}lit\text{-}assn$›
⟨proof⟩

abbreviation *pair-nat-ann-lit-assn* :: ‹$(nat, nat)\ ann\text{-}lit \Rightarrow ann\text{-}lit\text{-}wl \Rightarrow assn$› **where**
‹$pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn \equiv pure\ nat\text{-}ann\text{-}lit\text{-}rel$›

abbreviation *pair-nat-ann-lits-assn* :: ‹$(nat, nat)\ ann\text{-}lits \Rightarrow ann\text{-}lits\text{-}wl \Rightarrow assn$› **where**
‹$pair\text{-}nat\text{-}ann\text{-}lits\text{-}assn \equiv list\text{-}assn\ pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn$›

abbreviation *pair-nat-ann-lit-fast-assn* :: ‹$(nat, nat)\ ann\text{-}lit \Rightarrow ann\text{-}lit\text{-}wl\text{-}fast \Rightarrow assn$› **where**
‹$pair\text{-}nat\text{-}ann\text{-}lit\text{-}fast\text{-}assn \equiv hr\text{-}comp\ (uint32\text{-}assn *a\ option\text{-}assn\ uint64\text{-}nat\text{-}assn)\ nat\text{-}ann\text{-}lit\text{-}rel$›

abbreviation *pair-nat-ann-lits-fast-assn* :: ‹$(nat, nat)\ ann\text{-}lits \Rightarrow ann\text{-}lits\text{-}wl\text{-}fast \Rightarrow assn$› **where**
‹$pair\text{-}nat\text{-}ann\text{-}lits\text{-}fast\text{-}assn \equiv list\text{-}assn\ pair\text{-}nat\text{-}ann\text{-}lit\text{-}fast\text{-}assn$›

Code

lemma [*sepref-fr-rules*]: ‹$(return\ o\ id, RETURN\ o\ nat\text{-}of\text{-}lit) \in unat\text{-}lit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨proof⟩

lemma ‹$(return\ o\ (\lambda n.\ shiftr\ n\ 1), RETURN\ o\ shiftr1) \in word\text{-}nat\text{-}assn^k \rightarrow_a word\text{-}nat\text{-}assn$›
⟨proof⟩

lemma *propagated-hnr*[*sepref-fr-rules*]:
‹$(uncurry\ (return\ oo\ propagated), uncurry\ (RETURN\ oo\ Propagated)) \in$
$unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn$›
⟨proof⟩

lemma *decided-hnr*[*sepref-fr-rules*]:
‹$(return\ o\ decided, RETURN\ o\ Decided) \in$
$unat\text{-}lit\text{-}assn^k \rightarrow_a pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn$›
⟨proof⟩

lemma *uminus-lit-hnr*[*sepref-fr-rules*]:
‹$(return\ o\ uminus\text{-}code, RETURN\ o\ uminus) \in$
$unat\text{-}lit\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
⟨proof⟩

abbreviation *ann-lit-wl-assn* :: ‹$ann\text{-}lit\text{-}wl \Rightarrow ann\text{-}lit\text{-}wl \Rightarrow assn$› **where**

⟨*ann-lit-wl-assn* ≡ *uint32-assn* $*a$ (*option-assn nat-assn*)⟩

abbreviation *ann-lit-wl-fast-assn* :: ⟨*ann-lit-wl* ⇒ *ann-lit-wl-fast* ⇒ *assn*⟩ **where**
⟨*ann-lit-wl-fast-assn* ≡ *uint32-assn* $*a$ (*option-assn uint64-nat-assn*)⟩

abbreviation *ann-lits-wl-assn* :: ⟨*ann-lits-wl* ⇒ *ann-lits-wl* ⇒ *assn*⟩ **where**
⟨*ann-lits-wl-assn* ≡ *list-assn ann-lit-wl-assn*⟩

type-synonym *clause-wl* = ⟨*uint32 array*⟩
abbreviation *clause-ll-assn* :: ⟨*nat clause-l* ⇒ *clause-wl* ⇒ *assn*⟩ **where**
⟨*clause-ll-assn* ≡ *array-assn unat-lit-assn*⟩

abbreviation *clause-l-assn* :: ⟨*nat clause* ⇒ *uint32 list* ⇒ *assn*⟩ **where**
⟨*clause-l-assn* ≡ *list-mset-assn unat-lit-assn*⟩

abbreviation *clauses-l-assn* :: ⟨*nat clauses* ⇒ *uint32 list list* ⇒ *assn*⟩ **where**
⟨*clauses-l-assn* ≡ *list-mset-assn clause-l-assn*⟩

abbreviation *clauses-to-update-l-assn* :: ⟨*nat multiset* ⇒ *nat list* ⇒ *assn*⟩ **where**
⟨*clauses-to-update-l-assn* ≡ *list-mset-assn nat-assn*⟩

abbreviation *clauses-to-update-ll-assn* :: ⟨*nat list* ⇒ *nat list* ⇒ *assn*⟩ **where**
⟨*clauses-to-update-ll-assn* ≡ *list-assn nat-assn*⟩

type-synonym *unit-lits-wl* = ⟨*uint32 list list*⟩
abbreviation *unit-lits-assn* :: ⟨*nat clauses* ⇒ *unit-lits-wl* ⇒ *assn*⟩ **where**
⟨*unit-lits-assn* ≡ *list-mset-assn* (*list-mset-assn unat-lit-assn*)⟩

lemma *atm-of-hnr*[*sepref-fr-rules*]:
⟨(*return o atm-of-code*, *RETURN o op-atm-of*) ∈ $\textit{unat-lit-assn}^k \rightarrow_a$ *uint32-nat-assn*⟩
⟨*proof*⟩

lemma *lit-of-hnr*[*sepref-fr-rules*]:
⟨(*return o fst*, *RETURN o op-lit-of*) ∈ $\textit{pair-nat-ann-lit-assn}^k \rightarrow_a$ *unat-lit-assn*⟩
⟨*proof*⟩

lemma *lit-of-fast-hnr*[*sepref-fr-rules*]:
⟨(*return o fst*, *RETURN o op-lit-of*) ∈ $\textit{pair-nat-ann-lit-fast-assn}^k \rightarrow_a$ *unat-lit-assn*⟩
⟨*proof*⟩

lemma *op-eq-op-nat-lit-eq*[*sepref-fr-rules*]:
⟨(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) ∈
$(\textit{pure unat-lit-rel})^k *_a (\textit{pure unat-lit-rel})^k \rightarrow_a$ *bool-assn*⟩
⟨*proof*⟩

lemma (**in** −) *is-pos-hnr*[*sepref-fr-rules*]:
⟨(*return o is-pos-code*, *RETURN o is-pos*) ∈ $\textit{unat-lit-assn}^k \rightarrow_a$ *bool-assn*⟩
⟨*proof*⟩

lemma *lit-and-ann-of-propagated-hnr*[*sepref-fr-rules*]:
⟨(*return o lit-and-ann-of-propagated-code*, *RETURN o lit-and-ann-of-propagated*) ∈
$[\lambda L.\ \neg \textit{is-decided}\ L]_a\ \textit{pair-nat-ann-lit-assn}^k \rightarrow$ (*unat-lit-assn* $*a$ *nat-assn*)⟩
⟨*proof*⟩

lemma *Pos-unat-lit-assn*:

‹(return o (λn. two-uint32 ∗ n), RETURN o Pos) ∈ [λL. Pos L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ isasat-input-bounded $\mathcal{A}$]$_a$ uint32-nat-assn^k →
unat-lit-assn›
⟨proof⟩

lemma *Neg-unat-lit-assn*:
‹(return o (λn. two-uint32 ∗ n +1), RETURN o Neg) ∈ [λL. Pos L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ isasat-input-bounded $\mathcal{A}$]$_a$ uint32-nat-assn^k →
unat-lit-assn›
⟨proof⟩

lemma *Pos-unat-lit-assn′*:
‹(return o (λn. two-uint32 ∗ n), RETURN o Pos) ∈ [λL. L ≤ uint-max div 2]$_a$ uint32-nat-assn^k →
unat-lit-assn›
⟨proof⟩

lemma *Neg-unat-lit-assn′*:
‹(return o (λn. two-uint32 ∗ n + 1), RETURN o Neg) ∈ [λL. L ≤ uint-max div 2]$_a$ uint32-nat-assn^k
→
unat-lit-assn›
⟨proof⟩

0.1.5 Declaration of some Operators and Implementation

sepref-register ‹*watched-by* :: *nat twl-st-wl* ⇒ *nat literal* ⇒ *nat watched*›
:: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat watched*›

lemma [*def-pat-rules*]:
‹watched-app $ M $ L $ i ≡ op-watched-app $ M $ L $ i›
⟨proof⟩

sepref-definition *is-decided-wl-code*
is ‹(RETURN o is-decided-wl)›
:: ‹ann-lit-wl-assn^k →$_a$ bool-assn›
⟨proof⟩

sepref-definition *is-decided-wl-fast-code*
is ‹(RETURN o is-decided-wl)›
:: ‹ann-lit-wl-fast-assn^k →$_a$ bool-assn›
⟨proof⟩

lemma
is-decided-wl-code[*sepref-fr-rules*]:
‹(is-decided-wl-code, RETURN o is-decided) ∈ pair-nat-ann-lit-assn^k →$_a$ bool-assn› (**is** *?slow*) **and**
is-decided-wl-fast-code[*sepref-fr-rules*]:
‹(is-decided-wl-fast-code, RETURN o is-decided) ∈ pair-nat-ann-lit-fast-assn^k →$_a$ bool-assn›
(**is** *?fast*)
⟨proof⟩

end
theory *IsaSAT-Arena-SML*
imports *IsaSAT-Arena IsaSAT-Literals-SML Watched-Literals.IICF-Array-List64*
begin

abbreviation *arena-el-assn* :: *arena-el* ⇒ *uint32* ⇒ *assn* **where**
⟨*arena-el-assn* ≡ *hr-comp uint32-nat-assn arena-el-rel*⟩

abbreviation *arena-assn* :: *arena-el list* ⇒ *uint32 array-list* ⇒ *assn* **where**
⟨*arena-assn* ≡ *arl-assn arena-el-assn*⟩

abbreviation *arena-fast-assn* :: *arena-el list* ⇒ *uint32 array-list64* ⇒ *assn* **where**
⟨*arena-fast-assn* ≡ *arl64-assn arena-el-assn*⟩

abbreviation *status-assn* **where**
⟨*status-assn* ≡ *hr-comp uint32-nat-assn status-rel*⟩

abbreviation *clause-status-assn* **where**
⟨*clause-status-assn* ≡ (*id-assn* :: *clause-status* ⇒ -)⟩

lemma *IRRED-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return IRRED*), *uncurry0* (*RETURN IRRED*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *clause-status-assn*⟩
⟨*proof*⟩

lemma *LEARNED-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return LEARNED*), *uncurry0* (*RETURN LEARNED*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *clause-status-assn*⟩
⟨*proof*⟩

lemma *DELETED-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return DELETED*), *uncurry0* (*RETURN DELETED*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *clause-status-assn*⟩
⟨*proof*⟩

lemma *ACTIVITY-SHIFT-hnr*:
⟨(*uncurry0* (*return 3*), *uncurry0* (*RETURN ACTIVITY-SHIFT*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *uint64-nat-assn*⟩
⟨*proof*⟩
lemma *STATUS-SHIFT-hnr*:
⟨(*uncurry0* (*return 4*), *uncurry0* (*RETURN STATUS-SHIFT*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *uint64-nat-assn*⟩
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
⟨(*uncurry0* (*return 1*), *uncurry0* (*RETURN SIZE-SHIFT*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *nat-assn*⟩
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
⟨(*return o id*, *RETURN o xarena-length*) ∈ $[\mathit{is\text{-}Size}]_a\ \mathit{arena\text{-}el\text{-}assn}^k \rightarrow$ *uint32-nat-assn*⟩
⟨*proof*⟩

lemma (**in** −) *POS-SHIFT-uint64-hnr*:
⟨(*uncurry0* (*return 5*), *uncurry0* (*RETURN POS-SHIFT*)) ∈ $\mathit{unit\text{-}assn}^k \rightarrow_a$ *uint64-nat-assn*⟩
⟨*proof*⟩

lemma *nat-of-uint64-eq-2-iff*[*simp*]: ⟨*nat-of-uint64 c* = *2* ⟷ *c* = *2*⟩
⟨*proof*⟩

lemma *arena-el-assn-alt-def*:
⟨*arena-el-assn* = *hr-comp uint32-assn* (*uint32-nat-rel O arena-el-rel*)⟩
⟨*proof*⟩

lemma *arena-el-comp*: ⟨*hn-val* (*uint32-nat-rel O arena-el-rel*) = *hn-ctxt arena-el-assn*⟩
⟨*proof*⟩

lemma *status-assn-hnr-eq*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) ∈ $status\text{-}assn^k *_a status\text{-}assn^k \rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *IRRED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 0*), *uncurry0* (*RETURN IRRED*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *status-assn*›
⟨*proof*⟩

lemma *LEARNED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN LEARNED*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *status-assn*›
⟨*proof*⟩

lemma *DELETED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 3*), *uncurry0* (*RETURN DELETED*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *status-assn*›
⟨*proof*⟩

lemma *status-assn-alt-def*:
‹*status-assn* = *pure* (*uint32-nat-rel O status-rel*)›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 2*), *uncurry0* (*RETURN LBD-SHIFT*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *nat-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN STATUS-SHIFT*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *nat-assn*›
⟨*proof*⟩

lemma (**in** −) *LBD-SHIFT-hnr*:
‹(*uncurry0* (*return 2*), *uncurry0* (*RETURN LBD-SHIFT*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *MAX-LENGTH-SHORT-CLAUSE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN MAX-LENGTH-SHORT-CLAUSE*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

definition *four-uint32* **where** ‹*four-uint32* = (*4* :: *uint32*)›

lemma *four-uint32-hnr*:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN* (*four-uint32* :: *uint32*))) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 5*), *uncurry0* (*RETURN POS-SHIFT*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *nat-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*return o id*, *RETURN o xarena-lit*) ∈ $[is\text{-}Lit]_a\ arena\text{-}el\text{-}assn^k \rightarrow$ *unat-lit-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-length-code*
is ‹*uncurry isa-arena-length*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

lemma *isa-arena-length-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-code*, *uncurry* (*RETURN* ∘∘ *arena-length*))
∈ [*uncurry arena-is-valid-clause-idx*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} → *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-length-fast-code*
is ‹*uncurry isa-arena-length*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

lemma *isa-arena-length-fast-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-length*))
∈ [*uncurry arena-is-valid-clause-idx*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} → *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-length-fast-code2*
is ‹*uncurry isa-arena-length*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

lemma *isa-arena-length-fast-code2-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-length*))
∈ [*uncurry arena-is-valid-clause-idx*]$_a$
arena-assn^k $*_a$ *uint64-nat-assn*^{k} → *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-lit-code*
is ‹*uncurry isa-arena-lit*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

lemma *isa-arena-lit-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-lit-code*, *uncurry* (*RETURN* ∘∘ *arena-lit*))
∈ [*uncurry arena-lit-pre*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-definition (**in**−) *isa-arena-lit-fast-code*
is ‹*uncurry isa-arena-lit*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

declare *isa-arena-lit-fast-code.refine*

lemma *isa-arena-lit-fast-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-lit-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-lit*))
∈ [*uncurry arena-lit-pre*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-definition (**in**−) *isa-arena-lit-fast-code2*
is ‹*uncurry isa-arena-lit*›

:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

declare *isa-arena-lit-fast-code2.refine*

lemma *isa-arena-lit-fast-code2-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-lit-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-lit*))
$\in [uncurry\ arena\text{-}lit\text{-}pre]_a$
$arena\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-definition *arena-status-code*
is ‹*uncurry isa-arena-status*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

lemma *isa-arena-status-refine*[*sepref-fr-rules*]:
‹(*uncurry arena-status-code*, *uncurry* (*RETURN* ∘∘ *arena-status*))
$\in [uncurry\ arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom]_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow status\text{-}assn$›
⟨*proof*⟩

sepref-definition *swap-lits-code*
is ‹*Sepref-Misc.uncurry3 isa-arena-swap*›
:: ‹$nat\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a (arl\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *swap-lits-refine*[*sepref-fr-rules*]:
‹(*uncurry3 swap-lits-code*, *uncurry3* (*RETURN oooo swap-lits*))
$\in [uncurry3\ swap\text{-}lits\text{-}pre]_a\ nat\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a arena\text{-}assn^d \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-update-lbd-code*
is ‹*uncurry2 isa-update-lbd*›
:: ‹$nat\text{-}assn^k *_a uint32\text{-}assn^k *_a (arl\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *update-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-lbd-code*, *uncurry2* (*RETURN ooo update-lbd*))
$\in [update\text{-}lbd\text{-}pre]_a\ nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a arena\text{-}assn^d \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition (**in** −)*isa-update-lbd-fast-code*
is ‹*uncurry2 isa-update-lbd*›
:: ‹$uint64\text{-}nat\text{-}assn^k *_a uint32\text{-}assn^k *_a (arl64\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl64\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *update-lbd-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-lbd-fast-code*, *uncurry2* (*RETURN ooo update-lbd*))
$\in [update\text{-}lbd\text{-}pre]_a\ uint64\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

sepref-definition (**in** −)*isa-update-lbd-fast-code2*

is ‹*uncurry2 isa-update-lbd*›
:: ‹$uint64\text{-}nat\text{-}assn^k *_a uint32\text{-}assn^k *_a (arl\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *update-lbd-fast-hnr2*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-lbd-fast-code2*, *uncurry2* (*RETURN ooo update-lbd*))
$\in [update\text{-}lbd\text{-}pre]_a\ uint64\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a arena\text{-}assn^d \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-get-clause-LBD-code*
is ‹*uncurry isa-get-clause-LBD*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

lemma *isa-get-clause-LBD-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-clause-LBD-code*, *uncurry* (*RETURN* ∘∘ *get-clause-LBD*))
$\in [uncurry\ get\text{-}clause\text{-}LBD\text{-}pre]_a\ arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-get-saved-pos-fast-code*
is ‹*uncurry isa-get-saved-pos*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

lemma *get-saved-pos-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
$\in [uncurry\ get\text{-}saved\text{-}pos\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-get-saved-pos-code*
is ‹*uncurry isa-get-saved-pos*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

lemma *get-saved-pos-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-code*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
$\in [uncurry\ get\text{-}saved\text{-}pos\text{-}pre]_a\ arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-get-saved-pos-code'*
is ‹*uncurry isa-get-saved-pos'*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a nat\text{-}assn$›
⟨*proof*⟩

lemma *get-saved-pos-code'*:
‹(*uncurry isa-get-saved-pos-code'*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
$\in [uncurry\ get\text{-}saved\text{-}pos\text{-}pre]_a\ arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-get-saved-pos-fast-code2*
is ‹*uncurry isa-get-saved-pos*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

lemma *get-saved-pos-code2*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-pos*))

$\in$ [*uncurry get-saved-pos-pre*]$_a$ *arena-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *isa-update-pos-code*
is ‹*uncurry2 isa-update-pos*›
:: ‹*nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ (*arl-assn uint32-assn*)^d $\rightarrow_a$ *arl-assn uint32-assn*›
⟨*proof*⟩

lemma *isa-update-pos-code-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-pos-code*, *uncurry2* (*RETURN ooo arena-update-pos*))
$\in$ [*isa-update-pos-pre*]$_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *arena-assn*^{d} $\rightarrow$ *arena-assn*›
⟨*proof*⟩

sepref-definition *mark-garbage-code*
is ‹*uncurry mark-garbage*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ *arl-assn uint32-assn*›
⟨*proof*⟩

lemma *mark-garbage-hnr*[*sepref-fr-rules*]:
‹(*uncurry mark-garbage-code*, *uncurry* (*RETURN oo extra-information-mark-to-delete*))
$\in$ [*mark-garbage-pre*]$_a$ *arena-assn*^{d} $*_a$ *nat-assn*^{k} $\rightarrow$ *arena-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-act-code*
is ‹*uncurry isa-arena-act*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

lemma *isa-arena-act-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-act-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-act*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-incr-act-code*
is ‹*uncurry isa-arena-incr-act*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
⟨*proof*⟩

lemma *isa-arena-incr-act-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-incr-act-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-incr-act*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *nat-assn*^{k} $\rightarrow$ *arena-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-decr-act-code*
is ‹*uncurry isa-arena-decr-act*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
⟨*proof*⟩

lemma *isa-arena-decr-act-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-decr-act-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-decr-act*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *nat-assn*^{k} $\rightarrow$ *arena-assn*›
⟨*proof*⟩

sepref-definition *isa-arena-decr-act-fast-code*
is ‹*uncurry isa-arena-decr-act*›

:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a (arl64\text{-}assn\ uint32\text{-}assn)$›
⟨*proof*⟩

lemma *isa-arena-decr-act-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-decr-act-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-decr-act*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-mark-used-code*
is ‹*uncurry isa-mark-used*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^d *_a nat\text{-}assn^k \rightarrow_a (arl\text{-}assn\ uint32\text{-}assn)$›
⟨*proof*⟩

lemma *isa-mark-used-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-used-code*, *uncurry* (*RETURN* ∘∘ *mark-used*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}assn^d *_a nat\text{-}assn^k \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-mark-used-fast-code*
is ‹*uncurry isa-mark-used*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a (arl\text{-}assn\ uint32\text{-}assn)$›
⟨*proof*⟩

lemma *isa-mark-used-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-used-fast-code*, *uncurry* (*RETURN* ∘∘ *mark-used*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-mark-unused-code*
is ‹*uncurry isa-mark-unused*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^d *_a nat\text{-}assn^k \rightarrow_a (arl\text{-}assn\ uint32\text{-}assn)$›
⟨*proof*⟩

lemma *isa-mark-unused-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-unused-code*, *uncurry* (*RETURN* ∘∘ *mark-unused*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}assn^d *_a nat\text{-}assn^k \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-mark-unused-fast-code*
is ‹*uncurry isa-mark-unused*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a (arl\text{-}assn\ uint32\text{-}assn)$›
⟨*proof*⟩

lemma *isa-mark-unused-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-unused-fast-code*, *uncurry* (*RETURN* ∘∘ *mark-unused*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-marked-as-used-code*
is ‹*uncurry isa-marked-as-used*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

lemma *isa-marked-as-used-code*[*sepref-fr-rules*]:
‹(*uncurry isa-marked-as-used-code*, *uncurry* (*RETURN* ∘∘ *marked-as-used*))
$\in$ [*uncurry marked-as-used-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

sepref-definition (**in** −) *isa-arena-incr-act-fast-code*
is ‹*uncurry isa-arena-incr-act*›
:: ‹(*arl64-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl64-assn uint32-assn*)›
⟨*proof*⟩

lemma *isa-arena-incr-act-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-incr-act-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-incr-act*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *uint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

sepref-definition *arena-status-fast-code*
is ‹*uncurry isa-arena-status*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

lemma *isa-arena-status-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry arena-status-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-status*))
$\in$ [*uncurry arena-is-valid-clause-vdom*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} → *status-assn*›
⟨*proof*⟩

context
notes [*fcomp-norm-unfold*] = *arl64-assn-def*[*symmetric*] *arl64-assn-comp′*
notes [*intro*!] = *hfrefI hn-refineI*[*THEN hn-refine-preI*]
notes [*simp*] = *pure-def hn-ctxt-def invalid-assn-def*
begin

definition *arl64-get2* :: *′a*::*heap array-list64* ⇒ *nat* ⇒ *′a Heap* **where**
arl64-get2 ≡ λ(*a*,*n*) *i*. *Array.nth a i*
thm *arl64-get-hnr-aux*
lemma *arl64-get2-hnr-aux*: (*uncurry arl64-get2*,*uncurry* (*RETURN oo op-list-get*)) $\in$ [λ(*l*,*i*). *i*<*length l*]$_a$ (*is-array-list64*^{k} $*_a$ *nat-assn*^{k}) → *id-assn*
⟨*proof*⟩

sepref-decl-impl *arl64-get2*: *arl64-get2-hnr-aux* ⟨*proof*⟩
end

sepref-definition *arena-status-fast-code2*
is ‹*uncurry isa-arena-status*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
⟨*proof*⟩

lemma *isa-arena-status-fast-hnr2*[*sepref-fr-rules*]:
‹(*uncurry arena-status-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-status*))
$\in$ [*uncurry arena-is-valid-clause-vdom*]$_a$
arena-fast-assn^k $*_a$ *nat-assn*^{k} → *status-assn*›
⟨*proof*⟩

sepref-definition *isa-update-pos-fast-code*
is ‹*uncurry2 isa-update-pos*›
:: ‹$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a (arl64\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl64\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *isa-update-pos-code-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-pos-fast-code*, *uncurry2* (*RETURN ooo arena-update-pos*))
$\in [isa\text{-}update\text{-}pos\text{-}pre]_a\ uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

declare *isa-update-pos-fast-code.refine*[*sepref-fr-rules*]
arena-status-fast-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Clauses*
imports *IsaSAT-Arena*
begin

Representation of Clauses

named-theorems *isasat-codegen* ‹*lemmas that should be unfolded to generate* (*efficient*) *code*›

type-synonym *clause-annot* = ‹*clause-status* × *nat* × *nat*›

type-synonym *clause-annots* = ‹*clause-annot list*›

definition *list-fmap-rel* :: ‹- ⇒ (*arena* × *nat clauses-l*) *set*› **where**
‹*list-fmap-rel vdom* = {(*arena*, *N*). *valid-arena arena N vdom*}›

lemma *nth-clauses-l*:
‹(*uncurry2* (*RETURN ooo* (λ*N i j*. *arena-lit N* (*i*+*j*))),
uncurry2 (*RETURN ooo* (λ*N i j*. *N* ∝ *i* ! *j*)))
$\in [\lambda((N, i), j).\ i \in\#\ dom\text{-}m\ N \wedge j < length\ (N \propto i)]_f$
$list\text{-}fmap\text{-}rel\ vdom \times_f nat\text{-}rel \times_f nat\text{-}rel \rightarrow \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

abbreviation *clauses-l-fmat* **where**
‹*clauses-l-fmat* ≡ *list-fmap-rel*›

type-synonym *vdom* = ‹*nat set*›

definition *fmap-rll* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll l i j* = *l* ∝ *i* ! *j*›

definition *fmap-rll-u* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll-u* = *fmap-rll*›

definition *fmap-rll-u64* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll-u64* = *fmap-rll*›

definition *fmap-length-rll-u* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
‹*fmap-length-rll-u l i* = *length-uint32-nat* (*l* ∝ *i*)›

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll-u64* :: (*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
⟨*fmap-length-rll-u64 l i* = *length-uint32-nat* (*l* ∝ *i*)⟩

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll* :: (*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
[*simp*]: ⟨*fmap-length-rll l i* = *length* (*l* ∝ *i*)⟩

definition *fmap-swap-ll* **where**
[*simp*]: ⟨*fmap-swap-ll N i j f* = (*N*(*i* ↪ *swap* (*N* ∝ *i*) *j f*))⟩

From a performance point of view, appending several time a single element is less efficient than reserving a space that is large enough directly. However, in this case the list of clauses N is so large that there should not be any difference

definition *fm-add-new* **where**
⟨*fm-add-new b C N0* = *do* {
  let st = (*if b then AStatus IRRED False else AStatus LEARNED False*);
  let l = *length N0*;
  let s = *length C* − *2*;
  let N = (*if is-short-clause C then*
    (((*N0* @ [*st*]) @ [*AActivity zero-uint32-nat*]) @ [*ALBD s*]) @ [*ASize s*]
    else ((((*N0* @ [*APos zero-uint32-nat*]) @ [*st*]) @ [*AActivity zero-uint32-nat*]) @ [*ALBD s*]) @ [*ASize* (*s*)]);
  (*i*, *N*) ← $WHILE_T$ $^{\lambda(i,\ N).\ i\ <\ length\ C\ \longrightarrow\ length\ N\ <\ header\text{-}size\ C\ +\ length\ N0\ +\ length\ C}$
    (λ(*i*, *N*). *i* < *length C*)
    (λ(*i*, *N*). *do* {
      ASSERT(*i* < *length C*);
      RETURN (*i*+*one-uint64-nat*, *N* @ [*ALit* (*C* ! *i*)])
    })
    (*zero-uint64-nat*, *N*);
  RETURN (*N*, *l* + *header-size C*)
}⟩

lemma *header-size-Suc-def*:
⟨*header-size C* =
  (*if is-short-clause C then Suc* (*Suc* (*Suc* (*Suc 0*))) *else Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))⟩
⟨*proof*⟩

lemma *nth-append-clause*:
⟨*a* < *length C* ⟹ *append-clause b C N* ! (*length N* + *header-size C* + *a*) = *ALit* (*C* ! *a*)⟩
⟨*proof*⟩

lemma *fm-add-new-append-clause*:
⟨*fm-add-new b C N* ≤ *RETURN* (*append-clause b C N*, *length N* + *header-size C*)⟩
⟨*proof*⟩

definition *fm-add-new-at-position*
:: ⟨*bool* ⇒ *nat* ⇒ *'v clause-l* ⇒ *'v clauses-l* ⇒ *'v clauses-l*⟩
where
⟨*fm-add-new-at-position b i C N* = *fmupd i* (*C*, *b*) *N*⟩

definition *AStatus-IRRED* **where**
⟨*AStatus-IRRED* = *AStatus IRRED False*⟩

definition *AStatus-IRRED2* **where**
⟨*AStatus-IRRED2 = AStatus IRRED True*⟩

definition *AStatus-LEARNED* **where**
⟨*AStatus-LEARNED = AStatus LEARNED True*⟩

definition *AStatus-LEARNED2* **where**
⟨*AStatus-LEARNED2 = AStatus LEARNED False*⟩

definition (**in** −)*fm-add-new-fast* **where**
[*simp*]: ⟨*fm-add-new-fast = fm-add-new*⟩

lemma (**in** −)*append-and-length-code-fast*:
⟨*length ba* ≤ *Suc* (*Suc uint-max*) ⟹
  2 ≤ *length ba* ⟹
  length b ≤ *uint64-max* − (*uint-max* + *5*) ⟹
  (*aa*, *header-size ba*) ∈ *uint64-nat-rel* ⟹
  (*ab*, *length b*) ∈ *uint64-nat-rel* ⟹
  length b + *header-size ba* ≤ *uint64-max*⟩
⟨*proof*⟩

definition (**in** −)*four-uint64-nat* **where**
[*simp*]: ⟨*four-uint64-nat* = (*4* :: *nat*)⟩
definition (**in** −)*five-uint64-nat* **where**
[*simp*]: ⟨*five-uint64-nat* = (*5* :: *nat*)⟩

definition *append-and-length-fast-code-pre* **where**
⟨*append-and-length-fast-code-pre* ≡ λ((*b*, *C*), *N*). *length C* ≤ *uint32-max*+*2* ∧ *length C* ≥ *2* ∧
  length N + *length C* + *5* ≤ *uint64-max*⟩

lemma *fm-add-new-alt-def*:
⟨*fm-add-new b C N0* = *do* {
  let st = (*if b then AStatus-IRRED else AStatus-LEARNED2*);
  let l = *length-uint64-nat N0*;
  let s = *uint32-of-uint64-conv* (*length-uint64-nat C* − *two-uint64-nat*);
  let N =
   (*if is-short-clause C*
    then (((*N0* @ [*st*]) @ [*AActivity zero-uint32-nat*]) @ [*ALBD s*]) @
     [*ASize s*]
    else ((((*N0* @ [*APos zero-uint32-nat*]) @ [*st*]) @
      [*AActivity zero-uint32-nat*]) @
      [*ALBD s*]) @
     [*ASize s*]);
  (*i*, *N*) ←
   $WHILE_T$ λ(*i*, *N*). *i* < *length C* ⟶ *length N* < *header-size C* + *length N0* + *length C*
    (λ(*i*, *N*). *i* < *length-uint64-nat C*)
    (λ(*i*, *N*). *do* {
      - ← *ASSERT* (*i* < *length C*);
      RETURN (*i* + *one-uint64-nat*, *N* @ [*ALit* (*C* ! *i*)])
     })

$(zero\text{-}uint64\text{-}nat,\ N)$;
RETURN $(N,\ l + header\text{-}size\ C)$
}›
⟨*proof*⟩

definition *fmap-swap-ll-u64* **where**
[*simp*]: ‹*fmap-swap-ll-u64* = *fmap-swap-ll*›

lemma *slice-Suc-nth*:
‹$a < b \Longrightarrow a < length\ xs \Longrightarrow Suc\ a < b \Longrightarrow Misc.slice\ a\ b\ xs = xs\ !\ a\ \#\ Misc.slice\ (Suc\ a)\ b\ xs$›
⟨*proof*⟩

definition *fm-mv-clause-to-new-arena* **where**
‹*fm-mv-clause-to-new-arena C old-arena new-arena0* = *do* {
ASSERT(*arena-is-valid-clause-idx old-arena C*);
ASSERT($C \geq$ (*if nat-of-uint64-conv* (*arena-length old-arena C*) ≤ 4 *then 4 else 5*));
let st = $C -$ (*if nat-of-uint64-conv* (*arena-length old-arena C*) ≤ 4 *then 4 else 5*);
ASSERT($C +$ *nat-of-uint64-conv* (*arena-length old-arena C*) $\leq$ *length old-arena*);
let en = $C +$ *nat-of-uint64-conv* (*arena-length old-arena C*);
(*i, new-arena*) ←
$WHILE_T$ λ(*i, new-arena*). $i < en \longrightarrow$ *length new-arena* < *length new-arena0* + (*arena-length old-arena C*) + (*if nat-of-ui*
(λ(*i, new-arena*). $i < en$)
(λ(*i, new-arena*). *do* {
ASSERT ($i <$ *length old-arena* ∧ $i < en$);
RETURN ($i + 1$, *new-arena* @ [*old-arena* ! *i*])
})
(*st, new-arena0*);
RETURN (*new-arena*)
}›

lemma *valid-arena-append-clause-slice*:
assumes
‹*valid-arena old-arena N vd*› **and**
‹*valid-arena new-arena N′ vd′*› **and**
‹$C \in\#$ *dom-m N*›
shows ‹*valid-arena* (*new-arena* @ *clause-slice old-arena N C*)
(*fmupd* (*length new-arena* + *header-size* ($N \propto C$)) ($N \propto C$, *irred N C*) *N′*)
(*insert* (*length new-arena* + *header-size* ($N \propto C$)) *vd′*)›
⟨*proof*⟩

lemma *fm-mv-clause-to-new-arena*:
assumes ‹*valid-arena old-arena N vd*› **and**
‹*valid-arena new-arena N′ vd′*› **and**
‹$C \in\#$ *dom-m N*›
shows ‹*fm-mv-clause-to-new-arena C old-arena new-arena* ≤
SPEC(λ*new-arena′*.
new-arena′ = *new-arena* @ *clause-slice old-arena N C* ∧
valid-arena (*new-arena* @ *clause-slice old-arena N C*)
(*fmupd* (*length new-arena* + *header-size* ($N \propto C$)) ($N \propto C$, *irred N C*) *N′*)
(*insert* (*length new-arena* + *header-size* ($N \propto C$)) *vd′*))›
⟨*proof*⟩

lemma *size-learned-clss-dom-m*: ‹*size* (*learned-clss-l N*) ≤ *size* (*dom-m N*)›
⟨*proof*⟩

lemma *distinct-sum-mset-sum*:
‹$distinct\text{-}mset\ As \Longrightarrow (\sum A \in\#\ As.\ (f :: 'a \Rightarrow nat)\ A) = (\sum A \in set\text{-}mset\ As.\ f\ A)$›
⟨*proof*⟩

lemma *distinct-sorted-append*: ‹$distinct\ (xs\ @\ [x]) \Longrightarrow sorted\ (xs\ @\ [x]) \longleftrightarrow sorted\ xs \wedge (\forall\, y \in set\ xs.\ y < x)$›
⟨*proof*⟩

lemma (**in** *linordered-ab-semigroup-add*) *Max-add-commute2*:
fixes k
assumes *finite* S **and** $S \neq \{\}$
shows $Max\ ((\lambda x.\ x + k)\ `\ S) = Max\ S + k$
⟨*proof*⟩

lemma *valid-arena-ge-length-clauses*:
assumes ‹*valid-arena arena N vdom*›
shows ‹$length\ arena \geq (\sum C \in\#\ dom\text{-}m\ N.\ length\ (N \propto C) + header\text{-}size\ (N \propto C))$›
⟨*proof*⟩

lemma *valid-arena-size-dom-m-le-arena*: ‹$valid\text{-}arena\ arena\ N\ vdom \Longrightarrow size\ (dom\text{-}m\ N) \leq length\ arena$›
⟨*proof*⟩

end
theory *IsaSAT-Clauses-SML*
imports *IsaSAT-Clauses IsaSAT-Arena-SML*
begin

abbreviation *isasat-clauses-assn* **where**
‹$isasat\text{-}clauses\text{-}assn \equiv arlO\text{-}assn\ clause\text{-}ll\text{-}assn *a\ arl\text{-}assn\ (clause\text{-}status\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn)$›

lemma *AStatus-IRRED* [*sepref-fr-rules*]:
‹$(uncurry0\ (return\ 0),\ uncurry0\ (RETURN\ AStatus\text{-}IRRED)) \in unit\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *AStatus-IRRED2* [*sepref-fr-rules*]:
‹$(uncurry0\ (return\ 0b100),\ uncurry0\ (RETURN\ AStatus\text{-}IRRED2)) \in unit\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *AStatus-LEARNED* [*sepref-fr-rules*]:
‹$(uncurry0\ (return\ 0b101),\ uncurry0\ (RETURN\ AStatus\text{-}LEARNED)) \in unit\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *AStatus-LEARNED2* [*sepref-fr-rules*]:
‹$(uncurry0\ (return\ 0b001),\ uncurry0\ (RETURN\ AStatus\text{-}LEARNED2)) \in unit\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *AActivity-hnr*[*sepref-fr-rules*]:
‹$(return\ o\ id,\ RETURN\ o\ AActivity) \in uint32\text{-}nat\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *ALBD-hnr*[*sepref-fr-rules*]:
‹$(return\ o\ id,\ RETURN\ o\ ALBD) \in uint32\text{-}nat\text{-}assn^k \rightarrow_a arena\text{-}el\text{-}assn$›
⟨*proof*⟩

lemma *ASize-hnr*[*sepref-fr-rules*]:
‹(*return o id*, *RETURN o ASize*) ∈ *uint32-nat-assn*^{k} $\rightarrow_a$ *arena-el-assn*›
⟨*proof*⟩

lemma *APos-hnr*[*sepref-fr-rules*]:
‹(*return o id*, *RETURN o APos*) ∈ *uint32-nat-assn*^{k} $\rightarrow_a$ *arena-el-assn*›
⟨*proof*⟩

lemma *ALit-hnr*[*sepref-fr-rules*]:
‹(*return o id*, *RETURN o ALit*) ∈ *unat-lit-assn*^{k} $\rightarrow_a$ *arena-el-assn*›
⟨*proof*⟩
lemma (**in**−)
four-uint64-nat-hnr[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN four-uint64-nat*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*› **and**
five-uint64-nat-hnr[*sepref-fr-rules*]:
‹(*uncurry0* (*return 5*), *uncurry0* (*RETURN five-uint64-nat*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-register *fm-mv-clause-to-new-arena*

definition *clauses-ll-assn*
:: ‹*vdom* ⇒ *nat clauses-l* ⇒ *uint32 array-list* ⇒ *assn*›
where
‹*clauses-ll-assn vdom* = *hr-comp arena-assn* (*clauses-l-fmat vdom*)›

lemma *nth-raa-i-uint64-hnr'*:
assumes *p*: ‹*is-pure R*›
shows
‹(*uncurry2* (λ(*N*, -) *j*. *nth-raa-i-u64 N j*), *uncurry2* (*RETURN* ∘∘∘ (λ(*N*, -) *j*. *nth-rll N j*))) ∈
$[\lambda(((l, -),i),j).\ i < length\ l \wedge j < length\text{-}rll\ l\ i]_a$
(*arlO-assn* (*array-assn R*) ∗*a GG*)^k $*_a$ *nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *R*›
⟨*proof*⟩

lemma *nth-raa-hnr'*:
assumes *p*: ‹*is-pure R*›
shows
‹(*uncurry2* (λ(*N*, -) *j k*. *nth-raa N j k*), *uncurry2* (*RETURN* ∘∘∘ (λ(*N*, -) *i*. *nth-rll N i*))) ∈
$[\lambda(((l, -),i),j).\ i < length\ l \wedge j < length\text{-}rll\ l\ i]_a$
(*arlO-assn* (*array-assn R*) ∗*a GG*)^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} → *R*›
⟨*proof*⟩

sepref-definition *nth-rll-u32-i64-clauses*
is ‹*uncurry2* (*RETURN ooo* (λ(*N*, -) *j*. *nth-rll N j*))›
:: ‹$[\lambda(((xs, -), i), j).\ i < length\ xs \wedge j < length\ (xs\ !i)]_a$
(*isasat-clauses-assn*)^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-definition *nth-rll-u64-i64-clauses*
is ‹*uncurry2* (*RETURN ooo* (λ(*N*, -) *j*. *nth-rll N j*))›
:: ‹$[\lambda(((xs, -), i), j).\ i < length\ xs \wedge j < length\ (xs\ !i)]_a$
(*isasat-clauses-assn*)^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-definition *length-rll-n-uint32-clss*
is ‹*uncurry* (*RETURN oo* (λ(*N*, -) *i*. *length-rll-n-uint32 N i*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs* $\wedge$ *length* (*xs* ! *i*) $\leq$ *uint-max*]$_a$
isasat-clauses-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-i64-u-clss*
is ‹*uncurry* (*RETURN oo* (λ(*N*, -) *i*. *length-rll-n-uint32 N i*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs* $\wedge$ *length* (*xs* ! *i*) $\leq$ *uint-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-u64-clss*
is ‹*uncurry* ((*RETURN* ∘∘∘ *case-prod*) (λ*N* -. *length-rll-n-uint64 N*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs* $\wedge$ *length* (*xs* ! *i*) $\leq$ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-u32-u64-clss*
is ‹*uncurry* ((*RETURN* ∘∘∘ *case-prod*) (λ*N* -. *length-rll-n-uint64 N*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs* $\wedge$ *length* (*xs* ! *i*) $\leq$ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-u64-u64-clss*
is ‹*uncurry* ((*RETURN* ∘∘∘ *case-prod*) (λ*N* -. *length-rll-n-uint64 N*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs* $\wedge$ *length* (*xs* ! *i*) $\leq$ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-u32-clss*
is ‹*uncurry* (*RETURN* ∘∘ (λ(*N*, -) *i*. *length-rll N i*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs*]$_a$ *isasat-clauses-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *nat-assn*›
⟨*proof*⟩

sepref-definition *length-raa-clss*
is ‹*uncurry* (*RETURN* ∘∘ (λ(*N*, -) *i*. *length-rll N i*))›
:: ‹[λ((*xs*, -), *i*). $i <$ *length xs*]$_a$ *isasat-clauses-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *nat-assn*›
⟨*proof*⟩

sepref-definition *swap-aa-clss*
is ‹*uncurry3* (*RETURN oooo* (λ(*N*, *xs*) *i j k*. (*swap-ll N i j k*, *xs*)))›
:: ‹[λ((((*xs*, -), *k*), *i*), *j*). $k <$ *length xs* $\wedge$ $i <$ *length-rll xs k* $\wedge$ $j <$ *length-rll xs k*]$_a$
isasat-clauses-assn^d $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *isasat-clauses-assn*›
⟨*proof*⟩

sepref-definition *is-short-clause-code*
is ‹*RETURN o is-short-clause*›
:: ‹*clause-ll-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩
declare *is-short-clause-code.refine*[*sepref-fr-rules*]

sepref-definition *header-size-code*
is ‹*RETURN o header-size*›
:: ‹$clause\text{-}ll\text{-}assn^k \rightarrow_a nat\text{-}assn$›
⟨*proof*⟩

declare *header-size-code.refine*[*sepref-fr-rules*]

sepref-definition *append-and-length-code*
is ‹*uncurry2 fm-add-new*›
:: ‹$[\lambda((b,\ C),\ N).\ length\ C \leq uint32\text{-}max{+}2 \wedge length\ C \geq 2]_a\ bool\text{-}assn^k *_a clause\text{-}ll\text{-}assn^d *_a (arena\text{-}assn)^d \rightarrow$
$arena\text{-}assn *a\ nat\text{-}assn$›
⟨*proof*⟩

declare *append-and-length-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *header-size-fast-code*
is ‹*RETURN o header-size*›
:: ‹$clause\text{-}ll\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare (**in** −)*header-size-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −)*append-and-length-fast-code*
is ‹*uncurry2 fm-add-new-fast*›
:: ‹$[append\text{-}and\text{-}length\text{-}fast\text{-}code\text{-}pre]_a$
$bool\text{-}assn^k *_a clause\text{-}ll\text{-}assn^d *_a (arena\text{-}fast\text{-}assn)^d \rightarrow$
$arena\text{-}fast\text{-}assn *a\ uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *append-and-length-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *fmap-swap-ll-u64-clss*
is ‹$uncurry3\ (RETURN\ oooo\ (\lambda(N,\ xs)\ i\ j\ k.\ (swap\text{-}ll\ N\ i\ j\ k,\ xs)))$›
::‹$[\lambda((((xs,\ \text{-}),\ k),\ i),\ j).\ k < length\ xs \wedge i < length\text{-}rll\ xs\ k \wedge j < length\text{-}rll\ xs\ k]_a$
$(isasat\text{-}clauses\text{-}assn^d *_a nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k) \rightarrow$
$isasat\text{-}clauses\text{-}assn$›
⟨*proof*⟩

sepref-definition *fmap-rll-u-clss*
is ‹$uncurry2\ (RETURN\ ooo\ (\lambda(N,\ \text{-})\ i.\ nth\text{-}rll\ N\ i))$›
:: ‹$[\lambda(((l,\ \text{-}),\ i),\ j).\ i < length\ l \wedge j < length\text{-}rll\ l\ i]_a$
$isasat\text{-}clauses\text{-}assn^k *_a nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow$
$unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-definition *fmap-rll-u32-clss*
is ‹$uncurry2\ (RETURN\ ooo\ (\lambda(N,\ \text{-})\ i.\ nth\text{-}rll\ N\ i))$›
:: ‹$[\lambda(((l,\ \text{-}),\ i),\ j).\ i < length\ l \wedge j < length\text{-}rll\ l\ i]_a$
$isasat\text{-}clauses\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow$
$unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-definition *swap-lits-code*
is ‹*uncurry3 isa-arena-swap*›
:: ‹$nat\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a (arl\text{-}assn\ uint32\text{-}assn)^d \rightarrow_a arl\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *swap-lits-refine*[*sepref-fr-rules*]:
‹(*uncurry3 swap-lits-code*, *uncurry3* (*RETURN oooo swap-lits*))
$\in [uncurry3\ swap\text{-}lits\text{-}pre]_a\ nat\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a arena\text{-}assn^d \rightarrow arena\text{-}assn$›
⟨*proof*⟩

sepref-definition (**in** −) *swap-lits-fast-code*
is ‹*uncurry3 isa-arena-swap*›
:: ‹$[\lambda(((\text{-}, \text{-}), \text{-}), N).\ length\ N \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a (arl64\text{-}assn\ uint32\text{-}assn)^d \rightarrow$
$arl64\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *swap-lits-fast-refine*[*sepref-fr-rules*]:
‹(*uncurry3 swap-lits-fast-code*, *uncurry3* (*RETURN oooo swap-lits*))
$\in [\lambda(((C, i), j), arena).\ swap\text{-}lits\text{-}pre\ C\ i\ j\ arena \wedge length\ arena \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

declare *swap-lits-fast-code.refine*[*sepref-fr-rules*]
sepref-definition *fm-mv-clause-to-new-arena-code*
is ‹*uncurry2 fm-mv-clause-to-new-arena*›
:: ‹$nat\text{-}assn^k *_a arena\text{-}assn^k *_a arena\text{-}assn^d \rightarrow_a arena\text{-}assn$›
⟨*proof*⟩

declare *fm-mv-clause-to-new-arena-code.refine*[*sepref-fr-rules*]
sepref-definition *fm-mv-clause-to-new-arena-fast-code*
is ‹*uncurry2 fm-mv-clause-to-new-arena*›
:: ‹$[\lambda((n, arena_o), arena).\ length\ arena_o \leq uint64\text{-}max \wedge length\ arena + arena\text{-}length\ arena_o\ n +$
$(if\ arena\text{-}length\ arena_o\ \ n \leq 4\ then\ 4\ else\ 5) \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

declare *fm-mv-clause-to-new-arena-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT-Trail*
imports *IsaSAT-Literals*

begin

Trail

Our trail contains several additional information compared to the simple trail:

- the (reversed) trail in an array (i.e., the trail in the same order as presented in "Automated Reasoning");
- the mapping from any *literal* (and not an atom) to its polarity;

- the mapping from a *atom* to its level or reason (in two different arrays);
- the current level of the state;
- the control stack.

We copied the idea from the mapping from a literals to it polarity instead of an atom to its polarity from a comment by Armin Biere in CaDiCal. We only observed a (at best) faint performance increase, but as it seemed slightly faster and does not increase the length of the formalisation, we kept it.

The control stack is the latest addition: it contains the positions of the decisions in the trail. It is mostly to enable fast restarts (since it allows to directly iterate over all decision of the trail), but might also slightly speed up backjumping (since we know how far we are going back in the trail). Remark that the control stack contains is not updated during the backjumping, but only *after* doing it (as we keep only the the beginning of it).

Polarities **type-synonym** *tri-bool* = ‹*bool option*›
type-synonym *tri-bool-assn* = ‹*uint32*›

We define set/non set not as the trivial *None*, *Some True*, and *Some False*, because it is not clear whether the compiler can represent the values without pointers. Therefore, we use *uint32*.

definition *UNSET-code* :: ‹*tri-bool-assn*› **where**
  [*simp*]: ‹*UNSET-code* = *0*›

definition *SET-TRUE-code* :: ‹*tri-bool-assn*› **where**
  [*simp*]: ‹*SET-TRUE-code* = *2*›

definition *SET-FALSE-code* :: ‹*tri-bool-assn*› **where**
  [*simp*]: ‹*SET-FALSE-code* = *3*›

definition *UNSET* :: ‹*tri-bool*› **where**
  [*simp*]: ‹*UNSET* = *None*›

definition *SET-FALSE* :: ‹*tri-bool*› **where**
  [*simp*]: ‹*SET-FALSE* = *Some False*›

definition *SET-TRUE* :: ‹*tri-bool*› **where**
  [*simp*]: ‹*SET-TRUE* = *Some True*›

definition *tri-bool-ref* :: ‹(*tri-bool-assn* × *tri-bool*) *set*› **where**
  ‹*tri-bool-ref* = {(*SET-TRUE-code*, *SET-TRUE*), (*UNSET-code*, *UNSET*), (*SET-FALSE-code*, *SET-FALSE*)}›

definition (**in** −) *tri-bool-eq* :: ‹*tri-bool* ⇒ *tri-bool* ⇒ *bool*› **where**
  ‹*tri-bool-eq* = (=)›

Types **type-synonym** *trail-pol* =
  ‹*nat literal list* × *tri-bool list* × *nat list* × *nat list* × *nat* × *nat list*›

definition *get-level-atm* **where**
  ‹*get-level-atm M L* = *get-level M* (*Pos L*)›

definition *polarity-atm* **where**
  ‹*polarity-atm M L* =

(*if Pos L* ∈ *lits-of-l M then Some True*
else if Neg L ∈ *lits-of-l M then Some False*
else None)›

definition *defined-atm* :: ‹(*'v*, *nat*) *ann-lits* ⇒ *'v* ⇒ *bool*› **where**
‹*defined-atm M L* = *defined-lit M* (*Pos L*)›

abbreviation *undefined-atm* **where**
‹*undefined-atm M L* ≡ ¬*defined-atm M L*›

Control Stack **inductive** *control-stack* **where**
empty:
‹*control-stack* [] []› |
cons-prop:
‹*control-stack cs M* ⟹ *control-stack cs* (*Propagated L C* # *M*)› |
cons-dec:
‹*control-stack cs M* ⟹ *n* = *length M* ⟹ *control-stack* (*cs* @ [*n*]) (*Decided L* # *M*)›

inductive-cases *control-stackE*: ‹*control-stack cs M*›

lemma *control-stack-length-count-dec*:
‹*control-stack cs M* ⟹ *length cs* = *count-decided M*›
⟨*proof*⟩

lemma *control-stack-le-length-M*:
‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *c* < *length M*›
⟨*proof*⟩

lemma *control-stack-propa*[*simp*]:
‹*control-stack cs* (*Propagated x21 x22* # *list*) ⟷ *control-stack cs list*›
⟨*proof*⟩

lemma *control-stack-filter-map-nth*:
‹*control-stack cs M* ⟹ *filter is-decided* (*rev M*) = *map* (*nth* (*rev M*)) *cs*›
⟨*proof*⟩

lemma *control-stack-empty-cs*[*simp*]: ‹*control-stack* [] *M* ⟷ *count-decided M* = *0*›
⟨*proof*⟩

This is an other possible definition. It is not inductive, which makes it easier to reason about appending (or removing) some literals from the trail. It is however much less clear if the definition is correct.

definition *control-stack'* **where**
‹*control-stack' cs M* ⟷
(*length cs* = *count-decided M* ∧
(∀ *L*∈*set M*. *is-decided L* ⟶ (*cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧
rev M!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*)))›

lemma *control-stack-rev-get-lev*:
‹*control-stack cs M* ⟹
no-dup M ⟹ *L*∈*set M* ⟹ *is-decided L* ⟹ *rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
⟨*proof*⟩

lemma *control-stack-alt-def-imp*:
‹*no-dup M* ⟹ (⋀*L*. *L* ∈*set M* ⟹ *is-decided L* ⟹ *cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧

$rev\ M!(cs\ !\ (get\text{-}level\ M\ (lit\text{-}of\ L)\ -\ 1)) = L) \Longrightarrow$
$length\ cs = count\text{-}decided\ M \Longrightarrow$
$control\text{-}stack\ cs\ M$›
⟨*proof*⟩

lemma *control-stack-alt-def*: ‹$no\text{-}dup\ M \Longrightarrow control\text{-}stack'\ cs\ M \longleftrightarrow control\text{-}stack\ cs\ M$›
⟨*proof*⟩

lemma *control-stack-decomp*:
assumes
decomp: ‹$(Decided\ L\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$› **and**
cs: ‹$control\text{-}stack\ cs\ M$› **and**
n-d: ‹$no\text{-}dup\ M$›
shows ‹$control\text{-}stack\ (take\ (count\text{-}decided\ M1)\ cs)\ M1$›
⟨*proof*⟩

Encoding of the reasons **definition** *DECISION-REASON* :: *nat* **where**
‹$DECISION\text{-}REASON = 1$›

definition *ann-lits-split-reasons* **where**
‹$ann\text{-}lits\text{-}split\text{-}reasons\ \mathcal{A} = \{((M,\ reasons),\ M').\ M = map\ lit\text{-}of\ (rev\ M') \land$
$(\forall L \in set\ M'.\ is\text{-}proped\ L \longrightarrow$
$reasons\ !\ (atm\text{-}of\ (lit\text{-}of\ L)) = mark\text{-}of\ L \land mark\text{-}of\ L \neq DECISION\text{-}REASON) \land$
$(\forall L \in set\ M'.\ is\text{-}decided\ L \longrightarrow reasons\ !\ (atm\text{-}of\ (lit\text{-}of\ L)) = DECISION\text{-}REASON) \land$
$(\forall L \in\#\ \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ reasons)$
$\}$›

definition *trail-pol* :: ‹$nat\ multiset \Rightarrow (trail\text{-}pol \times (nat,\ nat)\ ann\text{-}lits)\ set$› **where**
‹$trail\text{-}pol\ \mathcal{A} =$
$\{((M',\ xs,\ lvls,\ reasons,\ k,\ cs),\ M).\ ((M',\ reasons),\ M) \in ann\text{-}lits\text{-}split\text{-}reasons\ \mathcal{A} \land$
$no\text{-}dup\ M \land$
$(\forall L \in\#\ \mathcal{L}_{all}\ \mathcal{A}.\ nat\text{-}of\text{-}lit\ L < length\ xs \land xs\ !\ (nat\text{-}of\text{-}lit\ L) = polarity\ M\ L) \land$
$(\forall L \in\#\ \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ lvls \land lvls\ !\ (atm\text{-}of\ L) = get\text{-}level\ M\ L) \land$
$k = count\text{-}decided\ M \land$
$(\forall L \in set\ M.\ lit\text{-}of\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}) \land$
$control\text{-}stack\ cs\ M \land$
$isasat\text{-}input\text{-}bounded\ \mathcal{A}\}$›

Definition of the full trail **lemma** *trail-pol-alt-def*:
‹$trail\text{-}pol\ \mathcal{A} = \{((M',\ xs,\ lvls,\ reasons,\ k,\ cs),\ M).$
$((M',\ reasons),\ M) \in ann\text{-}lits\text{-}split\text{-}reasons\ \mathcal{A} \land$
$no\text{-}dup\ M \land$
$(\forall L \in\#\ \mathcal{L}_{all}\ \mathcal{A}.\ nat\text{-}of\text{-}lit\ L < length\ xs \land xs\ !\ (nat\text{-}of\text{-}lit\ L) = polarity\ M\ L) \land$
$(\forall L \in\#\ \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ lvls \land lvls\ !\ (atm\text{-}of\ L) = get\text{-}level\ M\ L) \land$
$k = count\text{-}decided\ M \land$
$(\forall L \in set\ M.\ lit\text{-}of\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}) \land$
$control\text{-}stack\ cs\ M \land literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ \mathcal{A}\ M \land$
$length\ M < uint32\text{-}max \land$
$length\ M \leq uint32\text{-}max\ div\ 2 + 1 \land$
$count\text{-}decided\ M < uint32\text{-}max \land$
$length\ M' = length\ M \land$
$M' = map\ lit\text{-}of\ (rev\ M) \land$
$isasat\text{-}input\text{-}bounded\ \mathcal{A}$
$\}$›
⟨*proof*⟩

Code generation

Conversion between incomplete and complete mode **definition** *trail-fast-of-slow* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-fast-of-slow* = *id*›

definition *trail-pol-slow-of-fast* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-slow-of-fast* =
(λ(*M*, *val*, *lvls*, *reason*, *k*, *cs*). (*M*, *val*, *lvls*, *array-nat-of-uint64-conv reason*, *k*, *cs*))›

definition *trail-slow-of-fast* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-slow-of-fast* = *id*›

definition *trail-pol-fast-of-slow* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-fast-of-slow* =
(λ(*M*, *val*, *lvls*, *reason*, *k*, *cs*). (*M*, *val*, *lvls*, *array-uint64-of-nat-conv reason*, *k*, *cs*))›

lemma *trail-pol-slow-of-fast-alt-def*:
‹*trail-pol-slow-of-fast M* = *M*›
⟨*proof*⟩

lemma *trail-pol-fast-of-slow-trail-fast-of-slow*:
‹(*RETURN o trail-pol-fast-of-slow*, *RETURN o trail-fast-of-slow*)
∈ [λ*M*. (∀ *C L*. *Propagated L C* ∈ *set M* ⟶ *C* < *uint64-max*)]$_f$
trail-pol $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩ *nres-rel*›
⟨*proof*⟩

lemma *trail-pol-slow-of-fast-trail-slow-of-fast*:
‹(*RETURN o trail-pol-slow-of-fast*, *RETURN o trail-slow-of-fast*)
∈ *trail-pol* $\mathcal{A}$ $\rightarrow_f$ ⟨*trail-pol* $\mathcal{A}$⟩ *nres-rel*›
⟨*proof*⟩

lemma *trail-pol-same-length*[*simp*]: ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *length* (*fst M′*) = *length M*›
⟨*proof*⟩

definition *counts-maximum-level* **where**
‹*counts-maximum-level M C* = {*i*. *C* ≠ *None* ⟶ *i* = *card-max-lvl M* (*the C*)}›

lemma *counts-maximum-level-None*[*simp*]: ‹*counts-maximum-level M None* = *Collect* (λ-. *True*)›
⟨*proof*⟩

Level of a literal **definition** *get-level-atm-pol-pre* **where**
‹*get-level-atm-pol-pre* = (λ((*M*, *xs*, *lvls*, *k*), *L*). *L* < *length lvls*)›

definition *get-level-atm-pol* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat*› **where**
‹*get-level-atm-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *lvls* ! *L*)›

lemma *get-level-atm-pol-pre*:
assumes
‹*Pos L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$›
shows ‹*get-level-atm-pol-pre* (*M′*, *L*)›
⟨*proof*⟩

lemma (**in** −) *get-level-get-level-atm*: ‹*get-level M L* = *get-level-atm M* (*atm-of L*)›

⟨*proof*⟩

definition *get-level-pol* **where**
‹*get-level-pol M L = get-level-atm-pol M (atm-of L)*›

definition *get-level-pol-pre* **where**
‹*get-level-pol-pre = (λ((M, xs, lvls, k), L). atm-of L < length lvls)*›

lemma *get-level-pol-pre*:
assumes
‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and**
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$›
shows ‹*get-level-pol-pre* (M', L)›
⟨*proof*⟩

lemma *get-level-get-level-pol*:
assumes
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and** ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$›
shows ‹*get-level M L = get-level-pol M' L*›
⟨*proof*⟩

Current level **definition** (**in** −) *count-decided-pol* **where**
‹*count-decided-pol = (λ(-, -, -, -, k, -). k)*›

lemma *count-decided-trail-ref*:
‹*(RETURN o count-decided-pol, RETURN o count-decided)* $\in$ *trail-pol* $\mathcal{A} \rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

Polarity **definition** (**in** −) *polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option*› **where**
‹*polarity-pol = (λ(M, xs, lvls, k) L. do {*
xs ! (nat-of-lit L)
})›

definition *polarity-pol-pre* **where**
‹*polarity-pol-pre = (λ(M, xs, lvls, k) L. nat-of-lit L < length xs)*›

lemma *polarity-pol-polarity*:
‹*(uncurry (RETURN oo polarity-pol), uncurry (RETURN oo polarity))* $\in$
$[\lambda(M, L).\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_f$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *polarity-pol-pre*:
‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$ *polarity-pol-pre M' L*›
⟨*proof*⟩

0.1.6 Length of the trail

definition (**in** −) *isa-length-trail-pre* **where**
‹*isa-length-trail-pre = (λ (M', xs, lvls, reasons, k, cs). length M'* ≤ *uint32-max)*›

definition (**in** −) *isa-length-trail* **where**
‹*isa-length-trail = (λ (M', xs, lvls, reasons, k, cs). length-uint32-nat M')*›

lemma *isa-length-trail-pre*:
‹$(M, M') \in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *isa-length-trail-pre M*›

⟨*proof*⟩

lemma *isa-length-trail-length-u*:
‹(*RETURN o isa-length-trail*, *RETURN o length-uint32-nat*) ∈ *trail-pol* $\mathcal{A}$ $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

Consing elements **definition** *cons-trail-Propagated* :: ‹*nat literal* ⇒ *nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*cons-trail-Propagated L C M′* = *Propagated L C* # *M′*›

definition *cons-trail-Propagated-tr* :: ‹*nat literal* ⇒ *nat* ⇒ *trail-pol* ⇒ *trail-pol*› **where**
‹*cons-trail-Propagated-tr* = (λ*L C* (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*).
(*M′* @ [*L*], *let xs* = *xs*[*nat-of-lit L* := *Some True*] *in xs*[*nat-of-lit* (−*L*) := *Some False*],
lvls[*atm-of L* := *k*], *reasons*[*atm-of L*:= *C*], *k*, *cs*))›

lemma *in-list-pos-neg-notD*: ‹*Pos* (*atm-of* (*lit-of La*)) ∉ *lits-of-l bc* ⟹
Neg (*atm-of* (*lit-of La*)) ∉ *lits-of-l bc* ⟹
La ∈ *set bc* ⟹ *False*›
⟨*proof*⟩

lemma *cons-trail-Propagated-tr*:
‹(*uncurry2* (*RETURN ooo cons-trail-Propagated-tr*), *uncurry2* (*RETURN ooo cons-trail-Propagated*)) ∈
[λ((*L*, *C*), *M*). *undefined-lit M L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *C* ≠ *DECISION-REASON*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *trail-pol* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

lemma *undefined-lit-count-decided-uint-max*:
assumes
M-$\mathcal{L}_{all}$: ‹∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*undefined-lit M L*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*Suc* (*count-decided M*) ≤ *uint-max*›
⟨*proof*⟩

lemma *length-trail-uint-max*:
assumes
M-$\mathcal{L}_{all}$: ‹∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*length M* ≤ *uint-max*›
⟨*proof*⟩

definition *cons-trail-Propagated-tr-pre* **where**
‹*cons-trail-Propagated-tr-pre* = (λ((*L*, *C*), (*M*, *xs*, *lvls*, *reasons*, *k*)). *nat-of-lit L* < *length xs* ∧
nat-of-lit (−*L*) < *length xs* ∧ *atm-of L* < *length lvls* ∧ *atm-of L* < *length reasons* ∧ *length M* < *uint32-max*)›

lemma *cons-trail-Propagated-tr-pre*:
assumes ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
‹*undefined-lit M L*› **and**
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*C* ≠ *DECISION-REASON*›
shows ‹*cons-trail-Propagated-tr-pre* ((*L*, *C*), *M′*)›
⟨*proof*⟩

lemma *cons-trail-Propagated-tr2*:
$\langle(M', M) \in \textit{trail-pol } \mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow \textit{undefined-lit } M\ L \Longrightarrow C \neq \textit{DECISION-REASON} \Longrightarrow$
$(\textit{cons-trail-Propagated-tr } L\ C\ M', \textit{Propagated } L\ C\ \#\ M) \in \textit{trail-pol } \mathcal{A}\rangle$
$\langle \textit{proof} \rangle$

definition *last-trail-pol-pre* **where**
$\langle \textit{last-trail-pol-pre} = (\lambda(M, xs, lvls, reasons, k).\ \textit{atm-of } (\textit{last } M) < \textit{length reasons} \wedge M \neq [])\rangle$

definition (**in** $-$) *last-trail-pol* :: $\langle \textit{trail-pol} \Rightarrow (\textit{nat literal} \times \textit{nat option})\rangle$ **where**
$\langle \textit{last-trail-pol} = (\lambda(M, xs, lvls, reasons, k).$
$\quad \textit{let } r = \textit{reasons } !\ (\textit{atm-of } (\textit{last } M))\ \textit{in}$
$\quad (\textit{last } M, \textit{if } r = \textit{DECISION-REASON then None else Some } r))\rangle$

lemma (**in** $-$) *nat-ann-lit-rel-alt-def*: $\langle \textit{nat-ann-lit-rel} = (\textit{unat-lit-rel} \times_r \langle \textit{nat-rel}\rangle\ \textit{option-rel})\ O$
$\quad \{((L, C), L').$
$\quad (C = \textit{None} \longrightarrow L' = \textit{Decided } L) \wedge$
$\quad (C \neq \textit{None} \longrightarrow L' = \textit{Propagated } L\ (\textit{the } C))\}\rangle$
$\langle \textit{proof} \rangle$

definition *tl-trailt-tr* :: $\langle \textit{trail-pol} \Rightarrow \textit{trail-pol}\rangle$ **where**
$\langle \textit{tl-trailt-tr} = (\lambda(M', xs, lvls, reasons, k, cs).$
$\quad \textit{let } L = \textit{last } M'\ \textit{in}$
$\quad (\textit{butlast } M',$
$\quad \textit{let } xs = xs[\textit{nat-of-lit } L := \textit{None}]\ \textit{in } xs[\textit{nat-of-lit } (-L) := \textit{None}],$
$\quad lvls[\textit{atm-of } L := \textit{zero-uint32-nat}],$
$\quad \textit{reasons}, \textit{if reasons } !\ \textit{atm-of } L = \textit{DECISION-REASON then } k - \textit{one-uint32-nat else } k,$
$\quad\quad \textit{if reasons } !\ \textit{atm-of } L = \textit{DECISION-REASON then butlast } cs\ \textit{else } cs))\rangle$

definition *tl-trailt-tr-pre* **where**
$\langle \textit{tl-trailt-tr-pre} = (\lambda(M, xs, lvls, reason, k, cs).\ M \neq [] \wedge \textit{nat-of-lit}(\textit{last } M) < \textit{length } xs \wedge$
$\quad \textit{nat-of-lit}(-\textit{last } M) < \textit{length } xs \wedge \textit{atm-of } (\textit{last } M) < \textit{length lvls} \wedge$
$\quad \textit{atm-of } (\textit{last } M) < \textit{length reason} \wedge$
$\quad (\textit{reason } !\ \textit{atm-of } (\textit{last } M) = \textit{DECISION-REASON} \longrightarrow k \geq 1 \wedge cs \neq []))\rangle$

lemma *ann-lits-split-reasons-map-lit-of*:
$\langle((M, \textit{reasons}), M') \in \textit{ann-lits-split-reasons } \mathcal{A} \Longrightarrow M = \textit{map lit-of } (\textit{rev } M')\rangle$
$\langle \textit{proof} \rangle$

lemma *control-stack-dec-butlast*:
$\langle \textit{control-stack } b\ (\textit{Decided } x1\ \#\ M's) \Longrightarrow \textit{control-stack } (\textit{butlast } b)\ M's\rangle$
$\langle \textit{proof} \rangle$

lemma *tl-trail-tr*:
$\langle((\textit{RETURN } o\ \textit{tl-trailt-tr}), (\textit{RETURN } o\ \textit{tl})) \in$
$\quad [\lambda M.\ M \neq []]_f\ \textit{trail-pol } \mathcal{A} \rightarrow \langle \textit{trail-pol } \mathcal{A}\rangle \textit{nres-rel}\rangle$
$\langle \textit{proof} \rangle$

lemma *tl-trailt-tr-pre*:
assumes $\langle M \neq []\rangle$
$\langle(M', M) \in \textit{trail-pol } \mathcal{A}\rangle$
shows $\langle \textit{tl-trailt-tr-pre } M'\rangle$
$\langle \textit{proof} \rangle$

definition *tl-trail-propedt-tr* :: $\langle \textit{trail-pol} \Rightarrow \textit{trail-pol}\rangle$ **where**
$\langle \textit{tl-trail-propedt-tr} = (\lambda(M', xs, lvls, reasons, k, cs).$

let L = *last* M' *in*
(*butlast* M',
let xs = xs[*nat-of-lit* L := *None*] *in* xs[*nat-of-lit* ($-L$) := *None*],
lvls[*atm-of* L := *zero-uint32-nat*],
reasons, k, cs))›

definition *tl-trail-propedt-tr-pre* **where**
‹*tl-trail-propedt-tr-pre* =
(λ(M, xs, *lvls*, *reason*, k, cs). $M \neq []$ $\wedge$ *nat-of-lit*(*last* M) < *length* xs $\wedge$
nat-of-lit($-$*last* M) < *length* xs $\wedge$ *atm-of* (*last* M) < *length lvls* $\wedge$
atm-of (*last* M) < *length reason*)›

lemma *tl-trail-propedt-tr-pre*:
assumes ‹(M', M) $\in$ *trail-pol* $\mathcal{A}$› **and**
‹$M \neq []$›
shows ‹*tl-trail-propedt-tr-pre* M'›
⟨*proof*⟩

definition (**in** $-$) *lit-of-hd-trail* **where**
‹*lit-of-hd-trail* M = *lit-of* (*hd* M)›

definition (**in** $-$) *lit-of-last-trail-pol* **where**
‹*lit-of-last-trail-pol* = (λ(M, -). *last* M)›

lemma *lit-of-last-trail-pol-lit-of-last-trail*:
‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) $\in$
[λS. $S \neq []$]$_f$ *trail-pol* $\mathcal{A}$ $\rightarrow$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

Setting a new literal **definition** *cons-trail-Decided* :: ‹*nat literal* $\Rightarrow$ (*nat*, *nat*) *ann-lits* $\Rightarrow$ (*nat*, *nat*) *ann-lits*› **where**
‹*cons-trail-Decided* L M' = *Decided* L # M'›

definition *cons-trail-Decided-tr* :: ‹*nat literal* $\Rightarrow$ *trail-pol* $\Rightarrow$ *trail-pol*› **where**
‹*cons-trail-Decided-tr* = (λL (M', xs, *lvls*, *reasons*, k, cs). *do*{
let n = *length* M' *in*
(M' @ [L], *let* xs = xs[*nat-of-lit* L := *Some True*] *in* xs[*nat-of-lit* ($-L$) := *Some False*],
lvls[*atm-of* L := k+*1*], *reasons*[*atm-of* L := *DECISION-REASON*], k+*1*, cs @ [*nat-of-uint32-spec* n])})›

definition *cons-trail-Decided-tr-pre* **where**
‹*cons-trail-Decided-tr-pre* =
(λ(L, (M, xs, *lvls*, *reason*, k, cs)). *nat-of-lit* L < *length* xs $\wedge$ *nat-of-lit* ($-L$) < *length* xs $\wedge$
atm-of L < *length lvls* $\wedge$ *atm-of* L < *length reason* $\wedge$ *length* cs < *uint32-max* $\wedge$
Suc k $\leq$ *uint-max* $\wedge$ *length* M < *uint32-max*)›

lemma *length-cons-trail-Decided*[*simp*]:
‹*length* (*cons-trail-Decided* L M) = *Suc* (*length* M)›
⟨*proof*⟩

lemma *cons-trail-Decided-tr*:
‹(*uncurry* (*RETURN oo cons-trail-Decided-tr*), *uncurry* (*RETURN oo cons-trail-Decided*)) $\in$
[λ(L, M). *undefined-lit* M L $\wedge$ L $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\rightarrow$ ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

lemma *cons-trail-Decided-tr-pre*:
assumes ‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and**
‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹*undefined-lit M L*›
shows ‹*cons-trail-Decided-tr-pre* (L, M')›
⟨*proof*⟩

Polarity: Defined or Undefined **definition** (**in** −) *defined-atm-pol-pre* **where**
‹*defined-atm-pol-pre* = $(\lambda(M, xs, lvls, k)\ L.\ 2{*}L <$ *length xs* $\wedge$
$2{*}L \leq$ *uint-max*)›

definition (**in** −) *defined-atm-pol* **where**
‹*defined-atm-pol* = $(\lambda(M, xs, lvls, k)\ L.\ \neg((xs!(\textit{two-uint32-nat}{*}L)) = None))$›

lemma *undefined-atm-code*:
‹(*uncurry* (*RETURN oo defined-atm-pol*), *uncurry* (*RETURN oo defined-atm*)) $\in$
$[\lambda(M, L).\ Pos\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_r Id \rightarrow \langle$*bool-rel*$\rangle$ *nres-rel*› (**is** *?A*) **and**
defined-atm-pol-pre:
‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\#\ \ \mathcal{A} \Longrightarrow$ *defined-atm-pol-pre* $M'\ L$›
⟨*proof*⟩

Reasons **definition** *get-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-propagation-reason-pol* = (λ(-, -, -, *reasons*, -) *L. do* {
ASSERT(*atm-of L* < *length reasons*);
let r = *reasons* ! *atm-of L*;
RETURN (*if r* = *DECISION-REASON then None else Some r*)})›

lemma *get-propagation-reason-pol*:
‹(*uncurry get-propagation-reason-pol*, *uncurry get-propagation-reason*) $\in$
$[\lambda(M, L).\ L \in$ *lits-of-l* $M]_f$ *trail-pol* $\mathcal{A} \times_r Id \rightarrow \langle\langle$*nat-rel*$\rangle$*option-rel*$\rangle$ *nres-rel*›
⟨*proof*⟩

The version *get-propagation-reason* can return the reason, but does not have to: it can be more suitable for specification (like for the conflict minimisation, where finding the reason is not mandatory).

The following version *always* returns the reasons if there is one. Remark that both functions are linked to the same code (but *get-propagation-reason* can be called first with some additional filtering later).

definition (**in** −) *get-the-propagation-reason*
:: ‹(*'v*, *'mark*) *ann-lits* ⇒ *'v literal* ⇒ *'mark option nres*›
where
‹*get-the-propagation-reason M L* = *SPEC*(λ*C*.
($C \neq None \longleftrightarrow$ *Propagated L* (*the C*) $\in$ *set M*) $\wedge$
($C = None \longleftrightarrow$ *Decided L* $\in$ *set M* $\vee\ L \notin$ *lits-of-l M*))›

lemma *no-dup-Decided-PropedD*:
‹*no-dup ad* $\Longrightarrow$ *Decided L* $\in$ *set ad* $\Longrightarrow$ *Propagated L C* $\in$ *set ad* $\Longrightarrow$ *False*›
⟨*proof*⟩

definition *get-the-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-the-propagation-reason-pol* = (λ(-, *xs*, -, *reasons*, -) *L. do* {
ASSERT(*atm-of L* < *length reasons*);
ASSERT(*nat-of-lit L* < *length xs*);
let r = *reasons* ! *atm-of L*;

RETURN (*if xs* ! *nat-of-lit L* = *SET-TRUE* ∧ *r* ≠ *DECISION-REASON then Some r else None*)})›

lemma *get-the-propagation-reason-pol*:
‹(*uncurry get-the-propagation-reason-pol*, *uncurry get-the-propagation-reason*) ∈
$[\lambda(M, L).\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
⟨*proof*⟩

Direct access to elements in the trail **definition** (**in** −) *rev-trail-nth* **where**
‹*rev-trail-nth M i* = *lit-of* (*rev M* ! *i*)›

definition (**in** −) *isa-trail-nth* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isa-trail-nth* = (λ(*M*, -) *i*. *do* {
  ASSERT(*i* < *length M*);
  RETURN (*M* ! *i*)
})›

lemma *isa-trail-nth-rev-trail-nth*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) ∈
$[\lambda(M, i).\ i < length\ M]_f$ *trail-pol* $\mathcal{A} \times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

We here define a variant of the trail representation, where the the control stack is out of sync of the trail (i.e., there are some leftovers at the end). This might make backtracking a little faster.

definition *trail-pol-no-CS* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*›
where
‹*trail-pol-no-CS* $\mathcal{A}$ =
{((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*), *M*). ((*M'*, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
  no-dup M ∧
  (∀ *L* $\in\# \mathcal{L}_{all}\ \mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
  (∀ *L* $\in\# \mathcal{L}_{all}\ \mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
  (∀ *L*∈*set M*. *lit-of L* $\in\# \mathcal{L}_{all}\ \mathcal{A}$) ∧
  isasat-input-bounded $\mathcal{A}$ ∧
  control-stack (*take* (*count-decided M*) *cs*) *M*
}›

definition *tl-trailt-tr-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trailt-tr-no-CS* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*).
  let L = *last M'* *in*
  (*butlast M'*,
  let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* (−*L*) := *None*],
  lvls[*atm-of L* := *zero-uint32-nat*],
  reasons, *k*, *cs*))›

definition *tl-trailt-tr-no-CS-pre* **where**
‹*tl-trailt-tr-no-CS-pre* = (λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). *M* ≠ [] ∧ *nat-of-lit*(*last M*) < *length xs* ∧
    nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
    atm-of (*last M*) < *length reason*)›

lemma *control-stack-take-Suc-count-dec-unstack*:
‹*control-stack* (*take* (*Suc* (*count-decided M's*)) *cs*) (*Decided x1* # *M's*) ⟹
  control-stack (*take* (*count-decided M's*) *cs*) *M's*›
⟨*proof*⟩

lemma *tl-trailt-tr-no-CS-pre*:
assumes ‹(*M'*, *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$› **and** ‹*M* ≠ []›

shows ‹*tl-trailt-tr-no-CS-pre M'*›
⟨*proof*⟩

lemma *tl-trail-tr-no-CS*:
‹((*RETURN o tl-trailt-tr-no-CS*), (*RETURN o tl*)) ∈
$[\lambda M.\ M \neq []]_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *trail-conv-to-no-CS* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-to-no-CS M* = *M*›

definition *trail-pol-conv-to-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-conv-to-no-CS M* = *M*›

lemma *id-trail-conv-to-no-CS*:
‹(*RETURN o trail-pol-conv-to-no-CS*, *RETURN o trail-conv-to-no-CS*) ∈ *trail-pol* $\mathcal{A} \rightarrow_f$ ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *trail-conv-back* :: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-back j M* = *M*›

definition (**in** −) *trail-conv-back-imp* :: ‹*nat* ⇒ *trail-pol* ⇒ *trail-pol nres*› **where**
‹*trail-conv-back-imp j* = (λ(*M*, *xs*, *lvls*, *reason*, -, *cs*). *do* {
ASSERT($j \leq$ *length cs*); *RETURN* (*M*, *xs*, *lvls*, *reason*, *j*, *take* (*nat-of-uint32-conv j*) *cs*)})›

lemma *trail-conv-back*:
‹(*uncurry trail-conv-back-imp*, *uncurry* (*RETURN oo trail-conv-back*))
∈ $[\lambda(k, M).\ k =$ *count-decided* $M]_f$ *nat-rel* $\times_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −)*take-arl* **where**
‹*take-arl* = (λ*i* (*xs*, *j*). (*xs*, *i*))›

lemma *isa-trail-nth-rev-trail-nth-no-CS*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) ∈
$[\lambda(M, i).\ i <$ *length* $M]_f$ *trail-pol-no-CS* $\mathcal{A} \times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *trail-pol-no-CS-alt-def*:
‹*trail-pol-no-CS* $\mathcal{A}$ =
{((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*), *M*). ((*M'*, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack (*take* (*count-decided M*) *cs*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M* ∧
length M < *uint32-max* ∧
length M ≤ *uint32-max div 2* + *1* ∧
count-decided M < *uint32-max* ∧
length M' = *length M* ∧
isasat-input-bounded $\mathcal{A}$ ∧
M' = *map lit-of* (*rev M*)
}›
⟨*proof*⟩

lemma *isa-length-trail-length-u-no-CS*:
‹$(RETURN\ o\ isa\text{-}length\text{-}trail,\ RETURN\ o\ length\text{-}uint32\text{-}nat) \in trail\text{-}pol\text{-}no\text{-}CS\ \mathcal{A} \rightarrow_f \langle nat\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

end
theory *Watched-Literals-VMTF*
imports *IsaSAT-Literals*
begin

0.1.7 Variable-Move-to-Front

Variants around head and last

definition *option-hd* :: ‹$'a\ list \Rightarrow 'a\ option$› **where**
‹$option\text{-}hd\ xs = (if\ xs = []\ then\ None\ else\ Some\ (hd\ xs))$›

lemma *option-hd-None-iff*[*iff*]: ‹$option\text{-}hd\ zs = None \longleftrightarrow zs = []$› ‹$None = option\text{-}hd\ zs \longleftrightarrow zs = []$›
⟨*proof*⟩

lemma *option-hd-Some-iff*[*iff*]: ‹$option\text{-}hd\ zs = Some\ y \longleftrightarrow (zs \neq [] \wedge y = hd\ zs)$›
‹$Some\ y = option\text{-}hd\ zs \longleftrightarrow (zs \neq [] \wedge y = hd\ zs)$›
⟨*proof*⟩

lemma *option-hd-Some-hd*[*simp*]: ‹$zs \neq [] \Longrightarrow option\text{-}hd\ zs = Some\ (hd\ zs)$›
⟨*proof*⟩

lemma *option-hd-Nil*[*simp*]: ‹$option\text{-}hd\ [] = None$›
⟨*proof*⟩

definition *option-last* **where**
‹$option\text{-}last\ l = (if\ l = []\ then\ None\ else\ Some\ (last\ l))$›

lemma
option-last-None-iff[*iff*]: ‹$option\text{-}last\ l = None \longleftrightarrow l = []$› ‹$None = option\text{-}last\ l \longleftrightarrow l = []$› **and**
option-last-Some-iff[*iff*]:
‹$option\text{-}last\ l = Some\ a \longleftrightarrow l \neq [] \wedge a = last\ l$›
‹$Some\ a = option\text{-}last\ l \longleftrightarrow l \neq [] \wedge a = last\ l$›
⟨*proof*⟩

lemma *option-last-Some*[*simp*]: ‹$l \neq [] \Longrightarrow option\text{-}last\ l = Some\ (last\ l)$›
⟨*proof*⟩

lemma *option-last-Nil*[*simp*]: ‹$option\text{-}last\ [] = None$›
⟨*proof*⟩

lemma *option-last-remove1-not-last*:
‹$x \neq last\ xs \Longrightarrow option\text{-}last\ xs = option\text{-}last\ (remove1\ x\ xs)$›
⟨*proof*⟩

lemma *option-hd-rev*: ‹$option\text{-}hd\ (rev\ xs) = option\text{-}last\ xs$›
⟨*proof*⟩

lemma *map-option-option-last*:

‹*map-option f* (*option-last xs*) = *option-last* (*map f xs*)›
⟨*proof*⟩

Specification

type-synonym *'v abs-vmtf-ns* = ‹*'v set* × *'v set*›
type-synonym *'v abs-vmtf-ns-remove* = ‹*'v abs-vmtf-ns* × *'v set*›

datatype (*'v*, *'n*) *vmtf-node* = *VMTF-Node* (*stamp* : *'n*) (*get-prev*: ‹*'v option*›) (*get-next*: ‹*'v option*›)
type-synonym *nat-vmtf-node* = ‹(*nat*, *nat*) *vmtf-node*›

inductive *vmtf-ns* :: ‹*nat list* ⇒ *nat* ⇒ *nat-vmtf-node list* ⇒ *bool*› **where**
Nil: ‹*vmtf-ns* [] *st xs*› |
Cons1: ‹*a* < *length xs* ⟹ *m* ≥ *n* ⟹ *xs* ! *a* = *VMTF-Node* (*n*::*nat*) *None None* ⟹ *vmtf-ns* [*a*] *m xs*›
|
Cons: ‹*vmtf-ns* (*b* # *l*) *m xs* ⟹ *a* < *length xs* ⟹ *xs* ! *a* = *VMTF-Node n None* (*Some b*) ⟹
a ≠ *b* ⟹ *a* ∉ *set l* ⟹ *n* > *m* ⟹
xs' = *xs*[*b* := *VMTF-Node* (*stamp* (*xs*!*b*)) (*Some a*) (*get-next* (*xs*!*b*))] ⟹ *n'* ≥ *n* ⟹
vmtf-ns (*a* # *b* # *l*) *n' xs'*›

inductive-cases *vmtf-nsE*: ‹*vmtf-ns xs st zs*›

lemma *vmtf-ns-le-length*: ‹*vmtf-ns l m xs* ⟹ *i* ∈ *set l* ⟹ *i* < *length xs*›
⟨*proof*⟩

lemma *vmtf-ns-distinct*: ‹*vmtf-ns l m xs* ⟹ *distinct l*›
⟨*proof*⟩

lemma *vmtf-ns-eq-iff*:
assumes
‹∀ *i* ∈ *set l*. *xs* ! *i* = *zs* ! *i*› **and**
‹∀ *i* ∈ *set l*. *i* < *length xs* ∧ *i* < *length zs*›
shows ‹*vmtf-ns l m zs* ⟷ *vmtf-ns l m xs*› (**is** ‹*?A* ⟷ *?B*›)
⟨*proof*⟩

lemmas *vmtf-ns-eq-iffI* = *vmtf-ns-eq-iff*[*THEN iffD1*]

lemma *vmtf-ns-stamp-increase*: ‹*vmtf-ns xs p zs* ⟹ *p* ≤ *p'* ⟹ *vmtf-ns xs p' zs*›
⟨*proof*⟩

lemma *vmtf-ns-single-iff*: ‹*vmtf-ns* [*a*] *m xs* ⟷ (*a* < *length xs* ∧ *m* ≥ *stamp* (*xs* ! *a*) ∧
xs ! *a* = *VMTF-Node* (*stamp* (*xs* ! *a*)) *None None*)›
⟨*proof*⟩

lemma *vmtf-ns-append-decomp*:
assumes ‹*vmtf-ns* (*axs* @ [*ax*, *ay*] @ *azs*) *an ns*›
shows ‹(*vmtf-ns* (*axs* @ [*ax*]) *an* (*ns*[*ax*:= *VMTF-Node* (*stamp* (*ns*!*ax*)) (*get-prev* (*ns*!*ax*)) *None*]) ∧
vmtf-ns (*ay* # *azs*) (*stamp* (*ns*!*ay*)) (*ns*[*ay*:= *VMTF-Node* (*stamp* (*ns*!*ay*)) *None* (*get-next* (*ns*!*ay*))])
∧
stamp (*ns*!*ax*) > *stamp* (*ns*!*ay*))›
⟨*proof*⟩

lemma *vmtf-ns-append-rebuild*:
assumes
‹(*vmtf-ns* (*axs* @ [*ax*]) *an ns*) › **and**
‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*ns*!*ay*)) *ns*› **and**

‹$stamp\ (ns!ax) > stamp\ (ns!ay)$› **and**
‹$distinct\ (axs\ @\ [ax,\ ay]\ @\ azs)$›
shows ‹$vmtf\text{-}ns\ (axs\ @\ [ax,\ ay]\ @\ azs)\ an$
$(ns[ax := VMTF\text{-}Node\ (stamp\ (ns!ax))\ (get\text{-}prev\ (ns!ax))\ (Some\ ay)\ ,$
$ay := VMTF\text{-}Node\ (stamp\ (ns!ay))\ (Some\ ax)\ (get\text{-}next\ (ns!ay))])$›
⟨*proof*⟩

It is tempting to remove the *update-x*. However, it leads to more complicated reasoning later: What happens if x is not in the list, but its successor is? Moreover, it is unlikely to really make a big difference (performance-wise).

definition *ns-vmtf-dequeue* :: ‹$nat \Rightarrow nat\text{-}vmtf\text{-}node\ list \Rightarrow nat\text{-}vmtf\text{-}node\ list$› **where**
‹$ns\text{-}vmtf\text{-}dequeue\ y\ xs =$
$(let\ x = xs\ !\ y;$
$u\text{-}prev =$
$(case\ get\text{-}prev\ x\ of\ None \Rightarrow xs$
$|\ Some\ a \Rightarrow xs[a{:=}\ VMTF\text{-}Node\ (stamp\ (xs!a))\ (get\text{-}prev\ (xs!a))\ (get\text{-}next\ x)]);$
$u\text{-}next =$
$(case\ get\text{-}next\ x\ of\ None \Rightarrow u\text{-}prev$
$|\ Some\ a \Rightarrow u\text{-}prev[a{:=}\ VMTF\text{-}Node\ (stamp\ (u\text{-}prev!a))\ (get\text{-}prev\ x)\ (get\text{-}next\ (u\text{-}prev!a))]);$
$u\text{-}x = u\text{-}next[y{:=}\ VMTF\text{-}Node\ (stamp\ (u\text{-}next!y))\ None\ None]$
in
$u\text{-}x)$
›

lemma *vmtf-ns-different-same-neq*: ‹$vmtf\text{-}ns\ (b\ \#\ c\ \#\ l')\ m\ xs \Longrightarrow vmtf\text{-}ns\ (c\ \#\ l')\ m\ xs \Longrightarrow False$›
⟨*proof*⟩

lemma *vmtf-ns-last-next*:
‹$vmtf\text{-}ns\ (xs\ @\ [x])\ m\ ns \Longrightarrow get\text{-}next\ (ns\ !\ x) = None$›
⟨*proof*⟩

lemma *vmtf-ns-hd-prev*:
‹$vmtf\text{-}ns\ (x\ \#\ xs)\ m\ ns \Longrightarrow get\text{-}prev\ (ns\ !\ x) = None$›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-next*:
‹$vmtf\text{-}ns\ (xs\ @\ [x,\ y]\ @\ zs)\ m\ ns \Longrightarrow get\text{-}next\ (ns\ !\ x) = Some\ y$›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-next-option-hd*:
‹$vmtf\text{-}ns\ (xs\ @\ x\ \#\ zs)\ m\ ns \Longrightarrow get\text{-}next\ (ns\ !\ x) = option\text{-}hd\ zs$›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-prev*:
assumes ‹$vmtf\text{-}ns\ (xs\ @\ [x,\ y]\ @\ zs)\ m\ ns$›
shows ‹$get\text{-}prev\ (ns\ !\ y) = Some\ x$›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-prev-option-last*:
‹$vmtf\text{-}ns\ (xs\ @\ x\ \#\ zs)\ m\ ns \Longrightarrow get\text{-}prev\ (ns\ !\ x) = option\text{-}last\ xs$›
⟨*proof*⟩

lemma *length-ns-vmtf-dequeue*[*simp*]: ‹$length\ (ns\text{-}vmtf\text{-}dequeue\ x\ ns) = length\ ns$›
⟨*proof*⟩

lemma *vmtf-ns-skip-fst*:
  assumes *vmtf-ns*: ‹$vmtf\text{-}ns\ (x\ \#\ y'\ \#\ zs')\ m\ ns$›
  shows ‹$\exists\, n.\ vmtf\text{-}ns\ (y'\ \#\ zs')\ n\ (ns[y' := VMTF\text{-}Node\ (stamp\ (ns\ !\ y'))\ None\ (get\text{-}next\ (ns\ !\ y'))]) \wedge m \geq n$›
  ⟨*proof*⟩

definition *vmtf-ns-notin* **where**
  ‹$vmtf\text{-}ns\text{-}notin\ l\ m\ xs \longleftrightarrow (\forall\, i{<}length\ xs.\ i{\notin}set\ l \longrightarrow (get\text{-}prev\ (xs\ !\ i) = None \wedge get\text{-}next\ (xs\ !\ i) = None))$›

lemma *vmtf-ns-notinI*:
  ‹$(\bigwedge i.\ i\ {<}length\ xs \Longrightarrow i{\notin}set\ l \Longrightarrow get\text{-}prev\ (xs\ !\ i) = None \wedge get\text{-}next\ (xs\ !\ i) = None) \Longrightarrow vmtf\text{-}ns\text{-}notin\ l\ m\ xs$›
  ⟨*proof*⟩

lemma *stamp-ns-vmtf-dequeue*:
  ‹$axs < length\ zs \Longrightarrow stamp\ (ns\text{-}vmtf\text{-}dequeue\ x\ zs\ !\ axs) = stamp\ (zs\ !\ axs)$›
  ⟨*proof*⟩

lemma *sorted-many-eq-append*: ‹$sorted\ (xs\ @\ [x,\ y]) \longleftrightarrow sorted\ (xs\ @\ [x]) \wedge x \leq y$›
  ⟨*proof*⟩

lemma *vmtf-ns-stamp-sorted*:
  assumes ‹$vmtf\text{-}ns\ l\ m\ ns$›
  shows ‹$sorted\ (map\ (\lambda a.\ stamp\ (ns!a))\ (rev\ l)) \wedge (\forall\, a \in set\ l.\ stamp\ (ns!a) \leq m)$›
  ⟨*proof*⟩

lemma *vmtf-ns-ns-vmtf-dequeue*:
  assumes *vmtf-ns*: ‹$vmtf\text{-}ns\ l\ m\ ns$› **and** *notin*: ‹$vmtf\text{-}ns\text{-}notin\ l\ m\ ns$› **and** *valid*: ‹$x < length\ ns$›
  shows ‹$vmtf\text{-}ns\ (remove1\ x\ l)\ m\ (ns\text{-}vmtf\text{-}dequeue\ x\ ns)$›
⟨*proof*⟩

lemma *vmtf-ns-hd-next*:
  ‹$vmtf\text{-}ns\ (x\ \#\ a\ \#\ list)\ m\ ns \Longrightarrow get\text{-}next\ (ns\ !\ x) = Some\ a$›
  ⟨*proof*⟩

lemma *vmtf-ns-notin-dequeue*:
  assumes *vmtf-ns*: ‹$vmtf\text{-}ns\ l\ m\ ns$› **and** *notin*: ‹$vmtf\text{-}ns\text{-}notin\ l\ m\ ns$› **and** *valid*: ‹$x < length\ ns$›
  shows ‹$vmtf\text{-}ns\text{-}notin\ (remove1\ x\ l)\ m\ (ns\text{-}vmtf\text{-}dequeue\ x\ ns)$›
⟨*proof*⟩

lemma *vmtf-ns-stamp-distinct*:
  assumes ‹$vmtf\text{-}ns\ l\ m\ ns$›
  shows ‹$distinct\ (map\ (\lambda a.\ stamp\ (ns!a))\ l)$›
  ⟨*proof*⟩

lemma *vmtf-ns-thighten-stamp*:
  assumes *vmtf-ns*: ‹$vmtf\text{-}ns\ l\ m\ xs$› **and** *n*: ‹$\forall\, a{\in}set\ l.\ stamp\ (xs\ !\ a) \leq n$›
  shows ‹$vmtf\text{-}ns\ l\ n\ xs$›
⟨*proof*⟩

lemma *vmtf-ns-rescale*:
  assumes
    ‹$vmtf\text{-}ns\ l\ m\ xs$› **and**
    ‹$sorted\ (map\ (\lambda a.\ st\ !\ a)\ (rev\ l))$› **and** ‹$distinct\ (map\ (\lambda a.\ st\ !\ a)\ l)$›
    ‹$\forall\, a \in set\ l.\ get\text{-}prev\ (zs\ !\ a) = get\text{-}prev\ (xs\ !\ a)$› **and**

⟨∀ a ∈ *set l*. *get-next* (*zs* ! *a*) = *get-next* (*xs* ! *a*)⟩ **and**
⟨∀ a ∈ *set l*. *stamp* (*zs* ! *a*) = *st* ! *a*⟩ **and**
⟨*length xs* ≤ *length zs*⟩ **and**
⟨∀ *a*∈*set l*. *a* < *length st*⟩ **and**
m': ⟨∀ *a* ∈ *set l*. *st* ! *a* < *m'*⟩
shows ⟨*vmtf-ns l m' zs*⟩
⟨*proof*⟩

lemma *vmtf-ns-last-prev*:
assumes *vmtf*: ⟨*vmtf-ns* (*xs* @ [*x*]) *m ns*⟩
shows ⟨*get-prev* (*ns* ! *x*) = *option-last xs*⟩
⟨*proof*⟩

Abstract Invariants Invariants

- The atoms of *xs* and *ys* are always disjoint.
- The atoms of *ys* are *always* set.
- The atoms of *xs* *can* be set but do not have to.
- The atoms of *zs* are either in *xs* and *ys*.

definition *vmtf-*$\mathcal{L}_{all}$ :: ⟨*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat abs-vmtf-ns-remove* ⇒ *bool*⟩ **where**
⟨*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ≡ λ((*xs*, *ys*), *zs*).
(∀ *L*∈*ys*. *L* ∈ *atm-of* ‘ *lits-of-l M*) ∧
xs ∩ *ys* = {} ∧
zs ⊆ *xs* ∪ *ys* ∧
xs ∪ *ys* = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)
⟩

abbreviation *abs-vmtf-ns-inv* :: ⟨*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat abs-vmtf-ns* ⇒ *bool*⟩ **where**
⟨*abs-vmtf-ns-inv* $\mathcal{A}$ *M vm* ≡ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*vm*, {})⟩

Implementation

type-synonym (**in** −) *vmtf* = ⟨*nat-vmtf-node list* × *nat* × *nat* × *nat* × *nat option*⟩
type-synonym (**in** −) *vmtf-remove-int* = ⟨*vmtf* × *nat set*⟩

We use the opposite direction of the VMTF paper: The latest added element *fst-As* is at the beginning.

definition *vmtf* :: ⟨*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int set*⟩ **where**
⟨*vmtf* $\mathcal{A}$ *M* = {((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(∃ *xs'* *ys'*.
vmtf-ns (*ys'* @ *xs'*) *m ns* ∧ *fst-As* = *hd* (*ys'* @ *xs'*) ∧ *lst-As* = *last* (*ys'* @ *xs'*)
∧ *next-search* = *option-hd xs'*
∧ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)
∧ *vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*
∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧ (∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))
)}⟩

lemma *vmtf-consD*:
assumes *vmtf*: ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M*⟩
shows ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ (*L* # *M*)⟩
⟨*proof*⟩

type-synonym (**in** −) *vmtf-option-fst-As* = ‹*nat-vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*›

definition (**in** −) *vmtf-dequeue* :: ‹*nat* ⇒ *vmtf* ⇒ *vmtf-option-fst-As*› **where**
‹*vmtf-dequeue* ≡ (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
  (*let fst-As′* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*);
    next-search′ = *if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*;
    lst-As′ = *if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As in*
  (*ns-vmtf-dequeue L ns*, *m*, *fst-As′*, *lst-As′*, *next-search′*)))›

It would be better to distinguish whether L is set in M or not.

definition *vmtf-enqueue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf*› **where**
‹*vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
  (*case fst-As of*
    None ⇒ (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
      (*if defined-lit M* (*Pos L*) *then None else Some L*))
  | *Some fst-As* ⇒
    let fst-As′ = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*)) *in*
    (*ns*[*L* := *VMTF-Node* (*m*+*1*) *None* (*Some fst-As*), *fst-As* := *fst-As′*],
      m+*1*, *L*, *the lst-As*, (*if defined-lit M* (*Pos L*) *then next-search else Some L*))))›

definition (**in** −) *vmtf-en-dequeue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf*› **where**
‹*vmtf-en-dequeue* = (λ*M L vm*. *vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *abs-vmtf-ns-bump-vmtf-dequeue*:
  fixes *M*
  assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
    L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
    dequeue: ‹(*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*) =
      vmtf-dequeue L (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)› **and**
    $\mathcal{A}_{in}$*-nempty*: ‹*isasat-input-nempty* $\mathcal{A}$›
  shows ‹∃ *xs′ ys′*. *vmtf-ns* (*ys′* @ *xs′*) *m′ ns′* ∧ *fst-As′* = *option-hd* (*ys′* @ *xs′*)
  ∧ *lst-As′* = *option-last* (*ys′* @ *xs′*)
  ∧ *next-search′* = *option-hd xs′*
  ∧ *next-search′* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)
  ∧ *vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set xs′*), *set ys′*), *to-remove*)
  ∧ *vmtf-ns-notin* (*ys′* @ *xs′*) *m′ ns′* ∧
  L ∉ *set* (*ys′* @ *xs′*) ∧ (∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
  ⟨*proof*⟩

lemma *vmtf-ns-get-prev-not-itself*:
  ‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-prev* (*ns* ! *L*) ≠ *Some L*›
  ⟨*proof*⟩

lemma *vmtf-ns-get-next-not-itself*:
  ‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-next* (*ns* ! *L*) ≠ *Some L*›
  ⟨*proof*⟩

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue*:
  fixes *M*
  assumes
    vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
    L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
    to-remove: ‹*to-remove′* ⊆ *to-remove* − {*L*}› **and**
    nempty: ‹*isasat-input-nempty* $\mathcal{A}$›

shows ‹(*vmtf-en-dequeue M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*′) ∈ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue*′:
fixes *M*
assumes
vmtf: ‹(*vm*, *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
to-remove: ‹*to-remove*′ ⊆ *to-remove* − {*L*}› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹(*vmtf-en-dequeue M L vm*, *to-remove*′) ∈ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

definition (**in** −) *vmtf-unset* :: ‹*nat* ⇒ *vmtf-remove-int* ⇒ *vmtf-remove-int*› **where**
‹*vmtf-unset* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(*if next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)
then ((*ns*, *m*, *fst-As*, *lst-As*, *Some L*), *to-remove*)
else ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)))›

lemma *vmtf-atm-of-ys-iff*:
assumes
vmtf-ns: ‹*vmtf-ns* (*ys*′ @ *xs*′) *m ns*› **and**
next-search: ‹*next-search* = *option-hd xs*′› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs*′, *set ys*′), *to-remove*)› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
shows ‹*L* ∈ *set ys*′ ⟷ *next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)›
⟨*proof*⟩

lemma *vmtf*-$\mathcal{L}_{all}$-*to-remove-mono*:
assumes
‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*a*, *b*), *to-remove*)› **and**
‹*to-remove*′ ⊆ *to-remove*›
shows ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*a*, *b*), *to-remove*′)›
⟨*proof*⟩

lemma *abs-vmtf-ns-unset-vmtf-unset*:
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
to-remove: ‹*to-remove*′ ⊆ *to-remove*›
shows ‹(*vmtf-unset L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*′)) ∈ *vmtf* $\mathcal{A}$ *M*› (**is** ‹*?S* ∈ -›)
⟨*proof*⟩

definition (**in** −) *vmtf-dequeue-pre* **where**
‹*vmtf-dequeue-pre* = (λ(*L*,*ns*). *L* < *length ns* ∧
(*get-next* (*ns*!*L*) ≠ *None* ⟶ *the* (*get-next* (*ns*!*L*)) < *length ns*) ∧
(*get-prev* (*ns*!*L*) ≠ *None* ⟶ *the* (*get-prev* (*ns*!*L*)) < *length ns*))›

lemma (**in** −) *vmtf-dequeue-pre-alt-def*:
‹*vmtf-dequeue-pre* = (λ(*L*, *ns*). *L* < *length ns* ∧
(∀ *a*. *Some a* = *get-next* (*ns*!*L*) ⟶ *a* < *length ns*) ∧
(∀ *a*. *Some a* = *get-prev* (*ns*!*L*) ⟶ *a* < *length ns*))›
⟨*proof*⟩

definition *vmtf-en-dequeue-pre* :: ‹*nat multiset* ⇒ ((*nat*, *nat*) *ann-lits* × *nat*) × *vmtf* ⇒ *bool*› **where**

‹vmtf-en-dequeue-pre $\mathcal{A}$ = (λ((M, L),(ns,m,fst-As, lst-As, next-search)).
L < length ns ∧ vmtf-dequeue-pre (L, ns) ∧
fst-As < length ns ∧ (get-next (ns ! fst-As) ≠ None ⟶ get-prev (ns ! lst-As) ≠ None) ∧
(get-next (ns ! fst-As) = None ⟶ fst-As = lst-As) ∧
m+1 ≤ uint64-max ∧
Pos L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)›

lemma (**in** −) *id-reorder-list*:
‹(RETURN o id, reorder-list vm) ∈ ⟨nat-rel⟩list-rel $\rightarrow_f$ ⟨⟨nat-rel⟩list-rel⟩nres-rel›
⟨proof⟩

lemma *vmtf-vmtf-en-dequeue-pre-to-remove*:
assumes *vmtf*: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf $\mathcal{A}$ M› **and**
i: ‹A ∈ to-remove› **and**
m-le: ‹m + 1 ≤ uint64-max› **and**
nempty: ‹isasat-input-nempty $\mathcal{A}$›
shows ‹vmtf-en-dequeue-pre $\mathcal{A}$ ((M, A), (ns, m, fst-As, lst-As, next-search))›
⟨proof⟩

lemma *vmtf-vmtf-en-dequeue-pre-to-remove'*:
assumes *vmtf*: ‹(vm, to-remove) ∈ vmtf $\mathcal{A}$ M› **and**
i: ‹A ∈ to-remove› **and** ‹fst (snd vm) + 1 ≤ uint64-max› **and**
A: ‹isasat-input-nempty $\mathcal{A}$›
shows ‹vmtf-en-dequeue-pre $\mathcal{A}$ ((M, A), vm)›
⟨proof⟩

lemma *wf-vmtf-get-next*:
assumes *vmtf*: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf $\mathcal{A}$ M›
shows ‹wf {(get-next (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of ($\mathcal{L}_{all}$ $\mathcal{A}$)}› (**is** ‹wf ?R›)
⟨proof⟩

lemma *vmtf-next-search-take-next*:
assumes
vmtf: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf $\mathcal{A}$ M› **and**
n: ‹next-search ≠ None› **and**
def-n: ‹defined-lit M (Pos (the next-search))›
shows ‹((ns, m, fst-As, lst-As, get-next (ns!the next-search)), to-remove) ∈ vmtf $\mathcal{A}$ M›
⟨proof⟩

definition *vmtf-find-next-undef* :: ‹nat multiset ⇒ vmtf-remove-int ⇒ (nat, nat) ann-lits ⇒ (nat option) nres› **where**
‹vmtf-find-next-undef $\mathcal{A}$ = (λ((ns, m, fst-As, lst-As, next-search), to-remove) M. do {
$WHILE_T$$^{\lambda next\text{-}search.\ ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove) \in vmtf\ \mathcal{A}\ M\ \wedge\ \ \ \ \ \ (next\text{-}search \neq None \longrightarrow Pos\ (}$
(λnext-search. next-search ≠ None ∧ defined-lit M (Pos (the next-search)))
(λnext-search. do {
ASSERT(next-search ≠ None);
let n = the next-search;
ASSERT(Pos n ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
ASSERT (n < length ns);
RETURN (get-next (ns!n))
}
)
next-search
})›

lemma *vmtf-find-next-undef-ref*:
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*›
shows ‹*vmtf-find-next-undef* $\mathcal{A}$ ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) *M*
$\leq \Downarrow$ *Id* (*SPEC* (λ*L*. ((*ns*, *m*, *fst-As*, *lst-As*, *L*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M* $\wedge$
(*L* = *None* $\longrightarrow$ ($\forall$ *L*$\in$#$\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit* *M* *L*)) $\wedge$
(*L* $\neq$ *None* $\longrightarrow$ *Pos* (*the* *L*) $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *undefined-lit* *M* (*Pos* (*the* *L*)))))›
⟨*proof*⟩

definition *vmtf-mark-to-rescore*
:: ‹*nat* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$ *vmtf-remove-int*›
where
‹*vmtf-mark-to-rescore* *L* = (λ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
((*ns*, *m*, *fst-As*, *next-search*), *insert* *L* *to-remove*))›

lemma *vmtf-mark-to-rescore*:
assumes
L: ‹*L* $\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*›
shows ‹*vmtf-mark-to-rescore* *L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

lemma *vmtf-unset-vmtf-tl*:
fixes *M*
defines [*simp*]: ‹*L* $\equiv$ *atm-of* (*lit-of* (*hd* *M*))›
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* $\neq$ []›
shows ‹(*vmtf-unset* *L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)) $\in$ *vmtf* $\mathcal{A}$ (*tl* *M*)›
(**is** ‹*?S* $\in$ -›)
⟨*proof*⟩

definition *vmtf-mark-to-rescore-and-unset* :: ‹*nat* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$ *vmtf-remove-int*› **where**
‹*vmtf-mark-to-rescore-and-unset* *L* *M* = *vmtf-mark-to-rescore* *L* (*vmtf-unset* *L* *M*)›

lemma *vmtf-append-remove-iff*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *insert* *L* *b*) $\in$ *vmtf* $\mathcal{A}$ *M* $\longleftrightarrow$
L $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\wedge$ ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *b*) $\in$ *vmtf* $\mathcal{A}$ *M*›
(**is** ‹*?A* $\longleftrightarrow$ *?L* $\wedge$ *?B*›)
⟨*proof*⟩

lemma *vmtf-append-remove-iff'*:
‹(*vm*, *insert* *L* *b*) $\in$ *vmtf* $\mathcal{A}$ *M* $\longleftrightarrow$
L $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\wedge$ (*vm*, *b*) $\in$ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore-unset*:
fixes *M*
defines [*simp*]: ‹*L* $\equiv$ *atm-of* (*lit-of* (*hd* *M*))›
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* $\neq$ []›
shows ‹(*vmtf-mark-to-rescore-and-unset* *L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)) $\in$ *vmtf* $\mathcal{A}$ (*tl* *M*)›
(**is** ‹*?S* $\in$ -›)
⟨*proof*⟩

lemma *vmtf-insert-sort-nth-code-preD*:
  assumes *vmtf*: ‹$vm \in vmtf\ \mathcal{A}\ M$›
  shows ‹$\forall x{\in}snd\ vm.\ x < length\ (fst\ (fst\ vm))$›
⟨*proof*⟩

lemma *vmtf-ns-Cons*:
  assumes
    vmtf: ‹$vmtf\text{-}ns\ (b\ \#\ l)\ m\ xs$› **and**
    a-xs: ‹$a < length\ xs$› **and**
    ab: ‹$a \neq b$› **and**
    a-l: ‹$a \notin set\ l$› **and**
    nm: ‹$n > m$› **and**
    xs': ‹$xs' = xs[a := VMTF\text{-}Node\ n\ None\ (Some\ b),$
        $b := VMTF\text{-}Node\ (stamp\ (xs!b))\ (Some\ a)\ (get\text{-}next\ (xs!b))]$› **and**
    nn': ‹$n' \geq n$›
  shows ‹$vmtf\text{-}ns\ (a\ \#\ b\ \#\ l)\ n'\ xs'$›
⟨*proof*⟩

definition (**in** −) *vmtf-cons* **where**
‹*vmtf-cons ns L cnext st* =
  (*let*
    $ns = ns[L := VMTF\text{-}Node\ (Suc\ st)\ None\ cnext];$
    $ns = (case\ cnext\ of\ None \Rightarrow ns$
      $\mid Some\ cnext \Rightarrow ns[cnext := VMTF\text{-}Node\ (stamp\ (ns!cnext))\ (Some\ L)\ (get\text{-}next\ (ns!cnext))])$ *in*
  ns)
›

lemma *vmtf-notin-vmtf-cons*:
  assumes
    vmtf-ns: ‹$vmtf\text{-}ns\text{-}notin\ xs\ m\ ns$› **and**
    cnext: ‹$cnext = option\text{-}hd\ xs$› **and**
    L-xs: ‹$L \notin set\ xs$›
  shows
    ‹$vmtf\text{-}ns\text{-}notin\ (L\ \#\ xs)\ (Suc\ m)\ (vmtf\text{-}cons\ ns\ L\ cnext\ m)$›
⟨*proof*⟩

lemma *vmtf-cons*:
  assumes
    vmtf-ns: ‹$vmtf\text{-}ns\ xs\ m\ ns$› **and**
    cnext: ‹$cnext = option\text{-}hd\ xs$› **and**
    L-A: ‹$L < length\ ns$› **and**
    L-xs: ‹$L \notin set\ xs$›
  shows
    ‹$vmtf\text{-}ns\ (L\ \#\ xs)\ (Suc\ m)\ (vmtf\text{-}cons\ ns\ L\ cnext\ m)$›
⟨*proof*⟩

lemma *length-vmtf-cons*[*simp*]: ‹$length\ (vmtf\text{-}cons\ ns\ L\ n\ m) = length\ ns$›
  ⟨*proof*⟩

lemma *wf-vmtf-get-prev*:
  assumes *vmtf*: ‹$((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove) \in vmtf\ \mathcal{A}\ M$›
  shows ‹$wf\ \{(get\text{-}prev\ (ns\ !\ the\ a), a)\ |a.\ a \neq None \wedge the\ a \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})\}$› (**is** ‹*wf ?R*›)
⟨*proof*⟩

fun *update-stamp* **where**

‹*update-stamp xs n a* = *xs*[*a* := *VMTF-Node n* (*get-prev* (*xs*!*a*)) (*get-next* (*xs*!*a*))]›

definition *vmtf-rescale* :: ‹*vmtf* ⇒ *vmtf nres*› **where**
‹*vmtf-rescale* = (λ(*ns*, *m*, *fst-As*, *lst-As* :: *nat*, *next-search*). *do* {
  (*ns*, *m*, -) ← $WHILE_T{}^{\lambda\text{-}.\ True}$
    (λ(*ns*, *n*, *lst-As*). *lst-As* ≠*None*)
    (λ(*ns*, *n*, *a*). *do* {
      ASSERT(*a* ≠ *None*);
      ASSERT(*n*+*1* ≤ *uint32-max*);
      ASSERT(*the a* < *length ns*);
      RETURN (*update-stamp ns n* (*the a*), *n*+*1*, *get-prev* (*ns* ! *the a*))
    })
    (*ns*, *0*, *Some lst-As*);
  RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))
  })
›

lemma *vmtf-rescale-vmtf*:
  assumes *vmtf*: ‹(*vm*, *to-remove*) ∈ *vmtf* 𝒜 *M*› **and**
    nempty: ‹*isasat-input-nempty* 𝒜› **and**
    bounded: ‹*isasat-input-bounded* 𝒜›
  shows
    ‹*vmtf-rescale vm* ≤ *SPEC* (λ*vm*. (*vm*, *to-remove*) ∈ *vmtf* 𝒜 *M* ∧ *fst* (*snd vm*) ≤ *uint32-max*)›
    (**is** ‹*?A* ≤ *?R*›)
⟨*proof*⟩

definition *vmtf-flush*
  :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf-remove-int* ⇒ *vmtf-remove-int nres*›
where
  ‹*vmtf-flush* $\mathcal{A}_{in}$ = (λ*M* (*vm*, *to-remove*). *RES* (*vmtf* $\mathcal{A}_{in}$ *M*))›

definition *atoms-hash-rel* :: ‹*nat multiset* ⇒ (*bool list* × *nat set*) *set*› **where**
  ‹*atoms-hash-rel* 𝒜 = {(*C*, *D*). (∀ *L* ∈ *D*. *L* < *length C*) ∧ (∀ *L* < *length C*. *C* ! *L* ⟷ *L* ∈ *D*) ∧
    (∀ *L* ∈# 𝒜. *L* < *length C*) ∧ *D* ⊆ *set-mset* 𝒜}›

definition *distinct-hash-atoms-rel*
  :: ‹*nat multiset* ⇒ ((*'v list* × *'v set*) × *'v set*) *set*›
where
  ‹*distinct-hash-atoms-rel* 𝒜 = {((*C*, *h*), *D*). *set C* = *D* ∧ *h* = *D* ∧ *distinct C*}›

definition *distinct-atoms-rel*
  :: ‹*nat multiset* ⇒ ((*nat list* × *bool list*) × *nat set*) *set*›
where
  ‹*distinct-atoms-rel* 𝒜 = (*Id* $\times_r$ *atoms-hash-rel* 𝒜) *O distinct-hash-atoms-rel* 𝒜›

lemma *distinct-atoms-rel-alt-def*:
  ‹*distinct-atoms-rel* 𝒜 = {((*D'*, *C*), *D*). (∀ *L* ∈ *D*. *L* < *length C*) ∧ (∀ *L* < *length C*. *C* ! *L* ⟷ *L* ∈ *D*) ∧
    (∀ *L* ∈# 𝒜. *L* < *length C*) ∧ *set D'* = *D* ∧ *distinct D'* ∧ *set D'* ⊆ *set-mset* 𝒜}›
  ⟨*proof*⟩

lemma *distinct-atoms-rel-empty-hash-iff*:
  ‹(([], *h*), {}) ∈ *distinct-atoms-rel* 𝒜 ⟷ (∀ *L* ∈# 𝒜. *L* < *length h*) ∧ (∀ *i*∈*set h*. *i* = *False*)›
  ⟨*proof*⟩

definition *atoms-hash-del-pre* **where**
‹$atoms\text{-}hash\text{-}del\text{-}pre\ i\ xs = (i < length\ xs)$›

definition *atoms-hash-del* **where**
‹$atoms\text{-}hash\text{-}del\ i\ xs = xs[i := False]$›

definition *vmtf-flush-int* :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ - ⇒ - *nres*› **where**
‹$vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in} = (\lambda M\ (vm,\ (to\text{-}remove,\ h)).\ do\ \{$
$ASSERT(\forall x{\in}set\ to\text{-}remove.\ x < length\ (fst\ vm));$
$ASSERT(length\ to\text{-}remove \le uint32\text{-}max);$
$to\text{-}remove' \leftarrow reorder\text{-}list\ vm\ to\text{-}remove;$
$ASSERT(length\ to\text{-}remove' \le uint32\text{-}max);$
$vm \leftarrow (if\ length\ to\text{-}remove' + fst\ (snd\ vm) \ge uint64\text{-}max$
$then\ vmtf\text{-}rescale\ vm\ else\ RETURN\ vm);$
$ASSERT(length\ to\text{-}remove' + fst\ (snd\ vm) \le uint64\text{-}max);$
$(\text{-},\ vm,\ h) \leftarrow WHILE_T{}^{\lambda(i,\ vm',\ h).\ i \le length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge \quad (i < length\ to\text{-}remove}$
$(\lambda(i,\ vm,\ h).\ i < length\ to\text{-}remove')$
$(\lambda(i,\ vm,\ h).\ do\ \{$
$ASSERT(i < length\ to\text{-}remove');$
$ASSERT(to\text{-}remove'!i \in\#\ \mathcal{A}_{in});$
$ASSERT(atoms\text{-}hash\text{-}del\text{-}pre\ (to\text{-}remove'!i)\ h);$
$RETURN\ (i{+}1,\ vmtf\text{-}en\text{-}dequeue\ M\ (to\text{-}remove'!i)\ vm,\ atoms\text{-}hash\text{-}del\ (to\text{-}remove'!i)\ h)\})$
$(0,\ vm,\ h);$
$RETURN\ (vm,\ (emptied\text{-}list\ to\text{-}remove',\ h))$
$\})$›

lemma *vmtf-change-to-remove-order*:
assumes
vmtf: ‹$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove) \in vmtf\ \mathcal{A}_{in}\ M$› **and**
CD-rem: ‹$((C,\ D),\ to\text{-}remove) \in distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}$› **and**
nempty: ‹$isasat\text{-}input\text{-}nempty\ \mathcal{A}_{in}$› **and**
bounded: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}$›
shows ‹$vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in}\ M\ ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ (C,\ D))$
$\le \Downarrow(Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in})$
$(vmtf\text{-}flush\ \mathcal{A}_{in}\ M\ ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove))$›
⟨*proof*⟩

lemma *vmtf-change-to-remove-order'*:
‹$(uncurry\ (vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in}),\ uncurry\ (vmtf\text{-}flush\ \mathcal{A}_{in})) \in$
$[\lambda(M,\ vm).\ vm \in vmtf\ \mathcal{A}_{in}\ M \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}nempty\ \mathcal{A}_{in}]_f$
$Id \times_r (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}) \rightarrow \langle(Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in})\rangle\ nres\text{-}rel$›
⟨*proof*⟩

0.1.8 Phase saving

type-synonym *phase-saver* = ‹*bool list*›

definition *phase-saving* :: ‹*nat multiset* ⇒ *phase-saver* ⇒ *bool*› **where**
‹$phase\text{-}saving\ \mathcal{A}\ \varphi \longleftrightarrow (\forall L{\in}atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}).\ L < length\ \varphi)$›

Save phase as given (e.g. for literals in the trail):

definition *save-phase* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
‹$save\text{-}phase\ L\ \varphi = \varphi[atm\text{-}of\ L := is\text{-}pos\ L]$›

lemma *phase-saving-save-phase*[*simp*]:
‹*phase-saving* $\mathcal{A}$ (*save-phase* L φ) $\longleftrightarrow$ *phase-saving* $\mathcal{A}$ φ›
⟨*proof*⟩

Save opposite of the phase (e.g. for literals in the conflict clause):

definition *save-phase-inv* :: ‹*nat literal* $\Rightarrow$ *phase-saver* $\Rightarrow$ *phase-saver*› **where**
‹*save-phase-inv* L φ = φ[*atm-of* L := $\neg$*is-pos* L]›

end
theory *LBD*
imports *Watched-Literals.WB-Word IsaSAT-Literals*
begin

LBD

LBD (literal block distance) or glue is a measure of usefulness of clauses: It is the number of different levels involved in a clause. This measure has been introduced by Glucose in 2009 (Audemart and Simon).

LBD has also another advantage, explaining why we implemented it even before working on restarts: It can speed the conflict minimisation. Indeed a literal might be redundant only if there is a literal of the same level in the conflict.

The LBD data structure is well-suited to do so: We mark every level that appears in the conflict in a hash-table like data structure.

Types and relations **type-synonym** *lbd* = ‹*bool list*›
type-synonym *lbd-ref* = ‹*bool list* × *nat* × *nat*›
type-synonym *lbd-assn* = ‹*bool array* × *uint32* × *uint32*›

Beside the actual "lookup" table, we also keep the highest level marked so far to unmark all levels faster (but we currently don't save the LBD and have to iterate over the data structure). We also handle growing of the structure by hand instead of using a proper hash-table. We do so, because there are much stronger guarantees on the key that there is in a general hash-table (especially, our numbers are all small).

definition *lbd-ref* **where**
‹*lbd-ref* = {((*lbd*, *n*, *m*), *lbd'*). *lbd* = *lbd'* $\wedge$ *n* < *length lbd* $\wedge$
($\forall$ *k* > *n*. *k* < *length lbd* $\longrightarrow$ $\neg$*lbd*!*k*) $\wedge$
length lbd $\leq$ *Suc* (*Suc* (*uint-max div 2*)) $\wedge$ *n* < *length lbd* $\wedge$
m = *length* (*filter id lbd*)}›

Testing if a level is marked **definition** *level-in-lbd* :: ‹*nat* $\Rightarrow$ *lbd* $\Rightarrow$ *bool*› **where**
‹*level-in-lbd* *i* = (λ*lbd*. *i* < *length lbd* $\wedge$ *lbd*!*i*)›

definition *level-in-lbd-ref* :: ‹*nat* $\Rightarrow$ *lbd-ref* $\Rightarrow$ *bool*› **where**
‹*level-in-lbd-ref* = (λ*i* (*lbd*, -). *i* < *length-uint32-nat lbd* $\wedge$ *lbd*!*i*)›

lemma *level-in-lbd-ref-level-in-lbd*:
‹(*uncurry* (*RETURN oo level-in-lbd-ref*), *uncurry* (*RETURN oo level-in-lbd*)) $\in$
nat-rel $\times_r$ *lbd-ref* $\rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

Marking more levels **definition** *list-grow* **where**
‹*list-grow xs n x* = *xs* @ *replicate* (*n* − *length xs*) *x*›

definition *lbd-write* :: ‹*lbd* ⇒ *nat* ⇒ *lbd*› **where**
‹*lbd-write* = (λ*lbd i*.
(*if i* < *length lbd then* (*lbd*[*i* := *True*])
else ((*list-grow lbd* (*i* + *1*) *False*)[*i* := *True*])))›

definition *lbd-ref-write* :: ‹*lbd-ref* ⇒ *nat* ⇒ *lbd-ref nres*› **where**
‹*lbd-ref-write* = (λ(*lbd*, *m*, *n*) *i*. *do* {
ASSERT(*length lbd* ≤ *uint-max* ∧ *n* + *1* ≤ *uint-max*);
(*if i* < *length-uint32-nat lbd then*
let n = *if lbd* ! *i then n else n*+*one-uint32-nat in*
RETURN (*lbd*[*i* := *True*], *max i m*, *n*)
else do {
ASSERT(*i* + *1* ≤ *uint-max*);
RETURN ((*list-grow lbd* (*i* + *one-uint32-nat*) *False*)[*i* := *True*], *max i m*, *n* + *one-uint32-nat*)
})
})›

lemma *length-list-grow*[*simp*]:
‹*length* (*list-grow xs n a*) = *max* (*length xs*) *n*›
⟨*proof*⟩

lemma *list-update-append2*: ‹*i* ≥ *length xs* ⟹ (*xs* @ *ys*)[*i* := *x*] = *xs* @ *ys*[*i* − *length xs* := *x*]›
⟨*proof*⟩

lemma *lbd-ref-write-lbd-write*:
‹(*uncurry* (*lbd-ref-write*), *uncurry* (*RETURN oo lbd-write*)) ∈
[λ(*lbd*, *i*). *i* ≤ *Suc* (*uint-max div 2*)]$_f$
lbd-ref $\times_f$ *nat-rel* → ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

Cleaning the marked levels **definition** *lbd-emtpy-inv* :: ‹*nat* ⇒ *bool list* × *nat* ⇒ *bool*› **where**
‹*lbd-emtpy-inv m* = (λ(*xs*, *i*). *i* ≤ *Suc m* ∧ (∀*j* < *i*. *xs* ! *j* = *False*) ∧
(∀*j* > *m*. *j* < *length xs* ⟶ *xs* ! *j* = *False*))›

definition *lbd-empty-ref* **where**
‹*lbd-empty-ref* = (λ(*xs*, *m*, -). *do* {
(*xs*, *i*) ←
$WHILE_T$$^{lbd\text{-}emtpy\text{-}inv\ m}$
(λ(*xs*, *i*). *i* ≤ *m*)
(λ(*xs*, *i*). *do* {
ASSERT(*i* < *length xs*);
ASSERT(*i* + *one-uint32-nat* < *uint-max*);
RETURN (*xs*[*i* := *False*], *i* + *one-uint32-nat*)})
(*xs*, *zero-uint32-nat*);
RETURN (*xs*, *zero-uint32-nat*, *zero-uint32-nat*)
})›

definition *lbd-empty* **where**
‹*lbd-empty xs* = *RETURN* (*replicate* (*length xs*) *False*)›

lemma *lbd-empty-ref*:
assumes ‹((*xs*, *m*, *n*), *xs*) ∈ *lbd-ref*›
shows

‹*lbd-empty-ref* (*xs*, *m*, *n*) ≤ ⇓ *lbd-ref* (*RETURN* (*replicate* (*length xs*) *False*))›
⟨*proof*⟩

lemma *lbd-empty-ref-lbd-empty*:
‹(*lbd-empty-ref*, *lbd-empty*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −)*empty-lbd* :: ‹*lbd*› **where**
‹*empty-lbd* = (*replicate 32 False*)›

definition *empty-lbd-ref* :: ‹*lbd-ref*› **where**
‹*empty-lbd-ref* = (*replicate 32 False*, *zero-uint32-nat*, *zero-uint32-nat*)›

lemma *empty-lbd-ref-empty-lbd*:
‹(λ-. (*RETURN empty-lbd-ref*), λ-. (*RETURN empty-lbd*)) ∈ *unit-rel* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

Extracting the LBD We do not prove correctness of our algorithm, as we don't care about the actual returned value (for correctness).

definition *get-LBD* :: ‹*lbd* ⇒ *nat nres*› **where**
‹*get-LBD lbd* = *SPEC*(λ-. *True*)›

definition *get-LBD-ref* :: ‹*lbd-ref* ⇒ *nat nres*› **where**
‹*get-LBD-ref* = (λ(*xs*, *m*, *n*). *RETURN n*)›

lemma *get-LBD-ref*:
‹((*lbd*, *m*), *lbd'*) ∈ *lbd-ref* ⟹ *get-LBD-ref* (*lbd*, *m*) ≤ ⇓ *nat-rel* (*get-LBD lbd'*)›
⟨*proof*⟩

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

end
theory *LBD-SML*
imports *LBD Watched-Literals.WB-Word-Assn IsaSAT-Literals-SML*
begin

abbreviation *lbd-int-assn* :: ‹*lbd-ref* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-int-assn* ≡ *array-assn bool-assn* ∗*a uint32-nat-assn* ∗*a uint32-nat-assn*›

definition *lbd-assn* :: ‹*lbd* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-assn* ≡ *hr-comp lbd-int-assn lbd-ref*›

Testing if a level is marked **sepref-definition** *level-in-lbd-code*
is ‹*uncurry* (*RETURN oo level-in-lbd-ref*)›
:: ‹$[\lambda(n, (lbd, m)).\ length\ lbd \le uint\text{-}max]_a$
$uint32\text{-}nat\text{-}assn^k *_a lbd\text{-}int\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

lemma *level-in-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry level-in-lbd-code*, *uncurry* (*RETURN* ∘∘ *level-in-lbd*)) ∈ $uint32\text{-}nat\text{-}assn^k *_a$
$lbd\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

Marking more levels **lemma** *list-grow-array-hnr*[*sepref-fr-rules*]:
assumes ‹*CONSTRAINT is-pure R*›
shows
‹(*uncurry2* (λ*xs u. array-grow xs* (*nat-of-uint32 u*)),
uncurry2 (*RETURN ooo list-grow*)) $\in$
$[\lambda((xs, n), x).\ n \geq length\ xs]_a\ (array\text{-}assn\ R)^d *_a uint32\text{-}nat\text{-}assn^d *_a R^k \rightarrow$
array-assn R›
⟨*proof*⟩

sepref-definition *lbd-write-code*
is ‹*uncurry lbd-ref-write*›
:: ‹$lbd\text{-}int\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a lbd\text{-}int\text{-}assn$›
⟨*proof*⟩

lemma *lbd-write-hnr-*[*sepref-fr-rules*]:
‹(*uncurry lbd-write-code*, *uncurry* (*RETURN* $\circ\circ$ *lbd-write*))
$\in [\lambda(lbd, i).\ i \leq Suc\ (uint\text{-}max\ div\ 2)]_a$
$lbd\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow lbd\text{-}assn$›
⟨*proof*⟩

sepref-definition *lbd-empty-code*
is ‹*lbd-empty-ref*›
:: ‹$lbd\text{-}int\text{-}assn^d \rightarrow_a lbd\text{-}int\text{-}assn$›
⟨*proof*⟩

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
‹(*lbd-empty-code*, *lbd-empty*) $\in lbd\text{-}assn^d \rightarrow_a lbd\text{-}assn$›
⟨*proof*⟩

sepref-definition *empty-lbd-code*
is ‹*uncurry0* (*RETURN empty-lbd-ref*)›
:: ‹$unit\text{-}assn^k \rightarrow_a lbd\text{-}int\text{-}assn$›
⟨*proof*⟩

lemma *empty-lbd-hnr*[*sepref-fr-rules*]:
‹(*Sepref-Misc.uncurry0 empty-lbd-code*, *Sepref-Misc.uncurry0* (*RETURN empty-lbd*)) $\in unit\text{-}assn^k \rightarrow_a$ *lbd-assn*›
⟨*proof*⟩

sepref-definition *get-LBD-code*
is ‹*get-LBD-ref*›
:: ‹$lbd\text{-}int\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) $\in lbd\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

end
theory *Version*
imports *Main*
begin

This code was taken from IsaFoR and adapted to git.

local-setup ‹
let

val version =
  trim-line (*#1* (*Isabelle-System.bash-output* (*cd* \$*ISAFOL*/ && *git rev−parse −−short HEAD* || *echo unknown*)))
in
  Local-Theory.define
    ((***binding*** ‹*version*›, *NoSyn*),
      ((***binding*** ‹*version-def*›, []), *HOLogic.mk-literal version*)) *#>* *#2*
end
›

declare *version-def* [*code*]

end
theory *IsaSAT-Watch-List*
imports *IsaSAT-Literals*
  Watched-Literals.WB-Word
begin

There is not much to say about watch lists since they are arrays of resizeable arrays, which are defined in a separate theory.

However, when replacing the elements in our watch lists from *nat* × *uint32* to *nat* × *uint32* × *bool*, we got a huge and unexpected slowdown, due to a much higher number of cache misses (roughly 3.5 times as many on a eq.atree.braun.8.unsat.cnf which also took 66s instead of 50s). While toying with the generated ML code, we found out that our version of the tuples with booleans were using 40 bytes instead of 24 previously. Just merging the *uint32* and the *bool* to a single *uint64* was sufficient to get the performance back.

Remark that however, the evaluation of terms like 2^{32} was not done automatically and even worse, was redone each time, leading to a complete performance blow-up (75s on my macbook for eq.atree.braun.7.unsat.cnf instead of 7s).

definition *watcher-enc* **where**
‹*watcher-enc* = {(*n*, (*L*, *b*)). ∃ *L'*. (*L'*, *L*) ∈ *unat-lit-rel* ∧
  n = *uint64-of-uint32 L'* + (*if b then 1* << *32 else 0*)}›

definition *take-only-lower32* :: ‹*uint64* ⇒ *uint64*› **where**
[*code del*]: ‹*take-only-lower32 n* = *n AND* ((*1* << *32*) − *1*)›

lemma *nat-less-numeral-unfold*: **fixes** *n* :: *nat* **shows**
n < *numeral w* ⟷ *n* = *pred-numeral w* ∨ *n* < *pred-numeral w*
⟨*proof*⟩

lemma *bin-nth2-32-iff*: ‹*bin-nth 4294967295 na* ⟷ *na* < *32*›
⟨*proof*⟩

lemma *take-only-lower32-le-uint32-max*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *take-only-lower32 n* = *n*›
⟨*proof*⟩

lemma *uint32-of-uint64-uint64-of-uint32*[*simp*]: ‹*uint32-of-uint64* (*uint64-of-uint32 n*) = *n*›
⟨*proof*⟩

lemma *take-only-lower32-le-uint32-max-ge-uint32-max*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *nat-of-uint64 m* ≥ *uint32-max* ⟹ *take-only-lower32 m* = *0* ⟹

take-only-lower32 (*n* + *m*) = *n*›
⟨*proof*⟩

lemma *take-only-lower32-1-32*: ‹*take-only-lower32* (*1* << *32*) = *0*›
⟨*proof*⟩

lemma *nat-of-uint64-1-32*: ‹*nat-of-uint64* (*1* << *32*) = *uint32-max* + *1*›
⟨*proof*⟩

lemma *watcher-enc-extract-blit*:
assumes ‹(*n*, (*L*, *b*)) ∈ *watcher-enc*›
shows ‹(*uint32-of-uint64* (*take-only-lower32 n*), *L*) ∈ *unat-lit-rel*›
⟨*proof*⟩

fun *blit-of* **where**
‹*blit-of* (-, (*L*, -)) = *L*›

fun *blit-of-code* **where**
‹*blit-of-code* (*n*, *bL*) = *uint32-of-uint64* (*take-only-lower32 bL*)›

fun *is-marked-binary* **where**
‹*is-marked-binary* (-, (-, *b*)) = *b*›

fun *is-marked-binary-code* :: ‹- × *uint64* ⇒ *bool*› **where**
[*code del*]: ‹*is-marked-binary-code* (-, *bL*) = (*bL AND* ((*2* :: *uint64*)^*32*) ≠ *0*)›

lemma [*code*]:
‹*is-marked-binary-code* (*n*, *bL*) = (*bL AND 4294967296* ≠ *0*)›
⟨*proof*⟩

lemma *AND-2-32-bool*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n* + (*1* << *32*) *AND 4294967296* = *4294967296*›
⟨*proof*⟩

lemma *watcher-enc-extract-bool-True*:
assumes ‹(*n*, (*L*, *True*)) ∈ *watcher-enc*›
shows ‹*n AND 4294967296* = *4294967296*›
⟨*proof*⟩

lemma *le-uint32-max-AND2-32-eq0*: ‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n AND 4294967296* = *0*›
⟨*proof*⟩

lemma *watcher-enc-extract-bool-False*:
assumes ‹(*n*, (*L*, *False*)) ∈ *watcher-enc*›
shows ‹(*n AND 4294967296* = *0*)›
⟨*proof*⟩

lemma *watcher-enc-extract-bool*:
assumes ‹(*n*, (*L*, *b*)) ∈ *watcher-enc* ›
shows ‹*b* ⟷ (*n AND 4294967296* ≠ *0*)›
⟨*proof*⟩

definition *watcher-of* :: ‹*nat* × (*nat literal* × *bool*) ⇒ -› **where**
[*simp*]: ‹*watcher-of* = *id*›

definition *watcher-of-code* :: ‹*nat* × *uint64* ⇒ *nat* × (*uint32* × *bool*)› **where**
‹*watcher-of-code* = (λ(*a*, *b*). (*a*, (*blit-of-code* (*a*, *b*), *is-marked-binary-code* (*a*, *b*))))›

definition *watcher-of-fast-code* :: ‹*uint64* × *uint64* ⇒ *uint64* × (*uint32* × *bool*)› **where**
‹*watcher-of-fast-code* = (λ(*a*, *b*). (*a*, (*blit-of-code* (*a*, *b*), *is-marked-binary-code* (*a*, *b*))))›

definition *to-watcher* :: ‹*nat* ⇒ *nat literal* ⇒ *bool* ⇒ -› **where**
[*simp*]: ‹*to-watcher n L b* = (*n*, (*L*, *b*))›

definition *to-watcher-code* :: ‹*nat* ⇒ *uint32* ⇒ *bool* ⇒ *nat* × *uint64*› **where**
[*code del*]:
‹*to-watcher-code* = (λ*a L b*. (*a*, *uint64-of-uint32 L OR* (*if b then 1* << *32 else* (*0* :: *uint64*))))›

lemma *to-watcher-code*[*code*]:
‹*to-watcher-code a L b* = (*a*, *uint64-of-uint32 L OR* (*if b then 4294967296 else* (*0* :: *uint64*)))›
⟨*proof*⟩

lemma *OR-int64-0*[*simp*]: ‹*A OR* (*0* :: *uint64*) = *A*›
⟨*proof*⟩

lemma *OR-132-is-sum*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n OR* (*1* << *32*) = *n* + (*1* << *32*)›
⟨*proof*⟩

definition *to-watcher-fast* **where**
[*simp*]: ‹*to-watcher-fast* = *to-watcher*›

definition *to-watcher-fast-code* :: ‹*uint64* ⇒ *uint32* ⇒ *bool* ⇒ *uint64* × *uint64*› **where**
‹*to-watcher-fast-code* = (λ*a L b*. (*a*, *uint64-of-uint32 L OR* (*if b then 1* << *32 else* (*0* :: *uint64*))))›

lemma *take-only-lower-code*[*code*]:
‹*take-only-lower32 n* = *n AND 4294967295*›
⟨*proof*⟩

end
theory *IsaSAT-Watch-List-SML*
imports *Watched-Literals.Array-Array-List64 IsaSAT-Watch-List IsaSAT-Literals-SML*
begin

type-synonym *watched-wl* = ‹((*nat* × *uint64*) *array-list*) *array*›
type-synonym *watched-wl-uint32* = ‹((*uint64* × *uint64*) *array-list64*) *array*›

abbreviation *watcher-enc-assn* **where**
‹*watcher-enc-assn* ≡ *pure watcher-enc*›

abbreviation *watcher-assn* **where**
‹*watcher-assn* ≡ *nat-assn* ∗*a watcher-enc-assn*›

abbreviation *watcher-fast-assn* **where**
‹*watcher-fast-assn* ≡ *uint64-nat-assn* ∗*a watcher-enc-assn*›

lemma *is-marked-binary-code-hnr*:

‹(*return o is-marked-binary-code*, *RETURN o is-marked-binary*) ∈ *watcher-assn*^{k} →$_a$ *bool-assn*›
⟨*proof*⟩

lemma
pair-nat-ann-lit-assn-Decided-Some:
‹*pair-nat-ann-lit-assn* (*Decided x1*) (*aba*, *Some x2*) = *false*› **and**
pair-nat-ann-lit-assn-Propagated-None:
‹*pair-nat-ann-lit-assn* (*Propagated x21 x22*) (*aba*, *None*) = *false*›
⟨*proof*⟩

lemma *blit-of-code-hnr*:
‹(*return o blit-of-code*, *RETURN o blit-of*) ∈ *watcher-assn*^{k} →$_a$ *unat-lit-assn*›
⟨*proof*⟩

lemma *watcher-of-code-hnr*[*sepref-fr-rules*]:
‹(*return o watcher-of-code*, *RETURN o watcher-of*) ∈
watcher-assn^k →$_a$ (*nat-assn* ∗*a unat-lit-assn* ∗*a bool-assn*)›
⟨*proof*⟩

lemma *watcher-of-fast-code-hnr*[*sepref-fr-rules*]:
‹(*return o watcher-of-fast-code*, *RETURN o watcher-of*) ∈
watcher-fast-assn^k →$_a$ (*uint64-nat-assn* ∗*a unat-lit-assn* ∗*a bool-assn*)›
⟨*proof*⟩

lemma *to-watcher-code-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-watcher-code*), *uncurry2* (*RETURN ooo to-watcher*)) ∈
nat-assn^k ∗$_a$ *unat-lit-assn*^{k} ∗$_a$ *bool-assn*^{k} →$_a$ *watcher-assn*›
⟨*proof*⟩

lemma *to-watcher-fast-code-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-watcher-fast-code*), *uncurry2* (*RETURN ooo to-watcher-fast*)) ∈
uint64-nat-assn^k ∗$_a$ *unat-lit-assn*^{k} ∗$_a$ *bool-assn*^{k} →$_a$ *watcher-fast-assn*›
⟨*proof*⟩

end
theory *IsaSAT-Lookup-Conflict*
imports
IsaSAT-Literals
Watched-Literals.CDCL-Conflict-Minimisation
LBD
IsaSAT-Clauses
IsaSAT-Watch-List
IsaSAT-Trail
begin

hide-const *Autoref-Fix-Rel.CONSTRAINT*
no-notation *Ref.update* (- := - *62*)

Clauses Encoded as Positions

We use represent the conflict in two data structures close to the one used by the most SAT solvers: We keep an array that represent the clause (for efficient iteration on the clause) and a "hash-table" to efficiently test if a literal belongs to the clause.

The first data structure is simply an array to represent the clause. This theory is only about the second data structure. We refine it from the clause (seen as a multiset) in two steps:

1. First, we represent the clause as a “hash-table”, where the i-th position indicates *Some True* (respectively *Some False*, *None*) if *Pos i* is present in the clause (respectively *Neg i*, not at all). This allows to represent every not-tautological clause whose literals fits in the underlying array.

2. Then we refine it to an array of booleans indicating if the atom is present or not. This information is redundant because we already know that a literal can only appear negated compared to the trail.

The first step makes it easier to reason about the clause (since we have the full clause), while the second step should generate (slightly) more efficient code.

Most solvers also merge the underlying array with the array used to cache information for the conflict minimisation (see theory *Watched-Literals.CDCL-Conflict-Minimisation*, where we only test if atoms appear in the clause, not literals).

As far as we know, versat stops at the first refinement (stating that there is no significant overhead, which is probably true, but the second refinement is not much additional work anyhow and we don’t have to rely on the ability of the compiler to not represent the option type on booleans as a pointer, which it might be able to or not).

This is the first level of the refinement. We tried a few different definitions (including a direct one, i.e., mapping a position to the inclusion in the set) but the inductive version turned out to the easiest one to use.

inductive *mset-as-position* :: ‹*bool option list* ⇒ *nat literal multiset* ⇒ *bool*› **where**
empty:
  ‹*mset-as-position* (*replicate n None*) {#}› |
add:
  ‹*mset-as-position xs'* (*add-mset L P*)›
  if ‹*mset-as-position xs P*› **and** ‹*atm-of L* < *length xs*› **and** ‹$L \notin\# P$› **and** ‹$-L \notin\# P$› **and**
    ‹*xs'* = *xs*[*atm-of L* := *Some* (*is-pos L*)]›

lemma *mset-as-position-distinct-mset*:
  ‹*mset-as-position xs P* ⟹ *distinct-mset P*›
  ⟨*proof*⟩

lemma *mset-as-position-atm-le-length*:
  ‹*mset-as-position xs P* ⟹ $L \in\# P$ ⟹ *atm-of L* < *length xs*›
  ⟨*proof*⟩

lemma *mset-as-position-nth*:
  ‹*mset-as-position xs P* ⟹ $L \in\# P$ ⟹ *xs* ! (*atm-of L*) = *Some* (*is-pos L*)›
  ⟨*proof*⟩

lemma *mset-as-position-in-iff-nth*:
  ‹*mset-as-position xs P* ⟹ *atm-of L* < *length xs* ⟹ $L \in\# P$ ⟷ *xs* ! (*atm-of L*) = *Some* (*is-pos L*)›
  ⟨*proof*⟩

lemma *mset-as-position-tautology*: ‹*mset-as-position xs C* ⟹ ¬*tautology C*›
  ⟨*proof*⟩

lemma *mset-as-position-right-unique*:
  assumes
    map: ‹*mset-as-position xs D*› **and**
    map': ‹*mset-as-position xs D'*›

shows ‹$D = D'$›
⟨*proof*⟩

lemma *mset-as-position-mset-union*:
fixes P *xs*
defines ‹$xs' \equiv fold\ (\lambda L\ xs.\ xs[atm\text{-}of\ L := Some\ (is\text{-}pos\ L)])\ P\ xs$›
assumes
mset: ‹$mset\text{-}as\text{-}position\ xs\ P'$› **and**
atm-P-xs: ‹$\forall L \in set\ P.\ atm\text{-}of\ L < length\ xs$› **and**
uL-P: ‹$\forall L \in set\ P.\ -L \notin\#\ P'$› **and**
dist: ‹*distinct* P› **and**
tauto: ‹$\neg tautology\ (mset\ P)$›
shows ‹$mset\text{-}as\text{-}position\ xs'\ (mset\ P \cup\#\ P')$›
⟨*proof*⟩

lemma *mset-as-position-empty-iff*: ‹$mset\text{-}as\text{-}position\ xs\ \{\#\} \longleftrightarrow (\exists n.\ xs = replicate\ n\ None)$›
⟨*proof*⟩

type-synonym **(in** $-$**)** *lookup-clause-rel* = ‹$nat \times bool\ option\ list$›

definition *lookup-clause-rel* :: ‹$nat\ multiset \Rightarrow (lookup\text{-}clause\text{-}rel \times nat\ literal\ multiset)\ set$› **where**
‹$lookup\text{-}clause\text{-}rel\ \mathcal{A} = \{((n, xs), C).\ n = size\ C \wedge mset\text{-}as\text{-}position\ xs\ C \wedge$
$(\forall L{\in}atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}).\ L < length\ xs)\}$›

lemma *lookup-clause-rel-empty-iff*: ‹$((n, xs), C) \in lookup\text{-}clause\text{-}rel\ \mathcal{A} \Longrightarrow n = 0 \longleftrightarrow C = \{\#\}$›
⟨*proof*⟩

lemma *conflict-atm-le-length*: ‹$((n, xs), C) \in lookup\text{-}clause\text{-}rel\ \mathcal{A} \Longrightarrow L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}) \Longrightarrow$
$L < length\ xs$›
⟨*proof*⟩

lemma *conflict-le-length*:
assumes
c-rel: ‹$((n, xs), C) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}$› **and**
$L\text{-}\mathcal{L}_{all}$: ‹$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›
shows ‹$atm\text{-}of\ L < length\ xs$›
⟨*proof*⟩

lemma *lookup-clause-rel-atm-in-iff*:
‹$((n, xs), C) \in lookup\text{-}clause\text{-}rel\ \mathcal{A} \Longrightarrow L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow L \in\#\ C \longleftrightarrow xs!(atm\text{-}of\ L) = Some\ (is\text{-}pos\ L)$›
⟨*proof*⟩

lemma
assumes
c: ‹$((n,xs), C) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}$› **and**
bounded: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}$›
shows
lookup-clause-rel-not-tautolgy: ‹$\neg tautology\ C$› **and**
lookup-clause-rel-distinct-mset: ‹$distinct\text{-}mset\ C$› **and**
lookup-clause-rel-size: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ C \Longrightarrow size\ C \le 1 + uint\text{-}max\ div\ 2$›
⟨*proof*⟩

type-synonym *lookup-clause-assn* = ‹$uint32 \times bool\ array$›

definition *option-bool-rel* :: ‹(*bool* × *'a option*) *set*› **where**
‹*option-bool-rel* = {(*b*, *x*). *b* ⟷ ¬(*is-None x*)}›

definition *NOTIN* :: ‹*bool option*› **where**
[*simp*]: ‹*NOTIN* = *None*›

definition *ISIN* :: ‹*bool* ⇒ *bool option*› **where**
[*simp*]: ‹*ISIN b* = *Some b*›

definition *is-NOTIN* :: ‹*bool option* ⇒ *bool*› **where**
[*simp*]: ‹*is-NOTIN x* ⟷ *x* = *NOTIN*›

definition *option-lookup-clause-rel* **where**
‹*option-lookup-clause-rel* $\mathcal{A}$ = {((*b*,(*n*,*xs*)), *C*). *b* = (*C* = *None*) ∧
(*C* = *None* ⟶ ((*n*,*xs*), {#}) ∈ *lookup-clause-rel* $\mathcal{A}$) ∧
(*C* ≠ *None* ⟶ ((*n*,*xs*), *the C*) ∈ *lookup-clause-rel* $\mathcal{A}$)}
›

lemma *option-lookup-clause-rel-lookup-clause-rel-iff*:
‹((*False*, (*n*, *xs*)), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟷
((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

type-synonym (**in** −) *option-lookup-clause-assn* = ‹*bool* × *uint32* × *bool array*›

type-synonym (**in** −) *conflict-option-rel* = ‹*bool* × *nat* × *bool option list*›

definition (**in** −) *lookup-clause-assn-is-None* :: ‹*-* ⇒ *bool*› **where**
‹*lookup-clause-assn-is-None* = (λ(*b*, *-*, *-*). *b*)›

lemma *lookup-clause-assn-is-None-is-None*:
‹(*RETURN o lookup-clause-assn-is-None*, *RETURN o is-None*) ∈
option-lookup-clause-rel $\mathcal{A}$ $\to_f$ ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *lookup-clause-assn-is-empty* :: ‹*-* ⇒ *bool*› **where**
‹*lookup-clause-assn-is-empty* = (λ(*-*, *n*, *-*). *n* = *0*)›

lemma *lookup-clause-assn-is-empty-is-empty*:
‹(*RETURN o lookup-clause-assn-is-empty*, *RETURN o* (λ*D*. *Multiset.is-empty*(*the D*))) ∈
[λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *size-lookup-conflict* :: ‹*-* ⇒ *nat*› **where**
‹*size-lookup-conflict* = (λ(*-*, *n*, *-*). *n*)›

definition *size-conflict-wl-heur* :: ‹*-* ⇒ *nat*› **where**
‹*size-conflict-wl-heur* = (λ(*M*, *N*, *U*, *D*, *-*, *-*, *-*, *-*). *size-lookup-conflict D*)›

lemma (**in** −) *mset-as-position-length-not-None*:
‹*mset-as-position x2 C* ⟹ *size C* = *length* (*filter* ((≠) *None*) *x2*)›
⟨*proof*⟩

definition (**in** $-$) *is-in-lookup-conflict* **where**
‹*is-in-lookup-conflict* = (λ(*n*, *xs*) *L*. $\neg$*is-None* (*xs* ! *atm-of L*))›

lemma *mset-as-position-remove*:
‹*mset-as-position xs D* $\Longrightarrow$ *L* < *length xs* $\Longrightarrow$
mset-as-position (*xs*[*L* := *None*]) (*remove1-mset* (*Pos L*) (*remove1-mset* (*Neg L*) *D*))›
⟨*proof*⟩

definition (**in** $-$) *delete-from-lookup-conflict*
:: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel nres*› **where**
‹*delete-from-lookup-conflict* = (λ*L* (*n*, *xs*). *do* {
ASSERT(*n*≥*1*);
ASSERT(*atm-of L* < *length xs*);
RETURN (*fast-minus n one-uint32-nat*, *xs*[*atm-of L* := *None*])
})›

lemma *delete-from-lookup-conflict-op-mset-delete*:
‹(*uncurry delete-from-lookup-conflict*, *uncurry* (*RETURN oo remove1-mset*)) ∈
[λ(*L*, *D*). $-L \notin\#$ *D* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *L* ∈# *D*]$_f$ *Id* $\times_f$ *lookup-clause-rel* $\mathcal{A}$ →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *delete-from-lookup-conflict-pre* **where**
‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ = (λ(*a*, *b*). $-$ *a* ∉# *b* ∧ *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *a* ∈# *b*)›

definition *set-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*set-conflict-m M N i* - - - - - =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*N*∝*i*)) ∧ *n* = *card-max-lvl M* (*mset* (*N*∝*i*)) ∧
out-learned M C outl)›

definition *merge-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m M N i D* - - - =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *merge-conflict-m-g*
:: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m-g init M Ni D* =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *add-to-lookup-conflict* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*add-to-lookup-conflict* = (λ*L* (*n*, *xs*). (*if xs* ! *atm-of L* = *NOTIN then n* + *1 else n*,
xs[*atm-of L* := *ISIN* (*is-pos L*)]))›

definition *lookup-conflict-merge′-step*
:: ‹*nat multiset* ⇒ *nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *nat* ⇒ *lookup-clause-rel* ⇒ *nat clause-l* ⇒ *nat clause* ⇒ *out-learned* ⇒ *bool*›
where
‹*lookup-conflict-merge′-step* $\mathcal{A}$ *init M i clvls zs D C outl* = (
let D' = *mset* (*take* (*i* − *init*) (*drop init D*));
E = *remdups-mset* (D' + *C*) *in*
((*False*, *zs*), *Some E*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
out-learned M (*Some E*) *outl* ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ *E* ∧ *clvls* = *card-max-lvl M E*)›

lemma *option-lookup-clause-rel-update-None*:
assumes ‹((*False*, (*n*, *xs*)), *Some D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and** *L-xs* : ‹*L* < *length xs*›
shows ‹((*False*, (*if xs*!*L* = *None then n else n* − *1*, *xs*[*L* := *None*])),
Some (*D* − {# *Pos L*, *Neg L* #})) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

lemma *add-to-lookup-conflict-lookup-clause-rel*:
assumes
confl: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
uL-C: ‹−*L* ∉# *C*› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹(*add-to-lookup-conflict L* (*n*, *xs*), {#*L*#} ∪# *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

definition *outlearned-add*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
‹*outlearned-add* = (λ*M L zs outl*.
(*if get-level M L* < *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
else outl))›

definition *clvls-add*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*clvls-add* = (λ*M L zs clvls*.
(*if get-level M L* = *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
else clvls))›

definition *lookup-conflict-merge*
:: ‹*nat* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*lookup-conflict-merge init M D* = (λ(*b*, *xs*) *clvls lbd outl*. *do* {
(-, *clvls*, *zs*, *lbd*, *outl*) ← $WHILE_T$λ(*i*::*nat*, *clvls* :: *nat*, *zs*, *lbd*, *outl*). *length* (*snd zs*) = *length* (*snd xs*) ∧ *Su*
(λ(*i* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *i* < *length-uint32-nat D*)
(λ(*i* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *do* {
ASSERT(*i* < *length-uint32-nat D*);
ASSERT(*Suc i* ≤ *uint-max*);
let lbd = *lbd-write lbd* (*get-level M* (*D*!*i*));
ASSERT(¬*is-in-lookup-conflict zs* (*D*!*i*) ⟶ *length outl* < *uint32-max*);
let outl = *outlearned-add M* (*D*!*i*) *zs outl*;
let clvls = *clvls-add M* (*D*!*i*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*D*!*i*) *zs*;
RETURN(*Suc i*, *clvls*, *zs*, *lbd*, *outl*)

})
(*init*, *clvls*, *xs*, *lbd*, *outl*);
RETURN ((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

definition *resolve-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*resolve-lookup-conflict-aa M N i xs clvls lbd outl* =
lookup-conflict-merge 1 M (*N* ∝ *i*) *xs clvls lbd outl*›

definition *set-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒(*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*set-lookup-conflict-aa M C i xs clvls lbd outl* =
lookup-conflict-merge zero-uint32-nat M (*C*∝*i*) *xs clvls lbd outl*›

definition *isa-outlearned-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
‹*isa-outlearned-add* = (λ*M L zs outl*.
(*if get-level-pol M L* < *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
else outl))›

lemma *isa-outlearned-add-outlearned-add*:
‹(M', *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-outlearned-add M' *L zs outl* = *outlearned-add M L zs outl*›
⟨*proof*⟩

definition *isa-clvls-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*isa-clvls-add* = (λ*M L zs clvls*.
(*if get-level-pol M L* = *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
else clvls))›

lemma *isa-clvls-add-clvls-add*:
‹(M', *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-clvls-add M' *L zs outl* = *clvls-add M L zs outl*›
⟨*proof*⟩

definition *isa-lookup-conflict-merge*
:: ‹*nat* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*isa-lookup-conflict-merge init M N i* = (λ(*b*, *xs*) *clvls lbd outl*. *do* {
ASSERT(*arena-is-valid-clause-idx N i*);
(-, *clvls*, *zs*, *lbd*, *outl*) ← $WHILE_T$λ(*i*::*nat*, *clvls* :: *nat*, *zs*, *lbd*, *outl*). *length* (*snd zs*) = *length* (*snd xs*) ∧ *Su*
(λ(*j* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *j* < *i* + *arena-length N i*)
(λ(*j* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *do* {
ASSERT(*j* < *length N*);
ASSERT(*arena-lit-pre N j*);
ASSERT(*get-level-pol-pre* (*M*, *arena-lit N j*));
ASSERT(*get-level-pol M* (*arena-lit N j*) ≤ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level-pol M* (*arena-lit N j*));

ASSERT(*atm-of* (*arena-lit N j*) < *length* (*snd zs*));
ASSERT(¬*is-in-lookup-conflict zs* (*arena-lit N j*) ⟶ *length outl* < *uint32-max*);
let outl = *isa-outlearned-add M* (*arena-lit N j*) *zs outl*;
let clvls = *isa-clvls-add M* (*arena-lit N j*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*arena-lit N j*) *zs*;
RETURN(*Suc j*, *clvls*, *zs*, *lbd*, *outl*)
})
(*i*+*init*, *clvls*, *xs*, *lbd*, *outl*);
RETURN ((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

definition *isa-set-lookup-conflict* **where**
‹*isa-set-lookup-conflict* = *isa-lookup-conflict-merge 0*›

lemma *isa-lookup-conflict-merge-lookup-conflict-merge-ext*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
bxs: ‹((*b*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
M′M: ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isa-lookup-conflict-merge init M′ arena i* (*b*, *xs*) *clvls lbd outl* ≤ ⇓ *Id*
(*lookup-conflict-merge init M* (*N* ∝ *i*) (*b*, *xs*) *clvls lbd outl*)›
⟨*proof*⟩

abbreviation (**in** −) *minimize-status-rel* **where**
‹*minimize-status-rel* ≡ *Id* :: (*minimize-status* × *minimize-status*) *set*›

lemma (**in** −) *arena-is-valid-clause-idx-le-uint64-max*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *uint64-max* ⟹
bd + *arena-length be bd* ≤ *uint64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *uint64-max* ⟹
bd ≤ *uint64-max*›
⟨*proof*⟩

definition *isa-set-lookup-conflict-aa* **where**
‹*isa-set-lookup-conflict-aa* = *isa-lookup-conflict-merge 0*›

definition *isa-set-lookup-conflict-aa-pre* **where**
‹*isa-set-lookup-conflict-aa-pre* =
(λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), *out*). *i* < *length N*)›

lemma *lookup-conflict-merge′-spec*:
assumes
o: ‹((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C*› **and**
‹*clvls* = *card-max-lvl M C*› **and**
CD: ‹⋀*L*. *L* ∈ *set* (*drop init D*) ⟹ −*L* ∉# *C*› **and**
‹*Suc init* ≤ *uint-max*› **and**
‹*out-learned M* (*Some C*) *outl*› **and**

bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*lookup-conflict-merge init M D* (*b*, *n*, *xs*) *clvls lbd outl* $\leq$
$\Downarrow$(*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*merge-conflict-m-g init M D* (*Some C*))›
(**is** ‹- $\leq$ $\Downarrow$ *?Ref ?Spec*›)
⟨*proof*⟩

lemma *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$:
assumes *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
i: ‹*i* $\in\#$ *dom-m N*›
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* $\propto$ *i*))›
⟨*proof*⟩

lemma *isa-set-lookup-conflict*:
‹(*uncurry6 isa-set-lookup-conflict-aa*, *uncurry6 set-conflict-m*) $\in$
[λ((((((*M*, *N*), *i*), *xs*), *clvls*), *lbd*), *outl*). *i* $\in\#$ *dom-m N* $\wedge$ *xs* = *None* $\wedge$ *distinct* (*N* $\propto$ *i*) $\wedge$
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) $\wedge$
$\neg$*tautology* (*mset* (*N* $\propto$ *i*)) $\wedge$ *clvls* = *0* $\wedge$
out-learned M None outl $\wedge$
isasat-input-bounded $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ $\times_f$
nat-rel $\times_f$ *Id*
$\times_f$ *Id* $\rightarrow$
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*›
⟨*proof*⟩

definition *merge-conflict-m-pre* **where**
‹*merge-conflict-m-pre* $\mathcal{A}$ =
(λ((((((*M*, *N*), *i*), *xs*), *clvls*), *lbd*), *out*). *i* $\in\#$ *dom-m N* $\wedge$ *xs* $\neq$ *None* $\wedge$ *distinct* (*N* $\propto$ *i*) $\wedge$
$\neg$*tautology* (*mset* (*N* $\propto$ *i*)) $\wedge$
($\forall$ *L* $\in$ *set* (*tl* (*N* $\propto$ *i*)). $-$ *L* $\notin\#$ *the xs*) $\wedge$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) $\wedge$ *clvls* = *card-max-lvl M* (*the xs*) $\wedge$
out-learned M xs out $\wedge$ *no-dup M* $\wedge$
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) $\wedge$
isasat-input-bounded $\mathcal{A}$)›

definition *isa-resolve-merge-conflict-gt2* **where**
‹*isa-resolve-merge-conflict-gt2* = *isa-lookup-conflict-merge 1*›

lemma *isa-resolve-merge-conflict-gt2*:
‹(*uncurry6 isa-resolve-merge-conflict-gt2*, *uncurry6 merge-conflict-m*) $\in$
[*merge-conflict-m-pre* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$
$\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$ *Id* $\rightarrow$
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*›
⟨*proof*⟩

definition (**in** $-$) *is-in-conflict* :: ‹*nat literal* $\Rightarrow$ *nat clause option* $\Rightarrow$ *bool*› **where**
[*simp*]: ‹*is-in-conflict L C* $\longleftrightarrow$ *L* $\in\#$ *the C*›

definition (**in** $-$) *is-in-lookup-option-conflict*
:: ‹*nat literal* $\Rightarrow$ (*bool* $\times$ *nat* $\times$ *bool option list*) $\Rightarrow$ *bool*›
where
‹*is-in-lookup-option-conflict* = (λ*L* (-, -, *xs*). *xs* ! *atm-of L* = *Some* (*is-pos L*))›

lemma *is-in-lookup-option-conflict-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-lookup-option-conflict*),
uncurry (*RETURN oo is-in-conflict*)) $\in$
$[\lambda(L, C).\ C \neq None \wedge L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f\ Id \times_r$ *option-lookup-clause-rel* $\mathcal{A}$ $\rightarrow$
$\langle Id\rangle$*nres-rel*›
$\langle proof\rangle$

definition *conflict-from-lookup* **where**
‹*conflict-from-lookup* = $(\lambda(n, xs).\ SPEC(\lambda D.\ mset\text{-}as\text{-}position\ xs\ D \wedge n = size\ D))$›

lemma *Ex-mset-as-position*:
‹*Ex* (*mset-as-position xs*)›
$\langle proof\rangle$

lemma *id-conflict-from-lookup*:
‹(*RETURN o id*, *conflict-from-lookup*) $\in [\lambda(n, xs).\ \exists D.\ ((n, xs), D) \in$ *lookup-clause-rel* $\mathcal{A}]_f\ Id \rightarrow$
$\langle$*lookup-clause-rel* $\mathcal{A}\rangle$*nres-rel*›
$\langle proof\rangle$

lemma *lookup-clause-rel-exists-le-uint-max*:
assumes *ocr*: ‹$((n, xs), D) \in$ *lookup-clause-rel* $\mathcal{A}$› **and** ‹$n > 0$› **and**
le-i: ‹$\forall k<i.\ xs\ !\ k = None$› **and** *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ D› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹$\exists j.\ j \geq i \wedge j <$ *length xs* $\wedge\ j <$ *uint-max* $\wedge\ xs\ !\ j \neq None$›
$\langle proof\rangle$

During the conflict analysis, the literal of highest level is at the beginning. During the rest of the time the conflict is *None*.

definition *highest-lit* **where**
‹*highest-lit M C L* $\longleftrightarrow$
$(L = None \longrightarrow C = \{\#\}) \wedge$
$(L \neq None \longrightarrow$ *get-level M* (*fst* (*the L*)) = *snd* (*the L*) $\wedge$
snd (*the L*) = *get-maximum-level M C* $\wedge$
fst (*the L*) $\in\#$ *C*
)›

Conflict Minimisation

definition *iterate-over-conflict-inv* **where**
‹*iterate-over-conflict-inv* $M\ D_0{}' = (\lambda(D, D').\ D \subseteq\# D_0{}' \wedge D' \subseteq\# D)$›

definition *is-literal-redundant-spec* **where**
‹*is-literal-redundant-spec K NU UNE D L* = $SPEC(\lambda b.\ b \longrightarrow$
$NU + UNE \models pm$ *remove1-mset L* (*add-mset K D*))›

definition *iterate-over-conflict*
:: ‹$'v$ *literal* $\Rightarrow$ ($'v$, $'mark$) *ann-lits* $\Rightarrow$ $'v$ *clauses* $\Rightarrow$ $'v$ *clauses* $\Rightarrow$ $'v$ *clause* $\Rightarrow$
$'v$ *clause nres*›
where
‹*iterate-over-conflict K M NU UNE* $D_0{}'$ = *do* {
(*D*, -) $\leftarrow$
$WHILE_T{}^{iterate\text{-}over\text{-}conflict\text{-}inv\ M\ D_0{}'}$
$(\lambda(D, D').\ D' \neq \{\#\})$
$(\lambda(D, D').\ do\{$
$x \leftarrow SPEC\ (\lambda x.\ x \in\# D')$;

$red \leftarrow is\text{-}literal\text{-}redundant\text{-}spec\ K\ NU\ UNE\ D\ x;$
$if\ \neg red$
$then\ RETURN\ (D,\ remove1\text{-}mset\ x\ D')$
$else\ RETURN\ (remove1\text{-}mset\ x\ D,\ remove1\text{-}mset\ x\ D')$
$\})$
$(D_0',\ D_0');$
$RETURN\ D$
$\}\rangle$

definition *minimize-and-extract-highest-lookup-conflict-inv* **where**
$\langle minimize\text{-}and\text{-}extract\text{-}highest\text{-}lookup\text{-}conflict\text{-}inv = (\lambda(D,\ i,\ s,\ outl).$
$length\ outl \leq uint\text{-}max \wedge mset\ (tl\ outl) = D \wedge outl \neq [] \wedge i \geq 1)\rangle$

type-synonym $'v\ conflict\text{-}highest\text{-}conflict = \langle('v\ literal \times nat)\ option\rangle$

definition (**in** $-$) *atm-in-conflict* **where**
$\langle atm\text{-}in\text{-}conflict\ L\ D \longleftrightarrow L \in atms\text{-}of\ D\rangle$

definition *atm-in-conflict-lookup* :: $\langle nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool\rangle$ **where**
$\langle atm\text{-}in\text{-}conflict\text{-}lookup = (\lambda L\ (\text{-},\ xs).\ xs\ !\ L \neq None)\rangle$

definition *atm-in-conflict-lookup-pre* :: $\langle nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool\rangle$ **where**
$\langle atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre\ L\ xs \longleftrightarrow L < length\ (snd\ xs)\rangle$

lemma *atm-in-conflict-lookup-atm-in-conflict*:
$\langle(uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict\text{-}lookup),\ uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict)) \in$
$[\lambda(L,\ xs).\ L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})]_f\ Id \times_f lookup\text{-}clause\text{-}rel\ \mathcal{A} \rightarrow \langle bool\text{-}rel\rangle nres\text{-}rel\rangle$
$\langle proof\rangle$

lemma *atm-in-conflict-lookup-pre*:
fixes $x1$ :: $\langle nat\rangle$ **and** $x2$ :: $\langle nat\rangle$
assumes
$\langle x1n \in\# \mathcal{L}_{all}\ \mathcal{A}\rangle$ **and**
$\langle(x2f,\ x2a) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}\rangle$
shows $\langle atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre\ (atm\text{-}of\ x1n)\ x2f\rangle$
$\langle proof\rangle$

definition *is-literal-redundant-lookup-spec* **where**
$\langle is\text{-}literal\text{-}redundant\text{-}lookup\text{-}spec\ \mathcal{A}\ M\ NU\ NUE\ D'\ L\ s =$
$SPEC(\lambda(s',\ b).\ b \longrightarrow (\forall D.\ (D',\ D) \in lookup\text{-}clause\text{-}rel\ \mathcal{A} \longrightarrow$
$(mset\ `\#\ mset\ (tl\ NU)) + NUE \models pm\ remove1\text{-}mset\ L\ D))\rangle$

type-synonym (**in** $-$) $conflict\text{-}min\text{-}cach\text{-}l = \langle minimize\text{-}status\ list \times nat\ list\rangle$

definition (**in** $-$) *conflict-min-cach-set-removable-l*
:: $\langle conflict\text{-}min\text{-}cach\text{-}l \Rightarrow nat \Rightarrow conflict\text{-}min\text{-}cach\text{-}l\ nres\rangle$
where
$\langle conflict\text{-}min\text{-}cach\text{-}set\text{-}removable\text{-}l = (\lambda(cach,\ sup)\ L.\ do\ \{$
$ASSERT(L < length\ cach);$
$ASSERT(length\ sup \leq 1 + uint32\text{-}max\ div\ 2);$
$RETURN\ (cach[L := SEEN\text{-}REMOVABLE],\ if\ cach\ !\ L = SEEN\text{-}UNKNOWN\ then\ sup\ @\ [L]\ else$
$sup)$
$\})\rangle$

definition (**in** $-$) *conflict-min-cach* :: $\langle nat\ conflict\text{-}min\text{-}cach \Rightarrow nat \Rightarrow minimize\text{-}status\rangle$ **where**

[*simp*]: ‹*conflict-min-cach cach L* = *cach L*›

definition *lit-redundant-reason-stack2*
:: ‹$'v$ *literal* ⇒ $'v$ *clauses-l* ⇒ *nat* ⇒ (*nat* × *nat* × *bool*)› **where**
‹*lit-redundant-reason-stack2 L NU C'* =
(*if length* (*NU* ∝ *C'*) > *2 then* (*C'*, *1*, *False*)
else if NU ∝ *C'* ! *0* = *L then* (*C'*, *1*, *False*)
else (*C'*, *0*, *True*))›

definition *ana-lookup-rel*
:: ‹*nat clauses-l* ⇒ ((*nat* × *nat* × *bool*) × (*nat* × *nat* × *nat* × *nat*)) *set*›
where
‹*ana-lookup-rel NU* = {((*C*, *i*, *b*), (*C'*, *k'*, *i'*, *len'*)).
C = *C'* ∧ *k'* = (*if b then 1 else 0*) ∧ *i* = *i'* ∧
len' = (*if b then 1 else length* (*NU* ∝ *C*))}›

lemma *ana-lookup-rel-alt-def*:
‹((*C*, *i*, *b*), (*C'*, *k'*, *i'*, *len'*)) ∈ *ana-lookup-rel NU* ⟷
C = *C'* ∧ *k'* = (*if b then 1 else 0*) ∧ *i* = *i'* ∧
len' = (*if b then 1 else length* (*NU* ∝ *C*))›
⟨*proof*⟩

abbreviation *ana-lookups-rel* **where**
‹*ana-lookups-rel NU* ≡ ⟨*ana-lookup-rel NU*⟩*list-rel*›

definition *ana-lookup-conv* :: ‹*nat clauses-l* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv NU* = (λ(*C*, *i*, *b*). (*C*, (*if b then 1 else 0*), *i*, (*if b then 1 else length* (*NU* ∝ *C*))))›

definition *get-literal-and-remove-of-analyse-wl2*
:: ‹$'v$ *clause-l* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ $'v$ *literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*get-literal-and-remove-of-analyse-wl2 C analyse* =
(*let* (*i*, *j*, *b*) = *last analyse in*
(*C* ! *j*, *analyse*[*length analyse* − *1* := (*i*, *j* + *1*, *b*)]))›

definition *lit-redundant-rec-wl-inv2* **where**
‹*lit-redundant-rec-wl-inv2 M NU D* =
(λ(*cach*, *analyse*, *b*). ∃ *analyse'*. (*analyse*, *analyse'*) ∈ *ana-lookups-rel NU* ∧
lit-redundant-rec-wl-inv M NU D (*cach*, *analyse'*, *b*))›

definition *mark-failed-lits-stack-inv2* **where**
‹*mark-failed-lits-stack-inv2 NU analyse* = (λ*cach*.
∃ *analyse'*. (*analyse*, *analyse'*) ∈ *ana-lookups-rel NU* ∧
mark-failed-lits-stack-inv NU analyse' cach)›

definition *lit-redundant-rec-wl-lookup*
:: ‹*nat multiset* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒
- ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›
where
‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* =
$WHILE_T{}^{lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D}$
(λ(*cach*, *analyse*, *b*). *analyse* ≠ [])
(λ(*cach*, *analyse*, *b*). *do* {
ASSERT(*analyse* ≠ []);
ASSERT(*length analyse* ≤ *length M*);
let (*C*,*k*, *i*, *len*) = *ana-lookup-conv NU* (*last analyse*);

$ASSERT(C \in\# \textit{dom-m } NU)$;
$ASSERT(\textit{length } (NU \propto C) > k)$; — >= 2 would work too
$ASSERT\ (NU \propto C\ !\ k \in \textit{lits-of-l } M)$;
$ASSERT(NU \propto C\ !\ k \in\# \mathcal{L}_{all}\ \mathcal{A})$;
$ASSERT(\textit{literals-are-in-}\mathcal{L}_{in}\ \mathcal{A}\ (\textit{mset } (NU \propto C)))$;
$ASSERT(\textit{length } (NU \propto C) \leq \textit{Suc } (\textit{uint32-max div 2}))$;
$ASSERT(\textit{len} \leq \textit{length } (NU \propto C))$; — makes the refinement easier
$\textit{let } C = NU \propto C$;
$\textit{if } i \geq \textit{len}$
$\textit{then}$
$RETURN(\textit{cach } (\textit{atm-of } (C\ !\ k) := \textit{SEEN-REMOVABLE}), \textit{butlast analyse}, \textit{True})$
$\textit{else do}$ {
$\textit{let } (L, \textit{analyse}) = \textit{get-literal-and-remove-of-analyse-wl2 } C \textit{ analyse}$;
$ASSERT(L \in\# \mathcal{L}_{all}\ \mathcal{A})$;
$\textit{let } b = \neg\textit{level-in-lbd } (\textit{get-level } M\ L)\ \textit{lbd}$;
$\textit{if } (\textit{get-level } M\ L = \textit{zero-uint32-nat} \vee$
$\textit{conflict-min-cach cach } (\textit{atm-of } L) = \textit{SEEN-REMOVABLE} \vee$
$\textit{atm-in-conflict } (\textit{atm-of } L)\ D)$
$\textit{then } RETURN\ (\textit{cach}, \textit{analyse}, \textit{False})$
$\textit{else if } b \vee \textit{conflict-min-cach cach } (\textit{atm-of } L) = \textit{SEEN-FAILED}$
$\textit{then do}$ {
$ASSERT(\textit{mark-failed-lits-stack-inv2 } NU \textit{ analyse cach})$;
$\textit{cach} \leftarrow \textit{mark-failed-lits-wl } NU \textit{ analyse cach}$;
$RETURN\ (\textit{cach}, [], \textit{False})$
}
$\textit{else do}$ {
$ASSERT(-\ L \in \textit{lits-of-l } M)$;
$C \leftarrow \textit{get-propagation-reason } M\ (-L)$;
$\textit{case } C \textit{ of}$
$\textit{Some } C \Rightarrow \textit{do}$ {
$ASSERT(C \in\# \textit{dom-m } NU)$;
$ASSERT(\textit{length } (NU \propto C) \geq \textit{2})$;
$ASSERT(\textit{literals-are-in-}\mathcal{L}_{in}\ \mathcal{A}\ (\textit{mset } (NU \propto C)))$;
$ASSERT(\textit{length } (NU \propto C) \leq \textit{Suc } (\textit{uint32-max div 2}))$;
$RETURN\ (\textit{cach}, \textit{analyse } @\ [\textit{lit-redundant-reason-stack2 } (-L)\ NU\ C], \textit{False})$
}
$\mid \textit{None} \Rightarrow \textit{do}$ {
$ASSERT(\textit{mark-failed-lits-stack-inv2 } NU \textit{ analyse cach})$;
$\textit{cach} \leftarrow \textit{mark-failed-lits-wl } NU \textit{ analyse cach}$;
$RETURN\ (\textit{cach}, [], \textit{False})$
}
}
}
})
$(\textit{cach}, \textit{analysis}, \textit{False})$›

lemma *lit-redundant-rec-wl-ref-butlast*:
‹$\textit{lit-redundant-rec-wl-ref } NU\ x \Longrightarrow \textit{lit-redundant-rec-wl-ref } NU\ (\textit{butlast } x)$›
⟨*proof*⟩

lemma *lit-redundant-rec-wl-lookup-mark-failed-lits-stack-inv*:
assumes
‹$(x, x') \in \textit{Id}$› **and**
‹$\textit{case } x \textit{ of } (\textit{cach}, \textit{analyse}, b) \Rightarrow \textit{analyse} \neq []$› **and**
‹$\textit{lit-redundant-rec-wl-inv } M\ NU\ D\ x'$› **and**
‹$\neg\ \textit{snd } (\textit{snd } (\textit{snd } (\textit{last } x1a))) \leq \textit{fst } (\textit{snd } (\textit{snd } (\textit{last } x1a)))$› **and**

‹get-literal-and-remove-of-analyse-wl (NU ∝ fst (last x1c)) x1c = (x1e, x2e)› **and**
‹x2 = (x1a, x2a)› **and**
‹x′ = (x1, x2)› **and**
‹x2b = (x1c, x2c)› **and**
‹x = (x1b, x2b)›
shows ‹mark-failed-lits-stack-inv NU x2e x1b›
⟨proof⟩

context
fixes M D 𝒜 NU analysis analysis′
assumes
M-D: ‹M ⊨as CNot D› **and**
n-d: ‹no-dup M› **and**
lits: ‹literals-are-in-$\mathcal{L}_{in}$-trail 𝒜 M› **and**
ana: ‹(analysis, analysis′) ∈ ana-lookups-rel NU› **and**
lits-NU: ‹literals-are-in-$\mathcal{L}_{in}$-mm 𝒜 ((mset ∘ fst) '# ran-m NU)› **and**
bounded: ‹isasat-input-bounded 𝒜›
begin
lemma ccmin-rel:
assumes ‹lit-redundant-rec-wl-inv M NU D (cach, analysis′, False)›
shows ‹((cach, analysis, False), cach, analysis′, False)
∈ {((cach, ana, b), cach′, ana′, b′).
(ana, ana′) ∈ ana-lookups-rel NU ∧
b = b′ ∧ cach = cach′ ∧ lit-redundant-rec-wl-inv M NU D (cach, ana′, b)}›
⟨proof⟩

context
fixes x :: ‹(nat ⇒ minimize-status) × (nat × nat × bool) list × bool› **and**
x′ :: ‹(nat ⇒ minimize-status) × (nat × nat × nat × nat) list × bool›
assumes x-x′: ‹(x, x′) ∈ {((cach, ana, b), (cach′, ana′, b′)).
(ana, ana′) ∈ ana-lookups-rel NU ∧ b = b′ ∧ cach = cach′ ∧
lit-redundant-rec-wl-inv M NU D (cach, ana′, b)}›
begin

lemma ccmin-lit-redundant-rec-wl-inv2:
assumes ‹lit-redundant-rec-wl-inv M NU D x′›
shows ‹lit-redundant-rec-wl-inv2 M NU D x›
⟨proof⟩

context
assumes
‹lit-redundant-rec-wl-inv2 M NU D x› **and**
‹lit-redundant-rec-wl-inv M NU D x′›
begin

lemma ccmin-cond:
fixes x1 :: ‹nat ⇒ minimize-status› **and**
x2 :: ‹(nat × nat × bool) list × bool› **and**
x1a :: ‹(nat × nat × bool) list› **and**
x2a :: ‹bool› **and** x1b :: ‹nat ⇒ minimize-status› **and**
x2b :: ‹(nat × nat × nat × nat) list × bool› **and**
x1c :: ‹(nat × nat × nat × nat) list› **and** x2c :: ‹bool›
assumes
‹x2 = (x1a, x2a)›
‹x = (x1, x2)›

‹$x2b = (x1c, x2c)$›
‹$x' = (x1b, x2b)$›
shows ‹$(x1a \neq []) = (x1c \neq [])$›
⟨*proof*⟩

end

context
assumes
‹*case x of* $(cach, analyse, b) \Rightarrow analyse \neq []$› **and**
‹*case x′ of* $(cach, analyse, b) \Rightarrow analyse \neq []$› **and**
inv2: ‹*lit-redundant-rec-wl-inv2 M NU D x*› **and**
‹*lit-redundant-rec-wl-inv M NU D x′*›
begin

context
fixes *x1* :: ‹$nat \Rightarrow$ *minimize-status*› **and**
x2 :: ‹$(nat \times nat \times nat \times nat)$ *list* $\times$ *bool*› **and**
x1a :: ‹$(nat \times nat \times nat \times nat)$ *list*› **and** *x2a* :: ‹*bool*› **and**
x1b :: ‹$nat \Rightarrow$ *minimize-status*› **and**
x2b :: ‹$(nat \times nat \times bool)$ *list* $\times$ *bool*› **and**
x1c :: ‹$(nat \times nat \times bool)$ *list*› **and**
x2c :: ‹*bool*›
assumes *st*:
‹$x2 = (x1a, x2a)$›
‹$x' = (x1, x2)$›
‹$x2b = (x1c, x2c)$›
‹$x = (x1b, x2b)$› **and**
x1a: ‹$x1a \neq []$›
begin

private lemma *st*:
‹$x2 = (x1a, x2a)$›
‹$x' = (x1, x1a, x2a)$›
‹$x2b = (x1c, x2a)$›
‹$x = (x1, x1c, x2a)$›
‹$x1b = x1$›
‹$x2c = x2a$› **and**
x1c: ‹$x1c \neq []$›
⟨*proof*⟩

lemma *ccmin-nempty*:
shows ‹$x1c \neq []$›
⟨*proof*⟩

context
notes -[*simp*] = *st*
fixes *x1d* :: ‹*nat*› **and** *x2d* :: ‹$nat \times nat \times nat$› **and**
x1e :: ‹*nat*› **and** *x2e* :: ‹$nat \times nat$› **and**
x1f :: ‹*nat*› **and**
x2f :: ‹*nat*› **and** *x1g* :: ‹*nat*› **and**
x2g :: ‹$nat \times nat \times nat$› **and**
x1h :: ‹*nat*› **and**
x2h :: ‹$nat \times nat$› **and**
x1i :: ‹*nat*› **and**

x2i :: ‹*nat*›
assumes
ana-lookup-conv: ‹*ana-lookup-conv NU* (*last x1c*) = (*x1g*, *x2g*)› **and**
last: ‹*last x1a* = (*x1d*, *x2d*)› **and**
dom: ‹*x1d* ∈# *dom-m NU*› **and**
le: ‹*x1e* < *length* (*NU* ∝ *x1d*)› **and**
in-lits: ‹*NU* ∝ *x1d* ! *x1e* ∈ *lits-of-l M*› **and**
st2:
‹*x2g* = (*x1h*, *x2h*)›
‹*x2e* = (*x1f*, *x2f*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x2h* = (*x1i*, *x2i*)›
begin

private lemma *x1g-x1d*:
‹*x1g* = *x1d*›
‹*x1h* = *x1e*›
‹*x1i* = *x1f*›
⟨*proof*⟩ **definition** *j* **where**
‹*j* = *fst* (*snd* (*last x1c*))›

private definition *b* **where**
‹*b* = *snd* (*snd* (*last x1c*))›

private lemma *last-x1c*[*simp*]:
‹*last x1c* = (*x1d*, *x1f*, *b*)›
⟨*proof*⟩ **lemma**
ana: ‹(*x1d*, (*if b then 1 else 0*), *x1f*, (*if b then 1 else length* (*NU* ∝ *x1d*))) = (*x1d*, *x1e*, *x1f*, *x2i*)› **and**
st3:
‹*x1e* = (*if b then 1 else 0*)›
‹*x1f* = *j*›
‹*x2f* = (*if b then 1 else length* (*NU* ∝ *x1d*))›
‹*x2d* = (*if b then 1 else 0*, *j*, *if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*j* ≤ (*if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*x1d* ∈# *dom-m NU*› **and**
‹*0* < *x1d*› **and**
‹(*if b then 1 else length* (*NU* ∝ *x1d*)) ≤ *length* (*NU* ∝ *x1d*)› **and**
‹(*if b then 1 else 0*) < *length* (*NU* ∝ *x1d*)› **and**
dist: ‹*distinct* (*NU* ∝ *x1d*)› **and**
tauto: ‹¬ *tautology* (*mset* (*NU* ∝ *x1d*))›
⟨*proof*⟩

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
⟨*proof*⟩

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
⟨*proof*⟩

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
⟨*proof*⟩

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›

⟨*proof*⟩

lemma *ccmin-le-uint32-max*:
‹$length\ (NU \propto x1g) \leq Suc\ (uint32\text{-}max\ div\ 2)$›
⟨*proof*⟩

lemma *ccmin-in-all-lits*:
shows ‹$NU \propto x1g\ !\ x1h \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›
⟨*proof*⟩

lemma *ccmin-less-length*:
shows ‹$x2i \leq length\ (NU \propto x1g)$›
⟨*proof*⟩

lemma *ccmin-same-cond*:
shows ‹$(x2i \leq x1i) = (x2f \leq x1f)$›
⟨*proof*⟩

lemma *list-rel-butlast*:
assumes *rel*: ‹$(xs, ys) \in \langle R\rangle list\text{-}rel$›
shows ‹$(butlast\ xs, butlast\ ys) \in \langle R\rangle list\text{-}rel$›
⟨*proof*⟩

lemma *ccmin-set-removable*:
assumes
‹$x2i \leq x1i$› **and**
‹$x2f \leq x1f$› **and** ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D\ x$›
shows ‹$((x1b(atm\text{-}of\ (NU \propto x1g\ !\ x1h) := SEEN\text{-}REMOVABLE), butlast\ x1c, True),$
$x1(atm\text{-}of\ (NU \propto x1d\ !\ x1e) := SEEN\text{-}REMOVABLE), butlast\ x1a, True)$
$\in \{((cach, ana, b), cach', ana', b').$
$(ana, ana') \in ana\text{-}lookups\text{-}rel\ NU \wedge$
$b = b' \wedge cach = cach' \wedge lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ (cach, ana', b)\}$›
⟨*proof*⟩

context
assumes
le: ‹$\neg\ x2i \leq x1i$› ‹$\neg\ x2f \leq x1f$›
begin

context
notes -[*simp*]= *x1g-x1d st2 last*
fixes *x1j* :: ‹*nat literal*› **and** *x2j* :: ‹$(nat \times nat \times nat \times nat)\ list$› **and**
x1k :: ‹*nat literal*› **and** *x2k* :: ‹$(nat \times nat \times bool)\ list$›
assumes
rem: ‹$get\text{-}literal\text{-}and\text{-}remove\text{-}of\text{-}analyse\text{-}wl\ (NU \propto x1d)\ x1a = (x1j, x2j)$› **and**
rem2:‹$get\text{-}literal\text{-}and\text{-}remove\text{-}of\text{-}analyse\text{-}wl2\ (NU \propto x1g)\ x1c = (x1k, x2k)$› **and**
‹$fst\ (snd\ (snd\ (last\ x2j))) \neq 0$› **and**
ux1j-M: ‹$-\ x1j \in lits\text{-}of\text{-}l\ M$›
begin

private lemma *confl-min-last*: ‹$(last\ x1c, last\ x1a) \in ana\text{-}lookup\text{-}rel\ NU$›
⟨*proof*⟩ **lemma** *rel*: ‹$(x1c[length\ x1c - Suc\ 0 := (x1d, Suc\ x1f, b)], x1a$
$[length\ x1a - Suc\ 0 := (x1d, x1e, Suc\ x1f, x2f)])$
$\in ana\text{-}lookups\text{-}rel\ NU$›
⟨*proof*⟩ **lemma** *x1k-x1j*: ‹$x1k = x1j$› ‹$x1j = NU \propto x1d\ !\ x1f$› **and**
x2k-x2j: ‹$(x2k, x2j) \in ana\text{-}lookups\text{-}rel\ NU$›

⟨*proof*⟩

lemma *ccmin-x1k-all*:
shows ‹$x1k \in\# \mathcal{L}_{all}\ \mathcal{A}$›
⟨*proof*⟩

context
notes -[*simp*]= *x1k-x1j*
fixes *b* :: ‹*bool*› **and** *lbd*
assumes *b*: ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd*, *b*) ∈ *bool-rel*›
begin

private lemma *in-conflict-atm-in*:
‹− *x1e′* ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e′*) *D* ⟷ *x1e′* ∈# *D*› **for** *x1e′*
⟨*proof*⟩

lemma *ccmin-already-seen*:
shows ‹(*get-level M x1k* = *zero-uint32-nat* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*) =
(*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j*, *False*)›
⟨*proof*⟩

lemma *ccmin-already-seen-rel*:
assumes
‹*get-level M x1k* = *zero-uint32-nat* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*› **and**
‹*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*›
shows ‹((*x1b*, *x2k*, *False*), *x1*, *x2j*, *False*)
∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
(*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
⟨*proof*⟩

context
assumes
‹¬ (*get-level M x1k* = *zero-uint32-nat* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*)› **and**
‹¬ (*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
begin
lemma *ccmin-already-failed*:
shows ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*) =
(*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
⟨*proof*⟩

context
assumes
‹¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*› **and**

‹$b \vee x1\ (atm\text{-}of\ x1j) = SEEN\text{-}FAILED$›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-lbd*:
shows ‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
⟨*proof*⟩

lemma *ccmin-mark-failed-lits-wl-lbd*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
$\leq \Downarrow Id$
(*mark-failed-lits-wl NU x2j x1*)›
⟨*proof*⟩

lemma *ccmin-rel-lbd*:
fixes *cach* :: ‹$nat \Rightarrow minimize\text{-}status$› **and** *cacha* :: ‹$nat \Rightarrow minimize\text{-}status$›
assumes ‹$(cach, cacha) \in Id$›
shows ‹$((cach, [], False), cacha, [], False) \in \{((cach, ana, b), cach', ana', b').$
$(ana, ana') \in ana\text{-}lookups\text{-}rel\ NU\ \wedge$
$b = b' \wedge cach = cach' \wedge lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ (cach, ana', b)\}$›
⟨*proof*⟩

end

context
assumes
‹$\neg (\neg\ level\text{-}in\text{-}lbd\ (get\text{-}level\ M\ x1k)\ lbd\ \vee$
$conflict\text{-}min\text{-}cach\ x1b\ (atm\text{-}of\ x1k) = SEEN\text{-}FAILED)$› **and**
‹$\neg (b \vee x1\ (atm\text{-}of\ x1j) = SEEN\text{-}FAILED)$›
begin

lemma *ccmin-lit-in-trail*:
‹$-\ x1k \in lits\text{-}of\text{-}l\ M$›
⟨*proof*⟩

lemma *ccmin-lit-eq*:
‹$-\ x1k = -\ x1j$›
⟨*proof*⟩

context
fixes *xa* :: ‹*nat option*› **and** *x'a* :: ‹*nat option*›
assumes *xa-x'a*: ‹$(xa, x'a) \in \langle nat\text{-}rel\rangle option\text{-}rel$›
begin

lemma *ccmin-lit-eq2*:
‹$(xa, x'a) \in Id$›
⟨*proof*⟩

context
assumes
[*simp*]: ‹$xa = None$› ‹$x'a = None$›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-dec*:

‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
⟨*proof*⟩

lemma *ccmin-mark-failed-lits-stack-wl-dec*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
$\le \Downarrow$ *Id*
(*mark-failed-lits-wl NU x2j x1*)›
⟨*proof*⟩

lemma *ccmin-rel-dec*:
fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
assumes ‹(*cach*, *cacha*) ∈ *Id*›
shows ‹((*cach*, [], *False*), *cacha*, [], *False*)
∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
(*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
⟨*proof*⟩

end

context
fixes *xb* :: ‹*nat*› **and** *x′b* :: ‹*nat*›
assumes *H*:
‹*xa* = *Some xb*›
‹*x′a* = *Some x′b*›
‹(*xb*, *x′b*) ∈ *nat-rel*›
‹*x′b* ∈# *dom-m NU*›
‹*2* ≤ *length* (*NU* ∝ *x′b*)›
‹*x′b* > *0*›
‹*distinct* (*NU* ∝ *x′b*) ∧ ¬ *tautology* (*mset* (*NU* ∝ *x′b*))›
begin

lemma *ccmin-stack-pre*:
shows ‹*xb* ∈# *dom-m NU*› ‹*2* ≤ *length* (*NU* ∝ *xb*)›
⟨*proof*⟩

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *xb*))›
⟨*proof*⟩

lemma *ccmin-le-uint32-max-xb*:
‹*length* (*NU* ∝ *xb*) ≤ *Suc* (*uint32-max div 2*)›
⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv3*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j* @ [*lit-redundant-reason-stack* (− *NU* ∝ *x1d* ! *x1f*) *NU x′b*], *False*)›
⟨*proof*⟩

lemma *ccmin-stack-rel*:
shows ‹((*x1b*, *x2k* @ [*lit-redundant-reason-stack2* (− *x1k*) *NU xb*], *False*), *x1*,
x2j @ [*lit-redundant-reason-stack* (− *x1j*) *NU x′b*], *False*)
∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
(*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
⟨*proof*⟩

end
end
end
end
end
end
end
end
end
end
end
end

lemma *lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*:
  assumes
    M-D: ‹$M \models as\ CNot\ D$› **and**
    n-d: ‹*no-dup M*› **and**
    lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
    ‹$(analysis, analysis') \in$ *ana-lookups-rel NU*› **and**
    ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**
    ‹*isasat-input-bounded* $\mathcal{A}$›
  shows
   ‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* ≤
     ⇓ (*Id* $\times_r$ (*ana-lookups-rel NU*) $\times_r$ *bool-rel*) (*lit-redundant-rec-wl M NU D cach analysis′ lbd*)›
⟨*proof*⟩

definition *literal-redundant-wl-lookup* **where**
  ‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* = *do* {
     ASSERT($L \in\# \mathcal{L}_{all}\ \mathcal{A}$);
     if get-level M L = *0* ∨ *cach* (*atm-of L*) = *SEEN-REMOVABLE*
     then RETURN (*cach*, [], *True*)
     else if cach (*atm-of L*) = *SEEN-FAILED*
     then RETURN (*cach*, [], *False*)
     else do {
       ASSERT(−*L* ∈ *lits-of-l M*);
       C ← *get-propagation-reason M* (−*L*);
       case C of
         Some C ⇒ *do* {
    ASSERT(*C* ∈# *dom-m NU*);
    ASSERT(*length* (*NU* ∝ *C*) ≥ *2*);
    ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *C*)));
    ASSERT(*distinct* (*NU* ∝ *C*) ∧ ¬*tautology* (*mset* (*NU* ∝ *C*)));
    ASSERT(*length* (*NU* ∝ *C*) ≤ *Suc* (*uint32-max div 2*));
    lit-redundant-rec-wl-lookup $\mathcal{A}$ *M NU D cach* [*lit-redundant-reason-stack2* (−*L*) *NU C*] *lbd*
  }
       | *None* ⇒ *do* {
           RETURN (*cach*, [], *False*)
       }
     }
  }›

lemma *literal-redundant-wl-lookup-literal-redundant-wl*:
  assumes ‹$M \models as\ CNot\ D$› ‹*no-dup M*› ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
    ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**

⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows
⟨*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* ≤
⇓ (*Id* $\times_f$ (*ana-lookups-rel NU* $\times_f$ *bool-rel*)) (*literal-redundant-wl M NU D cach L lbd*)⟩
⟨*proof*⟩

definition (**in** −) *lookup-conflict-nth* **where**
[*simp*]: ⟨*lookup-conflict-nth* = (λ(-, *xs*) *i*. *xs* ! *i*)⟩

definition (**in** −) *lookup-conflict-size* **where**
[*simp*]: ⟨*lookup-conflict-size* = (λ(*n*, *xs*). *n*)⟩

definition (**in** −) *lookup-conflict-upd-None* **where**
[*simp*]: ⟨*lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. (*n*−*1*, *xs* [*i* :=*None*]))⟩

definition *minimize-and-extract-highest-lookup-conflict*
:: ⟨*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒ (*nat* ⇒ *minimize-status*) ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause* × (*nat* ⇒ *minimize-status*) × *out-learned*) *nres*⟩
where
⟨*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ = (λ*M NU nxs s lbd outl*. *do* {
(*D*, -, *s*, *outl*) ←
$WHILE_T{}^{minimize\text{-}and\text{-}extract\text{-}highest\text{-}lookup\text{-}conflict\text{-}inv}$
(λ(*nxs*, *i*, *s*, *outl*). *i* < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(*x* < *length outl*);
let L = *outl* ! *x*;
ASSERT(*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
(*s'*, -, *red*) ← *literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x*+*1*, *s'*, *outl*)
else do {
ASSERT (*delete-from-lookup-conflict-pre* $\mathcal{A}$ (*L*, *nxs*));
RETURN (*remove1-mset L nxs*, *x*, *s'*, *delete-index-and-swap outl x*)
}
})
(*nxs*, *one-uint32-nat*, *s*, *outl*);
RETURN (*D*, *s*, *outl*)
})⟩

lemma *entails-uminus-filter-to-poslev-can-remove*:
assumes *NU-uL-E*: ⟨*NU* ⊨*p add-mset* (− *L*) (*filter-to-poslev M' L E*)⟩ **and**
NU-E: ⟨*NU* ⊨*p E*⟩ **and** *L-E*: ⟨*L* ∈# *E*⟩
shows ⟨*NU* ⊨*p remove1-mset L E*⟩
⟨*proof*⟩

lemma *minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*:
fixes *D* :: ⟨*nat clause*⟩ **and** *S'* :: ⟨*nat twl-st-l*⟩ **and** *NU* :: ⟨*nat clauses-l*⟩ **and** *S* :: ⟨*nat twl-st-wl*⟩
and *S''* :: ⟨*nat twl-st*⟩
defines
⟨*S'''* ≡ $state_W$*-of S''*⟩
defines
⟨*M* ≡ *get-trail-wl S*⟩ **and**
NU: ⟨*NU* ≡ *get-clauses-wl S*⟩ **and**
NU'-def: ⟨*NU'* ≡ *mset* '# *ran-mf NU*⟩ **and**

NUE: ‹$NUE \equiv$ *get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*› **and**
M': ‹$M' \equiv$ *trail S'''*›
assumes
S-S': ‹$(S, S') \in$ *state-wl-l None*› **and**
S'-S'': ‹$(S', S'') \in$ *twl-st-l None*› **and**
D'-D: ‹*mset* (*tl outl*) = *D*› **and**
M-D: ‹*M* $\models$*as CNot D*› **and**
dist-D: ‹*distinct-mset D*› **and**
tauto: ‹$\neg$*tautology D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
struct-invs: ‹*twl-struct-invs S''*› **and**
add-inv: ‹*twl-list-invs S'*› **and**
cach-init: ‹*conflict-min-analysis-inv M' s'* (*NU'* + *NUE*) *D*› **and**
NU-P-D: ‹*NU'* + *NUE* $\models$*pm add-mset K D*› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf NU*)› **and**
K: ‹*K* = *outl* ! *0*› **and**
outl-nempty: ‹*outl* $\neq$ []› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ *M NU D s' lbd outl* $\leq$
$\Downarrow$ ({((*E*, *s*, *outl*), *E'*). *E* = *E'* $\wedge$ *mset* (*tl outl*) = *E* $\wedge$ *outl* ! *0* = *K* $\wedge$
E' $\subseteq$# *D* $\wedge$ *outl* $\neq$ []})
(*iterate-over-conflict K M NU' NUE D*)›
(**is** ‹- $\leq \Downarrow$ *?R* -›)
⟨*proof*⟩

definition *cach-refinement-list*
:: ‹*nat multiset* $\Rightarrow$ (*minimize-status list* $\times$ (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement-list* $\mathcal{A}_{in}$ = ⟨*Id*⟩*map-fun-rel* {(*a*, *a'*). *a* = *a'* $\wedge$ *a* $\in$# $\mathcal{A}_{in}$}›

definition *cach-refinement-nonull*
:: ‹*nat multiset* $\Rightarrow$ ((*minimize-status list* $\times$ *nat list*) $\times$ *minimize-status list*) *set*›
where
‹*cach-refinement-nonull* $\mathcal{A}$ = {((*cach*, *support*), *cach'*). *cach* = *cach'* $\wedge$
($\forall$ *L* < *length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longleftrightarrow$ *L* $\in$ *set support*) $\wedge$
($\forall$ *L* $\in$ *set support*. *L* < *length cach*) $\wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}$}›

definition *cach-refinement*
:: ‹*nat multiset* $\Rightarrow$ ((*minimize-status list* $\times$ *nat list*) $\times$ (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement* $\mathcal{A}_{in}$ = *cach-refinement-nonull* $\mathcal{A}_{in}$ *O cach-refinement-list* $\mathcal{A}_{in}$›

lemma *cach-refinement-alt-def*:
‹*cach-refinement* $\mathcal{A}_{in}$ = {((*cach*, *support*), *cach'*).
($\forall$ *L* < *length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longleftrightarrow$ *L* $\in$ *set support*) $\wedge$
($\forall$ *L* $\in$ *set support*. *L* < *length cach*) $\wedge$
($\forall$ *L* $\in$# $\mathcal{A}_{in}$. *L* < *length cach* $\wedge$ *cach* ! *L* = *cach' L*) $\wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}_{in}$}›
⟨*proof*⟩

lemma *in-cach-refinement-alt-def*:
‹((*cach*, *support*), *cach'*) $\in$ *cach-refinement* $\mathcal{A}_{in}$ $\longleftrightarrow$

$(cach,\ cach') \in$ *cach-refinement-list* $\mathcal{A}_{in} \wedge$
$(\forall L{<}$*length cach. cach* ! $L \neq$ *SEEN-UNKNOWN* $\longleftrightarrow L \in$ *set support*$) \wedge$
$(\forall L \in$ *set support.* $L <$ *length cach*$) \wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}_{in}$›
⟨*proof*⟩

definition (**in** −) *conflict-min-cach-l* :: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *minimize-status*› **where**
‹*conflict-min-cach-l* = (λ(*cach*, *sup*) *L*.
(*cach* ! *L*)
)›

definition *conflict-min-cach-l-pre* **where**
‹*conflict-min-cach-l-pre* = (λ((*cach*, *sup*), *L*). *L* < *length cach*)›

lemma *conflict-min-cach-l-pre*:
fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
assumes
‹*x1n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*x1l*, *x1j*) ∈ *cach-refinement* $\mathcal{A}$›
shows ‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1n*)›
⟨*proof*⟩

lemma *nth-conflict-min-cach*:
‹(*uncurry* (*RETURN oo conflict-min-cach-l*), *uncurry* (*RETURN oo conflict-min-cach*)) ∈
$[\lambda(cach,\ L).\ L \in\#\ \mathcal{A}_{in}]_f$ *cach-refinement* $\mathcal{A}_{in} \times_r$ *nat-rel* → ⟨*minimize-status-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *conflict-min-cach-set-failed*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where
[*simp*]: ‹*conflict-min-cach-set-failed cach L* = *cach*(*L* := *SEEN-FAILED*)›

definition (**in** −) *conflict-min-cach-set-failed-l*
:: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *conflict-min-cach-l nres*›
where
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if cach* ! *L* = *SEEN-UNKNOWN then sup* @ [*L*] *else sup*)
})›

lemma *bounded-included-le*:
assumes *bounded*: ‹*isasat-input-bounded* $\mathcal{A}$› **and** ‹*distinct n*› **and** ‹*set n* ⊆ *set-mset* $\mathcal{A}$›
shows ‹*length n* ≤ *Suc* (*uint32-max div 2*)›
⟨*proof*⟩

lemma *conflict-min-cach-set-failed*:
‹(*uncurry conflict-min-cach-set-failed-l*, *uncurry* (*RETURN oo conflict-min-cach-set-failed*)) ∈
$[\lambda(cach,\ L).\ L \in\#\ \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *cach-refinement* $\mathcal{A}_{in} \times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *conflict-min-cach-set-removable*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where

[*simp*]: ‹*conflict-min-cach-set-removable cach L = cach*(*L*:= *SEEN-REMOVABLE*)›

lemma *conflict-min-cach-set-removable*:
‹(*uncurry conflict-min-cach-set-removable-l*,
uncurry (*RETURN oo conflict-min-cach-set-removable*)) ∈
$[\lambda(cach, L).\ L \in\# \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *cach-refinement* $\mathcal{A}_{in} \times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

definition *analyse-refinement-rel* **where**
‹*analyse-refinement-rel* = *nat-rel* $\times_f$ {(*n*, (*L*, *b*)). ∃ *L'*. (*L'*, *L*) ∈ *uint32-nat-rel* ∧
n = *uint64-of-uint32 L'* + (*if b then 1* << *32 else 0*)}›

definition *to-ana-ref-id* **where**
[*simp*]: ‹*to-ana-ref-id* = (λ*a b c*. (*a*, *b*, *c*))›

definition *to-ana-ref* :: ‹- ⇒ *uint32* ⇒ *bool* ⇒ -› **where**
‹*to-ana-ref* = (λ*a c b*. (*a*, *uint64-of-uint32 c OR* (*if b then 1* << *32 else* (*0* :: *uint64*))))›

definition *from-ana-ref-id* **where**
[*simp*]: ‹*from-ana-ref-id x* = *x*›

definition *from-ana-ref* **where**
‹*from-ana-ref* = (λ(*a*, *b*). (*a*, *uint32-of-uint64* (*take-only-lower32 b*), *is-marked-binary-code* (*a*, *b*)))›

definition *isa-mark-failed-lits-stack* **where**
‹*isa-mark-failed-lits-stack NU analyse cach* = *do* {
let l = *length analyse*;
ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
(-, *cach*) ← $WHILE_T{}^{\lambda(\text{-},\ cach).\ True}$
(λ(*i*, *cach*). *i* < *l*)
(λ(*i*, *cach*). *do* {
ASSERT(*i* < *length analyse*);
let (*cls-idx*, *idx*, -) = *from-ana-ref-id* (*analyse* ! *i*);
ASSERT(*cls-idx* + *idx* ≥ *1*);
ASSERT(*cls-idx* + *idx* − *1* < *length NU*);
ASSERT(*arena-lit-pre NU* (*cls-idx* + *idx* − *1*));
cach ← *conflict-min-cach-set-failed-l cach* (*atm-of* (*arena-lit NU* (*cls-idx* + *idx* − *1*)));
RETURN (*i+1*, *cach*)
})
(*0*, *cach*);
RETURN cach
}›

context
begin
lemma *mark-failed-lits-stack-inv-helper1*: ‹*mark-failed-lits-stack-inv a ba a2'* ⟹
a1' < *length ba* ⟹
(*a1'a*, *a2'a*) = *ba* ! *a1'* ⟹
a1'a ∈# *dom-m a*›
⟨*proof*⟩

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2'* ⟹
a1' < *length ba* ⟹

$(a1'a,\ xx,\ a2'a,\ yy) = ba\ !\ a1' \Longrightarrow$
$a2'a - Suc\ 0 < length\ (a \propto a1'a)$›
⟨*proof*⟩

lemma *isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*:
assumes ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
shows ‹(*uncurry2 isa-mark-failed-lits-stack*, *uncurry2* (*mark-failed-lits-stack* $\mathcal{A}_{in}$)) ∈
[λ((*N*, *ana*), *cach*). *length ana* ≤ *1* + *uint32-max div 2*]$_f$
{(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *ana-lookups-rel NU* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$ →
⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

definition *isa-get-literal-and-remove-of-analyse-wl*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *nat literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl C analyse* =
(*let* (*i*, *j*, *b*) = *from-ana-ref-id* (*last analyse*) *in*
(*arena-lit C* (*i* + *j*), *analyse*[*length analyse* − *1* := *to-ana-ref-id i* (*j* + *1*) *b*]))›

definition *isa-get-literal-and-remove-of-analyse-wl-pre*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *bool*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ⟷
(*let* (*i*, *j*, *b*) = *last analyse in*
analyse ≠ [] ∧ *arena-lit-pre arena* (*i*+*j*) ∧ *j* < *uint32-max*)›

lemma *arena-lit-pre-le*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* ≤ *uint64-max*›
⟨*proof*⟩

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* < *uint64-max*›
⟨*proof*⟩

definition *lit-redundant-reason-stack-wl-lookup-pre* :: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *bool*› **where**
‹*lit-redundant-reason-stack-wl-lookup-pre L NU C* ⟷
arena-lit-pre NU C ∧
arena-is-valid-clause-idx NU C›

definition *lit-redundant-reason-stack-wl-lookup*
:: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *nat* × *nat* × *bool*›
where
‹*lit-redundant-reason-stack-wl-lookup L NU C* =
(*if arena-length NU C* > *2 then to-ana-ref-id C 1 False*
else if arena-lit NU C = *L*
then to-ana-ref-id C 1 False
else to-ana-ref-id C 0 True)›

definition *ana-lookup-conv-lookup* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv-lookup NU* = (λ(*C*, *i*, *b*).
(*C*, (*if b then 1 else 0*), *i*, (*if b then 1 else arena-length NU C*)))›

definition *ana-lookup-conv-lookup-pre* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ *bool*› **where**
‹*ana-lookup-conv-lookup-pre NU* = (λ(*C*, *i*, *b*). *arena-is-valid-clause-idx NU C*)›

definition *isa-lit-redundant-rec-wl-lookup*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒

- ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›

where

‹*isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd* =
  $WHILE_T{}^{\lambda\text{-}.\ True}$
    (λ(*cach*, *analyse*, *b*). *analyse* ≠ [])
    (λ(*cach*, *analyse*, *b*). *do* {
      ASSERT(*analyse* ≠ []);
      ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
      ASSERT(*arena-is-valid-clause-idx NU* (*fst* (*last analyse*)));
  ASSERT(*ana-lookup-conv-lookup-pre NU* (*from-ana-ref-id* (*last analyse*)));
  let (*C*, *k*, *i*, *len*) = *ana-lookup-conv-lookup NU* (*from-ana-ref-id* (*last analyse*));
      ASSERT(*C* < *length NU*);
      ASSERT(*arena-is-valid-clause-idx NU C*);
      ASSERT(*arena-lit-pre NU* (*C* + *k*));
      if i ≥ *nat-of-uint64-conv len*
      then do {
    cach ← *conflict-min-cach-set-removable-l cach* (*atm-of* (*arena-lit NU* (*C* + *k*)));
        RETURN(*cach*, *butlast analyse*, *True*)
  }
      else do {
    ASSERT (*isa-get-literal-and-remove-of-analyse-wl-pre NU analyse*);
    let (*L*, *analyse*) = *isa-get-literal-and-remove-of-analyse-wl NU analyse*;
        ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
    ASSERT(*get-level-pol-pre* (*M*, *L*));
    let b = ¬*level-in-lbd* (*get-level-pol M L*) *lbd*;
    ASSERT(*atm-in-conflict-lookup-pre* (*atm-of L*) *D*);
    ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
    if (*get-level-pol M L* = *zero-uint32-nat* ∨
   conflict-min-cach-l cach (*atm-of L*) = *SEEN-REMOVABLE* ∨
   atm-in-conflict-lookup (*atm-of L*) *D*)
    then RETURN (*cach*, *analyse*, *False*)
    else if b ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-FAILED*
    then do {
 cach ← *isa-mark-failed-lits-stack NU analyse cach*;
 RETURN (*cach*, [], *False*)
    }
    else do {
 C ← *get-propagation-reason-pol M* (−*L*);
 case C of
  Some C ⇒ *do* {
    ASSERT(*lit-redundant-reason-stack-wl-lookup-pre* (−*L*) *NU C*);
    RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack-wl-lookup* (−*L*) *NU C*], *False*)
  }
 | *None* ⇒ *do* {
    cach ← *isa-mark-failed-lits-stack NU analyse cach*;
    RETURN (*cach*, [], *False*)
    }
      }
    }
  })
  (*cach*, *analysis*, *False*)›

lemma *isa-lit-redundant-rec-wl-lookup-alt-def*:
‹*isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd* =
  $WHILE_T{}^{\lambda\text{-}.\ True}$
    (λ(*cach*, *analyse*, *b*). *analyse* ≠ [])

(λ(*cach*, *analyse*, *b*). *do* {
ASSERT(*analyse* ≠ []);
ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
let (*C*, *i*, *b*) = *last analyse*;
ASSERT(*arena-is-valid-clause-idx NU* (*fst* (*last analyse*)));
ASSERT(*ana-lookup-conv-lookup-pre NU* (*from-ana-ref-id* (*last analyse*)));
let (*C*, *k*, *i*, *len*) = *ana-lookup-conv-lookup NU* (*from-ana-ref-id* (*C*, *i*, *b*));
ASSERT(*C* < *length NU*);
let - = *map xarena-lit*
((*Misc.slice*
C
(*C* + *arena-length NU C*))
NU);
ASSERT(*arena-is-valid-clause-idx NU C*);
ASSERT(*arena-lit-pre NU* (*C* + *k*));
if i ≥ *nat-of-uint64-conv len*
then do {
cach ← *conflict-min-cach-set-removable-l cach* (*atm-of* (*arena-lit NU* (*C* + *k*)));
RETURN(*cach*, *butlast analyse*, *True*)
}
else do {
ASSERT (*isa-get-literal-and-remove-of-analyse-wl-pre NU analyse*);
let (*L*, *analyse*) = *isa-get-literal-and-remove-of-analyse-wl NU analyse*;
ASSERT(*length analyse* ≤ *1*+ *uint32-max div 2*);
ASSERT(*get-level-pol-pre* (*M*, *L*));
let b = ¬*level-in-lbd* (*get-level-pol M L*) *lbd*;
ASSERT(*atm-in-conflict-lookup-pre* (*atm-of L*) *D*);
ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
if (*get-level-pol M L* = *zero-uint32-nat* ∨
conflict-min-cach-l cach (*atm-of L*) = *SEEN-REMOVABLE* ∨
atm-in-conflict-lookup (*atm-of L*) *D*)
then RETURN (*cach*, *analyse*, *False*)
else if b ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-FAILED*
then do {
cach ← *isa-mark-failed-lits-stack NU analyse cach*;
RETURN (*cach*, [], *False*)
}
else do {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*lit-redundant-reason-stack-wl-lookup-pre* (−*L*) *NU C*);
RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack-wl-lookup* (−*L*) *NU C*], *False*)
}
| *None* ⇒ *do* {
cach ← *isa-mark-failed-lits-stack NU analyse cach*;
RETURN (*cach*, [], *False*)
}
}
}
})
(*cach*, *analysis*, *False*)›
⟨*proof*⟩

lemma *lit-redundant-rec-wl-lookup-alt-def*:
‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* =

$WHILE_T{}^{lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D}$
  (λ(*cach*, *analyse*, *b*). *analyse* $\neq$ [])
  (λ(*cach*, *analyse*, *b*). *do* {
    ASSERT(*analyse* $\neq$ []);
    ASSERT(*length analyse* $\leq$ *length M*);
let (*C*, *k*, *i*, *len*) = *ana-lookup-conv NU* (*last analyse*);
    ASSERT(*C* $\in\#$ *dom-m NU*);
    ASSERT(*length* (*NU* $\propto$ *C*) > *k*); — >= 2 would work too
    ASSERT (*NU* $\propto$ *C* ! *k* $\in$ *lits-of-l M*);
    ASSERT(*NU* $\propto$ *C* ! *k* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* $\propto$ *C*)));
ASSERT(*length* (*NU* $\propto$ *C*) $\leq$ *Suc* (*uint32-max div 2*));
ASSERT(*len* $\leq$ *length* (*NU* $\propto$ *C*)); — makes the refinement easier
let (*C*,*k*, *i*, *len*) = (*C*,*k*,*i*,*len*);
    let C = *NU* $\propto$ *C*;
    if i $\geq$ *len*
    then
      RETURN(*cach* (*atm-of* (*C* ! *k*) := *SEEN-REMOVABLE*), *butlast analyse*, *True*)
    else do {
      let (*L*, *analyse*) = *get-literal-and-remove-of-analyse-wl2 C analyse*;
      ASSERT(*L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$);
      let b = $\neg$*level-in-lbd* (*get-level M L*) *lbd*;
      if (*get-level M L* = *zero-uint32-nat* $\vee$
        conflict-min-cach cach (*atm-of L*) = *SEEN-REMOVABLE* $\vee$
        atm-in-conflict (*atm-of L*) *D*)
      then RETURN (*cach*, *analyse*, *False*)
      else if b $\vee$ *conflict-min-cach cach* (*atm-of L*) = *SEEN-FAILED*
      then do {
        ASSERT(*mark-failed-lits-stack-inv2 NU analyse cach*);
        cach $\leftarrow$ *mark-failed-lits-wl NU analyse cach*;
        RETURN (*cach*, [], *False*)
      }
      else do {
  ASSERT($-$ *L* $\in$ *lits-of-l M*);
        C $\leftarrow$ *get-propagation-reason M* ($-$*L*);
        case C of
          Some C $\Rightarrow$ *do* {
 ASSERT(*C* $\in\#$ *dom-m NU*);
 ASSERT(*length* (*NU* $\propto$ *C*) $\geq$ *2*);
 ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* $\propto$ *C*)));
          ASSERT(*length* (*NU* $\propto$ *C*) $\leq$ *Suc* (*uint32-max div 2*));
 RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack2* ($-$*L*) *NU C*], *False*)
}
        | *None* $\Rightarrow$ *do* {
          ASSERT(*mark-failed-lits-stack-inv2 NU analyse cach*);
          cach $\leftarrow$ *mark-failed-lits-wl NU analyse cach*;
          RETURN (*cach*, [], *False*)
        }
      }
    }
  })
  (*cach*, *analysis*, *False*)›
⟨*proof*⟩

lemma *valid-arena-nempty*:
‹*valid-arena arena N vdom* $\Longrightarrow$ *i* $\in\#$ *dom-m N* $\Longrightarrow$ *N* $\propto$ *i* $\neq$ []›

⟨*proof*⟩

lemma *isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-lit-redundant-rec-wl-lookup*, *uncurry5* (*lit-redundant-rec-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) '# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *iterate-over-conflict-spec*:
fixes *D* :: ‹'*v clause*›
assumes ‹*NU* + *NUE* ⊨*pm add-mset K D*› **and** *dist*: ‹*distinct-mset D*›
shows
‹*iterate-over-conflict K M NU NUE D* ≤ ⇓ *Id* (*SPEC*(λ*D*'. *D*' ⊆# *D* ∧
NU + *NUE* ⊨*pm add-mset K D*'))›
⟨*proof*⟩

end

lemma
fixes *D* :: ‹*nat clause*› **and** *s* **and** *s*' **and** *NU* :: ‹*nat clauses-l*› **and**
S :: ‹*nat twl-st-wl*› **and** *S*' :: ‹*nat twl-st-l*› **and** *S*'' :: ‹*nat twl-st*›
defines
‹*S*''' ≡ *state*$_W$*-of S*''›
defines
‹*M* ≡ *get-trail-wl S*› **and**
NU: ‹*NU* ≡ *get-clauses-wl S*› **and**
NU'-def: ‹*NU*' ≡ *mset* '# *ran-mf NU*› **and**
NUE: ‹*NUE* ≡ *get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*› **and**
M': ‹*M*' ≡ *trail S*'''›
assumes
S-S': ‹(*S*, *S*') ∈ *state-wl-l None*› **and**
S'-S'': ‹(*S*', *S*'') ∈ *twl-st-l None*› **and**
D'-D: ‹*mset* (*tl outl*) = *D*› **and**
M-D: ‹*M* ⊨*as CNot D*› **and**
dist-D: ‹*distinct-mset D*› **and**
tauto: ‹¬*tautology D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
struct-invs: ‹*twl-struct-invs S*''› **and**
add-inv: ‹*twl-list-invs S*'› **and**
cach-init: ‹*conflict-min-analysis-inv M*' *s*' (*NU*' + *NUE*) *D*› **and**
NU-P-D: ‹*NU*' + *NUE* ⊨*pm add-mset K D*› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf NU*)› **and**
K: ‹*K* = *outl* ! *0*› **and**
outl-nempty: ‹*outl* ≠ []› **and**
‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ *M NU D s*' *lbd outl* ≤
⇓ ({((*E*, *s*, *outl*), *E*'). *E* = *E*' ∧ *mset* (*tl outl*) = *E* ∧ *outl*!*0* = *K* ∧
E' ⊆# *D*})
(*SPEC* (λ*D*'. *D*' ⊆# *D* ∧ *NU*' + *NUE* ⊨*pm add-mset K D*'))›

⟨*proof*⟩

lemma (**in** −) *lookup-conflict-upd-None-RETURN-def*:
‹*RETURN oo lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. *RETURN* (*n*− *one-uint32-nat*, *xs* [*i* := *NOTIN*]))›
⟨*proof*⟩

definition *isa-literal-redundant-wl-lookup* ::
trail-pol ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l*
⇒ *nat literal* ⇒ *lbd* ⇒ (*conflict-min-cach-l* × (*nat* × *nat* × *bool*) *list* × *bool*) *nres*
where
‹*isa-literal-redundant-wl-lookup M NU D cach L lbd* = *do* {
ASSERT(*get-level-pol-pre* (*M*, *L*));
ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
if get-level-pol M L = *0* ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-REMOVABLE*
then RETURN (*cach*, [], *True*)
else if conflict-min-cach-l cach (*atm-of L*) = *SEEN-FAILED*
then RETURN (*cach*, [], *False*)
else do {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*lit-redundant-reason-stack-wl-lookup-pre* (−*L*) *NU C*);
isa-lit-redundant-rec-wl-lookup M NU D cach
[*lit-redundant-reason-stack-wl-lookup* (−*L*) *NU C*] *lbd*}
| *None* ⇒ *do* {
RETURN (*cach*, [], *False*)
}
}
}›

lemma *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*[*intro*]: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of L* ∈# $\mathcal{A}$›
⟨*proof*⟩

lemma *isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-literal-redundant-wl-lookup*, *uncurry5* (*literal-redundant-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$ *cach-refinement*
$\mathcal{A}$
$\times_f$ *Id* $\times_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *lookup-conflict-remove1* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*lookup-conflict-remove1* =
(λ*L* (*n*,*xs*). (*n*−*1*, *xs* [*atm-of L* := *NOTIN*]))›

lemma *lookup-conflict-remove1*:
‹(*uncurry* (*RETURN oo lookup-conflict-remove1*), *uncurry* (*RETURN oo remove1-mset*))
∈ [λ(*L*,*C*). *L* ∈# *C* ∧ −*L* ∉# *C* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id $\times_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *lookup-conflict-remove1-pre* :: ‹*nat literal* × *nat* × *bool option list* ⇒ *bool*› **where**
‹*lookup-conflict-remove1-pre* = (λ(*L*,(*n*,*xs*)). *n* > *0* ∧ *atm-of L* < *length xs*)›

definition *isa-minimize-and-extract-highest-lookup-conflict*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l* ⇒ *lbd* ⇒
out-learned ⇒ (*lookup-clause-rel* × *conflict-min-cach-l* × *out-learned*) *nres*›
where
‹*isa-minimize-and-extract-highest-lookup-conflict* = (λ*M NU nxs s lbd outl*. *do* {
(*D*, -, *s*, *outl*) ←
$WHILE_T{}^{\lambda(nxs,\ i,\ s,\ outl).\ length\ outl\ \leq\ uint32\text{-}max}$
(λ(*nxs*, *i*, *s*, *outl*). *i* < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(*x* < *length outl*);
let L = *outl* ! *x*;
(*s′*, -, *red*) ← *isa-literal-redundant-wl-lookup M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x*+*1*, *s′*, *outl*)
else do {
ASSERT(*lookup-conflict-remove1-pre* (*L*, *nxs*));
RETURN (*lookup-conflict-remove1 L nxs*, *x*, *s′*, *delete-index-and-swap outl x*)
}
})
(*nxs*, *one-uint32-nat*, *s*, *outl*);
RETURN (*D*, *s*, *outl*)
})›

lemma *isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*,
uncurry5 (*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$)) ∈
[λ(((((-, *N*), *D*), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*) ∧
¬*tautology D*]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* →
⟨*lookup-clause-rel* $\mathcal{A}$ $\times_r$ *cach-refinement* $\mathcal{A}$ $\times_r$ *Id*⟩*nres-rel*›
⟨*proof*⟩

definition *set-empty-conflict-to-none* **where**
‹*set-empty-conflict-to-none D* = *None*›

definition *set-lookup-empty-conflict-to-none* **where**
‹*set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). (*True*, *n*, *xs*))›

lemma *set-empty-conflict-to-none-hnr*:
‹(*RETURN o set-lookup-empty-conflict-to-none*, *RETURN o set-empty-conflict-to-none*) ∈
[λ*D*. *D* = {#}]$_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *lookup-merge-eq2*
:: ‹*nat literal* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*› **where**
‹*lookup-merge-eq2 L M N* = (λ(-, *zs*) *clvls lbd outl*. *do* {
ASSERT(*length N* = *2*);
let L′ = (*if N* ! *0* = *L then N* ! *1 else N* ! *0*);
ASSERT(*get-level M L′* ≤ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level M L′*);

ASSERT(*atm-of* L' < *length* (*snd zs*));
ASSERT(*length outl* < *uint32-max*);
let outl = *outlearned-add M* L' *zs outl*;
ASSERT(*clvls* < *uint32-max*);
ASSERT(*fst zs* < *uint32-max*);
let clvls = *clvls-add M* L' *zs clvls*;
let zs = *add-to-lookup-conflict* L' *zs*;
RETURN((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

definition *merge-conflict-m-eq2*
:: ‹*nat literal* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m-eq2 L M Ni D* =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
n = *card-max-lvl M* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
out-learned M C outl)›

lemma *lookup-merge-eq2-spec*:
assumes
o: ‹((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› **and**
lits-tr: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
n-d: ‹*no-dup M*› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C*› **and**
no-tauto: ‹⋀*K*. *K* ∈ *set* (*remove1 L D*) ⟹ − *K* ∉# *C*›
‹*clvls* = *card-max-lvl M C*› **and**
out: ‹*out-learned M* (*Some C*) *outl*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
le2: ‹*length D* = *2*› **and**
L-D: ‹*L* ∈ *set D*›
shows
‹*lookup-merge-eq2 L M D* (*b*, *n*, *xs*) *clvls lbd outl* ≤
⇓(*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*merge-conflict-m-eq2 L M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
⟨*proof*⟩

definition *isasat-lookup-merge-eq2*
:: ‹*nat literal* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*› **where**
‹*isasat-lookup-merge-eq2 L M N C* = (λ(-, *zs*) *clvls lbd outl*. *do* {
ASSERT(*arena-lit-pre N C*);
ASSERT(*arena-lit-pre N* (*C*+*1*));
let L' = (*if arena-lit N C* = *L then arena-lit N* (*C* + *1*) *else arena-lit N C*);
ASSERT(*get-level-pol-pre* (*M*, L'));
ASSERT(*get-level-pol M* L' ≤ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level-pol M* L');
ASSERT(*atm-of* L' < *length* (*snd zs*));
ASSERT(*length outl* < *uint32-max*);
let outl = *isa-outlearned-add M* L' *zs outl*;
ASSERT(*clvls* < *uint32-max*);
ASSERT(*fst zs* < *uint32-max*);

let clvls = isa-clvls-add M L′ zs clvls;
let zs = add-to-lookup-conflict L′ zs;
RETURN((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

lemma *isasat-lookup-merge-eq2-lookup-merge-eq2*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
bxs: ‹((*b*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
M′M: ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isasat-lookup-merge-eq2 L M′ arena i* (*b*, *xs*) *clvls lbd outl* ≤ ⇓ *Id*
(*lookup-merge-eq2 L M* (*N* ∝ *i*) (*b*, *xs*) *clvls lbd outl*)›
⟨*proof*⟩

definition *merge-conflict-m-eq2-pre* **where**
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ =
(λ(((((((*L*, *M*), *N*), *i*), *xs*), *clvls*), *lbd*), *out*). *i* ∈# *dom-m N* ∧ *xs* ≠ *None* ∧ *distinct* (*N* ∝ *i*) ∧
¬*tautology* (*mset* (*N* ∝ *i*)) ∧
(∀ *K* ∈ *set* (*remove1 L* (*N* ∝ *i*)). − *K* ∉# *the xs*) ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) ∧ *clvls* = *card-max-lvl M* (*the xs*) ∧
out-learned M xs out ∧ *no-dup M* ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) ∧
isasat-input-bounded $\mathcal{A}$ ∧
length (*N* ∝ *i*) = *2* ∧
L ∈ *set* (*N* ∝ *i*))›

definition *merge-conflict-m-g-eq2* :: ‹-› **where**
‹*merge-conflict-m-g-eq2 L M N i D* - - - = *merge-conflict-m-eq2 L M* (*N* ∝ *i*) *D*›

lemma *isasat-lookup-merge-eq2*:
‹(*uncurry7 isasat-lookup-merge-eq2*, *uncurry7 merge-conflict-m-g-eq2*) ∈
[*merge-conflict-m-eq2-pre* $\mathcal{A}$]$_f$
Id $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel*
$\mathcal{A}$
$\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$ *Id* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-Setup*
imports
Watched-Literals-VMTF
Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Lookup-Conflict
IsaSAT-Clauses IsaSAT-Arena IsaSAT-Watch-List LBD
begin

TODO Move and make sure to merge in the right order!

no-notation *Ref.update* (- := - *62*)

0.1.9 Code Generation

We here define the last step of our refinement: the step with all the heuristics and fully deterministic code.

After the result of benchmarking, we concluded that the us of *nat* leads to worse performance than using *uint64*. As, however, the later is not complete, we do so with a switch: as long as it fits, we use the faster (called 'bounded') version. After that we switch to the 'unbounded' version (which is still bounded by memory anyhow).

We do keep some natural numbers:

1. to iterate over the watch list. Our invariant are currently not strong enough to prove that we do not need that.

2. to keep the indices of all clauses. This mostly simplifies the code if we add inprocessing: We can be sure to never have to switch mode in the middle of an operation (which would nearly impossible to do).

Types and Refinement Relations

Statistics We do some statistics on the run.

NB: the statistics are not proven correct (especially they might overflow), there are just there to look for regressions, do some comparisons (e.g., to conclude that we are propagating slower than the other solvers), or to test different option combination.

type-synonym *stats* = ‹*uint64* × *uint64* × *uint64* × *uint64* × *uint64* × *uint64* × *uint64* × *uint64*›

definition *incr-propagation* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-propagation* = (λ(*propa*, *confl*, *dec*). (*propa* + *1*, *confl*, *dec*))›

definition *incr-conflict* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-conflict* = (λ(*propa*, *confl*, *dec*). (*propa*, *confl* + *1*, *dec*))›

definition *incr-decision* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-decision* = (λ(*propa*, *confl*, *dec*, *res*). (*propa*, *confl*, *dec* + *1*, *res*))›

definition *incr-restart* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-restart* = (λ(*propa*, *confl*, *dec*, *res*, *lres*). (*propa*, *confl*, *dec*, *res* + *1*, *lres*))›

definition *incr-lrestart* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-lrestart* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*). (*propa*, *confl*, *dec*, *res*, *lres* + *1*, *uset*))›

definition *incr-uset* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-uset* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, (*uset*, *gcs*)). (*propa*, *confl*, *dec*, *res*, *lres*, *uset* + *1*, *gcs*))›

definition *incr-GC* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-GC* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbds*). (*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs* + *1*, *lbds*))›

definition *add-lbd* :: ‹*uint64* ⇒ *stats* ⇒ *stats*› **where**
‹*add-lbd lbd* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbds*). (*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbd* + *lbds*))›

Moving averages We use (at least hopefully) the variant of EMA-14 implemented in Cadical, but with fixed-point calculation (*1* is *1* >> *32*).

Remark that the coefficient β already should not take care of the fixed-point conversion of the glue. Otherwise, *value* is wrongly updated.

type-synonym *ema* = ‹*uint64* × *uint64* × *uint64* × *uint64* × *uint64*›

definition *ema-bitshifting* **where**
‹*ema-bitshifting* = (*1* << *32*)›

definition (**in** −) *ema-update* :: ‹*nat* ⇒ *ema* ⇒ *ema*› **where**
‹*ema-update* = (λ*lbd* (*value*, α, β, *wait*, *period*).
  let lbd = (*uint64-of-nat lbd*) ∗ *ema-bitshifting in*
  let value = *if lbd* > *value then value* + (β ∗ (*lbd* − *value*) >> *32*) *else value* − (β ∗ (*value* − *lbd*) >> *32*) *in*
  if β ≤ α ∨ *wait* > *0 then* (*value*, α, β, *wait* − *1*, *period*)
  else
    let wait = *2* ∗ *period* + *1 in*
    let period = *wait in*
    let β = β >> *1 in*
    let β = *if* β ≤ α *then* α *else* β *in*
    (*value*, α, β, *wait*, *period*))›

definition (**in** −) *ema-update-ref* :: ‹*uint32* ⇒ *ema* ⇒ *ema*› **where**
‹*ema-update-ref* = (λ*lbd* (*value*, α, β, *wait*, *period*).
  let lbd = (*uint64-of-uint32 lbd*) ∗ *ema-bitshifting in*
  let value = *if lbd* > *value then value* + (β ∗ (*lbd* − *value*) >> *32*) *else value* − (β ∗ (*value* − *lbd*) >> *32*) *in*
  if β ≤ α ∨ *wait* > *0 then* (*value*, α, β, *wait* − *1*, *period*)
  else
    let wait = *2* ∗ *period* + *1 in*
    let period = *wait in*
    let β = β >> *1 in*
    let β = *if* β ≤ α *then* α *else* β *in*
    (*value*, α, β, *wait*, *period*))›

definition (**in** −) *ema-init* :: ‹*uint64* ⇒ *ema*› **where**
‹*ema-init* α = (*0*, α, *ema-bitshifting*, *0*, *0*)›

fun *ema-reinit* **where**
‹*ema-reinit* (*value*, α, β, *wait*, *period*) = (*value*, α, *1* << *32*, *0*, *0*)›

fun *ema-get-value* :: ‹*ema* ⇒ *uint64*› **where**
‹*ema-get-value* (*v*, -) = *v*›

We use the default values for Cadical: $(3::'a) / (10::'a)^2$ and $(1::'a) / (10::'a)^5$ in our fixed-point version.

abbreviation *ema-fast-init* :: *ema* **where**
‹*ema-fast-init* ≡ *ema-init* (*128849010*)›

abbreviation *ema-slow-init* :: *ema* **where**
‹*ema-slow-init* ≡ *ema-init 429450*›

Information related to restarts **type-synonym** *restart-info* = ‹*uint64* × *uint64*›

definition *incr-conflict-count-since-last-restart* :: ‹*restart-info* ⇒ *restart-info*› **where**
‹*incr-conflict-count-since-last-restart* = (λ(*ccount*, *ema-lvl*). (*ccount* + *1*, *ema-lvl*))›

definition *restart-info-update-lvl-avg* :: ‹*uint32* ⇒ *restart-info* ⇒ *restart-info*› **where**
‹*restart-info-update-lvl-avg* = (λ*lvl* (*ccount*, *ema-lvl*). (*ccount*, *ema-lvl*))›

definition *restart-info-init* :: ‹*restart-info*› **where**
‹*restart-info-init* = (*0*, *0*)›

definition *restart-info-restart-done* :: ‹*restart-info* ⇒ *restart-info*› **where**
‹*restart-info-restart-done* = (λ(*ccount*, *lvl-avg*). (*0*, *lvl-avg*))›

VMTF

type-synonym *vmtf-assn* = ‹(*uint32*, *uint64*) *vmtf-node array* × *uint64* × *uint32* × *uint32* × *uint32 option*›

type-synonym *phase-saver-assn* = ‹*bool array*›

instance *vmtf-node* :: (*heap*, *heap*) *heap*
⟨*proof*⟩

definition (**in** −) *vmtf-node-rel* **where**
‹*vmtf-node-rel* = {(*a*′, *a*). (*stamp a*′, *stamp a*) ∈ *uint64-nat-rel* ∧
(*get-prev a*′, *get-prev a*) ∈ ⟨*uint32-nat-rel*⟩*option-rel* ∧
(*get-next a*′, *get-next a*) ∈ ⟨*uint32-nat-rel*⟩*option-rel*}›

type-synonym (**in** −) *isa-vmtf-remove-int* = ‹*vmtf* × (*nat list* × *bool list*)›

Options

type-synonym *opts* = ‹*bool* × *bool* × *bool*›

definition *opts-restart* **where**
‹*opts-restart* = (λ(*a*, *b*). *a*)›

definition *opts-reduce* **where**
‹*opts-reduce* = (λ(*a*, *b*, *c*). *b*)›

definition *opts-unbounded-mode* **where**
‹*opts-unbounded-mode* = (λ(*a*, *b*, *c*). *c*)›

Base state

type-synonym *out-learned* = ‹*nat clause-l*›

type-synonym *vdom* = ‹*nat list*›

heur stands for heuristic.

type-synonym *twl-st-wl-heur* =
‹*trail-pol* × *arena* ×
conflict-option-rel × *nat* × (*nat watcher*) *list list* × *isa-vmtf-remove-int* × *bool list* ×
nat × *conflict-min-cach-l* × *lbd* × *out-learned* × *stats* × *ema* × *ema* × *restart-info* ×
vdom × *vdom* × *nat* × *opts* × *arena*›

fun *get-clauses-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *arena*› **where**
‹*get-clauses-wl-heur* (*M*, *N*, *D*, -) = *N*›

fun *get-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *trail-pol*› **where**

‹*get-trail-wl-heur* (*M*, *N*, *D*, -) = *M*›

fun *get-conflict-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *conflict-option-rel*› **where**
‹*get-conflict-wl-heur* (-, -, *D*, -) = *D*›

fun *watched-by-int* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat watched*› **where**
‹*watched-by-int* (*M*, *N*, *D*, *Q*, *W*, -) *L* = *W* ! *nat-of-lit L*›

fun *get-watched-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*nat watcher*) *list list*› **where**
‹*get-watched-wl-heur* (-, -, -, -, *W*, -) = *W*›

fun *literals-to-update-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*literals-to-update-wl-heur* (*M*, *N*, *D*, *Q*, *W*, -, -) = *Q*›

fun *set-literals-to-update-wl-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*set-literals-to-update-wl-heur i* (*M*, *N*, *D*, -, *W'*) = (*M*, *N*, *D*, *i*, *W'*)›

definition *watched-by-app-heur-pre* **where**
‹*watched-by-app-heur-pre* = (λ((*S*, *L*), *K*). *nat-of-lit L* < *length* (*get-watched-wl-heur S*) ∧
K < *length* (*watched-by-int S L*))›

definition (**in** −) *watched-by-app-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app-heur S L K* = *watched-by-int S L* ! *K*›

lemma *watched-by-app-heur-alt-def*:
‹*watched-by-app-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L K*. *W* ! *nat-of-lit L* ! *K*)›
⟨*proof*⟩

definition *watched-by-app* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app S L K* = *watched-by S L* ! *K*›

fun *get-vmtf-heur* :: ‹*twl-st-wl-heur* ⇒ *isa-vmtf-remove-int*› **where**
‹*get-vmtf-heur* (-, -, -, -, -, *vm*, -) = *vm*›

fun *get-phase-saver-heur* :: ‹*twl-st-wl-heur* ⇒ *bool list*› **where**
‹*get-phase-saver-heur* (-, -, -, -, -, -, *φ*, -) = *φ*›

fun *get-count-max-lvls-heur* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*get-count-max-lvls-heur* (-, -, -, -, -, -, -, *clvls*, -) = *clvls*›

fun *get-conflict-cach*:: ‹*twl-st-wl-heur* ⇒ *conflict-min-cach-l*› **where**
‹*get-conflict-cach* (-, -, -, -, -, -, -, -, *cach*, -) = *cach*›

fun *get-lbd* :: ‹*twl-st-wl-heur* ⇒ *lbd*› **where**
‹*get-lbd* (-, -, -, -, -, -, -, -, -, *lbd*, -) = *lbd*›

fun *get-outlearned-heur* :: ‹*twl-st-wl-heur* ⇒ *out-learned*› **where**
‹*get-outlearned-heur* (-, -, -, -, -, -, -, -, -, -, *out*, -) = *out*›

fun *get-fast-ema-heur* :: ‹*twl-st-wl-heur* ⇒ *ema*› **where**
‹*get-fast-ema-heur* (-, -, -, -, -, -, -, -, -, -, -, -, *fast-ema*, -) = *fast-ema*›

fun *get-slow-ema-heur* :: ‹*twl-st-wl-heur* ⇒ *ema*› **where**
‹*get-slow-ema-heur* (-, -, -, -, -, -, -, -, -, -, -, -, -, *slow-ema*, -) = *slow-ema*›

fun *get-conflict-count-heur* :: ‹*twl-st-wl-heur* ⇒ *restart-info*› **where**

‹*get-conflict-count-heur* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, *ccount*, -) = *ccount*›

fun *get-vdom* :: ‹*twl-st-wl-heur* ⇒ *nat list*› **where**
‹*get-vdom* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*›

fun *get-avdom* :: ‹*twl-st-wl-heur* ⇒ *nat list*› **where**
‹*get-avdom* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*›

fun *get-learned-count* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*get-learned-count* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *lcount*, -) = *lcount*›

fun *get-ops* :: ‹*twl-st-wl-heur* ⇒ *opts*› **where**
‹*get-ops* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *opts*, -) = *opts*›

fun *get-old-arena* :: ‹*twl-st-wl-heur* ⇒ *arena*› **where**
‹*get-old-arena* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *old-arena*) = *old-arena*›

Setup to convert a list from *uint64* to *nat*.

definition *arl-copy-to* :: ‹(′*a* ⇒ ′*b*) ⇒ ′*a list* ⇒ ′*b list*› **where**
‹*arl-copy-to R xs* = *map R xs*›

definition *op-map-to*
:: ‹(′*b* ⇒ ′*a*) ⇒ ′*a* ⇒ ′*b list* ⇒ ′*a list list* ⇒ *nat* ⇒ ′*a list list nres*›
where
‹*op-map-to R e xs W j* = *do* {
(-, *zs*) ←
$WHILE_T{}^{\lambda(i,W').\ i \leq length\ xs\ \wedge\ W'!j\ =\ W!j\ @\ map\ R\ (take\ i\ xs)\ \wedge\ \ \ \ \ \ \ (\forall k.\ k \neq j \longrightarrow k < length\ W \longrightarrow W'!k = W}$
(λ(*i*, *W*′). *i* < *length xs*)
(λ(*i*, *W*′). *do* {
ASSERT(*i* < *length xs*);
let x = *xs* ! *i*;
RETURN (*i*+*1*, *append-ll W*′ *j* (*R x*))})
(*0*, *W*);
RETURN zs
}›

lemma *op-map-to-map*:
‹*j* < *length W*′ ⟹ *op-map-to R e xs W*′ *j* ≤ *RETURN* (*W*′[*j* := *W*′!*j* @ *map R xs*])›
⟨*proof*⟩

lemma *op-map-to-map-rel*:
‹(*uncurry2* (*op-map-to R e*), *uncurry2* (*RETURN ooo* (λ*xs W*′ *j*. *W*′[*j* := *W*′!*j* @ *map R xs*]))) ∈
$[\lambda((xs,\ ys),\ j).\ j < length\ ys]_f$
⟨*Id*⟩*list-rel* $\times_f$
⟨⟨*Id*⟩*list-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨⟨⟨*Id*⟩*list-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *convert-single-wl-to-nat* **where**
‹*convert-single-wl-to-nat W i W*′ *j* =
op-map-to (λ(*i*, *C*). (*nat-of-uint64-conv i*, *C*)) (*to-watcher 0* (*Pos 0*) *False*) (*W*!*i*) *W*′ *j*›

definition *convert-single-wl-to-nat-conv* **where**
‹*convert-single-wl-to-nat-conv xs i W*′ *j* =
W′[*j* := *map* (λ(*i*, *C*). (*nat-of-uint64-conv i*, *C*)) (*xs*!*i*)]›

lemma *convert-single-wl-to-nat*:
‹$(uncurry3\ convert\text{-}single\text{-}wl\text{-}to\text{-}nat,$
$uncurry3\ (RETURN\ oooo\ convert\text{-}single\text{-}wl\text{-}to\text{-}nat\text{-}conv)) \in$
$[\lambda(((xs, i), ys), j).\ i < length\ xs \wedge j < length\ ys \wedge ys!j = []]_f$
$\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \times_f$
$\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \to$
$\langle\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

The virtual domain is composed of the addressable (and accessible) elements, i.e., the domain and all the deleted clauses that are still present in the watch lists.

definition *vdom-m* :: ‹$nat\ multiset \Rightarrow (nat\ literal \Rightarrow (nat \times \text{-})\ list) \Rightarrow (nat, 'b)\ fmap \Rightarrow nat\ set$› **where**
‹$vdom\text{-}m\ \mathcal{A}\ W\ N = \bigcup(((`)\ fst)\ `\ set\ `\ W\ `\ set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A})) \cup set\text{-}mset\ (dom\text{-}m\ N)$›

lemma *vdom-m-simps*[*simp*]:
‹$bh \in\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (N(bh \hookrightarrow C)) = vdom\text{-}m\ \mathcal{A}\ W\ N$›
‹$bh \notin\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (N(bh \hookrightarrow C)) = insert\ bh\ (vdom\text{-}m\ \mathcal{A}\ W\ N)$›
⟨*proof*⟩

lemma *vdom-m-simps2*[*simp*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (W(L := W\ L\ @\ [(i, C)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
‹$bi \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L{:=}\ bp\ L\ @\ [(bi, av')]))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
⟨*proof*⟩

lemma *vdom-m-simps3*[*simp*]:
‹$fst\ biav' \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L{:=}\ bp\ L\ @\ [biav']))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
⟨*proof*⟩

What is the difference with the next lemma?

lemma [*simp*]:
‹$bf \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ bj\ (ax(bf \hookrightarrow C')) = vdom\text{-}m\ \mathcal{A}\ bj\ (ax)$›
⟨*proof*⟩

lemma *vdom-m-simps4*[*simp*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1 := W\ L1\ @\ [(i, C1)],\ L2 := W\ L2\ @\ [(i, C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
⟨*proof*⟩

This is $?i \in\#\ dom\text{-}m\ ?N \Longrightarrow vdom\text{-}m\ ?\mathcal{A}\ (?W(?L1.0 := ?W\ ?L1.0\ @\ [(?i, ?C1.0)],\ ?L2.0 := ?W\ ?L2.0\ @\ [(?i, ?C2.0)]))\ ?N = vdom\text{-}m\ ?\mathcal{A}\ ?W\ ?N$ if the assumption of distinctness is not present in the context.

lemma *vdom-m-simps4'*[*simp*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1 := W\ L1\ @\ [(i, C1), (i, C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
⟨*proof*⟩

We add a spurious dependency to the parameter of the locale:

definition *empty-watched* :: ‹$nat\ multiset \Rightarrow nat\ literal \Rightarrow (nat \times nat\ literal \times bool)\ list$› **where**
‹$empty\text{-}watched\ \mathcal{A} = (\lambda\text{-}.\ [])$›

lemma *vdom-m-empty-watched*[*simp*]:
‹$vdom\text{-}m\ \mathcal{A}\ (empty\text{-}watched\ \mathcal{A}')\ N = set\text{-}mset\ (dom\text{-}m\ N)$›
⟨*proof*⟩

The following rule makes the previous not applicable. Therefore, we do not mark this lemma as simp.

lemma *vdom-m-simps5*:
‹$i \notin\# \mathit{dom\text{-}m}\ N \Longrightarrow \mathit{vdom\text{-}m}\ \mathcal{A}\ W\ (\mathit{fmupd}\ i\ C\ N) = \mathit{insert}\ i\ (\mathit{vdom\text{-}m}\ \mathcal{A}\ W\ N)$›
⟨*proof*⟩

lemma *in-watch-list-in-vdom*:
assumes ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$w < \mathit{length}\ (\mathit{watched\text{-}by}\ S\ L)$›
shows ‹$\mathit{fst}\ (\mathit{watched\text{-}by}\ S\ L\ !\ w) \in \mathit{vdom\text{-}m}\ \mathcal{A}\ (\mathit{get\text{-}watched\text{-}wl}\ S)\ (\mathit{get\text{-}clauses\text{-}wl}\ S)$›
⟨*proof*⟩

lemma *in-watch-list-in-vdom'*:
assumes ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$A \in \mathit{set}\ (\mathit{watched\text{-}by}\ S\ L)$›
shows ‹$\mathit{fst}\ A \in \mathit{vdom\text{-}m}\ \mathcal{A}\ (\mathit{get\text{-}watched\text{-}wl}\ S)\ (\mathit{get\text{-}clauses\text{-}wl}\ S)$›
⟨*proof*⟩

lemma *in-dom-in-vdom*[*simp*]:
‹$x \in\# \mathit{dom\text{-}m}\ N \Longrightarrow x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ W\ N$›
⟨*proof*⟩

lemma *in-vdom-m-upd*:
‹$x1f \in \mathit{vdom\text{-}m}\ \mathcal{A}\ (g(x1e := (g\ x1e)[x2 := (x1f,\ x2f)]))\ b$›
if ‹$x2 < \mathit{length}\ (g\ x1e)$› **and** ‹$x1e \in\# \mathcal{L}_{all}\ \mathcal{A}$›
⟨*proof*⟩

lemma *in-vdom-m-fmdropD*:
‹$x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ ga\ (\mathit{fmdrop}\ C\ baa) \Longrightarrow x \in (\mathit{vdom\text{-}m}\ \mathcal{A}\ ga\ baa)$›
⟨*proof*⟩

definition *cach-refinement-empty* **where**
‹$\mathit{cach\text{-}refinement\text{-}empty}\ \mathcal{A}\ \mathit{cach} \longleftrightarrow$
$(\mathit{cach},\ \lambda\text{-}.\ \mathit{SEEN\text{-}UNKNOWN}) \in \mathit{cach\text{-}refinement}\ \mathcal{A}$›

definition *isa-vmtf* **where**
‹$\mathit{isa\text{-}vmtf}\ \mathcal{A}\ M =$
$((\mathit{Id} \times_r \mathit{nat\text{-}rel} \times_r \mathit{nat\text{-}rel} \times_r \mathit{nat\text{-}rel} \times_r \langle \mathit{nat\text{-}rel}\rangle \mathit{option\text{-}rel}) \times_f \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A})^{-1}$
$``\ \mathit{vmtf}\ \mathcal{A}\ M$›

lemma *isa-vmtfI*:
‹$(\mathit{vm},\ \mathit{to\text{-}remove}') \in \mathit{vmtf}\ \mathcal{A}\ M \Longrightarrow (\mathit{to\text{-}remove},\ \mathit{to\text{-}remove}') \in \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A} \Longrightarrow$
$(\mathit{vm},\ \mathit{to\text{-}remove}) \in \mathit{isa\text{-}vmtf}\ \mathcal{A}\ M$›
⟨*proof*⟩

lemma *isa-vmtf-consD*:
‹$((\mathit{ns},\ m,\ \mathit{fst\text{-}As},\ \mathit{lst\text{-}As},\ \mathit{next\text{-}search}),\ \mathit{remove}) \in \mathit{isa\text{-}vmtf}\ \mathcal{A}\ M \Longrightarrow$
$((\mathit{ns},\ m,\ \mathit{fst\text{-}As},\ \mathit{lst\text{-}As},\ \mathit{next\text{-}search}),\ \mathit{remove}) \in \mathit{isa\text{-}vmtf}\ \mathcal{A}\ (L\ \#\ M)$›
⟨*proof*⟩

lemma *isa-vmtf-consD2*:
‹$f \in \mathit{isa\text{-}vmtf}\ \mathcal{A}\ M \Longrightarrow$
$f \in \mathit{isa\text{-}vmtf}\ \mathcal{A}\ (L\ \#\ M)$›
⟨*proof*⟩

vdom is an upper bound on all the address of the clauses that are used in the state. *avdom* includes the active clauses.

definition *twl-st-heur* :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur* =
  {((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
      vdom, *avdom*, *lcount*, *opts*, *old-arena*),
    (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)).
  (*M'*, *M*) ∈ *trail-pol* (*all-atms N* (*NE* + *UE*)) ∧
  valid-arena N' N (*set vdom*) ∧
  (*D'*, *D*) ∈ *option-lookup-clause-rel* (*all-atms N* (*NE* + *UE*)) ∧
  (*D* = *None* ⟶ *j* ≤ *length M*) ∧
  Q = *uminus* '# *lit-of* '# *mset* (*drop j* (*rev M*)) ∧
  (*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE*))) ∧
  vm ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) *M* ∧
  phase-saving (*all-atms N* (*NE* + *UE*)) *φ* ∧
  no-dup M ∧
  clvls ∈ *counts-maximum-level M D* ∧
  cach-refinement-empty (*all-atms N* (*NE* + *UE*)) *cach* ∧
  out-learned M D outl ∧
  lcount = *size* (*learned-clss-lf N*) ∧
  vdom-m (*all-atms N* (*NE* + *UE*)) *W N* ⊆ *set vdom* ∧
  mset avdom ⊆# *mset vdom* ∧
  distinct vdom ∧
  isasat-input-bounded (*all-atms N* (*NE* + *UE*)) ∧
  isasat-input-nempty (*all-atms N* (*NE* + *UE*)) ∧
  old-arena = []
}›

lemma *twl-st-heur-state-simp*:
  assumes ‹(*S*, *S'*) ∈ *twl-st-heur*›
  shows
    ‹(*get-trail-wl-heur S*, *get-trail-wl S'*) ∈ *trail-pol* (*all-atms-st S'*)› **and**
    twl-st-heur-state-simp-watched: ‹*C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S'*) ⟹
      watched-by-int S C = *watched-by S' C*› **and**
    ‹*literals-to-update-wl S'* =
        uminus '# *lit-of* '# *mset* (*drop* (*literals-to-update-wl-heur S*) (*rev* (*get-trail-wl S'*)))›
  ⟨*proof*⟩

abbreviation *twl-st-heur'''*
  :: ‹*nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur''' r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
        length (*get-clauses-wl-heur S*) = *r*}›

definition *twl-st-heur'* :: ‹*nat multiset* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur' N* = {(*S*, *S'*). (*S*, *S'*) ∈ *twl-st-heur* ∧ *dom-m* (*get-clauses-wl S'*) = *N*}›

definition *twl-st-heur-conflict-ana*
  :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur-conflict-ana* =
  {((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*, *vdom*,
      avdom, *lcount*, *opts*, *old-arena*),
    (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)).
  (*M'*, *M*) ∈ *trail-pol* (*all-atms N* (*NE* + *UE*)) ∧
  valid-arena N' N (*set vdom*) ∧

$(D', D) \in$ *option-lookup-clause-rel* $($*all-atms* N $(NE + UE)) \wedge$
$(W', W) \in \langle Id \rangle$*map-fun-rel* $(D_0$ $($*all-atms* N $(NE + UE))) \wedge$
vm $\in$ *isa-vmtf* $($*all-atms* N $(NE + UE))$ $M \wedge$
phase-saving $($*all-atms* N $(NE + UE))$ $\varphi \wedge$
no-dup $M \wedge$
clvls $\in$ *counts-maximum-level* M $D \wedge$
cach-refinement-empty $($*all-atms* N $(NE + UE))$ *cach* $\wedge$
out-learned M D *outl* $\wedge$
lcount $=$ *size* $($*learned-clss-lf* $N) \wedge$
vdom-m $($*all-atms* N $(NE + UE))$ W $N \subseteq$ *set vdom* $\wedge$
mset avdom $\subseteq\#$ *mset vdom* $\wedge$
distinct vdom $\wedge$
isasat-input-bounded $($*all-atms* N $(NE + UE)) \wedge$
isasat-input-nempty $($*all-atms* N $(NE + UE)) \wedge$
old-arena $= []$
}›

lemma *twl-st-heur-twl-st-heur-conflict-ana*:
‹$(S, T) \in$ *twl-st-heur* $\Longrightarrow (S, T) \in$ *twl-st-heur-conflict-ana*›
⟨*proof*⟩

lemma *twl-st-heur-ana-state-simp*:
assumes ‹$(S, S') \in$ *twl-st-heur-conflict-ana*›
shows
‹$($*get-trail-wl-heur* S, *get-trail-wl* $S') \in$ *trail-pol* $($*all-atms-st* $S')$› **and**
‹$C \in\# \mathcal{L}_{all}$ $($*all-atms-st* $S') \Longrightarrow$ *watched-by-int* S $C =$ *watched-by* S' C›
⟨*proof*⟩

This relations decouples the conflict that has been minimised and appears abstractly from the refined state, where the conflict has been removed from the data structure to a separate array.

definition *twl-st-heur-bt* :: ‹$($*twl-st-wl-heur* $\times$ *nat twl-st-wl*$)$ *set*› **where**
‹*twl-st-heur-bt* $=$
$\{((M', N', D', Q', W', vm, \varphi, clvls, cach, lbd, outl, stats, -, -, -, vdom, avdom, lcount, opts,$
old-arena$),$
$(M, N, D, NE, UE, Q, W)).$
$(M', M) \in$ *trail-pol* $($*all-atms* N $(NE + UE)) \wedge$
valid-arena N' N $($*set vdom*$) \wedge$
$(D', None) \in$ *option-lookup-clause-rel* $($*all-atms* N $(NE + UE)) \wedge$
$(W', W) \in \langle Id \rangle$*map-fun-rel* $(D_0$ $($*all-atms* N $(NE + UE))) \wedge$
vm $\in$ *isa-vmtf* $($*all-atms* N $(NE + UE))$ $M \wedge$
phase-saving $($*all-atms* N $(NE + UE))$ $\varphi \wedge$
no-dup $M \wedge$
clvls $\in$ *counts-maximum-level* M *None* $\wedge$
cach-refinement-empty $($*all-atms* N $(NE + UE))$ *cach* $\wedge$
out-learned M *None outl* $\wedge$
lcount $=$ *size* $($*learned-clss-l* $N) \wedge$
vdom-m $($*all-atms* N $(NE + UE))$ W $N \subseteq$ *set vdom* $\wedge$
mset avdom $\subseteq\#$ *mset vdom* $\wedge$
distinct vdom $\wedge$
isasat-input-bounded $($*all-atms* N $(NE + UE)) \wedge$
isasat-input-nempty $($*all-atms* N $(NE + UE)) \wedge$
old-arena $= []$
}›

The difference between *isasat-unbounded-assn* and *isasat-bounded-assn* corresponds to the following condition:

definition *isasat-fast* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*isasat-fast S* ⟷ (*length* (*get-clauses-wl-heur S*) ≤ *uint64-max* − (*uint32-max div 2* + *6*))›

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* ⟹ *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*›
⟨*proof*⟩

Lift Operations to State

definition *polarity-st* :: ‹*'v twl-st-wl* ⇒ *'v literal* ⇒ *bool option*› **where**
‹*polarity-st S* = *polarity* (*get-trail-wl S*)›

definition *get-conflict-wl-is-None-heur* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-heur* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *b*)›

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-conflict-wl-is-None-heur-alt-def*:
‹*RETURN o get-conflict-wl-is-None-heur* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)›
⟨*proof*⟩

definition *count-decided-st* :: ‹*nat twl-st-wl* ⇒ *nat*› **where**
‹*count-decided-st* = (λ(*M*, -). *count-decided M*)›

definition *isa-count-decided-st* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*isa-count-decided-st* = (λ(*M*, -). *count-decided-pol M*)›

lemma *count-decided-st-count-decided-st*:
‹(*RETURN o isa-count-decided-st*, *RETURN o count-decided-st*) ∈ *twl-st-heur* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *count-decided-st-alt-def*: ‹*count-decided-st S* = *count-decided* (*get-trail-wl S*)›
⟨*proof*⟩

definition (**in** −) *is-in-conflict-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*› **where**
‹*is-in-conflict-st L S* ⟷ *is-in-conflict L* (*get-conflict-wl S*)›

definition *atm-is-in-conflict-st-heur* :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *bool*› **where**
‹*atm-is-in-conflict-st-heur L* = (λ(*M*, *N*, (-, *D*), -). *atm-in-conflict-lookup* (*atm-of L*) *D*)›

lemma *atm-is-in-conflict-st-heur-alt-def*:
‹*RETURN oo atm-is-in-conflict-st-heur* = (λ*L* (*M*, *N*, (-, (-, *D*)), -). *RETURN* (*D* ! (*atm-of L*) ≠ *None*))›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*), *uncurry* (*RETURN oo is-in-conflict-st*)) ∈
[λ(*L*, *S*). −*L* ∉# *the* (*get-conflict-wl S*) ∧ *get-conflict-wl S* ≠ *None* ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)]$_f$
Id $\times_r$ *twl-st-heur* → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st-ana*:
‹(*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*), *uncurry* (*RETURN oo is-in-conflict-st*)) ∈
$[\lambda(L, S). -L \notin\# \ the\ (get\text{-}conflict\text{-}wl\ S) \wedge get\text{-}conflict\text{-}wl\ S \neq None \wedge$
$L \in\# \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S)]_f$
$Id \times_r$ *twl-st-heur-conflict-ana* → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

definition *polarity-st-heur*
:: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *bool option*›
where
‹*polarity-st-heur S* =
polarity-pol (*get-trail-wl-heur S*)›

definition *polarity-st-pre* **where**
‹*polarity-st-pre* ≡ λ(*S*, *L*). $L \in\# \mathcal{L}_{all}$ (*all-atms-st S*)›

lemma *polarity-st-heur-alt-def*:
‹*polarity-st-heur* = (λ(*M*, -). *polarity-pol M*)›
⟨*proof*⟩

definition *polarity-st-heur-pre* **where**
‹*polarity-st-heur-pre* ≡ λ(*S*, *L*). *polarity-pol-pre* (*get-trail-wl-heur S*) *L*›

lemma *polarity-st-heur-pre*:
‹(S', *S*) ∈ *twl-st-heur* ⟹ $L \in\# \mathcal{L}_{all}$ (*all-atms-st S*) ⟹ *polarity-st-heur-pre* (S', *L*)›
⟨*proof*⟩

abbreviation *nat-lit-lit-rel* **where**
‹*nat-lit-lit-rel* ≡ *Id* :: (*nat literal* × -) *set*›

0.1.10 More theorems

lemma *valid-arena-DECISION-REASON*:
‹*valid-arena arena NU vdom* ⟹ *DECISION-REASON* ∉# *dom-m NU*›
⟨*proof*⟩

definition *count-decided-st-heur* :: ‹- ⇒ -› **where**
‹*count-decided-st-heur* = (λ((-,-,-,-,*n*, -), -). *n*)›

lemma *twl-st-heur-count-decided-st-alt-def*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
⟨*proof*⟩

lemma *twl-st-heur-isa-length-trail-get-trail-wl*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *isa-length-trail* (*get-trail-wl-heur S*) = *length* (*get-trail-wl T*)›
⟨*proof*⟩

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈ *trail-pol* $\mathcal{B}$›
⟨*proof*⟩

lemma *distinct-atoms-rel-cong*:

‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow L \in distinct\text{-}atoms\text{-}rel\ \mathcal{A} \Longrightarrow L \in distinct\text{-}atoms\text{-}rel\ \mathcal{B}$›
⟨*proof*⟩

lemma *vmtf-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow L \in vmtf\ \mathcal{A}\ M \Longrightarrow L \in vmtf\ \mathcal{B}\ M$›
⟨*proof*⟩

lemma *isa-vmtf-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow L \in isa\text{-}vmtf\ \mathcal{A}\ M \Longrightarrow L \in isa\text{-}vmtf\ \mathcal{B}\ M$›
⟨*proof*⟩

lemma *option-lookup-clause-rel-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow L \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \Longrightarrow L \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{B}$›
⟨*proof*⟩

lemma D_0*-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow D_0\ \mathcal{A} = D_0\ \mathcal{B}$›
⟨*proof*⟩

lemma *phase-saving-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow phase\text{-}saving\ \mathcal{A} = phase\text{-}saving\ \mathcal{B}$›
⟨*proof*⟩

lemma *distinct-subseteq-iff2*:
assumes *dist*: *distinct-mset M*
shows $set\text{-}mset\ M \subseteq set\text{-}mset\ N \longleftrightarrow M \subseteq\#\ N$
⟨*proof*⟩

lemma *cach-refinement-empty-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow cach\text{-}refinement\text{-}empty\ \mathcal{A} = cach\text{-}refinement\text{-}empty\ \mathcal{B}$›
⟨*proof*⟩

lemma *vdom-m-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow vdom\text{-}m\ \mathcal{A}\ x\ y = vdom\text{-}m\ \mathcal{B}\ x\ y$›
⟨*proof*⟩

lemma *isasat-input-bounded-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow isasat\text{-}input\text{-}bounded\ \mathcal{A} = isasat\text{-}input\text{-}bounded\ \mathcal{B}$›
⟨*proof*⟩

lemma *isasat-input-nempty-cong*:
‹$set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{B} \Longrightarrow isasat\text{-}input\text{-}nempty\ \mathcal{A} = isasat\text{-}input\text{-}nempty\ \mathcal{B}$›
⟨*proof*⟩

0.1.11 Shared Code Equations

definition *clause-not-marked-to-delete* **where**
‹$clause\text{-}not\text{-}marked\text{-}to\text{-}delete\ S\ C \longleftrightarrow C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S)$›

definition *clause-not-marked-to-delete-pre* **where**
‹$clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}pre =$
$(\lambda(S,\ C).\ C \in vdom\text{-}m\ (all\text{-}atms\text{-}st\ S)\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S))$›

definition *clause-not-marked-to-delete-heur-pre* **where**
‹*clause-not-marked-to-delete-heur-pre* =
  (λ(*S*, *C*). *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*)›

definition *clause-not-marked-to-delete-heur* :: ‹- ⇒ *nat* ⇒ *bool*›
where
‹*clause-not-marked-to-delete-heur S C* ⟷
  arena-status (*get-clauses-wl-heur S*) *C* ≠ *DELETED*›

lemma *clause-not-marked-to-delete-rel*:
‹(*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*),
  uncurry (*RETURN oo clause-not-marked-to-delete*)) ∈
  [*clause-not-marked-to-delete-pre*]$_f$
  twl-st-heur $\times_f$ *nat-rel* → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *access-lit-in-clauses-heur-pre* **where**
‹*access-lit-in-clauses-heur-pre* =
  (λ((*S*, *i*), *j*).
    arena-lit-pre (*get-clauses-wl-heur S*) (*i*+*j*))›

definition (**in** −) *access-lit-in-clauses-heur* **where**
‹*access-lit-in-clauses-heur S i j* = *arena-lit* (*get-clauses-wl-heur S*) (*i* + *j*)›

lemma *access-lit-in-clauses-heur-alt-def*:
‹*access-lit-in-clauses-heur* = (λ(*M*, *N*, -) *i j*. *arena-lit N* (*i* + *j*))›
⟨*proof*⟩

lemma *access-lit-in-clauses-heur-fast-pre*:
‹*arena-lit-pre* (*get-clauses-wl-heur a*) (*ba* + *b*) ⟹
  isasat-fast a ⟹ *ba* + *b* ≤ *uint64-max*›
⟨*proof*⟩

lemma *eq-insertD*: ‹*A* = *insert a B* ⟹ *a* ∈ *A* ∧ *B* ⊆ *A*›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$*-add-mset*:
‹*set-mset* ($\mathcal{L}_{all}$ (*add-mset L C*)) = *insert* (*Pos L*) (*insert* (*Neg L*) (*set-mset* ($\mathcal{L}_{all}$ *C*)))›
⟨*proof*⟩

lemma *correct-watching-dom-watched*:
  assumes ‹*correct-watching S*› **and** ‹⋀*C*. *C* ∈# *ran-mf* (*get-clauses-wl S*) ⟹ *C* ≠ []›
  shows ‹*set-mset* (*dom-m* (*get-clauses-wl S*)) ⊆
    ⋃(((‘) *fst*) ‘ *set* ‘ (*get-watched-wl S*) ‘ *set-mset* ($\mathcal{L}_{all}$ (*all-atms-st S*)))›
    (**is** ‹*?A* ⊆ *?B*›)
⟨*proof*⟩

0.1.12 Rewatch

0.1.13 Rewatch

definition *rewatch-heur* **where**
‹*rewatch-heur vdom arena W* = *do* {
  let - = *vdom*;

nfoldli [*0*..<*length vdom*] (λ-. *True*)
(λi *W*. *do* {
ASSERT(i < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C $\neq$ *DELETED*
then do {
ASSERT(*arena-lit-pre arena C*);
ASSERT(*arena-lit-pre arena* (*C+1*));
let L1 = *arena-lit arena C*;
let L2 = *arena-lit arena* (*C* + *1*);
ASSERT(*nat-of-lit L1* < *length W*);
ASSERT(*arena-is-valid-clause-idx arena C*);
let b = (*arena-length arena C* = *2*);
ASSERT(*L1* $\neq$ *L2*);
ASSERT(*length* (*W* ! (*nat-of-lit L1*)) < *length arena*);
let W = *append-ll W* (*nat-of-lit L1*) (*to-watcher C L2 b*);
ASSERT(*nat-of-lit L2* < *length W*);
ASSERT(*length* (*W* ! (*nat-of-lit L2*)) < *length arena*);
let W = *append-ll W* (*nat-of-lit L2*) (*to-watcher C L1 b*);
RETURN W
}
else RETURN W
})
W
}›

lemma *rewatch-heur-rewatch*:
assumes
‹*valid-arena arena N vdom*› **and** ‹*set xs* $\subseteq$ *vdom*› **and** ‹*distinct xs*› **and** ‹*set-mset* (*dom-m N*) $\subseteq$ *set xs*› **and**
‹(W, W') $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and** *lall*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
‹*vdom-m* $\mathcal{A}$ W' *N* $\subseteq$ *set-mset* (*dom-m N*)›
shows
‹*rewatch-heur xs arena W* $\leq \Downarrow$ ({(W, W'). (W, W') $\in$⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) $\wedge$ *vdom-m* $\mathcal{A}$ W' *N* $\subseteq$ *set-mset* (*dom-m N*)}) (*rewatch N* W')›
⟨*proof*⟩

lemma *rewatch-heur-alt-def*:
‹*rewatch-heur vdom arena W* = *do* {
let - = *vdom*;
nfoldli [*0*..<*length vdom*] (λ-. *True*)
(λi *W*. *do* {
ASSERT(i < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C $\neq$ *DELETED*
then do {
let C = *uint64-of-nat-conv C*;
ASSERT(*arena-lit-pre arena C*);
ASSERT(*arena-lit-pre arena* (*C+1*));
let L1 = *arena-lit arena C*;
let L2 = *arena-lit arena* (*C* + *1*);
ASSERT(*nat-of-lit L1* < *length W*);
ASSERT(*arena-is-valid-clause-idx arena C*);
let b = (*arena-length arena C* = *2*);

        $ASSERT(L1 \neq L2)$;
        $ASSERT(length\ (W\ !\ (nat\text{-}of\text{-}lit\ L1)) < length\ arena)$;
        let $W = append\text{-}ll\ W\ (nat\text{-}of\text{-}lit\ L1)\ (to\text{-}watcher\ C\ L2\ b)$;
        $ASSERT(nat\text{-}of\text{-}lit\ L2 < length\ W)$;
        $ASSERT(length\ (W\ !\ (nat\text{-}of\text{-}lit\ L2)) < length\ arena)$;
        let $W = append\text{-}ll\ W\ (nat\text{-}of\text{-}lit\ L2)\ (to\text{-}watcher\ C\ L1\ b)$;
        RETURN W
      }
      else RETURN W
    })
   W
  }›
  ⟨*proof*⟩

lemma *arena-lit-pre-le-uint64-max*:
 ‹$length\ ba \leq uint64\text{-}max \Longrightarrow$
      $arena\text{-}lit\text{-}pre\ ba\ a \Longrightarrow a \leq uint64\text{-}max$›
  ⟨*proof*⟩

definition *rewatch-heur-st*
 :: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\ nres$›
where
‹$rewatch\text{-}heur\text{-}st = (\lambda(M, N0, D, Q, W, vm, \varphi, clvls, cach, lbd, outl,$
      $stats, fema, sema, t, vdom, avdom, ccount, lcount).$ *do* {
  $ASSERT(length\ vdom \leq length\ N0)$;
  $W \leftarrow rewatch\text{-}heur\ vdom\ N0\ W$;
  $RETURN\ (M, N0, D, Q, W, vm, \varphi, clvls, cach, lbd, outl,$
      $stats, fema, sema, t, vdom, avdom, ccount, lcount)$
  })›

definition *rewatch-heur-st-fast* **where**
  ‹$rewatch\text{-}heur\text{-}st\text{-}fast = rewatch\text{-}heur\text{-}st$›

definition *rewatch-heur-st-fast-pre* **where**
  ‹$rewatch\text{-}heur\text{-}st\text{-}fast\text{-}pre\ S =$
    $((\forall x \in set\ (get\text{-}vdom\ S).\ x \leq uint64\text{-}max) \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max)$›

definition *rewatch-st* :: ‹$'v\ twl\text{-}st\text{-}wl \Rightarrow 'v\ twl\text{-}st\text{-}wl\ nres$› **where**
  ‹$rewatch\text{-}st\ S = do\{$
    $(M, N, D, NE, UE, Q, W) \leftarrow RETURN\ S$;
    $W \leftarrow rewatch\ N\ W$;
    $RETURN\ ((M, N, D, NE, UE, Q, W))$
  }›

fun *remove-watched-wl* :: ‹$'v\ twl\text{-}st\text{-}wl \Rightarrow \text{-}$› **where**
  ‹$remove\text{-}watched\text{-}wl\ (M, N, D, NE, UE, Q, \text{-}) = (M, N, D, NE, UE, Q)$›

lemma *rewatch-st-correctness*:
  assumes ‹$get\text{-}watched\text{-}wl\ S = (\lambda\text{-}.\ [])$› **and**
   ‹$\bigwedge x.\ x \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow$
     $distinct\ ((get\text{-}clauses\text{-}wl\ S) \propto x) \wedge 2 \leq length\ ((get\text{-}clauses\text{-}wl\ S) \propto x)$›
  shows ‹$rewatch\text{-}st\ S \leq SPEC\ (\lambda T.\ remove\text{-}watched\text{-}wl\ S = remove\text{-}watched\text{-}wl\ T \wedge$
    $correct\text{-}watching\text{-}init\ T)$›
  ⟨*proof*⟩

0.1.14 Fast to slow conversion

Setup to convert a list from *uint64* to *nat*.

definition *convert-wlists-to-nat-conv* :: ‹'*a list list* ⇒ '*a list list*› **where**
‹*convert-wlists-to-nat-conv* = *id*›

definition *isasat-fast-slow* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*isasat-fast-slow* =
(λ(*M'*, *N'*, *D'*, *Q'*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*).
RETURN (*trail-pol-slow-of-fast M'*, *N'*, *D'*, *Q'*, *convert-wlists-to-nat-conv W'*, *vm*, *φ*,
clvls, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*, *ccount*, *vdom*, *avdom*, *nat-of-uint64-conv lcount*, *opts*, *old-arena*))›

definition (**in** −)*isasat-fast-slow-wl-D* **where**
‹*isasat-fast-slow-wl-D* = *id*›

lemma *isasat-fast-slow-alt-def*:
‹*isasat-fast-slow S* = *RETURN S*›
⟨*proof*⟩

lemma *isasat-fast-slow-isasat-fast-slow-wl-D*:
‹(*isasat-fast-slow*, *RETURN o isasat-fast-slow-wl-D*) ∈ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

abbreviation *twl-st-heur''*
:: ‹*nat multiset* ⇒ *nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur''* $\mathcal{D}$ *r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur'* $\mathcal{D}$ ∧
length (*get-clauses-wl-heur S*) = *r*}›

abbreviation *twl-st-heur-up''*
:: ‹*nat multiset* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur-up''* $\mathcal{D}$ *r s L* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur''* $\mathcal{D}$ *r* ∧
length (*watched-by T L*) = *s*}›

lemma *length-watched-le*:
assumes
prop-inv: ‹*correct-watching x1*› **and**
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *4*›
⟨*proof*⟩

lemma *length-watched-le2*:
assumes
prop-inv: ‹*correct-watching-except i j L x1*› **and**
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)› **and** *diff*: ‹*L* ≠ *x2*›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *4*›
⟨*proof*⟩

lemma *atm-of-all-lits-of-m*: ‹*atm-of* '# (*all-lits-of-m C*) = *atm-of* '# *C* + *atm-of* '# *C*›

‹*atm-of* ‘ *set-mset* (*all-lits-of-m* *C*) = *atm-of* ‘*set-mset* *C* ›
⟨*proof*⟩
end
theory *IsaSAT-Trail-SML*
imports *IsaSAT-Literals-SML Watched-Literals.Array-UInt IsaSAT-Trail*
Watched-Literals.IICF-Array-List32
begin

definition *tri-bool-assn* :: ‹*tri-bool* ⇒ *tri-bool-assn* ⇒ *assn*› **where**
‹*tri-bool-assn* = *hr-comp uint32-assn tri-bool-ref*›

lemma *UNSET-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return UNSET-code*), *uncurry0* (*RETURN UNSET*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
⟨*proof*⟩

lemma *equality-tri-bool-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry*(*RETURN oo tri-bool-eq*)) ∈
tri-bool-assn^k $*_a$ *tri-bool-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *SET-TRUE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return SET-TRUE-code*), *uncurry0* (*RETURN SET-TRUE*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
⟨*proof*⟩

lemma *SET-FALSE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return SET-FALSE-code*), *uncurry0* (*RETURN SET-FALSE*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
⟨*proof*⟩

lemma [*safe-constraint-rules*]:
‹*is-pure tri-bool-assn*›
⟨*proof*⟩

type-synonym *trail-pol-assn* =
‹*uint32 array-list* × *tri-bool-assn array* × *uint32 array* × *nat array* × *uint32* ×
uint32 array-list›

type-synonym *trail-pol-fast-assn* =
‹*uint32 array-list32* × *tri-bool-assn array* × *uint32 array* ×
uint64 array × *uint32* ×
uint32 array-list32›

lemma *DECISION-REASON-uint64*:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN DECISION-REASON*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *DECISION-REASON'*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN DECISION-REASON*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
⟨*proof*⟩

abbreviation *trail-pol-assn* :: ‹*trail-pol* ⇒ *trail-pol-assn* ⇒ *assn*› **where**
‹*trail-pol-assn* ≡
arl-assn unat-lit-assn ∗*a* *array-assn* (*tri-bool-assn*) ∗*a*
array-assn uint32-nat-assn ∗*a*
array-assn (*nat-assn*) ∗*a* *uint32-nat-assn* ∗*a* *arl-assn uint32-nat-assn*›

abbreviation *trail-pol-fast-assn* :: ‹*trail-pol* ⇒ *trail-pol-fast-assn* ⇒ *assn*› **where**
‹*trail-pol-fast-assn* ≡
arl32-assn unat-lit-assn ∗*a* *array-assn* (*tri-bool-assn*) ∗*a*
array-assn uint32-nat-assn ∗*a*
array-assn uint64-nat-assn ∗*a* *uint32-nat-assn* ∗*a*
arl32-assn uint32-nat-assn›

Code generation

Conversion between incomplete and complete mode **sepref-definition** *trail-pol-slow-of-fast-code*
is ‹*RETURN o trail-pol-slow-of-fast*›
:: ‹$\textit{trail-pol-fast-assn}^d \rightarrow_a$ *trail-pol-assn*›
⟨*proof*⟩

lemma *count-decided-trail*[*sepref-fr-rules*]:
‹(*return o count-decided-pol*, *RETURN o count-decided-pol*) ∈ $\textit{trail-pol-assn}^k \rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *count-decided-trail-fast*[*sepref-fr-rules*]:
‹(*return o count-decided-pol*, *RETURN o count-decided-pol*) ∈ $\textit{trail-pol-fast-assn}^k \rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

declare *trail-pol-slow-of-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-atm-code*
is ‹*uncurry* (*RETURN oo get-level-atm-pol*)›
:: ‹$[\textit{get-level-atm-pol-pre}]_a$
$\textit{trail-pol-assn}^k *_a \textit{uint32-nat-assn}^k \rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

declare *get-level-atm-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-atm-fast-code*
is ‹*uncurry* (*RETURN oo get-level-atm-pol*)›
:: ‹$[\textit{get-level-atm-pol-pre}]_a$
$\textit{trail-pol-fast-assn}^k *_a \textit{uint32-nat-assn}^k \rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

declare *get-level-atm-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-code*
is ‹*uncurry* (*RETURN oo get-level-pol*)›
:: ‹$[\textit{get-level-pol-pre}]_a$
$\textit{trail-pol-assn}^k *_a \textit{unat-lit-assn}^k \rightarrow$ *uint32-nat-assn*›
⟨*proof*⟩

declare *get-level-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-fast-code*
is ‹*uncurry* (*RETURN oo get-level-pol*)›
:: ‹$[\textit{get-level-pol-pre}]_a$
$\textit{trail-pol-fast-assn}^k *_a \textit{unat-lit-assn}^k \rightarrow$ *uint32-nat-assn*›

⟨proof⟩

declare *get-level-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *polarity-pol-code*
is ‹*uncurry* (*RETURN oo polarity-pol*)›
:: ‹[*uncurry polarity-pol-pre*]$_a$ *trail-pol-assn*^{k} $*_a$ *unat-lit-assn*^{k} → *tri-bool-assn*›
⟨proof⟩

declare *polarity-pol-code.refine*[*sepref-fr-rules*]

sepref-definition *polarity-pol-fast-code*
is ‹*uncurry* (*RETURN oo polarity-pol*)›
:: ‹[*uncurry polarity-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} → *tri-bool-assn*›
⟨proof⟩

declare *polarity-pol-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-length-trail-code*
is ‹*RETURN o isa-length-trail*›
:: ‹[*isa-length-trail-pre*]$_a$ *trail-pol-assn*^{k} → *uint32-nat-assn*›
⟨proof⟩

sepref-definition *isa-length-trail-fast-code*
is ‹*RETURN o isa-length-trail*›
:: ‹[*isa-length-trail-pre*]$_a$ *trail-pol-fast-assn*^{k} → *uint32-nat-assn*›
⟨proof⟩

declare *isa-length-trail-code.refine*[*sepref-fr-rules*]
isa-length-trail-fast-code.refine[*sepref-fr-rules*]

sepref-definition *cons-trail-Propagated-tr-code*
is ‹*uncurry2* (*RETURN ooo cons-trail-Propagated-tr*)›
:: ‹[*cons-trail-Propagated-tr-pre*]$_a$
unat-lit-assn^k $*_a$ *nat-assn*^{k} $*_a$ *trail-pol-assn*^{d} → *trail-pol-assn*›
⟨proof⟩

declare *cons-trail-Propagated-tr-code.refine*[*sepref-fr-rules*]

sepref-definition *cons-trail-Propagated-tr-fast-code*
is ‹*uncurry2* (*RETURN ooo cons-trail-Propagated-tr*)›
:: ‹[*cons-trail-Propagated-tr-pre*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *trail-pol-fast-assn*^{d} → *trail-pol-fast-assn*›
⟨proof⟩

declare *cons-trail-Propagated-tr-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −)*last-trail-code*
is ‹*RETURN o last-trail-pol*›
:: ‹[*last-trail-pol-pre*]$_a$
trail-pol-assn^k → *unat-lit-assn* ∗*a option-assn nat-assn*›
⟨proof⟩

declare *last-trail-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −)*last-trail-fast-code*

is ‹*RETURN o last-trail-pol*›
:: ‹[*last-trail-pol-pre*]$_a$
trail-pol-fast-assn^k → *unat-lit-assn* ∗*a option-assn uint64-nat-assn*›
⟨*proof*⟩

declare *last-trail-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *tl-trail-tr-code*
is ‹*RETURN o tl-trailt-tr*›
:: ‹[*tl-trailt-tr-pre*]$_a$
trail-pol-assn^d → *trail-pol-assn*›
⟨*proof*⟩

declare *tl-trail-tr-code.refine*[*sepref-fr-rules*]

sepref-definition *tl-trail-tr-fast-code*
is ‹*RETURN o tl-trailt-tr*›
:: ‹[*tl-trailt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*›
⟨*proof*⟩

declare *tl-trail-tr-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *tl-trail-proped-tr-code*
is ‹*RETURN o tl-trail-propedt-tr*›
:: ‹[*tl-trail-propedt-tr-pre*]$_a$
trail-pol-assn^d → *trail-pol-assn*›
⟨*proof*⟩

declare *tl-trail-proped-tr-code.refine*[*sepref-fr-rules*]

sepref-definition *tl-trail-proped-tr-fast-code*
is ‹*RETURN o tl-trail-propedt-tr*›
:: ‹[*tl-trail-propedt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*›
⟨*proof*⟩

declare *tl-trail-proped-tr-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *lit-of-last-trail-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹[λ(*M*, -). *M* ≠ []]$_a$ *trail-pol-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-definition (**in** −) *lit-of-last-trail-fast-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹[λ(*M*, -). *M* ≠ []]$_a$ *trail-pol-fast-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

declare *lit-of-last-trail-code.refine*[*sepref-fr-rules*]
declare *lit-of-last-trail-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *cons-trail-Decided-tr-code*
is ‹*uncurry* (*RETURN oo cons-trail-Decided-tr*)›
:: ‹[*cons-trail-Decided-tr-pre*]$_a$

unat-lit-assn^k $*_a$ *trail-pol-assn*^{d} $\rightarrow$ *trail-pol-assn*⟩
⟨*proof*⟩

declare *cons-trail-Decided-tr-code.refine*[*sepref-fr-rules*]

sepref-definition *cons-trail-Decided-tr-fast-code*
is ⟨*uncurry* (*RETURN oo cons-trail-Decided-tr*)⟩
:: ⟨[*cons-trail-Decided-tr-pre*]$_a$
unat-lit-assn^k $*_a$ *trail-pol-fast-assn*^{d} $\rightarrow$ *trail-pol-fast-assn*⟩
⟨*proof*⟩

declare *cons-trail-Decided-tr-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *defined-atm-code*
is ⟨*uncurry* (*RETURN oo defined-atm-pol*)⟩
:: ⟨[*uncurry defined-atm-pol-pre*]$_a$ *trail-pol-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *bool-assn*⟩
⟨*proof*⟩

declare *defined-atm-code.refine*[*sepref-fr-rules*]

sepref-definition *defined-atm-fast-code*
is ⟨*uncurry* (*RETURN oo defined-atm-pol*)⟩
:: ⟨[*uncurry defined-atm-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *bool-assn*⟩
⟨*proof*⟩

declare *defined-atm-code.refine*[*sepref-fr-rules*]
defined-atm-fast-code.refine[*sepref-fr-rules*]

sepref-register *get-propagation-reason*

sepref-definition *get-propagation-reason-code*
is ⟨*uncurry get-propagation-reason-pol*⟩
:: ⟨*trail-pol-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn nat-assn*⟩
⟨*proof*⟩

sepref-definition *get-propagation-reason-fast-code*
is ⟨*uncurry get-propagation-reason-pol*⟩
:: ⟨*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn uint64-nat-assn*⟩
⟨*proof*⟩

declare *get-propagation-reason-fast-code.refine*[*sepref-fr-rules*]
get-propagation-reason-code.refine[*sepref-fr-rules*]

sepref-definition *get-the-propagation-reason-code*
is ⟨*uncurry get-the-propagation-reason-pol*⟩
:: ⟨*trail-pol-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn nat-assn*⟩
⟨*proof*⟩

sepref-definition (**in** −) *get-the-propagation-reason-fast-code*
is ⟨*uncurry get-the-propagation-reason-pol*⟩
:: ⟨*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn uint64-nat-assn*⟩
⟨*proof*⟩

declare *get-the-propagation-reason-fast-code.refine*[*sepref-fr-rules*]
get-the-propagation-reason-code.refine[*sepref-fr-rules*]

sepref-definition *isa-trail-nth-code*
  is ‹*uncurry isa-trail-nth*›
  :: ‹*trail-pol-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*›
  ⟨*proof*⟩

sepref-definition *isa-trail-nth-fast-code*
  is ‹*uncurry isa-trail-nth*›
  :: ‹*trail-pol-fast-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*›
  ⟨*proof*⟩

declare *isa-trail-nth-code.refine*[*sepref-fr-rules*]
  isa-trail-nth-fast-code.refine[*sepref-fr-rules*]

sepref-definition *tl-trail-tr-no-CS-code*
  is ‹*RETURN o tl-trailt-tr-no-CS*›
  :: ‹[*tl-trailt-tr-no-CS-pre*]$_a$
      trail-pol-assn^d $\rightarrow$ *trail-pol-assn*›
  ⟨*proof*⟩

sepref-definition *tl-trail-tr-no-CS-fast-code*
  is ‹*RETURN o tl-trailt-tr-no-CS*›
  :: ‹[*tl-trailt-tr-no-CS-pre*]$_a$
      trail-pol-fast-assn^d $\rightarrow$ *trail-pol-fast-assn*›
  ⟨*proof*⟩

abbreviation (**in** −) *trail-pol-assn'* :: ‹*trail-pol* ⇒ *trail-pol-assn* ⇒ *assn*› **where**
  ‹*trail-pol-assn'* ≡
    arl-assn unat-lit-assn ∗*a array-assn* (*tri-bool-assn*) ∗*a*
    array-assn uint32-nat-assn ∗*a*
    array-assn nat-assn ∗*a uint32-nat-assn* ∗*a arl-assn uint32-nat-assn*›

abbreviation (**in** −) *trail-pol-fast-assn'* :: ‹*trail-pol* ⇒ *trail-pol-fast-assn* ⇒ *assn*› **where**
  ‹*trail-pol-fast-assn'* ≡
    arl32-assn unat-lit-assn ∗*a array-assn* (*tri-bool-assn*) ∗*a*
    array-assn uint32-nat-assn ∗*a*
    array-assn uint64-nat-assn ∗*a uint32-nat-assn* ∗*a arl32-assn uint32-nat-assn*›

lemma (**in** −) *take-arl-assn*[*sepref-fr-rules*]:
  ‹(*uncurry* (*return oo take-arl*), *uncurry* (*RETURN oo take*))
    $\in$ [λ(*j*, *xs*). *j* $\le$ *length xs*]$_a$ *nat-assn*^{k} $*_a$ (*arl-assn R*)^d $\rightarrow$ *arl-assn R*›
  ⟨*proof*⟩

sepref-definition (**in** −) *trail-conv-back-imp-code*
  is ‹*uncurry trail-conv-back-imp*›
  :: ‹*uint32-nat-assn*^{k} $*_a$ *trail-pol-assn'*^{d} $\rightarrow_a$ *trail-pol-assn'*›
  ⟨*proof*⟩

declare *trail-conv-back-imp-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *trail-conv-back-imp-fast-code*
  is ‹*uncurry trail-conv-back-imp*›
  :: ‹*uint32-nat-assn*^{k} $*_a$ *trail-pol-fast-assn'*^{d} $\rightarrow_a$ *trail-pol-fast-assn'*›
  ⟨*proof*⟩

declare *trail-conv-back-imp-fast-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT-Lookup-Conflict-SML*
imports
  IsaSAT-Lookup-Conflict
  IsaSAT-Trail-SML
  IsaSAT-Clauses-SML
  LBD-SML
begin

sepref-register *set-lookup-conflict-aa*

abbreviation *option-bool-assn* **where**
⟨*option-bool-assn* ≡ *pure option-bool-rel*⟩

type-synonym (**in** −) *out-learned-assn* = ⟨*uint32 array-list32*⟩

abbreviation (**in** −) *out-learned-assn* :: ⟨*out-learned* ⇒ *out-learned-assn* ⇒ *assn*⟩ **where**
⟨*out-learned-assn* ≡ *arl32-assn unat-lit-assn*⟩

abbreviation (**in** −) *minimize-status-assn* **where**
⟨*minimize-status-assn* ≡ (*id-assn* :: *minimize-status* ⇒ -)⟩

abbreviation (**in** −) *lookup-clause-rel-assn*
:: ⟨*lookup-clause-rel* ⇒ *lookup-clause-assn* ⇒ *assn*⟩
where
⟨*lookup-clause-rel-assn* ≡ (*uint32-nat-assn* ∗*a* *array-assn option-bool-assn*)⟩

abbreviation (**in** −)*conflict-option-rel-assn*
:: ⟨*conflict-option-rel* ⇒ *option-lookup-clause-assn* ⇒ *assn*⟩
where
⟨*conflict-option-rel-assn* ≡ (*bool-assn* ∗*a* *lookup-clause-rel-assn*)⟩

abbreviation *isasat-conflict-assn* **where**
⟨*isasat-conflict-assn* ≡ *bool-assn* ∗*a* *uint32-nat-assn* ∗*a* *array-assn option-bool-assn*⟩

definition (**in** −)*ana-refinement-assn* **where**
⟨*ana-refinement-assn* ≡ *hr-comp* (*nat-assn* ∗*a* *uint64-assn*) *analyse-refinement-rel*⟩

definition (**in** −)*ana-refinement-fast-assn* **where**
⟨*ana-refinement-fast-assn* ≡ *hr-comp* (*uint64-nat-assn* ∗*a* *uint64-assn*) *analyse-refinement-rel*⟩

abbreviation (**in** −)*analyse-refinement-assn* **where**
⟨*analyse-refinement-assn* ≡ *arl32-assn ana-refinement-assn*⟩

lemma *ex-assn-def-pure-eq-start*:
⟨$(\exists_A ba. \uparrow (ba = h) * P\ ba) = P\ h$⟩
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start'*:
⟨$(\exists_A ba. \uparrow (h = ba) * P\ ba) = P\ h$⟩
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start2*:

$\langle(\exists_A ba\ b.\ \uparrow (ba = h\ b) * P\ b\ ba) = (\exists_A b\ .\ \ P\ b\ (h\ b))\rangle$
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start3*:
$\langle(\exists_A ba\ b\ c.\ \uparrow (ba = h\ b) * P\ b\ ba\ c) = (\exists_A b\ c.\ \ P\ b\ (h\ b)\ c)\rangle$
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start3'*:
$\langle(\exists_A ba\ b\ c.\ \uparrow (bb = ba) * P\ b\ ba\ c) = (\exists_A b\ c.\ \ P\ b\ bb\ c)\rangle$
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start4'*:
$\langle(\exists_A ba\ b\ c\ d.\ \uparrow (bb = ba) * P\ b\ ba\ c\ d) = (\exists_A b\ c\ d.\ \ P\ b\ bb\ c\ d)\rangle$
⟨*proof*⟩

lemma *ex-assn-def-pure-eq-start1*:
$\langle(\exists_A ba.\ \uparrow (ba = h\ b) * P\ ba) = (P\ (h\ b))\rangle$
⟨*proof*⟩

lemma *ex-assn-cong*:
$\langle(\bigwedge x.\ P\ x = P'\ x) \Longrightarrow (\exists_A x.\ P\ x) = (\exists_A x.\ P'\ x)\rangle$
⟨*proof*⟩

abbreviation (**in** $-$)*analyse-refinement-fast-assn* **where**
⟨*analyse-refinement-fast-assn* ≡
arl32-assn ana-refinement-fast-assn⟩

lemma *lookup-clause-assn-is-None-lookup-clause-assn-is-None*:
⟨(*return o lookup-clause-assn-is-None*, *RETURN o lookup-clause-assn-is-None*) ∈
conflict-option-rel-assn^k $\rightarrow_a$ *bool-assn*⟩
⟨*proof*⟩

lemma *NOTIN-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return False*), *uncurry0* (*RETURN NOTIN*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *option-bool-assn*⟩
⟨*proof*⟩

lemma *POSIN-hnr*[*sepref-fr-rules*]:
⟨(*return o* (λ-. *True*), *RETURN o ISIN*) ∈ *bool-assn*^{k} $\rightarrow_a$ *option-bool-assn*⟩
⟨*proof*⟩

lemma *is-NOTIN-hnr*[*sepref-fr-rules*]:
⟨(*return o Not*, *RETURN o is-NOTIN*) ∈ *option-bool-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
⟨*proof*⟩

lemma (**in** $-$) *SEEN-REMOVABLE*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return SEEN-REMOVABLE*),*uncurry0* (*RETURN SEEN-REMOVABLE*)) ∈
unit-assn^k $\rightarrow_a$ *minimize-status-assn*⟩
⟨*proof*⟩

lemma (**in** $-$) *SEEN-FAILED*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return SEEN-FAILED*),*uncurry0* (*RETURN SEEN-FAILED*)) ∈
unit-assn^k $\rightarrow_a$ *minimize-status-assn*⟩
⟨*proof*⟩

lemma (**in** −) *SEEN-UNKNOWN*[*sepref-fr-rules*]:
‹(*Sepref-Misc.uncurry0* (*return SEEN-UNKNOWN*),*Sepref-Misc.uncurry0* (*RETURN SEEN-UNKNOWN*)) ∈
$unit\text{-}assn^k \rightarrow_a minimize\text{-}status\text{-}assn$›
⟨*proof*⟩

lemma *size-lookup-conflict*[*sepref-fr-rules*]:
‹(*return o* (λ(-, *n*, -). *n*), *RETURN o size-lookup-conflict*) ∈
$(bool\text{-}assn *a\ lookup\text{-}clause\text{-}rel\text{-}assn)^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

lemma *option-bool-assn-is-None*[*sepref-fr-rules*]:
‹(*return o Not*, *RETURN o is-None*) ∈ $option\text{-}bool\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *is-in-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-lookup-conflict*)›
:: ‹$[\lambda((n, xs), L).\ atm\text{-}of\ L < length\ xs]_a$
$lookup\text{-}clause\text{-}rel\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

declare *is-in-conflict-code.refine*[*sepref-fr-rules*]

lemma *lookup-clause-assn-is-empty-lookup-clause-assn-is-empty*:
‹(*return o lookup-clause-assn-is-empty*, *RETURN o lookup-clause-assn-is-empty*) ∈
$conflict\text{-}option\text{-}rel\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

lemma *to-ana-ref-id-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-ana-ref*), *uncurry2* (*RETURN ooo to-ana-ref-id*)) ∈
$uint64\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow_a$
ana-refinement-fast-assn›
⟨*proof*⟩

lemma *to-ana-ref-id-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-ana-ref*), *uncurry2* (*RETURN ooo to-ana-ref-id*)) ∈
$nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow_a$
ana-refinement-assn›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹((*return o from-ana-ref*), (*RETURN o from-ana-ref-id*)) ∈
$ana\text{-}refinement\text{-}fast\text{-}assn^k \rightarrow_a$
uint64-nat-assn ∗*a uint32-nat-assn* ∗*a bool-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹((*return o from-ana-ref*), (*RETURN o from-ana-ref-id*)) ∈
$ana\text{-}refinement\text{-}assn^k \rightarrow_a$
nat-assn ∗*a uint32-nat-assn* ∗*a bool-assn*›
⟨*proof*⟩

lemma *minimize-status-eq-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) ∈
$minimize\text{-}status\text{-}assn^k *_a minimize\text{-}status\text{-}assn^k \rightarrow_a bool\text{-}assn$›

⟨*proof*⟩

abbreviation (**in** −) *cach-refinement-l-assn* **where**
‹*cach-refinement-l-assn* ≡ *array-assn minimize-status-assn* ∗*a arl32-assn uint32-nat-assn*›

sepref-register *conflict-min-cach-l*
sepref-definition (**in** −) *delete-from-lookup-conflict-code*
is ‹*uncurry delete-from-lookup-conflict*›
:: ‹$\textit{unat-lit-assn}^k *_a \textit{lookup-clause-rel-assn}^d \rightarrow_a \textit{lookup-clause-rel-assn}$›
⟨*proof*⟩

sepref-definition *resolve-lookup-conflict-merge-code*
is ‹*uncurry6 isa-set-lookup-conflict*›
:: ‹$[\lambda((((((M, N), i), (-, xs)), -), -), out).\ i < \textit{length } N]_a$
$\textit{trail-pol-assn}^k *_a \textit{arena-assn}^k *_a \textit{nat-assn}^k *_a \textit{conflict-option-rel-assn}^d *_a$
$\textit{uint32-nat-assn}^k *_a \textit{lbd-assn}^d *_a \textit{out-learned-assn}^d \rightarrow$
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
⟨*proof*⟩

declare *resolve-lookup-conflict-merge-code.refine*[*sepref-fr-rules*]

sepref-definition *resolve-lookup-conflict-merge-fast-code*
is ‹*uncurry6 isa-set-lookup-conflict*›
:: ‹$[\lambda((((((M, N), i), (-, xs)), -), -), out).\ i < \textit{length } N \wedge$
$\textit{length } N \leq \textit{uint64-max}]_a$
$\textit{trail-pol-fast-assn}^k *_a \textit{arena-fast-assn}^k *_a \textit{uint64-nat-assn}^k *_a \textit{conflict-option-rel-assn}^d *_a$
$\textit{uint32-nat-assn}^k *_a \textit{lbd-assn}^d *_a \textit{out-learned-assn}^d \rightarrow$
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
⟨*proof*⟩

declare *resolve-lookup-conflict-merge-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *set-lookup-conflict-aa-code*
is ‹*uncurry6 isa-set-lookup-conflict-aa*›
:: ‹$\textit{trail-pol-assn}^k *_a \textit{arena-assn}^k *_a \textit{nat-assn}^k *_a \textit{conflict-option-rel-assn}^d *_a$
$\textit{uint32-nat-assn}^k *_a \textit{lbd-assn}^d *_a \textit{out-learned-assn}^d \rightarrow_a$
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
⟨*proof*⟩

declare *set-lookup-conflict-aa-code.refine*[*sepref-fr-rules*]

sepref-definition *set-lookup-conflict-aa-fast-code*
is ‹*uncurry6 isa-set-lookup-conflict-aa*›
:: ‹$[\lambda((((((M, N), i), (-, xs)), -), -), -).\ \textit{length } N \leq \textit{uint64-max}]_a$
$\textit{trail-pol-fast-assn}^k *_a \textit{arena-fast-assn}^k *_a \textit{uint64-nat-assn}^k *_a \textit{conflict-option-rel-assn}^d *_a$
$\textit{uint32-nat-assn}^k *_a \textit{lbd-assn}^d *_a \textit{out-learned-assn}^d \rightarrow$
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
⟨*proof*⟩

declare *set-lookup-conflict-aa-fast-code.refine*[*sepref-fr-rules*]

sepref-register *isa-resolve-merge-conflict-gt2*

sepref-definition *resolve-merge-conflict-code*
is ‹*uncurry6 isa-resolve-merge-conflict-gt2*›
:: ‹$[isa\text{-}set\text{-}lookup\text{-}conflict\text{-}aa\text{-}pre]_a$
$trail\text{-}pol\text{-}assn^k *_a arena\text{-}assn^k *_a nat\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d *_a$
$uint32\text{-}nat\text{-}assn^k *_a lbd\text{-}assn^d *_a out\text{-}learned\text{-}assn^d \rightarrow$
$conflict\text{-}option\text{-}rel\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ lbd\text{-}assn *a\ out\text{-}learned\text{-}assn$›
⟨*proof*⟩

declare *resolve-merge-conflict-code.refine*[*sepref-fr-rules*]

sepref-definition *resolve-merge-conflict-fast-code*
is ‹*uncurry6 isa-resolve-merge-conflict-gt2*›
:: ‹$[uncurry6\ (\lambda M\ N\ i\ (b, xs)\ clvls\ lbd\ outl.\ length\ N \leq uint64\text{-}max \wedge$
$isa\text{-}set\text{-}lookup\text{-}conflict\text{-}aa\text{-}pre\ ((((((M, N), i), (b, xs)), clvls), lbd), outl))]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d *_a$
$uint32\text{-}nat\text{-}assn^k *_a lbd\text{-}assn^d *_a out\text{-}learned\text{-}assn^d \rightarrow$
$conflict\text{-}option\text{-}rel\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ lbd\text{-}assn *a\ out\text{-}learned\text{-}assn$›
⟨*proof*⟩

declare *resolve-merge-conflict-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *atm-in-conflict-code*
is ‹*uncurry* (*RETURN oo atm-in-conflict-lookup*)›
:: ‹$[uncurry\ atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre]_a$
$uint32\text{-}nat\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

declare *atm-in-conflict-code.refine*[*sepref-fr-rules*]
sepref-definition (**in** −) *conflict-min-cach-l-code*
is ‹*uncurry* (*RETURN oo conflict-min-cach-l*)›
:: ‹$[conflict\text{-}min\text{-}cach\text{-}l\text{-}pre]_a\ cach\text{-}refinement\text{-}l\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow minimize\text{-}status\text{-}assn$›
⟨*proof*⟩

declare *conflict-min-cach-l-code.refine*[*sepref-fr-rules*]

lemma *conflict-min-cach-set-failed-l-alt-def*:
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if b then sup* @ [*L*] *else sup*)
})›
⟨*proof*⟩

lemma *le-uint32-max-div2-le-uint32-max*: ‹$a2' \leq Suc\ (uint\text{-}max\ div\ 2) \Longrightarrow a2' < uint\text{-}max$›
⟨*proof*⟩

sepref-definition (**in** −) *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹$cach\text{-}refinement\text{-}l\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a cach\text{-}refinement\text{-}l\text{-}assn$›
⟨*proof*⟩

lemma *conflict-min-cach-set-removable-l-alt-def*:
‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if b then sup* @ [*L*] *else sup*)
})›
⟨*proof*⟩

sepref-definition (**in** −) *conflict-min-cach-set-removable-l-code*
is ‹*uncurry conflict-min-cach-set-removable-l*›
:: ‹*cach-refinement-l-assn*^{d} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *cach-refinement-l-assn*›
⟨*proof*⟩

declare *conflict-min-cach-set-removable-l-code.refine*[*sepref-fr-rules*]

lemma *lookup-conflict-size-hnr*[*sepref-fr-rules*]:
‹(*return o fst*, *RETURN o lookup-conflict-size*) ∈ *lookup-clause-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
⟨*proof*⟩
lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure ana-refinement-fast-assn*›
⟨*proof*⟩

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure ana-refinement-assn*›
⟨*proof*⟩

sepref-register *lookup-conflict-remove1*

sepref-register *isa-lit-redundant-rec-wl-lookup*

abbreviation (**in** −) *highest-lit-assn* **where**
‹*highest-lit-assn* ≡ *option-assn* (*unat-lit-assn* ∗*a* *uint32-nat-assn*)›

sepref-register *from-ana-ref-id*

sepref-register *isa-mark-failed-lits-stack*

sepref-register *lit-redundant-rec-wl-lookup conflict-min-cach-set-removable-l*
get-propagation-reason-pol lit-redundant-reason-stack-wl-lookup

sepref-register *isa-minimize-and-extract-highest-lookup-conflict isa-literal-redundant-wl-lookup*

lemma *set-lookup-empty-conflict-to-none-hnr*[*sepref-fr-rules*]:
‹(*return o set-lookup-empty-conflict-to-none*, *RETURN o set-lookup-empty-conflict-to-none*) ∈
lookup-clause-rel-assn^d $\rightarrow_a$ *conflict-option-rel-assn*›
⟨*proof*⟩

lemma *isa-mark-failed-lits-stackI*:
assumes
‹*length ba* ≤ *Suc* (*uint-max div 2*)› **and**
‹*a1'* < *length ba*›

shows ‹*Suc a1′* ≤ *uint-max*›
⟨*proof*⟩

sepref-register *to-ana-ref-id*
sepref-definition *isa-mark-failed-lits-stack-code*
is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
:: ‹$arena\text{-}assn^k *_a analyse\text{-}refinement\text{-}assn^d *_a cach\text{-}refinement\text{-}l\text{-}assn^d \rightarrow_a$ *cach-refinement-l-assn*›
⟨*proof*⟩

sepref-definition *isa-mark-failed-lits-stack-fast-code*
is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
:: ‹$[\lambda((N, -), -).\ length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^k *_a analyse\text{-}refinement\text{-}fast\text{-}assn^d *_a cach\text{-}refinement\text{-}l\text{-}assn^d \rightarrow$
cach-refinement-l-assn›
⟨*proof*⟩

declare *isa-mark-failed-lits-stack-code.refine*[*sepref-fr-rules*]
isa-mark-failed-lits-stack-fast-code.refine[*sepref-fr-rules*]

sepref-definition *isa-get-literal-and-remove-of-analyse-wl-code*
is ‹*uncurry* (*RETURN oo isa-get-literal-and-remove-of-analyse-wl*)›
:: ‹$[uncurry\ isa\text{-}get\text{-}literal\text{-}and\text{-}remove\text{-}of\text{-}analyse\text{-}wl\text{-}pre]_a$
$arena\text{-}assn^k *_a analyse\text{-}refinement\text{-}assn^d \rightarrow$
unat-lit-assn ∗*a analyse-refinement-assn*›
⟨*proof*⟩

sepref-definition *isa-get-literal-and-remove-of-analyse-wl-fast-code*
is ‹*uncurry* (*RETURN oo isa-get-literal-and-remove-of-analyse-wl*)›
:: ‹$[\lambda(arena, analyse).\ isa\text{-}get\text{-}literal\text{-}and\text{-}remove\text{-}of\text{-}analyse\text{-}wl\text{-}pre\ arena\ analyse \wedge$
$length\ arena \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^k *_a analyse\text{-}refinement\text{-}fast\text{-}assn^d \rightarrow$
unat-lit-assn ∗*a analyse-refinement-fast-assn*›
⟨*proof*⟩

declare *isa-get-literal-and-remove-of-analyse-wl-code.refine*[*sepref-fr-rules*]
declare *isa-get-literal-and-remove-of-analyse-wl-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *ana-lookup-conv-lookup-fast-code*
is ‹*uncurry* (*RETURN oo ana-lookup-conv-lookup*)›
:: ‹$[uncurry\ ana\text{-}lookup\text{-}conv\text{-}lookup\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a$
$(uint64\text{-}nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ bool\text{-}assn)^k$
→ *uint64-nat-assn* ∗*a uint64-nat-assn* ∗*a uint64-nat-assn* ∗*a uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *ana-lookup-conv-lookup-code*
is ‹*uncurry* (*RETURN oo ana-lookup-conv-lookup*)›
:: ‹$[uncurry\ ana\text{-}lookup\text{-}conv\text{-}lookup\text{-}pre]_a\ arena\text{-}assn^k *_a$
$(nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ bool\text{-}assn)^k$
→ *nat-assn* ∗*a uint64-nat-assn* ∗*a uint64-nat-assn* ∗*a uint64-nat-assn*›
⟨*proof*⟩

declare *ana-lookup-conv-lookup-fast-code.refine*[*sepref-fr-rules*]
ana-lookup-conv-lookup-code.refine[*sepref-fr-rules*]

sepref-definition *lit-redundant-reason-stack-wl-lookup-code*
is ‹*uncurry2* (*RETURN ooo lit-redundant-reason-stack-wl-lookup*)›
:: ‹[*uncurry2 lit-redundant-reason-stack-wl-lookup-pre*]$_a$
unat-lit-assn^k $*_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$
ana-refinement-assn›
⟨*proof*⟩

sepref-definition *lit-redundant-reason-stack-wl-lookup-fast-code*
is ‹*uncurry2* (*RETURN ooo lit-redundant-reason-stack-wl-lookup*)›
:: ‹[*uncurry2 lit-redundant-reason-stack-wl-lookup-pre*]$_a$
unat-lit-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$
ana-refinement-fast-assn›
⟨*proof*⟩

declare *lit-redundant-reason-stack-wl-lookup-fast-code.refine*[*sepref-fr-rules*]
lit-redundant-reason-stack-wl-lookup-code.refine[*sepref-fr-rules*]

declare *get-propagation-reason-code.refine*[*sepref-fr-rules*]

lemma *isa-lit-redundant-rec-wl-lookupI*:
assumes
‹*length ba* $\leq$ *Suc* (*uint-max div 2*)›
shows ‹*length ba* $<$ *uint-max*›
⟨*proof*⟩

sepref-definition *lit-redundant-rec-wl-lookup-code*
is ‹*uncurry5* (*isa-lit-redundant-rec-wl-lookup*)›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *analysis*), *lbd*). *True*]$_a$
trail-pol-assn^k $*_a$ *arena-assn*^{k} $*_a$ (*uint32-nat-assn* $*a$ *array-assn option-bool-assn*)^k $*_a$
cach-refinement-l-assn^d $*_a$ *analyse-refinement-assn*^{d} $*_a$ *lbd-assn*^{k} $\rightarrow$
cach-refinement-l-assn $*a$ *analyse-refinement-assn* $*a$ *bool-assn*›
⟨*proof*⟩

declare *lit-redundant-rec-wl-lookup-code.refine*[*sepref-fr-rules*]

sepref-definition *lit-redundant-rec-wl-lookup-fast-code*
is ‹*uncurry5* (*isa-lit-redundant-rec-wl-lookup*)›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *analysis*), *lbd*). *length NU* $\leq$ *uint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ (*uint32-nat-assn* $*a$ *array-assn option-bool-assn*)^k $*_a$
cach-refinement-l-assn^d $*_a$ *analyse-refinement-fast-assn*^{d} $*_a$ *lbd-assn*^{k} $\rightarrow$
cach-refinement-l-assn $*a$ *analyse-refinement-fast-assn* $*a$ *bool-assn*›
⟨*proof*⟩

declare *lit-redundant-rec-wl-lookup-fast-code.refine*[*sepref-fr-rules*]

definition *arl32-butlast-nonresizing* :: ‹'*a array-list32* $\Rightarrow$ '*a array-list32*› **where**
‹*arl32-butlast-nonresizing* = (λ(*xs*, *a*). (*xs*, *a* $-$ *1*))›

lemma *butlast32-nonresizing-hnr*[*sepref-fr-rules*]:
‹(*return o arl32-butlast-nonresizing*, *RETURN o butlast-nonresizing*) $\in$
[λ*xs*. *xs* $\neq$ []]$_a$ (*arl32-assn R*)^d $\rightarrow$ *arl32-assn R*›
⟨*proof*⟩

find-theorems *butlast arl32-assn*
sepref-definition *delete-index-and-swap-code*
is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
:: ‹$[\lambda(xs, i).\ i < length\ xs]_a$
$(arl32\text{-}assn\ unat\text{-}lit\text{-}assn)^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow arl32\text{-}assn\ unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *delete-index-and-swap-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −)*lookup-conflict-upd-None-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-upd-None*)›
:: ‹$[\lambda((n, xs), i).\ i < length\ xs \wedge n > 0]_a$
$lookup\text{-}clause\text{-}rel\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow lookup\text{-}clause\text{-}rel\text{-}assn$›
⟨*proof*⟩

declare *lookup-conflict-upd-None-code.refine*[*sepref-fr-rules*]

lemma *uint32-max-ge0*: ‹*0* < *uint-max*› ⟨*proof*⟩
sepref-definition *literal-redundant-wl-lookup-code*
is ‹*uncurry5 isa-literal-redundant-wl-lookup*›
:: ‹$[\lambda(((((M, NU), D), cach), L), lbd).\ True]_a$
$trail\text{-}pol\text{-}assn^k *_a arena\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k *_a$
$cach\text{-}refinement\text{-}l\text{-}assn^d *_a unat\text{-}lit\text{-}assn^k *_a lbd\text{-}assn^k \rightarrow$
$cach\text{-}refinement\text{-}l\text{-}assn *a\ analyse\text{-}refinement\text{-}assn *a\ bool\text{-}assn$›
⟨*proof*⟩

declare *literal-redundant-wl-lookup-code.refine*[*sepref-fr-rules*]

sepref-definition *literal-redundant-wl-lookup-fast-code*
is ‹*uncurry5 isa-literal-redundant-wl-lookup*›
:: ‹$[\lambda(((((M, NU), D), cach), L), lbd).\ length\ NU \leq uint64\text{-}max]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k *_a$
$cach\text{-}refinement\text{-}l\text{-}assn^d *_a unat\text{-}lit\text{-}assn^k *_a lbd\text{-}assn^k \rightarrow$
$cach\text{-}refinement\text{-}l\text{-}assn *a\ analyse\text{-}refinement\text{-}fast\text{-}assn *a\ bool\text{-}assn$›
⟨*proof*⟩

declare *literal-redundant-wl-lookup-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *conflict-remove1-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-remove1*)›
:: ‹$[lookup\text{-}conflict\text{-}remove1\text{-}pre]_a\ unat\text{-}lit\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^d \rightarrow$
$lookup\text{-}clause\text{-}rel\text{-}assn$›
⟨*proof*⟩

declare *conflict-remove1-code.refine*[*sepref-fr-rules*]

find-theorems *delete-index-and-swap arl-assn*
sepref-definition *minimize-and-extract-highest-lookup-conflict-code*
is ‹*uncurry5* (*isa-minimize-and-extract-highest-lookup-conflict*)›
:: ‹$[\lambda(((((M, NU), D), cach), lbd), outl).\ True]_a$
$trail\text{-}pol\text{-}assn^k *_a arena\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^d *_a$
$cach\text{-}refinement\text{-}l\text{-}assn^d *_a lbd\text{-}assn^k *_a out\text{-}learned\text{-}assn^d \rightarrow$
$lookup\text{-}clause\text{-}rel\text{-}assn *a\ cach\text{-}refinement\text{-}l\text{-}assn *a\ out\text{-}learned\text{-}assn$›
⟨*proof*⟩

declare *minimize-and-extract-highest-lookup-conflict-code.refine*[*sepref-fr-rules*]

sepref-definition *minimize-and-extract-highest-lookup-conflict-fast-code*
is ‹*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*›
:: ‹$[\lambda(((((M,\ NU),\ D),\ cach),\ lbd),\ outl).\ length\ NU \le uint64\text{-}max]_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $*_a$
cach-refinement-l-assn^d $*_a$ *lbd-assn*^{k} $*_a$ *out-learned-assn*^{d} $\rightarrow$
lookup-clause-rel-assn $*a$ *cach-refinement-l-assn* $*a$ *out-learned-assn*›
⟨*proof*⟩

declare *minimize-and-extract-highest-lookup-conflict-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *isasat-lookup-merge-eq2-code*
is ‹*uncurry7 isasat-lookup-merge-eq2*›
:: ‹*unat-lit-assn*^{k} $*_a$ *trail-pol-assn*^{k} $*_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *conflict-option-rel-assn*^{d} $*_a$
uint32-nat-assn^k $*_a$ *lbd-assn*^{d} $*_a$ *out-learned-assn*^{d} $\rightarrow_a$
conflict-option-rel-assn $*a$ *uint32-nat-assn* $*a$ *lbd-assn* $*a$ *out-learned-assn*›
⟨*proof*⟩

sepref-definition *isasat-lookup-merge-eq2-fast-code*
is ‹*uncurry7 isasat-lookup-merge-eq2*›
:: ‹$[\lambda(((((((L,\ M),\ NU),\ \text{-}),\ \text{-}),\ \text{-}),\ \text{-}),\ \text{-}).\ length\ NU \le uint64\text{-}max]_a$
unat-lit-assn^k $*_a$ *trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$
conflict-option-rel-assn^d $*_a$ *uint32-nat-assn*^{k} $*_a$ *lbd-assn*^{d} $*_a$ *out-learned-assn*^{d} $\rightarrow$
conflict-option-rel-assn $*a$ *uint32-nat-assn* $*a$ *lbd-assn* $*a$ *out-learned-assn*›
⟨*proof*⟩

declare
isasat-lookup-merge-eq2-fast-code.refine[*sepref-fr-rules*]
isasat-lookup-merge-eq2-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Setup-SML*
imports *IsaSAT-Setup IsaSAT-Watch-List-SML IsaSAT-Lookup-Conflict-SML*
IsaSAT-Clauses-SML IsaSAT-Arena-SML LBD-SML Watched-Literals.IICF-Array-List32
begin

type-synonym *minimize-assn* = ‹*minimize-status array* × *uint32 array-list32*›
abbreviation *stats-assn* :: ‹*stats* ⇒ *stats* ⇒ *assn*› **where**
‹*stats-assn* ≡ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn*
$*a$ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn*›

abbreviation *ema-assn* :: ‹*ema* ⇒ *ema* ⇒ *assn*› **where**
‹*ema-assn* ≡ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn* $*a$ *uint64-assn*›

lemma *ema-get-value-hnr*[*sepref-fr-rules*]:
‹(*return o ema-get-value*, *RETURN o ema-get-value*) ∈ *ema-assn*^{k} $\rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

sepref-register *ema-bitshifting*

lemma *incr-propagation-hnr*[*sepref-fr-rules*]:
‹(*return o incr-propagation*, *RETURN o incr-propagation*) ∈ *stats-assn*^{d} $\rightarrow_a$ *stats-assn*›

⟨proof⟩

lemma *incr-conflict-hnr*[*sepref-fr-rules*]:
‹(*return o incr-conflict*, *RETURN o incr-conflict*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *incr-decision-hnr*[*sepref-fr-rules*]:
‹(*return o incr-decision*, *RETURN o incr-decision*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *incr-restart-hnr*[*sepref-fr-rules*]:
‹(*return o incr-restart*, *RETURN o incr-restart*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *incr-lrestart-hnr*[*sepref-fr-rules*]:
‹(*return o incr-lrestart*, *RETURN o incr-lrestart*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *incr-uset-hnr*[*sepref-fr-rules*]:
‹(*return o incr-uset*, *RETURN o incr-uset*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *incr-GC-hnr*[*sepref-fr-rules*]:
‹(*return o incr-GC*, *RETURN o incr-GC*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$›
⟨proof⟩

lemma *add-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo add-lbd*), *uncurry* (*RETURN oo add-lbd*)) ∈ $uint64\text{-}assn^k *_a stats\text{-}assn^d \rightarrow_a$ *stats-assn*›
⟨proof⟩

lemma *ema-bitshifting-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4294967296*), *uncurry0* (*RETURN ema-bitshifting*)) ∈
$unit\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
⟨proof⟩

lemma *ema-bitshifting-hnr2*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4294967296*), *uncurry0* (*RETURN ema-bitshifting*)) ∈
$unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨proof⟩

lemma (**in** −) *ema-update-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo ema-update-ref*), *uncurry* (*RETURN oo ema-update*)) ∈
$uint32\text{-}nat\text{-}assn^k *_a ema\text{-}assn^k \rightarrow_a ema\text{-}assn$›
⟨proof⟩

lemma *ema-reinit-hnr*[*sepref-fr-rules*]:
‹(*return o ema-reinit*, *RETURN o ema-reinit*) ∈ $ema\text{-}assn^k \rightarrow_a ema\text{-}assn$›
⟨proof⟩

lemma (**in** −) *ema-init-coeff-hnr*[*sepref-fr-rules*]:
‹(*return o ema-init*, *RETURN o ema-init*) ∈ $uint64\text{-}assn^k \rightarrow_a ema\text{-}assn$›
⟨proof⟩

abbreviation *restart-info-assn* **where**
‹*restart-info-assn* ≡ *uint64-assn* ∗*a* *uint64-assn*›

lemma *incr-conflict-count-since-last-restart-hnr*[*sepref-fr-rules*]:
‹(*return o incr-conflict-count-since-last-restart*, *RETURN o incr-conflict-count-since-last-restart*)
$\in$ *restart-info-assn*^{d} $\rightarrow_a$ *restart-info-assn*›
⟨*proof*⟩

lemma *restart-info-update-lvl-avg-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo restart-info-update-lvl-avg*),
uncurry (*RETURN oo restart-info-update-lvl-avg*))
$\in$ *uint32-assn*^{k} $*_a$ *restart-info-assn*^{d} $\rightarrow_a$ *restart-info-assn*›
⟨*proof*⟩

lemma *restart-info-init-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return restart-info-init*),
uncurry0 (*RETURN restart-info-init*))
$\in$ *unit-assn*^{k} $\rightarrow_a$ *restart-info-assn*›
⟨*proof*⟩

lemma *restart-info-restart-done-hnr*[*sepref-fr-rules*]:
‹(*return o restart-info-restart-done*, *RETURN o restart-info-restart-done*) $\in$
restart-info-assn^d $\rightarrow_a$ *restart-info-assn*›
⟨*proof*⟩

type-synonym *vmtf-remove-assn* = ‹*vmtf-assn* $\times$ (*uint32 array-list32* $\times$ *bool array*)›

abbreviation (**in** $-$)*vmtf-node-assn* **where**
‹*vmtf-node-assn* $\equiv$ *pure vmtf-node-rel*›

abbreviation *vmtf-conc* **where**
‹*vmtf-conc* $\equiv$ (*array-assn vmtf-node-assn* $*a$ *uint64-nat-assn* $*a$ *uint32-nat-assn* $*a$ *uint32-nat-assn*
$*a$ *option-assn uint32-nat-assn*)›

abbreviation *atoms-hash-assn* :: ‹*bool list* $\Rightarrow$ *bool array* $\Rightarrow$ *assn*› **where**
‹*atoms-hash-assn* $\equiv$ *array-assn bool-assn*›

abbreviation *distinct-atoms-assn* **where**
‹*distinct-atoms-assn* $\equiv$ *arl32-assn uint32-nat-assn* $*a$ *atoms-hash-assn*›

abbreviation *vmtf-remove-conc*
:: ‹*isa-vmtf-remove-int* $\Rightarrow$ *vmtf-remove-assn* $\Rightarrow$ *assn*›
where
‹*vmtf-remove-conc* $\equiv$ *vmtf-conc* $*a$ *distinct-atoms-assn*›

Options

abbreviation *opts-assn*
:: ‹*opts* $\Rightarrow$ *opts* $\Rightarrow$ *assn*›
where
‹*opts-assn* $\equiv$ *bool-assn* $*a$ *bool-assn* $*a$ *bool-assn*›

lemma *opts-restart-hnr*[*sepref-fr-rules*]:
‹(*return o opts-restart*, *RETURN o opts-restart*) $\in$ *opts-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *opts-reduce-hnr*[*sepref-fr-rules*]:
‹(*return o opts-reduce*, *RETURN o opts-reduce*) $\in$ *opts-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *opts-unbounded-mode-hnr*[*sepref-fr-rules*]:
‹(*return o opts-unbounded-mode*, *RETURN o opts-unbounded-mode*) $\in$ *opts-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

definition *convert-wlists-to-nat* **where**
‹*convert-wlists-to-nat* = *op-map* (*map* (λ(*n*, *L*, *b*). (*nat-of-uint64-conv n*, *L*, *b*))) []›

lemma *convert-wlists-to-nat-alt-def*:
‹*convert-wlists-to-nat* = *op-map id* []›
⟨*proof*⟩

lemma *convert-single-wl-to-nat-conv-alt-def*:
‹*convert-single-wl-to-nat-conv zs i xs i* = *xs*[*i* := *map* (λ(*i*, *y*, *y'*). (*nat-of-uint64-conv i*, *y*, *y'*)) (*zs* ! *i*)]›
⟨*proof*⟩

lemma *convert-wlists-to-nat-convert-wlists-to-nat-conv*:
‹(*convert-wlists-to-nat*, *RETURN o convert-wlists-to-nat-conv*) $\in$
⟨⟨*nat-rel* $\times_r$ *Id* $\times_r$ *Id*⟩*list-rel*⟩*list-rel* $\rightarrow_f$
⟨⟨⟨*nat-rel* $\times_r$ *Id* $\times_r$ *Id*⟩*list-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *convert-wlists-to-nat-alt-def2*:
‹*convert-wlists-to-nat xs* = *do* {
let n = *length xs*;
let zs = *init-lrl n*;
(*uu*, *zs*) $\leftarrow$
WHILE$_T$$^{\lambda(i,\ zs).}$ *i* $\leq$ *length xs* $\wedge$ *take i zs* = *map* (*map* (λ(*n*, *y*, *y'*). (*nat-of-uint64-c*
(λ(*i*, *zs*). *i* < *length zs*)
(λ(*i*, *zs*). *do* {
ASSERT (*i* < *length zs*);
RETURN
(*i* + *1*, *convert-single-wl-to-nat-conv xs i zs i*)
})
(*0*, *zs*);
RETURN zs
}›
⟨*proof*⟩

sepref-register *init-lrl*

abbreviation (**in** −) *watchers-assn* **where**
‹*watchers-assn* $\equiv$ *arl-assn* (*watcher-assn*)›

abbreviation (**in** −) *watchlist-assn* **where**
‹*watchlist-assn* $\equiv$ *arrayO-assn watchers-assn*›

abbreviation (**in** −) *watchers-fast-assn* **where**
‹*watchers-fast-assn* $\equiv$ *arl64-assn* (*watcher-fast-assn*)›

abbreviation (**in** −) *watchlist-fast-assn* **where**
‹*watchlist-fast-assn* $\equiv$ *arrayO-assn watchers-fast-assn*›

sepref-definition *convert-single-wl-to-nat-code*
is ‹*uncurry3 convert-single-wl-to-nat*›
:: ‹$[\lambda(((W, i), W'), j).\ i < \mathit{length}\ W \wedge j < \mathit{length}\ W']_a$
$(\mathit{watchlist\text{-}fast\text{-}assn})^k *_a \mathit{nat\text{-}assn}^k *_a$
$(\mathit{watchlist\text{-}assn})^d *_a \mathit{nat\text{-}assn}^k \rightarrow$
watchlist-assn›
⟨*proof*⟩

sepref-register *convert-single-wl-to-nat-conv*
lemma *convert-single-wl-to-nat-conv-hnr*[*sepref-fr-rules*]:
‹(*uncurry3 convert-single-wl-to-nat-code*,
uncurry3 (*RETURN* ∘∘∘ *convert-single-wl-to-nat-conv*))
$\in [\lambda(((a, b), ba), bb).\ b < \mathit{length}\ a \wedge bb < \mathit{length}\ ba \wedge ba\ !\ bb = []]_a$
$(\mathit{watchlist\text{-}fast\text{-}assn})^k *_a \mathit{nat\text{-}assn}^k *_a$
$(\mathit{watchlist\text{-}assn})^d *_a \mathit{nat\text{-}assn}^k \rightarrow$
watchlist-assn›
⟨*proof*⟩

sepref-definition *convert-wlists-to-nat-code*
is ‹*convert-wlists-to-nat*›
:: ‹$\mathit{watchlist\text{-}fast\text{-}assn}^d \rightarrow_a \mathit{watchlist\text{-}assn}$›
⟨*proof*⟩

lemma *convert-wlists-to-nat-conv-hnr*[*sepref-fr-rules*]:
‹(*convert-wlists-to-nat-code*, *RETURN* ∘ *convert-wlists-to-nat-conv*)
$\in (\mathit{watchlist\text{-}fast\text{-}assn})^d \rightarrow_a \mathit{watchlist\text{-}assn}$›
⟨*proof*⟩

abbreviation *vdom-assn* :: ‹*vdom* ⇒ *nat array-list* ⇒ *assn*› **where**
‹*vdom-assn* ≡ *arl-assn nat-assn*›

abbreviation *vdom-fast-assn* :: ‹*vdom* ⇒ *uint64 array-list64* ⇒ *assn*› **where**
‹*vdom-fast-assn* ≡ *arl64-assn uint64-nat-assn*›

type-synonym *vdom-assn* = ‹*nat array-list*›
type-synonym *vdom-fast-assn* = ‹*uint64 array-list64*›

type-synonym *isasat-clauses-assn* = ‹*uint32 array-list*›
type-synonym *isasat-clauses-fast-assn* = ‹*uint32 array-list64*›

abbreviation *phase-saver-conc* **where**
‹*phase-saver-conc* ≡ *array-assn bool-assn*›

type-synonym *twl-st-wll-trail* =
‹*trail-pol-assn* × *isasat-clauses-assn* × *option-lookup-clause-assn* ×
uint32 × *watched-wl* × *vmtf-remove-assn* × *phase-saver-assn* ×
uint32 × *minimize-assn* × *lbd-assn* × *out-learned-assn* × *stats* × *ema* × *ema* × *restart-info* ×
vdom-assn × *vdom-assn* × *nat* × *opts* × *isasat-clauses-assn*›

type-synonym *twl-st-wll-trail-fast* =
‹*trail-pol-fast-assn* × *isasat-clauses-fast-assn* × *option-lookup-clause-assn* ×
uint32 × *watched-wl-uint32* × *vmtf-remove-assn* × *phase-saver-assn* ×
uint32 × *minimize-assn* × *lbd-assn* × *out-learned-assn* × *stats* × *ema* × *ema* × *restart-info* ×
vdom-fast-assn × *vdom-fast-assn* × *uint64* × *opts* × *isasat-clauses-fast-assn*›

definition *isasat-unbounded-assn* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wll-trail* ⇒ *assn*⟩ **where**
⟨*isasat-unbounded-assn* =
  trail-pol-assn ∗a *arena-assn* ∗a
  isasat-conflict-assn ∗a
  uint32-nat-assn ∗a
  watchlist-assn ∗a
  vmtf-remove-conc ∗a *phase-saver-conc* ∗a
  uint32-nat-assn ∗a
  cach-refinement-l-assn ∗a
  lbd-assn ∗a
  out-learned-assn ∗a
  stats-assn ∗a
  ema-assn ∗a
  ema-assn ∗a
  restart-info-assn ∗a
  vdom-assn ∗a
  vdom-assn ∗a
  nat-assn ∗a
  opts-assn ∗a *arena-assn*⟩

definition *isasat-bounded-assn* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wll-trail-fast* ⇒ *assn*⟩ **where**
⟨*isasat-bounded-assn* =
  trail-pol-fast-assn ∗a *arena-fast-assn* ∗a
  isasat-conflict-assn ∗a
  uint32-nat-assn ∗a
  watchlist-fast-assn ∗a
  vmtf-remove-conc ∗a *phase-saver-conc* ∗a
  uint32-nat-assn ∗a
  cach-refinement-l-assn ∗a
  lbd-assn ∗a
  out-learned-assn ∗a
  stats-assn ∗a
  ema-assn ∗a
  ema-assn ∗a
  restart-info-assn ∗a
  vdom-fast-assn ∗a
  vdom-fast-assn ∗a
  uint64-nat-assn ∗a
  opts-assn ∗a *arena-fast-assn*⟩

sepref-definition *isasat-fast-slow-code*
  is ⟨*isasat-fast-slow*⟩
  :: ⟨$[\lambda S.\ length(get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max \wedge$
      $length\ (get\text{-}old\text{-}arena\ S) \leq uint64\text{-}max]_a$
    $isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}unbounded\text{-}assn$⟩
  ⟨*proof*⟩

declare *isasat-fast-slow-code.refine*[*sepref-fr-rules*]

Lift Operations to State

sepref-definition *get-conflict-wl-is-None-code*
  is ⟨*RETURN o get-conflict-wl-is-None-heur*⟩
  :: ⟨$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
  ⟨*proof*⟩

declare *get-conflict-wl-is-None-code.refine*[*sepref-fr-rules*]

sepref-definition *get-conflict-wl-is-None-fast-code*
is ‹*RETURN o get-conflict-wl-is-None-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *get-conflict-wl-is-None-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-count-decided-st-code*
is ‹*RETURN o isa-count-decided-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *isa-count-decided-st-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-count-decided-st-fast-code*
is ‹*RETURN o isa-count-decided-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *isa-count-decided-st-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *polarity-st-heur-pol*
is ‹*uncurry* (*RETURN oo polarity-st-heur*)›
:: ‹$[polarity\text{-}st\text{-}heur\text{-}pre]_a\ isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$›
⟨*proof*⟩

declare *polarity-st-heur-pol.refine*[*sepref-fr-rules*]

sepref-definition *polarity-st-heur-pol-fast*
is ‹*uncurry* (*RETURN oo polarity-st-heur*)›
:: ‹$[polarity\text{-}st\text{-}heur\text{-}pre]_a\ isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$›
⟨*proof*⟩

declare *polarity-st-heur-pol-fast.refine*[*sepref-fr-rules*]

0.1.15 More theorems

lemma *count-decided-st-heur*[*sepref-fr-rules*]:
‹(*return o count-decided-st-heur*, *RETURN o count-decided-st-heur*) ∈
$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
‹(*return o count-decided-st-heur*, *RETURN o count-decided-st-heur*) ∈
$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *access-lit-in-clauses-heur-code*
is ‹*uncurry2* (*RETURN ooo access-lit-in-clauses-heur*)›
:: ‹$[access\text{-}lit\text{-}in\text{-}clauses\text{-}heur\text{-}pre]_a$
$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *access-lit-in-clauses-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *access-lit-in-clauses-heur-fast-code*

is ‹*uncurry2* (*RETURN ooo access-lit-in-clauses-heur*)›
:: ‹[λ((*S*, *i*), *j*). *access-lit-in-clauses-heur-pre* ((*S*, *i*), *j*) $\wedge$
length (*get-clauses-wl-heur S*) $\leq$ *uint64-max*]$_a$
isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *unat-lit-assn*›
⟨*proof*⟩

declare *access-lit-in-clauses-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *rewatch-heur*
sepref-definition *rewatch-heur-code*
is ‹*uncurry2* (*rewatch-heur*)›
:: ‹*vdom-assn*^{k} $*_a$ *arena-assn*^{k} $*_a$ *watchlist-assn*^{d} $\rightarrow_a$ *watchlist-assn*›
⟨*proof*⟩

declare *rewatch-heur-code.refine*[*sepref-fr-rules*]
find-theorems *nfoldli WHILET*
sepref-definition *rewatch-heur-fast-code*
is ‹*uncurry2* (*rewatch-heur*)›
:: ‹[λ((*vdom*, *arena*), *W*). ($\forall$ *x* $\in$ *set vdom*. *x* $\leq$ *uint64-max*) $\wedge$ *length arena* $\leq$ *uint64-max* $\wedge$ *length vdom* $\leq$ *uint64-max*]$_a$
vdom-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *watchlist-fast-assn*^{d} $\rightarrow$ *watchlist-fast-assn*›
⟨*proof*⟩

sepref-register *append-ll*

declare *rewatch-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *rewatch-heur-st-code*
is ‹(*rewatch-heur-st*)›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-bounded-assn^d $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *rewatch-heur-st-code.refine*[*sepref-fr-rules*]
rewatch-heur-st-fast-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Inner-Propagation*
imports *IsaSAT-Setup*
IsaSAT-Clauses
begin

declare *all-atms-def*[*symmetric*,*simp*]

0.1.16 Propagations Step

lemma *unit-prop-body-wl-D-invD*:
fixes *S*
defines ‹$\mathcal{A}$ $\equiv$ *all-atms-st S*›
assumes ‹*unit-prop-body-wl-D-inv S j w L*›
shows

$\langle w < length\ (watched\text{-}by\ S\ L)\rangle$ **and**
$\langle j \leq w\rangle$ **and**
$\langle fst\ (snd\ (watched\text{-}by\text{-}app\ S\ L\ w)) \in\#\ \mathcal{L}_{all}\ \mathcal{A}\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S)\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \neq []\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow Suc\ 0 < length\ (get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w))\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w)\ !\ 0 \in\#\ \mathcal{L}_{all}\ \mathcal{A}\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w)\ !\ Suc\ 0 \in\#\ \mathcal{L}_{all}\ \mathcal{A}\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow L \in\#\ \mathcal{L}_{all}\ \mathcal{A}\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow fst\ (watched\text{-}by\text{-}app\ S\ L\ w) > 0\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow literals\text{-}are\text{-}\mathcal{L}_{in}\ \mathcal{A}\ S\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow get\text{-}conflict\text{-}wl\ S = None\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w)))\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow distinct\ (get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w))\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ \mathcal{A}\ (get\text{-}trail\text{-}wl\ S)\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow isasat\text{-}input\text{-}bounded\ \mathcal{A} \Longrightarrow$
$length\ (get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w)) \leq uint64\text{-}max\rangle$ **and**
$\langle fst\ (watched\text{-}by\text{-}app\ S\ L\ w) \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S) \Longrightarrow$
$L \in set\ (watched\text{-}l\ (get\text{-}clauses\text{-}wl\ S \propto fst\ (watched\text{-}by\text{-}app\ S\ L\ w)))\rangle$
$\langle proof\rangle$

definition (**in** $-$) *find-unwatched-wl-st* :: $\langle nat\ twl\text{-}st\text{-}wl \Rightarrow nat \Rightarrow nat\ option\ nres\rangle$ **where**
$\langle find\text{-}unwatched\text{-}wl\text{-}st = (\lambda(M,\ N,\ D,\ NE,\ UE,\ Q,\ W)\ i.\ do\ \{$
$find\text{-}unwatched\text{-}l\ M\ (N \propto i)$
$\})\rangle$

lemma *find-unwatched-l-find-unwatched-wl-s*:
$\langle find\text{-}unwatched\text{-}l\ (get\text{-}trail\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S \propto C) = find\text{-}unwatched\text{-}wl\text{-}st\ S\ C\rangle$
$\langle proof\rangle$

definition *find-non-false-literal-between* **where**
$\langle find\text{-}non\text{-}false\text{-}literal\text{-}between\ M\ a\ b\ C =$
$find\text{-}in\text{-}list\text{-}between\ (\lambda L.\ polarity\ M\ L \neq Some\ False)\ a\ b\ C\rangle$

definition *isa-find-unwatched-between*
:: $\langle\text{-} \Rightarrow trail\text{-}pol \Rightarrow arena \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow (nat\ option)\ nres\rangle$ **where**
$\langle isa\text{-}find\text{-}unwatched\text{-}between\ P\ M'\ NU\ a\ b\ C = do\ \{$
$ASSERT(C+a \leq length\ NU);$
$ASSERT(C+b \leq length\ NU);$
$(x,\ \text{-}) \leftarrow WHILE_T{}^{\lambda(found,\ i).\ True}$
$(\lambda(found,\ i).\ found = None \wedge i < C + b)$
$(\lambda(\text{-},\ i).\ do\ \{$
$ASSERT(i < C + nat\text{-}of\text{-}uint64\text{-}conv\ (arena\text{-}length\ NU\ C));$
$ASSERT(i \geq C);$
$ASSERT(i < C + b);$
$ASSERT(arena\text{-}lit\text{-}pre\ NU\ i);$

$ASSERT(polarity\text{-}pol\text{-}pre\ M'\ (arena\text{-}lit\ NU\ i))$;
$if\ P\ (arena\text{-}lit\ NU\ i)\ then\ RETURN\ (Some\ (i - C),\ i)\ else\ RETURN\ (None,\ i{+}1)$
})
$(None,\ C{+}a)$;
$RETURN\ x$
}
›

lemma *isa-find-unwatched-between-find-in-list-between-spec*:
assumes ‹$a \leq length\ (N \propto C)$› **and** ‹$b \leq length\ (N \propto C)$› **and** ‹$a \leq b$› **and**
‹*valid-arena arena N vdom*› **and** ‹$C \in\#\ dom\text{-}m\ N$› **and** *eq*: ‹$a' = a$› ‹$b' = b$› ‹$C' = C$› **and**
‹$\bigwedge L.\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow P'\ L = P\ L$› **and**
$M'M$: ‹$(M',\ M) \in trail\text{-}pol\ \mathcal{A}$›
assumes *lits*: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (N \propto C))$›
shows
‹$isa\text{-}find\text{-}unwatched\text{-}between\ P'\ M'\ arena\ a'\ b'\ C' \leq \Downarrow Id\ (find\text{-}in\text{-}list\text{-}between\ P\ a\ b\ (N \propto C))$›
⟨*proof*⟩

definition *isa-find-non-false-literal-between* **where**
‹*isa-find-non-false-literal-between M arena a b C* =
isa-find-unwatched-between ($\lambda L.\ polarity\text{-}pol\ M\ L \neq Some\ False$) *M arena a b C*›

definition *find-unwatched*
:: ‹($nat\ literal \Rightarrow bool) \Rightarrow nat\ clause\text{-}l \Rightarrow (nat\ option)\ nres$› **where**
‹*find-unwatched M C* = *do* {
$b \leftarrow SPEC(\lambda b{::}bool.\ True)$; — non-deterministic between full iteration (used in minisat), or starting in the middle (use in cadical)
if b then find-in-list-between M 2 (length C) C
else do {
$pos \leftarrow SPEC\ (\lambda i.\ i \leq length\ C \wedge i \geq 2)$;
$n \leftarrow find\text{-}in\text{-}list\text{-}between\ M\ pos\ (length\ C)\ C$;
if n = None then find-in-list-between M 2 pos C
else RETURN n
}
}
›

definition *find-unwatched-wl-st-heur-pre* **where**
‹*find-unwatched-wl-st-heur-pre* =
$(\lambda(S,\ i).\ arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ i)$›

definition *find-unwatched-wl-st'*
:: ‹$nat\ twl\text{-}st\text{-}wl \Rightarrow nat \Rightarrow nat\ option\ nres$› **where**
‹$find\text{-}unwatched\text{-}wl\text{-}st' = (\lambda(M,\ N,\ D,\ Q,\ W,\ vm,\ \varphi)\ i.\ do$ {
find-unwatched ($\lambda L.\ polarity\ M\ L \neq Some\ False$) $(N \propto i)$
})›

definition *isa-find-unwatched*
:: ‹($nat\ literal \Rightarrow bool) \Rightarrow trail\text{-}pol \Rightarrow arena \Rightarrow nat \Rightarrow (nat\ option)\ nres$›
where
‹*isa-find-unwatched P M' arena C* = *do* {
let l = nat-of-uint64-conv (arena-length arena C);

$b \leftarrow RETURN(arena\text{-}length\ arena\ C \leq MAX\text{-}LENGTH\text{-}SHORT\text{-}CLAUSE)$;
if b then isa-find-unwatched-between P M′ arena 2 l C
else do {
ASSERT(*get-saved-pos-pre arena C*);
pos ← *RETURN* (*nat-of-uint64-conv* (*arena-pos arena C*));
n ← *isa-find-unwatched-between P M′ arena pos l C*;
if n = *None then isa-find-unwatched-between P M′ arena 2 pos C*
else RETURN n
}
}
›

lemma *isa-find-unwatched-find-unwatched*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* ($N \propto C$))› **and**
ge2: ‹2 ≤ *length* ($N \propto C$)› **and**
C: ‹*C* ∈# *dom-m N*› **and**
M′M: ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$›
shows ‹*isa-find-unwatched P M′ arena C* ≤ ⇓ *Id* (*find-unwatched P* ($N \propto C$))›
⟨*proof*⟩

definition *isa-find-unwatched-wl-st-heur*
:: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*isa-find-unwatched-wl-st-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ) *i*. *do* {
isa-find-unwatched (λ*L*. *polarity-pol M L* ≠ *Some False*) *M N i*
})›

lemma *find-unwatched*:
assumes *n-d*: ‹*no-dup M*› **and** ‹*length C* ≥ 2› **and** ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)›
shows ‹*find-unwatched* (λ*L*. *polarity M L* ≠ *Some False*) *C* ≤ ⇓ *Id* (*find-unwatched-l M C*)›
⟨*proof*⟩

definition *find-unwatched-wl-st-pre* **where**
‹*find-unwatched-wl-st-pre* = (λ(*S*, *i*).
i ∈# *dom-m* (*get-clauses-wl S*) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ∧ 2 ≤ *length* (*get-clauses-wl S* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* ∝ *i*))
)›

theorem *find-unwatched-wl-st-heur-find-unwatched-wl-s*:
‹(*uncurry isa-find-unwatched-wl-st-heur*, *uncurry find-unwatched-wl-st′*)
∈ [*find-unwatched-wl-st-pre*]$_f$
twl-st-heur $\times_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *isa-save-pos* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*isa-save-pos C i* = (λ(*M*, *N*, *oth*). *do* {
ASSERT(*arena-is-valid-clause-idx N C*);
if arena-length N C > *MAX-LENGTH-SHORT-CLAUSE then do* {
ASSERT(*isa-update-pos-pre* ((*C*, *i*), *N*));
RETURN (*M*, *arena-update-pos C i N*, *oth*)
} *else RETURN* (*M*, *N*, *oth*)
})

›

lemma *isa-save-pos-is-Id*:
assumes
‹$(S, T) \in$ *twl-st-heur*›
‹$C \in\#$ *dom-m* (*get-clauses-wl* T)› **and**
‹*is-long-clause* (*get-clauses-wl* $T \propto C$)› **and**
‹$i \leq$ *length* (*get-clauses-wl* $T \propto C$)› **and**
‹$i \geq 2$›
shows ‹*isa-save-pos* C i $S \leq \Downarrow$ *twl-st-heur* (*RETURN* T)›
⟨*proof*⟩

lemmas *unit-prop-body-wl-D-invD'* =
unit-prop-body-wl-D-invD[*of* ‹(M, N, D, NE, UE, WS, Q)› **for** M N D NE UE WS Q,
unfolded watched-by-app-def,
simplified] *unit-prop-body-wl-D-invD*(7)

definition *set-conflict-wl'* :: ‹*nat* ⇒ *'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*set-conflict-wl'* =
(λC (M, N, D, NE, UE, Q, W). (M, N, *Some* (*mset* ($N \propto C$)), NE, UE, {#}, W))›

lemma *set-conflict-wl'-alt-def*:
‹*set-conflict-wl'* i S = *set-conflict-wl* (*get-clauses-wl* $S \propto i$) S›
⟨*proof*⟩

definition *set-conflict-wl-heur-pre* **where**
‹*set-conflict-wl-heur-pre* =
(λ(C, S). *True*)›

definition *set-conflict-wl-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*set-conflict-wl-heur* = (λC (M, N, D, Q, W, *vmtf*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*). *do* {
let n = *zero-uint32-nat*;
ASSERT(*curry6 isa-set-lookup-conflict-aa-pre* M N C D n *lbd outl*);
(D, *clvls*, *lbd*, *outl*) ← *isa-set-lookup-conflict-aa* M N C D n *lbd outl*;
ASSERT(*isa-length-trail-pre* M);
ASSERT(*arena-act-pre* N C);
RETURN (M, *arena-incr-act* N C, D, *isa-length-trail* M, W, *vmtf*, φ, *clvls*, *cach*, *lbd*, *outl*,
incr-conflict stats, *fema*, *sema*)})›

definition *update-clause-wl-code-pre* **where**
‹*update-clause-wl-code-pre* = (λ(((((((L, C), b), j), w), i), f), S).
arena-is-valid-clause-idx-and-access (*get-clauses-wl-heur* S) C f ∧
nat-of-lit L < *length* (*get-watched-wl-heur* S) ∧
nat-of-lit (*arena-lit* (*get-clauses-wl-heur* S) ($C + f$)) < *length* (*get-watched-wl-heur* S) ∧
w < *length* (*get-watched-wl-heur* S ! *nat-of-lit* L) ∧
$j \leq w$)›

definition *update-clause-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *bool* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒
(*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where

‹$update\text{-}clause\text{-}wl\text{-}heur = (\lambda(L::nat\ literal)\ C\ b\ j\ w\ i\ f\ (M,\ N,\ D,\ Q,\ W,\ vm).\ do\ \{$
$ASSERT(arena\text{-}lit\text{-}pre\ N\ (C{+}f));$
$let\ K' = arena\text{-}lit\ N\ (C + f);$
$ASSERT(swap\text{-}lits\text{-}pre\ C\ i\ f\ N);$
$ASSERT(w < length\ N);$
$let\ N' = swap\text{-}lits\ C\ i\ f\ N;$
$ASSERT(length\ (W\ !\ nat\text{-}of\text{-}lit\ K') < length\ N);$
$let\ W = W[nat\text{-}of\text{-}lit\ K' := W\ !\ (nat\text{-}of\text{-}lit\ K')\ @\ [to\text{-}watcher\ C\ L\ b]];$
$RETURN\ (j,\ w{+}1,\ (M,\ N',\ D,\ Q,\ W,\ vm))$
$\})$›

definition *update-clause-wl-pre* **where**
‹$update\text{-}clause\text{-}wl\text{-}pre\ K\ r = (\lambda(((((((L,\ C),\ b),\ j),\ w),\ i),\ f),\ S).\ C \in\#\ dom\text{-}m(get\text{-}clauses\text{-}wl\ S)\ \wedge$
$L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge i < length\ (get\text{-}clauses\text{-}wl\ S \propto C) \wedge$
$f < length\ (get\text{-}clauses\text{-}wl\ S \propto C) \wedge$
$L \neq get\text{-}clauses\text{-}wl\ S \propto C\ !\ f \wedge$
$length\ (watched\text{-}by\ S\ (get\text{-}clauses\text{-}wl\ S \propto C\ !\ f)) < r \wedge$
$w < r \wedge$
$L = K)$›

lemma *update-clause-wl-pre-alt-def*:
‹$update\text{-}clause\text{-}wl\text{-}pre\ K\ r = (\lambda(((((((L,\ C),\ b),\ j),\ w),\ i),\ f),\ S).\ C \in\#\ dom\text{-}m(get\text{-}clauses\text{-}wl\ S)\ \wedge$
$L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge i < length\ (get\text{-}clauses\text{-}wl\ S \propto C) \wedge$
$f < length\ (get\text{-}clauses\text{-}wl\ S \propto C) \wedge$
$L \neq get\text{-}clauses\text{-}wl\ S \propto C\ !\ f \wedge$
$length\ (watched\text{-}by\ S\ (get\text{-}clauses\text{-}wl\ S \propto C\ !\ f)) < r \wedge$
$w < r \wedge$
$get\text{-}clauses\text{-}wl\ S \propto C\ !\ f \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge$
$L = K)$›
⟨*proof*⟩

lemma *arena-lit-pre*:
‹$valid\text{-}arena\ NU\ N\ vdom \Longrightarrow C \in\#\ dom\text{-}m\ N \Longrightarrow i < length\ (N \propto C) \Longrightarrow arena\text{-}lit\text{-}pre\ NU\ (C + i)$›
⟨*proof*⟩

lemma *all-atms-swap*[*simp*]:
‹$C \in\#\ dom\text{-}m\ N \Longrightarrow i < length\ (N \propto C) \Longrightarrow j < length\ (N \propto C) \Longrightarrow$
$all\text{-}atms\ (N(C \hookrightarrow swap\ (N \propto C)\ i\ j)) = all\text{-}atms\ N$›
⟨*proof*⟩

lemma *update-clause-wl-heur-update-clause-wl*:
‹$(uncurry7\ update\text{-}clause\text{-}wl\text{-}heur,\ uncurry7\ (update\text{-}clause\text{-}wl)) \in$
$[update\text{-}clause\text{-}wl\text{-}pre\ K\ r]_f$
$Id \times_f nat\text{-}rel \times_f bool\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K \rightarrow$
$\langle nat\text{-}rel \times_r nat\text{-}rel \times_r twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\rangle nres\text{-}rel$›
⟨*proof*⟩

definition (**in** −) *access-lit-in-clauses* **where**
‹$access\text{-}lit\text{-}in\text{-}clauses\ S\ i\ j = (get\text{-}clauses\text{-}wl\ S) \propto i\ !\ j$›

lemma *twl-st-heur-get-clauses-access-lit*[*simp*]:
‹$(S,\ T) \in twl\text{-}st\text{-}heur \Longrightarrow C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T) \Longrightarrow$
$i < length\ (get\text{-}clauses\text{-}wl\ T \propto C) \Longrightarrow$
$get\text{-}clauses\text{-}wl\ T \propto C\ !\ i = access\text{-}lit\text{-}in\text{-}clauses\text{-}heur\ S\ C\ i$›
for $S\ T\ C\ i$

⟨proof⟩

lemma
find-unwatched-not-tauto:
‹¬*tautology*(*mset* (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L C*)))›
(**is** *?tauto* **is** ‹¬*tautology ?D*› **is** ‹¬*tautology* (*mset ?C*)›)
if
find-unw: ‹*unit-prop-body-wl-D-find-unwatched-inv None* (*fst* (*watched-by-app S L C*)) *S*› **and**
inv: ‹*unit-prop-body-wl-D-inv S j C L*› **and**
val: ‹*polarity-st S* (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L C*) !
(*1* − (*if access-lit-in-clauses S* (*fst* (*watched-by-app S L C*)) *0* = *L then 0 else 1*))) =
Some False›
(**is** ‹*polarity-st* - (- ∝ - ! *?i*) = *Some False*›) **and**
dom: ‹*fst* (*watched-by S L* ! *C*) ∈# *dom-m* (*get-clauses-wl S*)›
for *S C xj L*
⟨proof⟩

definition *propagate-lit-wl-heur-pre* **where**
‹*propagate-lit-wl-heur-pre* =
(λ(((*L*, *C*), *i*), *S*). *i* ≤ *1* ∧ *C* ≠ *DECISION-REASON*)›

definition *propagate-lit-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-heur* = (λ*L′ C i* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fema, *sema*). *do* {
ASSERT(*swap-lits-pre C 0* (*fast-minus 1 i*) *N*);
let N′ = *swap-lits C 0* (*fast-minus 1 i*) *N*;
ASSERT(*atm-of L′* < *length* φ);
ASSERT(*cons-trail-Propagated-tr-pre* ((*L′*, *C*), *M*));
let stats = *incr-propagation* (*if count-decided-pol M* = *0 then incr-uset stats else stats*);
RETURN (*cons-trail-Propagated-tr L′ C M*, *N′*, *D*, *Q*, *W*, *vm*, *save-phase L′* φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*)
})›

definition *propagate-lit-wl-pre* **where**
‹*propagate-lit-wl-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧
C ∈# *dom-m* (*get-clauses-wl S*) ∧ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
1 − *i* < *length* (*get-clauses-wl S* ∝ *C*) ∧
0 < *length* (*get-clauses-wl S* ∝ *C*))›

lemma *isa-vmtf-consD*:
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf* 𝒜 *M*›
shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf* 𝒜 (*L* # *M*)›
⟨proof⟩

lemma *propagate-lit-wl-heur-propagate-lit-wl*:
‹(*uncurry3 propagate-lit-wl-heur*, *uncurry3* (*RETURN oooo propagate-lit-wl*)) ∈
[*propagate-lit-wl-pre*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* 𝒟 *r s K* → ⟨*twl-st-heur-up″* 𝒟 *r s K*⟩*nres-rel*›
⟨proof⟩

definition *propagate-lit-wl-bin-pre* **where**

‹*propagate-lit-wl-bin-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* $\wedge$ *get-conflict-wl S* = *None* $\wedge$
C $\in\#$ *dom-m* (*get-clauses-wl S*) $\wedge$ *L* $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*))›

definition *propagate-lit-wl-bin-heur*
:: ‹*nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-bin-heur* = ($\lambda L'$ *C* - (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fema, *sema*). *do* {
ASSERT(*atm-of* L' < *length* φ);
let stats = *incr-propagation* (*if count-decided-pol M* = *0 then incr-uset stats else stats*);
ASSERT(*cons-trail-Propagated-tr-pre* ((L', *C*), *M*));
RETURN (*cons-trail-Propagated-tr* L' *C M*, *N*, *D*, *Q*, *W*, *vm*, *save-phase* L' φ, *clvls*, *cach*, *lbd*,
outl,
stats, *fema*, *sema*)
})›

lemma *propagate-lit-wl-bin-heur-propagate-lit-wl-bin*:
‹(*uncurry3 propagate-lit-wl-bin-heur*, *uncurry3* (*RETURN oooo propagate-lit-wl-bin*)) $\in$
[*propagate-lit-wl-bin-pre*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\rightarrow$ ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩*nres-rel*›
⟨*proof*⟩

lemma *undefined-lit-polarity-st-iff*:
‹*undefined-lit* (*get-trail-wl S*) *L* $\longleftrightarrow$
polarity-st S L $\neq$ *Some True* $\wedge$ *polarity-st S L* $\neq$ *Some False*›
⟨*proof*⟩

lemma *find-unwatched-le-length*:
‹*xj* < *length* (*get-clauses-wl S* $\propto$ *fst* (*watched-by-app S L C*))›
if
find-unw: ‹*RETURN* (*Some xj*) $\leq$
IsaSAT-Inner-Propagation.find-unwatched-wl-st S (*fst* (*watched-by-app S L C*))›
for *S L C xj*
⟨*proof*⟩

lemma *find-unwatched-in-D_0*:
‹*get-clauses-wl S* $\propto$ *fst* (*watched-by-app S L C*) ! *xj* $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*)›
if
find-unw: ‹*RETURN* (*Some xj*) $\leq$ *IsaSAT-Inner-Propagation.find-unwatched-wl-st S* (*fst* (*watched-by-app*
S L C))› **and**
inv: ‹*unit-prop-body-wl-D-inv S j C L*› **and**
dom: ‹*fst* (*watched-by-app S L C*) $\in\#$ *dom-m* (*get-clauses-wl S*)›
for *S C xj L*
⟨*proof*⟩

definition *unit-prop-body-wl-heur-inv* **where**
‹*unit-prop-body-wl-heur-inv S j w L* $\longleftrightarrow$
($\exists S'$. (*S*, S') $\in$ *twl-st-heur* $\wedge$ *unit-prop-body-wl-D-inv* S' *j w L*)›

definition *unit-prop-body-wl-D-find-unwatched-heur-inv* **where**
‹*unit-prop-body-wl-D-find-unwatched-heur-inv f C S* $\longleftrightarrow$
($\exists S'$. (*S*, S') $\in$ *twl-st-heur* $\wedge$ *unit-prop-body-wl-D-find-unwatched-inv f C* S')›

definition *keep-watch-heur* **where**
‹*keep-watch-heur* = (λ*L i j* (*M*, *N*, *D*, *Q*, *W*, *vm*). *do* {
ASSERT(*nat-of-lit L* < *length W*);
ASSERT(*i* < *length* (*W* ! *nat-of-lit L*));
ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
RETURN (*M*, *N*, *D*, *Q*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*i* := *W* ! (*nat-of-lit L*) ! *j*]], *vm*)
})›

definition *update-blit-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *bool* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒
(*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where
‹*update-blit-wl-heur* = (λ(*L*::*nat literal*) *C b j w K* (*M*, *N*, *D*, *Q*, *W*, *vm*). *do* {
ASSERT(*nat-of-lit L* < *length W*);
ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
ASSERT(*j* < *length N*);
ASSERT(*w* < *length N*);
RETURN (*j+1*, *w+1*, (*M*, *N*, *D*, *Q*, *W*[*nat-of-lit L* := (*W*!*nat-of-lit L*)[*j*:=*to-watcher C K b*]], *vm*))
})›

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv0* **where**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* =
(λ(*j*, *w*, *S'*). ∃*S*. (*S'*, *S*) ∈ *twl-st-heur* ∧ *unit-propagation-inner-loop-wl-loop-D-inv L* (*j*, *w*, *S*) ∧
length (*watched-by S L*) ≤ *length* (*get-clauses-wl-heur S'*) − *4*)›

definition *unit-propagation-inner-loop-body-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ (*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where
‹*unit-propagation-inner-loop-body-wl-heur L j w* (*S0* :: *twl-st-wl-heur*) = *do* {
ASSERT(*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* (*j*, *w*, *S0*));
ASSERT(*watched-by-app-heur-pre* ((*S0*, *L*), *w*));
let (*C*, *K*, *b*) = *watcher-of* (*watched-by-app-heur S0 L w*);
S ← *keep-watch-heur L j w S0*;
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
ASSERT(*unit-prop-body-wl-heur-inv S j w L*);
ASSERT(*polarity-st-heur-pre* (*S*, *K*));
ASSERT(*length* (*get-clauses-wl-heur S0*) ≤ *uint64-max* ⟶ *j* < *uint64-max* ∧ *w* < *uint64-max*);
let val-K = *polarity-st-heur S K*;
if val-K = *Some True*
then RETURN (*j+1*, *w+1*, *S*)
else do {
if b then do {
if val-K = *Some False*
then do {
ASSERT(*set-conflict-wl-heur-pre* (*C*, *S*));
S ← *set-conflict-wl-heur C S*;
RETURN (*j+1*, *w+1*, *S*)}
else do {
ASSERT(*access-lit-in-clauses-heur-pre* ((*S*, *C*), *0*));
let i = (*if access-lit-in-clauses-heur S C 0* = *L then 0 else 1*);
ASSERT(*propagate-lit-wl-heur-pre* (((*K*, *C*), *i*), *S*));
S ← *propagate-lit-wl-bin-heur K C i S*;
RETURN (*j+1*, *w+1*, *S*)}
}
else do {

— Now the costly operations:
ASSERT(*clause-not-marked-to-delete-heur-pre* (*S*, *C*));
if ¬*clause-not-marked-to-delete-heur S C*
then RETURN (*j*, *w+1*, *S*)
else do {
ASSERT(*access-lit-in-clauses-heur-pre* ((*S*, *C*), *0*));
let i = (*if access-lit-in-clauses-heur S C 0* = *L then 0 else 1*);
ASSERT(*access-lit-in-clauses-heur-pre* ((*S*, *C*), *1* − *i*));
let L′ = *access-lit-in-clauses-heur S C* (*1* − *i*);
ASSERT(*polarity-st-heur-pre* (*S*, *L′*));
let val-L′ = *polarity-st-heur S L′*;
if val-L′ = *Some True*
then update-blit-wl-heur L C b j w L′ S
else do {
ASSERT(*find-unwatched-wl-st-heur-pre* (*S*, *C*));
f ← *isa-find-unwatched-wl-st-heur S C*;
ASSERT (*unit-prop-body-wl-D-find-unwatched-heur-inv f C S*);
case f of
None ⇒ *do* {
if val-L′ = *Some False*
then do {
ASSERT(*set-conflict-wl-heur-pre* (*C*, *S*));
S ← *set-conflict-wl-heur C S*;
RETURN (*j+1*, *w+1*, *S*)}
else do {
ASSERT(*propagate-lit-wl-heur-pre* (((*L′*, *C*), *i*), *S*));
S ← *propagate-lit-wl-heur L′ C i S*;
RETURN (*j+1*, *w+1*, *S*)}
}
| *Some f* ⇒ *do* {
S ← *isa-save-pos C f S*;
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
ASSERT(*access-lit-in-clauses-heur-pre* ((*S*, *C*), *f*));
let K = *access-lit-in-clauses-heur S C f*;
ASSERT(*polarity-st-heur-pre* (*S*, *K*));
let val-L′ = *polarity-st-heur S K*;
if val-L′ = *Some True*
then update-blit-wl-heur L C b j w K S
else do {
ASSERT(*update-clause-wl-code-pre* (((((((*L*, *C*), *b*), *j*), *w*), *i*), *f*), *S*));
update-clause-wl-heur L C b j w i f S
}
}
}
}
}
}
})

lemma *set-conflict-wl′-alt-def2*:
‹*RETURN oo set-conflict-wl′* =
(*λC* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *do* {
let D = *Some* (*mset* (*N* ∝ *C*));
RETURN (*M*, *N*, *D*, *NE*, *UE*, {#}, *W*) })
›
⟨*proof*⟩

declare *RETURN-as-SPEC-refine*[*refine2 del*]

definition *set-conflict-wl′-pre* **where**
‹*set-conflict-wl′-pre i S* ⟷
get-conflict-wl S = *None* ∧ *i* ∈# *dom-m* (*get-clauses-wl S*) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf* (*get-clauses-wl S*)) ∧
¬ *tautology* (*mset* (*get-clauses-wl S* ∝ *i*)) ∧
distinct (*get-clauses-wl S* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)›

lemma *set-conflict-wl-heur-set-conflict-wl′*:
‹(*uncurry set-conflict-wl-heur*, *uncurry* (*RETURN oo set-conflict-wl′*)) ∈
[*uncurry set-conflict-wl′-pre*]$_f$
nat-rel $\times_r$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩*nres-rel*›
⟨*proof*⟩

lemma *in-Id-in-Id-option-rel*[*refine*]:
‹(*f*, *f′*) ∈ *Id* ⟹ (*f*, *f′*) ∈ ⟨*Id*⟩ *option-rel*›
⟨*proof*⟩

The assumption that that accessed clause is active has not been checked at this point!

definition *keep-watch-heur-pre* **where**
‹*keep-watch-heur-pre* =
(λ(((*L*, *j*), *w*), *S*). *j* < *length* (*watched-by S L*) ∧ *w* < *length* (*watched-by S L*) ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›

lemma *vdom-m-update-subset′*:
‹*fst C* ∈ *vdom-m* $\mathcal{A}$ *bh N* ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *C*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
⟨*proof*⟩

lemma *vdom-m-update-subset*:
‹*bg* < *length* (*bh ap*) ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *bh ap* ! *bg*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
⟨*proof*⟩

lemma *keep-watch-heur-keep-watch*:
‹(*uncurry3 keep-watch-heur*, *uncurry3* (*RETURN oooo keep-watch*)) ∈
[*keep-watch-heur-pre*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩ *nres-rel*›
⟨*proof*⟩

This is a slightly stronger version of the previous lemma:

lemma *keep-watch-heur-keep-watch′*:
‹*keep-watch-heur-pre* (((*L*, *j*), *w*), *S*) ⟹
((((*L′*, *j′*), *w′*), *S′*), (((*L*, *j*), *w*), *S*)
∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* ⟹
keep-watch-heur L′ j′ w′ S′ ≤ ⇓ {(*T*, *T′*). *get-vdom T* = *get-vdom S′* ∧
(*T*, *T′*) ∈ *twl-st-heur-up″* $\mathcal{D}$ *r s K*}
(*RETURN* (*keep-watch L j w S*))›
⟨*proof*⟩

definition *update-blit-wl-heur-pre* **where**
‹*update-blit-wl-heur-pre r* = (λ((((((*L*, *C*), *b*), *j*), *w*), *K*), *S*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧

$w < length\ (watched\text{-}by\ S\ L) \wedge w < r \wedge j < r \wedge$
$j < length\ (watched\text{-}by\ S\ L) \wedge C \in vdom\text{-}m\ (all\text{-}atms\text{-}st\ S)\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S))$›

lemma *update-blit-wl-heur-update-blit-wl*:
‹$(uncurry6\ update\text{-}blit\text{-}wl\text{-}heur,\ uncurry6\ update\text{-}blit\text{-}wl) \in$
$[update\text{-}blit\text{-}wl\text{-}heur\text{-}pre\ r]_f$
$nat\text{-}lit\text{-}lit\text{-}rel \times_f nat\text{-}rel \times_f bool\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f Id \times_f$
$twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K \rightarrow$
$\langle nat\text{-}rel \times_r nat\text{-}rel \times_r twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\rangle\ nres\text{-}rel$›
$\langle proof\rangle$

lemma *unit-propagation-inner-loop-body-wl-D-alt-def*:
‹*unit-propagation-inner-loop-body-wl-D L j w S = do* {
ASSERT(*unit-propagation-inner-loop-wl-loop-D-pre L* (*j*, *w*, *S*));
let (*C*, *K*, *b*) = (*watched-by S L*) ! *w*;
let S = *keep-watch L j w S*;
ASSERT(*unit-prop-body-wl-D-inv S j w L*);
let val-K = *polarity* (*get-trail-wl S*) *K*;
if val-K = *Some True*
then RETURN (*j+1*, *w+1*, *S*)
else do {
if b then do {
ASSERT (*propagate-proper-bin-case L K S C*);
if val-K = *Some False*
then
let S = *set-conflict-wl* (*get-clauses-wl S* $\propto$ *C*) *S in*
RETURN
(*j* + *1*, *w* + *1*, *S*)
else
let i = ((*if get-clauses-wl S* $\propto$ *C* ! *0* = *L then 0 else 1*)) *in*
let S = *propagate-lit-wl-bin K C i S in*
RETURN
(*j* + *1*, *w* + *1*, *S*)
}
else — Now the costly operations:
if C $\notin\#$ *dom-m* (*get-clauses-wl S*)
then RETURN (*j*, *w+1*, *S*)
else do {
let i = (*if* ((*get-clauses-wl S*)$\propto$*C*) ! *0* = *L then 0 else 1*);
let L' = ((*get-clauses-wl S*)$\propto$*C*) ! (*1* − *i*);
let val-L' = *polarity* (*get-trail-wl S*) *L'*;
if val-L' = *Some True*
then update-blit-wl L C b j w L' S
else do {
f ← *find-unwatched-l* (*get-trail-wl S*) (*get-clauses-wl S* $\propto$ *C*);
ASSERT (*unit-prop-body-wl-D-find-unwatched-inv f C S*);
case f of
None ⇒ *do* {
if val-L' = *Some False*
then do {
let S = *set-conflict-wl* (*get-clauses-wl S* $\propto$ *C*) *S*;
RETURN (*j+1*, *w+1*, *S*)
}
else do {
S ← *RETURN* (*propagate-lit-wl L' C i S*);
RETURN (*j+1*, *w+1*, *S*)

```
            }
          }
        | Some f ⇒ do {
            S ← RETURN S;
            let K = get-clauses-wl S ∝ C ! f;
            let val-L′ = polarity (get-trail-wl S) K;
            if val-L′ = Some True
            then update-blit-wl L C b j w K S
            else update-clause-wl L C b j w i f S
          }
      }
    }
  }
})›
⟨proof⟩
```

The lemmas below are used in the refinement proof of *unit-propagation-inner-loop-body-wl-D*. None of them makes sense in any other context. However having like below allows to share intermediate steps in a much easier fashion that in an Isar proof.

```
context
  fixes x y x1a L x2 x2a x1 S x1c x2d L′ x1d x2c T 𝒟 r s K
  assumes
    xy: ‹(x, y) ∈ nat-lit-lit-rel ×f nat-rel ×f nat-rel ×f
       twl-st-heur-up″ 𝒟 r s K› and
    pre: ‹unit-propagation-inner-loop-wl-loop-D-pre L (x2, x2a, T)› and
    pre-inv0: ‹unit-propagation-inner-loop-wl-loop-D-heur-inv0 L′ (x2c, x2d, S)› and
    st:
      ‹x1a = (L, x2)›
      ‹x1 = (x1a, x2a)›
      ‹y = (x1, T)›
      ‹x1d = (L′, x2c)›
      ‹x1c = (x1d, x2d)›
      ‹x = (x1c, S)› and
    L-K0: ‹case y of
       (x, xa) ⇒
         (case x of
          (x, xa) ⇒
            (case x of
             (L, i) ⇒
               λj S. length (watched-by S L) ≤ r − 4 ∧
                     L = K ∧ length (watched-by S L) = s)
             xa)
          xa›
begin

private lemma L-K: ‹L = K›
  ⟨proof⟩ lemma state-simp-ST:
  ‹x1a = (L, x2)›
  ‹x1 = ((L, x2), x2a)›
  ‹y = (((L, x2), x2a), T)›
  ‹x1d = (L, x2)›
  ‹x1c = ((L, x2), x2a)›
  ‹x = (((L, x2), x2a), S)›
  ‹L′ = L›
  ‹x2c = x2›
```

⟨*x2d* = *x2a*⟩ **and**
st: ⟨(*S*, *T*) ∈ *twl-st-heur*⟩
⟨*proof*⟩ **lemma** *length-clss-Sr*: ⟨*length* (*get-clauses-wl-heur S*) = *r*⟩
⟨*proof*⟩ **lemma**
x1b: ⟨$L \in\# \mathcal{L}_{all}$ (*all-atms-st T*)⟩ **and**
x2b: ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st T*) *T*⟩ **and**
loop-inv-T: ⟨*unit-propagation-inner-loop-wl-loop-inv L* (*x2*, *x2a*, *T*)⟩
⟨*proof*⟩ **lemma** *x2d-le*: ⟨*x2d* < *length* (*watched-by-int S L*)⟩ **and**
x1e-le: ⟨*nat-of-lit L* < *length* (*get-watched-wl-heur S*)⟩ **and**
x2-x2a: ⟨*x2* ≤ *x2a*⟩ **and**
x2a-le: ⟨*x2a* < *length* (*watched-by T L*)⟩ **and**
valid: ⟨*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))⟩
and
corr-T: ⟨*correct-watching-except x2 x2a L T*⟩
⟨*proof*⟩

lemma *watched-by-app-heur-pre*: ⟨*watched-by-app-heur-pre* ((*S*, *L'*), *x2d*)⟩
⟨*proof*⟩

lemma *keep-watch-heur-pre*: ⟨*keep-watch-heur-pre* (((*L*, *x2*), *x2a*), *T*)⟩
⟨*proof*⟩

context — Now we copy the watch literals
notes -[*simp*]= *state-simp-ST x1b x2b*
fixes *x1f x2f x1g x2g U x2e x2g' x2h x2f' x2f''*
assumes
xf: ⟨*watched-by T L* ! *x2a* = (*x1f*, *x2f'*)⟩ **and**
xg: ⟨*watched-by-int S L'* ! *x2d* = (*x1g*, *x2g'*)⟩ **and**
x2g': ⟨*x2g'* = (*x2g*, *x2h*)⟩ **and**
x2f': ⟨*x2f'* = (*x2f*, *x2f''*)⟩ **and**
U: ⟨(*U*, *keep-watch L x2 x2a T*)
∈ {(*GT*, *GT'*). *get-vdom GT* = *get-vdom S* ∧
(*GT*, *GT'*) ∈ *twl-st-heur-up''* $\mathcal{D}$ *r s K*}⟩ **and**
prop-inv: ⟨*unit-prop-body-wl-D-inv* (*keep-watch L x2 x2a T*) *x2 x2a L*⟩ **and**
prop-heur-inv: ⟨*unit-prop-body-wl-heur-inv U x2c x2d L'*⟩
begin

private lemma *U'*: ⟨(*U*, *keep-watch L x2 x2a T*) ∈ *twl-st-heur*⟩
⟨*proof*⟩ **lemma** *eq*: ⟨*watched-by T L* = *watched-by-int S L*⟩ ⟨*x1f* = *x1g*⟩ ⟨*x2f'* = *x2g'*⟩ ⟨*x2f* = *x2g*⟩
⟨*x2f''* = *x2h*⟩
⟨*proof*⟩

lemma *xg-S*: ⟨*watched-by-int S L* ! *x2a* = (*x1g*, *x2g'*)⟩
⟨*proof*⟩

lemma *xg-T*: ⟨*watched-by T L* ! *x2a* = (*x1g*, *x2g'*)⟩
⟨*proof*⟩

context
notes -[*simp*]= *eq xg-S xg-T x2g'*
begin

lemma *in-D0*:

shows ‹*polarity-st-heur-pre* (*U*, *x2g*)›
⟨*proof*⟩ **lemma** *x2g*: ‹*x2g* ∈# $\mathcal{L}_{all}$ (*all-atms-st T*)›
⟨*proof*⟩

lemma *polarity-eq*:
‹(*polarity-pol* (*get-trail-wl-heur U*) *x2g* = *Some True*) ⟷
(*polarity* (*get-trail-wl* (*keep-watch L x2 x2a T*)) *x2f* = *Some True*)›
⟨*proof*⟩

lemma
valid-UT:
‹*valid-arena* (*get-clauses-wl-heur U*) (*get-clauses-wl T*) (*set* (*get-vdom U*))› **and**
vdom-m-UT:
‹*vdom-m* (*all-atms-st T*) (*get-watched-wl* (*keep-watch L x2 x2a T*)) (*get-clauses-wl T*) ⊆ *set* (*get-vdom U*)›
⟨*proof*⟩ **lemma** *x1g-vdom*: ‹*x1f* ∈ *vdom-m* (*all-atms-st T*) (*get-watched-wl* (*keep-watch L x2 x2a T*))
(*get-clauses-wl* (*keep-watch L x2 x2a T*))›
⟨*proof*⟩

lemma *clause-not-marked-to-delete-heur-pre*:
‹*clause-not-marked-to-delete-heur-pre* (*U*, *x1g*)›
⟨*proof*⟩ **lemma** *clause-not-marked-to-delete-pre*:
‹*clause-not-marked-to-delete-pre* (*keep-watch L x2 x2a T*, *x1f*)›
⟨*proof*⟩

lemma *clause-not-marked-to-delete-heur-clause-not-marked-to-delete-iff*:
‹(¬ *clause-not-marked-to-delete-heur U x1g*) ⟷
(¬ *clause-not-marked-to-delete* (*keep-watch L x2 x2a T*) *x1f*)›
⟨*proof*⟩ **lemma** *lits-in-trail*:
‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st T*) (*get-trail-wl T*)› **and**
no-dup-T: ‹*no-dup* (*get-trail-wl T*)› **and**
pol-L: ‹*polarity* (*get-trail-wl T*) *L* = *Some False*› **and**
correct-watching-x2: ‹*correct-watching-except x2 x2a L T*›
⟨*proof*⟩

lemma *prop-fast-le*:
assumes *fast*: ‹*length* (*get-clauses-wl-heur S*) ≤ *uint64-max*›
shows ‹*x2c* < *uint64-max*› ‹*x2d* < *uint64-max*›
⟨*proof*⟩

context
fixes *x1i x2i x1i′ x2i′*
assumes *x2h*: ‹*x2f′* = (*x1i′*, *x2i′*)› **and**
x2h′: ‹*x2g′* = (*x1i*, *x2i*)›
begin

lemma *bin-last-eq*: ‹*x2i* = *x2i′*›
⟨*proof*⟩

context
assumes *proper*: ‹*propagate-proper-bin-case L x2f* (*keep-watch L x2 x2a T*) *x1f*›
begin

private lemma *bin-confl-T*: ‹*get-conflict-wl T* = *None*› **and**

bin-dist-Tx1g: ‹*distinct* (*get-clauses-wl* *T* ∝ *x1g*)› **and**
in-dom: ‹*x1f* ∈# *dom-m* (*get-clauses-wl* (*keep-watch* *L* *x2* *x2a* *T*))› **and**
length-clss-2: ‹*length* (*get-clauses-wl* *T* ∝ *x1g*) = *2*›
⟨*proof*⟩

lemma *bin-polarity-eq*:
‹(*polarity-pol* (*get-trail-wl-heur* *U*) *x2g* = *Some False*) ⟷
(*polarity* (*get-trail-wl* (*keep-watch* *L* *x2* *x2a* *T*)) *x2f* = *Some False*)›
⟨*proof*⟩

lemma *bin-set-conflict-wl-heur-pre*:
‹*set-conflict-wl-heur-pre* (*x1g*, *U*)›
⟨*proof*⟩

lemma *polarity-st-keep-watch*:
‹*polarity-st* (*keep-watch* *L* *x2* *x2a* *T*) = *polarity-st* *T*›
⟨*proof*⟩

lemma *access-lit-in-clauses-keep-watch*:
‹*access-lit-in-clauses* (*keep-watch* *L* *x2* *x2a* *T*) = *access-lit-in-clauses* *T*›
⟨*proof*⟩

lemma *bin-set-conflict-wl′-pre*:
‹*uncurry* *set-conflict-wl′-pre* (*x1f*, (*keep-watch* *L* *x2* *x2a* *T*))›
if *pol*: ‹*polarity-pol* (*get-trail-wl-heur* *U*) *x2g* = *Some False*›
⟨*proof*⟩

lemma *bin-conflict-rel*:
‹((*x1g*, *U*), *x1f*, *keep-watch* *L* *x2* *x2a* *T*)
∈ *nat-rel* $\times_f$ *twl-st-heur-up*′′ $\mathcal{D}$ *r* *s* *K*›
⟨*proof*⟩

lemma *bin-access-lit-in-clauses-heur-pre*:
‹*access-lit-in-clauses-heur-pre* ((*U*, *x1g*), *0*)›
⟨*proof*⟩

lemma *bin-propagate-lit-wl-heur-pre*:
‹*propagate-lit-wl-heur-pre*
(((*x2g*, *x1g*), *if* *arena-lit* (*get-clauses-wl-heur* *U*) (*x1g* + *0*) = *L*′ *then* *0* *else* *1*::*nat*), *U*)›
if *pol*: ‹*polarity-pol* (*get-trail-wl-heur* *U*) *x2g* ≠ *Some False*› **and**
pol′: ‹*polarity* (*get-trail-wl* (*keep-watch* *L* *x2* *x2a* *T*)) *x2f* ≠ *Some True*›
⟨*proof*⟩

lemma *bin-propagate-lit-wl-pre*:
‹*propagate-lit-wl-bin-pre*
(((*x2f*, *x1f*), *if* *get-clauses-wl* (*keep-watch* *L* *x2* *x2a* *T*) ∝ *x1f* ! *0* = *L* *then* *0* *else* *1*::*nat*),
(*keep-watch* *L* *x2* *x2a* *T*))›
if *pol*: ‹*polarity-pol* (*get-trail-wl-heur* *U*) *x2g* ≠ *Some False*› **and**
pol′: ‹*polarity* (*get-trail-wl* (*keep-watch* *L* *x2* *x2a* *T*)) *x2f* ≠ *Some True*›
⟨*proof*⟩ **lemma** *bin-arena-lit-eq*:
‹*i* < *2* ⟹ *arena-lit* (*get-clauses-wl-heur* *U*) (*x1g* + *i*) = *get-clauses-wl* *T* ∝ *x1g* ! *i*›
⟨*proof*⟩

lemma *bin-final-rel*:

‹((((x2g, x1g), if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L′ then 0 else 1::nat), U),
((x2f, x1f), if get-clauses-wl (keep-watch L x2 x2a T) ∝ x1f ! 0 = L then 0 else 1::nat),
(keep-watch L x2 x2a T)) ∈ Id $\times_f$ nat-rel $\times_f$
twl-st-heur-up′′ 𝒟 r s K›
⟨proof⟩

end

end

context — Now we know that the clause has not been deleted
assumes *not-del*: ‹¬ ¬ clause-not-marked-to-delete (keep-watch L x2 x2a T) x1f›
begin

private lemma *x1g*:
‹x1g ∈# dom-m (get-clauses-wl T)›
⟨proof⟩ **lemma** *Tx1g-le2*:
‹length (get-clauses-wl T ∝ x1g) ≥ 2›
⟨proof⟩

lemma *access-lit-in-clauses-heur-pre0*:
‹access-lit-in-clauses-heur-pre ((U, x1g), 0)›
⟨proof⟩ **definition** *i* :: *nat* **where**
‹i = ((if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L then 0 else 1))›

lemma *i-alt-def-L′*:
‹i = ((if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L′ then 0 else 1))›
⟨proof⟩

lemma *access-lit-in-clauses-heur-pre1i*:
‹access-lit-in-clauses-heur-pre ((U, x1g),
1 − ((if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L′ then 0 else 1)))›
⟨proof⟩ **lemma** *trail-UT*:
‹(get-trail-wl-heur U, get-trail-wl T) ∈ trail-pol (all-atms-st T)›
⟨proof⟩

lemma *polarity-st-pre1i*:
‹polarity-st-heur-pre (U, arena-lit (get-clauses-wl-heur U)
(x1g + (1 − (if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L′ then 0 else 1))))›
⟨proof⟩ **lemma**
access-x1g:
‹arena-lit (get-clauses-wl-heur U) (x1g + 0) =
get-clauses-wl (keep-watch L x2 x2a T) ∝ x1f ! 0› **and**
access-x1g1i:
‹arena-lit (get-clauses-wl-heur U) (x1g + (1 − i)) =
get-clauses-wl (keep-watch L x2 x2a T) ∝ x1f ! (1 − i)› **and**
i-alt-def:
‹i = (if get-clauses-wl (keep-watch L x2 x2a T) ∝ x1f ! 0 = L then 0 else 1)›
⟨proof⟩

lemma *polarity-other-watched-lit*:
‹(polarity-pol (get-trail-wl-heur U) (arena-lit (get-clauses-wl-heur U) (x1g +
(1 − (if arena-lit (get-clauses-wl-heur U) (x1g + 0) = L′ then 0 else 1)))) =
Some True) =
(polarity (get-trail-wl (keep-watch L x2 x2a T)) (get-clauses-wl (keep-watch L x2 x2a T) ∝

x1f ! (*1* − (*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0 else 1*))) =
Some True)›
⟨*proof*⟩

lemma *update-blit-wl-heur-pre*:
‹*update-blit-wl-heur-pre r* ((((((*L*, *x1f*), *x1f''*), *x2*), *x2a*), *get-clauses-wl* (*keep-watch L x2 x2a T*) ∝
x1f ! (*1* − (*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0 else 1*))),
keep-watch L x2 x2a T)›
⟨*proof*⟩

lemma *update-blit-wl-rel*:
‹(((((((*L'*, *x1g*), *x2h*), *x2c*), *x2d*),
arena-lit (*get-clauses-wl-heur U*)
(*x1g* + (*1* − (*if arena-lit* (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L'*
then 0 else 1)))), *U*),
(((((*L*, *x1f*), *x2f''*), *x2*), *x2a*),
get-clauses-wl (*keep-watch L x2 x2a T*) ∝ *x1f* ! (*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L*
then 0 else 1))),
keep-watch L x2 x2a T)
∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$
nat-rel $\times_f$
nat-rel $\times_f$
nat-lit-lit-rel $\times_f$
twl-st-heur-up'' $\mathcal{D}$ *r s K*›
⟨*proof*⟩

lemma *find-unwatched-wl-st-pre*:
‹*find-unwatched-wl-st-pre* (*keep-watch L x2 x2a T*, *x1f*)›
⟨*proof*⟩

lemma *find-unwatched-wl-st-heur-pre*:
‹*find-unwatched-wl-st-heur-pre* (*U*, *x1g*)›
⟨*proof*⟩

lemma *isa-find-unwatched-wl-st-heur-pre*:
‹((*U*, *x1g*), *keep-watch L x2 x2a T*, *x1f*) ∈ *twl-st-heur* $\times_f$ *nat-rel*› **and**
isa-find-unwatched-wl-st-heur-lits:
‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*keep-watch L x2 x2a T*)) (*keep-watch L x2 x2a T*)›
⟨*proof*⟩

context — Now we try to find another literal to watch
notes - [*simp*] = *x1g*
fixes *f f'*
assumes *ff*: ‹(*f*, *f'*) ∈ *Id*› **and**
find-unw-pre: ‹*unit-prop-body-wl-D-find-unwatched-inv f' x1f* (*keep-watch L x2 x2a T*)›
begin

private lemma *ff*: ‹*f* = *f'*›
⟨*proof*⟩

lemma *unit-prop-body-wl-D-find-unwatched-heur-inv*:
‹*unit-prop-body-wl-D-find-unwatched-heur-inv f x1g U*›
⟨*proof*⟩ **lemma** *confl-T*: ‹*get-conflict-wl T* = *None*› **and**
dist-Tx1g: ‹*distinct* (*get-clauses-wl T* ∝ *x1g*)› **and**

L-in-watched: ‹$L \in$ *set* (*watched-l* (*get-clauses-wl* $T \propto$ *x1g*))›
⟨*proof*⟩

context — No replacement found
notes -[*simp*] = *ff*
assumes
f: ‹*f* = *None*› **and**
f'[*simp*]: ‹*f'* = *None*›
begin

lemma *pol-other-lit-false*:
‹(*polarity-pol* (*get-trail-wl-heur U*)
(*arena-lit* (*get-clauses-wl-heur U*)
(*x1g* +
(*1* −
(*if arena-lit* (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L' then 0*
else 1)))) =
Some False) =
(*polarity* (*get-trail-wl* (*keep-watch L x2 x2a T*))
(*get-clauses-wl* (*keep-watch L x2 x2a T*) $\propto$ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) $\propto$ *x1f* ! *0* = *L then 0*
else 1))) =
Some False)›
⟨*proof*⟩

lemma *set-conflict-wl-heur-pre*: ‹*set-conflict-wl-heur-pre* (*x1g*, *U*)›
⟨*proof*⟩

lemma *i-alt-def2*:
‹*i* = (*if access-lit-in-clauses* (*keep-watch L x2 x2a T*) *x1f 0* = *L then 0*
else 1)›
⟨*proof*⟩

lemma *x2da-eq*: ‹(*x2d*, *x2a*) $\in$ *nat-rel*›
⟨*proof*⟩

context
assumes ‹*polarity-pol* (*get-trail-wl-heur U*)
(*arena-lit* (*get-clauses-wl-heur U*)
(*x1g* +
(*1* −
(*if arena-lit* (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L' then 0*
else 1)))) =
Some False› **and**
pol-false: ‹*polarity* (*get-trail-wl* (*keep-watch L x2 x2a T*))
(*get-clauses-wl* (*keep-watch L x2 x2a T*) $\propto$ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) $\propto$ *x1f* ! *0* = *L then 0*
else 1))) =
Some False›
begin

lemma *unc-set-conflict-wl'-pre*: ‹*uncurry set-conflict-wl'-pre* (*x1f*, *keep-watch L x2 x2a T*)›
⟨*proof*⟩

lemma *set-conflict-keep-watch-rel*:
$\langle((x1g,\ U),\ x1f,\ keep\text{-}watch\ L\ x2\ x2a\ T) \in nat\text{-}rel \times_f\ twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\rangle$
$\langle proof\rangle$

lemma *set-conflict-keep-watch-rel2*:
$\langle\bigwedge r.\ (W,\ W') \in nat\text{-}rel \times_f\ twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K \Longrightarrow$
$\quad((x2c + 1,\ W),\ x2 + 1,\ W') \in nat\text{-}rel \times_f\ (nat\text{-}rel \times_f\ twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K)\rangle$
$\langle proof\rangle$

end

context
assumes $\langle polarity\text{-}pol\ (get\text{-}trail\text{-}wl\text{-}heur\ U)$
$\quad(arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)$
$\quad\quad(x1g +$
$\quad\quad\ (1 -$
$\quad\quad\quad(if\ arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)\ (x1g + 0) = L'\ then\ 0$
$\quad\quad\quad\ else\ 1)))) \neq$
$\quad Some\ False\rangle$ **and**
$\quad pol\text{-}False$: $\langle polarity\ (get\text{-}trail\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T))$
$\quad\ (get\text{-}clauses\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T) \propto x1f\ !$
$\quad\quad(1 -$
$\quad\quad\ (if\ get\text{-}clauses\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T) \propto x1f\ !\ 0 = L\ then\ 0$
$\quad\quad\quad else\ 1))) \neq$
$\quad Some\ False\rangle$ **and**
$\langle polarity\text{-}pol\ (get\text{-}trail\text{-}wl\text{-}heur\ U)$
$\quad(arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)$
$\quad\quad(x1g +$
$\quad\quad\ (1 -$
$\quad\quad\quad(if\ arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)\ (x1g + 0) = L'\ then\ 0$
$\quad\quad\quad\ else\ 1)))) \neq$
$\quad Some\ True\rangle$ **and**
$pol\text{-}True$: $\langle polarity\ (get\text{-}trail\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T))$
$\quad(get\text{-}clauses\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T) \propto x1f\ !$
$\quad\ (1 -$
$\quad\quad(if\ get\text{-}clauses\text{-}wl\ (keep\text{-}watch\ L\ x2\ x2a\ T) \propto x1f\ !\ 0 = L\ then\ 0$
$\quad\quad\ else\ 1))) \neq$
$\quad Some\ True\rangle$
begin

private lemma *undef-lit1i*:
$\langle undefined\text{-}lit\ (get\text{-}trail\text{-}wl\ T)\ (get\text{-}clauses\text{-}wl\ T \propto x1g\ !\ (Suc\ 0 - i))\rangle$
$\langle proof\rangle$

lemma *propagate-lit-wl-heur-pre*:
$\langle propagate\text{-}lit\text{-}wl\text{-}heur\text{-}pre$
$\quad(((arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)$
$\quad\quad(x1g +$
$\quad\quad(1 -$
$\quad\quad\ (if\ arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)\ (x1g + 0) = L'\ then\ 0$
$\quad\quad\ else\ 1))),$
$\quad\ x1g),$
$\quad\ if\ arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ U)\ (x1g + 0) = L'\ then\ 0\ else\ (1{::}\ nat)),$
$\quad U)\rangle$ (**is** $?A$)
$\langle proof\rangle$ **lemma** *propagate-lit-wl-i-0-1*: $\langle i = 0 \lor i = 1\rangle$

⟨*proof*⟩

lemma *propagate-lit-wl-pre*: ‹*propagate-lit-wl-pre*
(((*get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0*
else 1)),
x1f),
if get-clauses-wl (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0 else 1*),
keep-watch L x2 x2a T)›
⟨*proof*⟩

lemma *propagate-lit-wl-rel*:
‹((((*arena-lit* (*get-clauses-wl-heur U*)
(*x1g* +
(*1* −
(*if arena-lit* (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L′ then 0*
else 1))),
x1g),
if arena-lit (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L′ then 0 else 1*),
U),
((*get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0*
else 1)),
x1f),
if get-clauses-wl (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0 else 1*),
keep-watch L x2 x2a T)
∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″ 𝒟 r s K*›
⟨*proof*⟩

end

end

context — No replacement found
fixes *i j*
assumes
f: ‹*f* = *Some i*› **and**
f′[*simp*]: ‹*f′* = *Some j*›
begin

private lemma *ij*: ‹*i* = *j*›
⟨*proof*⟩ **lemma**
‹*unit-prop-body-wl-find-unwatched-inv* (*Some j*) *x1g*
(*keep-watch L x2 x2a T*)› **and**
j-ge2: ‹*2* ≤ *j*› **and**
j-le: ‹*j* < *length* (*get-clauses-wl T* ∝ *x1g*)› **and**
T-x1g-j-neq0: ‹*get-clauses-wl T* ∝ *x1g* ! *j* ≠ *get-clauses-wl T* ∝ *x1g* ! *0*› **and**
T-x1g-j-neq1: ‹*get-clauses-wl T* ∝ *x1g* ! *j* ≠ *get-clauses-wl T* ∝ *x1g* ! *Suc 0*›
⟨*proof*⟩ **lemma** *isa-update-pos-pre*:
‹*MAX-LENGTH-SHORT-CLAUSE* < *arena-length* (*get-clauses-wl-heur U*) *x1g* ⟹
isa-update-pos-pre ((*x1g*, *j*), *get-clauses-wl-heur U*)›
⟨*proof*⟩ **abbreviation** *isa-save-pos-rel* **where**

‹*isa-save-pos-rel* ≡ {(V, V'). *get-vdom* V = *get-vdom* S ∧ (V, V') ∈ *twl-st-heur'* $\mathcal{D}$ ∧
V' = *keep-watch* L *x2* *x2a* T ∧ *get-trail-wl-heur* V = *get-trail-wl-heur* U ∧
length (*get-clauses-wl-heur* V) = *length* (*get-clauses-wl-heur* U) ∧
get-vdom V = *get-vdom* U ∧ *get-watched-wl-heur* V = *get-watched-wl-heur* U} ›

lemma *isa-save-pos*:
‹*isa-save-pos* *x1g* *i* U ≤ ⇓ *isa-save-pos-rel*
(*RETURN* (*keep-watch* L *x2* *x2a* T))›
⟨*proof*⟩

context
notes -[*simp*] = *ij*
fixes V V'
assumes *VV'*: ‹(V, V') ∈ *isa-save-pos-rel*›
begin

private lemma
‹*get-vdom* U = *get-vdom* S› **and**
V-T-rel: ‹(V, *keep-watch* L *x2* *x2a* T) ∈ *twl-st-heur-up''* $\mathcal{D}$ *r* *s* K› **and**
VV':
‹V' = *keep-watch* L *x2* *x2a* T›
‹*get-trail-wl-heur* V = *get-trail-wl-heur* U›
‹*get-vdom* V = *get-vdom* S›
‹*get-watched-wl-heur* V = *get-watched-wl-heur* U› **and**
valid-VT: ‹*valid-arena* (*get-clauses-wl-heur* V) (*get-clauses-wl* T) (*set* (*get-vdom* U))› **and**
trail-VT: ‹(*get-trail-wl-heur* V, *get-trail-wl* (*keep-watch* L *x2* *x2a* T))
∈ *trail-pol* (*all-atms-st* (*keep-watch* L *x2* *x2a* T))›
⟨*proof*⟩

lemma *access-lit-in-clauses-heur-pre3*: ‹*access-lit-in-clauses-heur-pre* ((V, *x1g*), *i*)›
⟨*proof*⟩ **lemma** *arena-lit-x1g-j*:
‹*arena-lit* (*get-clauses-wl-heur* V) (*x1g* + *j*) = *get-clauses-wl* T ∝ *x1g* ! *j*›
⟨*proof*⟩

lemma *polarity-st-pre-unwatched*: ‹*polarity-st-heur-pre* (V, *arena-lit* (*get-clauses-wl-heur* V) (*x1g* + *i*))›
⟨*proof*⟩ **lemma** *j-Lall*: ‹*get-clauses-wl* V' ∝ *x1g* ! *j* ∈# $\mathcal{L}_{all}$ (*all-atms-st* T)›
⟨*proof*⟩

lemma *polarity-eq-unwatched*: ‹(*polarity-pol* (*get-trail-wl-heur* V)
(*arena-lit* (*get-clauses-wl-heur* V) (*x1g* + *i*)) =
Some True) =
(*polarity* (*get-trail-wl* V')
(*get-clauses-wl* V' ∝ *x1f* ! *j*) =
Some True)›
⟨*proof*⟩

context
notes -[*simp*] = *VV'* *arena-lit-x1g-j*
assumes ‹*polarity* (*get-trail-wl* V') (*get-clauses-wl* V' ∝ *x1f* ! *j*) = *Some True*›
begin

lemma *update-blit-wl-heur-pre-unw*: ‹*update-blit-wl-heur-pre* *r*
((((((L, *x1f*), *x1f''*), *x2*), *x2a*), *get-clauses-wl* V' ∝ *x1f* ! *j*), V')›
⟨*proof*⟩

lemma *update-blit-unw-rel*:
‹(((((((L', *x1g*), *x2h*), *x2c*), *x2d*), *arena-lit* (*get-clauses-wl-heur* V) (*x1g* + *i*)),
V),
(((((L, *x1f*), *x2f*$''$), *x2*), *x2a*), *get-clauses-wl* $V' \propto$ *x1f* ! *j*), V')
$\in$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$
nat-lit-lit-rel $\times_f$
twl-st-heur-up$''$ $\mathcal{D}$ *r s K*›
⟨*proof*⟩

end

context
notes - [*simp*] = VV'
assumes ‹*polarity* (*get-trail-wl* V') (*get-clauses-wl* $V' \propto$ *x1f* ! *j*) $\neq$ *Some True*›
begin

private lemma *arena-is-valid-clause-idx-and-access-x1g-j*:
‹*arena-is-valid-clause-idx-and-access* (*get-clauses-wl-heur* V) *x1g j*›
⟨*proof*⟩ **lemma** *L-le*:
‹*nat-of-lit* L < *length* (*get-watched-wl-heur* V)›
‹*nat-of-lit* (*get-clauses-wl* $V' \propto$ *x1g* ! *j*) < *length* (*get-watched-wl-heur* V)›
⟨*proof*⟩ **lemma** *length-get-watched-wl-heur-U-T*:
‹*length* (*get-watched-wl-heur* U ! *nat-of-lit* L) = *length* (*get-watched-wl* T L)›
⟨*proof*⟩ **lemma** *length-get-watched-wl-heur-S-T*:
‹*length* (*watched-by-int* S L) = *length* (*get-watched-wl* T L)›
⟨*proof*⟩

lemma *update-clause-wl-code-pre-unw*: ‹*update-clause-wl-code-pre*
(((((((L', *x1g*), *x2h*), *x2c*), *x2d*),
if arena-lit (*get-clauses-wl-heur* U) (*x1g* + *0*) = L' *then 0 else 1*),
i),
V)›
⟨*proof*⟩ **lemma** *L-neq-j*:
‹$L \neq$ *get-clauses-wl* $T \propto$ *x1g* ! *j*›
⟨*proof*⟩

thm *corr-T*
find-theorems *S T*
find-theorems *correct-watching-except keep-watch*

private lemma *in-lall*: ‹*get-clauses-wl* $T \propto$ *x1g* ! *j*
$\in\#$ $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl* T) (*get-unit-clauses-wl* T))›
⟨*proof*⟩ **lemma** *length-le*: ‹*length* (*watched-by* T (*get-clauses-wl* $T \propto$ *x1g* ! *j*))
$\leq$ *length* (*get-clauses-wl-heur* S) − *4*›
⟨*proof*⟩

lemma *update-clause-wl-pre-unw*: ‹*update-clause-wl-pre K r*
(((((((L, *x1f*), *x1f*$''$), *x2*), *x2a*),
if get-clauses-wl (*keep-watch L x2 x2a T*) $\propto$ *x1f* ! *0* = L *then 0 else 1*),
j),
V')›
⟨*proof*⟩

lemma *update-watched-unw-rel*:
‹((((((((L', *x1g*), *x2h*), *x2c*), *x2d*),

if arena-lit (*get-clauses-wl-heur U*) (*x1g* + *0*) = *L'* *then 0 else 1*),
i),
V),
((((((*L*, *x1f*), *x2f''*), *x2*), *x2a*),
if get-clauses-wl (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0 else 1*),
j),
V')
∈ *Id* $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*›
⟨*proof*⟩

end

end

end

end

end

end

end

end

lemma *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D*:
‹(*uncurry3 unit-propagation-inner-loop-body-wl-heur*,
uncurry3 unit-propagation-inner-loop-body-wl-D)
∈ [λ(((*L*, *i*), *j*), *S*). *length* (*watched-by S L*) ≤ *r* − *4* ∧ *L* = *K* ∧
length (*watched-by S L*) = *s*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* →
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩*nres-rel*›
⟨*proof*⟩

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv* **where**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv* S_0 *L* =
(λ(*j*, *w*, *S'*). ∃ S_0' *S*. (S_0, S_0') ∈ *twl-st-heur* ∧ (*S'*, *S*) ∈ *twl-st-heur* ∧ *unit-propagation-inner-loop-wl-loop-D-inv* *L* (*j*, *w*, *S*) ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧ *dom-m* (*get-clauses-wl S*) = *dom-m* (*get-clauses-wl* S_0') ∧
length (*get-clauses-wl-heur* S_0) = *length* (*get-clauses-wl-heur S'*)))›

definition *unit-propagation-inner-loop-wl-loop-D-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ (*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where
‹*unit-propagation-inner-loop-wl-loop-D-heur L* S_0 = *do* {
ASSERT(*nat-of-lit L* < *length* (*get-watched-wl-heur* S_0));
ASSERT(*length* (*watched-by-int* S_0 *L*) ≤ *length* (*get-clauses-wl-heur* S_0));
let n = *length* (*watched-by-int* S_0 *L*);
$WHILE_T{}^{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv\ S_0\ L}$
(λ(*j*, *w*, *S*). *w* < *n* ∧ *get-conflict-wl-is-None-heur S*)
(λ(*j*, *w*, *S*). *do* {
unit-propagation-inner-loop-body-wl-heur L j w S
})

$(0, 0, S_0)$
}›

lemma *unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D*:
‹(*uncurry unit-propagation-inner-loop-wl-loop-D-heur*,
uncurry unit-propagation-inner-loop-wl-loop-D)
$\in [\lambda(L, S).\ length\ (watched\text{-}by\ S\ L) \le r - 4 \wedge L = K \wedge length\ (watched\text{-}by\ S\ L) = s \wedge$
$length\ (watched\text{-}by\ S\ L) \le r]_f$
$nat\text{-}lit\text{-}lit\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K \rightarrow$
$\langle nat\text{-}rel \times_r nat\text{-}rel \times_r twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *cut-watch-list-heur*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cut-watch-list-heur j w L* =(λ(*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
$ASSERT(j \le length\ (W!nat\text{-}of\text{-}lit\ L) \wedge j \le w \wedge nat\text{-}of\text{-}lit\ L < length\ W \wedge$
$w \le length\ (W\ !\ (nat\text{-}of\text{-}lit\ L)))$;
RETURN (*M*, *N*, *D*, *Q*,
$W[nat\text{-}of\text{-}lit\ L := take\ j\ (W!(nat\text{-}of\text{-}lit\ L))\ @\ drop\ w\ (W!(nat\text{-}of\text{-}lit\ L))]$, *oth*)
})›

definition *cut-watch-list-heur2*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cut-watch-list-heur2* = (λ*j w L* (*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
$ASSERT(j \le length\ (W\ !\ nat\text{-}of\text{-}lit\ L) \wedge j \le w \wedge nat\text{-}of\text{-}lit\ L < length\ W \wedge$
$w \le length\ (W\ !\ (nat\text{-}of\text{-}lit\ L)))$;
let $n = length\ (W!(nat\text{-}of\text{-}lit\ L))$;
$(j, w, W) \leftarrow WHILE_T{}^{\lambda(j, w, W).\ j \le w \wedge w \le n \wedge nat\text{-}of\text{-}lit\ L < length\ W}$
$(\lambda(j, w, W).\ w < n)$
$(\lambda(j, w, W).$ *do* {
$ASSERT(w < length\ (W!(nat\text{-}of\text{-}lit\ L)))$;
$RETURN\ (j+1, w+1, W[nat\text{-}of\text{-}lit\ L := (W!(nat\text{-}of\text{-}lit\ L))[j := W!(nat\text{-}of\text{-}lit\ L)!w]])$
})
(j, w, W);
$ASSERT(j \le length\ (W\ !\ nat\text{-}of\text{-}lit\ L) \wedge nat\text{-}of\text{-}lit\ L < length\ W)$;
let $W = W[nat\text{-}of\text{-}lit\ L := take\ j\ (W\ !\ nat\text{-}of\text{-}lit\ L)]$;
RETURN (*M*, *N*, *D*, *Q*, *W*, *oth*)
})›

lemma *cut-watch-list-heur2-cut-watch-list-heur*:
shows
‹*cut-watch-list-heur2 j w L S* ≤ ⇓ *Id* (*cut-watch-list-heur j w L S*)›
⟨*proof*⟩

lemma *vdom-m-cut-watch-list*:
‹$set\ xs \subseteq set\ (W\ L) \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (W(L := xs))\ d \subseteq vdom\text{-}m\ \mathcal{A}\ W\ d$›
⟨*proof*⟩

The following order allows the rule to be used as a destruction rule, make it more useful for refinement proofs.

lemma *vdom-m-cut-watch-listD*:

‹$x \in$ *vdom-m* $\mathcal{A}$ ($W(L := xs)$) $d \Longrightarrow$ *set xs* $\subseteq$ *set* (W L) $\Longrightarrow x \in$ *vdom-m* $\mathcal{A}$ W d›
⟨*proof*⟩

lemma *cut-watch-list-heur-cut-watch-list-heur*:
‹(*uncurry3 cut-watch-list-heur*, *uncurry3 cut-watch-list*) $\in$
$[\lambda(((j, w), L), S).\ L \in\#\ \mathcal{L}_{all}$ (*all-atms-st* S) $\wedge j \leq$ *length* (*watched-by* S L)$]_f$
nat-rel $\times_f$ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur''* $\mathcal{D}$ $r \rightarrow$ ⟨*twl-st-heur''* $\mathcal{D}$ r⟩*nres-rel*›
⟨*proof*⟩

definition *unit-propagation-inner-loop-wl-D-heur*
:: ‹*nat literal* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*› **where**
‹*unit-propagation-inner-loop-wl-D-heur* L S_0 = *do* {
  (j, w, S) $\leftarrow$ *unit-propagation-inner-loop-wl-loop-D-heur* L S_0;
  ASSERT(*length* (*watched-by-int* S L) $\leq$ *length* (*get-clauses-wl-heur* S_0) $-$ *4*);
  $S \leftarrow$ *cut-watch-list-heur2* j w L S;
  RETURN S
}›

lemma *unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D*:
‹(*uncurry unit-propagation-inner-loop-wl-D-heur*, *uncurry unit-propagation-inner-loop-wl-D*) $\in$
$[\lambda(L, S).$ *length*(*watched-by* S L) $\leq r-4]_f$
nat-lit-lit-rel $\times_f$ *twl-st-heur''* $\mathcal{D}$ $r \rightarrow$ ⟨*twl-st-heur''* $\mathcal{D}$ r⟩ *nres-rel*›
⟨*proof*⟩

definition *select-and-remove-from-literals-to-update-wl-heur*
:: ‹*twl-st-wl-heur* $\Rightarrow$ (*twl-st-wl-heur* $\times$ *nat literal*) *nres*›
where
‹*select-and-remove-from-literals-to-update-wl-heur* S = *do* {
  ASSERT(*literals-to-update-wl-heur* S < *length* (*fst* (*get-trail-wl-heur* S)));
  ASSERT(*literals-to-update-wl-heur* S + *1* $\leq$ *uint32-max*);
  $L \leftarrow$ *isa-trail-nth* (*get-trail-wl-heur* S) (*literals-to-update-wl-heur* S);
  RETURN (*set-literals-to-update-wl-heur* (*literals-to-update-wl-heur* S + *1*) S, $-L$)
}›

definition *unit-propagation-outer-loop-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *bool*›
where
‹*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S' $\longleftrightarrow$
  ($\exists\, S_0{}'$ S. (S_0, $S_0{}'$) $\in$ *twl-st-heur* $\wedge$ (S', S) $\in$ *twl-st-heur* $\wedge$
    unit-propagation-outer-loop-wl-D-inv S $\wedge$
    dom-m (*get-clauses-wl* S) = *dom-m* (*get-clauses-wl* $S_0{}'$) $\wedge$
    length (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* S_0) $\wedge$
    isa-length-trail-pre (*get-trail-wl-heur* S'))›

definition *unit-propagation-outer-loop-wl-D-heur*
:: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*› **where**
‹*unit-propagation-outer-loop-wl-D-heur* S_0 =
  $WHILE_T{}^{\text{unit-propagation-outer-loop-wl-D-heur-inv } S_0}$
   (λS. *literals-to-update-wl-heur* S < *isa-length-trail* (*get-trail-wl-heur* S))
   (λS. *do* {
     ASSERT(*literals-to-update-wl-heur* S < *isa-length-trail* (*get-trail-wl-heur* S));
     (S', L) $\leftarrow$ *select-and-remove-from-literals-to-update-wl-heur* S;
     ASSERT(*length* (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* S));
     unit-propagation-inner-loop-wl-D-heur L S'

})
S_0›

lemma *select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl*:
‹*literals-to-update-wl y* $\neq$ {#} $\wedge$ *length* (*get-trail-wl y*) < *uint-max* $\Longrightarrow$
(*x*, *y*) $\in$ *twl-st-heur''* $\mathcal{D}$*1 r1* $\Longrightarrow$
select-and-remove-from-literals-to-update-wl-heur x
$\leq$ $\Downarrow${((*S*, *L*), (*S'*, *L'*)). ((*S*, *L*), (*S'*, *L'*)) $\in$ *twl-st-heur''* $\mathcal{D}$*1 r1* $\times_f$ *nat-lit-lit-rel* $\wedge$
S' = *set-literals-to-update-wl* (*literals-to-update-wl y* − {#*L*#}) *y* $\wedge$
get-clauses-wl-heur S = *get-clauses-wl-heur x*}
(*select-and-remove-from-literals-to-update-wl y*)›
⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-heur-inv-length-trail-le*:
assumes
‹(*S*, *T*) $\in$ *twl-st-heur''* $\mathcal{D}$ *r*›
‹(*U*, *V*) $\in$ *twl-st-heur''* $\mathcal{D}$ *r*› **and**
‹*literals-to-update-wl-heur U* < *isa-length-trail* (*get-trail-wl-heur U*)› **and**
‹*literals-to-update-wl V* $\neq$ {#}› **and**
‹*unit-propagation-outer-loop-wl-D-heur-inv S U*› **and**
‹*unit-propagation-outer-loop-wl-D-inv V*› **and**
‹*literals-to-update-wl V* $\neq$ {#}› **and**
‹*literals-to-update-wl-heur U* < *isa-length-trail* (*get-trail-wl-heur U*)›
shows ‹*length* (*get-trail-wl V*) < *uint-max*›
⟨*proof*⟩

lemma *outer-loop-length-watched-le-length-arena*:
assumes
xa-x': ‹(*xa*, *x'*) $\in$ *twl-st-heur''* $\mathcal{D}$ *r*› **and**
prop-heur-inv: ‹*unit-propagation-outer-loop-wl-D-heur-inv x xa*› **and**
prop-inv: ‹*unit-propagation-outer-loop-wl-D-inv x'*› **and**
xb-x'a: ‹(*xb*, *x'a*) $\in$ {((*S*, *L*), (*S'*, *L'*)). ((*S*, *L*), (*S'*, *L'*)) $\in$ *twl-st-heur''* $\mathcal{D}$*1 r* $\times_f$ *nat-lit-lit-rel* $\wedge$
S' = *set-literals-to-update-wl* (*literals-to-update-wl x'* − {#*L*#}) *x'* $\wedge$
get-clauses-wl-heur S = *get-clauses-wl-heur xa*}› **and**
st: ‹*x'a* = (*x1*, *x2*)›
‹*xb* = (*x1a*, *x2a*)› **and**
x2: ‹*x2* $\in$# $\mathcal{L}_{all}$ (*all-atms-st x'*)› **and**
st': ‹(*x2*, *x1*) = (*x1b*, *x2b*)›
shows ‹*length* (*watched-by x2b x1b*) $\leq$ *r*−*4*›
⟨*proof*⟩

theorem *unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'*:
‹(*unit-propagation-outer-loop-wl-D-heur*, *unit-propagation-outer-loop-wl-D*) $\in$
twl-st-heur'' $\mathcal{D}$ *r* $\to_f$ ⟨*twl-st-heur''* $\mathcal{D}$ *r*⟩ *nres-rel*›
⟨*proof*⟩

lemma *twl-st-heur'D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge\mathcal{D}$. *f* $\in$ *twl-st-heur'* $\mathcal{D}$ $\to_f$ ⟨*twl-st-heur'* $\mathcal{D}$⟩ *nres-rel*)›
shows ‹*f* $\in$ *twl-st-heur* $\to_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- $\in$ *?A B*›)
⟨*proof*⟩

lemma *watched-by-app-watched-by-app-heur*:
‹(*uncurry2* (*RETURN ooo watched-by-app-heur*), *uncurry2* (*RETURN ooo watched-by-app*)) $\in$
[λ((*S*, *L*), *K*). *L* $\in$# $\mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$ *K* < *length* (*get-watched-wl S L*)]$_f$
twl-st-heur $\times_f$ *Id* $\times_f$ *Id* $\to$ ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *case-tri-bool-If*:
‹(*case a of*
None ⇒ *f1*
| *Some v* ⇒
(*if v then f2 else f3*)) =
(*let b = a in if b = UNSET*
then f1
else if b = SET-TRUE then f2 else f3)›
⟨*proof*⟩

definition *isa-find-unset-lit* :: ‹*trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*isa-find-unset-lit M = isa-find-unwatched-between* (λ*L. polarity-pol M L* ≠ *Some False*) *M*›
lemma *update-clause-wl-heur-pre-le-uint64*:
assumes
‹*arena-is-valid-clause-idx-and-access a1'a bf baa*› **and**
‹*length* (*get-clauses-wl-heur*
(*a1'*, *a1'a*, (*da*, *db*, *dc*), *a1'c*, *a1'd*, ((*eu*, *ev*, *ew*, *ex*, *ey*), *ez*), *fa*, *fb*,
fc, *fd*, *fe*, (*ff*, *fg*, *fh*, *fi*), *fj*, *fk*, *fl*, *fm*, *fn*)) ≤ *uint64-max*› **and**
‹*arena-lit-pre a1'a* (*bf* + *baa*)›
shows ‹*bf* + *baa* ≤ *uint64-max*›
‹*length a1'a* ≤ *uint64-max*›
⟨*proof*⟩

lemma *clause-not-marked-to-delete-heur-alt-def*:
‹*RETURN* ∘∘ *clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*.
RETURN (*arena-status arena C* ≠ *DELETED*))›
⟨*proof*⟩

end
theory *IsaSAT-Inner-Propagation-SML*
imports *IsaSAT-Setup-SML*
IsaSAT-Inner-Propagation
begin
sepref-register *isa-save-pos*
sepref-definition *isa-save-pos-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹$nat\text{-}assn^k *_a nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *isa-save-pos-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-save-pos-fast-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *isa-save-pos-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *watched-by-app-heur-code*
is ‹*uncurry2* (*RETURN ooo watched-by-app-heur*)›
:: ‹$[watched\text{-}by\text{-}app\text{-}heur\text{-}pre]_a$
$isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k \rightarrow watcher\text{-}assn$›
⟨*proof*⟩

declare *watched-by-app-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *watched-by-app-heur-fast-code*
  is ‹*uncurry2* (*RETURN ooo watched-by-app-heur*)›
  :: ‹[*watched-by-app-heur-pre*]$_a$
      isasat-bounded-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *watcher-fast-assn*›
  ⟨*proof*⟩

declare *watched-by-app-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *isa-find-unwatched-wl-st-heur isa-find-unwatched-between isa-find-unset-lit*

sepref-definition *isa-find-unwatched-between-code*
  is ‹*uncurry4 isa-find-unset-lit*›
  :: ‹*trail-pol-assn*^{k} $*_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow_a$
      option-assn nat-assn›
  ⟨*proof*⟩

declare *isa-find-unwatched-between-code.refine*[*sepref-fr-rules*]

sepref-register *polarity-pol arena-length nat-of-uint64-conv*

sepref-definition *find-unwatched-wl-st-heur-code*
  is ‹*uncurry isa-find-unwatched-wl-st-heur*›
  :: ‹[*find-unwatched-wl-st-heur-pre*]$_a$
      isasat-unbounded-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *option-assn nat-assn*›
  ⟨*proof*⟩

declare *find-unwatched-wl-st-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-find-unwatched-between-fast-code*
  is ‹*uncurry4 isa-find-unset-lit*›
  :: ‹[$\lambda((((M, N), -), -), -)$. *length* $N \leq$ *uint64-max*]$_a$
    trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k}
$\rightarrow$
      option-assn uint64-nat-assn›
  ⟨*proof*⟩

declare *isa-find-unwatched-between-fast-code.refine*[*sepref-fr-rules*]

declare *get-saved-pos-code*[*sepref-fr-rules*]

sepref-definition *find-unwatched-wl-st-heur-fast-code*
  is ‹*uncurry isa-find-unwatched-wl-st-heur*›
  :: ‹[($\lambda(S, C)$. *find-unwatched-wl-st-heur-pre* $(S, C) \wedge$
          length (*get-clauses-wl-heur* S) $\leq$ *uint64-max*)]$_a$
      isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *option-assn uint64-nat-assn*›
  ⟨*proof*⟩

declare *find-unwatched-wl-st-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *update-clause-wl-heur*
sepref-definition *update-clause-wl-code*
is ‹*uncurry7 update-clause-wl-heur*›
:: ‹[*update-clause-wl-code-pre*]$_a$
unat-lit-assn^k $*_a$ *nat-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k}
$*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *nat-assn* $*a$ *nat-assn* $*a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *update-clause-wl-code.refine*[*sepref-fr-rules*]

sepref-definition *update-clause-wl-fast-code*
is ‹*uncurry7 update-clause-wl-heur*›
:: ‹[$\lambda(((((((L, C), b), j), w), i), f), S)$. *update-clause-wl-code-pre* $(((((((L, C), b), j), w), i), f), S) \wedge$
length (*get-clauses-wl-heur* S) $\leq$ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k}
$*_a$
uint64-nat-assn^k
$*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *uint64-nat-assn* $*a$ *uint64-nat-assn* $*a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *update-clause-wl-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-lit-wl-code*
is ‹*uncurry3 propagate-lit-wl-heur*›
:: ‹[*propagate-lit-wl-heur-pre*]$_a$
unat-lit-assn^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *propagate-lit-wl-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-lit-wl-fast-code*
is ‹*uncurry3 propagate-lit-wl-heur*›
:: ‹[$\lambda(((L, C), i), S)$. *propagate-lit-wl-heur-pre*$(((L, C), i), S) \wedge$
length (*get-clauses-wl-heur* S) $\leq$ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *propagate-lit-wl-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-lit-wl-bin-code*
is ‹*uncurry3 propagate-lit-wl-bin-heur*›
:: ‹[*propagate-lit-wl-heur-pre*]$_a$
unat-lit-assn^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *propagate-lit-wl-bin-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-lit-wl-bin-fast-code*
is ‹*uncurry3 propagate-lit-wl-bin-heur*›
:: ‹[$\lambda(((L, C), i), S)$. *propagate-lit-wl-heur-pre*$(((L, C), i), S) \wedge$
length (*get-clauses-wl-heur* S) $\leq$ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$
isasat-bounded-assn›
⟨*proof*⟩

declare *propagate-lit-wl-bin-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *clause-not-marked-to-delete-heur-code*
is ‹*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*)›
:: ‹[*clause-not-marked-to-delete-heur-pre*]$_a$ *isasat-unbounded-assn*^{k} $*_a$ *nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

declare *clause-not-marked-to-delete-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *clause-not-marked-to-delete-heur-fast-code*
is ‹*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*)›
:: ‹[*clause-not-marked-to-delete-heur-pre*]$_a$ *isasat-bounded-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

declare *clause-not-marked-to-delete-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *update-blit-wl-heur-code*
is ‹*uncurry6 update-blit-wl-heur*›
:: ‹
unat-lit-assn^k $*_a$ *nat-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d}
$\rightarrow_a$
nat-assn $*a$ *nat-assn* $*a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *update-blit-wl-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *update-blit-wl-heur-fast-code*
is ‹*uncurry6 update-blit-wl-heur*›
:: ‹[λ((((((-, -), -), -), *C*), *i*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *bool-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$
unat-lit-assn^k $*_a$
isasat-bounded-assn^d →
uint64-nat-assn $*a$ *uint64-nat-assn* $*a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *update-blit-wl-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *keep-watch-heur*

sepref-definition *keep-watch-heur-code*
is ‹*uncurry3 keep-watch-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *keep-watch-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *keep-watch-heur-fast-code*
is ‹*uncurry3 keep-watch-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *keep-watch-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *isa-set-lookup-conflict-aa set-conflict-wl-heur*
sepref-definition *set-conflict-wl-heur-code*

is ‹*uncurry set-conflict-wl-heur*›
:: ‹[*set-conflict-wl-heur-pre*]$_a$
nat-assn^k $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *set-conflict-wl-heur-code.refine*[*sepref-fr-rules*]

sepref-register *arena-incr-act*

sepref-definition *set-conflict-wl-heur-fast-code*
is ‹*uncurry set-conflict-wl-heur*›
:: ‹[λ(C, S). *set-conflict-wl-heur-pre* (C, S) $\wedge$
length (*get-clauses-wl-heur* S) $\leq$ *uint64-max*]$_a$
uint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *set-conflict-wl-heur-fast-code.refine*[*sepref-fr-rules*]

Find a less hack-like solution

setup ‹*map-theory-claset* (*fn ctxt* => *ctxt delSWrapper split-all-tac*)›

sepref-register *update-blit-wl-heur clause-not-marked-to-delete-heur*
sepref-definition *unit-propagation-inner-loop-body-wl-heur-code*
is ‹*uncurry3 unit-propagation-inner-loop-body-wl-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *nat-assn* $*a$ *nat-assn* $*a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *unit-propagation-inner-loop-body-wl-fast-heur-code*
is ‹*uncurry3 unit-propagation-inner-loop-body-wl-heur*›
:: ‹[λ((L, w), S). *length* (*get-clauses-wl-heur* S) $\leq$ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$
uint64-nat-assn $*a$ *uint64-nat-assn* $*a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-register *unit-propagation-inner-loop-body-wl-heur*

declare *unit-propagation-inner-loop-body-wl-heur-code.refine*[*sepref-fr-rules*]
unit-propagation-inner-loop-body-wl-fast-heur-code.refine[*sepref-fr-rules*]
declare [[*show-types*]]
thm *unit-propagation-inner-loop-body-wl-fast-heur-code-def*
end
theory *IsaSAT-VMTF*
imports *Watched-Literals.WB-Sort IsaSAT-Setup*
begin

0.1.17 Code generation for the VMTF decision heuristic and the trail

definition *size-conflict-wl* :: ‹*nat twl-st-wl* $\Rightarrow$ *nat*› **where**
‹*size-conflict-wl* S = *size* (*the* (*get-conflict-wl* S))›

definition *size-conflict* :: ‹*nat clause option* $\Rightarrow$ *nat*› **where**
‹*size-conflict* D = *size* (*the* D)›

definition *size-conflict-int* :: ‹*conflict-option-rel* $\Rightarrow$ *nat*› **where**
‹*size-conflict-int* = (λ(-, n, -). n)›

definition *update-next-search* **where**
‹$update\text{-}next\text{-}search\ L = (\lambda((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove).$
$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ L),\ to\text{-}remove))$›

definition *vmtf-enqueue-pre* **where**
‹$vmtf\text{-}enqueue\text{-}pre =$
$(\lambda((M,\ L),(ns,m,fst\text{-}As,lst\text{-}As,\ next\text{-}search)).\ L < length\ ns \wedge$
$(fst\text{-}As \neq None \longrightarrow the\ fst\text{-}As < length\ ns) \wedge$
$(fst\text{-}As \neq None \longrightarrow lst\text{-}As \neq None) \wedge$
$m+1 \leq uint64\text{-}max)$›

definition *isa-vmtf-enqueue* :: ‹$trail\text{-}pol \Rightarrow nat \Rightarrow vmtf\text{-}option\text{-}fst\text{-}As \Rightarrow vmtf\ nres$› **where**
‹$isa\text{-}vmtf\text{-}enqueue = (\lambda M\ L\ (ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search).\ do\ \{$
$ASSERT(defined\text{-}atm\text{-}pol\text{-}pre\ M\ L);$
$de \leftarrow RETURN\ (defined\text{-}atm\text{-}pol\ M\ L);$
$RETURN\ (case\ fst\text{-}As\ of$
$None \Rightarrow (ns[L := VMTF\text{-}Node\ m\ fst\text{-}As\ None],\ m+1,\ L,\ L,$
$(if\ de\ then\ None\ else\ Some\ L))$
$|\ Some\ fst\text{-}As \Rightarrow$
$let\ fst\text{-}As' = VMTF\text{-}Node\ (stamp\ (ns!fst\text{-}As))\ (Some\ L)\ (get\text{-}next\ (ns!fst\text{-}As))\ in$
$(ns[L := VMTF\text{-}Node\ (m+1)\ None\ (Some\ fst\text{-}As),\ fst\text{-}As := fst\text{-}As'],$
$m+1,\ L,\ the\ lst\text{-}As,\ (if\ de\ then\ next\text{-}search\ else\ Some\ L)))\})$›

lemma *vmtf-enqueue-alt-def*:
‹$RETURN\ ooo\ vmtf\text{-}enqueue = (\lambda M\ L\ (ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search).\ do\ \{$
$let\ de = defined\text{-}lit\ M\ (Pos\ L);$
$RETURN\ (case\ fst\text{-}As\ of$
$None \Rightarrow (ns[L := VMTF\text{-}Node\ m\ fst\text{-}As\ None],\ m+1,\ L,\ L,$
$(if\ de\ then\ None\ else\ Some\ L))$
$|\ Some\ fst\text{-}As \Rightarrow$
$let\ fst\text{-}As' = VMTF\text{-}Node\ (stamp\ (ns!fst\text{-}As))\ (Some\ L)\ (get\text{-}next\ (ns!fst\text{-}As))\ in$
$(ns[L := VMTF\text{-}Node\ (m+1)\ None\ (Some\ fst\text{-}As),\ fst\text{-}As := fst\text{-}As'],$
$m+1,\ L,\ the\ lst\text{-}As,\ (if\ de\ then\ next\text{-}search\ else\ Some\ L)))\})$›
⟨*proof*⟩

lemma *isa-vmtf-enqueue*:
‹$(uncurry2\ isa\text{-}vmtf\text{-}enqueue,\ uncurry2\ (RETURN\ ooo\ vmtf\text{-}enqueue)) \in$
$[\lambda((M,\ L),\ \text{-}).\ L \in\#\ \mathcal{A}]_f\ (trail\text{-}pol\ \mathcal{A}) \times_f nat\text{-}rel \times_f Id \rightarrow \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *partition-vmtf-nth* :: ‹$nat\text{-}vmtf\text{-}node\ list \Rightarrow nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow (nat\ list \times nat)\ nres$›
where
‹$partition\text{-}vmtf\text{-}nth\ ns = partition\text{-}main\ (\leq)\ (\lambda n.\ stamp\ (ns\ !\ n))$›

definition *partition-between-ref-vmtf* :: ‹$nat\text{-}vmtf\text{-}node\ list \Rightarrow nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow (nat\ list \times nat)\ nres$› **where**
‹$partition\text{-}between\text{-}ref\text{-}vmtf\ ns = partition\text{-}between\text{-}ref\ (\leq)\ (\lambda n.\ stamp\ (ns\ !\ n))$›

definition *quicksort-vmtf-nth* :: ‹$nat\text{-}vmtf\text{-}node\ list \times 'c \Rightarrow nat\ list \Rightarrow nat\ list\ nres$› **where**
‹$quicksort\text{-}vmtf\text{-}nth = (\lambda(ns,\ \text{-}).\ full\text{-}quicksort\text{-}ref\ (\leq)\ (\lambda n.\ stamp\ (ns\ !\ n)))$›

definition *quicksort-vmtf-nth-ref*:: ‹$nat\text{-}vmtf\text{-}node\ list \Rightarrow nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow nat\ list\ nres$› **where**
‹$quicksort\text{-}vmtf\text{-}nth\text{-}ref\ ns\ a\ b\ c =$
$quicksort\text{-}ref\ (\leq)\ (\lambda n.\ stamp\ (ns\ !\ n))\ (a,\ b,\ c)$›

lemma (**in** −) *partition-vmtf-nth-code-helper*:
assumes ‹$\forall x{\in}set\ ba.\ x < length\ a$› **and**
‹$b < length\ ba$› **and**
mset: ‹$mset\ ba = mset\ a2'$› **and**
‹$a1' < length\ a2'$›
shows ‹$a2' \,!\, b < length\ a$›
⟨*proof*⟩

lemma *partition-vmtf-nth-code-helper2*:
‹$ba < length\ b \Longrightarrow (bia,\ ba) \in uint32\text{-}nat\text{-}rel \Longrightarrow$
$(aa,\ (ba - bb)\ div\ 2) \in uint32\text{-}nat\text{-}rel \Longrightarrow$
$(ab,\ bb) \in uint32\text{-}nat\text{-}rel \Longrightarrow bb + (ba - bb)\ div\ 2 \leq uint\text{-}max$›
⟨*proof*⟩

lemma *partition-vmtf-nth-code-helper3*:
‹$\forall x{\in}set\ b.\ x < length\ a \Longrightarrow$
$x'e < length\ a2' \Longrightarrow$
$mset\ a2' = mset\ b \Longrightarrow$
$a2' \,!\, x'e < length\ a$›
⟨*proof*⟩

definition (**in** −) *isa-vmtf-en-dequeue* :: ‹$trail\text{-}pol \Rightarrow nat \Rightarrow vmtf \Rightarrow vmtf\ nres$› **where**
‹$isa\text{-}vmtf\text{-}en\text{-}dequeue = (\lambda M\ L\ vm.\ isa\text{-}vmtf\text{-}enqueue\ M\ L\ (vmtf\text{-}dequeue\ L\ vm))$›

lemma *isa-vmtf-en-dequeue*:
‹$(uncurry2\ isa\text{-}vmtf\text{-}en\text{-}dequeue,\ uncurry2\ (RETURN\ ooo\ vmtf\text{-}en\text{-}dequeue)) \in$
$[\lambda((M,\ L),\ \text{-}).\ L \in\#\ \mathcal{A}]_f\ (trail\text{-}pol\ \mathcal{A}) \times_f nat\text{-}rel \times_f Id \to \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *isa-vmtf-en-dequeue-pre* :: ‹$(trail\text{-}pol \times nat) \times vmtf \Rightarrow bool$› **where**
‹$isa\text{-}vmtf\text{-}en\text{-}dequeue\text{-}pre = (\lambda((M,\ L),(ns,m,fst\text{-}As,\ lst\text{-}As,\ next\text{-}search)).$
$L < length\ ns \wedge vmtf\text{-}dequeue\text{-}pre\ (L,\ ns) \wedge$
$fst\text{-}As < length\ ns \wedge (get\text{-}next\ (ns\ !\ fst\text{-}As) \neq None \longrightarrow get\text{-}prev\ (ns\ !\ lst\text{-}As) \neq None) \wedge$
$(get\text{-}next\ (ns\ !\ fst\text{-}As) = None \longrightarrow fst\text{-}As = lst\text{-}As) \wedge$
$m{+}1 \leq uint64\text{-}max)$›

lemma *isa-vmtf-en-dequeue-preD*:
assumes ‹$isa\text{-}vmtf\text{-}en\text{-}dequeue\text{-}pre\ ((M,\ ah),\ a,\ aa,\ ab,\ ac,\ b)$›
shows ‹$ah < length\ a$› **and** ‹$vmtf\text{-}dequeue\text{-}pre\ (ah,\ a)$›
⟨*proof*⟩

lemma *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*:
‹$isa\text{-}vmtf\text{-}en\text{-}dequeue\text{-}pre\ ((M,\ L),\ a,\ st,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search) \Longrightarrow$
$vmtf\text{-}enqueue\text{-}pre\ ((M,\ L),\ vmtf\text{-}dequeue\ L\ (a,\ st,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search))$›
⟨*proof*⟩

lemma *insert-sort-reorder-list*:
assumes *trans*: ‹$\bigwedge\ x\ y\ z.\ [\![R\ (h\ x)\ (h\ y);\ R\ (h\ y)\ (h\ z)]\!] \Longrightarrow R\ (h\ x)\ (h\ z)$› **and** *lin*: ‹$\bigwedge x\ y.\ R\ (h\ x)\ (h\ y) \vee R\ (h\ y)\ (h\ x)$›
shows ‹$(full\text{-}quicksort\text{-}ref\ R\ h,\ reorder\text{-}list\ vm) \in \langle Id\rangle list\text{-}rel \to_f \langle Id\rangle\ nres\text{-}rel$›
⟨*proof*⟩

lemma *quicksort-vmtf-nth-reorder*:
‹$(uncurry\ quicksort\text{-}vmtf\text{-}nth,\ uncurry\ reorder\text{-}list) \in$

$Id \times_r \langle Id\rangle list\text{-}rel \rightarrow_f \langle Id\rangle\ nres\text{-}rel$›
⟨*proof*⟩

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) ∈
nat-rel $\times_r$ *atoms-hash-rel* $\mathcal{A} \rightarrow_f$ ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›
⟨*proof*⟩

lemma *vmtf-rescale-alt-def*:
‹*vmtf-rescale* = (λ(*ns*, *m*, *fst-As*, *lst-As* :: *nat*, *next-search*). *do* {
(*ns*, *m*, -) ← $WHILE_T{}^{\lambda\text{-}.\ True}$
(λ(*ns*, *n*, *lst-As*). *lst-As* ≠*None*)
(λ(*ns*, *n*, *a*). *do* {
ASSERT(*a* ≠ *None*);
ASSERT(*n*+*1* ≤ *uint32-max*);
ASSERT(*the a* < *length ns*);
let m = *the a*;
let c = *ns* ! *m*;
let nc = *get-next c*;
let pc = *get-prev c*;
RETURN (*ns*[*m* := *VMTF-Node n pc nc*], *n* + *1*, *pc*)
})
(*ns*, *0*, *Some lst-As*);
RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))
})›
⟨*proof*⟩

definition *isa-vmtf-flush-int* :: ‹*trail-pol* ⇒ - ⇒ - *nres*› **where**
‹*isa-vmtf-flush-int* = (λ*M* (*vm*, (*to-remove*, *h*)). *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length* (*fst vm*));
ASSERT(*length to-remove* ≤ *uint32-max*);
to-remove′ ← *reorder-list vm to-remove*;
ASSERT(*length to-remove*′ ≤ *uint32-max*);
vm ← (*if length to-remove*′ ≥ *uint64-max* − *fst* (*snd vm*)
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove*′ + *fst* (*snd vm*) ≤ *uint64-max*);
(-, *vm*, *h*) ← $WHILE_T{}^{\lambda(i,\ vm',\ h).\ i \le length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge}$ (*i* < *length to-remove*
(λ(*i*, *vm*, *h*). *i* < *length to-remove*′)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove*′);
ASSERT(*isa-vmtf-en-dequeue-pre* ((*M*, *to-remove*′!*i*), *vm*));
vm ← *isa-vmtf-en-dequeue M* (*to-remove*′!*i*) *vm*;
ASSERT(*atoms-hash-del-pre* (*to-remove*′!*i*) *h*);
RETURN (*i*+*1*, *vm*, *atoms-hash-del* (*to-remove*′!*i*) *h*)})
(*0*, *vm*, *h*);

$RETURN\ (vm,\ (emptied\text{-}list\ to\text{-}remove',\ h))$
})›

lemma *isa-vmtf-flush-int*:
‹$(uncurry\ isa\text{-}vmtf\text{-}flush\text{-}int,\ uncurry\ (vmtf\text{-}flush\text{-}int\ \mathcal{A})) \in trail\text{-}pol\ \mathcal{A} \times_f Id \rightarrow_f \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *atms-hash-insert-pre* :: ‹$nat \Rightarrow nat\ list \times bool\ list \Rightarrow bool$› **where**
‹$atms\text{-}hash\text{-}insert\text{-}pre\ i = (\lambda(n,\ xs).\ i < length\ xs \wedge (\neg xs!i \longrightarrow length\ n < uint32\text{-}max))$›

definition *atoms-hash-insert* :: ‹$nat \Rightarrow nat\ list \times bool\ list \Rightarrow (nat\ list \times bool\ list)$› **where**
‹$atoms\text{-}hash\text{-}insert\ i\ = (\lambda(n,\ xs).\ if\ xs\ !\ i\ then\ (n,\ xs)\ else\ (n\ @\ [i],\ xs[i := True]))$›

lemma *bounded-included-le*:
assumes *bounded*: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}$› **and** ‹$distinct\ n$› **and** ‹$set\ n \subseteq set\text{-}mset\ \mathcal{A}$ › **shows** ‹$length\ n < uint32\text{-}max$›
⟨*proof*⟩

lemma *atms-hash-insert-pre*:
assumes ‹$L \in\# \mathcal{A}$› **and** ‹$(x,\ x') \in distinct\text{-}atoms\text{-}rel\ \mathcal{A}$› **and** ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}$›
shows ‹$atms\text{-}hash\text{-}insert\text{-}pre\ L\ x$›
⟨*proof*⟩

lemma *atoms-hash-del-op-set-insert*:
‹$(uncurry\ (RETURN\ oo\ atoms\text{-}hash\text{-}insert),$
$uncurry\ (RETURN\ oo\ insert)) \in$
$[\lambda(i,\ xs).\ i \in\# \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}]_f$
$nat\text{-}rel \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in} \rightarrow \langle distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}\rangle nres\text{-}rel$›
⟨*proof*⟩

definition (**in** −) *atoms-hash-set-member* **where**
‹$atoms\text{-}hash\text{-}set\text{-}member\ i\ xs =\ \ do\ \{ASSERT(i < length\ xs);\ RETURN\ (xs\ !\ i)\}$›

definition *isa-vmtf-mark-to-rescore*
:: ‹$nat \Rightarrow isa\text{-}vmtf\text{-}remove\text{-}int \Rightarrow isa\text{-}vmtf\text{-}remove\text{-}int$›
where
‹$isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\ L = (\lambda((ns,\ m,\ fst\text{-}As,\ next\text{-}search),\ to\text{-}remove).$
$((ns,\ m,\ fst\text{-}As,\ next\text{-}search),\ atoms\text{-}hash\text{-}insert\ L\ to\text{-}remove))$›

definition *isa-vmtf-mark-to-rescore-pre* **where**
‹$isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}pre = (\lambda L\ ((ns,\ m,\ fst\text{-}As,\ next\text{-}search),\ to\text{-}remove).$
$atms\text{-}hash\text{-}insert\text{-}pre\ L\ to\text{-}remove)$›

lemma *isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*:
‹$(uncurry\ (RETURN\ oo\ isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore),\ uncurry\ (RETURN\ oo\ vmtf\text{-}mark\text{-}to\text{-}rescore)) \in$
$[\lambda(L,\ vm).\ L{\in\#}\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}]_f\ Id \times_f (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}) \rightarrow$
$\langle Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}\rangle nres\text{-}rel$›
⟨*proof*⟩

definition (**in** −) *isa-vmtf-unset* :: ‹$nat \Rightarrow isa\text{-}vmtf\text{-}remove\text{-}int \Rightarrow isa\text{-}vmtf\text{-}remove\text{-}int$› **where**

‹*isa-vmtf-unset* = (λL ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
  (*if next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)
  then ((*ns*, *m*, *fst-As*, *lst-As*, *Some L*), *to-remove*)
  else ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)))›

definition *vmtf-unset-pre* **where**
‹*vmtf-unset-pre* = (λL ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
  L < *length ns* ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length ns*))›

lemma *vmtf-unset-pre-vmtf*:
  assumes
    ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
    ‹*L* ∈# $\mathcal{A}$›
  shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
  ⟨*proof*⟩

lemma *vmtf-unset-pre*:
  assumes
    ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
    ‹*L* ∈# $\mathcal{A}$›
  shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
  ⟨*proof*⟩

lemma *vmtf-unset-pre'*:
  assumes
    ‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
    ‹*L* ∈# $\mathcal{A}$›
  shows ‹*vmtf-unset-pre L vm*›
  ⟨*proof*⟩

definition *isa-vmtf-mark-to-rescore-and-unset* :: ‹*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*›
where
  ‹*isa-vmtf-mark-to-rescore-and-unset L M* = *isa-vmtf-mark-to-rescore L* (*isa-vmtf-unset L M*)›

definition *isa-vmtf-mark-to-rescore-and-unset-pre* **where**
  ‹*isa-vmtf-mark-to-rescore-and-unset-pre* = (λ(*L*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*)).
      vmtf-unset-pre L ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*) ∧
      atms-hash-insert-pre L tor)›

definition *get-pos-of-level-in-trail* **where**
  ‹*get-pos-of-level-in-trail* M_0 *lev* =
     SPEC(λi. *i* < *length* M_0 ∧ *is-decided* (*rev* M_0!*i*) ∧ *get-level* M_0 (*lit-of* (*rev* M_0!*i*)) = *lev*+*1*)›

definition (**in** −) *get-pos-of-level-in-trail-imp* **where**
  ‹*get-pos-of-level-in-trail-imp* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*) *lev*. *do* {
      ASSERT(*lev* < *length cs*);
      RETURN (*cs* ! *lev*)
   })›

lemma *control-stack-is-decided*:
  ‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *is-decided* ((*rev M*)!*c*)›
  ⟨*proof*⟩

lemma *control-stack-distinct*:
  ‹*control-stack cs M* ⟹ *distinct cs*›

⟨*proof*⟩

lemma *control-stack-level-control-stack*:
assumes
cs: ‹*control-stack cs M*› **and**
n-d: ‹*no-dup M*› **and**
i: ‹$i < length\ cs$›
shows ‹$get\text{-}level\ M\ (lit\text{-}of\ (rev\ M\ !\ (cs\ !\ i))) = Suc\ i$›
⟨*proof*⟩

definition *get-pos-of-level-in-trail-pre* **where**
‹$get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}pre = (\lambda(M, lev).\ lev < count\text{-}decided\ M)$›

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail*:
‹$(uncurry\ get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}imp,\ uncurry\ get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail) \in$
$[get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}pre]_f\ trail\text{-}pol\text{-}no\text{-}CS\ \mathcal{A} \times_f nat\text{-}rel \rightarrow \langle nat\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*:
‹$(uncurry\ get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}imp,\ uncurry\ get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail) \in$
$[get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}pre]_f\ trail\text{-}pol\ \mathcal{A} \times_f nat\text{-}rel \rightarrow \langle nat\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *lit-of-last-trail-pol-lit-of-last-trail-no-CS*:
‹$(RETURN\ o\ lit\text{-}of\text{-}last\text{-}trail\text{-}pol,\ RETURN\ o\ lit\text{-}of\text{-}hd\text{-}trail) \in$
$[\lambda S.\ S \neq []]_f\ trail\text{-}pol\text{-}no\text{-}CS\ \mathcal{A} \rightarrow \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *size-conflict-int-size-conflict*:
‹$(RETURN\ o\ size\text{-}conflict\text{-}int,\ RETURN\ o\ size\text{-}conflict) \in [\lambda D.\ D \neq None]_f\ option\text{-}lookup\text{-}clause\text{-}rel$
$\mathcal{A} \rightarrow$
$\langle nat\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *rescore-clause*
:: ‹$nat\ multiset \Rightarrow nat\ clause\text{-}l \Rightarrow (nat,nat)ann\text{-}lits \Rightarrow vmtf\text{-}remove\text{-}int \Rightarrow phase\text{-}saver \Rightarrow$
$(vmtf\text{-}remove\text{-}int \times phase\text{-}saver)\ nres$›
where
‹$rescore\text{-}clause\ \mathcal{A}\ C\ M\ vm\ \varphi = SPEC\ (\lambda(vm', \varphi' :: bool\ list).\ vm' \in vmtf\ \mathcal{A}\ M \wedge phase\text{-}saving\ \mathcal{A}\ \varphi')$›

definition *find-decomp-w-ns-pre* **where**
‹$find\text{-}decomp\text{-}w\text{-}ns\text{-}pre\ \mathcal{A} = (\lambda((M, highest), vm).$
$no\text{-}dup\ M \wedge$
$highest < count\text{-}decided\ M \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ \mathcal{A}\ M \wedge$
$vm \in vmtf\ \mathcal{A}\ M)$›

definition *find-decomp-wl-imp*
:: ‹$nat\ multiset \Rightarrow (nat, nat)\ ann\text{-}lits \Rightarrow nat \Rightarrow vmtf\text{-}remove\text{-}int \Rightarrow$
$((nat, nat)\ ann\text{-}lits \times vmtf\text{-}remove\text{-}int)\ nres$›
where
‹$find\text{-}decomp\text{-}wl\text{-}imp\ \mathcal{A} = (\lambda M_0\ lev\ vm.\ do\ \{$
$let\ k = count\text{-}decided\ M_0;$
$let\ M_0 = trail\text{-}conv\text{-}to\text{-}no\text{-}CS\ M_0;$

$let\ n = length\ M_0;$
$pos \leftarrow get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\ M_0\ lev;$
$ASSERT((n - pos) \leq uint32\text{-}max);$
$let\ target = n - pos;$
$(\text{-}, M, vm') \leftarrow$
$WHILE_T{}^{\lambda(j, M, vm').\ j \leq target\ \wedge\ \ \ \ M = drop\ j\ M_0\ \wedge\ target \leq length\ M_0\ \wedge\ \ \ \ vm' \in vmtf\ \mathcal{A}\ M\ \wedge\ literals\text{-}ar}$
$(\lambda(j, M, vm).\ j < target)$
$(\lambda(j, M, vm).\ do\ \{$
$ASSERT(M \neq []);$
$ASSERT(Suc\ j \leq uint32\text{-}max);$
$let\ L = atm\text{-}of\ (lit\text{-}of\text{-}hd\text{-}trail\ M);$
$ASSERT(L \in\#\ \mathcal{A});$
$RETURN\ (j + one\text{-}uint32\text{-}nat, tl\ M, vmtf\text{-}unset\ L\ vm)$
$\})$
$(zero\text{-}uint32\text{-}nat, M_0, vm);$
$ASSERT(lev = count\text{-}decided\ M);$
$let\ M = trail\text{-}conv\text{-}back\ lev\ M;$
$RETURN\ (M, vm')$
$\})$›

definition *isa-find-decomp-wl-imp*
:: ‹$trail\text{-}pol \Rightarrow nat \Rightarrow isa\text{-}vmtf\text{-}remove\text{-}int \Rightarrow (trail\text{-}pol \times isa\text{-}vmtf\text{-}remove\text{-}int)\ nres$›
where
‹$isa\text{-}find\text{-}decomp\text{-}wl\text{-}imp = (\lambda M_0\ lev\ vm.\ do\ \{$
$let\ k = count\text{-}decided\text{-}pol\ M_0;$
$let\ M_0 = trail\text{-}pol\text{-}conv\text{-}to\text{-}no\text{-}CS\ M_0;$
$ASSERT(isa\text{-}length\text{-}trail\text{-}pre\ M_0);$
$let\ n = isa\text{-}length\text{-}trail\ M_0;$
$pos \leftarrow get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}imp\ M_0\ lev;$
$ASSERT((n - pos) \leq uint32\text{-}max);$
$let\ target = n - pos;$
$(\text{-}, M, vm') \leftarrow$
$WHILE_T{}^{\lambda(j, M, vm').\ j \leq target}$
$(\lambda(j, M, vm).\ j < target)$
$(\lambda(j, M, vm).\ do\ \{$
$ASSERT(Suc\ j \leq uint32\text{-}max);$
$ASSERT(case\ M\ of\ (M, \text{-}) \Rightarrow M \neq []);$
$ASSERT(tl\text{-}trailt\text{-}tr\text{-}no\text{-}CS\text{-}pre\ M);$
$let\ L = atm\text{-}of\ (lit\text{-}of\text{-}last\text{-}trail\text{-}pol\ M);$
$ASSERT(vmtf\text{-}unset\text{-}pre\ L\ vm);$
$RETURN\ (j + one\text{-}uint32\text{-}nat, tl\text{-}trailt\text{-}tr\text{-}no\text{-}CS\ M, isa\text{-}vmtf\text{-}unset\ L\ vm)$
$\})$
$(zero\text{-}uint32\text{-}nat, M_0, vm);$
$M \leftarrow trail\text{-}conv\text{-}back\text{-}imp\ lev\ M;$
$RETURN\ (M, vm')$
$\})$›

lemma *isa-vmtf-unset-vmtf-unset*:
‹$(uncurry\ (RETURN\ oo\ isa\text{-}vmtf\text{-}unset), uncurry\ (RETURN\ oo\ vmtf\text{-}unset)) \in$
$nat\text{-}rel \times_f (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \rightarrow_f$
$\langle (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A})\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *isa-vmtf-unset-isa-vmtf*:

assumes ‹$vm \in$ *isa-vmtf* $\mathcal{A}$ M› **and** ‹$L \in\#$ $\mathcal{A}$›
shows ‹*isa-vmtf-unset* L $vm \in$ *isa-vmtf* $\mathcal{A}$ M›
⟨*proof*⟩

lemma *isa-vmtf-tl-isa-vmtf*:
assumes ‹$vm \in$ *isa-vmtf* $\mathcal{A}$ M› **and** ‹$M \neq []$› **and** ‹*lit-of* (*hd* M) $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹L = (*atm-of* (*lit-of* (*hd* M)))›
shows ‹*isa-vmtf-unset* L $vm \in$ *isa-vmtf* $\mathcal{A}$ (*tl* M)›
⟨*proof*⟩

lemma *isa-find-decomp-wl-imp-find-decomp-wl-imp*:
‹(*uncurry2 isa-find-decomp-wl-imp*, *uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$)) $\in$
[λ((M, *lev*), *vm*). *lev* < *count-decided* M]$_f$ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)
$\rightarrow$
⟨*trail-pol* $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)⟩*nres-rel*›
⟨*proof*⟩

abbreviation *find-decomp-w-ns-prop* **where**
‹*find-decomp-w-ns-prop* $\mathcal{A}$ $\equiv$
(λ(M::(*nat*, *nat*) *ann-lits*) *highest* -.
(λ(*M1*, *vm*). $\exists$ K *M2*. (*Decided* K # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* M) $\wedge$
get-level M K = *Suc highest* $\wedge$ $vm \in$ *vmtf* $\mathcal{A}$ *M1*))›

definition *find-decomp-w-ns* **where**
‹*find-decomp-w-ns* $\mathcal{A}$ =
(λ(M::(*nat*, *nat*) *ann-lits*) *highest vm*.
SPEC(*find-decomp-w-ns-prop* $\mathcal{A}$ M *highest vm*))›

definition (**in** −) *find-decomp-wl-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *nat twl-st-wl nres*› **where**
‹*find-decomp-wl-st* = (λL (M, N, D, *oth*). *do*{
M' ← *find-decomp-wl'* M (*the* D) L;
RETURN (M', N, D, *oth*)
})›

definition *find-decomp-wl-st-int* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*find-decomp-wl-st-int* = (λ*highest* (M, N, D, Q, W, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*). *do*{
(M', *vm*) ← *isa-find-decomp-wl-imp* M *highest vm*;
RETURN (M', N, D, Q, W, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*)
})›

definition *vmtf-rescore-body*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf-remove-int* ⇒ *phase-saver* ⇒
(*nat* × *vmtf-remove-int* × *phase-saver*) *nres*›
where
‹*vmtf-rescore-body* $\mathcal{A}_{in}$ C - *vm* φ = *do* {
$WHILE_T{}^{\lambda(i,\ vm,\ \varphi).\ i \leq length\ C\ \wedge\ (\forall c \in set\ C.\ atm\text{-}of\ c < length\ \varphi \wedge atm\text{-}of\ c < length\ (fst\ (fst\ vm)))}$
(λ(i, *vm*, φ). i < *length* C)
(λ(i, *vm*, φ). *do* {
ASSERT(i < *length* C);
ASSERT(*atm-of* (C!i) $\in\#$ $\mathcal{A}_{in}$);
let vm' = *vmtf-mark-to-rescore* (*atm-of* (C!i)) *vm*;
RETURN(i+1, vm', φ)
})

$(0, vm, \varphi)$
})

definition *vmtf-rescore*
:: ⟨*nat multiset* ⇒ *nat clause-l* ⇒ *(nat,nat) ann-lits* ⇒ *vmtf-remove-int* ⇒ *phase-saver* ⇒ *(vmtf-remove-int* × *phase-saver) nres*⟩
where
⟨*vmtf-rescore* $\mathcal{A}_{in}$ *C M vm* φ = *do* {
(-, *vm*, φ) ← *vmtf-rescore-body* $\mathcal{A}_{in}$ *C M vm* φ;
RETURN (*vm*, φ)
}⟩

find-theorems *isa-vmtf-mark-to-rescore*

definition *isa-vmtf-rescore-body*
:: ⟨*nat clause-l* ⇒ *trail-pol* ⇒ *isa-vmtf-remove-int* ⇒ *phase-saver* ⇒ (*nat* × *isa-vmtf-remove-int* × *phase-saver*) *nres*⟩
where
⟨*isa-vmtf-rescore-body C - vm* φ = *do* {
$WHILE_T{}^{\lambda(i,\ vm,\ \varphi).\ i \leq length\ C\ \wedge\ (\forall c \in set\ C.\ atm\text{-}of\ c < length\ \varphi \wedge atm\text{-}of\ c < length\ (fst\ (fst\ vm)))}$
(λ(*i*, *vm*, φ). *i* < *length C*)
(λ(*i*, *vm*, φ). *do* {
ASSERT(*i* < *length C*);
ASSERT(*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*C*!*i*)) *vm*);
let vm′ = *isa-vmtf-mark-to-rescore* (*atm-of* (*C*!*i*)) *vm*;
RETURN(*i*+*1*, *vm′*, φ)
})
(0, *vm*, φ)
}⟩

definition *isa-vmtf-rescore*
:: ⟨*nat clause-l* ⇒ *trail-pol* ⇒ *isa-vmtf-remove-int* ⇒ *phase-saver* ⇒ (*isa-vmtf-remove-int* × *phase-saver*) *nres*⟩
where
⟨*isa-vmtf-rescore C M vm* φ = *do* {
(-, *vm*, φ) ← *isa-vmtf-rescore-body C M vm* φ;
RETURN (*vm*, φ)
}⟩

lemma *vmtf-rescore-score-clause*:
⟨(*uncurry3* (*vmtf-rescore* $\mathcal{A}$), *uncurry3* (*rescore-clause* $\mathcal{A}$)) ∈
$[\lambda(((C, M), vm), \varphi).\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C) \wedge vm \in vmtf\ \mathcal{A}\ M \wedge phase\text{-}saving\ \mathcal{A}\ \varphi]_f$
$(\langle Id\rangle list\text{-}rel \times_f Id \times_f Id \times_f Id) \rightarrow \langle Id \times_f Id\rangle\ nres\text{-}rel$⟩
⟨*proof*⟩

lemma *isa-vmtf-rescore-body*:
⟨(*uncurry3* (*isa-vmtf-rescore-body*), *uncurry3* (*vmtf-rescore-body* $\mathcal{A}$)) ∈ $[\lambda\text{-}.\ isasat\text{-}input\text{-}bounded\ \mathcal{A}]_f$
$(Id \times_f trail\text{-}pol\ \mathcal{A} \times_f (Id \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \times_f Id) \rightarrow \langle Id \times_r (Id \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \times_r Id\rangle\ nres\text{-}rel$⟩
⟨*proof*⟩

lemma *isa-vmtf-rescore*:
⟨(*uncurry3* (*isa-vmtf-rescore*), *uncurry3* (*vmtf-rescore* $\mathcal{A}$)) ∈ $[\lambda\text{-}.\ isasat\text{-}input\text{-}bounded\ \mathcal{A}]_f$
$(Id \times_f trail\text{-}pol\ \mathcal{A} \times_f (Id \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \times_f Id) \rightarrow \langle (Id \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \times_f Id\rangle\ nres\text{-}rel$⟩
⟨*proof*⟩

lemma
assumes
vm: ‹$vm \in vmtf\ \mathcal{A}\ M_0$› **and**
lits: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ \mathcal{A}\ M_0$› **and**
target: ‹$highest < count\text{-}decided\ M_0$› **and**
n-d: ‹$no\text{-}dup\ M_0$› **and**
bounded: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}$›
shows
find-decomp-wl-imp-le-find-decomp-wl′:
‹$find\text{-}decomp\text{-}wl\text{-}imp\ \mathcal{A}\ M_0\ highest\ vm \le find\text{-}decomp\text{-}w\text{-}ns\ \mathcal{A}\ M_0\ highest\ vm$›
(**is** *?decomp*)
⟨*proof*⟩

lemma *find-decomp-wl-imp-find-decomp-wl′*:
‹$(uncurry2\ (find\text{-}decomp\text{-}wl\text{-}imp\ \mathcal{A}),\ uncurry2\ (find\text{-}decomp\text{-}w\text{-}ns\ \mathcal{A})) \in [find\text{-}decomp\text{-}w\text{-}ns\text{-}pre\ \mathcal{A}]_f\ Id \times_f Id \times_f Id \to \langle Id \times_f Id\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *find-decomp-wl-imp-code-conbine-cond*:
‹$(\lambda((b, a), c).\ find\text{-}decomp\text{-}w\text{-}ns\text{-}pre\ \mathcal{A}\ ((b, a), c) \land a < count\text{-}decided\ b) = (\lambda((b, a), c).\ find\text{-}decomp\text{-}w\text{-}ns\text{-}pre\ \mathcal{A}\ ((b, a), c))$›
⟨*proof*⟩

definition *vmtf-mark-to-rescore-clause* **where**
‹$vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}clause\ \mathcal{A}_{in}\ arena\ C\ vm = do\ \{$
$ASSERT(arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ arena\ C);$
$nfoldli$
$([C..<C + nat\text{-}of\text{-}uint64\text{-}conv\ (arena\text{-}length\ arena\ C)])$
$(\lambda\text{-}.\ True)$
$(\lambda i\ vm.\ do\ \{$
$ASSERT(i < length\ arena);$
$ASSERT(arena\text{-}lit\text{-}pre\ arena\ i);$
$ASSERT(atm\text{-}of\ (arena\text{-}lit\ arena\ i) \in\#\ \mathcal{A}_{in});$
$RETURN\ (vmtf\text{-}mark\text{-}to\text{-}rescore\ (atm\text{-}of\ (arena\text{-}lit\ arena\ i))\ vm)$
$\})$
vm
$\}$›

definition *isa-vmtf-mark-to-rescore-clause* **where**
‹$isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}clause\ arena\ C\ vm = do\ \{$
$ASSERT(arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ arena\ C);$
$nfoldli$
$([C..<C + nat\text{-}of\text{-}uint64\text{-}conv\ (arena\text{-}length\ arena\ C)])$
$(\lambda\text{-}.\ True)$
$(\lambda i\ vm.\ do\ \{$
$ASSERT(i < length\ arena);$
$ASSERT(arena\text{-}lit\text{-}pre\ arena\ i);$
$ASSERT(isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}pre\ (atm\text{-}of\ (arena\text{-}lit\ arena\ i))\ vm);$
$RETURN\ (isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\ (atm\text{-}of\ (arena\text{-}lit\ arena\ i))\ vm)$

})
vm
}›

lemma *isa-vmtf-mark-to-rescore-clause-vmtf-mark-to-rescore-clause*:
‹(*uncurry2 isa-vmtf-mark-to-rescore-clause*, *uncurry2* (*vmtf-mark-to-rescore-clause* $\mathcal{A}$)) $\in$ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
Id $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\rightarrow$ $\langle$*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

lemma *vmtf-mark-to-rescore-clause-spec*:
‹*vm* $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *valid-arena arena N vdom* $\Longrightarrow$ *C* $\in\#$ *dom-m N* $\Longrightarrow$
($\forall$ *C* $\in$ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$) $\Longrightarrow$
vmtf-mark-to-rescore-clause $\mathcal{A}$ *arena C vm* $\leq$ *RES* (*vmtf* $\mathcal{A}$ *M*)›
$\langle$*proof*$\rangle$

definition *vmtf-mark-to-rescore-also-reasons*
:: ‹*nat multiset* $\Rightarrow$ (*nat*, *nat*) *ann-lits* $\Rightarrow$ *arena* $\Rightarrow$ *nat literal list* $\Rightarrow$ - $\Rightarrow$-› **where**
‹*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$ *M arena outl vm* = *do* {
ASSERT(*length outl* $\leq$ *uint32-max*);
nfoldli
([*0*..<*length outl*])
(λ-. *True*)
(λ*i vm*. *do* {
ASSERT(*i* < *length outl*); *ASSERT*(*length outl* $\leq$ *uint32-max*);
ASSERT($-$*outl* ! *i* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$);
C $\leftarrow$ *get-the-propagation-reason M* ($-$(*outl* ! *i*));
case C of
None $\Rightarrow$ *RETURN* (*vmtf-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*)
| *Some C* $\Rightarrow$ *if C* = *0 then RETURN vm else vmtf-mark-to-rescore-clause* $\mathcal{A}$ *arena C vm*
})
vm
}›

definition *isa-vmtf-mark-to-rescore-also-reasons*
:: ‹*trail-pol* $\Rightarrow$ *arena* $\Rightarrow$ *nat literal list* $\Rightarrow$ - $\Rightarrow$-› **where**
‹*isa-vmtf-mark-to-rescore-also-reasons M arena outl vm* = *do* {
ASSERT(*length outl* $\leq$ *uint32-max*);
nfoldli
([*0*..<*length outl*])
(λ-. *True*)
(λ*i vm*. *do* {
ASSERT(*i* < *length outl*); *ASSERT*(*length outl*$\leq$ *uint32-max*);
C $\leftarrow$ *get-the-propagation-reason-pol M* ($-$(*outl* ! *i*));
case C of
None $\Rightarrow$ *do* {
ASSERT (*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*outl* ! *i*)) *vm*);
RETURN (*isa-vmtf-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*)
}
| *Some C* $\Rightarrow$ *if C* = *0 then RETURN vm else isa-vmtf-mark-to-rescore-clause arena C vm*
})
vm
}›

lemma *isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons*:

‹(*uncurry3 isa-vmtf-mark-to-rescore-also-reasons*, *uncurry3* (*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) → ⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore*′:
‹L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ⟹ *vm* ∈ *vmtf* $\mathcal{A}$ M ⟹ *vmtf-mark-to-rescore* L *vm* ∈ *vmtf* $\mathcal{A}$ M›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore-also-reasons-spec*:
‹*vm* ∈ *vmtf* $\mathcal{A}$ M ⟹ *valid-arena arena* N *vdom* ⟹ *length outl* ≤ *uint32-max* ⟹
(∀ L ∈ *set outl*. L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
(∀ L ∈ *set outl*. ∀ C. (*Propagated* (−L) C ∈ *set* M ⟶ C ≠ *0* ⟶ (C ∈# *dom-m* N ∧
(∀ C ∈ *set* [C..<C + *arena-length arena* C]. *arena-lit arena* C ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)))) ⟹
vmtf-mark-to-rescore-also-reasons $\mathcal{A}$ M *arena outl vm* ≤ *RES* (*vmtf* $\mathcal{A}$ M)›
⟨*proof*⟩

definition *isa-vmtf-find-next-undef* :: ‹*isa-vmtf-remove-int* ⇒ *trail-pol* ⇒ (*nat option*) *nres*› **where**
‹*isa-vmtf-find-next-undef* = (λ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) *M*. *do* {
  WHILE$_T$$^{\lambda next\text{-}search.\ next\text{-}search \neq None \longrightarrow defined\text{-}atm\text{-}pol\text{-}pre\ M\ (the\ next\text{-}search)}$
   (λ*next-search*. *next-search* ≠ *None* ∧ *defined-atm-pol* M (*the next-search*))
   (λ*next-search*. *do* {
     ASSERT(*next-search* ≠ *None*);
     let n = *the next-search*;
     ASSERT (n < *length ns*);
     RETURN (*get-next* (*ns*!n))
    }
)
   next-search
 })›

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
‹(*uncurry isa-vmtf-find-next-undef*, *uncurry* (*vmtf-find-next-undef* $\mathcal{A}$)) ∈
(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\times_r$ *trail-pol* $\mathcal{A}$ $\rightarrow_f$ ⟨⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
⟨*proof*⟩

end
theory *IsaSAT-VMTF-SML*
imports *Watched-Literals.WB-Sort IsaSAT-VMTF IsaSAT-Setup-SML*
begin

lemma *size-conflict-code-refine-raw*:
‹(*return o* (λ(-, *n*, -). *n*), *RETURN o size-conflict-int*) ∈ *conflict-option-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

concrete-definition (**in** −) *size-conflict-code*
uses *size-conflict-code-refine-raw*
is ‹(*?f*,-)∈-›

prepare-code-thms (**in** −) *size-conflict-code-def*

lemmas *size-conflict-code-hnr*[*sepref-fr-rules*] = *size-conflict-code.refine*

lemma *VMTF-Node-ref*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo VMTF-Node*), *uncurry2* (*RETURN ooo VMTF-Node*)) ∈

uint64-nat-assn^k $*_a$ (*option-assn uint32-nat-assn*)^k $*_a$ (*option-assn uint32-nat-assn*)^k $\rightarrow_a$
vmtf-node-assn›
⟨*proof*⟩

lemma *stamp-ref*[*sepref-fr-rules*]:
‹(*return o stamp*, *RETURN o stamp*) ∈ *vmtf-node-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *get-next-ref*[*sepref-fr-rules*]:
‹(*return o get-next*, *RETURN o get-next*) ∈ *vmtf-node-assn*^{k} $\rightarrow_a$
option-assn uint32-nat-assn›
⟨*proof*⟩

lemma *get-prev-ref*[*sepref-fr-rules*]:
‹(*return o get-prev*, *RETURN o get-prev*) ∈ *vmtf-node-assn*^{k} $\rightarrow_a$
option-assn uint32-nat-assn›
⟨*proof*⟩

sepref-definition *atoms-hash-del-code*
is ‹*uncurry* (*RETURN oo atoms-hash-del*)›
:: ‹[*uncurry atoms-hash-del-pre*]$_a$ *uint32-nat-assn*^{k} $*_a$ (*array-assn bool-assn*)^d → *array-assn bool-assn*›
⟨*proof*⟩

declare *atoms-hash-del-code.refine*[*sepref-fr-rules*]
sepref-definition (**in** −) *atoms-hash-insert-code*
is ‹*uncurry* (*RETURN oo atoms-hash-insert*)›
:: ‹[*uncurry atms-hash-insert-pre*]$_a$
uint32-nat-assn^k $*_a$ (*arl32-assn uint32-nat-assn ∗a array-assn bool-assn*)^d →
arl32-assn uint32-nat-assn ∗a array-assn bool-assn›
⟨*proof*⟩

declare *atoms-hash-insert-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *get-pos-of-level-in-trail-imp-fast-code*
is ‹*uncurry get-pos-of-level-in-trail-imp*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

declare *tl-trail-tr-no-CS-code.refine*[*sepref-fr-rules*] *tl-trail-tr-no-CS-fast-code.refine*[*sepref-fr-rules*]

sepref-register *find-decomp-wl-imp*
sepref-register *rescore-clause vmtf-flush*
sepref-register *vmtf-mark-to-rescore*
sepref-register *vmtf-mark-to-rescore-clause*

sepref-register *vmtf-mark-to-rescore-also-reasons get-the-propagation-reason-pol*

sepref-register *find-decomp-w-ns*
sepref-definition (**in** −) *get-pos-of-level-in-trail-imp-code*
is ‹*uncurry get-pos-of-level-in-trail-imp*›
:: ‹*trail-pol-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

declare *get-pos-of-level-in-trail-imp-code.refine*[*sepref-fr-rules*]
get-pos-of-level-in-trail-imp-fast-code.refine[*sepref-fr-rules*]

lemma *update-next-search-ref*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo update-next-search*), *uncurry* (*RETURN oo update-next-search*)) ∈
(*option-assn uint32-nat-assn*)^k $*_a$ *vmtf-remove-conc*^{d} $\rightarrow_a$ *vmtf-remove-conc*›
⟨*proof*⟩

sepref-definition (**in** −)*ns-vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo ns-vmtf-dequeue*)›
:: ‹[*vmtf-dequeue-pre*]$_a$
uint32-nat-assn^k $*_a$ (*array-assn vmtf-node-assn*)^d → *array-assn vmtf-node-assn*›
⟨*proof*⟩

declare *ns-vmtf-dequeue-code.refine*[*sepref-fr-rules*]

abbreviation *vmtf-conc-option-fst-As* **where**
‹*vmtf-conc-option-fst-As* ≡
(*array-assn vmtf-node-assn* ∗*a uint64-nat-assn* ∗*a option-assn uint32-nat-assn*
∗*a option-assn uint32-nat-assn* ∗*a option-assn uint32-nat-assn*)›

sepref-definition *vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo vmtf-dequeue*)›
:: ‹[λ(*L*,(*ns*,*m*,*fst-As*,*next-search*)). *L* < *length ns* ∧ *vmtf-dequeue-pre* (*L*, *ns*)]$_a$
uint32-nat-assn^k $*_a$ *vmtf-conc*^{d} → *vmtf-conc-option-fst-As*›
⟨*proof*⟩

declare *vmtf-dequeue-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-enqueue-code*
is ‹*uncurry2 isa-vmtf-enqueue*›
:: ‹[*vmtf-enqueue-pre*]$_a$
trail-pol-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-conc-option-fst-As*^{d} → *vmtf-conc*›
⟨*proof*⟩

declare *vmtf-enqueue-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-enqueue-fast-code*
is ‹*uncurry2 isa-vmtf-enqueue*›
:: ‹[*vmtf-enqueue-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-conc-option-fst-As*^{d} → *vmtf-conc*›
⟨*proof*⟩

declare *vmtf-enqueue-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *partition-vmtf-nth-code*
is ‹*uncurry3 partition-vmtf-nth*›
:: ‹[λ(((*ns*, -), *hi*), *xs*). (∀ *x*∈*set xs*. *x* < *length ns*) ∧ *length xs* ≤ *uint32-max*]$_a$
(*array-assn vmtf-node-assn*)^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ (*arl32-assn uint32-nat-assn*)^d
→
arl32-assn uint32-nat-assn ∗*a uint32-nat-assn*›
⟨*proof*⟩

declare *partition-vmtf-nth-code.refine*[*sepref-fr-rules*]

sepref-register *partition-between-ref*

lemma *uint32-nat-assn-minus-fast*:
‹$(uncurry\ (return\ oo\ (-)),\ uncurry\ (RETURN\ oo\ (-))) \in$
$[\lambda(a,\ b).\ a \geq b]_a\ \ uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition (**in** −) *partition-between-ref-vmtf-code*
is ‹*uncurry3 partition-between-ref-vmtf*›
:: ‹$[\lambda(((vm),\ \text{-}),\ remove).\ (\forall x{\in}\#mset\ remove.\ x < length\ (fst\ vm)) \wedge length\ remove \leq uint32\text{-}max]_a$
$(array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a (arl32\text{-}assn\ uint32\text{-}nat\text{-}assn)^d$
$\rightarrow$
$arl32\text{-}assn\ uint32\text{-}nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-register *partition-between-ref-vmtf quicksort-vmtf-nth-ref*
declare *partition-between-ref-vmtf-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *quicksort-vmtf-nth-ref-code*
is ‹*uncurry3 quicksort-vmtf-nth-ref*›
:: ‹$[\lambda((vm,\ \text{-}),\ remove).\ (\forall x{\in}\#mset\ remove.\ x < length\ (fst\ vm)) \wedge length\ remove \leq uint32\text{-}max]_a$
$(array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a (arl32\text{-}assn\ uint32\text{-}nat\text{-}assn)^d$
$\rightarrow$
$arl32\text{-}assn\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *quicksort-vmtf-nth-ref-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *quicksort-vmtf-nth-code*
is ‹*uncurry quicksort-vmtf-nth*›
:: ‹$[\lambda(vm,\ remove).\ (\forall x{\in}\#mset\ remove.\ x < length\ (fst\ vm)) \wedge length\ remove \leq uint32\text{-}max]_a$
$vmtf\text{-}conc^k *_a (arl32\text{-}assn\ uint32\text{-}nat\text{-}assn)^d \rightarrow$
$arl32\text{-}assn\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *quicksort-vmtf-nth-code.refine*[*sepref-fr-rules*]

lemma *quicksort-vmtf-nth-code-reorder-list*[*sepref-fr-rules*]:
‹$(uncurry\ quicksort\text{-}vmtf\text{-}nth\text{-}code,\ uncurry\ reorder\text{-}list) \in$
$[\lambda((a,\ \text{-}),\ b).\ (\forall x{\in}set\ b.\ x < length\ a) \wedge length\ b \leq uint32\text{-}max]_a$
$vmtf\text{-}conc^k *_a (arl32\text{-}assn\ uint32\text{-}nat\text{-}assn)^d \rightarrow arl32\text{-}assn\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩
sepref-register *isa-vmtf-enqueue*

lemma *current-stamp-hnr*[*sepref-fr-rules*]:
‹$(return\ o\ current\text{-}stamp,\ RETURN\ o\ current\text{-}stamp) \in vmtf\text{-}conc^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *vmtf-en-dequeue-code*
is ‹*uncurry2 isa-vmtf-en-dequeue*›
:: ‹$[isa\text{-}vmtf\text{-}en\text{-}dequeue\text{-}pre]_a$
$trail\text{-}pol\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}conc^d \rightarrow vmtf\text{-}conc$›
⟨*proof*⟩

declare *vmtf-en-dequeue-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-en-dequeue-fast-code*
  is ‹*uncurry2 isa-vmtf-en-dequeue*›
  :: ‹[*isa-vmtf-en-dequeue-pre*]$_a$
    $\textit{trail-pol-fast-assn}^k *_a \textit{uint32-nat-assn}^k *_a \textit{vmtf-conc}^d \rightarrow \textit{vmtf-conc}$›
  ⟨*proof*⟩

declare *vmtf-en-dequeue-fast-code.refine*[*sepref-fr-rules*]

sepref-register *vmtf-rescale*
sepref-definition *vmtf-rescale-code*
  is ‹*vmtf-rescale*›
  :: ‹$\textit{vmtf-conc}^d \rightarrow_a \textit{vmtf-conc}$›
  ⟨*proof*⟩

declare *vmtf-rescale-code.refine*[*sepref-fr-rules*]
lemma *uint64-nal-rel-le-uint64-max*: ‹$(a, b) \in \textit{uint64-nat-rel} \Longrightarrow b \leq \textit{uint64-max}$›
  ⟨*proof*⟩

This functions deletes all elements of a resizable array, without resizing it.

definition *emptied-arl* :: ‹$'a\ \textit{array-list32} \Rightarrow 'a\ \textit{array-list32}$› **where**
‹$\textit{emptied-arl} = (\lambda(a, n).\ (a, 0))$›

lemma *emptied-arl-refine*[*sepref-fr-rules*]:
  ‹$(\textit{return o emptied-arl}, \textit{RETURN o emptied-list}) \in (\textit{arl32-assn}\ R)^d \rightarrow_a \textit{arl32-assn}\ R$›
  ⟨*proof*⟩

sepref-register *isa-vmtf-en-dequeue*
sepref-definition *isa-vmtf-flush-code*
  is ‹*uncurry isa-vmtf-flush-int*›
  :: ‹$\textit{trail-pol-assn}^k *_a (\textit{vmtf-conc} *a (\textit{arl32-assn uint32-nat-assn} *a \textit{atoms-hash-assn}))^d \rightarrow_a$
    $(\textit{vmtf-conc} *a (\textit{arl32-assn uint32-nat-assn} *a \textit{atoms-hash-assn}))$›
  ⟨*proof*⟩

declare *isa-vmtf-flush-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-vmtf-flush-fast-code*
  is ‹*uncurry isa-vmtf-flush-int*›
  :: ‹$\textit{trail-pol-fast-assn}^k *_a (\textit{vmtf-conc} *a (\textit{arl32-assn uint32-nat-assn} *a \textit{atoms-hash-assn}))^d \rightarrow_a$
    $(\textit{vmtf-conc} *a (\textit{arl32-assn uint32-nat-assn} *a \textit{atoms-hash-assn}))$›
  ⟨*proof*⟩

declare *isa-vmtf-flush-code.refine*[*sepref-fr-rules*]
  isa-vmtf-flush-fast-code.refine[*sepref-fr-rules*]

sepref-register *isa-vmtf-mark-to-rescore*
sepref-definition *isa-vmtf-mark-to-rescore-code*
  is ‹*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore*)›
  :: ‹[*uncurry isa-vmtf-mark-to-rescore-pre*]$_a$
    $\textit{uint32-nat-assn}^k *_a \textit{vmtf-remove-conc}^d \rightarrow \textit{vmtf-remove-conc}$›
  ⟨*proof*⟩

declare *isa-vmtf-mark-to-rescore-code.refine*[*sepref-fr-rules*]

sepref-register *isa-vmtf-unset*

sepref-definition *isa-vmtf-unset-code*
is ‹*uncurry* (*RETURN oo isa-vmtf-unset*)›
:: ‹$[uncurry\ vmtf\text{-}unset\text{-}pre]_a$
$uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d \to vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *isa-vmtf-unset-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-mark-to-rescore-and-unset-code*
is ‹*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore-and-unset*)›
:: ‹$[isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}and\text{-}unset\text{-}pre]_a$
$uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d \to vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *vmtf-mark-to-rescore-and-unset-code.refine*[*sepref-fr-rules*]
sepref-definition *find-decomp-wl-imp-code*
is ‹*uncurry2* (*isa-find-decomp-wl-imp*)›
:: ‹$[\lambda((M,\ lev),\ vm).\ True]_a\ trail\text{-}pol\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d$
$\to trail\text{-}pol\text{-}assn *a\ vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *find-decomp-wl-imp-code.refine*[*sepref-fr-rules*]

sepref-definition *find-decomp-wl-imp-fast-code*
is ‹*uncurry2* (*isa-find-decomp-wl-imp*)›
:: ‹$[\lambda((M,\ lev),\ vm).\ True]_a\ trail\text{-}pol\text{-}fast\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d$
$\to trail\text{-}pol\text{-}fast\text{-}assn *a\ vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *find-decomp-wl-imp-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-rescore-code*
is ‹*uncurry3 isa-vmtf-rescore*›
:: ‹$(array\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a trail\text{-}pol\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d *_a phase\text{-}saver\text{-}conc^d \to_a$
$vmtf\text{-}remove\text{-}conc *a\ phase\text{-}saver\text{-}conc$›
⟨*proof*⟩

sepref-definition *vmtf-rescore-fast-code*
is ‹*uncurry3 isa-vmtf-rescore*›
:: ‹$(array\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a trail\text{-}pol\text{-}fast\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d *_a phase\text{-}saver\text{-}conc^d \to_a$
$vmtf\text{-}remove\text{-}conc *a\ phase\text{-}saver\text{-}conc$›
⟨*proof*⟩

declare
vmtf-rescore-code.refine[*sepref-fr-rules*]
vmtf-rescore-fast-code.refine[*sepref-fr-rules*]

sepref-definition *find-decomp-wl-imp′-code*
is ‹*uncurry find-decomp-wl-st-int*›
:: ‹$uint32\text{-}nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \to_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *find-decomp-wl-imp′-code.refine*[*sepref-fr-rules*]

sepref-definition *find-decomp-wl-imp′-fast-code*
is ‹*uncurry find-decomp-wl-st-int*›

:: ‹$uint32\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a$
$isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *find-decomp-wl-imp′-fast-code.refine*[*sepref-fr-rules*]
sepref-definition *vmtf-mark-to-rescore-clause-code*
is ‹*uncurry2* (*isa-vmtf-mark-to-rescore-clause*)›
:: ‹$arena\text{-}assn^k *_a nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d \rightarrow_a vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *vmtf-mark-to-rescore-clause-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-mark-to-rescore-also-reasons-code*
is ‹*uncurry3* (*isa-vmtf-mark-to-rescore-also-reasons*)›
:: ‹$trail\text{-}pol\text{-}assn^k *_a arena\text{-}assn^k *_a (arl32\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a vmtf\text{-}remove\text{-}conc^d \rightarrow_a vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *vmtf-mark-to-rescore-also-reasons-code.refine*[*sepref-fr-rules*]

sepref-definition (**in**−) *isa-arena-lit-fast-code2*
is ‹*uncurry isa-arena-lit*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

declare *isa-arena-lit-fast-code.refine*

lemma *isa-arena-lit-fast-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-lit-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-lit*))
∈ [*uncurry arena-lit-pre*]$_a$
$arena\text{-}fast\text{-}assn^k *_a nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-definition *vmtf-mark-to-rescore-clause-fast-code*
is ‹*uncurry2* (*isa-vmtf-mark-to-rescore-clause*)›
:: ‹$[\lambda((N, \text{-}), \text{-}).\ length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d \rightarrow vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *vmtf-mark-to-rescore-clause-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-mark-to-rescore-also-reasons-fast-code*
is ‹*uncurry3* (*isa-vmtf-mark-to-rescore-also-reasons*)›
:: ‹$[\lambda(((\text{-}, N), \text{-}), \text{-}).\ length\ N \leq uint64\text{-}max]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a (arl32\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a vmtf\text{-}remove\text{-}conc^d \rightarrow$
$vmtf\text{-}remove\text{-}conc$›
⟨*proof*⟩

declare *vmtf-mark-to-rescore-also-reasons-fast-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT-Backtrack*
imports *IsaSAT-Setup IsaSAT-VMTF*
begin

0.1.18 Backtrack

Backtrack with direct extraction of literal if highest level

Empty conflict **definition** (**in** −) *empty-conflict-and-extract-clause*
:: ‹(*nat*,*nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat clause-l* ⇒
(*nat clause option* × *nat clause-l* × *nat*) *nres*›
where
‹*empty-conflict-and-extract-clause M D outl* =
$SPEC(\lambda(D, C, n).\ D = None \land mset\ C = mset\ outl \land C!0 = outl!0 \land$
$(length\ C > 1 \longrightarrow highest\text{-}lit\ M\ (mset\ (tl\ C))\ (Some\ (C!1,\ get\text{-}level\ M\ (C!1)))) \land$
$(length\ C > 1 \longrightarrow n = get\text{-}level\ M\ (C!1)) \land$
$(length\ C = 1 \longrightarrow n = 0)$
)›

definition *empty-conflict-and-extract-clause-heur-inv* **where**
‹*empty-conflict-and-extract-clause-heur-inv M outl* =
$(\lambda(E, C, i).\ mset\ (take\ i\ C) = mset\ (take\ i\ outl) \land$
$length\ C = length\ outl \land C\ !\ 0 = outl\ !\ 0 \land i \geq 1 \land i \leq length\ outl \land$
$(1 < length\ (take\ i\ C) \longrightarrow$
$highest\text{-}lit\ M\ (mset\ (tl\ (take\ i\ C)))$
$(Some\ (C\ !\ 1,\ get\text{-}level\ M\ (C\ !\ 1)))))$›

definition *empty-conflict-and-extract-clause-heur* ::
nat multiset ⇒ (*nat*, *nat*) *ann-lits*
⇒ *lookup-clause-rel*
⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*
where
‹*empty-conflict-and-extract-clause-heur* $\mathcal{A}$ *M D outl* = *do* {
let $C = replicate\ (length\ outl)\ (outl!0)$;
$(D, C, \text{-}) \leftarrow WHILE_T{}^{empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\text{-}heur\text{-}inv\ M\ outl}$
$(\lambda(D, C, i).\ i < length\text{-}uint32\text{-}nat\ outl)$
$(\lambda(D, C, i).$ *do* {
$ASSERT(i < length\ outl)$;
$ASSERT(i < length\ C)$;
$ASSERT(lookup\text{-}conflict\text{-}remove1\text{-}pre\ (outl\ !\ i,\ D))$;
let $D = lookup\text{-}conflict\text{-}remove1\ (outl\ !\ i)\ D$;
let $C = C[i := outl\ !\ i]$;
$ASSERT(C!i \in\# \mathcal{L}_{all}\ \mathcal{A} \land C!1 \in\# \mathcal{L}_{all}\ \mathcal{A} \land 1 < length\ C)$;
let $C = (if\ get\text{-}level\ M\ (C!i) > get\text{-}level\ M\ (C!one\text{-}uint32\text{-}nat)\ then\ swap\ C\ one\text{-}uint32\text{-}nat\ i$
$else\ C)$;
$ASSERT(i+1 \leq uint\text{-}max)$;
$RETURN\ (D, C, i+one\text{-}uint32\text{-}nat)$
})
$(D, C, one\text{-}uint32\text{-}nat)$;
$ASSERT(length\ outl \neq 1 \longrightarrow length\ C > 1)$;
$ASSERT(length\ outl \neq 1 \longrightarrow C!1 \in\# \mathcal{L}_{all}\ \mathcal{A})$;
$RETURN\ ((True, D), C, if\ length\ outl = 1\ then\ zero\text{-}uint32\text{-}nat\ else\ get\text{-}level\ M\ (C!1))$
}›

lemma *empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause*:
assumes
D: ‹$D = mset\ (tl\ outl)$› **and**
outl: ‹$outl \neq []$› **and**
dist: ‹*distinct outl*› **and**
lits: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ outl)$› **and**

DD′: ‹$(D', D) \in$ *lookup-clause-rel* $\mathcal{A}$› **and**
consistent: ‹$\neg$ *tautology* (*mset outl*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*empty-conflict-and-extract-clause-heur* $\mathcal{A}$ M D' *outl* $\leq \Downarrow$ (*option-lookup-clause-rel* $\mathcal{A} \times_r Id \times_r Id$)
(*empty-conflict-and-extract-clause* M D *outl*)›
⟨*proof*⟩

definition *isa-empty-conflict-and-extract-clause-heur* ::
trail-pol ⇒ *lookup-clause-rel* ⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*
where
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(D, C, -) ← $WHILE_T$
($\lambda(D, C, i)$. i < *length-uint32-nat outl*)
($\lambda(D, C, i)$. *do* {
ASSERT(i < *length outl*);
ASSERT(i < *length* C);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! i, D));
let D = *lookup-conflict-remove1* (*outl* ! i) D;
let C = C[i := *outl* ! i];
ASSERT(*get-level-pol-pre* (M, C!i));
ASSERT(*get-level-pol-pre* (M, C!*one-uint32-nat*));
ASSERT(*one-uint32-nat* < *length* C);
let C = (*if get-level-pol* M (C!i) > *get-level-pol* M (C!*one-uint32-nat*) *then swap* C *one-uint32-nat*
i *else* C);
ASSERT(i+*1* ≤ *uint-max*);
RETURN (D, C, i+*one-uint32-nat*)
})
(D, C, *one-uint32-nat*);
ASSERT(*length outl* ≠ *1* ⟶ *length* C > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (M, C!*1*));
RETURN ((*True*, D), C, *if length outl* = *1 then zero-uint32-nat else get-level-pol* M (C!*1*))
}›

lemma *isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur*:
‹(*uncurry2 isa-empty-conflict-and-extract-clause-heur*, *uncurry2* (*empty-conflict-and-extract-clause-heur*
$\mathcal{A}$)) ∈
trail-pol $\mathcal{A} \times_f Id \times_f Id \rightarrow_f \langle Id\rangle$*nres-rel* ›
⟨*proof*⟩

definition *extract-shorter-conflict-wl-nlit* **where**
‹*extract-shorter-conflict-wl-nlit K M NU D NE UE* =
SPEC($\lambda D'$. $D' \neq$ *None* ∧ *the* D' ⊆# *the* D ∧ K ∈# *the* D' ∧
mset ‘# *ran-mf NU* + *NE* + *UE* ⊨*pm the* D')›

definition *extract-shorter-conflict-wl-nlit-st*
:: ‹′*v twl-st-wl* ⇒ ′*v twl-st-wl nres*›
where
‹*extract-shorter-conflict-wl-nlit-st* =
($\lambda(M, N, D, NE, UE, WS, Q)$. *do* {
let K = −*lit-of* (*hd* M);
D ← *extract-shorter-conflict-wl-nlit K M N D NE UE*;
RETURN (M, N, D, NE, UE, WS, Q)})›

definition *empty-lookup-conflict-and-highest*
:: ‹$'v$ *twl-st-wl* $\Rightarrow$ ($'v$ *twl-st-wl* $\times$ *nat*) *nres*›
where
‹*empty-lookup-conflict-and-highest* =
(λ(*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*). *do* {
let K = $-$*lit-of* (*hd M*);
let n = *get-maximum-level M* (*remove1-mset K* (*the D*));
RETURN ((*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*), *n*)})›

definition *backtrack-wl-D-heur-inv* **where**
‹*backtrack-wl-D-heur-inv S* $\longleftrightarrow$ ($\exists$ S'. (S, S') $\in$ *twl-st-heur-conflict-ana* $\wedge$ *backtrack-wl-D-inv* S')›

definition *extract-shorter-conflict-heur* **where**
‹*extract-shorter-conflict-heur* = (λ*M NU NUE C outl*. *do* {
let K = *lit-of* (*hd M*);
let C = *Some* (*remove1-mset* ($-K$) (*the C*));
C $\leftarrow$ *iterate-over-conflict* ($-K$) *M NU NUE* (*the C*);
RETURN (*Some* (*add-mset* ($-K$) *C*))
})›

definition (**in** $-$) *empty-cach* **where**
‹*empty-cach cach* = (λ-. *SEEN-UNKNOWN*)›

definition *empty-conflict-and-extract-clause-pre*
:: ‹(((*nat*,*nat*) *ann-lits* $\times$ *nat clause*) $\times$ *nat clause-l*) $\Rightarrow$ *bool*› **where**
‹*empty-conflict-and-extract-clause-pre* =
(λ((*M*, *D*), *outl*). *D* = *mset* (*tl outl*) $\wedge$ *outl* $\neq$ [] $\wedge$ *distinct outl* $\wedge$
$\neg$*tautology* (*mset outl*) $\wedge$ *length outl* $\leq$ *uint-max*)›

definition (**in** $-$) *empty-cach-ref* **where**
‹*empty-cach-ref* = (λ(*cach*, *support*). (*replicate* (*length cach*) *SEEN-UNKNOWN*, []))›

definition *empty-cach-ref-set-inv* **where**
‹*empty-cach-ref-set-inv cach0 support* =
(λ(*i*, *cach*). *length cach* = *length cach0* $\wedge$
($\forall$ *L* $\in$ *set* (*drop i support*). *L* < *length cach*) $\wedge$
($\forall$ *L* $\in$ *set* (*take i support*). *cach* ! *L* = *SEEN-UNKNOWN*) $\wedge$
($\forall$ *L* < *length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longrightarrow$ *L* $\in$ *set* (*drop i support*)))›

definition *empty-cach-ref-set* **where**
‹*empty-cach-ref-set* = (λ(*cach0*, *support*). *do* {
let n = *length support*;
ASSERT(*n* $\leq$ *Suc* (*uint32-max div 2*));
(-, *cach*) $\leftarrow$ $WHILE_T{}^{empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support}$
(λ(*i*, *cach*). *i* < *length support*)
(λ(*i*, *cach*). *do* {
ASSERT(*i* < *length support*);
ASSERT(*support* ! *i* < *length cach*);
RETURN(*i*+*1*, *cach*[*support* ! *i* := *SEEN-UNKNOWN*])
})
(*0*, *cach0*);
RETURN (*cach*, *emptied-list support*)
})›

lemma *empty-cach-ref-set-empty-cach-ref*:

‹(*empty-cach-ref-set*, *RETURN o empty-cach-ref*) ∈
[λ(*cach*, *supp*). (∀ *L* ∈ *set supp*. *L* < *length cach*) ∧ *length supp* ≤ *Suc* (*uint32-max div 2*) ∧
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set supp*)]$_f$
Id → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *empty-cach-ref-empty-cach*:
‹*isasat-input-bounded* 𝒜 ⟹ (*RETURN o empty-cach-ref*, *RETURN o empty-cach*) ∈ *cach-refinement* 𝒜 →$_f$ ⟨*cach-refinement* 𝒜⟩ *nres-rel*›
⟨*proof*⟩

definition *empty-cach-ref-pre* **where**
‹*empty-cach-ref-pre* = (λ(*cach* :: *minimize-status list*, *supp* :: *nat list*).
(∀ *L*∈*set supp*. *L* < *length cach*) ∧
length supp ≤ *Suc* (*uint-max div 2*) ∧
(∀ *L*<*length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set supp*))›

Minimisation of the conflict

definition *extract-shorter-conflict-list-heur-st*
:: ‹*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × - × -) *nres*›
where
‹*extract-shorter-conflict-list-heur-st* = (λ(*M*, *N*, (-, *D*), *Q′*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *ccont*, *vdom*). *do* {
ASSERT(*fst M* ≠ []);
let K = *lit-of-last-trail-pol M*;
ASSERT(*0* < *length outl*);
ASSERT(*lookup-conflict-remove1-pre* (−*K*, *D*));
let D = *lookup-conflict-remove1* (−*K*) *D*;
let outl = *outl*[*0* := −*K*];
vm ← *isa-vmtf-mark-to-rescore-also-reasons M N outl vm*;
(*D*, *cach*, *outl*) ← *isa-minimize-and-extract-highest-lookup-conflict M N D cach lbd outl*;
ASSERT(*empty-cach-ref-pre cach*);
let cach = *empty-cach-ref cach*;
ASSERT(*outl* ≠ [] ∧ *length outl* ≤ *uint-max*);
(*D*, *C*, *n*) ← *isa-empty-conflict-and-extract-clause-heur M D outl*;
RETURN ((*M*, *N*, *D*, *Q′*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *take 1 outl*, *stats*, *ccont*, *vdom*), *n*, *C*)
})›

lemma *the-option-lookup-clause-assn*:
‹(*RETURN o snd*, *RETURN o the*) ∈ [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* 𝒜 → ⟨*lookup-clause-rel* 𝒜⟩*nres-rel*›
⟨*proof*⟩

definition *propagate-bt-wl-D-heur*
:: ‹*nat literal* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*length vdom* ≤ *length N0*);
ASSERT(*length avdom* ≤ *length N0*);
ASSERT(*nat-of-lit* (*C*!*1*) < *length W0* ∧ *nat-of-lit* (−*L*) < *length W0*);
ASSERT(*length C* > *1*);
let L′ = *C*!*1*;
ASSERT(*length C* ≤ *uint32-max div 2* + *1*);
(*vm*, φ) ← *isa-vmtf-rescore C M vm0* φ*0*;
glue ← *get-LBD lbd*;

let b = False;
let b′ = (length C = 2);
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, φ0, y, cach, lbd, outl, stats, fema, sema,
res-info, vdom, avdom, lcount, opts) ⟶ append-and-length-fast-code-pre ((b, C), N0));
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, φ0, y, cach, lbd, outl, stats, fema, sema,
res-info, vdom, avdom, lcount, opts) ⟶ lcount < uint64-max);
(N, i) ← fm-add-new b C N0;
ASSERT(update-lbd-pre ((i, glue), N));
let N = update-lbd i glue N;
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, φ0, y, cach, lbd, outl, stats, fema, sema,
res-info, vdom, avdom, lcount, opts) ⟶ length-ll W0 (nat-of-lit (−L)) < uint64-max);
let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [to-watcher i L′ b′]];
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, φ0, y, cach, lbd, outl, stats, fema, sema,
res-info, vdom, avdom, lcount, opts) ⟶ length-ll W (nat-of-lit L′) < uint64-max);
let W = W[nat-of-lit L′ := W!nat-of-lit L′ @ [to-watcher i (−L) b′]];
lbd ← lbd-empty lbd;
ASSERT(isa-length-trail-pre M);
let j = isa-length-trail M;
ASSERT(i ≠ DECISION-REASON);
ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
let M = cons-trail-Propagated-tr (− L) i M;
vm ← isa-vmtf-flush-int M vm;
ASSERT(atm-of L < length φ);
RETURN (M, N, D, j, W, vm, save-phase (−L) φ, zero-uint32-nat,
cach, lbd, outl, add-lbd (uint64-of-nat glue) stats, ema-update glue fema, ema-update glue sema,
incr-conflict-count-since-last-restart res-info, vdom @ [nat-of-uint32-conv i],
avdom @ [nat-of-uint32-conv i],
lcount + 1, opts)
})›

definition (**in** −) *lit-of-hd-trail-st-heur* :: ‹*twl-st-wl-heur ⇒ nat literal*› **where**
‹*lit-of-hd-trail-st-heur S = lit-of-last-trail-pol (get-trail-wl-heur S)*›

definition *remove-last*
:: ‹*nat literal ⇒ nat clause option ⇒ nat clause option nres*›
where
‹*remove-last - - = SPEC((=) None)*›

definition *propagate-unit-bt-wl-D-int*
:: ‹*nat literal ⇒ twl-st-wl-heur ⇒ twl-st-wl-heur nres*›
where
‹*propagate-unit-bt-wl-D-int = (λL (M, N, D, Q, W, vm, φ, clvls, cach, lbd, outl, stats,*
fema, sema, res-info, vdom). do {
vm ← isa-vmtf-flush-int M vm;
glue ← get-LBD lbd;
lbd ← lbd-empty lbd;
ASSERT(isa-length-trail-pre M);
let j = isa-length-trail M;
ASSERT(0 ≠ DECISION-REASON);
ASSERT(cons-trail-Propagated-tr-pre ((− L, 0::nat), M));
let M = cons-trail-Propagated-tr (− L) 0 M;
let stats = incr-uset stats;
RETURN (M, N, D, j, W, vm, φ, clvls, cach, lbd, outl, stats,
ema-update glue fema, ema-update glue sema,
incr-conflict-count-since-last-restart res-info, vdom)})›

Full function **definition** *backtrack-wl-D-nlit-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*backtrack-wl-D-nlit-heur* S_0 =
do {
ASSERT(*backtrack-wl-D-heur-inv* S_0);
ASSERT(*fst* (*get-trail-wl-heur* S_0) ≠ []);
let L = *lit-of-hd-trail-st-heur* S_0;
(*S*, *n*, *C*) ← *extract-shorter-conflict-list-heur-st* S_0;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);
S ← *find-decomp-wl-st-int n S*;

ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);
if size C > *1*
then do {
propagate-bt-wl-D-heur L C S
}
else do {
propagate-unit-bt-wl-D-int L S
}
}›

lemma *get-all-ann-decomposition-get-level*:
assumes
L′: ‹*L′* = *lit-of* (*hd M′*)› **and**
nd: ‹*no-dup M′*› **and**
decomp: ‹(*Decided K* # *a*, *M2*) ∈ *set* (*get-all-ann-decomposition M′*)› **and**
lev-K: ‹*get-level M′ K* = *Suc* (*get-maximum-level M′* (*remove1-mset* (− *L′*) *y*))› **and**
L: ‹*L* ∈# *remove1-mset* (− *lit-of* (*hd M′*)) *y*›
shows ‹*get-level a L* = *get-level M′ L*›
⟨*proof*⟩

definition *del-conflict-wl* :: ‹*′v twl-st-wl* ⇒ *′v twl-st-wl*› **where**
‹*del-conflict-wl* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). (*M*, *N*, *None*, *NE*, *UE*, *Q*, *W*))›

lemma [*simp*]:
‹*get-clauses-wl* (*del-conflict-wl S*) = *get-clauses-wl S*›
⟨*proof*⟩

lemma *lcount-add-clause*[*simp*]: ‹*i* ∉# *dom-m N* ⟹
size (*learned-clss-l* (*fmupd i* (*C*, *False*) *N*)) = *Suc* (*size* (*learned-clss-l N*))›
⟨*proof*⟩

lemma *length-watched-le*:
assumes
prop-inv: ‹*correct-watching x1*› **and**
xb-x′a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-conflict-ana*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
shows ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*) − *2*›
⟨*proof*⟩

lemma *backtrack-wl-D-nlit-backtrack-wl-D*:
‹(*backtrack-wl-D-nlit-heur*, *backtrack-wl-D*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧ *length* (*get-clauses-wl-heur S*) = *r*} $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *6* + *r* + *uint32-max div 2*}⟩*nres-rel*›
(**is** ‹- ∈ *?R* $\rightarrow_f$ ⟨*?S*⟩*nres-rel*›)

⟨proof⟩

Backtrack with direct extraction of literal if highest level

lemma *le-uint32-max-div-2-le-uint32-max*: ‹$a \leq$ *uint-max div 2* $+ 1 \Longrightarrow a \leq$ *uint32-max*›
⟨proof⟩

lemma *propagate-bt-wl-D-heur-alt-def*:
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*length vdom* ≤ *length N0*);
ASSERT(*length avdom* ≤ *length N0*);
ASSERT(*nat-of-lit* (*C*!*1*) < *length W0* ∧ *nat-of-lit* (−*L*) < *length W0*);
ASSERT(*length C* > *1*);
let L′ = *C*!*1*;
ASSERT(*length C* ≤ *uint32-max div 2* + *1*);
(*vm*, φ) ← *isa-vmtf-rescore C M vm0* φ*0*;
glue ← *get-LBD lbd*;
let b = *False*;
let b′ = (*length C* = *2*);
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*) ⟶ *append-and-length-fast-code-pre* ((*b*, *C*), *N0*));
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*) ⟶ *lcount* < *uint64-max*);
(*N*, *i*) ← *fm-add-new-fast b C N0*;
ASSERT(*update-lbd-pre* ((*i*, *glue*), *N*));
let N = *update-lbd i glue N*;
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*) ⟶ *length-ll W0* (*nat-of-lit* (−*L*)) < *uint64-max*);
let W = *W0*[*nat-of-lit* (− *L*) := *W0* ! *nat-of-lit* (− *L*) @ [*to-watcher-fast* (*i*) *L′ b′*]];
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, φ*0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
res-info, *vdom*, *avdom*, *lcount*, *opts*) ⟶ *length-ll W* (*nat-of-lit L′*) < *uint64-max*);
let W = *W*[*nat-of-lit L′* := *W*!*nat-of-lit L′* @ [*to-watcher-fast* (*i*) (−*L*) *b′*]];
lbd ← *lbd-empty lbd*;
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
ASSERT(*i* ≠ *DECISION-REASON*);
ASSERT(*cons-trail-Propagated-tr-pre* ((−*L*, *i*), *M*));
let M = *cons-trail-Propagated-tr* (− *L*) *i M*;
vm ← *isa-vmtf-flush-int M vm*;
ASSERT(*atm-of L* < *length* φ);
RETURN (*M*, *N*, *D*, *j*, *W*, *vm*, *save-phase* (−*L*) φ, *zero-uint32-nat*,
cach, *lbd*, *outl*, *add-lbd* (*uint64-of-nat glue*) *stats*, *ema-update glue fema*, *ema-update glue sema*,
incr-conflict-count-since-last-restart res-info, *vdom* @ [*nat-of-uint64-conv i*],
avdom @ [*nat-of-uint64-conv i*],
lcount + *1*, *opts*)
})›
⟨proof⟩

lemma *propagate-bt-wl-D-fast-code-isasat-fastI2*: ‹*isasat-fast b* ⟹
b = (*a1′*, *a2′*) ⟹
a2′ = (*a1′a*, *a2′a*) ⟹
a < *length a1′a* ⟹ *a* ≤ *uint64-max*›
⟨proof⟩

lemma *propagate-bt-wl-D-fast-code-isasat-fastI3*: ‹*isasat-fast b* $\Longrightarrow$
$b = (a1', a2') \Longrightarrow$
$a2' = (a1'a, a2'a) \Longrightarrow$
$a \leq$ *length* $a1'a \Longrightarrow a <$ *uint64-max*›
⟨*proof*⟩

lemma *lit-of-hd-trail-st-heur-alt-def*:
‹*lit-of-hd-trail-st-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ). *lit-of-last-trail-pol M*)›
⟨*proof*⟩

end
theory *IsaSAT-Backtrack-SML*
imports *IsaSAT-Backtrack IsaSAT-VMTF-SML IsaSAT-Setup-SML*
begin

lemma *isa-empty-conflict-and-extract-clause-heur-alt-def*:
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*nat-of-uint32-conv* (*length outl*)) (*outl*!*0*);
(*D*, *C*, -) ← $WHILE_T$
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*get-level-pol-pre* (*M*, *C*!*i*));
ASSERT(*get-level-pol-pre* (*M*, *C*!*one-uint32-nat*));
ASSERT(*one-uint32-nat* < *length C*);
let L1 = *C*!*i*;
let L2 = *C*!*one-uint32-nat*;
let C = (*if get-level-pol M L1* > *get-level-pol M L2 then swap C one-uint32-nat i else C*);
ASSERT(*i*+*1* ≤ *uint-max*);
RETURN (*D*, *C*, *i*+*one-uint32-nat*)
})
(*D*, *C*, *one-uint32-nat*);
ASSERT(*length outl* ≠ *1* ⟶ *length C* > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (*M*, *C*!*1*));
RETURN ((*True*, *D*), *C*, *if length outl* = *1 then zero-uint32-nat else get-level-pol M* (*C*!*1*))
}›
⟨*proof*⟩

sepref-definition *empty-conflict-and-extract-clause-heur-code*
is ‹*uncurry2* (*isa-empty-conflict-and-extract-clause-heur*)›
:: ‹$[\lambda((M, D), outl).\ outl \neq [] \wedge length\ outl \leq uint\text{-}max]_a$
$trail\text{-}pol\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^d *_a out\text{-}learned\text{-}assn^k \rightarrow$
(*bool-assn* ∗*a uint32-nat-assn* ∗*a array-assn option-bool-assn*) ∗*a clause-ll-assn* ∗*a uint32-nat-assn*›
⟨*proof*⟩

declare *empty-conflict-and-extract-clause-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *empty-conflict-and-extract-clause-heur-fast-code*
is ‹*uncurry2* (*isa-empty-conflict-and-extract-clause-heur*)›
:: ‹$[\lambda((M, D), outl).\ outl \neq [] \wedge length\ outl \leq uint\text{-}max]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^d *_a out\text{-}learned\text{-}assn^k \rightarrow$

(*bool-assn* $*a$ *uint32-nat-assn* $*a$ *array-assn option-bool-assn*) $*a$ *clause-ll-assn* $*a$ *uint32-nat-assn*›
⟨*proof*⟩

declare *empty-conflict-and-extract-clause-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *empty-cach-code*
is ‹*empty-cach-ref-set*›
:: ‹*cach-refinement-l-assn*^{d} $\rightarrow_a$ *cach-refinement-l-assn*›
⟨*proof*⟩

declare *empty-cach-code.refine*[*sepref-fr-rules*]

theorem *empty-cach-code-empty-cach-ref*[*sepref-fr-rules*]:
‹(*empty-cach-code*, *RETURN* ∘ *empty-cach-ref*)
∈ [*empty-cach-ref-pre*]$_a$
cach-refinement-l-assn^d → *cach-refinement-l-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
⟨*proof*⟩

lemma *uint64-of-uint32-uint64-of-nat*[*sepref-fr-rules*]:
‹(*return o uint64-of-uint32*, *RETURN o uint64-of-nat*) ∈ *uint32-nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

sepref-definition *propagate-bt-wl-D-code*
is ‹*uncurry2 propagate-bt-wl-D-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *clause-ll-assn*^{d} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-register *fm-add-new-fast*

Find a less hack-like solution

setup ‹*map-theory-claset* (*fn ctxt* => *ctxt delSWrapper split-all-tac*)›

sepref-definition *propagate-bt-wl-D-fast-code*
is ‹*uncurry2 propagate-bt-wl-D-heur*›
:: ‹[λ((*L*, *C*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k $*_a$ *clause-ll-assn*^{d} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

declare
propagate-bt-wl-D-code.refine[*sepref-fr-rules*]
propagate-bt-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-definition *propagate-unit-bt-wl-D-code*
is ‹*uncurry propagate-unit-bt-wl-D-int*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *propagate-unit-bt-wl-D-fast-code*
is ‹*uncurry propagate-unit-bt-wl-D-int*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare

propagate-unit-bt-wl-D-fast-code.refine[*sepref-fr-rules*]
propagate-unit-bt-wl-D-code.refine[*sepref-fr-rules*]

sepref-register *isa-minimize-and-extract-highest-lookup-conflict*
empty-conflict-and-extract-clause-heur

sepref-definition *extract-shorter-conflict-list-heur-st-code*
is ‹*extract-shorter-conflict-list-heur-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ clause\text{-}ll\text{-}assn$›
⟨*proof*⟩

declare *extract-shorter-conflict-list-heur-st-code.refine*[*sepref-fr-rules*]

sepref-definition *extract-shorter-conflict-list-heur-st-fast*
is ‹*extract-shorter-conflict-list-heur-st*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ clause\text{-}ll\text{-}assn$›
⟨*proof*⟩

declare *extract-shorter-conflict-list-heur-st-fast.refine*[*sepref-fr-rules*]

sepref-register *find-lit-of-max-level-wl*
extract-shorter-conflict-list-heur-st lit-of-hd-trail-st-heur propagate-bt-wl-D-heur
propagate-unit-bt-wl-D-int
sepref-register *backtrack-wl-D*

sepref-definition *lit-of-hd-trail-st-heur-code*
is ‹*RETURN o lit-of-hd-trail-st-heur*›
:: ‹$[\lambda S.\ fst\ (get\text{-}trail\text{-}wl\text{-}heur\ S) \neq []]_a\ isasat\text{-}unbounded\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *lit-of-hd-trail-st-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *lit-of-hd-trail-st-heur-fast-code*
is ‹*RETURN o lit-of-hd-trail-st-heur*›
:: ‹$[\lambda S.\ fst\ (get\text{-}trail\text{-}wl\text{-}heur\ S) \neq []]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *lit-of-hd-trail-st-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *backtrack-wl-D-fast-code*
is ‹*backtrack-wl-D-nlit-heur*›
:: ‹$[isasat\text{-}fast]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *backtrack-wl-D-code*
is ‹*backtrack-wl-D-nlit-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *backtrack-wl-D-fast-code.refine*[*sepref-fr-rules*]
backtrack-wl-D-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Initialisation*

imports *Watched-Literals.Watched-Literals-Watch-List-Initialisation IsaSAT-Setup IsaSAT-VMTF Automatic-Refinement.Relators* — for more lemmas
begin

lemma *fold-eq-nfoldli*:
$RETURN\ (fold\ f\ l\ s) = nfoldli\ l\ (\lambda\text{-}.\ True)\ (\lambda x\ s.\ RETURN\ (f\ x\ s))\ s$
⟨*proof*⟩

no-notation *Ref.update* (- := - *62*)
hide-const *Autoref-Fix-Rel.CONSTRAINT*

0.2 Code for the initialisation of the Data Structure

The initialisation is done in three different steps:

1. First, we extract all the atoms that appear in the problem and initialise the state with empty values. This part is called *initialisation* below.
2. Then, we go over all clauses and insert them in our memory module. We call this phase *parsing*.
3. Finally, we calculate the watch list.

Splitting the second from the third step makes it easier to add preprocessing and more important to add a bounded mode.

0.2.1 Initialisation of the state

definition (**in** −) *atoms-hash-empty* **where**
[*simp*]: ‹*atoms-hash-empty* - = {}›

definition (**in** −) *atoms-hash-int-empty* **where**
‹$atoms\text{-}hash\text{-}int\text{-}empty\ n = RETURN\ (replicate\ n\ False)$›

lemma *atoms-hash-int-empty-atoms-hash-empty*:
‹$(atoms\text{-}hash\text{-}int\text{-}empty,\ RETURN\ o\ atoms\text{-}hash\text{-}empty) \in$
$[\lambda n.\ (\forall L \in \# \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < n)]_f\ nat\text{-}rel \rightarrow \langle atoms\text{-}hash\text{-}rel\ \mathcal{A}\rangle nres\text{-}rel$›
⟨*proof*⟩

definition (**in** −) *distinct-atms-empty* **where**
‹*distinct-atms-empty* - = {}›

definition (**in** −) *distinct-atms-int-empty* **where**
‹$distinct\text{-}atms\text{-}int\text{-}empty\ n = RETURN\ ([],\ replicate\ n\ False)$›

lemma *distinct-atms-int-empty-distinct-atms-empty*:
‹$(distinct\text{-}atms\text{-}int\text{-}empty,\ RETURN\ o\ distinct\text{-}atms\text{-}empty) \in$
$[\lambda n.\ (\forall L \in \# \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < n)]_f\ nat\text{-}rel \rightarrow \langle distinct\text{-}atoms\text{-}rel\ \mathcal{A}\rangle nres\text{-}rel$›
⟨*proof*⟩

type-synonym *vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat set*›

type-synonym *isa-vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat list* × *bool list*›

definition *vmtf-init*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int-option-fst-As set*›
where
‹*vmtf-init* $\mathcal{A}_{in}$ *M* = {((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
$\mathcal{A}_{in}$ ≠ {#} ⟶ (*fst-As* ≠ *None* ∧ *lst-As* ≠ *None* ∧ ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*)}›

definition *isa-vmtf-init* **where**
‹*isa-vmtf-init* $\mathcal{A}$ *M* =
((*Id* $\times_r$ *nat-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel*) $\times_f$
distinct-atoms-rel $\mathcal{A}$)$^{-1}$
‘‘ *vmtf-init* $\mathcal{A}$ *M*›

lemma *isa-vmtf-initI*:
‹(*vm*, *to-remove'*) ∈ *vmtf-init* $\mathcal{A}$ *M* ⟹ (*to-remove*, *to-remove'*) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹
(*vm*, *to-remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ *M*›
⟨*proof*⟩

lemma *isa-vmtf-init-consD*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ⟹
((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ (*L* # *M*)›
⟨*proof*⟩

lemma *vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf-init* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf-init* $\mathcal{B}$ *M*›
⟨*proof*⟩

lemma *isa-vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ⟹ *L* ∈ *isa-vmtf-init* $\mathcal{B}$ *M*›
⟨*proof*⟩

type-synonym *vdom-fast* = ‹*uint64 list*›

type-synonym (**in** −) *twl-st-wl-heur-init* =
‹*trail-pol* × *arena* × *conflict-option-rel* × *nat* ×
(*nat* × *nat literal* × *bool*) *list list* × *isa-vmtf-remove-int-option-fst-As* × *bool list* ×
nat × *conflict-min-cach-l* × *lbd* × *vdom* × *bool*›

type-synonym (**in** −) *twl-st-wl-heur-init-full* =
‹*trail-pol* × *arena* × *conflict-option-rel* × *nat* ×
(*nat* × *nat literal* × *bool*) *list list* × *isa-vmtf-remove-int-option-fst-As* × *bool list* ×
nat × *conflict-min-cach-l* × *lbd* × *vdom* × *bool*›

The initialisation relation is stricter in the sense that it already includes the relation of atom inclusion.

Remark that we replace $D = None \longrightarrow j \leq length\ M$ by $j \leq length\ M$: this simplifies the proofs and does not make a difference in the generated code, since there are no conflict analysis at that level anyway.

KILL duplicates below, but difference: vmtf vs vmtf_init watch list vs no WL OC vs non-OC

definition *twl-st-heur-parsing-no-WL*
:: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init* × *nat twl-st-wl-init*) *set*›
where
‹*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* =
{((M′, N′, D′, j, W′, vm, φ, clvls, cach, lbd, vdom, failed), ((M, N, D, NE, UE, Q), OC)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena* N′ N (*set vdom*) ∧
set-mset
(*all-lits-of-mm*
({#*mset* (*fst* x). x ∈# *ran-m* N#} + NE + UE)) ⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
mset vdom = *dom-m* N)) ∧
(M′, M) ∈ *trail-pol* $\mathcal{A}$ ∧
(D′, D) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length* M ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) ∧
vm ∈ *isa-vmtf-init* $\mathcal{A}$ M ∧
phase-saving $\mathcal{A}$ φ ∧
no-dup M ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧
(W′, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
distinct vdom
}›

definition *twl-st-heur-parsing*
:: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init* × (*nat twl-st-wl* × *nat clauses*)) *set*›
where
‹*twl-st-heur-parsing* $\mathcal{A}$ *unbdd* =
{((M′, N′, D′, j, W′, vm, φ, clvls, cach, lbd, vdom, failed), ((M, N, D, NE, UE, Q, W), OC)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
((M′, M) ∈ *trail-pol* $\mathcal{A}$ ∧
valid-arena N′ N (*set vdom*) ∧
(D′, D) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length* M ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) ∧
vm ∈ *isa-vmtf-init* $\mathcal{A}$ M ∧
phase-saving $\mathcal{A}$ φ ∧
no-dup M ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧
mset vdom = *dom-m* N ∧
vdom-m $\mathcal{A}$ W N = *set-mset* (*dom-m* N) ∧
set-mset
(*all-lits-of-mm*
({#*mset* (*fst* x). x ∈# *ran-m* N#} + NE + UE)) ⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
(W′, W) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
distinct vdom))
}›

definition *twl-st-heur-parsing-no-WL-wl* :: ‹*nat multiset* ⇒ *bool* ⇒ (- × *nat twl-st-wl-init′*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd* =
{((M′, N′, D′, j, W′, vm, φ, clvls, cach, lbd, vdom, failed), (M, N, D, NE, UE, Q)).

$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$(valid\text{-}arena\ N'\ N\ (set\ vdom) \wedge set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom)) \wedge$
$(M', M) \in trail\text{-}pol\ \mathcal{A} \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in isa\text{-}vmtf\text{-}init\ \mathcal{A}\ M \wedge$
$phase\text{-}saving\ \mathcal{A}\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', empty\text{-}watched\ \mathcal{A}) \in \langle Id \rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$distinct\ vdom$
$\}\rangle$

definition *twl-st-heur-parsing-no-WL-wl-no-watched* :: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init-full* × *nat twl-st-wl-init*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl-no-watched* $\mathcal{A}$ *unbdd* =
$\{((M', N', D', j, W', vm, \varphi, clvls, cach, lbd, vdom, failed), ((M, N, D, NE, UE, Q), OC)).$
$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$(valid\text{-}arena\ N'\ N\ (set\ vdom) \wedge set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom)) \wedge (M', M) \in trail\text{-}pol\ \mathcal{A} \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in isa\text{-}vmtf\text{-}init\ \mathcal{A}\ M \wedge$
$phase\text{-}saving\ \mathcal{A}\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', empty\text{-}watched\ \mathcal{A}) \in \langle Id \rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$distinct\ vdom$
$\}\rangle$

definition *twl-st-heur-post-parsing-wl* :: ‹*bool* ⇒ (*twl-st-wl-heur-init-full* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-post-parsing-wl unbdd* =
$\{((M', N', D', j, W', vm, \varphi, clvls, cach, lbd, vdom, failed), (M, N, D, NE, UE, Q, W)).$
$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$((M', M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE)) \wedge$
$set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom \wedge$
$valid\text{-}arena\ N'\ N\ (set\ vdom))) \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE)) \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in isa\text{-}vmtf\text{-}init\ (all\text{-}atms\ N\ (NE + UE))\ M \wedge$
$phase\text{-}saving\ (all\text{-}atms\ N\ (NE + UE))\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE))\ cach \wedge$
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE))\ W\ N \subseteq set\ vdom \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE))$

$\subseteq$ *set-mset* ($\mathcal{L}_{all}$ (*all-atms* N (NE + UE))) $\wedge$
(W', W) $\in$ $\langle Id\rangle$*map-fun-rel* (D_0 (*all-atms* N (NE + UE))) $\wedge$
isasat-input-bounded (*all-atms* N (NE + UE)) $\wedge$
distinct vdom
}›

VMTF

definition *initialise-VMTF* :: ‹*uint32 list* ⇒ *nat* ⇒ *isa-vmtf-remove-int-option-fst-As nres*› **where**
‹*initialise-VMTF N n* = *do* {
let A = *replicate n* (*VMTF-Node zero-uint64-nat None None*);
to-remove ← *distinct-atms-int-empty n*;
ASSERT(*length N* ≤ *uint32-max*);
(n, A, *cnext*) ← $WHILE_T$
(λ(i, A, *cnext*). i < *length-uint32-nat* N)
(λ(i, A, *cnext*). *do* {
ASSERT(i < *length-uint32-nat* N);
let L = *nat-of-uint32* (N ! i);
ASSERT(L < *length A*);
ASSERT(*cnext* ≠ *None* ⟶ *the cnext* < *length A*);
ASSERT(i + *1* ≤ *uint-max*);
RETURN (i + *one-uint32-nat*, *vmtf-cons A L cnext* (*uint64-of-uint32-conv i*), *Some L*)
})
(*zero-uint32-nat*, A, *None*);
RETURN ((A, *uint64-of-uint32-conv n*, *cnext*, (*if* N = [] *then None else Some* (*nat-of-uint32* (N!*0*))), *cnext*), *to-remove*)
}›

lemma *initialise-VMTF*:
shows ‹(*uncurry initialise-VMTF*, *uncurry* (λN n. *RES* (*vmtf-init N* []))) ∈
[λ(N,n). (∀ L∈# N. L < n) ∧ (*distinct-mset N*) ∧ *size N* < *uint32-max* ∧ *set-mset N* = *set-mset* $\mathcal{A}]_f$
(⟨*uint32-nat-rel*⟩*list-rel-mset-rel*) $\times_f$ *nat-rel* →
⟨((⟨*Id*⟩*list-rel* $\times_r$ *nat-rel* $\times_r$ ⟨*nat-rel*⟩ *option-rel* $\times_r$ ⟨*nat-rel*⟩ *option-rel* $\times_r$ ⟨*nat-rel*⟩ *option-rel*)
$\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
(**is** ‹(*?init*, *?R*) ∈ -›)
⟨*proof*⟩

0.2.2 Parsing

fun (**in** −)*get-conflict-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *conflict-option-rel*› **where**
‹*get-conflict-wl-heur-init* (-, -, D, -) = D›

fun (**in** −)*get-clauses-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *arena*› **where**
‹*get-clauses-wl-heur-init* (-, N, -) = N›

fun (**in** −) *get-trail-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *trail-pol*› **where**
‹*get-trail-wl-heur-init* (M, -, -, -, -, -, -) = M›

fun (**in** −) *get-vdom-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *nat list*› **where**
‹*get-vdom-heur-init* (-, -, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*›

fun (**in** −) *is-failed-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool*› **where**
‹*is-failed-heur-init* (-, -, -, -, -, -, -, -, -, -, -, *failed*) = *failed*›

definition *propagate-unit-cls*
:: ⟨*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*⟩
where
⟨*propagate-unit-cls* = (λL ((M, N, D, NE, UE, Q), OC).
(($Propagated\ L\ 0$ # M, N, D, *add-mset* {#L#} NE, UE, Q), OC))⟩

definition *propagate-unit-cls-heur*
:: ⟨*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩
where
⟨*propagate-unit-cls-heur* = (λL (M, N, D, Q). *do* {
ASSERT(*cons-trail-Propagated-tr-pre* ((L, *0* :: *nat*), M));
RETURN (*cons-trail-Propagated-tr* L *0* M, N, D, Q)})⟩

fun *get-unit-clauses-init-wl* :: ⟨$'v$ *twl-st-wl-init* ⇒ $'v$ *clauses*⟩ **where**
⟨*get-unit-clauses-init-wl* ((M, N, D, NE, UE, Q), OC) = NE + UE⟩

abbreviation *all-lits-st-init* :: ⟨$'v$ *twl-st-wl-init* ⇒ $'v$ *literal multiset*⟩ **where**
⟨*all-lits-st-init* S ≡ *all-lits* (*get-clauses-init-wl* S) (*get-unit-clauses-init-wl* S)⟩

definition *all-atms-init* :: ⟨- ⇒ - ⇒ $'v$ *multiset*⟩ **where**
⟨*all-atms-init* N NUE = *atm-of* '# *all-lits* N NUE⟩

abbreviation *all-atms-st-init* :: ⟨$'v$ *twl-st-wl-init* ⇒ $'v$ *multiset*⟩ **where**
⟨*all-atms-st-init* S ≡ *atm-of* '# *all-lits-st-init* S⟩

lemma *DECISION-REASON0*[*simp*]: ⟨*DECISION-REASON* ≠ *0*⟩
⟨*proof*⟩

lemma *propagate-unit-cls-heur-propagate-unit-cls*:
⟨(*uncurry propagate-unit-cls-heur*, *uncurry* (*RETURN oo propagate-unit-init-wl*)) ∈
[λ(L, S). *undefined-lit* (*get-trail-init-wl* S) L ∧ L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*⟩
⟨*proof*⟩

definition *already-propagated-unit-cls*
:: ⟨*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*⟩
where
⟨*already-propagated-unit-cls* = (λL ((M, N, D, NE, UE, Q), OC).
((M, N, D, *add-mset* {#L#} NE, UE, Q), OC))⟩

definition *already-propagated-unit-cls-heur*
:: ⟨*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩
where
⟨*already-propagated-unit-cls-heur* = (λL (M, N, D, Q, *oth*).
RETURN (M, N, D, Q, *oth*))⟩

lemma *already-propagated-unit-cls-heur-already-propagated-unit-cls*:
⟨(*uncurry already-propagated-unit-cls-heur*, *uncurry* (*RETURN oo already-propagated-unit-init-wl*)) ∈
[λ(C, S). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ C]$_f$
list-mset-rel $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*⟩
⟨*proof*⟩

definition (**in** −) *set-conflict-unit* :: ⟨*nat literal* ⇒ *nat clause option* ⇒ *nat clause option*⟩ **where**
⟨*set-conflict-unit* L - = *Some* {#L#}⟩

definition *set-conflict-unit-heur* **where**
‹*set-conflict-unit-heur* = (λ *L* (*b*, *n*, *xs*). *RETURN* (*False*, *1*, *xs*[*atm-of L* := *Some* (*is-pos L*)]))›

lemma *set-conflict-unit-heur-set-conflict-unit*:
‹(*uncurry set-conflict-unit-heur*, *uncurry* (*RETURN oo set-conflict-unit*)) $\in$
[λ(*L*, *D*). *D* = *None* $\wedge$ *L* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ $\rightarrow$
⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *conflict-propagated-unit-cls*
:: ‹*nat literal* $\Rightarrow$ *nat twl-st-wl-init* $\Rightarrow$ *nat twl-st-wl-init*›
where
‹*conflict-propagated-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*).
((*M*, *N*, *set-conflict-unit L D*, *add-mset* {#*L*#} *NE*, *UE*, {#}), *OC*))›

definition *conflict-propagated-unit-cls-heur*
:: ‹*nat literal* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*›
where
‹*conflict-propagated-unit-cls-heur* = (λ*L* (*M*, *N*, *D*, *Q*, *oth*). *do* {
ASSERT(*atm-of L* < *length* (*snd* (*snd D*)));
D $\leftarrow$ *set-conflict-unit-heur L D*;
ASSERT(*isa-length-trail-pre M*);
RETURN (*M*, *N*, *D*, *isa-length-trail M*, *oth*)
})›

lemma *conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls*:
‹(*uncurry conflict-propagated-unit-cls-heur*, *uncurry* (*RETURN oo set-conflict-init-wl*)) $\in$
[λ(*L*, *S*). *L* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *get-conflict-init-wl S* = *None*]$_f$
nat-lit-lit-rel $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
nres-rel›
⟨*proof*⟩

definition *add-init-cls-heur*
:: ‹*bool* $\Rightarrow$ *nat clause-l* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur unbdd* = (λ*C* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*). *do* {
let C = *C*;
ASSERT(*length C* $\leq$ *uint-max* + *2*);
ASSERT(*length C* $\geq$ *2*);
if unbdd $\vee$ (*length N* $\leq$ *uint64-max* $-$ *length C* $-$ *5* $\wedge$ $\neg$*failed*)
then do {
ASSERT(*length vdom* $\leq$ *length N*);
(*N*, *i*) $\leftarrow$ *fm-add-new True C N*;
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom* @ [*nat-of-uint32-conv i*], *failed*)
} *else RETURN* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *True*)})›

definition *add-init-cls-heur-unb* :: ‹*nat clause-l* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-unb* = *add-init-cls-heur True*›

definition *add-init-cls-heur-b* :: ‹*nat clause-l* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b* = *add-init-cls-heur False*›

lemma *length-C-nempty-iff*: ‹*length C* $\geq$ *2* $\longleftrightarrow$ *C* $\neq$ [] $\wedge$ *tl C* $\neq$ []›
⟨*proof*⟩

context
fixes *unbdd* :: *bool* **and** $\mathcal{A}$ :: ‹*nat multiset*› **and**

x :: ‹*nat literal list* ×
(*nat literal list* ×
bool option list × *nat list* × *nat list* × *nat* × *nat list*) ×
arena-el list ×
(*bool* × *nat* × *bool option list*) ×
nat ×
(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*› **and** y :: ‹*nat literal list* ×
((*nat literal*, *nat literal*,
nat) *annotated-lit list* ×
(*nat*, *nat literal list* × *bool*) *fmap* ×
nat literal multiset option ×
nat literal multiset multiset ×
nat literal multiset multiset ×
nat literal multiset) ×
nat literal multiset multiset› **and** $x1$ :: ‹*nat literal list*› **and** $x2$ :: ‹((*nat literal*,
nat literal, *nat*) *annotated-lit list* ×
(*nat*, *nat literal list* × *bool*) *fmap* ×
nat literal multiset option ×
nat literal multiset multiset ×
nat literal multiset multiset ×
nat literal multiset) ×
nat literal multiset multiset› **and** $x1a$ :: ‹(*nat literal*,
nat literal, *nat*) *annotated-lit list* ×
(*nat*, *nat literal list* × *bool*) *fmap* ×
nat literal multiset option ×
nat literal multiset multiset ×
nat literal multiset multiset ×
nat literal multiset› **and** $x1b$ :: ‹(*nat literal*,
nat literal,
nat) *annotated-lit list*› **and** $x2a$ :: ‹(*nat*,
nat literal list × *bool*) *fmap* ×
nat literal multiset option ×
nat literal multiset multiset ×
nat literal multiset multiset ×
nat literal multiset› **and** $x1c$ :: ‹(*nat*,
nat literal list ×
bool) *fmap*› **and** $x2b$ :: ‹*nat literal multiset option* ×
nat literal multiset multiset ×
nat literal multiset multiset ×
nat literal multiset› **and** $x1d$ :: ‹*nat literal multiset option*› **and** $x2c$ ::
‹*nat literal multiset multiset* ×
nat literal multiset multiset ×
nat literal multiset› **and** $x1e$ :: ‹*nat literal multiset multiset*› **and** $x2d$ :: ‹*nat literal multiset multiset* ×
nat literal multiset› **and** $x1f$ :: ‹*nat literal multiset multiset*› **and** $x2e$ :: ‹*nat literal multiset*› **and** $x2f$ :: ‹*nat literal multiset multiset*› **and** $x1g$ :: ‹*nat literal list*› **and** $x2g$ :: ‹(*nat literal list* ×
bool option list × *nat list* × *nat list* × *nat* × *nat list*) ×

arena-el list ×
(*bool* × *nat* × *bool option list*) ×
nat ×
(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1h* :: ⟨*nat literal list* ×
bool option list ×
nat list ×
nat list ×
nat ×
nat list⟩ **and** *x2h* :: ⟨*arena-el list* ×
(*bool* × *nat* × *bool option list*) ×
nat ×
(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1i* :: ⟨*arena-el list*⟩ **and** *x2i* :: ⟨(*bool* ×
nat × *bool option list*) ×
nat ×
(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1j* :: ⟨*bool* ×
nat ×
bool option list⟩ **and** *x2j* :: ⟨*nat* ×
(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1k* :: ⟨*nat*⟩ **and** *x2k* :: ⟨(*nat* × *nat literal* × *bool*) *list list* ×
(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×

nat list × *bool*⟩ **and** *x1l* :: ⟨(*nat* ×
nat literal ×
bool) *list list*⟩ **and** *x2l* :: ⟨(((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list × *bool list*) ×
bool list ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × -⟩ **and** *x1m* :: ⟨((*nat*, *nat*) *vmtf-node list* ×
nat × *nat option* × *nat option* × *nat option*) ×
nat list ×
bool list⟩ **and** *x2m* :: ⟨*bool list* ×
nat ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1n* :: ⟨*bool list*⟩ **and** *x2n* :: ⟨*nat* ×
(*minimize-status list* × *nat list*) ×
bool list ×
nat list × *bool*⟩ **and** *x1o* :: ⟨*nat*⟩ **and** *x2o* :: ⟨(*minimize-status list* ×
nat list) ×
bool list ×
nat list × *bool*⟩ **and** *x1p* :: ⟨*minimize-status list* ×
nat list⟩ **and** *x2p* :: ⟨*bool list* ×
nat list × *bool*⟩ **and** *x1q* :: ⟨*bool list*⟩ **and** *x2q* :: ⟨*nat list* × *bool*⟩ **and** *x1r'* :: ⟨*nat list*⟩ **and** *x2r'* :: *bool*

assumes

pre: ⟨*case y of*
$(C, S) \Rightarrow 2 \leq$ *length C* $\wedge$ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) $\wedge$ *distinct C*⟩ **and**
xy: ⟨$(x, y) \in$ *Id* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ **and**
st:
⟨*x2d* = (*x1f*, *x2e*)⟩
⟨*x2c* = (*x1e*, *x2d*)⟩
⟨*x2b* = (*x1d*, *x2c*)⟩
⟨*x2a* = (*x1c*, *x2b*)⟩
⟨*x1a* = (*x1b*, *x2a*)⟩
⟨*x2* = (*x1a*, *x2f*)⟩
⟨*y* = (*x1*, *x2*)⟩
⟨*x2q* = (*x1r'*, *x2r'*)⟩
⟨*x2p* = (*x1q*, *x2q*)⟩
⟨*x2o* = (*x1p*, *x2p*)⟩
⟨*x2n* = (*x1o*, *x2o*)⟩
⟨*x2m* = (*x1n*, *x2n*)⟩
⟨*x2l* = (*x1m*, *x2m*)⟩
⟨*x2k* = (*x1l*, *x2l*)⟩
⟨*x2j* = (*x1k*, *x2k*)⟩
⟨*x2i* = (*x1j*, *x2j*)⟩
⟨*x2h* = (*x1i*, *x2i*)⟩
⟨*x2g* = (*x1h*, *x2h*)⟩
⟨*x* = (*x1g*, *x2g*)⟩

begin

lemma *add-init-pre1*: ⟨*length x1g* $\leq$ *uint-max* + *2*⟩
⟨*proof*⟩

lemma *add-init-pre2*: ⟨*2* $\leq$ *length x1g*⟩

⟨*proof*⟩ **lemma**
x1g-x1: ‹$x1g = x1$› **and**
‹$(x1h, x1b) \in$ *trail-pol* $\mathcal{A}$› **and**
valid: ‹ *valid-arena x1i x1c* (*set* $x1r'$)› **and**
‹$(x1j, x1d) \in$ *option-lookup-clause-rel* $\mathcal{A}$› **and** ‹$x1k \leq$ *length x1b*› **and**
‹$x2e = \{\#-$ *lit-of x*. $x \in\#$ *mset* (*drop x1k* (*rev x1b*))$\#\}$› **and**
‹$x1m \in$ *isa-vmtf-init* $\mathcal{A}$ *x1b*› **and**
‹*phase-saving* $\mathcal{A}$ *x1n*› **and**
‹*no-dup x1b*› **and**
‹*cach-refinement-empty* $\mathcal{A}$ *x1p*› **and**
vdom: ‹*mset* $x1r'$ = *dom-m x1c*› **and**
var-incl:
‹*set-mset* (*all-lits-of-mm* ($\{\#$*mset* (*fst x*). $x \in\#$ *ran-m x1c*$\#\}$ + *x1e* + *x1f*))
$\subseteq$ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
watched: ‹(*x1l*, *empty-watched* $\mathcal{A}$) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
if ‹$\neg x2r'$ $\vee$ *unbdd*›
⟨*proof*⟩

lemma *init-fm-add-new*:
‹$\neg x2r'$ $\vee$ *unbdd* $\Longrightarrow$ *fm-add-new True x1g x1i*
$\leq \Downarrow$ $\{((arena, i), (N', i')).$ *valid-arena arena* N' (*insert i* (*set* $x1r'$)) $\wedge i = i' \wedge$
$i \notin\#$ *dom-m x1c* $\wedge$ i = *length x1i* + *header-size x1g* $\wedge$
$i \notin$ *set* $x1r'\}$
(*SPEC*
($\lambda(N', ia)$.
$0 < ia \wedge ia \notin\#$ *dom-m x1c* $\wedge$ N' = *fmupd ia* (*x1*, *True*) *x1c*))›
(**is** ‹- $\Longrightarrow$ - $\leq \Downarrow$ *?qq* -›)
⟨*proof*⟩

lemma *add-init-cls-final-rel*:
fixes *xa* :: ‹*arena-el list* $\times$
nat› **and** x' :: ‹(*nat*, *nat literal list* $\times$ *bool*) *fmap* $\times$
nat› **and** *x1r* :: ‹(*nat*,
nat literal list $\times$
bool) *fmap*› **and** *x2r* :: ‹*nat*› **and** *x1s* :: ‹*arena-el list*› **and** *x2s* :: ‹*nat*›
assumes
‹(xa, x')
$\in \{((arena, i), (N', i')).$ *valid-arena arena* N' (*insert i* (*set* $x1r'$)) $\wedge i = i' \wedge$
$i \notin\#$ *dom-m x1c* $\wedge$ i = *length x1i* + *header-size x1g* $\wedge$
$i \notin$ *set* $x1r'\}$› **and**
‹$x' \in \{(N', ia)$.
$0 < ia \wedge ia \notin\#$ *dom-m x1c* $\wedge$ N' = *fmupd ia* (*x1*, *True*) *x1c*$\}$› **and**
‹x' = (*x1r*, *x2r*)› **and**
‹*xa* = (*x1s*, *x2s*)›
shows ‹((*x1h*, *x1s*, *x1j*, *x1k*, *x1l*, *x1m*, *x1n*, *x1o*, *x1p*, *x1q*,
$x1r'$ @ [*nat-of-uint32-conv x2s*], $x2r'$),
(*x1b*, *x1r*, *x1d*, *x1e*, *x1f*, *x2e*), *x2f*)
$\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
⟨*proof*⟩
end

lemma *add-init-cls-heur-add-init-cls*:
‹(*uncurry* (*add-init-cls-heur unbdd*), *uncurry* (*add-to-clauses-init-wl*)) $\in$

$[\lambda(C, S).\ length\ C \geq 2 \wedge literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C) \wedge distinct\ C]_f$
$Id \times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow \langle$*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*$\rangle$ *nres-rel*›
⟨*proof*⟩

definition *already-propagated-unit-cls-conflict*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls-conflict* = (λL ((M, N, D, NE, UE, Q), OC).
((M, N, D, *add-mset* {#L#} NE, UE, {#}), OC))›

definition *already-propagated-unit-cls-conflict-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-conflict-heur* = (λL (M, N, D, Q, *oth*). *do* {
ASSERT (*isa-length-trail-pre* M);
RETURN (M, N, D, *isa-length-trail* M, *oth*)
})›

lemma *already-propagated-unit-cls-conflict-heur-already-propagated-unit-cls-conflict*:
‹(*uncurry already-propagated-unit-cls-conflict-heur*,
uncurry (*RETURN oo already-propagated-unit-cls-conflict*)) ∈
$[\lambda(L, S).\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ $Id \times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow \langle$*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*$\rangle$ *nres-rel*›
⟨*proof*⟩

definition (**in** −) *set-conflict-empty* :: ‹*nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-empty* - = *Some* {#}›

definition (**in** −) *lookup-set-conflict-empty* :: ‹*conflict-option-rel* ⇒ *conflict-option-rel*› **where**
‹*lookup-set-conflict-empty* = (λ(b, s) . (*False*, s))›

lemma *lookup-set-conflict-empty-set-conflict-empty*:
‹(*RETURN o lookup-set-conflict-empty*, *RETURN o set-conflict-empty*) ∈
$[\lambda D.\ D = None]_f$ *option-lookup-clause-rel* $\mathcal{A} \rightarrow \langle$*option-lookup-clause-rel* $\mathcal{A}\rangle$*nres-rel*›
⟨*proof*⟩

definition *set-empty-clause-as-conflict-heur*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*set-empty-clause-as-conflict-heur* = (λ (M, N, (-, (n, xs)), Q, WS). *do* {
ASSERT(*isa-length-trail-pre* M);
RETURN (M, N, (*False*, (n, xs)), *isa-length-trail* M, WS)})›

lemma *set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict*:
‹(*set-empty-clause-as-conflict-heur*, *RETURN o add-empty-conflict-init-wl*) ∈
$[\lambda S.\ get\text{-}conflict\text{-}init\text{-}wl\ S = None]_f$
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* $\rightarrow \langle$*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*$\rangle$ *nres-rel*›
⟨*proof*⟩

definition (**in** −) *add-clause-to-others-heur*
:: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-clause-to-others-heur* = (λ - (M, N, D, Q, WS).
RETURN (M, N, D, Q, WS))›

lemma *add-clause-to-others-heur-add-clause-to-others*:

‹(*uncurry add-clause-to-others-heur*, *uncurry* (*RETURN oo add-to-other-init*)) ∈ ⟨*Id*⟩*list-rel* $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
⟨*proof*⟩

definition (**in** −)*list-length-1* **where**
[*simp*]: ‹*list-length-1 C* ⟷ *length C* = *1*›

definition (**in** −)*list-length-1-code* **where**
‹*list-length-1-code C* ⟷ (*case C of* [-] ⇒ *True* | - ⇒ *False*)›

definition (**in** −) *get-conflict-wl-is-None-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-heur-init* = (λ(*M*, *N*, (*b*, -), *Q*, -). *b*)›

definition *init-dt-step-wl-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-step-wl-heur unbdd C S* = *do* {
  if get-conflict-wl-is-None-heur-init S
  then do {
    if is-Nil C
    then set-empty-clause-as-conflict-heur S
    else if list-length-1 C
    then do {
      ASSERT (*C* ≠ []);
      let L = *hd C*;
      ASSERT(*polarity-pol-pre* (*get-trail-wl-heur-init S*) *L*);
      let val-L = *polarity-pol* (*get-trail-wl-heur-init S*) *L*;
      if val-L = *None*
      then propagate-unit-cls-heur L S
      else
        if val-L = *Some True*
        then already-propagated-unit-cls-heur C S
        else conflict-propagated-unit-cls-heur L S
    }
    else do {
      ASSERT(*length C* ≥ *2*);
      add-init-cls-heur unbdd C S
    }
  }
  else add-clause-to-others-heur C S
}›

named-theorems *twl-st-heur-parsing-no-WL*
lemma [*twl-st-heur-parsing-no-WL*]:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
shows ‹(*get-trail-wl-heur-init S*, *get-trail-init-wl T*) ∈ *trail-pol* $\mathcal{A}$›
⟨*proof*⟩

definition *get-conflict-wl-is-None-init* :: ‹*nat twl-st-wl-init* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-init* = (λ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*). *is-None D*)›

lemma *get-conflict-wl-is-None-init-alt-def*:

‹*get-conflict-wl-is-None-init S* ⟷ *get-conflict-init-wl S* = *None*›
⟨*proof*⟩

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*:
‹(*RETURN o get-conflict-wl-is-None-heur-init*, *RETURN o get-conflict-wl-is-None-init*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *get-conflict-wl-is-None-init′* **where**
‹*get-conflict-wl-is-None-init′* = *get-conflict-wl-is-None*›

lemma *init-dt-step-wl-heur-init-dt-step-wl*:
‹(*uncurry* (*init-dt-step-wl-heur unbdd*), *uncurry init-dt-step-wl*) ∈
[λ(*C*, *S*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*]$_f$
Id $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
⟨*proof*⟩

lemma (**in** −) *get-conflict-wl-is-None-heur-init-alt-def*:
‹*RETURN o get-conflict-wl-is-None-heur-init* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)›
⟨*proof*⟩

definition *polarity-st-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ - ⇒ *bool option*› **where**
‹*polarity-st-heur-init* = (λ(*M*, -) *L*. *polarity-pol M L*)›

lemma *polarity-st-heur-init-alt-def*:
‹*polarity-st-heur-init S L* = *polarity-pol* (*get-trail-wl-heur-init S*) *L*›
⟨*proof*⟩

definition *polarity-st-init* :: ‹′*v twl-st-wl-init* ⇒ ′*v literal* ⇒ *bool option*› **where**
‹*polarity-st-init S* = *polarity* (*get-trail-init-wl S*)›

lemma *get-conflict-wl-is-None-init*:
‹*get-conflict-init-wl S* = *None* ⟷ *get-conflict-wl-is-None-init S*›
⟨*proof*⟩

definition *init-dt-wl-heur*
:: ‹*bool* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur unbdd CS S* = *nfoldli CS* (λ-. *True*)
(λ*C S*. *do* {
init-dt-step-wl-heur unbdd C S}) *S*›

definition *init-dt-step-wl-heur-unb* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-step-wl-heur-unb* = *init-dt-step-wl-heur True*›

definition *init-dt-wl-heur-unb* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-unb* = *init-dt-wl-heur True*›

definition *init-dt-step-wl-heur-b* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-step-wl-heur-b* = *init-dt-step-wl-heur False*›

definition *init-dt-wl-heur-b* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*init-dt-wl-heur-b* = *init-dt-wl-heur False*›

0.2.3 Extractions of the atoms in the state

definition *init-valid-rep* :: *nat list* ⇒ *nat set* ⇒ *bool* **where**
‹*init-valid-rep xs l* ⟷
(∀ *L*∈*l*. *L* < *length xs*) ∧
(∀ *L* ∈ *l*. (*xs* ! *L*) *mod 2* = *1*) ∧
(∀ *L*. *L* < *length xs* ⟶ (*xs* ! *L*) *mod 2* = *1* ⟶ *L* ∈ *l*)›

definition *isasat-atms-ext-rel* :: ‹((*nat list* × *nat* × *nat list*) × *nat set*) *set*› **where**
‹*isasat-atms-ext-rel* = {((*xs*, *n*, *atms*), *l*).
init-valid-rep xs l ∧
n = *Max* (*insert 0 l*) ∧
length xs < *uint-max* ∧
(∀ *s*∈*set xs*. *s* ≤ *uint64-max*) ∧
finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0*
}›

lemma *distinct-length-le-Suc-Max*:
assumes ‹*distinct* (*b* :: *nat list*)›
shows ‹*length b* ≤ *Suc* (*Max* (*insert 0* (*set b*)))›
⟨*proof*⟩

lemma *isasat-atms-ext-rel-alt-def*:
‹*isasat-atms-ext-rel* = {((*xs*, *n*, *atms*), *l*).
init-valid-rep xs l ∧
n = *Max* (*insert 0 l*) ∧
length xs < *uint-max* ∧
(∀ *s*∈*set xs*. *s* ≤ *uint64-max*) ∧
finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0* ∧
length atms ≤ *Suc n*
}›
⟨*proof*⟩

definition *in-map-atm-of* :: ‹'*a* ⇒ '*a list* ⇒ *bool*› **where**
‹*in-map-atm-of L N* ⟷ *L* ∈ *set N*›

definition (**in** −) *init-next-size* **where**
‹*init-next-size L* = *2* ∗ *L*›

lemma *init-next-size*: ‹*L* ≠ *0* ⟹ *L* + *1* ≤ *uint-max* ⟹ *L* < *init-next-size L*›
⟨*proof*⟩

definition *add-to-atms-ext* **where**
‹*add-to-atms-ext* = (λ*i* (*xs*, *n*, *atms*). *do* {
ASSERT(*i* ≤ *uint-max div 2*);

$ASSERT(length\ xs \leq uint\text{-}max)$;
$ASSERT(length\ atms \leq Suc\ n)$;
$let\ n = max\ i\ n$;
$(if\ i < length\text{-}uint32\text{-}nat\ xs\ then\ do\ \{$
$ASSERT(xs!i \leq uint64\text{-}max)$;
$let\ atms = (if\ xs!i\ AND\ one\text{-}uint64\text{-}nat = one\text{-}uint64\text{-}nat\ then\ atms\ else\ atms\ @\ [i])$;
$RETURN\ (xs[i := (sum\text{-}mod\text{-}uint64\text{-}max\ (xs\ !\ i)\ 2)\ OR\ one\text{-}uint64\text{-}nat], n, atms)$
$\}$
$else\ do\ \{$
$ASSERT(i + 1 \leq uint\text{-}max)$;
$ASSERT(length\text{-}uint32\text{-}nat\ xs \neq 0)$;
$ASSERT(i < init\text{-}next\text{-}size\ i)$;
$RETURN\ ((list\text{-}grow\ xs\ (init\text{-}next\text{-}size\ i)\ zero\text{-}uint64\text{-}nat)[i := one\text{-}uint64\text{-}nat], n,$
$atms\ @\ [i])$
$\})$
$\})$›

lemma *init-valid-rep-upd-OR*:
‹$init\text{-}valid\text{-}rep\ (x1b[x1a := a\ OR\ one\text{-}uint64\text{-}nat])\ x2 \longleftrightarrow$
$init\text{-}valid\text{-}rep\ (x1b[x1a := one\text{-}uint64\text{-}nat])\ x2$ › (**is** ‹$?A \longleftrightarrow ?B$›)
⟨*proof*⟩

lemma *init-valid-rep-insert*:
assumes *val*: ‹$init\text{-}valid\text{-}rep\ x1b\ x2$› **and** *le*: ‹$x1a < length\ x1b$›
shows ‹$init\text{-}valid\text{-}rep\ (x1b[x1a := one\text{-}uint64\text{-}nat])\ (insert\ x1a\ x2)$›
⟨*proof*⟩

lemma *init-valid-rep-extend*:
‹$init\text{-}valid\text{-}rep\ (x1b\ @\ replicate\ n\ 0)\ x2 \longleftrightarrow init\text{-}valid\text{-}rep\ (x1b)\ x2$›
(**is** ‹$?A \longleftrightarrow ?B$› **is** ‹$init\text{-}valid\text{-}rep\ ?x1b\ \text{-} \longleftrightarrow \text{-}$›)
⟨*proof*⟩

lemma *init-valid-rep-in-set-iff*:
‹$init\text{-}valid\text{-}rep\ x1b\ x2 \implies x \in x2 \longleftrightarrow (x < length\ x1b \wedge (x1b!x)\ mod\ 2 = 1)$›
⟨*proof*⟩

lemma *add-to-atms-ext-op-set-insert*:
‹$(uncurry\ add\text{-}to\text{-}atms\text{-}ext, uncurry\ (RETURN\ oo\ Set.insert))$
$\in [\lambda(n, l).\ n \leq uint\text{-}max\ div\ 2]_f\ nat\text{-}rel \times_f isasat\text{-}atms\text{-}ext\text{-}rel \to \langle isasat\text{-}atms\text{-}ext\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *extract-atms-cls* :: ‹$'a\ clause\text{-}l \Rightarrow 'a\ set \Rightarrow 'a\ set$› **where**
‹$extract\text{-}atms\text{-}cls\ C\ \mathcal{A}_{in} = fold\ (\lambda L\ \mathcal{A}_{in}.\ insert\ (atm\text{-}of\ L)\ \mathcal{A}_{in})\ C\ \mathcal{A}_{in}$›

definition *extract-atms-cls-i* :: ‹$nat\ clause\text{-}l \Rightarrow nat\ set \Rightarrow nat\ set\ nres$› **where**
‹$extract\text{-}atms\text{-}cls\text{-}i\ C\ \mathcal{A}_{in} = nfoldli\ C\ (\lambda\text{-}.\ True)$
$(\lambda L\ \mathcal{A}_{in}.\ do\ \{$
$ASSERT(atm\text{-}of\ L \leq uint\text{-}max\ div\ 2)$;
$RETURN(insert\ (atm\text{-}of\ L)\ \mathcal{A}_{in})\})$
$\mathcal{A}_{in}$›

lemma *fild-insert-insert-swap*:
‹$fold\ (\lambda L.\ insert\ (f\ L))\ C\ (insert\ a\ \mathcal{A}_{in}) = insert\ a\ (fold\ (\lambda L.\ insert\ (f\ L))\ C\ \mathcal{A}_{in})$›
⟨*proof*⟩

lemma *extract-atms-cls-alt-def*: ‹$extract\text{-}atms\text{-}cls\ C\ \mathcal{A}_{in} = \mathcal{A}_{in} \cup atm\text{-}of\ `\ set\ C$›

⟨*proof*⟩

lemma *extract-atms-cls-i-extract-atms-cls*:
‹(*uncurry extract-atms-cls-i*, *uncurry* (*RETURN oo extract-atms-cls*))
$\in [\lambda(C, \mathcal{A}_{in}).\ \forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint\text{-}max]_f$
$\langle Id\rangle list\text{-}rel \times_f Id \rightarrow \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

definition *extract-atms-clss*:: ‹*'a clause-l list* ⇒ *'a set* ⇒ *'a set*› **where**
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = *fold extract-atms-cls N* $\mathcal{A}_{in}$›

definition *extract-atms-clss-i* :: ‹*nat clause-l list* ⇒ *nat set* ⇒ *nat set nres*› **where**
‹*extract-atms-clss-i N* $\mathcal{A}_{in}$ = *nfoldli N* (λ-. *True*) *extract-atms-cls-i* $\mathcal{A}_{in}$›

lemma *extract-atms-clss-i-extract-atms-clss*:
‹(*uncurry extract-atms-clss-i*, *uncurry* (*RETURN oo extract-atms-clss*))
$\in [\lambda(N, \mathcal{A}_{in}).\ \forall C{\in}set\ N.\ \forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint\text{-}max]_f$
$\langle Id\rangle list\text{-}rel \times_f Id \rightarrow \langle Id\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *fold-extract-atms-cls-union-swap*:
‹*fold extract-atms-cls N* ($\mathcal{A}_{in}$ ∪ *a*) = *fold extract-atms-cls N* $\mathcal{A}_{in}$ ∪ *a*›
⟨*proof*⟩

lemma *extract-atms-clss-alt-def*:
‹*extract-atms-clss N* $\mathcal{A}_{in} = \mathcal{A}_{in} \cup ((\bigcup C{\in}set\ N.\ atm\text{-}of\ `\ set\ C))$›
⟨*proof*⟩

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss CS'* {})› **for** *CS'*
⟨*proof*⟩

definition *op-extract-list-empty* **where**
‹*op-extract-list-empty* = {}›

definition *extract-atms-clss-imp-empty-rel* **where**
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*replicate 1024 0*, *0*, []))›

lemma *extract-atms-clss-imp-empty-rel*:
‹(λ-. *extract-atms-clss-imp-empty-rel*, λ-. (*RETURN op-extract-list-empty*)) ∈
unit-rel $\rightarrow_f$ ⟨*isasat-atms-ext-rel*⟩ *nres-rel*›
⟨*proof*⟩

lemma *extract-atms-cls-Nil*[*simp*]:
‹*extract-atms-cls* [] $\mathcal{A}_{in} = \mathcal{A}_{in}$›
⟨*proof*⟩

lemma *extract-atms-clss-Cons*[*simp*]:
‹*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)›
⟨*proof*⟩

definition (**in** −) *all-lits-of-atms-m* :: ‹*'a multiset* ⇒ *'a clause*› **where**

‹*all-lits-of-atms-m* N = *poss* N + *negs* N›

lemma (**in** −) *all-lits-of-atms-m-nil*[*simp*]: ‹*all-lits-of-atms-m* {#} = {#}›
⟨*proof*⟩

definition (**in** −) *all-lits-of-atms-mm* :: ‹'*a multiset multiset* ⇒ '*a clause*› **where**
‹*all-lits-of-atms-mm* N = *poss* (⋃# N) + *negs* (⋃# N)›

lemma *all-lits-of-atms-m-all-lits-of-m*:
‹*all-lits-of-atms-m* N = *all-lits-of-m* (*poss* N)›
⟨*proof*⟩

Creation of an initial state

definition *init-dt-wl-heur-spec*
:: ‹*bool* ⇒ *nat multiset* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init* ⇒ *bool*›
where
‹*init-dt-wl-heur-spec unbdd* $\mathcal{A}$ *CS T TOC* ⟷
(∃ T' TOC'. (TOC, TOC') ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧ (T, T') ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧
init-dt-wl-spec CS T' TOC')›

definition *init-state-wl* :: ‹*nat twl-st-wl-init'*› **where**
‹*init-state-wl* = ([], *fmempty*, *None*, {#}, {#}, {#})›

definition *init-state-wl-heur* :: ‹*nat multiset* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*init-state-wl-heur* $\mathcal{A}$ = *do* {
M ← *SPEC*(λM. (M, []) ∈ *trail-pol* $\mathcal{A}$);
D ← *SPEC*(λD. (D, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$);
W ← *SPEC* (λW. (W, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$));
vm ← *RES* (*isa-vmtf-init* $\mathcal{A}$ []);
φ ← *SPEC* (*phase-saving* $\mathcal{A}$);
cach ← *SPEC* (*cach-refinement-empty* $\mathcal{A}$);
let lbd = *empty-lbd*;
let vdom = [];
RETURN (M, [], D, *zero-uint32-nat*, W, *vm*, φ, *zero-uint32-nat*, *cach*, *lbd*, *vdom*, *False*)}›

definition *init-state-wl-heur-fast* **where**
‹*init-state-wl-heur-fast* = *init-state-wl-heur*›

lemma *init-state-wl-heur-init-state-wl*:
‹(λ-. (*init-state-wl-heur* $\mathcal{A}$), λ-. (*RETURN init-state-wl*)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$ *unit-rel* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −)*to-init-state* :: ‹*nat twl-st-wl-init'* ⇒ *nat twl-st-wl-init*› **where**
‹*to-init-state* S = (S, {#})›

definition (**in** −) *from-init-state* :: ‹*nat twl-st-wl-init-full* ⇒ *nat twl-st-wl*› **where**
‹*from-init-state* = *fst*›

definition (**in** −) *to-init-state-code* **where**
‹*to-init-state-code* = *id*›

definition *from-init-state-code* **where**
‹*from-init-state-code* = *id*›

definition (**in** −) *conflict-is-None-heur-wl* **where**
‹*conflict-is-None-heur-wl* = (λ(*M*, *N*, *U*, *D*, -). *is-None D*)›

definition (**in** −) *finalise-init* **where**
‹*finalise-init* = *id*›

0.2.4 Parsing

lemma *init-dt-wl-heur-init-dt-wl*:
‹(*uncurry* (*init-dt-wl-heur unbdd*), *uncurry init-dt-wl*) ∈
[λ(*CS*, *S*). (∀ *C* ∈ *set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)) ∧ *distinct-mset-set* (*mset* ‘ *set CS*)]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
⟨*proof*⟩

definition *rewatch-heur-st*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*rewatch-heur-st* = (λ(*M′*, *N′*, *D′*, *j*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*). *do* {
ASSERT(*length vdom* ≤ *length N′*);
W ← *rewatch-heur vdom N′ W*;
RETURN (*M′*, *N′*, *D′*, *j*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*)
})›

lemma *rewatch-heur-st-correct-watching*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*› **and** *failed*: ‹¬*is-failed-heur-init S*›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* (*get-clauses-init-wl T*))› **and**
‹⋀*x*. *x* ∈# *dom-m* (*get-clauses-init-wl T*) ⟹ *distinct* (*get-clauses-init-wl T* ∝ *x*) ∧
2 ≤ *length* (*get-clauses-init-wl T* ∝ *x*)›
shows ‹*rewatch-heur-st S* ≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *unbdd*)
(*SPEC* (λ((*M*,*N*, *D*, *NE*, *UE*, *Q*, *W*), *OC*). *T* = ((*M*,*N*,*D*,*NE*,*UE*,*Q*), *OC*)∧
correct-watching (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)))›
⟨*proof*⟩

Full Initialisation

definition *rewatch-heur-st-fast* **where**
‹*rewatch-heur-st-fast* = *rewatch-heur-st*›

definition *rewatch-heur-st-fast-pre* **where**
‹*rewatch-heur-st-fast-pre S* =
((∀ *x* ∈ *set* (*get-vdom-heur-init S*). *x* ≤ *uint64-max*) ∧ *length* (*get-clauses-wl-heur-init S*) ≤ *uint64-max*)›

definition *init-dt-wl-heur-full*
:: ‹*bool* ⇒ - ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-full unb CS S* = *do* {
S ← *init-dt-wl-heur unb CS S*;
ASSERT(¬*is-failed-heur-init S*);
rewatch-heur-st S

}⟩

definition *init-dt-wl-heur-full-unb*
:: ⟨- ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩
where
⟨*init-dt-wl-heur-full-unb* = *init-dt-wl-heur-full True*⟩

lemma *init-dt-wl-heur-full-init-dt-wl-full*:
assumes
⟨*init-dt-wl-pre CS T*⟩ **and**
⟨$\forall$ *C*∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)⟩ **and**
⟨*distinct-mset-set* (*mset ' set CS*)⟩ **and**
⟨(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*⟩
shows ⟨*init-dt-wl-heur-full True CS S*
≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*init-dt-wl-full CS T*)⟩
⟨*proof*⟩

lemma *init-dt-wl-heur-full-init-dt-wl-spec-full*:
assumes
⟨*init-dt-wl-pre CS T*⟩ **and**
⟨$\forall$ *C*∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)⟩ **and**
⟨*distinct-mset-set* (*mset ' set CS*)⟩ **and**
⟨(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*⟩
shows ⟨*init-dt-wl-heur-full True CS S*
≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*SPEC* (*init-dt-wl-spec-full CS T*))⟩
⟨*proof*⟩

0.2.5 Conversion to normal state

definition *extract-lits-sorted* **where**
⟨*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {
vars ← — insert_sort_nth2 xs vars*RETURN vars*;
RETURN (*vars*, *n*)
})⟩

definition *lits-with-max-rel* **where**
⟨*lits-with-max-rel* = {((*xs*, *n*), $\mathcal{A}_{in}$). *mset xs* = $\mathcal{A}_{in}$ ∧ *n* = *Max* (*insert 0* (*set xs*)) ∧
length xs < *uint32-max*}⟩

lemma *extract-lits-sorted-mset-set*:
⟨(*extract-lits-sorted*, *RETURN o mset-set*)
∈ *isasat-atms-ext-rel* $\rightarrow_f$ ⟨*lits-with-max-rel*⟩*nres-rel*⟩
⟨*proof*⟩

TODO Move

The value 160 is random (but larger than the default 16 for array lists).

definition *finalise-init-code* :: ⟨*opts* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*finalise-init-code opts* =
(λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*, *cach*,
lbd, *vdom*, -). *do* {
ASSERT(*lst-As* ≠ *None* ∧ *fst-As* ≠ *None*);
let init-stats = (*0::uint64*, *0::uint64*, *0::uint64*, *0::uint64*, *0::uint64*, *0::uint64*, *0::uint64*, *0::uint64*);
let fema = *ema-fast-init*;

let sema = *ema-slow-init*;
let ccount = *restart-info-init*;
let lcount = *zero-uint64-nat*;
RETURN $(M', N', D', Q', W'$, ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*), *to-remove*), φ,
clvls, *cach*, *lbd*, *take 1*(*replicate 160* (*Pos zero-uint32-nat*)), *init-stats*,
fema, *sema*, *ccount*, *vdom*, [], *lcount*, *opts*, [])
})›

lemma *isa-vmtf-init-nemptyD*: ‹((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*)
$\in$ *isa-vmtf-init* $\mathcal{A}$ *au* $\Longrightarrow \mathcal{A} \neq \{\#\} \Longrightarrow\ \exists y.$ *an* = *Some y*›
‹((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*)
$\in$ *isa-vmtf-init* $\mathcal{A}$ *au* $\Longrightarrow \mathcal{A} \neq \{\#\} \Longrightarrow\ \exists y.$ *am* = *Some y*›
⟨*proof*⟩

lemma *isa-vmtf-init-isa-vmtf*: ‹$\mathcal{A} \neq \{\#\} \Longrightarrow$ ((*ak*, *al*, *Some am*, *Some an*, *bc*), *ao*, *bd*)
$\in$ *isa-vmtf-init* $\mathcal{A}$ *au* $\Longrightarrow$ ((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*)
$\in$ *isa-vmtf* $\mathcal{A}$ *au*›
⟨*proof*⟩

lemma *finalise-init-finalise-init-full*:
‹*get-conflict-wl S* = *None* $\Longrightarrow$
all-atms-st S $\neq \{\#\} \Longrightarrow$ *size* (*learned-clss-l* (*get-clauses-wl S*)) = *0* $\Longrightarrow$
$((ops', T), ops, S) \in Id \times_f$ *twl-st-heur-post-parsing-wl True* $\Longrightarrow$
finalise-init-code ops' T $\leq \Downarrow \{(S', T').\ (S', T') \in$ *twl-st-heur* $\wedge$
get-clauses-wl-heur-init T = *get-clauses-wl-heur S'*} (*RETURN* (*finalise-init S*))›
⟨*proof*⟩

lemma *finalise-init-finalise-init*:
‹(*uncurry finalise-init-code*, *uncurry* (*RETURN oo* (λ-. *finalise-init*))) $\in$
[λ(-, *S*::*nat twl-st-wl*). *get-conflict-wl S* = *None* $\wedge$ *all-atms-st S* $\neq \{\#\} \wedge$
size (*learned-clss-l* (*get-clauses-wl S*)) = *0*$]_f$ *Id* $\times_r$
twl-st-heur-post-parsing-wl True → ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *init-rll* :: ‹*nat* ⇒ (*nat*, '*v clause-l* × *bool*) *fmap*› **where**
‹*init-rll n* = *fmempty*›

definition (**in** −) *init-aa* :: ‹*nat* ⇒ '*v list*› **where**
‹*init-aa n* = []›

definition (**in** −) *init-aa'* :: ‹*nat* ⇒ (*clause-status* × *nat* × *nat*) *list*› **where**
‹*init-aa' n* = []›

definition *init-trail-D* :: ‹*uint32 list* ⇒ *nat* ⇒ *nat* ⇒ *trail-pol nres*› **where**
‹*init-trail-D* $\mathcal{A}_{in}$ *n m* = *do* {
let M0 = [];
let cs = [];
let M = *replicate m UNSET*;
let M' = *replicate n zero-uint32-nat*;
let M'' = *replicate n 1*;
RETURN ((*M0*, *M*, *M'*, *M''*, *zero-uint32-nat*, *cs*))
}›

definition *init-trail-D-fast* **where**

‹$init\text{-}trail\text{-}D\text{-}fast = init\text{-}trail\text{-}D$›

definition $init\text{-}state\text{-}wl\text{-}D'$ :: ‹$uint32\ list \times uint32 \Rightarrow (trail\text{-}pol \times \text{-} \times \text{-})\ nres$› **where**
‹$init\text{-}state\text{-}wl\text{-}D' = (\lambda(\mathcal{A}_{in}, n).\ do\ \{$
$ASSERT(Suc\ (2 * (nat\text{-}of\text{-}uint32\ n)) \leq uint32\text{-}max);$
$let\ n = Suc\ (nat\text{-}of\text{-}uint32\ n);$
$let\ m = 2 * n;$
$M \leftarrow init\text{-}trail\text{-}D\ \mathcal{A}_{in}\ n\ m;$
$let\ N = [];$
$let\ D = (True, zero\text{-}uint32\text{-}nat, replicate\ n\ NOTIN);$
$let\ WS = replicate\ m\ [];$
$vm \leftarrow initialise\text{-}VMTF\ \mathcal{A}_{in}\ n;$
$let\ \varphi = replicate\ n\ False;$
$let\ cach = (replicate\ n\ SEEN\text{-}UNKNOWN, []);$
$let\ lbd = empty\text{-}lbd;$
$let\ vdom = [];$
$RETURN\ (M, N, D, zero\text{-}uint32\text{-}nat, WS, vm, \varphi, zero\text{-}uint32\text{-}nat, cach, lbd, vdom, False)$
$\})$›

lemma $init\text{-}trail\text{-}D\text{-}ref$:
‹$(uncurry2\ init\text{-}trail\text{-}D, uncurry2\ (RETURN\ ooo\ (\lambda\ \text{-}\ \text{-}\ \text{-}.\ []))) \in [\lambda((N, n), m).\ mset\ N = \mathcal{A}_{in} \wedge$
$distinct\ N \wedge (\forall L\in set\ N.\ L < n) \wedge m = 2 * n \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}]_f$
$\langle uint32\text{-}nat\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \rightarrow$
$\langle trail\text{-}pol\ \mathcal{A}_{in}\rangle\ nres\text{-}rel$›
$\langle proof\rangle$

definition [$to\text{-}relAPP$]: $mset\text{-}rel\ A \equiv p2rel\ (rel\text{-}mset\ (rel2p\ A))$
lemma $in\text{-}mset\text{-}rel\text{-}eq\text{-}f\text{-}iff$:
‹$(a, b) \in \langle\{(c, a).\ a = f\ c\}\rangle mset\text{-}rel \longleftrightarrow b = f\ `\#\ a$›
$\langle proof\rangle$

lemma $in\text{-}mset\text{-}rel\text{-}eq\text{-}f\text{-}iff\text{-}set$:
‹$\langle\{(c, a).\ a = f\ c\}\rangle mset\text{-}rel = \{(b, a).\ a = f\ `\#\ b\}$›
$\langle proof\rangle$

lemma $init\text{-}state\text{-}wl\text{-}D0$:
‹$(init\text{-}state\text{-}wl\text{-}D', init\text{-}state\text{-}wl\text{-}heur) \in$
$[\lambda N.\ N = \mathcal{A}_{in} \wedge distinct\text{-}mset\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}]_f$
$lits\text{-}with\text{-}max\text{-}rel\ O\ \langle uint32\text{-}nat\text{-}rel\rangle mset\text{-}rel \rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r nat\text{-}rel \times_r \langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_r$
$Id \times_r \langle bool\text{-}rel\rangle list\text{-}rel \times_r Id \times_r Id \times_r Id\rangle nres\text{-}rel$›
(**is** ‹$?C \in [?Pre]_f\ ?arg \rightarrow \langle ?im\rangle nres\text{-}rel$›)
$\langle proof\rangle$

lemma $init\text{-}state\text{-}wl\text{-}D'$:
‹$(init\text{-}state\text{-}wl\text{-}D', init\text{-}state\text{-}wl\text{-}heur) \in$
$[\lambda\mathcal{A}_{in}.\ distinct\text{-}mset\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}]_f$
$lits\text{-}with\text{-}max\text{-}rel\ O\ \langle uint32\text{-}nat\text{-}rel\rangle mset\text{-}rel \rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r nat\text{-}rel \times_r \langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_r$
$Id \times_r \langle bool\text{-}rel\rangle list\text{-}rel \times_r Id \times_r Id \times_r Id \times_r Id\rangle nres\text{-}rel$›

⟨*proof*⟩

lemma *init-state-wl-heur-init-state-wl′*:
‹(*init-state-wl-heur*, *RETURN o* (λ-. *init-state-wl*))
∈ $[\lambda N.\ N = \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *Id* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}_{in}$ *True*⟩*nres-rel*›
⟨*proof*⟩

lemma *all-blits-are-in-problem-init-blits-in*: ‹*all-blits-are-in-problem-init S* ⟹ *blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

lemma *correct-watching-init-blits-in-*$\mathcal{L}_{in}$:
assumes ‹*correct-watching-init S*›
shows ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

fun *append-empty-watched* **where**
‹*append-empty-watched* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *Q*, (λ-. [])), *OC*)›

fun *remove-watched* :: ‹′*v twl-st-wl-init-full* ⇒ ′*v twl-st-wl-init*› **where**
‹*remove-watched* ((*M*, *N*, *D*, *NE*, *UE*, *Q*, -), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*)›

definition *init-dt-wl′* :: ‹′*v clause-l list* ⇒ ′*v twl-st-wl-init* ⇒ ′*v twl-st-wl-init-full nres*› **where**
‹*init-dt-wl′ CS S* = *do*{
  S ← *init-dt-wl CS S*;
  RETURN (*append-empty-watched S*)
}›

lemma *init-dt-wl′-spec*: ‹*init-dt-wl-pre CS S* ⟹ *init-dt-wl′ CS S* ≤ ⇓
({(*S* :: ′*v twl-st-wl-init-full*, *S′* :: ′*v twl-st-wl-init*).
  remove-watched S = *S′*}) (*SPEC* (*init-dt-wl-spec CS S*))›
⟨*proof*⟩

lemma *init-dt-wl′-init-dt*:
‹*init-dt-wl-pre CS S* ⟹ (*S*, *S′*) ∈ *state-wl-l-init* ⟹ ∀ *C*∈*set CS*. *distinct C* ⟹
init-dt-wl′ CS S ≤ ⇓
({(*S* :: ′*v twl-st-wl-init-full*, *S′* :: ′*v twl-st-wl-init*).
  remove-watched S = *S′*} *O state-wl-l-init*) (*init-dt CS S′*)›
⟨*proof*⟩

definition *isasat-init-fast-slow* :: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*isasat-init-fast-slow* =
  (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*).
    RETURN (*trail-pol-slow-of-fast M′*, *N′*, *D′*, *j*, *convert-wlists-to-nat-conv W′*, *vm*, φ,
      clvls, *cach*, *lbd*, *vdom*, *failed*))›

lemma *isasat-init-fast-slow-alt-def*:
‹*isasat-init-fast-slow S* = *RETURN S*›
⟨*proof*⟩

end
theory *IsaSAT-Initialisation-SML*
imports *IsaSAT-Setup-SML IsaSAT-VMTF-SML Watched-Literals.Watched-Literals-Watch-List-Initialisation*
Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Initialisation

begin

abbreviation (**in** −) *vmtf-conc-option-fst-As* **where**
⟨*vmtf-conc-option-fst-As* ≡ (*array-assn vmtf-node-assn* ∗*a uint64-nat-assn* ∗*a option-assn uint32-nat-assn* ∗*a option-assn uint32-nat-assn* ∗*a option-assn uint32-nat-assn*)⟩

type-synonym (**in** −)*vmtf-assn-option-fst-As* =
⟨(*uint32*, *uint64*) *vmtf-node array* × *uint64* × *uint32 option* × *uint32 option* × *uint32 option*⟩

type-synonym (**in** −)*vmtf-remove-assn-option-fst-As* =
⟨*vmtf-assn-option-fst-As* × (*uint32 array-list32*) × *bool array*⟩

abbreviation *vmtf-remove-conc-option-fst-As*
:: ⟨*isa-vmtf-remove-int-option-fst-As* ⇒ *vmtf-remove-assn-option-fst-As* ⇒ *assn*⟩
where
⟨*vmtf-remove-conc-option-fst-As* ≡ *vmtf-conc-option-fst-As* ∗*a distinct-atoms-assn*⟩

sepref-register *atoms-hash-empty*
sepref-definition (**in** −) *atoms-hash-empty-code*
is ⟨*atoms-hash-int-empty*⟩
:: ⟨$nat\text{-}assn^k \rightarrow_a$ *phase-saver-conc*⟩
⟨*proof*⟩

find-theorems *replicate arl64-assn*
sepref-definition *distinct-atms-empty-code*
is ⟨*distinct-atms-int-empty*⟩
:: ⟨$nat\text{-}assn^k \rightarrow_a$ *arl32-assn uint32-nat-assn* ∗*a atoms-hash-assn*⟩
⟨*proof*⟩

declare *distinct-atms-empty-code.refine*[*sepref-fr-rules*]

type-synonym (**in** −)*twl-st-wll-trail-init* =
⟨*trail-pol-fast-assn* × *isasat-clauses-fast-assn* × *option-lookup-clause-assn* × *uint32* × *watched-wl-uint32* × *vmtf-remove-assn-option-fst-As* × *phase-saver-assn* × *uint32* × *minimize-assn* × *lbd-assn* × *vdom-fast-assn* × *bool*⟩

definition *isasat-init-assn*
:: ⟨*twl-st-wl-heur-init* ⇒ *twl-st-wll-trail-init* ⇒ *assn*⟩
where
⟨*isasat-init-assn* =
trail-pol-fast-assn ∗*a arena-fast-assn* ∗*a*
isasat-conflict-assn ∗*a*
uint32-nat-assn ∗*a*
watchlist-fast-assn ∗*a*
vmtf-remove-conc-option-fst-As ∗*a phase-saver-conc* ∗*a*
uint32-nat-assn ∗*a*
cach-refinement-l-assn ∗*a*
lbd-assn ∗*a*
vdom-fast-assn ∗*a*
bool-assn⟩

type-synonym (**in** −)*twl-st-wll-trail-init-unbounded* =
⟨*trail-pol-assn* × *isasat-clauses-assn* × *option-lookup-clause-assn* × *uint32* × *watched-wl* × *vmtf-remove-assn-option-fst-As* × *phase-saver-assn* × *uint32* × *minimize-assn* × *lbd-assn* × *vdom-assn* × *bool*⟩

definition *isasat-init-unbounded-assn*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wll-trail-init-unbounded* ⇒ *assn*›
where
‹*isasat-init-unbounded-assn* =
trail-pol-assn ∗*a* *arena-assn* ∗*a*
isasat-conflict-assn ∗*a*
uint32-nat-assn ∗*a*
watchlist-assn ∗*a*
vmtf-remove-conc-option-fst-As ∗*a* *phase-saver-conc* ∗*a*
uint32-nat-assn ∗*a*
cach-refinement-l-assn ∗*a*
lbd-assn ∗*a*
vdom-assn ∗*a*
bool-assn›

sepref-definition *initialise-VMTF-code*
is ‹*uncurry initialise-VMTF*›
:: ‹$[\lambda(N, n).\ True]_a$ $(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow vmtf\text{-}remove\text{-}conc\text{-}option\text{-}fst\text{-}As$›
⟨*proof*⟩

declare *initialise-VMTF-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-unit-cls-code*
is ‹*uncurry* (*propagate-unit-cls-heur*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

sepref-definition *propagate-unit-cls-code-unb*
is ‹*uncurry* (*propagate-unit-cls-heur*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *propagate-unit-cls-code-unb.refine*[*sepref-fr-rules*]
propagate-unit-cls-code.refine[*sepref-fr-rules*]

sepref-definition *already-propagated-unit-cls-code*
is ‹*uncurry already-propagated-unit-cls-heur*›
:: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

sepref-definition *already-propagated-unit-cls-code-unb*
is ‹*uncurry already-propagated-unit-cls-heur*›
:: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *already-propagated-unit-cls-code.refine*[*sepref-fr-rules*]
already-propagated-unit-cls-code-unb.refine[*sepref-fr-rules*]

sepref-definition *set-conflict-unit-code*
is ‹*uncurry set-conflict-unit-heur*›
:: ‹$[\lambda(L, (b, n, xs)).\ atm\text{-}of\ L < length\ xs]_a$
$unat\text{-}lit\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d \rightarrow conflict\text{-}option\text{-}rel\text{-}assn$›
⟨*proof*⟩

declare *set-conflict-unit-code.refine*[*sepref-fr-rules*]

sepref-definition *conflict-propagated-unit-cls-code*
  is ‹*uncurry* (*conflict-propagated-unit-cls-heur*)›
  :: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
  ⟨*proof*⟩

sepref-definition *conflict-propagated-unit-cls-code-unb*
  is ‹*uncurry conflict-propagated-unit-cls-heur*›
  :: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
  ⟨*proof*⟩

declare *conflict-propagated-unit-cls-code.refine*[*sepref-fr-rules*]
  conflict-propagated-unit-cls-code-unb.refine[*sepref-fr-rules*]

sepref-register *fm-add-new*

sepref-definition *add-init-cls-code*
  is ‹*uncurry add-init-cls-heur-unb*›
  :: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
  ⟨*proof*⟩

sepref-register *fm-add-new-fast*

lemma *add-init-cls-code-bI*:
  assumes
    ‹*length at* ≤ *Suc* (*Suc uint-max*)› **and**
    ‹*2* ≤ *length at*› **and**
    ‹*length a1'j* ≤ *length a1'a*› **and**
    ‹*length a1'a* ≤ *uint64-max* − *length at* − *5*›
  shows ‹*append-and-length-fast-code-pre* ((*True*, *at*), *a1'a*)› ‹*5* ≤ *uint64-max* − *length at*›
  ⟨*proof*⟩

lemma *add-init-cls-code-bI2*:
  assumes
    ‹*length at* ≤ *Suc* (*Suc uint-max*)›
  shows ‹*5* ≤ *uint64-max* − *length at*›
  ⟨*proof*⟩

lemma *add-init-clss-codebI*:
  assumes
    ‹*length at* ≤ *Suc* (*Suc uint-max*)› **and**
    ‹*2* ≤ *length at*› **and**
    ‹*length a1'j* ≤ *length a1'a*› **and**
    ‹*length a1'a* ≤ *uint64-max* − (*length at* + *5*)›
  shows ‹*length a1'j* < *uint64-max*›
  ⟨*proof*⟩

sepref-definition *add-init-cls-code-b*
  is ‹*uncurry add-init-cls-heur-b*›
  :: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
  ⟨*proof*⟩

declare *add-init-cls-code.refine*[*sepref-fr-rules*]
add-init-cls-code-b.refine[*sepref-fr-rules*]

sepref-definition *already-propagated-unit-cls-conflict-code*
is ‹*uncurry already-propagated-unit-cls-conflict-heur*›
:: ‹$\mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{isasat\text{-}init\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}init\text{-}assn}$›
⟨*proof*⟩

declare *already-propagated-unit-cls-conflict-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *set-conflict-empty-code*
is ‹*RETURN o lookup-set-conflict-empty*›
:: ‹$\mathit{conflict\text{-}option\text{-}rel\text{-}assn}^d \rightarrow_a \mathit{conflict\text{-}option\text{-}rel\text{-}assn}$›
⟨*proof*⟩

declare *set-conflict-empty-code.refine*[*sepref-fr-rules*]

sepref-definition *set-empty-clause-as-conflict-code*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹$\mathit{isasat\text{-}init\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}init\text{-}assn}$›
⟨*proof*⟩

sepref-definition *set-empty-clause-as-conflict-code-unb*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹$\mathit{isasat\text{-}init\text{-}unbounded\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}init\text{-}unbounded\text{-}assn}$›
⟨*proof*⟩

declare *set-empty-clause-as-conflict-code.refine*[*sepref-fr-rules*]
set-empty-clause-as-conflict-code-unb.refine[*sepref-fr-rules*]

sepref-definition *add-clause-to-others-code*
is ‹*uncurry add-clause-to-others-heur*›
:: ‹$(\mathit{list\text{-}assn}\ \mathit{unat\text{-}lit\text{-}assn})^k *_a \mathit{isasat\text{-}init\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}init\text{-}assn}$›
⟨*proof*⟩

sepref-definition *add-clause-to-others-code-unb*
is ‹*uncurry add-clause-to-others-heur*›
:: ‹$(\mathit{list\text{-}assn}\ \mathit{unat\text{-}lit\text{-}assn})^k *_a \mathit{isasat\text{-}init\text{-}unbounded\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}init\text{-}unbounded\text{-}assn}$›
⟨*proof*⟩

declare *add-clause-to-others-code.refine*[*sepref-fr-rules*]
add-clause-to-others-code-unb.refine[*sepref-fr-rules*]

lemma (**in** −)*list-length-1-hnr*[*sepref-fr-rules*]:
assumes ‹*CONSTRAINT is-pure R* ›
shows ‹$(\mathit{return}\ o\ \mathit{list\text{-}length\text{-}1\text{-}code}, \mathit{RETURN}\ o\ \mathit{list\text{-}length\text{-}1}) \in (\mathit{list\text{-}assn}\ R)^k \rightarrow_a \mathit{bool\text{-}assn}$›
⟨*proof*⟩

sepref-definition *get-conflict-wl-is-None-init-code*
is ‹*RETURN o get-conflict-wl-is-None-heur-init*›
:: ‹$\mathit{isasat\text{-}init\text{-}assn}^k \rightarrow_a \mathit{bool\text{-}assn}$›
⟨*proof*⟩

sepref-definition *get-conflict-wl-is-None-init-code-unb*
is ‹*RETURN o get-conflict-wl-is-None-heur-init*›

:: ‹*isasat-init-unbounded-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

declare *get-conflict-wl-is-None-init-code.refine*[*sepref-fr-rules*]
get-conflict-wl-is-None-init-code-unb.refine[*sepref-fr-rules*]

sepref-definition *polarity-st-heur-init-code*
is ‹*uncurry* (*RETURN oo polarity-st-heur-init*)›
:: ‹[λ(*S*, *L*). *polarity-pol-pre* (*get-trail-wl-heur-init S*) *L*]$_a$ *isasat-init-assn*^{k} $*_a$ *unat-lit-assn*^{k} → *tri-bool-assn*›
⟨*proof*⟩

sepref-definition *polarity-st-heur-init-code-unb*
is ‹*uncurry* (*RETURN oo polarity-st-heur-init*)›
:: ‹[λ(*S*, *L*). *polarity-pol-pre* (*get-trail-wl-heur-init S*) *L*]$_a$
isasat-init-unbounded-assn^k $*_a$ *unat-lit-assn*^{k} → *tri-bool-assn*›
⟨*proof*⟩

declare *polarity-st-heur-init-code.refine*[*sepref-fr-rules*]
polarity-st-heur-init-code-unb.refine[*sepref-fr-rules*]

lemma *is-Nil-hnr*[*sepref-fr-rules*]:
‹(*return o is-Nil*, *RETURN o is-Nil*) ∈ (*list-assn R*)^k$\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

sepref-register *init-dt-step-wl*
get-conflict-wl-is-None-heur-init already-propagated-unit-cls-heur
conflict-propagated-unit-cls-heur add-clause-to-others-heur
add-init-cls-heur set-empty-clause-as-conflict-heur

sepref-register *polarity-st-heur-init propagate-unit-cls-heur*

sepref-definition *init-dt-step-wl-code-unb*
is ‹*uncurry* (*init-dt-step-wl-heur-unb*)›
:: ‹[λ(*C*, *S*). *True*]$_a$ (*list-assn unat-lit-assn*)^d $*_a$ *isasat-init-unbounded-assn*^{d} →
isasat-init-unbounded-assn›
⟨*proof*⟩

sepref-definition *init-dt-step-wl-code-b*
is ‹*uncurry* (*init-dt-step-wl-heur-b*)›
:: ‹[λ(*C*, *S*). *True*]$_a$ (*list-assn unat-lit-assn*)^d $*_a$ *isasat-init-assn*^{d} →
isasat-init-assn›
⟨*proof*⟩

declare
init-dt-step-wl-code-unb.refine[*sepref-fr-rules*]
init-dt-step-wl-code-b.refine[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-unb*

abbreviation *isasat-atms-ext-rel-assn* **where**
‹*isasat-atms-ext-rel-assn* ≡ *array-assn uint64-nat-assn* ∗*a* *uint32-nat-assn* ∗*a*
arl-assn uint32-nat-assn›

abbreviation *nat-lit-list-hm-assn* **where**
‹*nat-lit-list-hm-assn* ≡ *hr-comp isasat-atms-ext-rel-assn isasat-atms-ext-rel*›

lemma (**in** −) [*sepref-fr-rules*]:
‹(*return o init-next-size*, *RETURN o init-next-size*)
$\in [\lambda L.\ L \leq \textit{uint32-max div 2}]_a\ \textit{uint32-nat-assn}^k \rightarrow \textit{uint32-nat-assn}$›
⟨*proof*⟩

sepref-definition *nat-lit-lits-init-assn-assn-in*
is ‹*uncurry add-to-atms-ext*›
:: ‹$\textit{uint32-nat-assn}^k *_a \textit{isasat-atms-ext-rel-assn}^d \rightarrow_a \textit{isasat-atms-ext-rel-assn}$›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*uncurry nat-lit-lits-init-assn-assn-in*, *uncurry* (*RETURN* ∘∘ *op-set-insert*))
$\in [\lambda(a, b).\ a \leq \textit{uint-max div 2}]_a$
$\textit{uint32-nat-assn}^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow \textit{nat-lit-list-hm-assn}$›
⟨*proof*⟩

sepref-definition *extract-atms-cls-imp*
is ‹*uncurry extract-atms-cls-i*›
:: ‹$(\textit{list-assn unat-lit-assn})^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow_a \textit{nat-lit-list-hm-assn}$›
⟨*proof*⟩

declare *extract-atms-cls-imp.refine*[*sepref-fr-rules*]

sepref-definition *extract-atms-clss-imp*
is ‹*uncurry extract-atms-clss-i*›
:: ‹$(\textit{list-assn}\ (\textit{list-assn unat-lit-assn}))^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow_a \textit{nat-lit-list-hm-assn}$›
⟨*proof*⟩

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
‹(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* ∘∘ *extract-atms-clss*))
$\in [\lambda(a, b).\ \forall C\in \textit{set}\ a.\ \forall L\in \textit{set}\ C.\ \textit{nat-of-lit}\ L \leq \textit{uint-max}]_a$
$(\textit{list-assn}\ (\textit{list-assn unat-lit-assn}))^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow \textit{nat-lit-list-hm-assn}$›
⟨*proof*⟩

sepref-definition *extract-atms-clss-imp-empty-assn*
is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
:: ‹$\textit{unit-assn}^k \rightarrow_a \textit{isasat-atms-ext-rel-assn}$›
⟨*proof*⟩

lemma *extract-atms-clss-imp-empty-assn*[*sepref-fr-rules*]:
‹(*uncurry0 extract-atms-clss-imp-empty-assn*, *uncurry0* (*RETURN op-extract-list-empty*))
$\in \textit{unit-assn}^k \rightarrow_a \textit{nat-lit-list-hm-assn}$›
⟨*proof*⟩

declare *atm-of-hnr*[*sepref-fr-rules*]

lemma *extract-atms-clss-imp-empty-rel-alt-def*:
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*op-array-replicate 1024 zero-uint64-nat*, *0*, []))›
⟨*proof*⟩

Full Initialisation

sepref-definition *rewatch-heur-st-code*
is ‹(*rewatch-heur-st*)›
:: ‹$isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
⟨*proof*⟩
find-theorems *nfoldli WHILET*

sepref-definition *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹$[rewatch\text{-}heur\text{-}st\text{-}fast\text{-}pre]_a$
$isasat\text{-}init\text{-}assn^d \rightarrow isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

declare *rewatch-heur-st-code.refine*[*sepref-fr-rules*]
rewatch-heur-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *rewatch-heur-st init-dt-step-wl-heur*

sepref-definition *init-dt-wl-heur-code-unb*
is ‹*uncurry* (*init-dt-wl-heur-unb*)›
:: ‹$(list\text{-}assn\ (list\text{-}assn\ unat\text{-}lit\text{-}assn))^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a$
$isasat\text{-}init\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *init-dt-wl-heur-code-b*
is ‹*uncurry* (*init-dt-wl-heur-b*)›
:: ‹$(list\text{-}assn\ (list\text{-}assn\ unat\text{-}lit\text{-}assn))^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a$
$isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

declare
init-dt-wl-heur-code-unb.refine[*sepref-fr-rules*]
init-dt-wl-heur-code-b.refine[*sepref-fr-rules*]

sepref-definition *init-dt-wl-heur-full-code*
is ‹*uncurry* (*init-dt-wl-heur-full-unb*)›
:: ‹$(list\text{-}assn\ (list\text{-}assn\ unat\text{-}lit\text{-}assn))^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a$
$isasat\text{-}init\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *init-dt-wl-heur-full-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *extract-lits-sorted-code*
is ‹*extract-lits-sorted*›
:: ‹$[\lambda(xs, n, vars).\ (\forall x \in \# mset\ vars.\ x < length\ xs)]_a$
$isasat\text{-}atms\text{-}ext\text{-}rel\text{-}assn^d \rightarrow$
$arl\text{-}assn\ uint32\text{-}nat\text{-}assn * a\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *extract-lits-sorted-code.refine*[*sepref-fr-rules*]

abbreviation *lits-with-max-assn* **where**

‹*lits-with-max-assn* ≡ *hr-comp* (*arl-assn uint32-nat-assn* ∗*a uint32-nat-assn*) *lits-with-max-rel*›

lemma *extract-lits-sorted-hnr*[*sepref-fr-rules*]:
‹(*extract-lits-sorted-code*, *RETURN* ∘ *mset-set*) ∈ $nat\text{-}lit\text{-}list\text{-}hm\text{-}assn^d \rightarrow_a$ *lits-with-max-assn*›
(**is** ‹*?c* ∈ $[?pre]_a$ *?im* → *?f*›)
⟨*proof*⟩

term *op-arl32-replicate*
find-theorems *op-arl-replicate arl-assn*

definition *arl32-replicate* **where**
arl32-replicate init-cap x ≡ *do* {
let n = *max* (*nat-of-uint32 init-cap*) *minimum-capacity*;
a ← *Array.new n x*;
return (*a*, *init-cap*)
}

definition [*simp*]: ‹*op-arl32-replicate* = *op-list-replicate*›
lemma *arl32-fold-custom-replicate*:
‹*replicate* = *op-arl32-replicate*›
⟨*proof*⟩

lemma *list-replicate-arl32-hnr*[*sepref-fr-rules*]:
assumes *p*: ‹*CONSTRAINT is-pure R*›
shows ‹(*uncurry arl32-replicate*, *uncurry* (*RETURN oo op-arl32-replicate*)) ∈ $uint32\text{-}nat\text{-}assn^k *_a R^k \rightarrow_a$ *arl32-assn R*›
⟨*proof*⟩

definition *INITIAL-OUTL-SIZE* :: ‹*nat*› **where**
[*simp*]: ‹*INITIAL-OUTL-SIZE* = *160*›
lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 160*), *uncurry0* (*RETURN INITIAL-OUTL-SIZE*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *finalise-init-code′*
is ‹*uncurry finalise-init-code*›
:: ‹[λ(-, *S*). *length* (*get-clauses-wl-heur-init S*) ≤ *uint64-max*]$_a$
$opts\text{-}assn^d *_a isasat\text{-}init\text{-}assn^d \rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-definition *finalise-init-code-unb*
is ‹*uncurry finalise-init-code*›
:: ‹$opts\text{-}assn^d *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *finalise-init-code′.refine*[*sepref-fr-rules*]
finalise-init-code-unb.refine[*sepref-fr-rules*]

lemma (**in** −)*arrayO-raa-empty-sz-empty-list*[*sepref-fr-rules*]:
‹(*arrayO-raa-empty-sz*, *RETURN o init-aa*) ∈
$nat\text{-}assn^k \rightarrow_a$ (*arlO-assn clause-ll-assn*)›
⟨*proof*⟩

lemma *init-aa′-alt-def*: ‹*RETURN o init-aa′* = (λ*n*. *RETURN op-arl-empty*)›
⟨*proof*⟩

sepref-definition *init-aa′-code*
is ‹*RETURN o init-aa′*›
:: ‹$nat\text{-}assn^k \rightarrow_a$ *arl-assn* (*clause-status-assn* ∗*a* *uint32-nat-assn* ∗*a* *uint32-nat-assn*)›
⟨*proof*⟩

declare *init-aa′-code.refine*[*sepref-fr-rules*]

sepref-register *initialise-VMTF*

sepref-definition *init-trail-D-code*
is ‹*uncurry2 init-trail-D*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a$ *trail-pol-assn*›
⟨*proof*⟩

declare *init-trail-D-code.refine*[*sepref-fr-rules*]

sepref-definition *init-trail-D-fast-code*
is ‹*uncurry2 init-trail-D-fast*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a$ *trail-pol-fast-assn*›
⟨*proof*⟩

declare *init-trail-D-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *init-state-wl-D′-code*
is ‹*init-state-wl-D′*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn *a\ uint32\text{-}assn)^d \rightarrow_a$ *isasat-init-assn*›
⟨*proof*⟩

sepref-definition *init-state-wl-D′-code-unb*
is ‹*init-state-wl-D′*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn *a\ uint32\text{-}assn)^d \rightarrow_a$ *trail-pol-assn* ∗*a* *arena-assn* ∗*a*
conflict-option-rel-assn ∗*a*
uint32-nat-assn ∗*a*
watchlist-assn ∗*a*
vmtf-remove-conc-option-fst-As ∗*a*
phase-saver-conc ∗*a* *uint32-nat-assn* ∗*a*
cach-refinement-l-assn ∗*a* *lbd-assn* ∗*a* *vdom-assn* ∗*a* *bool-assn*›
⟨*proof*⟩

declare *init-state-wl-D′-code.refine*[*sepref-fr-rules*]
init-state-wl-D′-code-unb.refine[*sepref-fr-rules*]

lemma *to-init-state-code-hnr*:
‹(*return o to-init-state-code*, *RETURN o id*) ∈ $isasat\text{-}init\text{-}assn^d \rightarrow_a$ *isasat-init-assn*›
⟨*proof*⟩

abbreviation (**in** −)*lits-with-max-assn-clss* **where**
‹*lits-with-max-assn-clss* ≡ *hr-comp lits-with-max-assn* (⟨*nat-rel*⟩*mset-rel*)›

end
theory *IsaSAT-Conflict-Analysis*

imports *IsaSAT-Setup IsaSAT-VMTF*
begin

Skip and resolve **lemma** *get-maximum-level-remove-count-max-lvls*:
assumes *L*: ‹$L = -$*lit-of* (*hd M*)› **and** *LD*: ‹$L \in\# D$› **and** *M-nempty*: ‹$M \neq []$›
shows ‹*get-maximum-level-remove M D L* = *count-decided M* $\longleftrightarrow$
(*count-decided M* = *0* $\vee$ *card-max-lvl M D* > *1*)›
(**is** ‹*?max* $\longleftrightarrow$ *?count*›)
⟨*proof*⟩

definition *maximum-level-removed-eq-count-dec* **where**
‹*maximum-level-removed-eq-count-dec L S* $\longleftrightarrow$
get-maximum-level-remove (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L* =
count-decided (*get-trail-wl S*)›

definition *maximum-level-removed-eq-count-dec-heur* **where**
‹*maximum-level-removed-eq-count-dec-heur L S* $\longleftrightarrow$
get-count-max-lvls-heur S > *one-uint32-nat*›

definition *maximum-level-removed-eq-count-dec-pre* **where**
‹*maximum-level-removed-eq-count-dec-pre* =
(λ(*L*, *S*). $L = -$*lit-of* (*hd* (*get-trail-wl S*)) $\wedge$ $L \in\#$ *the* (*get-conflict-wl S*) $\wedge$
get-conflict-wl S $\neq$ *None* $\wedge$ *get-trail-wl S* $\neq$ [] $\wedge$ *count-decided* (*get-trail-wl S*) $\geq$ *1*)›

lemma *maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec*:
‹(*uncurry* (*RETURN oo maximum-level-removed-eq-count-dec-heur*),
uncurry (*RETURN oo maximum-level-removed-eq-count-dec*)) $\in$
[*maximum-level-removed-eq-count-dec-pre*]$_f$
Id $\times_r$ *twl-st-heur-conflict-ana* $\to$ ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-trail-wl-heur-def*: ‹*get-trail-wl-heur* = (λ(*M*, *S*). *M*)›
⟨*proof*⟩

definition *lit-and-ann-of-propagated-st* :: ‹*nat twl-st-wl* $\Rightarrow$ *nat literal* $\times$ *nat*› **where**
‹*lit-and-ann-of-propagated-st S* = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*))›

definition *lit-and-ann-of-propagated-st-heur*
:: ‹*twl-st-wl-heur* $\Rightarrow$ *nat literal* $\times$ *nat*›
where
‹*lit-and-ann-of-propagated-st-heur* = (λ((*M*, -, -, *reasons*, -), -). (*last M*, *reasons* ! (*atm-of* (*last M*))))›

lemma *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*:
‹(*RETURN o lit-and-ann-of-propagated-st-heur*, *RETURN o lit-and-ann-of-propagated-st*) $\in$
[λ*S*. *is-proped* (*hd* (*get-trail-wl S*)) $\wedge$ *get-trail-wl S* $\neq$ []]$_f$ *twl-st-heur-conflict-ana* $\to$ ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *twl-st-heur-conflict-ana-lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*:
‹(*x*, *y*) $\in$ *twl-st-heur-conflict-ana* $\Longrightarrow$ *is-proped* (*hd* (*get-trail-wl y*)) $\Longrightarrow$ *get-trail-wl y* $\neq$ [] $\Longrightarrow$
lit-and-ann-of-propagated-st-heur x = *lit-and-ann-of-propagated-st y*›
⟨*proof*⟩

definition *tl-state-wl-heur-pre* :: ‹*twl-st-wl-heur* $\Rightarrow$ *bool*› **where**
‹*tl-state-wl-heur-pre* =
(λ(*M*, *N*, *D*, *WS*, *Q*, ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, -). *fst M* $\neq$ [] $\wedge$

tl-trailt-tr-pre M ∧
vmtf-unset-pre (*atm-of* (*last* (*fst M*))) ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∧
atm-of (*last* (*fst M*)) < *length* φ ∧
atm-of (*last* (*fst M*)) < *length A* ∧
(*next-search* ≠ *None* ⟶ *the next-search* < *length A*))›

definition *tl-state-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*).
(*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of* (*lit-of-last-trail-pol M*)) *vmtf*, φ, *clvls*))›

lemma *tl-state-wl-heur-alt-def*:
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*).
(*let L* = *lit-of-last-trail-pol M in*
(*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of L*) *vmtf*, φ, *clvls*)))›
⟨*proof*⟩

lemma *card-max-lvl-Cons*:
assumes ‹*no-dup* (*L* # *a*)› ‹*distinct-mset y*›‹¬*tautology y*› ‹¬*is-decided L*›
shows ‹*card-max-lvl* (*L* # *a*) *y* =
(*if* (*lit-of L* ∈# *y* ∨ −*lit-of L* ∈# *y*) ∧ *count-decided a* ≠ *0 then card-max-lvl a y* + *1*
else card-max-lvl a y)›
⟨*proof*⟩

lemma *card-max-lvl-tl*:
assumes ‹*a* ≠ []› ‹*distinct-mset y*›‹¬*tautology y*› ‹¬*is-decided* (*hd a*)› ‹*no-dup a*›
‹*count-decided a* ≠ *0*›
shows ‹*card-max-lvl* (*tl a*) *y* =
(*if* (*lit-of*(*hd a*) ∈# *y* ∨ −*lit-of*(*hd a*) ∈# *y*)
then card-max-lvl a y − *1 else card-max-lvl a y*)›
⟨*proof*⟩

definition *tl-state-wl-pre* **where**
‹*tl-state-wl-pre S* ⟷ *get-trail-wl S* ≠ [] ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
−(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
¬*is-decided* (*hd* (*get-trail-wl S*)) ∧
count-decided (*get-trail-wl S*) > *0*›

lemma *tl-state-out-learned*:
‹*lit-of* (*hd a*) ∉# *the at* ⟹
− *lit-of* (*hd a*) ∉# *the at* ⟹
¬ *is-decided* (*hd a*) ⟹
out-learned (*tl a*) *at an* ⟷ *out-learned a at an*›
⟨*proof*⟩

lemma *tl-state-wl-heur-tl-state-wl*:
‹(*RETURN o tl-state-wl-heur*, *RETURN o tl-state-wl*) ∈
[*tl-state-wl-pre*]$_f$ *twl-st-heur-conflict-ana* → ⟨*twl-st-heur-conflict-ana*⟩*nres-rel*›
⟨*proof*⟩

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹

arena-act-pre (*mark-used arena C*) *C*›
⟨*proof*⟩

definition (**in** −) *get-max-lvl-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*get-max-lvl-st S L* = *get-maximum-level-remove* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L*›

definition *update-confl-tl-wl-heur*
:: ‹*nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*update-confl-tl-wl-heur* = (λ*C L* (*M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*). *do* {
    ASSERT (*clvls* ≥ *1*);
    let L′ = *atm-of L*;
    ASSERT(*arena-length N C* ≠ *2* ⟶
      curry6 isa-set-lookup-conflict-aa-pre M N C (*b*, (*n*, *xs*)) *clvls lbd outl*);
    ASSERT(*arena-is-valid-clause-idx N C*);
    ((*b*, (*n*, *xs*)), *clvls*, *lbd*, *outl*) ←
      if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C* (*b*, (*n*, *xs*)) *clvls lbd outl*
      else isa-resolve-merge-conflict-gt2 M N C (*b*, (*n*, *xs*)) *clvls lbd outl*;
    ASSERT(*curry lookup-conflict-remove1-pre L* (*n*, *xs*) ∧ *clvls* ≥ *1*);
    let (*n*, *xs*) = *lookup-conflict-remove1 L* (*n*, *xs*);
    ASSERT(*arena-act-pre N C*);
    let N = *mark-used N C*;
    ASSERT(*arena-act-pre N C*);
    let N = *arena-incr-act N C*;
    ASSERT(*vmtf-unset-pre L′ vm*);
    ASSERT(*tl-trailt-tr-pre M*);
    RETURN (*False*, (*tl-trailt-tr M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *isa-vmtf-unset L′ vm*,
        φ, *fast-minus clvls one-uint32-nat*, *cach*, *lbd*, *outl*, *stats*))
  })›

lemma *card-max-lvl-remove1-mset-hd*:
‹−*lit-of* (*hd M*) ∈# *y* ⟹ *is-proped* (*hd M*) ⟹
   card-max-lvl M (*remove1-mset* (−*lit-of* (*hd M*)) *y*) = *card-max-lvl M y* − *1*›
⟨*proof*⟩

lemma *update-confl-tl-wl-heur-state-helper*:
‹(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ⟹ *get-trail-wl S* ≠ [] ⟹
 is-proped (*hd* (*get-trail-wl S*)) ⟹ *L* = *lit-of* (*hd* (*get-trail-wl S*))›
⟨*proof*⟩

lemma (**in** −) *not-ge-Suc0*: ‹¬*Suc 0* ≤ *n* ⟷ *n* = *0*›
⟨*proof*⟩

definition *update-confl-tl-wl-pre* **where**
‹*update-confl-tl-wl-pre* = (λ((*C*, *L*), *S*).
    C ∈# *dom-m* (*get-clauses-wl S*) ∧
    get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧
    − *L* ∈# *the* (*get-conflict-wl S*) ∧
    (*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ∧
    L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
    is-proped (*hd* (*get-trail-wl S*)) ∧
    C > *0* ∧
    card-max-lvl (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1* ∧
    distinct-mset (*the* (*get-conflict-wl S*)) ∧
    − *L* ∉ *set* (*get-clauses-wl S* ∝ *C*) ∧

$(length\ (get\text{-}clauses\text{-}wl\ S \propto C) > 2 \longrightarrow$
$L \notin set\ (tl\ (get\text{-}clauses\text{-}wl\ S \propto C)) \wedge$
$get\text{-}clauses\text{-}wl\ S \propto C\ !\ 0 = L) \wedge$
$L \in set\ (watched\text{-}l\ (get\text{-}clauses\text{-}wl\ S \propto C)) \wedge$
$distinct\ (get\text{-}clauses\text{-}wl\ S \propto C) \wedge$
$\neg tautology\ (the\ (get\text{-}conflict\text{-}wl\ S)) \wedge$
$\neg tautology\ (mset\ (get\text{-}clauses\text{-}wl\ S \propto C)) \wedge$
$\neg tautology\ (remove1\text{-}mset\ L\ (remove1\text{-}mset\ (-\ L)$
$((the\ (get\text{-}conflict\text{-}wl\ S) \cup\#\ mset\ (get\text{-}clauses\text{-}wl\ S \propto C))))) \wedge$
$count\text{-}decided\ (get\text{-}trail\text{-}wl\ S) > 0 \wedge$
$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ (all\text{-}atms\text{-}st\ S)\ (the\ (get\text{-}conflict\text{-}wl\ S)) \wedge$
$literals\text{-}are\text{-}\mathcal{L}_{in}\ (all\text{-}atms\text{-}st\ S)\ S \wedge$
$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ (all\text{-}atms\text{-}st\ S)\ (get\text{-}trail\text{-}wl\ S)$
)›

lemma (**in** −)*out-learned-add-mset-highest-level*:
‹$L = lit\text{-}of\ (hd\ M) \Longrightarrow out\text{-}learned\ M\ (Some\ (add\text{-}mset\ (-\ L)\ A))\ outl \longleftrightarrow$
$out\text{-}learned\ M\ (Some\ A)\ outl$›
⟨*proof*⟩

lemma (**in** −)*out-learned-tl-Some-notin*:
‹$is\text{-}proped\ (hd\ M) \Longrightarrow lit\text{-}of\ (hd\ M) \notin\#\ C \Longrightarrow -lit\text{-}of\ (hd\ M) \notin\#\ C \Longrightarrow$
$out\text{-}learned\ M\ (Some\ C)\ outl \longleftrightarrow out\text{-}learned\ (tl\ M)\ (Some\ C)\ outl$›
⟨*proof*⟩

abbreviation *twl-st-heur-conflict-ana′* :: ‹$nat \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$› **where**
‹$twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana'\ r \equiv \{(S,\ T).\ (S,\ T) \in twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana \wedge$
$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) = r\}$›

lemma *literals-are-in-$\mathcal{L}_{in}$-mm-all-atms-self*[*simp*]:
‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}mm\ (all\text{-}atms\ ca\ NUE)\ \{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ ca\#\}$›
⟨*proof*⟩

lemma *update-confl-tl-wl-heur-update-confl-tl-wl*:
‹$(uncurry2\ (update\text{-}confl\text{-}tl\text{-}wl\text{-}heur),\ uncurry2\ (RETURN\ ooo\ update\text{-}confl\text{-}tl\text{-}wl)) \in$
$[update\text{-}confl\text{-}tl\text{-}wl\text{-}pre]_f$
$nat\text{-}rel \times_f Id \times_f twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana'\ r \rightarrow \langle bool\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana'\ r\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *phase-saving-le*: ‹$phase\text{-}saving\ \mathcal{A}\ \varphi \Longrightarrow A \in\#\ \mathcal{A} \Longrightarrow A < length\ \varphi$›
‹$phase\text{-}saving\ \mathcal{A}\ \varphi \Longrightarrow B \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow atm\text{-}of\ B < length\ \varphi$›
⟨*proof*⟩

lemma *isa-vmtf-le*:
‹$((a,\ b),\ M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow A \in\#\ \mathcal{A} \Longrightarrow A < length\ a$›
‹$((a,\ b),\ M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow B \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow atm\text{-}of\ B < length\ a$›
⟨*proof*⟩

lemma *isa-vmtf-next-search-le*:
‹$((a,\ b,\ c,\ c',\ Some\ d),\ M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow d < length\ a$›
⟨*proof*⟩

lemma *trail-pol-nempty*: ‹$\neg(([],\ aa,\ ab,\ ac,\ ad,\ b),\ L\ \#\ ys) \in trail\text{-}pol\ \mathcal{A}$›
⟨*proof*⟩

definition *is-decided-hd-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*is-decided-hd-trail-wl-heur* = (λS. *is-None* (*snd* (*last-trail-pol* (*get-trail-wl-heur* S))))›

lemma *is-decided-hd-trail-wl-heur-hd-get-trail*:
‹(*RETURN o is-decided-hd-trail-wl-heur*, *RETURN o* (λM. *is-decided* (*hd* (*get-trail-wl* M))))
∈ [λM. *get-trail-wl* $M \neq []$]$_f$ *twl-st-heur-conflict-ana′* r → ⟨*bool-rel*⟩ *nres-rel*›
⟨*proof*⟩

definition *is-decided-hd-trail-wl-heur-pre* **where**
‹*is-decided-hd-trail-wl-heur-pre* =
(λS. *fst* (*get-trail-wl-heur* S) ≠ [] ∧ *last-trail-pol-pre* (*get-trail-wl-heur* S))›

definition *skip-and-resolve-loop-wl-D-heur-inv* **where**
‹*skip-and-resolve-loop-wl-D-heur-inv* $S_0{}'$ =
(λ(*brk*, S'). ∃ S S_0. (S', S) ∈ *twl-st-heur-conflict-ana* ∧ ($S_0{}'$, S_0) ∈ *twl-st-heur-conflict-ana* ∧
skip-and-resolve-loop-wl-D-inv S_0 *brk* S ∧
length (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* $S_0{}'$) ∧
is-decided-hd-trail-wl-heur-pre S')›

definition *update-confl-tl-wl-heur-pre*
:: ‹(*nat* × *nat literal*) × *twl-st-wl-heur* ⇒ *bool*›
where
‹*update-confl-tl-wl-heur-pre* =
(λ((i, L), (M, N, D, W, Q, ((A, m, *fst-As*, *lst-As*, *next-search*), -), φ, *clvls*, *cach*, *lbd*,
outl, -)).
$i > 0$ ∧
(*fst* M) ≠ [] ∧
atm-of ((*last* (*fst* M))) < *length* φ ∧
atm-of ((*last* (*fst* M))) < *length* A ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length* A) ∧
L = (*last* (*fst* M))
)›

definition *lit-and-ann-of-propagated-st-heur-pre* **where**
‹*lit-and-ann-of-propagated-st-heur-pre* = (λ((M, -, -, *reasons*, -), -). *atm-of* (*last* M) < *length reasons*
∧ M ≠ [])›

definition *atm-is-in-conflict-st-heur-pre*
:: ‹*nat literal* × *twl-st-wl-heur* ⇒ *bool*›
where
‹*atm-is-in-conflict-st-heur-pre* = (λ(L, (M,N,(-, (-, D)), -)). *atm-of* L < *length* D)›

definition *skip-and-resolve-loop-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*skip-and-resolve-loop-wl-D-heur* S_0 =
do {
(-, S) ←
$WHILE_T{}^{skip\text{-}and\text{-}resolve\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
(λ(*brk*, S). ¬*brk* ∧ ¬*is-decided-hd-trail-wl-heur* S)
(λ(*brk*, S).
do {
ASSERT(¬*brk* ∧ ¬*is-decided-hd-trail-wl-heur* S);
ASSERT(*lit-and-ann-of-propagated-st-heur-pre* S);
let (L, C) = *lit-and-ann-of-propagated-st-heur* S;
ASSERT(*atm-is-in-conflict-st-heur-pre* ($-L$, S));

if $\neg$*atm-is-in-conflict-st-heur* $(-L)$ *S then*
do {
ASSERT (*tl-state-wl-heur-pre S*);
RETURN (*False*, *tl-state-wl-heur S*)}
else
if maximum-level-removed-eq-count-dec-heur $(-L)$ *S*
then do {
ASSERT(*update-confl-tl-wl-heur-pre* ((*C*, *L*), *S*));
update-confl-tl-wl-heur C L S}
else
RETURN (*True*, *S*)
}
)
(*False*, S_0);
RETURN S
}
›

context
fixes *x y xa x′ x1 x2 x1b x2b r*
assumes
xy: ‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana′ r*› **and**
confl: ‹*get-conflict-wl y* ≠ *None*› **and**
xa-x′: ‹(*xa*, *x′*) ∈ *bool-rel* $\times_f$ *twl-st-heur-conflict-ana′* (*length* (*get-clauses-wl-heur x*))› **and**
x′: ‹*x′* = (*x1*, *x2*)› **and**
xa: ‹*xa* = (*x1b*, *x2b*)› **and**
sor-inv: ‹*case x′ of* (*x*, *xa*) ⇒ *skip-and-resolve-loop-wl-D-inv y x xa*›
begin

private lemma *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st x2*) *x2*› **and**
confl-x2: ‹*get-conflict-wl x2* ≠ *None*› **and**
trail-nempty: ‹*get-trail-wl x2* ≠ []› **and**
not-tauto: ‹¬*tautology* (*the* (*get-conflict-wl x2*))› **and**
dist-confl: ‹*distinct-mset* (*the* (*get-conflict-wl x2*))› **and**
count-dec-not0: ‹*count-decided* (*get-trail-wl x2*) ≠ *0*› **and**
no-dup-x2: ‹*no-dup* (*get-trail-wl x2*)› **and**
lits-trail: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st x2*) (*get-trail-wl x2*)› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st x2*) (*the* (*get-conflict-wl x2*))›
⟨*proof*⟩ **lemma** *sor-heur-inv-heur1*:
‹*fst* (*get-trail-wl-heur x2b*) ≠ []›
⟨*proof*⟩ **lemma** *sor-heur-inv-heur2*:
‹*last-trail-pol-pre* (*get-trail-wl-heur x2b*)›
⟨*proof*⟩

lemma *sor-heur-inv*:
‹*skip-and-resolve-loop-wl-D-heur-inv x xa*›
⟨*proof*⟩

lemma *conflict-ana-same-cond*:
‹(¬ *x1b* ∧ ¬ *is-decided-hd-trail-wl-heur x2b*) =
(¬ *x1* ∧ ¬ *is-decided* (*hd* (*get-trail-wl x2*)))›
⟨*proof*⟩

context
fixes *x1a x2a x1c x2c*

assumes
hd-xa: ‹*lit-and-ann-of-propagated* (*hd* (*get-trail-wl x2*)) = (*x1a*, *x2a*)› **and**
cond-heur: ‹*case xa of* (*brk*, *S*) ⇒ ¬ *brk* ∧ ¬ *is-decided-hd-trail-wl-heur S*› **and**
cond: ‹*case x′ of* (*brk*, *S*) ⇒ ¬ *brk* ∧ ¬ *is-decided* (*hd* (*get-trail-wl S*))› **and**
xc: ‹*lit-and-ann-of-propagated-st-heur x2b* = (*x1c*, *x2c*)› **and**
assert: ‹¬ *x1* ∧ ¬ *is-decided* (*hd* (*get-trail-wl x2*))› **and**
assert′: ‹¬ *x1b* ∧ ¬ *is-decided-hd-trail-wl-heur x2b*›
begin

lemma *st*[*simp*]: ‹*x1* = *False*› ‹*x1b* = *False*› **and**
x2b-x2: ‹(*x2b*, *x2*) ∈ *twl-st-heur-conflict-ana′* (*length* (*get-clauses-wl-heur x*))›
⟨*proof*⟩ **lemma**
x1c: ‹*x1c* ∈# $\mathcal{L}_{all}$ (*all-atms-st x2*)› **and**
x1c-notin: ‹*x1c* ∉# *the* (*get-conflict-wl x2*)› **and**
not-dec-ge0: ‹*0* < *mark-of* (*hd* (*get-trail-wl x2*))› **and**
x2c-dom: ‹*x2c* ∈# *dom-m* (*get-clauses-wl x2*)› **and**
hd-x2: ‹*hd* (*get-trail-wl x2*) = *Propagated x1c x2c*› **and**
‹*length* (*get-clauses-wl x2* ∝ *x2c*) > *2* ⟶ *hd* (*get-clauses-wl x2* ∝ *x2c*) = *x1c*› **and**
‹*get-clauses-wl x2* ∝ *x2c* ≠ []› **and**
ux1c-notin-tl: ‹− *x1c* ∉ *set* (*get-clauses-wl x2* ∝ *x2c*)› **and**
x1c-notin-tl: ‹*length* (*get-clauses-wl x2* ∝ *x2c*) > *2* ⟶ *x1c* ∉ *set* (*tl* (*get-clauses-wl x2* ∝ *x2c*))› **and**
not-tauto-x2c: ‹¬*tautology* (*mset* (*get-clauses-wl x2* ∝ *x2c*))› **and**
dist-x2c: ‹*distinct* (*get-clauses-wl x2* ∝ *x2c*)› **and**
not-tauto-resolved: ‹¬*tautology* (*remove1-mset x1c* (*remove1-mset* (− *x1c*) (*the* (*get-conflict-wl x2*)
∪# *mset* (*get-clauses-wl x2* ∝ *x2c*))))› **and**
st2[*simp*]: ‹*x1a* = *x1c*›‹*x2a* = *x2c*› **and**
x1c-NC-0: ‹*2* < *length* (*get-clauses-wl x2* ∝ *x2c*) ⟶ *get-clauses-wl x2* ∝ *x2c* ! *0* = *x1c*› **and**
x1c-watched: ‹*x1c* ∈ *set* (*watched-l* (*get-clauses-wl x2* ∝ *x2c*))›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-ana-is-in-conflict-st*:
‹(*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*), *uncurry* (*RETURN oo is-in-conflict-st*)) ∈
[λ(*L*, *S*). −*L* ∉# *the* (*get-conflict-wl S*) ∧ *get-conflict-wl S* ≠ *None* ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)]$_f$
Id $\times_r$ *twl-st-heur-conflict-ana′* (*length* (*get-clauses-wl-heur x*)) → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-iff*: ‹(¬ *atm-is-in-conflict-st-heur* (− *x1c*) *x2b*) =
(− *x1a* ∉# *the* (*get-conflict-wl x2*))›
⟨*proof*⟩

lemma *ca-lit-and-ann-of-propagated-st-heur-pre*:
‹*lit-and-ann-of-propagated-st-heur-pre x2b*›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-pre*: ‹*atm-is-in-conflict-st-heur-pre* (− *x1c*, *x2b*)›
⟨*proof*⟩

context
assumes *x1a-notin*: ‹− *x1a* ∉# *the* (*get-conflict-wl x2*)›
begin

lemma *tl-state-wl-heur-pre*: ‹*tl-state-wl-heur-pre x2b*›

⟨*proof*⟩ **lemma** *tl-state-wl-pre*: ‹*tl-state-wl-pre x2*›
⟨*proof*⟩ **lemma** *length-tl*: ‹*length* (*get-clauses-wl-heur* (*tl-state-wl-heur x2b*)) = *length* (*get-clauses-wl-heur x2b*)›
⟨*proof*⟩

lemma *tl-state-wl-heur-rel*:
‹((*False*, *tl-state-wl-heur x2b*), *False*, *tl-state-wl x2*) ∈ *bool-rel* $\times_f$ *twl-st-heur-conflict-ana'* (*length* (*get-clauses-wl-heur x*))›
⟨*proof*⟩

end

context
assumes *x1a-notin*: ‹¬ − *x1a* ∉# *the* (*get-conflict-wl x2*)›
begin
lemma *maximum-level-removed-eq-count-dec-pre*:
‹*maximum-level-removed-eq-count-dec-pre* (− *x1a*, *x2*)›
⟨*proof*⟩

lemma *skip-rel*:
‹((− *x1c*, *x2b*), − *x1a*, *x2*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana*›
⟨*proof*⟩

context
assumes ‹*maximum-level-removed-eq-count-dec-heur* (− *x1c*) *x2b*› **and**
max-lvl: ‹*maximum-level-removed-eq-count-dec* (− *x1a*) *x2*›
begin

lemma *update-confl-tl-wl-heur-pre*:
‹*update-confl-tl-wl-heur-pre* ((*x2c*, *x1c*), *x2b*)›
⟨*proof*⟩ **lemma** *counts-maximum-level*:
‹*get-count-max-lvls-heur x2b* ∈ *counts-maximum-level* (*get-trail-wl x2*) (*get-conflict-wl x2*)›
⟨*proof*⟩ **lemma** *card-max-lvl-ge0*:
‹*Suc 0* ≤ *card-max-lvl* (*get-trail-wl x2*) (*the* (*get-conflict-wl x2*))›
⟨*proof*⟩

lemma *update-confl-tl-wl-pre*:
‹*update-confl-tl-wl-pre* ((*x2a*, *x1a*), *x2*)›
⟨*proof*⟩

lemma *update-confl-tl-rel*: ‹(((*x2c*, *x1c*), *x2b*), (*x2a*, *x1a*), *x2*) ∈ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana'* (*length* (*get-clauses-wl-heur x*))›
⟨*proof*⟩

end
end

declare *st*[*simp del*] *st2*[*simp del*]
end

end

lemma *skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D*:
‹(*skip-and-resolve-loop-wl-D-heur*, *skip-and-resolve-loop-wl-D*) ∈ *twl-st-heur-conflict-ana'* *r* $\rightarrow_f$ ⟨*twl-st-heur-conflict-ana'* *r*⟩*nres-rel*›

⟨*proof*⟩

definition (**in** −) *get-count-max-lvls-code* **where**
‹*get-count-max-lvls-code* = (λ(-, -, -, -, -, -, -, *clvls*, -). *clvls*)›

lemma *is-decided-hd-trail-wl-heur-alt-def*:
‹*is-decided-hd-trail-wl-heur* = (λ(*M*, -). *is-None* (*snd* (*last-trail-pol M*)))›
⟨*proof*⟩

lemma *atm-of-in-atms-of*: ‹*atm-of x* ∈ *atms-of C* ⟷ *x* ∈# *C* ∨ −*x* ∈# *C*›
⟨*proof*⟩

definition *atm-is-in-conflict* **where**
‹*atm-is-in-conflict L D* ⟷ *atm-of L* ∈ *atms-of* (*the D*)›

fun *is-in-option-lookup-conflict* **where**
is-in-option-lookup-conflict-def[*simp del*]:
‹*is-in-option-lookup-conflict L* (*a*, *n*, *xs*) ⟷ *is-in-lookup-conflict* (*n*, *xs*) *L*›

lemma *is-in-option-lookup-conflict-atm-is-in-conflict-iff*:
assumes
‹*ba* ≠ *None*› **and** *aa*: ‹*aa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *uaa*: ‹− *aa* ∉# *the ba*› **and**
‹((*b*, *c*, *d*), *ba*) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
shows ‹*is-in-option-lookup-conflict aa* (*b*, *c*, *d*) =
atm-is-in-conflict aa ba›
⟨*proof*⟩

lemma *is-in-option-lookup-conflict-atm-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-option-lookup-conflict*), *uncurry* (*RETURN oo atm-is-in-conflict*))
∈ [λ(*L*, *D*). *D* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ −*L* ∉# *the D*]$_f$
Id $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *is-in-option-lookup-conflict-alt-def*:
‹*RETURN oo is-in-option-lookup-conflict* =
RETURN oo (λ*L* (-, *n*, *xs*). *is-in-lookup-conflict* (*n*, *xs*) *L*)›
⟨*proof*⟩

lemma *skip-and-resolve-loop-wl-DI*:
assumes
‹*skip-and-resolve-loop-wl-D-heur-inv S* (*b*, *T*)›
shows ‹*is-decided-hd-trail-wl-heur-pre T*›
⟨*proof*⟩

lemma *isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv*:
‹*isasat-fast x* ⟹
skip-and-resolve-loop-wl-D-heur-inv x
(*False*, *a2'*) ⟹ *isasat-fast a2'*›
⟨*proof*⟩

end
theory *IsaSAT-Conflict-Analysis-SML*
imports *IsaSAT-Conflict-Analysis IsaSAT-VMTF-SML IsaSAT-Setup-SML*

begin

lemma *mark-of-refine*[*sepref-fr-rules*]:
‹(*return o* (λ*C*. *the* (*snd C*)), *RETURN o mark-of*) ∈
[λ*C*. *is-proped C*]$_a$ *pair-nat-ann-lit-assn*^{k} → *nat-assn*›
⟨*proof*⟩

lemma *mark-of-fast-refine*[*sepref-fr-rules*]:
‹(*return o* (λ*C*. *the* (*snd C*)), *RETURN o mark-of*) ∈
[λ*C*. *is-proped C*]$_a$ *pair-nat-ann-lit-fast-assn*^{k} → *uint64-nat-assn*›
⟨*proof*⟩

lemma *get-count-max-lvls-heur-hnr*[*sepref-fr-rules*]:
‹(*return o get-count-max-lvls-code*, *RETURN o get-count-max-lvls-heur*) ∈
isasat-unbounded-assn^k →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *get-count-max-lvls-heur-fast-hnr*[*sepref-fr-rules*]:
‹(*return o get-count-max-lvls-code*, *RETURN o get-count-max-lvls-heur*) ∈
isasat-bounded-assn^k →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *maximum-level-removed-eq-count-dec-code*
is ‹*uncurry* (*RETURN oo maximum-level-removed-eq-count-dec-heur*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{k} →$_a$ *bool-assn*›
⟨*proof*⟩

sepref-definition *maximum-level-removed-eq-count-dec-fast-code*
is ‹*uncurry* (*RETURN oo maximum-level-removed-eq-count-dec-heur*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} →$_a$ *bool-assn*›
⟨*proof*⟩

declare *maximum-level-removed-eq-count-dec-code.refine*[*sepref-fr-rules*]
maximum-level-removed-eq-count-dec-fast-code.refine[*sepref-fr-rules*]

sepref-definition *is-decided-hd-trail-wl-code*
is ‹*RETURN o is-decided-hd-trail-wl-heur*›
:: ‹[*is-decided-hd-trail-wl-heur-pre*]$_a$
isasat-unbounded-assn^k → *bool-assn*›
⟨*proof*⟩

sepref-definition *is-decided-hd-trail-wl-fast-code*
is ‹*RETURN o is-decided-hd-trail-wl-heur*›
:: ‹[*is-decided-hd-trail-wl-heur-pre*]$_a$ *isasat-bounded-assn*^{k} → *bool-assn*›
⟨*proof*⟩

declare *is-decided-hd-trail-wl-code.refine*[*sepref-fr-rules*]
is-decided-hd-trail-wl-fast-code.refine[*sepref-fr-rules*]

sepref-definition *lit-and-ann-of-propagated-st-heur-code*
is ‹*RETURN o lit-and-ann-of-propagated-st-heur*›
:: ‹[*lit-and-ann-of-propagated-st-heur-pre*]$_a$
isasat-unbounded-assn^k → (*unat-lit-assn* ∗*a nat-assn*)›
⟨*proof*⟩

sepref-definition *lit-and-ann-of-propagated-st-heur-fast-code*
is ‹*RETURN o lit-and-ann-of-propagated-st-heur*›
:: ‹$[$*lit-and-ann-of-propagated-st-heur-pre*$]_a$
isasat-bounded-assn^k → (*unat-lit-assn* ∗*a* *uint64-nat-assn*)›
⟨*proof*⟩

declare *lit-and-ann-of-propagated-st-heur-fast-code.refine*[*sepref-fr-rules*]
lit-and-ann-of-propagated-st-heur-code.refine[*sepref-fr-rules*]

declare *isa-vmtf-unset-code.refine*[*sepref-fr-rules*]

sepref-definition *tl-state-wl-heur-code*
is ‹*RETURN o tl-state-wl-heur*›
:: ‹$[$*tl-state-wl-heur-pre*$]_a$
isasat-unbounded-assn^d → *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *tl-state-wl-heur-fast-code*
is ‹*RETURN o tl-state-wl-heur*›
:: ‹$[$*tl-state-wl-heur-pre*$]_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
⟨*proof*⟩

declare
tl-state-wl-heur-code.refine[*sepref-fr-rules*]
tl-state-wl-heur-fast-code.refine[*sepref-fr-rules*]
sepref-register *isasat-lookup-merge-eq2 update-confl-tl-wl-heur*
sepref-definition *update-confl-tl-wl-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹$[$*update-confl-tl-wl-heur-pre*$]_a$
nat-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} → *bool-assn* ∗*a* *isasat-unbounded-assn*›
⟨*proof*⟩

find-theorems *mark-used arena-assn*

sepref-definition *isa-mark-used-fast-code2*
is ‹*uncurry isa-mark-used*›
:: ‹(*arl64-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl64-assn uint32-assn*)›
⟨*proof*⟩

lemma *isa-mark-used-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-used-fast-code2*, *uncurry* (*RETURN* ∘∘ *mark-used*))
∈ $[$*uncurry arena-act-pre*$]_a$ *arena-fast-assn*^{d} $*_a$ *uint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

thm *isa-mark-used-code*
sepref-definition *update-confl-tl-wl-fast-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹$[\lambda((i, L), S).$ *update-confl-tl-wl-heur-pre* $((i, L), S) \wedge$ *isasat-fast* $S]_a$
uint64-nat-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *bool-assn* ∗*a* *isasat-bounded-assn*›
⟨*proof*⟩

declare *update-confl-tl-wl-code.refine*[*sepref-fr-rules*]
update-confl-tl-wl-fast-code.refine[*sepref-fr-rules*]

sepref-definition *is-in-option-lookup-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-option-lookup-conflict*)›
:: ‹$[\lambda(L, (c, n, xs)).\ atm\text{-}of\ L < length\ xs]_a$
$unat\text{-}lit\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *atm-is-in-conflict-st-heur-fast-code*
is ‹*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*)›
:: ‹$[atm\text{-}is\text{-}in\text{-}conflict\text{-}st\text{-}heur\text{-}pre]_a\ unat\text{-}lit\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *atm-is-in-conflict-st-heur-code*
is ‹*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*)›
:: ‹$[atm\text{-}is\text{-}in\text{-}conflict\text{-}st\text{-}heur\text{-}pre]_a\ unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

declare *atm-is-in-conflict-st-heur-fast-code.refine*[*sepref-fr-rules*]
atm-is-in-conflict-st-heur-code.refine[*sepref-fr-rules*]

sepref-register *skip-and-resolve-loop-wl-D is-in-conflict-st*
sepref-definition *skip-and-resolve-loop-wl-D*
is ‹*skip-and-resolve-loop-wl-D-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *skip-and-resolve-loop-wl-D-fast*
is ‹*skip-and-resolve-loop-wl-D-heur*›
:: ‹$[\lambda S.\ isasat\text{-}fast\ S]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *skip-and-resolve-loop-wl-D-fast.refine*[*sepref-fr-rules*]
skip-and-resolve-loop-wl-D.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Propagate-Conflict*
imports *IsaSAT-Setup IsaSAT-Inner-Propagation*
begin

Refining Propagate And Conflict

Unit Propagation, Inner Loop **definition** (**in** −) *length-ll-fs* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs* = (λ(-, -, -, -, -, -, *W*) *L*. *length* (*W L*))›

definition (**in** −) *length-ll-fs-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs-heur S L* = *length* (*watched-by-int S L*)›

lemma *length-ll-fs-heur-alt-def*:
‹*length-ll-fs-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L*. *length* (*W* ! *nat-of-lit L*))›
⟨*proof*⟩

lemma (**in** −) *get-watched-wl-heur-def*: ‹*get-watched-wl-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -). *W*)›
⟨*proof*⟩

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur* b) $\leq$ *uint64-max* $\Longrightarrow$
unit-propagation-inner-loop-wl-loop-D-heur-inv b a ($a1'$, $a1'a$, $a2'a$) $\Longrightarrow$
length (*get-clauses-wl-heur* $a2'a$) $\leq$ *uint64-max*›
⟨*proof*⟩

lemma *unit-propagation-inner-loop-wl-loop-D-heur-alt-def*:
‹*unit-propagation-inner-loop-wl-loop-D-heur* L S_0 = *do* {
ASSERT (*nat-of-lit* L < *length* (*get-watched-wl-heur* S_0));
ASSERT (*length* (*watched-by-int* S_0 L) $\leq$ *length* (*get-clauses-wl-heur* S_0));
let n = *length* (*watched-by-int* S_0 L);
let b = (*zero-uint64-nat*, *zero-uint64-nat*, S_0);
$WHILE_T{}^{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv\ S_0\ L}$
(λ(j, w, S). $w < n \wedge$ *get-conflict-wl-is-None-heur* S)
(λ(j, w, S). *do* {
unit-propagation-inner-loop-body-wl-heur L j w S
})
b
}›
⟨*proof*⟩

Unit propagation, Outer Loop **lemma** *select-and-remove-from-literals-to-update-wl-heur-alt-def*:
‹*select-and-remove-from-literals-to-update-wl-heur* =
(λ(M', N', D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*). *do* {
ASSERT(j < *length* (*fst* M'));
ASSERT(j + *1* $\leq$ *uint32-max*);
$L \leftarrow$ *isa-trail-nth* M' j;
RETURN ((M', N', D', j+*1*, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*), $-L$)
})
›
⟨*proof*⟩

definition *literals-to-update-wl-literals-to-update-wl-empty* :: ‹*twl-st-wl-heur* $\Rightarrow$ *bool*› **where**
‹*literals-to-update-wl-literals-to-update-wl-empty* S $\longleftrightarrow$
literals-to-update-wl-heur S < *isa-length-trail* (*get-trail-wl-heur* S)›

lemma *literals-to-update-wl-literals-to-update-wl-empty-alt-def*:
‹*literals-to-update-wl-literals-to-update-wl-empty* =
(λ(M', N', D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*). j < *isa-length-trail* M')›
⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-invI*:
‹*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S $\Longrightarrow$
isa-length-trail-pre (*get-trail-wl-heur* S)›
⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
‹*length* (*get-clauses-wl-heur* x) $\leq$ *uint64-max* $\Longrightarrow$
unit-propagation-outer-loop-wl-D-heur-inv x s' $\Longrightarrow$
length (*get-clauses-wl-heur* $a1'$) =

length (*get-clauses-wl-heur s'*) $\Longrightarrow$
length (*get-clauses-wl-heur s'*) $\leq$ *uint64-max*›
⟨*proof*⟩

end
theory *IsaSAT-Propagate-Conflict-SML*
imports *IsaSAT-Propagate-Conflict IsaSAT-Inner-Propagation-SML*
begin
sepref-definition *length-ll-fs-heur-code*
is ‹*uncurry* (*RETURN oo length-ll-fs-heur*)›
:: ‹[λ(*S*, *L*). *nat-of-lit L* < *length* (*get-watched-wl-heur S*)]$_a$
isasat-unbounded-assn^k $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *nat-assn*›
⟨*proof*⟩

declare *length-ll-fs-heur-code.refine*[*sepref-fr-rules*]

definition *length-aa64-u32* :: ‹(*'a::heap array-list64*) *array* $\Rightarrow$ *uint32* $\Rightarrow$ *uint64 Heap*› **where**
‹*length-aa64-u32 xs i* = *do* {
x $\leftarrow$ *nth-u-code xs i*;
arl64-length x}›

lemma *length-aa64-rule*[*sep-heap-rules*]:
‹*b* < *length xs* $\Longrightarrow$ (*b'*, *b*) $\in$ *uint32-nat-rel* $\Longrightarrow$ <*arrayO-assn* (*arl64-assn R*) *xs a*> *length-aa64-u32 a b'*
<λ*r*. *arrayO-assn* (*arl64-assn R*) *xs a* $*$ $\uparrow$ (*nat-of-uint64 r* = *length-ll xs b*)>$_t$›
⟨*proof*⟩

lemma *length-aa64-u32-hnr*[*sepref-fr-rules*]: ‹(*uncurry length-aa64-u32*, *uncurry* (*RETURN* $\circ\circ$ *length-ll*)) $\in$
[λ(*xs*, *i*). *i* < *length xs*]$_a$ (*arrayO-assn* (*arl64-assn R*))^k $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *length-ll-fs-heur-fast-code*
is ‹*uncurry* (*RETURN oo length-ll-fs-heur*)›
:: ‹[λ(*S*, *L*). *nat-of-lit L* < *length* (*get-watched-wl-heur S*)]$_a$
isasat-bounded-assn^k $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

declare *length-ll-fs-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-register *unit-propagation-inner-loop-body-wl-heur*

sepref-definition *unit-propagation-inner-loop-wl-loop-D*
is ‹*uncurry unit-propagation-inner-loop-wl-loop-D-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *nat-assn* $*a$ *nat-assn* $*a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *unit-propagation-inner-loop-wl-loop-D.refine*[*sepref-fr-rules*]

sepref-definition *unit-propagation-inner-loop-wl-loop-D-fast*
is ‹*uncurry unit-propagation-inner-loop-wl-loop-D-heur*›
:: ‹[λ(*L*, *S*). *length* (*get-clauses-wl-heur S*) $\leq$ *uint64-max*]$_a$
unat-lit-assn^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *uint64-nat-assn* $*a$ *uint64-nat-assn* $*a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *unit-propagation-inner-loop-wl-loop-D-fast.refine*[*sepref-fr-rules*]

sepref-register *length-ll-fs-heur*

sepref-register *unit-propagation-inner-loop-wl-loop-D-heur cut-watch-list-heur2*
sepref-definition *cut-watch-list-heur2-code*
 is ‹*uncurry3 cut-watch-list-heur2*›
 :: ‹$nat\text{-}assn^k *_a nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
  $isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
 ⟨*proof*⟩

declare *cut-watch-list-heur2-code.refine*[*sepref-fr-rules*]

definition (**in** −) *shorten-take-aa64-u32* **where**
 ‹*shorten-take-aa64-u32 L j W* = *do* {
  (*a*, *n*) ← *nth-u-code W L*;
  Array-upd-u L (*a*, *j*) *W*
 }›

lemma *shorten-take-aa-hnr*[*sepref-fr-rules*]:
 ‹(*uncurry2 shorten-take-aa64-u32*, *uncurry2* (*RETURN ooo shorten-take-ll*)) ∈
  $[\lambda((L, j), W).\ j \leq length\ (W\ !\ L) \wedge L < length\ W]_a$
  $uint32\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a (arrayO\text{-}assn\ (arl64\text{-}assn\ R))^d \rightarrow arrayO\text{-}assn\ (arl64\text{-}assn$
$R)$›
 ⟨*proof*⟩

find-theorems *shorten-take-ll arl64-assn*
thm *shorten-take-aa-hnr*
sepref-definition *cut-watch-list-heur2-fast-code*
 is ‹*uncurry3 cut-watch-list-heur2*›
 :: ‹$[\lambda(((j, w), L), S).\ length\ (watched\text{-}by\text{-}int\ S\ L) \leq uint64\text{-}max-4]_a$
  $uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
  $isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
 ⟨*proof*⟩

declare *cut-watch-list-heur2-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *unit-propagation-inner-loop-wl-D-code*
 is ‹*uncurry unit-propagation-inner-loop-wl-D-heur*›
 :: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
 ⟨*proof*⟩

declare *unit-propagation-inner-loop-wl-D-code.refine*[*sepref-fr-rules*]

sepref-definition *unit-propagation-inner-loop-wl-D-fast-code*
 is ‹*uncurry unit-propagation-inner-loop-wl-D-heur*›
 :: ‹$[\lambda(L, S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
  $unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
 ⟨*proof*⟩

declare *unit-propagation-inner-loop-wl-D-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *select-and-remove-from-literals-to-update-wl-code*
 is ‹*select-and-remove-from-literals-to-update-wl-heur*›
 :: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ unat\text{-}lit\text{-}assn$›

⟨*proof*⟩

declare *select-and-remove-from-literals-to-update-wl-code.refine*[*sepref-fr-rules*]

sepref-definition *select-and-remove-from-literals-to-update-wlfast-code*
is ‹*select-and-remove-from-literals-to-update-wl-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn *a\ unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *select-and-remove-from-literals-to-update-wlfast-code.refine*[*sepref-fr-rules*]

sepref-definition *literals-to-update-wl-literals-to-update-wl-empty-code*
is ‹*RETURN o literals-to-update-wl-literals-to-update-wl-empty*›
:: ‹$[\lambda S.\ isa\text{-}length\text{-}trail\text{-}pre\ (get\text{-}trail\text{-}wl\text{-}heur\ S)]_a\ isasat\text{-}unbounded\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

declare *literals-to-update-wl-literals-to-update-wl-empty-code.refine*[*sepref-fr-rules*]

sepref-definition *literals-to-update-wl-literals-to-update-wl-empty-fast-code*
is ‹*RETURN o literals-to-update-wl-literals-to-update-wl-empty*›
:: ‹$[\lambda S.\ isa\text{-}length\text{-}trail\text{-}pre\ (get\text{-}trail\text{-}wl\text{-}heur\ S)]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow bool\text{-}assn$›
⟨*proof*⟩

declare *literals-to-update-wl-literals-to-update-wl-empty-fast-code.refine*[*sepref-fr-rules*]

sepref-register *literals-to-update-wl-literals-to-update-wl-empty*
select-and-remove-from-literals-to-update-wl-heur

sepref-definition *unit-propagation-outer-loop-wl-D-code*
is ‹*unit-propagation-outer-loop-wl-D-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *unit-propagation-outer-loop-wl-D-code.refine*[*sepref-fr-rules*]

sepref-definition *unit-propagation-outer-loop-wl-D-fast-code*
is ‹*unit-propagation-outer-loop-wl-D-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *unit-propagation-outer-loop-wl-D-fast-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT-Decide*
imports *IsaSAT-Setup IsaSAT-VMTF*
begin

Decide **lemma** (**in** −)*not-is-None-not-None*: ‹$\neg is\text{-}None\ s \Longrightarrow s \neq None$›
⟨*proof*⟩

definition *vmtf-find-next-undef-upd*
:: ‹$nat\ multiset \Rightarrow (nat,nat)ann\text{-}lits \Rightarrow vmtf\text{-}remove\text{-}int \Rightarrow$
$(((nat,nat)ann\text{-}lits \times vmtf\text{-}remove\text{-}int) \times nat\ option)nres$›
where

‹*vmtf-find-next-undef-upd* $\mathcal{A}$ = (λ*M vm*. *do*{
  L ← *vmtf-find-next-undef* $\mathcal{A}$ *vm M*;
  RETURN ((*M*, *update-next-search L vm*), *L*)
})›

definition *isa-vmtf-find-next-undef-upd*
:: ‹*trail-pol* ⇒ *isa-vmtf-remove-int* ⇒
  ((*trail-pol* × *isa-vmtf-remove-int*) × *nat option*)*nres*›
where
‹*isa-vmtf-find-next-undef-upd* = (λ*M vm*. *do*{
  L ← *isa-vmtf-find-next-undef vm M*;
  RETURN ((*M*, *update-next-search L vm*), *L*)
})›

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
‹(*uncurry isa-vmtf-find-next-undef-upd*, *uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$)) ∈
  trail-pol $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\rightarrow_f$
    ⟨*trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
⟨*proof*⟩

definition *lit-of-found-atm* **where**
‹*lit-of-found-atm* φ *L* = *SPEC* (λ*K*. (*L* = *None* ⟶ *K* = *None*) ∧
  (*L* ≠ *None* ⟶ *K* ≠ *None* ∧ *atm-of* (*the K*) = *the L*))›

definition *find-undefined-atm*
:: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf-remove-int* ⇒
  (((*nat*,*nat*) *ann-lits* × *vmtf-remove-int*) × *nat option*) *nres*›
where
‹*find-undefined-atm* $\mathcal{A}$ *M* - = *SPEC*(λ((*M'*, *vm*), *L*).
  (*L* ≠ *None* ⟶ *Pos* (*the L*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *undefined-atm M* (*the L*)) ∧
  (*L* = *None* ⟶ (∀ *K*∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M K*)) ∧ *M* = *M'* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*)›

definition *lit-of-found-atm-D-pre* **where**
‹*lit-of-found-atm-D-pre* = (λ(φ, *L*). *L* ≠ *None* ⟶ (*the L* < *length* φ ∧ *the L* ≤ *uint-max div 2*))›

definition *find-unassigned-lit-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × *nat literal option*) *nres*›
where
‹*find-unassigned-lit-wl-D-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vm*, φ, *clvls*). *do* {
  ((*M*, *vm*), *L*) ← *isa-vmtf-find-next-undef-upd M vm*;
  ASSERT(*lit-of-found-atm-D-pre* (φ, *L*));
  L ← *lit-of-found-atm* φ *L*;
  RETURN ((*M*, *N*, *D*, *WS*, *Q*, *vm*, φ, *clvls*), *L*)
})›

lemma *lit-of-found-atm-D-pre*:
‹*phase-saving* $\mathcal{A}$ φ ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹ (*L* ≠ *None* ⟹ *the L* ∈# $\mathcal{A}$) ⟹ *lit-of-found-atm-D-pre* (φ, *L*)›
⟨*proof*⟩

definition *find-unassigned-lit-wl-D-heur-pre* **where**
‹*find-unassigned-lit-wl-D-heur-pre S* ⟷
  (
  ∃ *T U*.
    (*S*, *T*) ∈ *state-wl-l None* ∧
    (*T*, *U*) ∈ *twl-st-l None* ∧

twl-struct-invs U ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* S) S ∧
get-conflict-wl S = *None*
)›

lemma *vmtf-find-next-undef-upd*:
‹(*uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$), *uncurry* (*find-undefined-atm* $\mathcal{A}$)) ∈
[λ(M, *vm*). *vm* ∈ *vmtf* $\mathcal{A}$ M]$_f$ *Id* $\times_f$ *Id* → ⟨*Id* $\times_f$ *Id* $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *find-unassigned-lit-wl-D′-find-unassigned-lit-wl-D*:
‹(*find-unassigned-lit-wl-D-heur*, *find-unassigned-lit-wl-D*) ∈
[*find-unassigned-lit-wl-D-heur-pre*]$_f$
twl-st-heur‴ r → ⟨{((T, L), (T', L')). (T, T') ∈ *twl-st-heur‴* r ∧ L = L' ∧
(L ≠ *None* ⟶ *undefined-lit* (*get-trail-wl* T') (*the* L) ∧ *the* L ∈# $\mathcal{L}_{all}$ (*all-atms-st* T')) ∧
get-conflict-wl T' = *None*}⟩*nres-rel*›
⟨*proof*⟩

definition *lit-of-found-atm-D*
:: ‹*bool list* ⇒ *nat option* ⇒ (*nat literal option*)*nres*› **where**
‹*lit-of-found-atm-D* = (λ(φ::*bool list*) L. *do*{
case L *of*
None ⇒ *RETURN None*
| *Some* L ⇒ *do* {
if φ!L *then RETURN* (*Some* (*Pos* L)) *else RETURN* (*Some* (*Neg* L))
}
})›

lemma *lit-of-found-atm-D-lit-of-found-atm*:
‹(*uncurry lit-of-found-atm-D*, *uncurry lit-of-found-atm*) ∈
[*lit-of-found-atm-D-pre*]$_f$ *Id* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *decide-lit-wl-heur* :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*decide-lit-wl-heur* = (λL' (M, N, D, Q, W, *vmtf*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*). *do* {
ASSERT(*isa-length-trail-pre* M);
let j = *isa-length-trail* M;
ASSERT(*cons-trail-Decided-tr-pre* (L', M));
RETURN (*cons-trail-Decided-tr* L' M, N, D, j, W, *vmtf*, φ, *clvls*, *cach*, *lbd*, *outl*, *incr-decision stats*,
fema, *sema*)})›

definition *decide-wl-or-skip-D-heur*
:: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*decide-wl-or-skip-D-heur* S = (*do* {
(S, L) ← *find-unassigned-lit-wl-D-heur* S;
case L *of*
None ⇒ *RETURN* (*True*, S)
| *Some* L ⇒ *do* {T ← *decide-lit-wl-heur* L S; *RETURN* (*False*, T)}

})
›

lemma *decide-wl-or-skip-D-heur-decide-wl-or-skip-D*:
‹(*decide-wl-or-skip-D-heur*, *decide-wl-or-skip-D*) $\in$ *twl-st-heur'''* *r* $\rightarrow_f$ ⟨*bool-rel* $\times_f$ *twl-st-heur'''* *r*⟩ *nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-Decide-SML*
imports *IsaSAT-Decide IsaSAT-VMTF-SML IsaSAT-Setup-SML*
begin

sepref-register *vmtf-find-next-undef*

sepref-definition *vmtf-find-next-undef-code*
is ‹*uncurry* (*isa-vmtf-find-next-undef*)›
:: ‹*vmtf-remove-conc*^{k} $*_a$ *trail-pol-assn*^{k} $\rightarrow_a$ *option-assn uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *vmtf-find-next-undef-fast-code*
is ‹*uncurry* (*isa-vmtf-find-next-undef*)›
:: ‹*vmtf-remove-conc*^{k} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *option-assn uint32-nat-assn*›
⟨*proof*⟩

declare *vmtf-find-next-undef-code.refine*[*sepref-fr-rules*]
vmtf-find-next-undef-fast-code.refine[*sepref-fr-rules*]

sepref-register *vmtf-find-next-undef-upd*
sepref-definition *vmtf-find-next-undef-upd-code*
is ‹*uncurry* (*isa-vmtf-find-next-undef-upd*)›
:: ‹*trail-pol-assn*^{d} $*_a$ *vmtf-remove-conc*^{d} $\rightarrow_a$
(*trail-pol-assn* $*a$ *vmtf-remove-conc*) $*a$
option-assn uint32-nat-assn›
⟨*proof*⟩

sepref-definition *vmtf-find-next-undef-upd-fast-code*
is ‹*uncurry isa-vmtf-find-next-undef-upd*›
:: ‹*trail-pol-fast-assn*^{d} $*_a$ *vmtf-remove-conc*^{d} $\rightarrow_a$
(*trail-pol-fast-assn* $*a$ *vmtf-remove-conc*) $*a$
option-assn uint32-nat-assn›
⟨*proof*⟩

declare *vmtf-find-next-undef-upd-code.refine*[*sepref-fr-rules*]
vmtf-find-next-undef-upd-fast-code.refine[*sepref-fr-rules*]

sepref-definition *lit-of-found-atm-D-code*
is ‹*uncurry lit-of-found-atm-D*›
:: ‹[*lit-of-found-atm-D-pre*]$_a$
(*array-assn bool-assn*)^k $*_a$ (*option-assn uint32-nat-assn*)^d $\rightarrow$
option-assn unat-lit-assn›
⟨*proof*⟩

declare *lit-of-found-atm-D-code.refine*[*sepref-fr-rules*]

lemma *lit-of-found-atm-hnr*[*sepref-fr-rules*]:
‹(*uncurry lit-of-found-atm-D-code*, *uncurry lit-of-found-atm*)
$\in$ [*lit-of-found-atm-D-pre*]$_a$
phase-saver-conc^k $*_a$ (*option-assn uint32-nat-assn*)^d $\rightarrow$
option-assn unat-lit-assn›
⟨*proof*⟩

sepref-register *find-undefined-atm*
sepref-definition *find-unassigned-lit-wl-D-code*
is ‹*find-unassigned-lit-wl-D-heur*›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ (*isasat-unbounded-assn* $*a$ *option-assn unat-lit-assn*)›
⟨*proof*⟩

sepref-definition *find-unassigned-lit-wl-D-fast-code*
is ‹*find-unassigned-lit-wl-D-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ (*isasat-bounded-assn* $*a$ *option-assn unat-lit-assn*)›
⟨*proof*⟩

declare *find-unassigned-lit-wl-D-code.refine*[*sepref-fr-rules*]
find-unassigned-lit-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-definition *decide-lit-wl-code*
is ‹*uncurry decide-lit-wl-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *decide-lit-wl-fast-code*
is ‹*uncurry decide-lit-wl-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *decide-lit-wl-code.refine*[*sepref-fr-rules*]
decide-lit-wl-fast-code.refine[*sepref-fr-rules*]

sepref-register *decide-wl-or-skip-D find-unassigned-lit-wl-D-heur decide-lit-wl-heur*
sepref-definition *decide-wl-or-skip-D-code*
is ‹*decide-wl-or-skip-D-heur*›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ *bool-assn* $*a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *decide-wl-or-skip-D-fast-code*
is ‹*decide-wl-or-skip-D-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool-assn* $*a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *decide-wl-or-skip-D-code.refine*[*sepref-fr-rules*]
decide-wl-or-skip-D-fast-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Show*
imports
Show.Show-Instances
IsaSAT-Setup

begin

0.2.6 Printing information about progress

We provide a function to print some information about the state. This is mostly meant to ease extracting statistics and printing information during the run. Remark that this function is basically an FFI (to follow Andreas Lochbihler words) and is not unsafe (since printing has not side effects), but we do not need any correctness theorems.

However, it seems that the PolyML as targeted by *export-code checking* does not support that print function. Therefore, we cannot provide the code printing equations by default.

definition *println-string* :: ‹*String.literal* ⇒ *unit*› **where**
‹*println-string* - = ()›

instantiation *uint64* :: *show*
begin
definition *shows-prec-uint64* :: ‹*nat* ⇒ *uint64* ⇒ *char list* ⇒ *char list*› **where**
‹*shows-prec-uint64 n m xs* = *shows-prec n* (*nat-of-uint64 m*) *xs*›

definition *shows-list-uint64* :: ‹*uint64 list* ⇒ *char list* ⇒ *char list*› **where**
‹*shows-list-uint64 xs ys* = *shows-list* (*map nat-of-uint64 xs*) *ys*›
instance
⟨*proof*⟩
end

instantiation *uint32* :: *show*
begin
definition *shows-prec-uint32* :: ‹*nat* ⇒ *uint32* ⇒ *char list* ⇒ *char list*› **where**
‹*shows-prec-uint32 n m xs* = *shows-prec n* (*nat-of-uint32 m*) *xs*›

definition *shows-list-uint32* :: ‹*uint32 list* ⇒ *char list* ⇒ *char list*› **where**
‹*shows-list-uint32 xs ys* = *shows-list* (*map nat-of-uint32 xs*) *ys*›
instance
⟨*proof*⟩
end

code-printing constant
println-string ⇀ (*SML*) *ignore*/ (*PolyML.print*/ ((-) ^ \n))

definition *test* **where**
‹*test* = *println-string*›

code-printing constant
println-string ⇀ (*SML*)

0.2.7 Print Information for IsaSAT

definition *isasat-header* :: *string* **where**
‹*isasat-header* = *show* ''*Conflict* | *Decision* | *Propagation* | *Restarts*''›

Printing the information slows down the solver by a huge factor.

definition *isasat-banner-content* **where**
‹*isasat-banner-content* =
''*c conflicts decisions restarts uset avg-lbd*
'' @

''c propagations reductions GC Learnt
'' @
''c clauses ''›

definition *isasat-information-banner* :: ‹*- ⇒ unit nres*› **where**
‹*isasat-information-banner - =*
RETURN (println-string (String.implode (show isasat-banner-content)))›

definition *zero-some-stats* :: ‹*stats ⇒ stats*› **where**
‹*zero-some-stats = (λ(propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds).*
(propa, confl, decs, frestarts, lrestarts, uset, gcs, 0))›

definition *isasat-current-information* :: ‹*stats ⇒ - ⇒ stats*› **where**
‹*isasat-current-information =*
(λ(propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds) lcount.
if confl AND 8191 = 8191 — *(8191::'b) = (8192::'b) − (1::'b)*, i.e., we print when all first bits are 1.
then let c = '' | '' in
let - = println-string (String.implode (show ''c | '' @ show confl @ show c @ show propa @
show c @ show decs @ show c @ show frestarts @ show c @ show lrestarts
@ show c @ show gcs @ show c @ show uset @ show c @ show lcount @ show c @ show (lbds >> 13))) in
zero-some-stats (propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds)
else (propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds)
)›

definition *print-current-information* :: ‹*stats ⇒ - ⇒ stats*› **where**
‹*print-current-information = (λ(propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds) -.*
if confl AND 8191 = 8191 then (propa, confl, decs, frestarts, lrestarts, uset, gcs, 0)
else (propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds))›

definition *isasat-current-status* :: ‹*twl-st-wl-heur ⇒ twl-st-wl-heur nres*› **where**
‹*isasat-current-status =*
(λ(M', N', D', j, W', vm, φ, clvls, cach, lbd, outl, stats,
fast-ema, slow-ema, ccount, avdom,
vdom, lcount, opts, old-arena).
let stats = (print-current-information stats lcount)
in RETURN (M', N', D', j, W', vm, φ, clvls, cach, lbd, outl, stats,
fast-ema, slow-ema, ccount, avdom,
vdom, lcount, opts, old-arena))›

lemma *isasat-current-status-id*:
‹*(isasat-current-status, RETURN o id)* ∈
{*(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ r*} $\rightarrow_f$
⟨{*(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ r*}⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-CDCL*
imports *IsaSAT-Propagate-Conflict IsaSAT-Conflict-Analysis IsaSAT-Backtrack*
IsaSAT-Decide IsaSAT-Show
begin

Combining Together: the Other Rules **definition** *cdcl-twl-o-prog-wl-D-heur*

:: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*cdcl-twl-o-prog-wl-D-heur S* =
  do {
    if get-conflict-wl-is-None-heur S
    then decide-wl-or-skip-D-heur S
    else do {
      if count-decided-st-heur S > *zero-uint32-nat*
      then do {
        T ← *skip-and-resolve-loop-wl-D-heur S*;
        ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur T*));
        U ← *backtrack-wl-D-nlit-heur T*;
        U ← *isasat-current-status U*; — Print some information every once in a while
        RETURN (*False*, *U*)
      }
      else RETURN (*True*, *S*)
    }
  }
›

lemma *twl-st-heur''D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge\mathcal{D}$ *r*. *f* ∈ *twl-st-heur''* $\mathcal{D}$ *r* $\to_f$ ⟨*twl-st-heur''* $\mathcal{D}$ *r*⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* $\to_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
⟨*proof*⟩

lemma *twl-st-heur'''D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge$*r*. *f* ∈ *twl-st-heur'''* *r* $\to_f$ ⟨*twl-st-heur'''* *r*⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* $\to_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
⟨*proof*⟩

lemma *twl-st-heur'''D-twl-st-heurD-prod*:
assumes *H*: ‹($\bigwedge$*r*. *f* ∈ *twl-st-heur'''* *r* $\to_f$ ⟨*A* $\times_r$ *twl-st-heur'''* *r*⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* $\to_f$ ⟨*A* $\times_r$ *twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
⟨*proof*⟩

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*:
‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl-D*) ∈
  {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) = *r*} $\to_f$
    ⟨*bool-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
      length (*get-clauses-wl-heur S*) ≤ *r* + *6* + *uint32-max div 2*}⟩*nres-rel*›
⟨*proof*⟩

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2*:
‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl-D*) ∈
  {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*} $\to_f$
    ⟨*bool-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*}⟩*nres-rel*›
⟨*proof*⟩

Combining Together: Full Strategy **definition** *cdcl-twl-stgy-prog-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cdcl-twl-stgy-prog-wl-D-heur* S_0 =
do {
  do {

(brk, T) ← $WHILE_T$
(λ(brk, -). ¬brk)
(λ(brk, S).
do {
T ← *unit-propagation-outer-loop-wl-D-heur* S;
cdcl-twl-o-prog-wl-D-heur T
})
(*False*, S_0);
RETURN T
}
}
›

theorem *unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D*:
‹(*unit-propagation-outer-loop-wl-D-heur*, *unit-propagation-outer-loop-wl-D*) ∈
twl-st-heur $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*›
⟨*proof*⟩

lemma *cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D*:
‹(*cdcl-twl-stgy-prog-wl-D-heur*, *cdcl-twl-stgy-prog-wl-D*) ∈ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

definition *cdcl-twl-stgy-prog-break-wl-D-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cdcl-twl-stgy-prog-break-wl-D-heur* S_0 =
do {
b ← *RETURN* (*isasat-fast* S_0);
(b, brk, T) ← $WHILE_T{}^{\lambda(b,\ brk,\ T).\ True}$
(λ(b, brk, -). b ∧ ¬brk)
(λ(b, brk, S).
do {
ASSERT(*isasat-fast* S);
T ← *unit-propagation-outer-loop-wl-D-heur* S;
ASSERT(*isasat-fast* T);
(brk, T) ← *cdcl-twl-o-prog-wl-D-heur* T;
b ← *RETURN* (*isasat-fast* T);
RETURN(b, brk, T)
})
(b, *False*, S_0);
if brk then RETURN T
else cdcl-twl-stgy-prog-wl-D-heur T
}›

end
theory *IsaSAT-Show-SML*
imports
IsaSAT-Show
IsaSAT-Setup-SML
begin

definition *isasat-information-banner-code* :: ‹- ⇒ *unit Heap*› **where**
‹*isasat-information-banner-code* - =
return (*println-string* (*String.implode* (*show isasat-banner-content*)))›

sepref-register *isasat-information-banner*
lemma *isasat-information-banner-hnr*[*sepref-fr-rules*]:
‹(*isasat-information-banner-code*, *isasat-information-banner*) ∈
$R^k \rightarrow_a$ *id-assn*›
⟨*proof*⟩

sepref-register *print-current-information*

lemma *print-current-information-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo isasat-current-information*), *uncurry* (*RETURN oo print-current-information*)) ∈
stats-assn$^k *_a$ *nat-assn*$^k \rightarrow_a$ *stats-assn*›
⟨*proof*⟩

lemma *print-current-information-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo isasat-current-information*), *uncurry* (*RETURN oo print-current-information*)) ∈
stats-assn$^k *_a$ *uint64-nat-assn*$^k \rightarrow_a$ *stats-assn*›
⟨*proof*⟩

sepref-definition *isasat-current-status-code*
is ‹*isasat-current-status*›
:: ‹*isasat-unbounded-assn*$^d \rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

declare *isasat-current-status-code.refine*[*sepref-fr-rules*]

sepref-definition *isasat-current-status-fast-code*
is ‹*isasat-current-status*›
:: ‹*isasat-bounded-assn*$^d \rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *isasat-current-status-fast-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT-CDCL-SML*
imports *IsaSAT-CDCL IsaSAT-Propagate-Conflict-SML IsaSAT-Conflict-Analysis-SML*
IsaSAT-Backtrack-SML
IsaSAT-Decide-SML IsaSAT-Show-SML
begin

sepref-register *get-conflict-wl-is-None decide-wl-or-skip-D-heur skip-and-resolve-loop-wl-D-heur*
backtrack-wl-D-nlit-heur isasat-current-status count-decided-st-heur get-conflict-wl-is-None-heur

sepref-register *cdcl-twl-o-prog-wl-D*

sepref-definition *cdcl-twl-o-prog-wl-D-code*
is ‹*cdcl-twl-o-prog-wl-D-heur*›
:: ‹*isasat-unbounded-assn*$^d \rightarrow_a$ *bool-assn* ∗*a* *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *cdcl-twl-o-prog-wl-D-fast-code*
is ‹*cdcl-twl-o-prog-wl-D-heur*›

:: ‹[*isasat-fast*]$_a$
isasat-bounded-assn^d → *bool-assn* ∗*a* *isasat-bounded-assn*›
⟨*proof*⟩

declare *cdcl-twl-o-prog-wl-D-code.refine*[*sepref-fr-rules*]
cdcl-twl-o-prog-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-register *cdcl-twl-stgy-prog-wl-D unit-propagation-outer-loop-wl-D-heur*
cdcl-twl-o-prog-wl-D-heur

sepref-definition *cdcl-twl-stgy-prog-wl-D-code*
is ‹*cdcl-twl-stgy-prog-wl-D-heur*›
:: ‹*isasat-unbounded-assn*^{d} →$_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

export-code *cdcl-twl-stgy-prog-wl-D-code* **in** *SML-imp* **module-name** *SAT-Solver*
file *code/CDCL-Cached-Array-Trail.sml*

end
theory *IsaSAT-Restart-Heuristics*
imports *Watched-Literals.WB-Sort Watched-Literals.Watched-Literals-Watch-List-Domain-Restart*
IsaSAT-Setup IsaSAT-VMTF
begin

This is a list of comments (how does it work for glucose and cadical) to prepare the future refinement:

1. Reduction

   - every 2000+300*n (rougly since inprocessing changes the real number, cadical) (split over initialisation file); don't restart if level < 2 or if the level is less than the fast average
   - curRestart * nbclausesbeforereduce; curRestart = (conflicts / nbclausesbeforereduce) + 1 (glucose)

2. Killed

   - half of the clauses that **can** be deleted (i.e., not used since last restart), not strictly LBD, but a probability of being useful.
   - half of the clauses

3. Restarts:

   - EMA-14, aka restart if enough clauses and slow_glue_avg * opts.restartmargin > fast_glue (file ema.cpp)
   - (lbdQueue.getavg() * K) > (sumLBD / conflictsRestarts), *conflictsRestarts* > *LOWER-BOUND-FO* && *lbdQueue.isvalid*() && *trail.size*() > *R* ∗ *trailQueue.getavg*()

declare *all-atms-def*[*symmetric*,*simp*]

definition *twl-st-heur-restart* :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**

‹*twl-st-heur-restart* =
  $\{((M', N', D', j, W', vm, \varphi, clvls, cach, lbd, outl, stats, fast\text{-}ema, slow\text{-}ema, ccount,$
    $vdom, avdom, lcount, opts, old\text{-}arena),$
   $(M, N, D, NE, UE, Q, W)).$
  $(M', M) \in trail\text{-}pol\ (all\text{-}init\text{-}atms\ N\ NE) \wedge$
  $valid\text{-}arena\ N'\ N\ (set\ vdom) \wedge$
  $(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}init\text{-}atms\ N\ NE) \wedge$
  $(D = None \longrightarrow j \le length\ M) \wedge$
  $Q = uminus\ \text{`}\#\ lit\text{-}of\ \text{`}\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
  $(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}init\text{-}atms\ N\ NE)) \wedge$
  $vm \in isa\text{-}vmtf\ (all\text{-}init\text{-}atms\ N\ NE)\ M \wedge$
  $phase\text{-}saving\ (all\text{-}init\text{-}atms\ N\ NE)\ \varphi \wedge$
  $no\text{-}dup\ M \wedge$
  $clvls \in counts\text{-}maximum\text{-}level\ M\ D \wedge$
  $cach\text{-}refinement\text{-}empty\ (all\text{-}init\text{-}atms\ N\ NE)\ cach \wedge$
  $out\text{-}learned\ M\ D\ outl \wedge$
  $lcount = size\ (learned\text{-}clss\text{-}lf\ N) \wedge$
  $vdom\text{-}m\ (all\text{-}init\text{-}atms\ N\ NE)\ \ W\ N \subseteq set\ vdom \wedge$
  $mset\ avdom \subseteq\#\ mset\ vdom \wedge$
  $isasat\text{-}input\text{-}bounded\ (all\text{-}init\text{-}atms\ N\ NE) \wedge$
  $isasat\text{-}input\text{-}nempty\ (all\text{-}init\text{-}atms\ N\ NE) \wedge$
  $distinct\ vdom \wedge old\text{-}arena = []$
  }›

abbreviation *twl-st-heur''''* **where**
‹$twl\text{-}st\text{-}heur''''\ r \equiv \{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \le r\}$›

abbreviation *twl-st-heur-restart'''* **where**
‹$twl\text{-}st\text{-}heur\text{-}restart'''\ r \equiv \{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\text{-}restart \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) = r\}$›

abbreviation *twl-st-heur-restart''''* **where**
‹$twl\text{-}st\text{-}heur\text{-}restart''''\ r \equiv \{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\text{-}restart \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \le r\}$›

definition *twl-st-heur-restart-ana* :: ‹$nat \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$› **where**
‹$twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r = \{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\text{-}restart \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) = r\}$›

lemma *twl-st-heur-restart-anaD*: ‹$x \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r \Longrightarrow x \in twl\text{-}st\text{-}heur\text{-}restart$›
⟨*proof*⟩

lemma *twl-st-heur-restartD*: ‹$x \in twl\text{-}st\text{-}heur\text{-}restart \Longrightarrow x \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ (fst\ x)))$›
⟨*proof*⟩

definition *clause-score-ordering* **where**
‹$clause\text{-}score\text{-}ordering = (\lambda(lbd, act)\ (lbd', act').\ lbd < lbd' \vee (lbd = lbd' \wedge act \le act'))$›

lemma *unbounded-id*: ‹$unbounded\ (id :: nat \Rightarrow nat)$›
⟨*proof*⟩

global-interpretation *twl-restart-ops id*
⟨*proof*⟩

global-interpretation *twl-restart id*
⟨*proof*⟩

We first fix the function that proves termination. We don't take the "smallest" function possible (other possibilites that are growing slower include $\lambda n.\ n >> 50$). Remark that this scheme is not compatible with Luby (TODO: use Luby restart scheme every once in a while like CryptoMinisat?)

lemma *get-slow-ema-heur-alt-def*:
‹*RETURN o get-slow-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, (*ccount*, -), *lcount*). *RETURN sema*)›
⟨*proof*⟩

lemma *get-fast-ema-heur-alt-def*:
‹*RETURN o get-fast-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, *ccount*, *lcount*). *RETURN fema*)›
⟨*proof*⟩

fun (**in** −) *get-conflict-count-since-last-restart-heur* :: ‹*twl-st-wl-heur* ⇒ *uint64*› **where**
‹*get-conflict-count-since-last-restart-heur* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, (*ccount*, -), -)
= *ccount*›

lemma (**in** −) *get-counflict-count-heur-alt-def*:
‹*RETURN o get-conflict-count-since-last-restart-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, (*ccount*, -), *lcount*). *RETURN ccount*)›
⟨*proof*⟩

lemma *get-learned-count-alt-def*:
‹*RETURN o get-learned-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*). *RETURN lcount*)›
⟨*proof*⟩

definition (**in** −) *find-local-restart-target-level-int-inv* **where**
‹*find-local-restart-target-level-int-inv ns cs* =
(λ(*brk*, *i*). *i* ≤ *length cs* ∧ *length cs* < *uint32-max*)›

definition *find-local-restart-target-level-int*
:: ‹*trail-pol* ⇒ *isa-vmtf-remove-int* ⇒ *nat nres*›
where
‹*find-local-restart-target-level-int* =
(λ(*M*, *xs*, *lvls*, *reasons*, *k*, *cs*) ((*ns* :: *nat-vmtf-node list*, *m* :: *nat*, *fst-As*::*nat*, *lst-As*::*nat*,
next-search::*nat option*), -). *do* {
(*brk*, *i*) ← $WHILE_T{}^{find\text{-}local\text{-}restart\text{-}target\text{-}level\text{-}int\text{-}inv\ ns\ cs}$
(λ(*brk*, *i*). ¬*brk* ∧ *i* < *length-uint32-nat cs*)
(λ(*brk*, *i*). *do* {
ASSERT(*i* < *length cs*);
let t = (*cs* ! *i*);
ASSERT(*t* < *length M*);
let L = *atm-of* (*M* ! *t*);
ASSERT(*L* < *length ns*);
let brk = *stamp* (*ns* ! *L*) < *m*;
RETURN (*brk*, *if brk then i else i*+*one-uint32-nat*)
})
(*False*, *zero-uint32-nat*);
RETURN i
})›

definition *find-local-restart-target-level* **where**
‹*find-local-restart-target-level M - = SPEC*(λ*i. i* ≤ *count-decided M*)›

lemma *find-local-restart-target-level-alt-def*:
‹*find-local-restart-target-level M vm* = *do* {
    (*b*, *i*) ← *SPEC*(λ(*b*::*bool*, *i*). *i* ≤ *count-decided M*);
      RETURN i
  }›
⟨*proof*⟩

lemma *find-local-restart-target-level-int-find-local-restart-target-level*:
‹(*uncurry find-local-restart-target-level-int*, *uncurry find-local-restart-target-level*) ∈
  [λ(*M*, *vm*). *vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*]$_f$ *trail-pol* $\mathcal{A}$ $\times_r$ *Id* → ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *empty-Q* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*empty-Q* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*, *ccount*, *vdom*, *lcount*). *do*{
  ASSERT(*isa-length-trail-pre M*);
  let j = *isa-length-trail M*;
  RETURN (*M*, *N*, *D*, *j*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
    restart-info-restart-done ccount, *vdom*, *lcount*)
})›

definition *incr-restart-stat* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*incr-restart-stat* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
    res-info, *vdom*, *avdom*, *lcount*). *do*{
  RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *incr-restart stats*,
    ema-reinit fast-ema, *ema-reinit slow-ema*,
    restart-info-restart-done res-info, *vdom*, *avdom*, *lcount*)
})›

definition *incr-lrestart-stat* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*incr-lrestart-stat* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
  res-info, *vdom*, *avdom*, *lcount*). *do*{
  RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *incr-lrestart stats*,
    fast-ema, *slow-ema*,
    restart-info-restart-done res-info,
    vdom, *avdom*, *lcount*)
})›

definition *restart-abs-wl-heur-pre* :: ‹*twl-st-wl-heur* ⇒ *bool* ⇒ *bool*› **where**
‹*restart-abs-wl-heur-pre S brk* ⟷ (∃ *T*. (*S*, *T*) ∈ *twl-st-heur* ∧ *restart-abs-wl-D-pre T brk*)›

find-decomp-wl-st-int is the wrong function here, because unlike in the backtrack case, we also have to update the queue of literals to update. This is done in the function *empty-Q*.

definition *find-local-restart-target-level-st* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
‹*find-local-restart-target-level-st S* = *do* {
  find-local-restart-target-level-int (*get-trail-wl-heur S*) (*get-vmtf-heur S*)
}›

lemma *find-local-restart-target-level-st-alt-def*:
‹*find-local-restart-target-level-st* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*). *do* {
    find-local-restart-target-level-int M vm})›

⟨*proof*⟩

definition *cdcl-twl-local-restart-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cdcl-twl-local-restart-wl-D-heur* = (λ*S*. *do* {
ASSERT(*restart-abs-wl-heur-pre S False*);
lvl ← *find-local-restart-target-level-st S*;
if lvl = *count-decided-st-heur S*
then RETURN S
else do {
S ← *find-decomp-wl-st-int lvl S*;
S ← *empty-Q S*;
incr-lrestart-stat S
}
})›

named-theorems *twl-st-heur-restart*

lemma [*twl-st-heur-restart*]:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹(*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)›
⟨*proof*⟩

lemma *trail-pol-literals-are-in-$\mathcal{L}_{in}$-trail*:
‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *literals-are-in-$\mathcal{L}_{in}$-trail* $\mathcal{A}$ M›
⟨*proof*⟩

lemma *refine-generalise1*: $A \leq B \Longrightarrow do\ \{x \leftarrow B;\ C\ x\} \leq D \Longrightarrow do\ \{x \leftarrow A;\ C\ x\} \leq (D{::}\ 'a\ nres)$
⟨*proof*⟩

lemma *refine-generalise2*: $A \leq B \Longrightarrow do\ \{x \leftarrow do\ \{x \leftarrow B;\ A'\ x\};\ C\ x\} \leq D \Longrightarrow$
$do\ \{x \leftarrow do\ \{x \leftarrow A;\ A'\ x\};\ C\ x\} \leq (D{::}\ 'a\ nres)$
⟨*proof*⟩

lemma *cdcl-twl-local-restart-wl-D-spec-int*:
‹*cdcl-twl-local-restart-wl-D-spec* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) ≥ (*do* {
ASSERT(*restart-abs-wl-D-pre* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) *False*);
i ← *SPEC*(λ-. *True*);
if i
then RETURN (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)
else do {
(*M*, *Q'*) ← *SPEC*(λ(*M'*, *Q'*). (∃ *K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition* *M*) ∧
Q' = {#}) ∨ (*M'* = *M* ∧ *Q'* = *Q*));
RETURN (*M*, *N*, *D*, *NE*, *UE*, *Q'*, *W*)
}
})›
⟨*proof*⟩

lemma *trail-pol-no-dup*: ‹(*M*, *M'*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *no-dup M'*›
⟨*proof*⟩

lemma *cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec*:
‹(*cdcl-twl-local-restart-wl-D-heur*, *cdcl-twl-local-restart-wl-D-spec*) ∈

twl-st-heur''' r $\to_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*›
⟨*proof*⟩

definition *remove-all-annot-true-clause-imp-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* ⇒ *nat watcher list* ⇒ *nat* × *twl-st-wl-heur* ⇒ *bool*›
where
‹*remove-all-annot-true-clause-imp-wl-D-heur-inv S xs* = (λ(*i*, *T*).
∃ *S'* *T'*. (*S*, *S'*) ∈ *twl-st-heur-restart* ∧ (*T*, *T'*) ∈ *twl-st-heur-restart* ∧
remove-all-annot-true-clause-imp-wl-D-inv S' (*map fst xs*) (*i*, *T'*))
›

definition *remove-all-annot-true-clause-one-imp-heur*
:: ‹*nat* × *nat* × *arena* ⇒ (*nat* × *arena*) *nres*›
where
‹*remove-all-annot-true-clause-one-imp-heur* = (λ(*C*, *j*, *N*). *do* {
case arena-status N C of
DELETED ⇒ *RETURN* (*j*, *N*)
| *IRRED* ⇒ *RETURN* (*j*, *extra-information-mark-to-delete N C*)
| *LEARNED* ⇒ *RETURN* (*j*−*1*, *extra-information-mark-to-delete N C*)
})›

definition *remove-all-annot-true-clause-imp-wl-D-heur-pre* **where**
‹*remove-all-annot-true-clause-imp-wl-D-heur-pre L S* ⟷
(∃ *S'*. (*S*, *S'*) ∈ *twl-st-heur-restart*
∧ *remove-all-annot-true-clause-imp-wl-D-pre* (*all-init-atms-st S'*) *L S'*)›

definition *remove-all-annot-true-clause-imp-wl-D-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*remove-all-annot-true-clause-imp-wl-D-heur* = (λ*L* (*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *fast-ema*, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*remove-all-annot-true-clause-imp-wl-D-heur-pre L* (*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*,
cach, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*));
let xs = *W*!(*nat-of-lit L*);
(-, *lcount'*, *N*) ← $WHILE_T$λ(*i*, *j*, *N*). *remove-all-annot-true-clause-imp-wl-D-heur-inv* (*M*, *N0*, *D*, *Q*, *W*, *vm*,
(λ(*i*, *j*, *N*). *i* < *length xs*)
(λ(*i*, *j*, *N*). *do* {
ASSERT(*i* < *length xs*);
if clause-not-marked-to-delete-heur (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*) *i*
then do {
(*j*, *N*) ← *remove-all-annot-true-clause-one-imp-heur* (*fst* (*xs*!*i*), *j*, *N*);
ASSERT(*remove-all-annot-true-clause-imp-wl-D-heur-inv*
(*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*) *xs*
(*i*, *M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *j*, *opts*));
RETURN (*i*+*1*, *j*, *N*)
}
else
RETURN (*i*+*1*, *j*, *N*)
})

$(0, lcount, N0)$;
  RETURN $(M, N, D, Q, W, vm, \varphi, clvls, cach, lbd, outl, stats,$
 $fast\text{-}ema, slow\text{-}ema, ccount, vdom, avdom, lcount', opts)$
 })›

definition *minimum-number-between-restarts* :: ‹*uint64*› **where**
 ‹*minimum-number-between-restarts* = *50*›

definition *five-uint64* :: ‹*uint64*› **where**
 ‹*five-uint64* = *5*›

definition *upper-restart-bound-not-reached* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
 ‹*upper-restart-bound-not-reached* = $(\lambda(M', N', D', j, W', vm, \varphi, clvls, cach, lbd, outl, (props, decs,$ $confl, restarts, \text{-}), fast\text{-}ema, slow\text{-}ema, ccount,$
 $vdom, avdom, lcount, opts).$
 $lcount < 3000 + 1000 * nat\text{-}of\text{-}uint64\ restarts)$›

definition (**in** −) *lower-restart-bound-not-reached* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
 ‹*lower-restart-bound-not-reached* = $(\lambda(M', N', D', j, W', vm, \varphi, clvls, cach, lbd, outl,$
 $(props, decs, confl, restarts, \text{-}), fast\text{-}ema, slow\text{-}ema, ccount,$
 $vdom, avdom, lcount, opts, old).$
 $(\neg opts\text{-}reduce\ opts \vee (opts\text{-}restart\ opts \wedge (lcount < 2000 + 1000 * nat\text{-}of\text{-}uint64\ restarts))))$›

definition (**in** −) *clause-score-extract* :: ‹*arena* ⇒ *nat* ⇒ *nat* × *nat*› **where**
 ‹*clause-score-extract arena C* = (
 if arena-status arena C = *DELETED*
 then (*uint32-max*, *zero-uint32-nat*) — deleted elements are the largest possible
 else
 let lbd = *get-clause-LBD arena C in*
 let act = *arena-act arena C in*
 (*lbd*, *act*)
)›

definition *valid-sort-clause-score-pre-at* **where**
 ‹*valid-sort-clause-score-pre-at arena C* ⟷
 $(\exists\, i\ vdom.\ C = vdom\ !\ i \wedge arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom\ arena\ (vdom!i) \wedge$
 $(arena\text{-}status\ arena\ (vdom!i) \neq DELETED \longrightarrow$
 $(get\text{-}clause\text{-}LBD\text{-}pre\ arena\ (vdom!i) \wedge arena\text{-}act\text{-}pre\ arena\ (vdom!i)))$
 $\wedge\ i < length\ vdom)$›

definition (**in** −)*valid-sort-clause-score-pre* **where**
 ‹*valid-sort-clause-score-pre arena vdom* ⟷
 $(\forall\, C \in set\ vdom.\ arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom\ arena\ C \wedge$
 $(arena\text{-}status\ arena\ C \neq DELETED \longrightarrow$
 $(get\text{-}clause\text{-}LBD\text{-}pre\ arena\ C \wedge arena\text{-}act\text{-}pre\ arena\ C)))$›

definition *reorder-vdom-wl* :: ‹*'v twl-st-wl* ⇒ *'v twl-st-wl nres*› **where**
 ‹*reorder-vdom-wl S* = *RETURN S*›

definition (**in** −) *quicksort-clauses-by-score* :: ‹*arena* ⇒ *nat list* ⇒ *nat list nres*› **where**
 ‹*quicksort-clauses-by-score arena* =
 full-quicksort-ref clause-score-ordering (*clause-score-extract arena*)›

definition *remove-deleted-clauses-from-avdom* :: ‹-› **where**

‹remove-deleted-clauses-from-avdom N avdom0 = do {
  let n = length avdom0;
  (i, j, avdom) ← WHILE$_T$ λ(i, j, avdom). i ≤ j ∧ j ≤ n ∧ length avdom = length avdom0 ∧ mset (take i avdom @ dro
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < length avdom);
      if (avdom ! j) ∈# dom-m N then RETURN (i+1, j+1, swap avdom i j)
      else RETURN (i, j+1, avdom)
    })
    (0, 0, avdom0);
  ASSERT(i ≤ length avdom);
  RETURN (take i avdom)
}›

lemma *remove-deleted-clauses-from-avdom*: ‹remove-deleted-clauses-from-avdom N avdom0 ≤ SPEC(λavdom. mset avdom ⊆# mset avdom0)›
  ⟨proof⟩

definition *isa-remove-deleted-clauses-from-avdom* :: ‹-› **where**
‹isa-remove-deleted-clauses-from-avdom arena avdom0 = do {
  ASSERT(length avdom0 ≤ length arena);
  let n = length avdom0;
  (i, j, avdom) ← WHILE$_T$ λ(i, j, -). i ≤ j ∧ j ≤ n
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < n);
      ASSERT(arena-is-valid-clause-vdom arena (avdom!j) ∧ j < length avdom ∧ i < length avdom);
      if arena-status arena (avdom ! j) ≠ DELETED then RETURN (i+1, j+1, swap avdom i j)
      else RETURN (i, j+1, avdom)
    }) (0, 0, avdom0);
  ASSERT(i ≤ length avdom);
  RETURN (take i avdom)
}›

lemma *isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*:
  ‹valid-arena arena N (set vdom) ⟹ mset avdom0 ⊆# mset vdom ⟹ distinct vdom ⟹
  isa-remove-deleted-clauses-from-avdom arena avdom0 ≤ ⇓ Id (remove-deleted-clauses-from-avdom N avdom0)›
  ⟨proof⟩

definition (**in** −) *sort-vdom-heur* :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› **where**
  ‹sort-vdom-heur = (λ(M′, arena, D′, j, W′, vm, φ, clvls, cach, lbd, outl, stats, fast-ema, slow-ema, ccount,
      vdom, avdom, lcount). do {
    ASSERT(length avdom ≤ length arena);
    avdom ← isa-remove-deleted-clauses-from-avdom arena avdom;
    ASSERT(valid-sort-clause-score-pre arena avdom);
    ASSERT(length avdom ≤ length arena);
    avdom ← quicksort-clauses-by-score arena avdom;
    RETURN (M′, arena, D′, j, W′, vm, φ, clvls, cach, lbd, outl, stats, fast-ema, slow-ema, ccount,
      vdom, avdom, lcount)
  })›

lemma *sort-clauses-by-score-reorder*:

⟨*quicksort-clauses-by-score arena vdom* ≤ *SPEC*(λ*vdom'*. *mset vdom* = *mset vdom'*)⟩
⟨*proof*⟩

lemma *sort-vdom-heur-reorder-vdom-wl*:
⟨(*sort-vdom-heur*, *reorder-vdom-wl*) ∈ *twl-st-heur-restart-ana r* $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*⟩
⟨*proof*⟩

lemma (**in** −) *insort-inner-clauses-by-score-invI*:
⟨*valid-sort-clause-score-pre a ba* ⟹
mset ba = *mset a2'* ⟹
a1' < *length a2'* ⟹
valid-sort-clause-score-pre-at a (*a2'* ! *a1'*)⟩
⟨*proof*⟩

lemma *sort-clauses-by-score-invI*:
⟨*valid-sort-clause-score-pre a b* ⟹
mset b = *mset a2'* ⟹ *valid-sort-clause-score-pre a a2'*⟩
⟨*proof*⟩

definition *partition-main-clause* **where**
⟨*partition-main-clause arena* = *partition-main clause-score-ordering* (*clause-score-extract arena*)⟩

definition *partition-clause* **where**
⟨*partition-clause arena* = *partition-between-ref clause-score-ordering* (*clause-score-extract arena*)⟩

lemma *valid-sort-clause-score-pre-swap*:
⟨*valid-sort-clause-score-pre a b* ⟹ *x* < *length b* ⟹
ba < *length b* ⟹ *valid-sort-clause-score-pre a* (*swap b x ba*)⟩
⟨*proof*⟩

definition *div2* **where** [*simp*]: ⟨*div2 n* = *n div 2*⟩

definition *safe-minus* **where** ⟨*safe-minus a b* = (*if b* ≥ *a then 0 else a* − *b*)⟩

definition *opts-restart-st* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*opts-restart-st* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*, -). (*opts-restart opts*))⟩

definition *opts-reduction-st* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*opts-reduction-st* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*, -). (*opts-reduce opts*))⟩

definition *max-restart-decision-lvl* :: *nat* **where**
⟨*max-restart-decision-lvl* = *300*⟩

definition *max-restart-decision-lvl-code* :: *uint32* **where**
⟨*max-restart-decision-lvl-code* = *300*⟩

definition *restart-required-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool nres* **where**
⟨*restart-required-heur S n* = *do* {
let opt-red = *opts-reduction-st S*;
let opt-res = *opts-restart-st S*;
let sema = *ema-get-value* (*get-slow-ema-heur S*);
let limit = (*11* ∗ *sema*) >> *4*;
let fema = *ema-get-value* (*get-fast-ema-heur S*);

let ccount = get-conflict-count-since-last-restart-heur S;
let lcount = get-learned-count S;
let can-res = (*lcount* > *n*);
let min-reached = (*ccount* > *minimum-number-between-restarts*);
let level = count-decided-st-heur S;
let should-not-reduce = (¬*opt-red* ∨ *upper-restart-bound-not-reached S*);
RETURN ((*opt-res* ∨ *opt-red*) ∧
(*should-not-reduce* ⟶ *limit* > *fema*) ∧ *min-reached* ∧ *can-res* ∧
level > *two-uint32-nat* ∧ ~~This comment from Marijn Heule seems not to help:~~ ~~**term** *level* ≤ *max-restart-decision-lvl*~~
uint64-of-uint32-conv level > *nat-of-uint64-id-conv* (*fema* >> *32*))}
⟩

fun (**in** −) *get-reductions-count* :: ⟨*twl-st-wl-heur* ⇒ *uint64*⟩ **where**
⟨*get-reductions-count* (-, -, -, -, -, -, -,-,-,-,-,
(-, -, -, *lres*, -, -), -)
= *lres*⟩

lemma (**in** −) *get-reduction-count-alt-def*:
⟨*RETURN o get-reductions-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
(-, -, -, *lres*, -, -), *fema*, *sema*, -, *lcount*). *RETURN lres*)⟩
⟨*proof*⟩

definition *GC-EVERY* :: *uint64* **where**
⟨*GC-EVERY* = *15*⟩ — hard-coded limit

definition *GC-required-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool nres* **where**
⟨*GC-required-heur S n* = *do* {
let lres = get-reductions-count S;
RETURN (*lres AND GC-EVERY* = *GC-EVERY*)} ~~Temporary measure~~
⟩

definition *mark-to-delete-clauses-wl-D-heur-pre* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*mark-to-delete-clauses-wl-D-heur-pre S* ⟷
(∃ *S'*. (*S*, *S'*) ∈ *twl-st-heur-restart* ∧ *mark-to-delete-clauses-wl-D-pre S'*)⟩

lemma *mark-to-delete-clauses-wl-post-alt-def*:
⟨*mark-to-delete-clauses-wl-post S0 S* ⟷
(∃ *T0 T*.
(*S0*, *T0*) ∈ *state-wl-l None* ∧
(*S*, *T*) ∈ *state-wl-l None* ∧
(∃ *U0 U*. (*T0*, *U0*) ∈ *twl-st-l None* ∧
(*T*, *U*) ∈ *twl-st-l None* ∧
*remove-one-annot-true-clause*** *T0 T* ∧
twl-list-invs T0 ∧
twl-struct-invs U0 ∧
twl-list-invs T ∧
twl-struct-invs U ∧
get-conflict-l T0 = *None* ∧
clauses-to-update-l T0 = {#}) ∧
correct-watching S0 ∧ *correct-watching S*)⟩
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-pre-alt-def*:

⟨*mark-to-delete-clauses-wl-D-heur-pre S* ⟷
  (∃ *S'*. (*S*, *S'*) ∈ *twl-st-heur* ∧ *mark-to-delete-clauses-wl-D-pre S'*)⟩ (**is** *?A*) **and**
mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur:
  ⟨*mark-to-delete-clauses-wl-D-pre T* ⟹
    (*S*, *T*) ∈ *twl-st-heur* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart*⟩ (**is** ⟨- ⟹ - *?B*⟩) **and**
mark-to-delete-clauses-wl-post-twl-st-heur:
  ⟨*mark-to-delete-clauses-wl-post T0 T* ⟹
    (*S*, *T*) ∈ *twl-st-heur* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart*⟩ (**is** ⟨- ⟹ - *?C*⟩)
⟨*proof*⟩

definition *mark-garbage-heur* :: ⟨*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*⟩ **where**
⟨*mark-garbage-heur C i* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
    vdom, *avdom*, *lcount*, *opts*, *old-arena*).
  (*M'*, *extra-information-mark-to-delete N' C*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
    vdom, *delete-index-and-swap avdom i*, *lcount* − *1*, *opts*, *old-arena*))⟩

lemma *get-vdom-mark-garbage*[*simp*]:
⟨*get-vdom* (*mark-garbage-heur C i S*) = *get-vdom S*⟩
⟨*get-avdom* (*mark-garbage-heur C i S*) = *delete-index-and-swap* (*get-avdom S*) *i*⟩
⟨*proof*⟩

lemma *mark-garbage-heur-wl*:
assumes
  ⟨(*S*, *T*) ∈ *twl-st-heur-restart*⟩ **and**
  ⟨*C* ∈# *dom-m* (*get-clauses-wl T*)⟩ **and**
  ⟨¬ *irred* (*get-clauses-wl T*) *C*⟩ **and** ⟨*i* < *length* (*get-avdom S*)⟩
shows ⟨(*mark-garbage-heur C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart*⟩
⟨*proof*⟩

lemma *mark-garbage-heur-wl-ana*:
assumes
  ⟨(*S*, *T*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
  ⟨*C* ∈# *dom-m* (*get-clauses-wl T*)⟩ **and**
  ⟨¬ *irred* (*get-clauses-wl T*) *C*⟩ **and** ⟨*i* < *length* (*get-avdom S*)⟩
shows ⟨(*mark-garbage-heur C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart-ana r*⟩
⟨*proof*⟩

definition *mark-unused-st-heur* :: ⟨*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*⟩ **where**
⟨*mark-unused-st-heur C* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
    stats, *fast-ema*, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*).
  (*M'*, *arena-decr-act* (*mark-unused N' C*) *C*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*,
    lbd, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
    vdom, *avdom*, *lcount*, *opts*))⟩

lemma *mark-unused-st-heur-simp*[*simp*]:
⟨*get-avdom* (*mark-unused-st-heur C T*) = *get-avdom T*⟩
⟨*get-vdom* (*mark-unused-st-heur C T*) = *get-vdom T*⟩
⟨*proof*⟩

lemma *mark-unused-st-heur*:
assumes
  ⟨(*S*, *T*) ∈ *twl-st-heur-restart*⟩ **and**
  ⟨*C* ∈# *dom-m* (*get-clauses-wl T*)⟩

shows ‹$(mark\text{-}unused\text{-}st\text{-}heur\ C\ S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart$›
⟨*proof*⟩

lemma *mark-unused-st-heur-ana*:
assumes
‹$(S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r$› **and**
‹$C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T)$›
shows ‹$(mark\text{-}unused\text{-}st\text{-}heur\ C\ S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r$›
⟨*proof*⟩

lemma *twl-st-heur-restart-valid-arena*[*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart$›
shows ‹$valid\text{-}arena\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ (get\text{-}clauses\text{-}wl\ T)\ (set\ (get\text{-}vdom\ S))$›
⟨*proof*⟩

lemma *twl-st-heur-restart-get-avdom-nth-get-vdom*[*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart$› ‹$i < length\ (get\text{-}avdom\ S)$›
shows ‹$get\text{-}avdom\ S\ !\ i \in set\ (get\text{-}vdom\ S)$›
⟨*proof*⟩

lemma [*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in twl\text{-}st\text{-}heur\text{-}restart$› **and**
‹$C \in set\ (get\text{-}avdom\ S)$›
shows ‹$clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur\ S\ C \longleftrightarrow$
$(C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T))$› **and**
‹$C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T) \Longrightarrow arena\text{-}lit\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C = get\text{-}clauses\text{-}wl\ T \propto C\ !\ 0$›**and**
‹$C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T) \Longrightarrow arena\text{-}status\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C = LEARNED \longleftrightarrow \neg irred\ (get\text{-}clauses\text{-}wl\ T)\ C$›
‹$C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ T) \Longrightarrow arena\text{-}length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C = length\ (get\text{-}clauses\text{-}wl\ T \propto C)$›
⟨*proof*⟩

definition *number-clss-to-keep* :: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow nat$› **where**
‹$number\text{-}clss\text{-}to\text{-}keep = (\lambda(M',\ N',\ D',\ j,\ W',\ vm,\ \varphi,\ clvls,\ cach,\ lbd,\ outl,$
$(props,\ decs,\ confl,\ restarts,\ \text{-}),\ fast\text{-}ema,\ slow\text{-}ema,\ ccount,$
$vdom,\ avdom,\ lcount).$
$nat\text{-}of\text{-}uint64\ (1000 + 150 * restarts))$›

definition *access-vdom-at* :: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow nat \Rightarrow nat$› **where**
‹$access\text{-}vdom\text{-}at\ S\ i = get\text{-}avdom\ S\ !\ i$›

lemma *access-vdom-at-alt-def*:
‹$access\text{-}vdom\text{-}at = (\lambda(M',\ N',\ D',\ j,\ W',\ vm,\ \varphi,\ clvls,\ cach,\ lbd,\ outl,\ stats,\ fast\text{-}ema,\ slow\text{-}ema,$
$ccount,\ vdom,\ avdom,\ lcount)\ i.\ avdom\ !\ i)$›
⟨*proof*⟩

definition *access-vdom-at-pre* **where**
‹$access\text{-}vdom\text{-}at\text{-}pre\ S\ i \longleftrightarrow i < length\ (get\text{-}avdom\ S)$›

definition (**in** $-$) *MINIMUM-DELETION-LBD* :: *nat* **where**

‹*MINIMUM-DELETION-LBD* = *3*›

definition *delete-index-vdom-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*›**where**
‹*delete-index-vdom-heur* = (λ*i* (*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *avdom*, *lcount*).
(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *delete-index-and-swap avdom i*, *lcount*))›

lemma *in-set-delete-index-and-swapD*:
‹*x* ∈ *set* (*delete-index-and-swap xs i*) ⟹ *x* ∈ *set xs*›
⟨*proof*⟩

lemma *delete-index-vdom-heur-twl-st-heur-restart*:
‹(*S*, *T*) ∈ *twl-st-heur-restart* ⟹ *i* < *length* (*get-avdom S*) ⟹
(*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart*›
⟨*proof*⟩

lemma *delete-index-vdom-heur-twl-st-heur-restart-ana*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *i* < *length* (*get-avdom S*) ⟹
(*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart-ana r*›
⟨*proof*⟩

definition *mark-clauses-as-unused-wl-D-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*mark-clauses-as-unused-wl-D-heur* = (λ*i S*. *do* {
(-, *T*) ← $WHILE_T$
(λ(*i*, *S*). *i* < *length* (*get-avdom S*))
(λ(*i*, *T*). *do* {
ASSERT(*i* < *length* (*get-avdom T*));
ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-avdom S*));
ASSERT(*access-vdom-at-pre T i*);
let C = *get-avdom T* ! *i*;
ASSERT(*clause-not-marked-to-delete-heur-pre* (*T*, *C*));
if ¬*clause-not-marked-to-delete-heur T C then RETURN* (*i*, *delete-index-vdom-heur i T*)
else do {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*i+1*, *mark-unused-st-heur C T*)
}
})
(*i*, *S*);
RETURN T
})›

lemma *avdom-delete-index-vdom-heur*[*simp*]:
‹*get-avdom* (*delete-index-vdom-heur i S*) =
delete-index-and-swap (*get-avdom S*) *i*›
⟨*proof*⟩

lemma *mark-clauses-as-unused-wl-D-heur*:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*›
shows ‹*mark-clauses-as-unused-wl-D-heur i S* ≤ ⇓ (*twl-st-heur-restart-ana r*) (*SPEC* ((=) *T*))›
⟨*proof*⟩

definition *mark-to-delete-clauses-wl-D-heur*
:: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
where

```
‹mark-to-delete-clauses-wl-D-heur  = (λS0. do {
    ASSERT(mark-to-delete-clauses-wl-D-heur-pre S0);
    S ← sort-vdom-heur S0;
    let l = number-clss-to-keep S;
    ASSERT(length (get-avdom S) ≤ length (get-clauses-wl-heur S0));
    (i, T) ← WHILE_T^λ-. True
      (λ(i, S). i < length (get-avdom S))
      (λ(i, T). do {
        ASSERT(i < length (get-avdom T));
        ASSERT(access-vdom-at-pre T i);
        let C = get-avdom T ! i;
        ASSERT(clause-not-marked-to-delete-heur-pre (T, C));
        if ¬clause-not-marked-to-delete-heur T C then RETURN (i, delete-index-vdom-heur i T)
        else do {
          ASSERT(access-lit-in-clauses-heur-pre ((T, C), 0));
          ASSERT(length (get-clauses-wl-heur T) ≤ length (get-clauses-wl-heur S0));
          ASSERT(length (get-avdom T) ≤ length (get-clauses-wl-heur T));
          let L = access-lit-in-clauses-heur T C 0;
          D ← get-the-propagation-reason-pol (get-trail-wl-heur T) L;
          ASSERT(get-clause-LBD-pre (get-clauses-wl-heur T) C);
          ASSERT(arena-is-valid-clause-vdom (get-clauses-wl-heur T) C);
          ASSERT(arena-status (get-clauses-wl-heur T) C = LEARNED ⟶
            arena-is-valid-clause-idx (get-clauses-wl-heur T) C);
          ASSERT(arena-status (get-clauses-wl-heur T) C = LEARNED ⟶
    marked-as-used-pre (get-clauses-wl-heur T) C);
          let can-del = (D ≠ Some C) ∧
     arena-lbd (get-clauses-wl-heur T) C > MINIMUM-DELETION-LBD ∧
            arena-status (get-clauses-wl-heur T) C = LEARNED ∧
            arena-length (get-clauses-wl-heur T) C ≠ two-uint64-nat ∧
     ¬marked-as-used (get-clauses-wl-heur T) C;
          if can-del
          then
            do {
              ASSERT(mark-garbage-pre (get-clauses-wl-heur T, C) ∧ get-learned-count T ≥ 1);
              RETURN (i, mark-garbage-heur C i T)
            }
          else do {
    ASSERT(arena-act-pre (get-clauses-wl-heur T) C);
            RETURN (i+1, mark-unused-st-heur C T)
  }
        }
      })
      (l, S);
    ASSERT(length (get-avdom T) ≤ length (get-clauses-wl-heur S0));
    T ← mark-clauses-as-unused-wl-D-heur i T;
    incr-restart-stat T
  })›
```

lemma *twl-st-heur-restart-same-annotD*:

```
‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
    Propagated L C' ∈ set (get-trail-wl T) ⟹ C = C'›
‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
```

Decided L ∈ *set* (*get-trail-wl T*) ⟹ *False*›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$-*mono*:
‹*set-mset* $\mathcal{A}$ ⊆ *set-mset* $\mathcal{B}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{B}$›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$-*init-all*:
‹*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*) ⟹ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1a*)›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-alt-def*:
‹*mark-to-delete-clauses-wl-D-heur* = (λ*S0*. *do* {
ASSERT(*mark-to-delete-clauses-wl-D-heur-pre S0*);
S ← *sort-vdom-heur S0*;
- ← *RETURN* (*get-avdom S*);
l ← *RETURN* (*number-clss-to-keep S*);
ASSERT(*length* (*get-avdom S*) ≤ *length*(*get-clauses-wl-heur S0*));
(*i*, *T*) ← $WHILE_T{}^{\lambda\text{-}.\ True}$
(λ(*i*, *S*). *i* < *length* (*get-avdom S*))
(λ(*i*, *T*). *do* {
ASSERT(*i* < *length* (*get-avdom T*));
ASSERT(*access-vdom-at-pre T i*);
let C = *get-avdom T* ! *i*;
ASSERT(*clause-not-marked-to-delete-heur-pre* (*T*, *C*));
if(¬*clause-not-marked-to-delete-heur T C*) *then RETURN* (*i*, *delete-index-vdom-heur i T*)
else do {
ASSERT(*access-lit-in-clauses-heur-pre* ((*T*, *C*), *0*));
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *length* (*get-clauses-wl-heur S0*));
ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-clauses-wl-heur T*));
let L = *access-lit-in-clauses-heur T C 0*;
D ← *get-the-propagation-reason-pol* (*get-trail-wl-heur T*) *L*;
ASSERT(*get-clause-LBD-pre* (*get-clauses-wl-heur T*) *C*);
ASSERT(*arena-is-valid-clause-vdom* (*get-clauses-wl-heur T*) *C*);
ASSERT(*arena-status* (*get-clauses-wl-heur T*) *C* = *LEARNED* ⟶
arena-is-valid-clause-idx (*get-clauses-wl-heur T*) *C*);
ASSERT(*arena-status* (*get-clauses-wl-heur T*) *C* = *LEARNED* ⟶
marked-as-used-pre (*get-clauses-wl-heur T*) *C*);
let can-del = (*D* ≠ *Some C*) ∧
arena-lbd (*get-clauses-wl-heur T*) *C* > *MINIMUM-DELETION-LBD* ∧
arena-status (*get-clauses-wl-heur T*) *C* = *LEARNED* ∧
arena-length (*get-clauses-wl-heur T*) *C* ≠ *two-uint64-nat* ∧
¬*marked-as-used* (*get-clauses-wl-heur T*) *C*;
if can-del
then do {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur T*, *C*) ∧ *get-learned-count T* ≥ *1*);
RETURN (*i*, *mark-garbage-heur C i T*)
}
else do {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*i+1*, *mark-unused-st-heur C T*)
}
}
})
(*l*, *S*);
ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-clauses-wl-heur S0*));

$T \leftarrow$ *mark-clauses-as-unused-wl-D-heur i T*;
incr-restart-stat T
})›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D*:
‹(*mark-to-delete-clauses-wl-D-heur*, *mark-to-delete-clauses-wl-D*) $\in$
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

definition *cdcl-twl-full-restart-wl-prog-heur* **where**
‹*cdcl-twl-full-restart-wl-prog-heur S = do* {
- $\leftarrow$ *ASSERT* (*mark-to-delete-clauses-wl-D-heur-pre S*);
$T \leftarrow$ *mark-to-delete-clauses-wl-D-heur S*;
RETURN T
}›

lemma *cdcl-twl-full-restart-wl-prog-heur-cdcl-twl-full-restart-wl-prog-D*:
‹(*cdcl-twl-full-restart-wl-prog-heur*, *cdcl-twl-full-restart-wl-prog-D*) $\in$
twl-st-heur''' r $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*›
⟨*proof*⟩

definition *cdcl-twl-restart-wl-heur* **where**
‹*cdcl-twl-restart-wl-heur S = do* {
let b = lower-restart-bound-not-reached S;
if b then cdcl-twl-local-restart-wl-D-heur S
else cdcl-twl-full-restart-wl-prog-heur S
}›

lemma *cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog*:
‹(*cdcl-twl-restart-wl-heur*, *cdcl-twl-restart-wl-D-prog*) $\in$
twl-st-heur''' r $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*›
⟨*proof*⟩

definition *isasat-replace-annot-in-trail*
:: ‹*nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*isasat-replace-annot-in-trail L C* = (λ((*M*, *val*, *lvls*, *reason*, *k*), *oth*). *do* {
ASSERT(*atm-of L* < *length reason*);
RETURN ((*M*, *val*, *lvls*, *reason*[*atm-of L* := *0*], *k*), *oth*)
})›

lemma *trail-pol-replace-annot-in-trail-spec*:
assumes
‹*atm-of x2* < *length x1e*› **and**
x2: ‹*atm-of x2* $\in$# *all-init-atms-st* (*ys* @ *Propagated x2 C* # *zs*, *x2n'*)› **and**
‹(((*x1b*, *x1c*, *x1d*, *x1e*, *x2d*), *x2n*),
(*ys* @ *Propagated x2 C* # *zs*, *x2n'*))
$\in$ *twl-st-heur-restart-ana r*›
shows
‹(((*x1b*, *x1c*, *x1d*, *x1e*[*atm-of x2* := *0*], *x2d*), *x2n*),
(*ys* @ *Propagated x2 0* # *zs*, *x2n'*))
$\in$ *twl-st-heur-restart-ana r*›
⟨*proof*⟩

lemmas *trail-pol-replace-annot-in-trail-spec2* =
trail-pol-replace-annot-in-trail-spec[*of* ‹– -›, *simplified*]

lemma *isasat-replace-annot-in-trail-replace-annot-in-trail-spec*:
‹(*uncurry2 isasat-replace-annot-in-trail*,
uncurry2 replace-annot-l) ∈
$[\lambda((L, C), S).$
Propagated $L\ C \in$ *set* (*get-trail-wl* S) $\wedge$ *atm-of* $L \in\#$ *all-init-atms-st* $S]_f$
$Id \times_f Id \times_f$ *twl-st-heur-restart-ana* $r \rightarrow \langle$*twl-st-heur-restart-ana* $r\rangle$*nres-rel*›
⟨*proof*⟩

definition *mark-garbage-heur2* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mark-garbage-heur2* C = (λ(M', N', D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*). *do*{
let st = *arena-status* N' C = *IRRED*;
ASSERT($\neg$*st* $\longrightarrow$ *lcount* $\geq$ *1*);
RETURN (M', *extra-information-mark-to-delete* N' C, D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *if st then lcount else lcount* − *1*, *opts*) })›

definition *remove-one-annot-true-clause-one-imp-wl-D-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ (*nat* × *twl-st-wl-heur*) *nres*›
where
‹*remove-one-annot-true-clause-one-imp-wl-D-heur* = ($\lambda i\ S$. *do* {
(L, C) ← *do* {
L ← *isa-trail-nth* (*get-trail-wl-heur* S) i;
C ← *get-the-propagation-reason-pol* (*get-trail-wl-heur* S) L;
RETURN (L, C)};
ASSERT($C \neq$ *None* $\wedge$ i + *1* $\leq$ *uint32-max*);
if the C = *0 then RETURN* (i+*1*, S)
else do {
ASSERT($C \neq$ *None*);
S ← *isasat-replace-annot-in-trail* L (*the* C) S;
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur* S, *the* C) $\wedge$ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur* S) (*the* C));
S ← *mark-garbage-heur2* (*the* C) S;
— S ← *remove-all-annot-true-clause-imp-wl-D-heur* L S;
RETURN (i+*1*, S)
}
})›

definition *cdcl-twl-full-restart-wl-D-GC-prog-heur-post* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur* ⇒ *bool*› **where**
‹*cdcl-twl-full-restart-wl-D-GC-prog-heur-post* S T $\longleftrightarrow$
($\exists S'\ T'$. (S, S') ∈ *twl-st-heur-restart* ∧ (T, T') ∈ *twl-st-heur-restart* ∧
cdcl-twl-full-restart-wl-D-GC-prog-post S' T')›

definition *remove-one-annot-true-clause-imp-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* ⇒ (*nat* × *twl-st-wl-heur*) ⇒ *bool*› **where**
‹*remove-one-annot-true-clause-imp-wl-D-heur-inv* S = (λ(i, T).
($\exists S'\ T'$. (S, S') ∈ *twl-st-heur-restart* ∧ (T, T') ∈ *twl-st-heur-restart* ∧
remove-one-annot-true-clause-imp-wl-D-inv S' (i, T')))›

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where

‹remove-one-annot-true-clause-imp-wl-D-heur = (λS. *do* {
  ASSERT((*isa-length-trail-pre o get-trail-wl-heur*) S);
  $k \leftarrow$ (*if count-decided-st-heur* $S = 0$
    then RETURN (*isa-length-trail* (*get-trail-wl-heur* S))
    else get-pos-of-level-in-trail-imp (*get-trail-wl-heur* S) *0*);
  (-, S) $\leftarrow$ $WHILE_T{}^{remove\text{-}one\text{-}annot\text{-}true\text{-}clause\text{-}imp\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S}$
    ($\lambda(i, S)$. $i < k$)
    ($\lambda(i, S)$. *remove-one-annot-true-clause-one-imp-wl-D-heur* i S)
    (*0*, S);
  RETURN S
})›

lemma *get-pos-of-level-in-trail-le-decomp*:
assumes
‹(S, T) $\in$ *twl-st-heur-restart*›
shows ‹*get-pos-of-level-in-trail* (*get-trail-wl* T) *0*
  $\leq$ *SPEC*
    (λk. $\exists$ *M1*. ($\exists$ *M2* K.
        (*Decided* K # *M1*, *M2*)
        $\in$ *set* (*get-all-ann-decomposition* (*get-trail-wl* T))) $\wedge$
      count-decided M1 $= 0 \wedge k =$ *length M1*)›
⟨*proof*⟩

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
‹(S, T) $\in$ *twl-st-heur-restart-ana* $r \Longrightarrow$ *isa-length-trail* (*get-trail-wl-heur* S) $=$ *length* (*get-trail-wl* T)›
⟨*proof*⟩

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
fixes S :: *twl-st-wl-heur*
shows ‹(S, T) $\in$ *twl-st-heur-restart-ana* $r \Longrightarrow$ *count-decided-st-heur* $S =$ *count-decided* (*get-trail-wl* T)›
⟨*proof*⟩

lemma *twl-st-heur-restart-trailD*:
‹(S, T) $\in$ *twl-st-heur-restart-ana* $r \Longrightarrow$
  (*get-trail-wl-heur* S, *get-trail-wl* T)
  $\in$ *trail-pol* (*all-init-atms* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T))›
⟨*proof*⟩

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup* $M \Longrightarrow k <$ *length* $M \Longrightarrow M$! $k =$ *Propagated* L $C \Longrightarrow$ *Decided* $L \notin$ *set* M›
⟨*proof*⟩

lemma *remove-all-annot-true-clause-imp-wl-inv-length-cong*:
‹*remove-all-annot-true-clause-imp-wl-inv* S *xs* $T \Longrightarrow$
  length xs $=$ *length ys* $\Longrightarrow$ *remove-all-annot-true-clause-imp-wl-inv* S *ys* T›
⟨*proof*⟩

lemma *get-literal-and-reason*:
assumes
‹((k, S), k', T) $\in$ *nat-rel* $\times_f$ *twl-st-heur-restart-ana* r› **and**
‹*remove-one-annot-true-clause-one-imp-wl-D-pre* k' T› **and**
proped: ‹*is-proped* (*rev* (*get-trail-wl* T) ! k')›
shows ‹*do* {
    $L \leftarrow$ *isa-trail-nth* (*get-trail-wl-heur* S) k;

$C \leftarrow$ *get-the-propagation-reason-pol* (*get-trail-wl-heur* S) L;
RETURN $(L,\ C)$
} $\leq \Downarrow \{((L,\ C),\ L',\ C').\ L = L' \wedge C' = \textit{the } C \wedge C \neq \textit{None}\}$
(*SPEC* ($\lambda p.$ *rev* (*get-trail-wl* T) ! k' = *Propagated* (*fst* p) (*snd* p)))›
⟨*proof*⟩

lemma *red-in-dom-number-of-learned-ge1*: ‹$C' \in\#$ *dom-m baa* $\Longrightarrow \neg$ *irred baa* $C' \Longrightarrow$ *Suc 0* $\leq$ *size* (*learned-clss-l baa*)›
⟨*proof*⟩

lemma *mark-garbage-heur2-remove-and-add-cls-l*:
‹$(S,\ T) \in$ *twl-st-heur-restart-ana* $r \Longrightarrow (C,\ C') \in$ *Id* $\Longrightarrow$
$C \in\#$ *dom-m* (*get-clauses-wl* T) $\Longrightarrow$
mark-garbage-heur2 C S
$\leq \Downarrow$ (*twl-st-heur-restart-ana* r) (*remove-and-add-cls-l* C' T)›
⟨*proof*⟩

lemma *remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D*:
‹(*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*,
uncurry remove-one-annot-true-clause-one-imp-wl-D) $\in$
nat-rel $\times_f$ *twl-st-heur-restart-ana* $r \rightarrow_f$ ⟨*nat-rel* $\times_f$ *twl-st-heur-restart-ana* r⟩*nres-rel*›
⟨*proof*⟩

lemma *RES-RETURN-RES5*:
‹*SPEC* $\Phi \gg\!= (\lambda(T1,\ T2,\ T3,\ T4,\ T5).$ *RETURN* (f $T1$ $T2$ $T3$ $T4$ $T5$)) =
RES $((\lambda(a,\ b,\ c,\ d,\ e).\ f\ a\ b\ c\ d\ e)$ ‘ $\{T.\ \Phi\ T\})$›
⟨*proof*⟩

lemma *RES-RETURN-RES6*:
‹*SPEC* $\Phi \gg\!= (\lambda(T1,\ T2,\ T3,\ T4,\ T5,\ T6).$ *RETURN* (f $T1$ $T2$ $T3$ $T4$ $T5$ $T6$)) =
RES $((\lambda(a,\ b,\ c,\ d,\ e,\ f').\ f\ a\ b\ c\ d\ e\ f')$ ‘ $\{T.\ \Phi\ T\})$›
⟨*proof*⟩

lemma *RES-RETURN-RES7*:
‹*SPEC* $\Phi \gg\!= (\lambda(T1,\ T2,\ T3,\ T4,\ T5,\ T6,\ T7).$ *RETURN* (f $T1$ $T2$ $T3$ $T4$ $T5$ $T6$ $T7$)) =
RES $((\lambda(a,\ b,\ c,\ d,\ e,\ f',\ g).\ f\ a\ b\ c\ d\ e\ f'\ g)$ ‘ $\{T.\ \Phi\ T\})$›
⟨*proof*⟩

definition *find-decomp-wl0* **where**
‹*find-decomp-wl0* = $(\lambda(M,\ N,\ D,\ NE,\ UE,\ Q,\ W)\ (M',\ N',\ D',\ NE',\ UE',\ Q',\ W').$
$(\exists K\ M2.$ (*Decided* K # M', $M2$) $\in$ *set* (*get-all-ann-decomposition* M) $\wedge$
count-decided $M' = 0) \wedge$
$(N',\ D',\ NE',\ UE',\ Q',\ W') = (N,\ D,\ NE,\ UE,\ Q,\ W))$›

definition *empty-Q-wl* :: ‹-› **where**
‹*empty-Q-wl* = $(\lambda(M',\ N,\ D,\ NE,\ UE,\ \text{-},\ W).\ (M',\ N,\ D,\ NE,\ UE,\ \{\#\},\ W))$›

lemma *cdcl-twl-local-restart-wl-spec0-alt-def*:
‹*cdcl-twl-local-restart-wl-spec0* = ($\lambda S.$
if count-decided (*get-trail-wl* S) > *0*
then do {
$T \leftarrow$ *SPEC*(*find-decomp-wl0* S);
RETURN (*empty-Q-wl* T)

} *else RETURN S*)›
⟨*proof*⟩

lemma *cdcl-twl-local-restart-wl-spec0*:
assumes *Sy*: ‹$(S, y) \in$ *twl-st-heur-restart-ana r*› **and**
‹*get-conflict-wl y = None*›
shows ‹*do* {
if count-decided-st-heur S > 0
then do {
S ← find-decomp-wl-st-int 0 S;
empty-Q S
} *else RETURN S*
}
$\leq \Downarrow$ (*twl-st-heur-restart-ana r*) (*cdcl-twl-local-restart-wl-spec0 y*)›
⟨*proof*⟩

lemma *no-get-all-ann-decomposition-count-dec0*:
‹(∀ *M1*. (∀ *M2 K*. (*Decided K # M1*, *M2*) ∉ *set* (*get-all-ann-decomposition M*))) ⟷
count-decided M = 0›
⟨*proof*⟩

lemma *get-pos-of-level-in-trail-decomp-iff*:
assumes ‹*no-dup M*›
shows ‹((∃ *M1 M2 K*.
(*Decided K # M1*, *M2*)
∈ *set* (*get-all-ann-decomposition M*) ∧
count-decided M1 = 0 ∧ *k = length M1*)) ⟷
k < length M ∧ *count-decided M > 0* ∧ *is-decided* (*rev M ! k*) ∧ *get-level M* (*lit-of* (*rev M ! k*)) =
1›
(**is** ‹*?A* ⟷ *?B*›)
⟨*proof*⟩

lemma *remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D*:
‹(*remove-one-annot-true-clause-imp-wl-D-heur*, *remove-one-annot-true-clause-imp-wl-D*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl2-D*:
‹(*mark-to-delete-clauses-wl-D-heur*, *mark-to-delete-clauses-wl2-D*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

definition *iterate-over-VMTF* **where**
‹*iterate-over-VMTF* ≡ (λ*f* (*I* :: '*a* ⇒ *bool*) (*ns* :: (*nat*, *nat*) *vmtf-node list*, *n*) *x*. *do* {
(-, *x*) ← $WHILE_T{}^{\lambda(n,\ x).\ I\ x}$
(λ(*n*, -). *n* ≠ *None*)
(λ(*n*, *x*). *do* {
ASSERT(*n* ≠ *None*);
let A = the n;
ASSERT(*A < length ns*);
ASSERT(*A ≤ uint32-max div 2*);
x ← f A x;
RETURN (*get-next* ((*ns ! A*)), *x*)
})
(*n*, *x*);

$RETURN\ x$
})›

definition *iterate-over-$\mathcal{L}_{all}$* **where**
‹*iterate-over-$\mathcal{L}_{all}$* = ($\lambda f\ \mathcal{A}_0\ I\ x.\ do$ {
$\mathcal{A} \leftarrow SPEC(\lambda\mathcal{A}.\ set\text{-}mset\ \mathcal{A} = set\text{-}mset\ \mathcal{A}_0 \wedge distinct\text{-}mset\ \mathcal{A})$;
$(\text{-}, x) \leftarrow WHILE_T{}^{\lambda(\text{-},\ x).\ I\ x}$
$(\lambda(\mathcal{B}, \text{-}).\ \mathcal{B} \neq \{\#\})$
$(\lambda(\mathcal{B}, x).\ do$ {
$ASSERT(\mathcal{B} \neq \{\#\})$;
$A \leftarrow SPEC\ (\lambda A.\ A \in\# \mathcal{B})$;
$x \leftarrow f\ A\ x$;
$RETURN\ (remove1\text{-}mset\ A\ \mathcal{B}, x)$
})
$(\mathcal{A}, x)$;
$RETURN\ x$
})›

lemma *iterate-over-VMTF-iterate-over-$\mathcal{L}_{all}$*:
fixes $x :: 'a$
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
nempty: ‹$\mathcal{A} \neq \{\#\}$› ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*iterate-over-VMTF f I* (*ns*, *Some fst-As*) $x \leq \Downarrow$ *Id* (*iterate-over-$\mathcal{L}_{all}$ f* $\mathcal{A}$ *I x*)›
⟨*proof*⟩

definition *arena-is-packed* :: ‹*arena* $\Rightarrow$ *nat clauses-l* $\Rightarrow$ *bool*› **where**
‹*arena-is-packed arena N* $\longleftrightarrow$ *length arena* = ($\sum C \in\#$ *dom-m N*. *length* ($N \propto C$) + *header-size* ($N \propto C$))›

lemma *arena-is-packed-empty*[*simp*]: ‹*arena-is-packed* [] *fmempty*›
⟨*proof*⟩

lemma *sum-mset-cong*:
‹($\bigwedge A.\ A \in\# M \Longrightarrow f\ A = g\ A) \Longrightarrow (\sum\ A \in\# M.\ f\ A) = (\sum\ A \in\# M.\ g\ A)$›
⟨*proof*⟩
lemma *arena-is-packed-append*:
assumes ‹*arena-is-packed* (*arena*) *N*› **and**
[*simp*]: ‹*length C* = *length* (*fst C'*) + *header-size* (*fst C'*)› **and**
[*simp*]: ‹$a \notin\#$ *dom-m N*›
shows ‹*arena-is-packed* (*arena* @ *C*) (*fmupd a C' N*)›
⟨*proof*⟩

lemma *arena-is-packed-append-valid*:
assumes
in-dom: ‹*fst C* $\in\#$ *dom-m x1a*› **and**
valid0: ‹*valid-arena x1c x1a vdom0*› **and**
valid: ‹*valid-arena x1d x2a* (*set x2d*)› **and**
packed: ‹*arena-is-packed x1d x2a*› **and**
n: ‹*n* = *header-size* (*x1a* $\propto$ (*fst C*))›
shows ‹*arena-is-packed*
(*x1d* @
Misc.slice (*fst C* − *n*)
(*fst C* + *arena-length x1c* (*fst C*)) *x1c*)
(*fmupd* (*length x1d* + *n*) (*the* (*fmlookup x1a* (*fst C*))) *x2a*)›

⟨*proof*⟩

definition *move-is-packed* :: ‹*arena* ⇒ - ⇒ *arena* ⇒ - ⇒ *bool*› **where**
‹*move-is-packed arena*$_o$ N_o *arena* $N \longleftrightarrow$
 $((\sum C \in\# \textit{dom-m } N_o.\ \textit{length } (N_o \propto C) + \textit{header-size } (N_o \propto C)) +$
 $(\sum C \in\# \textit{dom-m } N.\ \textit{length } (N \propto C) + \textit{header-size } (N \propto C)) \leq \textit{length arena}_o)$›

definition *isasat-GC-clauses-prog-copy-wl-entry*
 :: ‹*arena* ⇒ (*nat watcher*) *list list* ⇒ *nat literal* ⇒
  (*arena* × - × -) ⇒ (*arena* × (*arena* × - × -)) *nres*›
where
‹*isasat-GC-clauses-prog-copy-wl-entry* = (λ*N0 W A* (*N′*, *vdm*, *avdm*). *do* {
 ASSERT(*nat-of-lit A* < *length W*);
 ASSERT(*length* (*W* ! *nat-of-lit A*) ≤ *length N0*);
 let le = *length* (*W* ! *nat-of-lit A*);
 (*i*, *N*, *N′*, *vdm*, *avdm*) ← $\textit{WHILE}_T$
  (λ(*i*, *N*, *N′*, *vdm*, *avdm*). *i* < *le*)
  (λ(*i*, *N*, (*N′*, *vdm*, *avdm*)). *do* {
   ASSERT(*i* < *length* (*W* ! *nat-of-lit A*));
   let C = *fst* (*W* ! *nat-of-lit A* ! *i*);
   ASSERT(*arena-is-valid-clause-vdom N C*);
   let st = *arena-status N C*;
   if st ≠ *DELETED then do* {
    ASSERT(*arena-is-valid-clause-idx N C*);
    ASSERT(*length N′* + (*if arena-length N C* > *4 then 5 else 4*) + *arena-length N C* ≤ *length N0*);
    ASSERT(*length N* = *length N0*);
    ASSERT(*length vdm* < *length N0*);
    ASSERT(*length avdm* < *length N0*);
    let D = *length N′* + (*if arena-length N C* > *4 then 5 else 4*);
    N′ ← *fm-mv-clause-to-new-arena C N N′*;
    ASSERT(*mark-garbage-pre* (*N*, *C*));
 RETURN (*i+1*, *extra-information-mark-to-delete N C*, *N′*, *vdm* @ [*D*],
   (*if st* = *LEARNED then avdm* @ [*D*] *else avdm*))
   } *else RETURN* (*i+1*, *N*, (*N′*, *vdm*, *avdm*))
  }) (*0*, *N0*, (*N′*, *vdm*, *avdm*));
 RETURN (*N*, (*N′*, *vdm*, *avdm*))
 })›

definition *isasat-GC-entry* :: ‹-› **where**
‹*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena-old W′* = {((*arena*$_o$, (*arena*, *vdom*, *avdom*)), (N_o, *N*)). *valid-arena arena*$_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set vdom*) ∧ *vdom-m* $\mathcal{A}$ *W′* N_o ⊆ *vdom0* ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom* ∧
 arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena*$_o$ = *length arena-old* ∧
 move-is-packed arena$_o$ N_o *arena N*}›

definition *isasat-GC-refl* :: ‹-› **where**
‹*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena-old* = {((*arena*$_o$, (*arena*, *vdom*, *avdom*), *W*), (N_o, *N*, *W′*)). *valid-arena arena*$_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set vdom*) ∧
 (*W*, *W′*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W′* N_o ⊆ *vdom0* ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom* ∧
 arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena*$_o$ = *length arena-old* ∧
 (∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W′ L*) ≤ *length arena*$_o$) ∧*move-is-packed arena*$_o$ N_o *arena N*}›

lemma *move-is-packed-empty*[*simp*]: ‹*valid-arena arena N vdom* ⟹ *move-is-packed arena N* [] *fmempty*›
 ⟨*proof*⟩

lemma *move-is-packed-append*:
  assumes
    dom: ‹$C \in\#$ *dom-m x1a*› **and**
    E: ‹*length E* = *length* (*x1a* $\propto$ *C*) + *header-size* (*x1a* $\propto$ *C*)› ‹(*fst E'*) = (*x1a* $\propto$ *C*)›
     ‹*n* = *header-size* (*x1a* $\propto$ *C*)› **and**
    valid: ‹*valid-arena x1d x2a D'*› **and**
    packed: ‹*move-is-packed x1c x1a x1d x2a*›
  shows ‹*move-is-packed* (*extra-information-mark-to-delete x1c C*)
        (*fmdrop C x1a*)
        (*x1d* @ *E*)
        (*fmupd* (*length x1d* + *n*) *E' x2a*)›
⟨*proof*⟩

definition *arena-header-size* :: ‹*arena* $\Rightarrow$ *nat* $\Rightarrow$ *nat*› **where**
‹*arena-header-size arena C* = (*if arena-length arena C* > *4* *then 5 else 4*)›

lemma *valid-arena-header-size*:
  ‹*valid-arena arena N vdom* $\Longrightarrow$ $C \in\#$ *dom-m N* $\Longrightarrow$ *arena-header-size arena C* = *header-size* (*N* $\propto$ *C*)›

  ⟨*proof*⟩
lemma *isasat-GC-clauses-prog-copy-wl-entry*:
  assumes ‹*valid-arena arena N vdom0*› **and**
    ‹*valid-arena arena' N'* (*set vdom*)› **and**
    vdom: ‹*vdom-m* $\mathcal{A}$ *W N* $\subseteq$ *vdom0*› **and**
    L: ‹*atm-of A* $\in\#$ $\mathcal{A}$› **and**
    L'-L: ‹(*A'*, *A*) $\in$ *nat-lit-lit-rel*› **and**
    W: ‹(*W'* , *W*) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
    ‹*dom-m N'* = *mset vdom*› ‹*distinct vdom*› **and**
   ‹*arena-is-packed arena' N'*› **and**
    avdom: ‹*mset avdom* $\subseteq\#$ *mset vdom*› **and**
    r: ‹*length arena* = *r*› **and**
    le: ‹$\forall L \in\# \mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W L*) $\leq$ *length arena*› **and**
    packed: ‹*move-is-packed arena N arena' N'*›
  shows ‹*isasat-GC-clauses-prog-copy-wl-entry arena W' A'* (*arena'*, *vdom*, *avdom*)
    $\leq \Downarrow$ (*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena W*)
        (*cdcl-GC-clauses-prog-copy-wl-entry N* (*W A*) *A N'*)›
    (**is** ‹- $\leq \Downarrow$ (*?R*) -›)
⟨*proof*⟩

definition *isasat-GC-clauses-prog-single-wl*
  :: ‹*arena* $\Rightarrow$ (*arena* $\times$ - $\times$ -) $\Rightarrow$ (*nat watcher*) *list list* $\Rightarrow$ *nat* $\Rightarrow$
      (*arena* $\times$ (*arena* $\times$ - $\times$ -) $\times$ (*nat watcher*) *list list*) *nres*›
where
‹*isasat-GC-clauses-prog-single-wl* = (λ*N0 N' WS A. do* {
    let L = *Pos A*; ~~*use phase saving instead*~~
    ASSERT(*nat-of-lit L* < *length WS*);
    ASSERT(*nat-of-lit* ($-L$) < *length WS*);
    (*N*, (*N'*, *vdom*, *avdom*)) $\leftarrow$ *isasat-GC-clauses-prog-copy-wl-entry N0 WS L N'*;
    let WS = *WS*[*nat-of-lit L* := []];
    ASSERT(*length N* = *length N0*);
    (*N*, *N'*) $\leftarrow$ *isasat-GC-clauses-prog-copy-wl-entry N WS* ($-L$) (*N'*, *vdom*, *avdom*);
    let WS = *WS*[*nat-of-lit* ($-L$) := []];
    RETURN (*N*, *N'*, *WS*)
  })›

lemma *isasat-GC-clauses-prog-single-wl*:
  assumes
    ‹$(X, X') \in$ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*› **and**
    X: ‹$X = ($*arena*, (*arena'*, *vdom*, *avdom*), $W)$› ‹$X' = (N, N', W')$› **and**
    L: ‹$A \in\# \mathcal{A}$› **and**
    st: ‹$(A, A') \in Id$› **and** *st'*: ‹*narena* = (*arena'*, *vdom*, *avdom*)› **and**
    ae: ‹*length arena0* = *length arena*› **and**
    le-all: ‹$\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}$. *length* $(W'\ L) \leq$ *length arena*›
  shows ‹*isasat-GC-clauses-prog-single-wl arena narena W A*
    $\leq \Downarrow$ (*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*)
        (*cdcl-GC-clauses-prog-single-wl* $N\ W'\ A'\ N'$)›
    (**is** ‹- $\leq \Downarrow$ *?R* -›)
⟨*proof*⟩

definition *isasat-GC-clauses-prog-wl2* **where**
  ‹*isasat-GC-clauses-prog-wl2* ≡ (λ(*ns* :: (*nat*, *nat*) *vmtf-node list*, *n*) *x0*. *do* {
      (-, *x*) ← $WHILE_T{}^{\lambda(n,\ x).\ length\ (fst\ x)\ =\ length\ (fst\ x0)}$
        (λ(*n*, -). *n* ≠ *None*)
        (λ(*n*, *x*). *do* {
          ASSERT(*n* ≠ *None*);
          let A = *the n*;
          ASSERT(*A* < *length ns*);
          ASSERT(*A* ≤ *uint32-max div 2*);
          x ← (λ($arena_o$, *arena*, *W*). *isasat-GC-clauses-prog-single-wl* $arena_o$ *arena W A*) *x*;
          RETURN (*get-next* ((*ns* ! *A*)), *x*)
        })
        (*n*, *x0*);
      RETURN x
    })›

definition *cdcl-GC-clauses-prog-wl2* **where**
  ‹*cdcl-GC-clauses-prog-wl2* = (λ*N0* $\mathcal{A}0$ *WS*. *do* {
    $\mathcal{A}$ ← *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}0$);
    (-, (*N*, *N'*, *WS*)) ← $WHILE_T{}^{cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl\text{-}inv\ \mathcal{A}\ N0}$
      (λ($\mathcal{B}$, -). $\mathcal{B}$ ≠ {#})
      (λ($\mathcal{B}$, (*N*, *N'*, *WS*)). *do* {
        ASSERT($\mathcal{B}$ ≠ {#});
        A ← *SPEC* (λ*A*. $A \in\# \mathcal{B}$);
        (*N*, *N'*, *WS*) ← *cdcl-GC-clauses-prog-single-wl N WS A N'*;
        RETURN (*remove1-mset A* $\mathcal{B}$, (*N*, *N'*, *WS*))
      })
      ($\mathcal{A}$, (*N0*, *fmempty*, *WS*));
    RETURN (*N*, *N'*, *WS*)
  })›

lemma *WHILEIT-refine-with-invariant-and-break*:
  assumes *R0*: $I'\ x' \Longrightarrow (x,x') \in R$
  assumes *IREF*: $\bigwedge x\ x'.\ [\![\ (x,x') \in R;\ I'\ x'\ ]\!] \Longrightarrow I\ x$
  assumes *COND-REF*: $\bigwedge x\ x'.\ [\![\ (x,x') \in R;\ I\ x;\ I'\ x'\ ]\!] \Longrightarrow b\ x = b'\ x'$
  assumes *STEP-REF*:
    $\bigwedge x\ x'.\ [\![\ (x,x') \in R;\ b\ x;\ b'\ x';\ I\ x;\ I'\ x'\ ]\!] \Longrightarrow f\ x \leq \Downarrow R\ (f'\ x')$
  shows *WHILEIT I b f x* $\leq \Downarrow \{(x, x').\ (x, x') \in R \wedge I\ x \wedge\ I'\ x' \wedge \neg b'\ x'\}$ (*WHILEIT* $I'\ b'\ f'\ x'$)

(**is** ‹- $\leq \Downarrow$ *?R′* -›)
⟨*proof*⟩

lemma *cdcl-GC-clauses-prog-wl-inv-cong-empty*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N* ({#}, *x*) $\Longrightarrow$ *cdcl-GC-clauses-prog-wl-inv* $\mathcal{B}$ *N* ({#}, *x*)›
⟨*proof*⟩

lemma *isasat-GC-clauses-prog-wl2*:
assumes ‹*valid-arena* $arena_o$ N_o *vdom0*› **and**
‹*valid-arena arena N* (*set vdom*)› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W′* N_o $\subseteq$ *vdom0*› **and**
vmtf: ‹((*ns*, *m*, *n*, *lst-As1*, *next-search1*), *to-remove1*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
nempty: ‹$\mathcal{A} \neq$ {#}› **and**
W-W′: ‹(*W*, *W′*) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and** *old*: ‹*old-arena* = []› **and**
le-all: ‹$\forall$ *L* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W′ L*) $\leq$ *length* $arena_o$›
shows
‹*isasat-GC-clauses-prog-wl2* (*ns*, *Some n*) ($arena_o$, (*old-arena*, [], []), *W*)
$\leq \Downarrow$ ({((*$arena_o$′*, (*arena*, *vdom*, *avdom*), *W*), (N_o′, *N*, *W′*)). *valid-arena* $arena_o$′ N_o′ *vdom0* $\wedge$
valid-arena arena N (*set vdom*) $\wedge$
(*W*, *W′*) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) $\wedge$ *vdom-m* $\mathcal{A}$ *W′* N_o′ $\subseteq$ *vdom0* $\wedge$
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ N_o ({#}, N_o′, *N*, *W′*) $\wedge$ *dom-m N* = *mset vdom* $\wedge$ *distinct vdom*
$\wedge$
arena-is-packed arena N $\wedge$ *mset avdom* $\subseteq$# *mset vdom* $\wedge$ *length* $arena_o$′ = *length* $arena_o$})
(*cdcl-GC-clauses-prog-wl2* N_o $\mathcal{A}$ *W′*)›
⟨*proof*⟩

lemma *cdcl-GC-clauses-prog-wl-alt-def*:
‹*cdcl-GC-clauses-prog-wl* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *Q*, *WS*). *do* {
ASSERT(*cdcl-GC-clauses-pre-wl* (*M*, *N0*, *D*, *NE*, *UE*, *Q*, *WS*));
(*N*, *N′*, *WS*) $\leftarrow$ *cdcl-GC-clauses-prog-wl2 N0* (*all-init-atms N0 NE*) *WS*;
RETURN (*M*, *N′*, *D*, *NE*, *UE*, *Q*, *WS*)
})›
⟨*proof*⟩

definition *isasat-GC-clauses-prog-wl* :: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*› **where**
‹*isasat-GC-clauses-prog-wl* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, ((*ns*, *st*, *fst-As*, *lst-As*, *nxt*), *to-remove*), φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
ASSERT(*old-arena* = []);
(*N*, (*N′*, *vdom*, *avdom*), *WS*) $\leftarrow$ *isasat-GC-clauses-prog-wl2* (*ns*, *Some fst-As*) (*N′*, (*old-arena*, *take 0 vdom*, *take 0 avdom*), *W′*);
RETURN (*M′*, *N′*, *D′*, *j*, *WS*, ((*ns*, *st*, *fst-As*, *lst-As*, *nxt*), *to-remove*), φ, *clvls*, *cach*, *lbd*, *outl*, *incr-GC stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*, *take 0 N*)
})›

lemma *length-watched-le″*:
assumes
xb-x′a: ‹(*x1a*, *x1*) $\in$ *twl-st-heur-restart*› **and**
prop-inv: ‹*correct-watching″ x1*›
shows ‹$\forall$ *x2* $\in$# $\mathcal{L}_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) $\leq$ *length* (*get-clauses-wl-heur x1a*)›
⟨*proof*⟩

lemma *isasat-GC-clauses-prog-wl*:
‹(*isasat-GC-clauses-prog-wl*, *cdcl-GC-clauses-prog-wl*) ∈
twl-st-heur-restart $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}⟩*nres-rel*›
(**is** ‹- ∈ *?T* $\rightarrow_f$ -›)
⟨*proof*⟩

definition *cdcl-remap-st* :: ‹*'v twl-st-wl* ⇒ *'v twl-st-wl nres*› **where**
‹*cdcl-remap-st* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *Q'*, *WS'*). (*M'*, *D'*, *NE'*, *UE'*, *Q'*) = (*M*, *D*, *NE*, *UE*, *Q*) ∧
(∃ *m*. *GC-remap*** (*N0*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *N'*)) ∧
0 ∉# *dom-m N'*))›

definition *rewatch-spec* :: ‹*nat twl-st-wl* ⇒ *nat twl-st-wl nres*› **where**
‹*rewatch-spec* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *Q'*, *WS'*). (*M'*, *N'*, *D'*, *NE'*, *UE'*, *Q'*) = (*M*, *N*, *D*, *NE*, *UE*, *Q*) ∧
correct-watching' (*M*, *N'*, *D*, *NE*, *UE*, *Q'*, *WS'*) ∧
blits-in-$\mathcal{L}_{in}$*'* (*M*, *N'*, *D*, *NE*, *UE*, *Q'*, *WS'*)))›

lemma *RES-RES7-RETURN-RES*:
‹*RES A* ≫= (λ(*a*, *b*, *c*, *d*, *e*, *g*, *h*). *RES* (*f a b c d e g h*)) = *RES* (⋃((λ(*a*, *b*, *c*, *d*, *e*, *g*, *h*). *f a b c d e g h*) ‘ *A*))›
⟨*proof*⟩

lemma *cdcl-GC-clauses-wl-D-alt-def*:
‹*cdcl-GC-clauses-wl-D* = (λ*S*. *do* {
ASSERT(*cdcl-GC-clauses-pre-wl-D S*);
let b = *True*;
if b then do {
S ← *cdcl-remap-st S*;
S ← *rewatch-spec S*;
RETURN S
}
else RETURN S})›
⟨*proof*⟩

definition *isasat-GC-clauses-pre-wl-D* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*isasat-GC-clauses-pre-wl-D S* ⟷ (
∃ *T*. (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *cdcl-GC-clauses-pre-wl-D T*
)›

definition *isasat-GC-clauses-wl-D* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*isasat-GC-clauses-wl-D* = (λ*S*. *do* {
ASSERT(*isasat-GC-clauses-pre-wl-D S*);
let b = *True*;
if b then do {
T ← *isasat-GC-clauses-prog-wl S*;
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *length* (*get-clauses-wl-heur S*));
ASSERT(∀ *i* ∈ *set* (*get-vdom T*). *i* < *length* (*get-clauses-wl-heur S*));
U ← *rewatch-heur-st T*;
RETURN U
}

else RETURN S})›

lemma *cdcl-GC-clauses-prog-wl2-st*:
assumes ‹$(T, S) \in$ *state-wl-l None*›
‹*correct-watching''* $T \wedge$ *cdcl-GC-clauses-pre* $S \wedge$
set-mset (*dom-m* (*get-clauses-wl* T)) $\subseteq$ *clauses-pointed-to*
(*Neg* ‘ *set-mset* (*all-init-atms* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T)) $\cup$
Pos ‘ *set-mset* (*all-init-atms* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T)))
(*get-watched-wl* T)› **and**
‹*get-clauses-wl* T = *N0'*›
shows
‹*cdcl-GC-clauses-prog-wl* $T \le$
$\Downarrow$ {((M', N'', D', NE', UE', Q', WS'), (N, N')).
(M', D', NE', UE', Q') = (*get-trail-wl* T, *get-conflict-wl* T, *get-unit-init-clss-wl* T,
get-unit-learned-clss-wl T, *literals-to-update-wl* T) $\wedge$ N'' = N $\wedge$
($\forall L \in \#$*all-init-lits* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T). WS' L = []) $\wedge$
all-init-lits (*get-clauses-wl* T) (*get-unit-init-clss-wl* T) = *all-init-lits* N NE' $\wedge$
($\exists$ *m*. *GC-remap*** (*get-clauses-wl* T, *Map.empty*, *fmempty*)
(*fmempty*, *m*, N))}
(*SPEC*(λ(N'::(*nat*, *'a literal list* $\times$ *bool*) *fmap*, *m*).
GC-remap** (*N0'*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, N') $\wedge$
0 $\notin\#$ *dom-m* N'))›
⟨*proof*⟩

lemma *correct-watching''-clauses-pointed-to*:
assumes
xa-xb: ‹(*xa*, *xb*) $\in$ *state-wl-l None*› **and**
corr: ‹*correct-watching''* *xa*› **and**
pre: ‹*cdcl-GC-clauses-pre* *xb*› **and**
L: ‹*literals-are-*$\mathcal{L}_{in}$*'*
(*all-init-atms* (*get-clauses-wl* *xa*) (*get-unit-init-clss-wl* *xa*)) *xa*›
shows ‹*set-mset* (*dom-m* (*get-clauses-wl* *xa*))
$\subseteq$ *clauses-pointed-to*
(*Neg* ‘
set-mset
(*all-init-atms* (*get-clauses-wl* *xa*) (*get-unit-init-clss-wl* *xa*)) $\cup$
Pos ‘
set-mset
(*all-init-atms* (*get-clauses-wl* *xa*) (*get-unit-init-clss-wl* *xa*)))
(*get-watched-wl* *xa*)›
(**is** ‹- $\subseteq$ *?A*›)
⟨*proof*⟩

abbreviation *isasat-GC-clauses-rel* **where**
‹*isasat-GC-clauses-rel* $y \equiv$ {(S, T). (S, T) $\in$ *twl-st-heur-restart* $\wedge$
($\forall L \in \#$*all-init-lits* (*get-clauses-wl* y) (*get-unit-init-clss-wl* y). *get-watched-wl* T L = [])$\wedge$
all-init-lits-st y = *all-init-lits* (*get-clauses-wl* y) (*get-unit-init-clss-wl* y) $\wedge$
get-trail-wl T = *get-trail-wl* y $\wedge$
get-conflict-wl T = *get-conflict-wl* y $\wedge$
get-unit-init-clss-wl T = *get-unit-init-clss-wl* y $\wedge$
get-unit-learned-clss-wl T = *get-unit-learned-clss-wl* y $\wedge$
($\exists$ *m*. *GC-remap*** (*get-clauses-wl* y, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *get-clauses-wl* T)) $\wedge$
arena-is-packed (*get-clauses-wl-heur* S) (*get-clauses-wl* T)}›

lemma *ref-two-step''*: ‹$R \subseteq R' \Longrightarrow A \le B \Longrightarrow \Downarrow R\ A \le\ \Downarrow R'\ B$›

⟨proof⟩

lemma *isasat-GC-clauses-prog-wl-cdcl-remap-st*:
assumes
‹$(x, y) \in$ *twl-st-heur-restart'''* r› **and**
‹*cdcl-GC-clauses-pre-wl-D* y›
shows ‹*isasat-GC-clauses-prog-wl* $x \le \Downarrow$ (*isasat-GC-clauses-rel* y) (*cdcl-remap-st* y)›
⟨proof⟩

fun *correct-watching'''* :: ‹- ⇒ 'v *twl-st-wl* ⇒ *bool*› **where**
‹*correct-watching'''* $\mathcal{A}$ $(M, N, D, NE, UE, Q, W) \longleftrightarrow$
$(\forall L \in\#$ *all-lits-of-mm* $\mathcal{A}$.
distinct-watched $(W\ L) \wedge$
$(\forall (i, K, b)\in\#$*mset* $(W\ L)$.
$i \in\#$ *dom-m* $N \wedge K \in$ *set* $(N \propto i) \wedge K \neq L \wedge$
correctly-marked-as-binary N $(i, K, b)) \wedge$
fst ‘# *mset* $(W\ L)$ = *clause-to-update* L $(M, N, D, NE, UE, \{\#\}, \{\#\}))$›

declare *correct-watching'''.simps*[*simp del*]

lemma *correct-watching'''-add-clause*:
assumes
corr: ‹*correct-watching'''* $\mathcal{A}$ $((a, aa, CD, ac, ad, Q, b))$› **and**
leC: ‹$2 \le$ *length* C› **and**
i-notin[*simp*]: ‹$i \notin\#$ *dom-m* aa› **and**
dist[*iff*]: ‹$C\ !\ 0 \neq C\ !\ Suc\ 0$›
shows ‹*correct-watching'''* $\mathcal{A}$
$((a,$ *fmupd* i (C, red) aa, CD, ac, ad, Q, b
$(C\ !\ 0 := b\ (C\ !\ 0)$ @ $[(i, C\ !\ Suc\ 0,$ *length* $C = 2)],$
$C\ !\ Suc\ 0 := b\ (C\ !\ Suc\ 0)$ @ $[(i, C\ !\ 0,$ *length* $C = 2)])))$›
⟨proof⟩

lemma *rewatch-correctness*:
assumes *empty*: ‹$\bigwedge L.\ L \in\#$ *all-lits-of-mm* $\mathcal{A} \Longrightarrow W\ L = []$› **and**
H[*dest*]: ‹$\bigwedge x.\ x \in\#$ *dom-m* $N \Longrightarrow$ *distinct* $(N \propto x) \wedge$ *length* $(N \propto x) \ge 2$› **and**
incl: ‹*set-mset* (*all-lits-of-mm* (*mset* ‘# *ran-mf* N)) $\subseteq$ *set-mset* (*all-lits-of-mm* $\mathcal{A}$)›
shows
‹*rewatch* N $W \le SPEC(\lambda W.$ *correct-watching'''* $\mathcal{A}$ $(M, N, C, NE, UE, Q, W))$›
⟨proof⟩

inductive-cases *GC-remapE*: ‹*GC-remap* (a, aa, b) (ab, ac, ba)›
lemma *rtranclp-GC-remap-ran-m-remap*:
‹*GC-remap*** (old, m, new) (old', m', new') $\Longrightarrow C \in\#$ *dom-m* $old \Longrightarrow C \notin\#$ *dom-m* $old' \Longrightarrow$
$m'\ C \neq None \wedge$
fmlookup new' (*the* $(m'\ C)$) = *fmlookup* old C›
⟨proof⟩

lemma *GC-remap-ran-m-exists-earlier*:
‹*GC-remap* (old, m, new) (old', m', new') $\Longrightarrow C \in\#$ *dom-m* $new' \Longrightarrow C \notin\#$ *dom-m* $new \Longrightarrow$
$\exists D.\ m'\ D = Some\ C \wedge D \in\#$ *dom-m* $old \wedge$
fmlookup new' C = *fmlookup* old D›
⟨proof⟩

lemma *rtranclp-GC-remap-ran-m-exists-earlier*:

‹$GC\text{-}remap^{**}$ $(old,\ m,\ new)$ $(old',\ m',\ new')$ $\Longrightarrow C \in\#\ dom\text{-}m\ new' \Longrightarrow C \notin\#\ dom\text{-}m\ new \Longrightarrow$
$\exists D.\ m'\ D = Some\ C \wedge D \in\#\ dom\text{-}m\ old \wedge$
$fmlookup\ new'\ C = fmlookup\ old\ D$›
⟨*proof*⟩

lemma *rewatch-heur-st-correct-watching*:
assumes
pre: ‹*cdcl-GC-clauses-pre-wl-D y*› **and**
S-T: ‹$(S,\ T) \in isasat\text{-}GC\text{-}clauses\text{-}rel\ y$›
shows ‹$rewatch\text{-}heur\text{-}st\ S \leq \Downarrow (twl\text{-}st\text{-}heur\text{-}restart''' (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)))$
$(rewatch\text{-}spec\ T)$›
⟨*proof*⟩

lemma *GC-remap-dom-m-subset*:
‹$GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\text{-}m\ old' \subseteq\#\ dom\text{-}m\ old$›
⟨*proof*⟩

lemma *rtranclp-GC-remap-dom-m-subset*:
‹$rtranclp\ GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\text{-}m\ old' \subseteq\#\ dom\text{-}m\ old$›
⟨*proof*⟩

lemma *GC-remap-mapping-unchanged*:
‹$GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow C \in dom\ m \Longrightarrow m'\ C = m\ C$›
⟨*proof*⟩

lemma *rtranclp-GC-remap-mapping-unchanged*:
‹$GC\text{-}remap^{**}\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow C \in dom\ m \Longrightarrow m'\ C = m\ C$›
⟨*proof*⟩

lemma *GC-remap-mapping-dom-extended*:
‹$GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\ m' = dom\ m \cup set\text{-}mset\ (dom\text{-}m\ old - dom\text{-}m\ old')$›
⟨*proof*⟩

lemma *rtranclp-GC-remap-mapping-dom-extended*:
‹$GC\text{-}remap^{**}\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\ m' = dom\ m \cup set\text{-}mset\ (dom\text{-}m\ old - dom\text{-}m\ old')$›
⟨*proof*⟩

lemma *GC-remap-dom-m*:
‹$GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\text{-}m\ new' = dom\text{-}m\ new + the\ `\#\ m'\ `\#\ (dom\text{-}m\ old - dom\text{-}m\ old')$›
⟨*proof*⟩

lemma *rtranclp-GC-remap-dom-m*:
‹$rtranclp\ GC\text{-}remap\ (old,\ m,\ new)\ (old',\ m',\ new') \Longrightarrow dom\text{-}m\ new' = dom\text{-}m\ new + the\ `\#\ m'\ `\#\ (dom\text{-}m\ old - dom\text{-}m\ old')$›
⟨*proof*⟩

lemma *isasat-GC-clauses-rel-packed-le*:
assumes
xy: ‹$(x,\ y) \in twl\text{-}st\text{-}heur\text{-}restart'''\ r$› **and**
ST: ‹$(S,\ T) \in isasat\text{-}GC\text{-}clauses\text{-}rel\ y$›
shows ‹$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x)$› **and**
‹$\forall C \in set\ (get\text{-}vdom\ S).\ C < length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x)$›

⟨proof⟩

lemma *isasat-GC-clauses-wl-D*:
‹(*isasat-GC-clauses-wl-D*, *cdcl-GC-clauses-wl-D*)
$\in$ *twl-st-heur-restart′′′ r* $\rightarrow_f$ ⟨*twl-st-heur-restart′′′′ r*⟩*nres-rel*›
⟨proof⟩

definition *cdcl-twl-full-restart-wl-D-GC-heur-prog* **where**
‹*cdcl-twl-full-restart-wl-D-GC-heur-prog S0 = do* {
S ← *do* {
if count-decided-st-heur S0 > 0
then do {
S ← *find-decomp-wl-st-int 0 S0*;
empty-Q S
} *else RETURN S0*
};
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
T ← *remove-one-annot-true-clause-imp-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S0*));
U ← *mark-to-delete-clauses-wl-D-heur T*;
ASSERT(*length* (*get-clauses-wl-heur U*) = *length* (*get-clauses-wl-heur S0*));
V ← *isasat-GC-clauses-wl-D U*;
RETURN V
}›

lemma
cdcl-twl-full-restart-wl-GC-prog-pre-heur:
‹*cdcl-twl-full-restart-wl-GC-prog-pre T* ⟹
(*S*, *T*) ∈ *twl-st-heur′′′ r* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart-ana r*› (**is** ‹- ⟹ - *?A*›) **and**
cdcl-twl-full-restart-wl-D-GC-prog-post-heur:
‹*cdcl-twl-full-restart-wl-D-GC-prog-post S0 T* ⟹
(*S*, *T*) ∈ *twl-st-heur* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart*› (**is** ‹- ⟹ - *?B*›)
⟨proof⟩

lemma *cdcl-twl-full-restart-wl-D-GC-heur-prog*:
‹(*cdcl-twl-full-restart-wl-D-GC-heur-prog*, *cdcl-twl-full-restart-wl-D-GC-prog*) ∈
twl-st-heur′′′ r $\rightarrow_f$ ⟨*twl-st-heur′′′′ r*⟩*nres-rel*›
⟨proof⟩

definition *restart-prog-wl-D-heur*
:: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool* ⇒ (*twl-st-wl-heur* × *nat*) *nres*
where
‹*restart-prog-wl-D-heur S n brk = do* {
b ← *restart-required-heur S n*;
b2 ← *GC-required-heur S n*;
if ¬*brk* ∧ *b* ∧ *b2*
then do {
T ← *cdcl-twl-full-restart-wl-D-GC-heur-prog S*;
RETURN (*T*, *n+1*)
}
else if ¬*brk* ∧ *b*
then do {
T ← *cdcl-twl-restart-wl-heur S*;

        RETURN (T, n+1)
      }
    else RETURN (S, n)
  }›

lemma *restart-required-heur-restart-required-wl*:
  ‹(*uncurry restart-required-heur*, *uncurry restart-required-wl*) ∈
    twl-st-heur $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
    ⟨*proof*⟩

lemma *restart-required-heur-restart-required-wl0*:
  ‹(*uncurry restart-required-heur*, *uncurry restart-required-wl*) ∈
    twl-st-heur''' *r* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
    ⟨*proof*⟩

lemma *restart-prog-wl-D-heur-restart-prog-wl-D*:
  ‹(*uncurry2 restart-prog-wl-D-heur*, *uncurry2 restart-prog-wl-D*) ∈
    twl-st-heur''' *r* $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\rightarrow_f$ ⟨*twl-st-heur''''* *r* $\times_f$ *nat-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *restart-prog-wl-D-heur-restart-prog-wl-D2*:
  ‹(*uncurry2 restart-prog-wl-D-heur*, *uncurry2 restart-prog-wl-D*) ∈
  twl-st-heur $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\rightarrow_f$ ⟨*twl-st-heur* $\times_f$ *nat-rel*⟩*nres-rel*›
  ⟨*proof*⟩

definition *isasat-trail-nth-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isasat-trail-nth-st S i* = *isa-trail-nth* (*get-trail-wl-heur S*) *i*›

lemma *isasat-trail-nth-st-alt-def*:
  ‹*isasat-trail-nth-st* = (λ(*M*, -) *i*. *isa-trail-nth M i*)›
  ⟨*proof*⟩

definition *get-the-propagation-reason-pol-st* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-the-propagation-reason-pol-st S i* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *i*›

lemma *get-the-propagation-reason-pol-st-alt-def*:
  ‹*get-the-propagation-reason-pol-st* = (λ(*M*, -) *i*. *get-the-propagation-reason-pol M i*)›
  ⟨*proof*⟩

definition *isasat-length-trail-st* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*isasat-length-trail-st S* = *isa-length-trail* (*get-trail-wl-heur S*)›

lemma *isasat-length-trail-st-alt-def*:
  ‹*isasat-length-trail-st* = (λ(*M*, -). *isa-length-trail M*)›
  ⟨*proof*⟩

definition *get-pos-of-level-in-trail-imp-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat nres*› **where**
‹*get-pos-of-level-in-trail-imp-st S* = *get-pos-of-level-in-trail-imp* (*get-trail-wl-heur S*)›

lemma *get-pos-of-level-in-trail-imp-alt-def*:
  ‹*get-pos-of-level-in-trail-imp-st* = (λ(*M*, -). *get-pos-of-level-in-trail-imp M*)›
  ⟨*proof*⟩

definition *rewatch-heur-st-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**

$\langle$*rewatch-heur-st-pre S* $\longleftrightarrow$ ($\forall$ *i* < *length* (*get-vdom S*). *get-vdom S* ! *i* $\leq$ *uint64-max*)$\rangle$

lemma *isasat-GC-clauses-wl-D-rewatch-pre*:
  assumes
    $\langle$*length* (*get-clauses-wl-heur x*) $\leq$ *uint64-max*$\rangle$ **and**
    $\langle$*length* (*get-clauses-wl-heur xc*) $\leq$ *length* (*get-clauses-wl-heur x*)$\rangle$ **and**
    $\langle\forall$ *i* $\in$ *set* (*get-vdom xc*). *i* $\leq$ *length* (*get-clauses-wl-heur x*)$\rangle$
  shows $\langle$*rewatch-heur-st-pre xc*$\rangle$
  $\langle$*proof*$\rangle$

lemma *li-uint32-maxdiv2-le-unit32-max*: $\langle$*a* $\leq$ *uint32-max div 2* + *1* $\Longrightarrow$ *a* $\leq$ *uint32-max*$\rangle$
  $\langle$*proof*$\rangle$

end
theory *IsaSAT-Restart-Heuristics-SML*
  imports *IsaSAT-Restart-Heuristics IsaSAT-Setup-SML*
    IsaSAT-VMTF-SML
begin

lemma *clause-score-ordering-hnr*[*sepref-fr-rules*]:
  $\langle$(*uncurry* (*return oo clause-score-ordering*), *uncurry* (*RETURN oo clause-score-ordering*)) $\in$
    (*uint32-nat-assn* $*a$ *uint32-nat-assn*)^k $*_a$ (*uint32-nat-assn* $*a$ *uint32-nat-assn*)^k $\rightarrow_a$ *bool-assn*$\rangle$
  $\langle$*proof*$\rangle$

sepref-definition *get-slow-ema-heur-fast-code*
  is $\langle$*RETURN o get-slow-ema-heur*$\rangle$
  :: $\langle$*isasat-bounded-assn*^{k} $\rightarrow_a$ *ema-assn*$\rangle$
  $\langle$*proof*$\rangle$

sepref-definition *get-slow-ema-heur-slow-code*
  is $\langle$*RETURN o get-slow-ema-heur*$\rangle$
  :: $\langle$*isasat-unbounded-assn*^{k} $\rightarrow_a$ *ema-assn*$\rangle$
  $\langle$*proof*$\rangle$

declare *get-slow-ema-heur-fast-code.refine*[*sepref-fr-rules*]
  get-slow-ema-heur-slow-code.refine[*sepref-fr-rules*]

sepref-definition *get-fast-ema-heur-fast-code*
  is $\langle$*RETURN o get-fast-ema-heur*$\rangle$
  :: $\langle$*isasat-bounded-assn*^{k} $\rightarrow_a$ *ema-assn*$\rangle$
  $\langle$*proof*$\rangle$

sepref-definition *get-fast-ema-heur-slow-code*
  is $\langle$*RETURN o get-fast-ema-heur*$\rangle$
  :: $\langle$*isasat-unbounded-assn*^{k} $\rightarrow_a$ *ema-assn*$\rangle$
  $\langle$*proof*$\rangle$

declare *get-fast-ema-heur-slow-code.refine*[*sepref-fr-rules*]
  get-fast-ema-heur-fast-code.refine[*sepref-fr-rules*]

sepref-definition *get-conflict-count-since-last-restart-heur-fast-code*
  is $\langle$*RETURN o get-conflict-count-since-last-restart-heur*$\rangle$
  :: $\langle$*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint64-assn*$\rangle$

⟨*proof*⟩

sepref-definition *get-conflict-count-since-last-restart-heur-slow-code*
is ⟨*RETURN o get-conflict-count-since-last-restart-heur*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint64\text{-}assn$⟩
⟨*proof*⟩

declare *get-conflict-count-since-last-restart-heur-fast-code.refine*[*sepref-fr-rules*]
get-conflict-count-since-last-restart-heur-slow-code.refine[*sepref-fr-rules*]

sepref-definition *get-learned-count-fast-code*
is ⟨*RETURN o get-learned-count*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$⟩
⟨*proof*⟩

sepref-definition *get-learned-count-slow-code*
is ⟨*RETURN o get-learned-count*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a nat\text{-}assn$⟩
⟨*proof*⟩

declare *get-learned-count-fast-code.refine*[*sepref-fr-rules*]
get-learned-count-slow-code.refine[*sepref-fr-rules*]

sepref-definition *find-local-restart-target-level-code*
is ⟨*uncurry find-local-restart-target-level-int*⟩
:: ⟨$trail\text{-}pol\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
⟨*proof*⟩

sepref-definition *find-local-restart-target-level-fast-code*
is ⟨*uncurry find-local-restart-target-level-int*⟩
:: ⟨$trail\text{-}pol\text{-}fast\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
⟨*proof*⟩

declare *find-local-restart-target-level-code.refine*[*sepref-fr-rules*]
find-local-restart-target-level-fast-code.refine[*sepref-fr-rules*]

sepref-definition *incr-restart-stat-slow-code*
is ⟨*incr-restart-stat*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
⟨*proof*⟩

sepref-register *incr-restart-stat*

sepref-definition *incr-restart-stat-fast-code*
is ⟨*incr-restart-stat*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$⟩
⟨*proof*⟩

declare *incr-restart-stat-slow-code.refine*[*sepref-fr-rules*]
incr-restart-stat-fast-code.refine[*sepref-fr-rules*]

sepref-definition *incr-lrestart-stat-slow-code*

is ‹*incr-lrestart-stat*›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-register *incr-lrestart-stat*

sepref-definition *incr-lrestart-stat-fast-code*
is ‹*incr-lrestart-stat*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *incr-lrestart-stat-slow-code.refine*[*sepref-fr-rules*]
incr-lrestart-stat-fast-code.refine[*sepref-fr-rules*]

sepref-definition *find-local-restart-target-level-st-code*
is ‹*find-local-restart-target-level-st*›
:: ‹*isasat-unbounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *find-local-restart-target-level-st-fast-code*
is ‹*find-local-restart-target-level-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

declare *find-local-restart-target-level-st-code.refine*[*sepref-fr-rules*]
find-local-restart-target-level-st-fast-code.refine[*sepref-fr-rules*]

sepref-definition *empty-Q-code*
is ‹*empty-Q*›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *empty-Q-fast-code*
is ‹*empty-Q*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *empty-Q-code.refine*[*sepref-fr-rules*]
empty-Q-fast-code.refine[*sepref-fr-rules*]

sepref-register *cdcl-twl-local-restart-wl-D-heur*
empty-Q find-decomp-wl-st-int

sepref-definition *cdcl-twl-local-restart-wl-D-heur-code*
is ‹*cdcl-twl-local-restart-wl-D-heur*›
:: ‹*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *cdcl-twl-local-restart-wl-D-heur-fast-code*
is ‹*cdcl-twl-local-restart-wl-D-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *cdcl-twl-local-restart-wl-D-heur-code.refine*[*sepref-fr-rules*]
cdcl-twl-local-restart-wl-D-heur-fast-code.refine[*sepref-fr-rules*]

lemma *five-uint64*[*sepref-fr-rules*]:
‹(*uncurry0* (*return five-uint64*), *uncurry0* (*RETURN five-uint64*))
∈ $unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

definition *two-uint64* :: ‹*uint64*› **where**
‹*two-uint64* = *2*›

lemma *two-uint64*[*sepref-fr-rules*]:
‹(*uncurry0* (*return two-uint64*), *uncurry0* (*RETURN two-uint64*))
∈ $unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

sepref-register *upper-restart-bound-not-reached*
sepref-definition *upper-restart-bound-not-reached-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *upper-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *upper-restart-bound-not-reached-impl.refine*[*sepref-fr-rules*]
upper-restart-bound-not-reached-fast-impl.refine[*sepref-fr-rules*]

sepref-register *lower-restart-bound-not-reached*
sepref-definition *lower-restart-bound-not-reached-impl*
is ‹(*RETURN o lower-restart-bound-not-reached*)›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *lower-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o lower-restart-bound-not-reached*)›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *lower-restart-bound-not-reached-impl.refine*[*sepref-fr-rules*]
lower-restart-bound-not-reached-fast-impl.refine[*sepref-fr-rules*]

sepref-register *clause-score-extract*

sepref-definition (**in** −) *clause-score-extract-code*
is ‹*uncurry* (*RETURN oo clause-score-extract*)›
:: ‹[*uncurry valid-sort-clause-score-pre-at*]$_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *clause-score-extract-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-get-clause-LBD-code2*
is ‹*uncurry isa-get-clause-LBD*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

lemma *isa-get-clause-LBD-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-clause-LBD-code2*, *uncurry* (*RETURN* ∘∘ *get-clause-LBD*))
∈ $[uncurry\ get\text{-}clause\text{-}LBD\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩
sepref-definition *isa-arena-act-code2*
is ‹*uncurry isa-arena-act*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
⟨*proof*⟩

lemma *isa-arena-act-code2*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-act-code2*, *uncurry* (*RETURN* ∘∘ *arena-act*))
∈ $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

find-theorems *arena-act*
thm *isa-arena-act-code*
sepref-definition (**in** −) *clause-score-extract-fast-code*
is ‹*uncurry* (*RETURN oo clause-score-extract*)›
:: ‹$[uncurry\ valid\text{-}sort\text{-}clause\text{-}score\text{-}pre\text{-}at]_a$
$arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn *a\ uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *clause-score-extract-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *partition-main-clause-code*
is ‹*uncurry3 partition-main-clause*›
:: ‹$[\lambda(((arena, i), j), vdom).\ valid\text{-}sort\text{-}clause\text{-}score\text{-}pre\ arena\ vdom]_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a vdom\text{-}assn^d \rightarrow vdom\text{-}assn *a\ nat\text{-}assn$›
⟨*proof*⟩

sepref-definition (**in** −) *partition-main-clause-fast-code*
is ‹*uncurry3 partition-main-clause*›
:: ‹$[\lambda(((arena, i), j), vdom).\ length\ vdom \leq uint64\text{-}max \wedge valid\text{-}sort\text{-}clause\text{-}score\text{-}pre\ arena\ vdom]_a$
$arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a vdom\text{-}fast\text{-}assn^d \rightarrow vdom\text{-}fast\text{-}assn *a$
$uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-register *partition-main-clause-code*
declare *partition-main-clause-code.refine*[*sepref-fr-rules*]
partition-main-clause-fast-code.refine[*sepref-fr-rules*]

sepref-definition (**in** −) *partition-clause-code*
is ‹*uncurry3 partition-clause*›
:: ‹$[\lambda(((arena, i), j), vdom).\ valid\text{-}sort\text{-}clause\text{-}score\text{-}pre\ arena\ vdom]_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a vdom\text{-}assn^d \rightarrow vdom\text{-}assn *a\ nat\text{-}assn$›
⟨*proof*⟩

lemma *div2-hnr*[*sepref-fr-rules*]: ‹(*return o* (*λn. n >> 1*), *RETURN o div2*) ∈ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition (**in** −) *partition-clause-fast-code*
is ‹*uncurry3 partition-clause*›
:: ‹[λ(((*arena*, *i*), *j*), *vdom*). *length vdom* ≤ *uint64-max* ∧ *valid-sort-clause-score-pre arena vdom*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *vdom-fast-assn*^{d} → *vdom-fast-assn* ∗*a* *uint64-nat-assn*›
⟨*proof*⟩

declare *partition-clause-code.refine*[*sepref-fr-rules*]
partition-clause-fast-code.refine[*sepref-fr-rules*]

sepref-definition (**in** −) *sort-clauses-by-score-code*
is ‹*uncurry quicksort-clauses-by-score*›
:: ‹[*uncurry valid-sort-clause-score-pre*]$_a$
arena-assn^k $*_a$ *vdom-assn*^{d} → *vdom-assn*›
⟨*proof*⟩

lemma *minus-uint64-safe*:
‹(*uncurry* (*return oo safe-minus*), *uncurry* (*RETURN oo* (−))) ∈ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition (**in** −) *sort-clauses-by-score-fast-code*
is ‹*uncurry quicksort-clauses-by-score*›
:: ‹[λ(*arena*, *vdom*). *length vdom* ≤ *uint64-max* ∧ *valid-sort-clause-score-pre arena vdom*]$_a$
arena-fast-assn^k $*_a$ *vdom-fast-assn*^{d} → *vdom-fast-assn*›
⟨*proof*⟩

lemma *arl64-take*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo arl64-take*), *uncurry* (*RETURN oo take*)) ∈
[λ(*n*, *xs*). *n* ≤ *length xs*]$_a$ *uint64-nat-assn*^{k} $*_a$ (*arl64-assn R*)^d → *arl64-assn R*›
⟨*proof*⟩

sepref-register *remove-deleted-clauses-from-avdom*
sepref-definition *remove-deleted-clauses-from-avdom-fast-code*
is ‹*uncurry isa-remove-deleted-clauses-from-avdom*›
:: ‹[λ(*N*, *vdom*). *length vdom* ≤ *uint64-max*]$_a$ *arena-fast-assn*^{k} $*_a$ *vdom-fast-assn*^{d} → *vdom-fast-assn*›
⟨*proof*⟩

sepref-definition *remove-deleted-clauses-from-avdom-code*
is ‹*uncurry isa-remove-deleted-clauses-from-avdom*›
:: ‹*arena-assn*^{k} $*_a$ *vdom-assn*^{d} $\rightarrow_a$ *vdom-assn*›
⟨*proof*⟩

declare *remove-deleted-clauses-from-avdom-fast-code.refine*[*sepref-fr-rules*]
remove-deleted-clauses-from-avdom-code.refine[*sepref-fr-rules*]

sepref-definition *sort-vdom-heur-code*
is ‹*sort-vdom-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *sort-vdom-heur-fast-code*
is ‹*sort-vdom-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *sort-vdom-heur-code.refine*[*sepref-fr-rules*]
sort-vdom-heur-fast-code.refine[*sepref-fr-rules*]

sepref-definition *opts-restart-st-code*
is ‹*RETURN o opts-restart-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *opts-restart-st-fast-code*
is ‹*RETURN o opts-restart-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *opts-restart-st-code.refine*[*sepref-fr-rules*]
opts-restart-st-fast-code.refine[*sepref-fr-rules*]

sepref-definition *opts-reduction-st-code*
is ‹*RETURN o opts-reduction-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *opts-reduction-st-fast-code*
is ‹*RETURN o opts-reduction-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *opts-reduction-st-code.refine*[*sepref-fr-rules*]
opts-reduction-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *opts-reduction-st opts-restart-st*

sepref-register *max-restart-decision-lvl*

lemma *minimum-number-between-restarts*[*sepref-fr-rules*]:
‹(*uncurry0* (*return minimum-number-between-restarts*), *uncurry0* (*RETURN minimum-number-between-restarts*))
$\in$ $unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

lemma *max-restart-decision-lvl-code-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return max-restart-decision-lvl-code*), *uncurry0* (*RETURN max-restart-decision-lvl*)) $\in$
$unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return GC-EVERY*), *uncurry0* (*RETURN GC-EVERY*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint64-assn*›
⟨*proof*⟩

lemma (**in** −) *MINIMUM-DELETION-LBD-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 3*), *uncurry0* (*RETURN MINIMUM-DELETION-LBD*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-definition *restart-required-heur-fast-code*
is ‹*uncurry restart-required-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *restart-required-heur-slow-code*
is ‹*uncurry restart-required-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *restart-required-heur-fast-code.refine*[*sepref-fr-rules*]
restart-required-heur-slow-code.refine[*sepref-fr-rules*]

sepref-definition *get-reductions-count-fast-code*
is ‹*RETURN o get-reductions-count*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

sepref-definition *get-reductions-count-code*
is ‹*RETURN o get-reductions-count*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

sepref-register *get-reductions-count*
declare *get-reductions-count-fast-code.refine*[*sepref-fr-rules*]
declare *get-reductions-count-code.refine*[*sepref-fr-rules*]

sepref-definition *GC-required-heur-fast-code*
is ‹*uncurry GC-required-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

sepref-definition *GC-required-heur-slow-code*
is ‹*uncurry GC-required-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
⟨*proof*⟩

declare *GC-required-heur-fast-code.refine*[*sepref-fr-rules*]
GC-required-heur-slow-code.refine[*sepref-fr-rules*]

sepref-register *isa-trail-nth*

sepref-register *isasat-trail-nth-st*

sepref-definition *isasat-trail-nth-st-code*
is ‹*uncurry isasat-trail-nth-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-definition *isasat-trail-nth-st-slow-code*
is ‹*uncurry isasat-trail-nth-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

declare *isasat-trail-nth-st-code.refine*[*sepref-fr-rules*]
isasat-trail-nth-st-slow-code.refine[*sepref-fr-rules*]

sepref-register *get-the-propagation-reason-pol-st*

sepref-definition *get-the-propagation-reason-pol-st-code*
is ‹*uncurry get-the-propagation-reason-pol-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a option\text{-}assn\ uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *get-the-propagation-reason-pol-st-slow-code*
is ‹*uncurry get-the-propagation-reason-pol-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a option\text{-}assn\ nat\text{-}assn$›
⟨*proof*⟩

declare *get-the-propagation-reason-pol-st-code.refine*[*sepref-fr-rules*]
get-the-propagation-reason-pol-st-slow-code.refine[*sepref-fr-rules*]

sepref-register *isasat-replace-annot-in-trail*
sepref-definition *isasat-replace-annot-in-trail-code*
is ‹*uncurry2 isasat-replace-annot-in-trail*›
:: ‹$unat\text{-}lit\text{-}assn^k *_a (uint64\text{-}nat\text{-}assn)^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *isasat-replace-annot-in-trail-slow-code*
is ‹*uncurry2 isasat-replace-annot-in-trail*›
:: ‹$unat\text{-}lit\text{-}assn^k *_a (nat\text{-}assn)^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *mark-garbage-fast-code*
is ‹*uncurry mark-garbage*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a arl64\text{-}assn\ uint32\text{-}assn$›
⟨*proof*⟩

lemma *mark-garbage-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry mark-garbage-fast-code*, *uncurry* (*RETURN oo extra-information-mark-to-delete*))
$\in [mark\text{-}garbage\text{-}pre]_a\ \ arena\text{-}fast\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$›
⟨*proof*⟩

context
  notes [*fcomp-norm-unfold*] = *arl64-assn-def*[*symmetric*] *arl64-assn-comp′*
  notes [*intro*!] = *hfrefI hn-refineI*[*THEN hn-refine-preI*]
  notes [*simp*] = *pure-def hn-ctxt-def invalid-assn-def*
begin
definition *arl64-set-nat* :: *′a::heap array-list64* ⇒ *nat* ⇒ *′a* ⇒ *′a array-list64 Heap* **where**
  arl64-set-nat ≡ λ(*a*,*n*) *i x*. *do* { *a* ← *Array.upd i x a*; *return* (*a*,*n*)}

  lemma *arl64-set-hnr-aux*: (*uncurry2 arl64-set-nat*,*uncurry2* (*RETURN ooo op-list-set*)) ∈ $[\lambda((l,i),\text{-}).\ i{<}length\ l]_a\ (is\text{-}array\text{-}list64^d *_a nat\text{-}assn^k *_a id\text{-}assn^k) \to is\text{-}array\text{-}list64$
    ⟨*proof*⟩
  sepref-decl-impl *arl64-set-nat*: *arl64-set-hnr-aux* ⟨*proof*⟩

end

sepref-definition *mark-garbage-fast-code2*
  is ‹*uncurry mark-garbage*›
  :: $\langle(arl64\text{-}assn\ uint32\text{-}assn)^d *_a nat\text{-}assn^k \to_a arl64\text{-}assn\ uint32\text{-}assn\rangle$
  ⟨*proof*⟩

lemma *mark-garbage-fast-hnr2*[*sepref-fr-rules*]:
  ‹(*uncurry mark-garbage-fast-code2*, *uncurry* (*RETURN oo extra-information-mark-to-delete*))
  $\in [mark\text{-}garbage\text{-}pre]_a\ \ arena\text{-}fast\text{-}assn^d *_a nat\text{-}assn^k \to arena\text{-}fast\text{-}assn\rangle$
  ⟨*proof*⟩

sepref-register *mark-garbage-heur2*
sepref-definition *mark-garbage-heur2-code*
  is ‹*uncurry mark-garbage-heur2*›
  :: ‹[λ(*C*, *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*]$_a$
     $uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn\rangle$
  ⟨*proof*⟩

sepref-definition *mark-garbage-heur2-slow-code*
  is ‹*uncurry mark-garbage-heur2*›
  :: ‹[λ(*C*, *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*]$_a$
     $nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \to isasat\text{-}unbounded\text{-}assn\rangle$
  ⟨*proof*⟩

declare *isasat-replace-annot-in-trail-code.refine*[*sepref-fr-rules*]
  isasat-replace-annot-in-trail-slow-code.refine[*sepref-fr-rules*]
  mark-garbage-heur2-code.refine[*sepref-fr-rules*]
  mark-garbage-heur2-slow-code.refine[*sepref-fr-rules*]

sepref-register *remove-one-annot-true-clause-one-imp-wl-D-heur*

sepref-definition *remove-one-annot-true-clause-one-imp-wl-D-heur-code*
  is ‹*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*›
  :: $\langle uint32\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \to_a uint32\text{-}nat\text{-}assn *a\ isasat\text{-}bounded\text{-}assn\rangle$
  ⟨*proof*⟩

sepref-definition *remove-one-annot-true-clause-one-imp-wl-D-heur-slow-code*
  is ‹*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*›

:: ‹$uint32\text{-}nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a uint32\text{-}nat\text{-}assn *a\ isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *remove-one-annot-true-clause-one-imp-wl-D-heur-slow-code.refine*[*sepref-fr-rules*]
remove-one-annot-true-clause-one-imp-wl-D-heur-code.refine[*sepref-fr-rules*]

sepref-register *isasat-length-trail-st*

sepref-definition *isasat-length-trail-st-code*
is ‹*RETURN o isasat-length-trail-st*›
:: ‹$[isa\text{-}length\text{-}trail\text{-}pre\ o\ get\text{-}trail\text{-}wl\text{-}heur]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *isasat-length-trail-st-slow-code*
is ‹*RETURN o isasat-length-trail-st*›
:: ‹$[isa\text{-}length\text{-}trail\text{-}pre\ o\ get\text{-}trail\text{-}wl\text{-}heur]_a\ isasat\text{-}unbounded\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *isasat-length-trail-st-slow-code.refine*[*sepref-fr-rules*]
isasat-length-trail-st-code.refine[*sepref-fr-rules*]

sepref-register *get-pos-of-level-in-trail-imp-st*

sepref-definition *get-pos-of-level-in-trail-imp-st-code*
is ‹*uncurry get-pos-of-level-in-trail-imp-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *get-pos-of-level-in-trail-imp-st-slow-code*
is ‹*uncurry get-pos-of-level-in-trail-imp-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

declare *get-pos-of-level-in-trail-imp-st-slow-code.refine*[*sepref-fr-rules*]
get-pos-of-level-in-trail-imp-st-code.refine[*sepref-fr-rules*]

sepref-register *remove-one-annot-true-clause-imp-wl-D-heur*

sepref-definition *remove-one-annot-true-clause-imp-wl-D-heur-code*
is ‹*remove-one-annot-true-clause-imp-wl-D-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *remove-one-annot-true-clause-imp-wl-D-heur-slow-code*
is ‹*remove-one-annot-true-clause-imp-wl-D-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *remove-one-annot-true-clause-imp-wl-D-heur-code.refine*[*sepref-fr-rules*]
remove-one-annot-true-clause-imp-wl-D-heur-slow-code.refine[*sepref-fr-rules*]

declare *fm-mv-clause-to-new-arena-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *isasat-GC-clauses-prog-copy-wl-entry-code*
is ‹*uncurry3 isasat-GC-clauses-prog-copy-wl-entry*›
:: ‹$[\lambda(((N, -), -), -).\ length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^d *_a watchlist\text{-}fast\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
$(arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn)^d \rightarrow$
$(arena\text{-}fast\text{-}assn *a\ (arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn))$›
⟨*proof*⟩

sepref-definition *isasat-GC-clauses-prog-copy-wl-entry-slow-code*
is ‹*uncurry3 isasat-GC-clauses-prog-copy-wl-entry*›
:: ‹$arena\text{-}assn^d *_a watchlist\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn)^d \rightarrow_a$
$(arena\text{-}assn *a\ (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn))$›
⟨*proof*⟩

sepref-register *isasat-GC-clauses-prog-copy-wl-entry*
declare *isasat-GC-clauses-prog-copy-wl-entry-code.refine*[*sepref-fr-rules*]
isasat-GC-clauses-prog-copy-wl-entry-slow-code.refine[*sepref-fr-rules*]

lemma *shorten-take-ll-0*: ‹$shorten\text{-}take\text{-}ll\ L\ 0\ W = W[L := []]$›
⟨*proof*⟩

lemma *length-shorten-take-ll*[*simp*]: ‹$length\ (shorten\text{-}take\text{-}ll\ a\ j\ W) = length\ W$›
⟨*proof*⟩

sepref-definition *isasat-GC-clauses-prog-single-wl-code*
is ‹*uncurry3 isasat-GC-clauses-prog-single-wl*›
:: ‹$[\lambda(((N, -), -), A).\ A \leq uint32\text{-}max\ div\ 2 \wedge length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^d *_a (arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn)^d *_a watchlist\text{-}fast\text{-}assn^d$
$*_a uint32\text{-}nat\text{-}assn^k \rightarrow$
$(arena\text{-}fast\text{-}assn *a\ (arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn) *a\ watchlist\text{-}fast\text{-}assn)$›
⟨*proof*⟩

sepref-definition *isasat-GC-clauses-prog-single-wl-slow-code*
is ‹*uncurry3 isasat-GC-clauses-prog-single-wl*›
:: ‹$[\lambda(((-, -), -), A).\ A \leq uint32\text{-}max\ div\ 2]_a$
$arena\text{-}assn^d *_a (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn)^d *_a watchlist\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow$
$(arena\text{-}assn *a\ (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn) *a\ watchlist\text{-}assn)$›
⟨*proof*⟩

declare *isasat-GC-clauses-prog-single-wl-code.refine*[*sepref-fr-rules*]
isasat-GC-clauses-prog-single-wl-slow-code.refine[*sepref-fr-rules*]

definition *isasat-GC-clauses-prog-wl2'* **where**
‹$isasat\text{-}GC\text{-}clauses\text{-}prog\text{-}wl2'\ ns\ fst' = (isasat\text{-}GC\text{-}clauses\text{-}prog\text{-}wl2\ (ns, fst'))$›

sepref-register *isasat-GC-clauses-prog-wl2*
sepref-definition *isasat-GC-clauses-prog-wl2-code*
is ‹*uncurry2 isasat-GC-clauses-prog-wl2'*›
:: ‹$[\lambda((-, -), (N, -)).\ length\ N \leq uint64\text{-}max]_a$
$(array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a (option\text{-}assn\ uint32\text{-}nat\text{-}assn)^k *_a$
$(arena\text{-}fast\text{-}assn *a\ (arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn) *a\ watchlist\text{-}fast\text{-}assn)^d$
$\rightarrow$
$(arena\text{-}fast\text{-}assn *a\ (arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn) *a\ watchlist\text{-}fast\text{-}assn)$›

⟨*proof*⟩

sepref-definition *isasat-GC-clauses-prog-wl2-slow-code*
is ‹*uncurry2 isasat-GC-clauses-prog-wl2'*›
:: ‹$(array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a (option\text{-}assn\ uint32\text{-}nat\text{-}assn)^k *_a$
$(arena\text{-}assn *a (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn) *a\ watchlist\text{-}assn)^d \rightarrow_a$
$(arena\text{-}assn *a (arena\text{-}assn *a\ vdom\text{-}assn *a\ vdom\text{-}assn) *a\ watchlist\text{-}assn)$›
⟨*proof*⟩

declare *isasat-GC-clauses-prog-wl2-code.refine*[*sepref-fr-rules*]
isasat-GC-clauses-prog-wl2-slow-code.refine[*sepref-fr-rules*]

sepref-register *isasat-GC-clauses-prog-wl isasat-GC-clauses-prog-wl2' rewatch-heur-st*
sepref-definition *isasat-GC-clauses-prog-wl-code*
is ‹*isasat-GC-clauses-prog-wl*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *isasat-GC-clauses-prog-wl-slow-code*
is ‹*isasat-GC-clauses-prog-wl*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *isa-arena-length-fast-code2*
is ‹*uncurry isa-arena-length*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
⟨*proof*⟩

lemma *isa-arena-length-fast-code2-refine*[*sepref-fr-rules*]:
‹$(uncurry\ isa\text{-}arena\text{-}length\text{-}fast\text{-}code2,\ uncurry\ (RETURN \circ\circ arena\text{-}length))$
$\in [uncurry\ arena\text{-}is\text{-}valid\text{-}clause\text{-}idx]_a$
$arena\text{-}fast\text{-}assn^k *_a nat\text{-}assn^k \rightarrow uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

lemma *rewatch-heur-st-pre-alt-def*:
‹$rewatch\text{-}heur\text{-}st\text{-}pre\ S \longleftrightarrow (\forall i \in set\ (get\text{-}vdom\ S).\ i \leq uint64\text{-}max)$›
⟨*proof*⟩
find-theorems $\forall x <$ *length* -. - -!- $\forall$ - $\in$ *set* -. -
sepref-definition *rewatch-heur-st-code*
is ‹*rewatch-heur-st*›
:: ‹$[\lambda S.\ rewatch\text{-}heur\text{-}st\text{-}pre\ S \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d$
$\rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *rewatch-heur-st-slow-code*
is ‹*rewatch-heur-st*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *isasat-GC-clauses-prog-wl-code.refine*[*sepref-fr-rules*]
isasat-GC-clauses-prog-wl-slow-code.refine[*sepref-fr-rules*]
rewatch-heur-st-slow-code.refine[*sepref-fr-rules*]
rewatch-heur-st-code.refine[*sepref-fr-rules*]

sepref-register *isasat-GC-clauses-wl-D*

sepref-definition *isasat-GC-clauses-wl-D-code*
  is ‹*isasat-GC-clauses-wl-D*›
  :: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
  ⟨*proof*⟩

sepref-definition *isasat-GC-clauses-wl-D-slow-code*
  is ‹*isasat-GC-clauses-wl-D*›
  :: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
  ⟨*proof*⟩

declare *isasat-GC-clauses-wl-D-code.refine*[*sepref-fr-rules*]
    isasat-GC-clauses-wl-D-slow-code.refine[*sepref-fr-rules*]

sepref-register *number-clss-to-keep*

sepref-register *access-vdom-at*

lemma (**in** −) *uint32-max-nat-hnr*:
  ‹$(uncurry0\ (return\ uint32\text{-}max),\ uncurry0\ (RETURN\ uint32\text{-}max)) \in$
    $unit\text{-}assn^k \rightarrow_a nat\text{-}assn$›
  ⟨*proof*⟩
lemma *nat-of-uint64*:
  ‹$(return\ o\ id,\ RETURN\ o\ nat\text{-}of\text{-}uint64) \in$
    $(uint64\text{-}assn)^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

sepref-definition *number-clss-to-keep-impl*
  is ‹*RETURN o number-clss-to-keep*›
  :: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a nat\text{-}assn$›
  ⟨*proof*⟩

sepref-definition *number-clss-to-keep-fast-impl*
  is ‹*RETURN o number-clss-to-keep*›
  :: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

declare *number-clss-to-keep-impl.refine*[*sepref-fr-rules*]
    number-clss-to-keep-fast-impl.refine[*sepref-fr-rules*]

sepref-definition *access-vdom-at-code*
  is ‹*uncurry* (*RETURN oo access-vdom-at*)›
  :: ‹$[uncurry\ access\text{-}vdom\text{-}at\text{-}pre]_a\ isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow nat\text{-}assn$›
  ⟨*proof*⟩

sepref-definition *access-vdom-at-fast-code*
  is ‹*uncurry* (*RETURN oo access-vdom-at*)›
  :: ‹$[uncurry\ access\text{-}vdom\text{-}at\text{-}pre]_a\ isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint64\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

declare *access-vdom-at-fast-code.refine*[*sepref-fr-rules*]
  access-vdom-at-code.refine[*sepref-fr-rules*]

end
theory *IsaSAT-Restart*
  imports *IsaSAT-Restart-Heuristics IsaSAT-CDCL*
begin

definition *cdcl-twl-stgy-restart-abs-wl-heur-inv* **where**
  ‹*cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 *brk T n* ⟷
    (∃ S_0' *T'*. (S_0, S_0') ∈ *twl-st-heur* ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧
      cdcl-twl-stgy-restart-abs-wl-D-inv S_0' *brk T' n*)›

definition *cdcl-twl-stgy-restart-prog-wl-heur*
   :: *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*
where
  ‹*cdcl-twl-stgy-restart-prog-wl-heur* S_0 = *do* {
    (*brk*, *T*, -) ← $WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n}$
      (λ(*brk*, -). ¬*brk*)
      (λ(*brk*, *S*, *n*).
      do {
        T ← *unit-propagation-outer-loop-wl-D-heur S*;
        (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
        (*T*, *n*) ← *restart-prog-wl-D-heur T n brk*;
        RETURN (*brk*, *T*, *n*)
      })
      (*False*, S_0::*twl-st-wl-heur*, *0*);
    RETURN T
  }›

lemma *cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D*:
  ‹(*cdcl-twl-stgy-restart-prog-wl-heur*, *cdcl-twl-stgy-restart-prog-wl-D*) ∈
    twl-st-heur $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

definition *fast-number-of-iterations* :: ‹- ⇒ *bool*› **where**
‹*fast-number-of-iterations n* ⟷ *n* < *uint64-max* >> *1*›

definition *cdcl-twl-stgy-restart-prog-early-wl-heur*
   :: *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*
where
  ‹*cdcl-twl-stgy-restart-prog-early-wl-heur* S_0 = *do* {
    ebrk ← *RETURN* (¬*isasat-fast* S_0);
    (*ebrk*, *brk*, *T*, *n*) ←
    $WHILE_T{}^{\lambda(ebrk,\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n\ \wedge}$ $(\neg ebrk \longrightarrow isasat\text{-}fast\ T) \wedge length\ (get\text{-}c$
      (λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
      (λ(*ebrk*, *brk*, *S*, *n*).
      do {
        ASSERT(¬*brk* ∧ ¬*ebrk*);
        ASSERT(*length* (*get-clauses-wl-heur S*) ≤ *uint64-max*);
        T ← *unit-propagation-outer-loop-wl-D-heur S*;
        ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *uint64-max*);
        ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*));
        (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
        ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *uint64-max*);

$(T, n) \leftarrow$ *restart-prog-wl-D-heur T n brk*;
ebrk $\leftarrow$ *RETURN* ($\neg$*isasat-fast T*);
RETURN (*ebrk*, *brk*, *T*, *n*)
})
(*ebrk*, *False*, S_0::*twl-st-wl-heur*, *0*);
ASSERT(*length* (*get-clauses-wl-heur T*) $\leq$ *uint64-max* $\wedge$
get-old-arena T = []);
if $\neg$*brk then do* {
$T \leftarrow$ *isasat-fast-slow T*;
(*brk*, *T*, -) $\leftarrow$ $WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n}$
(λ(*brk*, -). $\neg$*brk*)
(λ(*brk*, *S*, *n*).
do {
$T \leftarrow$ *unit-propagation-outer-loop-wl-D-heur S*;
(*brk*, *T*) $\leftarrow$ *cdcl-twl-o-prog-wl-D-heur T*;
(*T*, *n*) $\leftarrow$ *restart-prog-wl-D-heur T n brk*;
RETURN (*brk*, *T*, *n*)
})
(*False*, *T*, *n*);
RETURN T
}
else isasat-fast-slow T
}›

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
assumes *r*: ‹$r \leq$ *uint64-max*›
shows ‹(*cdcl-twl-stgy-restart-prog-early-wl-heur*, *cdcl-twl-stgy-restart-prog-early-wl-D*) $\in$
twl-st-heur''' $r \rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

definition *length-avdom* :: ‹*twl-st-wl-heur* $\Rightarrow$ *nat*› **where**
‹*length-avdom S* = *length* (*get-avdom S*)›

lemma *length-avdom-alt-def*:
‹*length-avdom* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *avdom*, *lcount*). *length avdom*)›
⟨*proof*⟩

definition *get-the-propagation-reason-heur*
:: ‹*twl-st-wl-heur* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat option nres*›
where
‹*get-the-propagation-reason-heur S* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*)›

lemma *get-the-propagation-reason-heur-alt-def*:
‹*get-the-propagation-reason-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *lcount*) *L* . *get-the-propagation-reason-pol M' L*)›
⟨*proof*⟩

definition *clause-is-learned-heur* :: *twl-st-wl-heur* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where
‹*clause-is-learned-heur S C* $\longleftrightarrow$ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED*›

lemma *clause-is-learned-heur-alt-def*:
‹*clause-is-learned-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *lcount*) *C* . *arena-status N' C* = *LEARNED*)›
⟨*proof*⟩

definition *clause-lbd-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *nat*
where
‹*clause-lbd-heur S C* = *arena-lbd* (*get-clauses-wl-heur S*) *C*›

lemma *clause-lbd-heur-alt-def*:
‹*clause-lbd-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *lcount*) *C* . *get-clause-LBD N' C*)›
⟨*proof*⟩

definition (**in** −) *access-length-heur* **where**
‹*access-length-heur S i* = *arena-length* (*get-clauses-wl-heur S*) *i*›

lemma *access-length-heur-alt-def*:
‹*access-length-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *lcount*) *C* . *arena-length N' C*)›
⟨*proof*⟩

definition *marked-as-used-st* **where**
‹*marked-as-used-st T C* =
marked-as-used (*get-clauses-wl-heur T*) *C*›

lemma *marked-as-used-st-alt-def*:
‹*marked-as-used-st* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *lcount*) *C* . *marked-as-used N' C*)›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹*D* = *Some C* ⟷ *D* ≠ *None* ∧ (*nat-of-uint64-conv* (*the D*) = *C*)›
⟨*proof*⟩

lemma (**in** −) *isasat-fast-alt-def*:
‹*RETURN o isasat-fast* = (λ(*M*, *N*, -). *RETURN* (*length N* ≤ *uint64-max* − (*uint32-max div 2* +
6)))›
⟨*proof*⟩

definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
:: *twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*
where
‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur* S_0 = *do* {
ebrk ← *RETURN* (¬*isasat-fast* S_0);
(*ebrk*, *brk*, *T*, *n*) ←
$WHILE_T{}^{\lambda(ebrk,\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n\ \wedge\ (\neg ebrk\ \longrightarrow isasat\text{-}fast\ T)\ \wedge\ length\ (get\text{-}c}$
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(*ebrk*, *brk*, *S*, *n*).
do {
ASSERT(¬*brk* ∧ ¬*ebrk*);
ASSERT(*length* (*get-clauses-wl-heur S*) ≤ *uint64-max*);
T ← *unit-propagation-outer-loop-wl-D-heur S*;

        ASSERT(*length* (*get-clauses-wl-heur* T) $\leq$ *uint64-max*);
        ASSERT(*length* (*get-clauses-wl-heur* T) = *length* (*get-clauses-wl-heur* S));
        (*brk*, T) $\leftarrow$ *cdcl-twl-o-prog-wl-D-heur* T;
        ASSERT(*length* (*get-clauses-wl-heur* T) $\leq$ *uint64-max*);
        (T, n) $\leftarrow$ *restart-prog-wl-D-heur* T n *brk*;
 ebrk $\leftarrow$ *RETURN* ($\neg$*isasat-fast* T);
        RETURN (*ebrk*, *brk*, T, n)
      })
      (*ebrk*, *False*, S_0::*twl-st-wl-heur*, *0*);
    RETURN (*brk*, T)
  }›

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D*:
  assumes *r*: ‹*r* $\leq$ *uint64-max*›
  shows ‹(*cdcl-twl-stgy-restart-prog-bounded-wl-heur*, *cdcl-twl-stgy-restart-prog-bounded-wl-D*) $\in$
   twl-st-heur''' *r* $\rightarrow_f$ ⟨*bool-rel* $\times_r$ *twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-Restart-SML*
  imports *IsaSAT-Restart IsaSAT-Restart-Heuristics-SML IsaSAT-CDCL-SML*
begin
sepref-register *length-avdom*

Find a less hack-like solution

setup ‹*map-theory-claset* (*fn ctxt* => *ctxt delSWrapper split-all-tac*)›

sepref-register *clause-is-learned-heur*

sepref-definition *length-avdom-code*
  is ‹*RETURN o length-avdom*›
  :: ‹*isasat-unbounded-assn*^{k} $\rightarrow_a$ *nat-assn*›
  ⟨*proof*⟩

sepref-definition *length-avdom-fast-code*
  is ‹*RETURN o length-avdom*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
  ⟨*proof*⟩

declare *length-avdom-code.refine*[*sepref-fr-rules*]
    length-avdom-fast-code.refine[*sepref-fr-rules*]

sepref-register *get-the-propagation-reason-heur*
sepref-definition *get-the-propagation-reason-heur-code*
  is ‹*uncurry get-the-propagation-reason-heur*›
  :: ‹*isasat-unbounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn nat-assn*›
  ⟨*proof*⟩

sepref-definition *get-the-propagation-reason-heur-fast-code*
  is ‹*uncurry get-the-propagation-reason-heur*›
  :: ‹*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *option-assn uint64-nat-assn*›
  ⟨*proof*⟩

declare *get-the-propagation-reason-heur-fast-code.refine*[*sepref-fr-rules*]
    get-the-propagation-reason-heur-code.refine[*sepref-fr-rules*]

sepref-definition *clause-is-learned-heur-code*
  is ‹*uncurry* (*RETURN oo* (*clause-is-learned-heur*))›
  :: ‹$[\lambda(S, C)$. *arena-is-valid-clause-vdom* (*get-clauses-wl-heur* S) $C]_a$
    isasat-unbounded-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *bool-assn*›
  ⟨*proof*⟩

sepref-definition *clause-is-learned-heur-code2*
  is ‹*uncurry* (*RETURN oo* (*clause-is-learned-heur*))›
  :: ‹$[\lambda(S, C)$. *arena-is-valid-clause-vdom* (*get-clauses-wl-heur* S) $C]_a$
    isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *bool-assn*›
  ⟨*proof*⟩

declare *clause-is-learned-heur-code.refine*[*sepref-fr-rules*]
  clause-is-learned-heur-code2.refine[*sepref-fr-rules*]

sepref-register *clause-lbd-heur*
sepref-definition *clause-lbd-heur-code*
  is ‹*uncurry* (*RETURN oo* (*clause-lbd-heur*))›
  :: ‹$[\lambda(S, C)$. *get-clause-LBD-pre* (*get-clauses-wl-heur* S) $C]_a$
    isasat-unbounded-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
  ⟨*proof*⟩

sepref-definition *clause-lbd-heur-code2*
  is ‹*uncurry* (*RETURN oo clause-lbd-heur*)›
  :: ‹$[\lambda(S, C)$. *get-clause-LBD-pre* (*get-clauses-wl-heur* S) $C]_a$
    isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
  ⟨*proof*⟩

declare *clause-lbd-heur-code2.refine*[*sepref-fr-rules*]
  clause-lbd-heur-code.refine[*sepref-fr-rules*]

sepref-register *mark-garbage-heur*

sepref-definition *mark-garbage-heur-code*
  is ‹*uncurry2* (*RETURN ooo mark-garbage-heur*)›
  :: ‹$[\lambda((C, i), S)$. *mark-garbage-pre* (*get-clauses-wl-heur* S, C) $\wedge$ $i <$ *length-avdom* $S]_a$
    nat-assn^k $*_a$ *nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
  ⟨*proof*⟩

definition *butlast-arl64* :: ‹'*a array-list64* ⇒ -› **where**
  ‹*butlast-arl64* = ($\lambda(xs, i)$. (xs, *fast-minus* i *1*))›

lemma *butlast-arl-hnr*[*sepref-fr-rules*]:
  ‹(*return o butlast-arl64*, *RETURN o op-list-butlast*) $\in [\lambda xs.\ xs \neq []]_a$ (*arl64-assn* $A)^d \rightarrow$ *arl64-assn* A›
⟨*proof*⟩

declare *butlast-arl-hnr*[*unfolded op-list-butlast-def butlast-nonresizing-def*[*symmetric*], *sepref-fr-rules*]

sepref-definition *mark-garbage-heur-code2*
  is ‹*uncurry2* (*RETURN ooo mark-garbage-heur*)›
  :: ‹$[\lambda((C, i), S)$. *mark-garbage-pre* (*get-clauses-wl-heur* S, C) $\wedge$ $i <$ *length-avdom* S $\wedge$

get-learned-count S $\geq$ *1*$]_a$
uint64-nat-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *mark-garbage-heur-code.refine*[*sepref-fr-rules*]
mark-garbage-heur-code2.refine[*sepref-fr-rules*]

sepref-register *delete-index-vdom-heur*
sepref-definition *delete-index-vdom-heur-code*
is ‹*uncurry* (*RETURN oo delete-index-vdom-heur*)›
:: ‹[λ(*i*, *S*). *i* < *length-avdom S*$]_a$
nat-assn^k $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *delete-index-vdom-heur-fast-code2*
is ‹*uncurry* (*RETURN oo delete-index-vdom-heur*)›
:: ‹[λ(*i*, *S*). *i* < *length-avdom S*$]_a$
uint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *delete-index-vdom-heur-code.refine*[*sepref-fr-rules*]
delete-index-vdom-heur-fast-code2.refine[*sepref-fr-rules*]

sepref-register *access-length-heur*
sepref-definition *access-length-heur-code*
is ‹*uncurry* (*RETURN oo access-length-heur*)›
:: ‹[λ(*S*, *C*). *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *C*$]_a$
isasat-unbounded-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-definition *access-length-heur-fast-code2*
is ‹*uncurry* (*RETURN oo access-length-heur*)›
:: ‹[λ(*S*, *C*). *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *C*$]_a$
isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
⟨*proof*⟩

declare *access-length-heur-code.refine*[*sepref-fr-rules*]
access-length-heur-fast-code2.refine[*sepref-fr-rules*]

sepref-definition *isa-marked-as-used-fast-code*
is ‹*uncurry isa-marked-as-used*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *isa-marked-as-used-code*[*sepref-fr-rules*]:
‹(*uncurry isa-marked-as-used-fast-code*, *uncurry* (*RETURN* $\circ\circ$ *marked-as-used*))
$\in$ [*uncurry marked-as-used-pre*$]_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *bool-assn*›
⟨*proof*⟩

sepref-definition *isa-marked-as-used-fast-code2*
is ‹*uncurry isa-marked-as-used*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

lemma *isa-marked-as-used-code2*[*sepref-fr-rules*]:
‹(*uncurry isa-marked-as-used-fast-code2*, *uncurry* (*RETURN* ∘∘ *marked-as-used*))
∈ [*uncurry marked-as-used-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

sepref-register *marked-as-used-st*
sepref-definition *marked-as-used-st-code*
is ‹*uncurry* (*RETURN oo marked-as-used-st*)›
:: ‹[λ(*S*, *C*). *marked-as-used-pre* (*get-clauses-wl-heur S*) *C*]$_a$
isasat-unbounded-assn^k $*_a$ *nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

sepref-definition *marked-as-used-st-fast-code*
is ‹*uncurry* (*RETURN oo marked-as-used-st*)›
:: ‹[λ(*S*, *C*). *marked-as-used-pre* (*get-clauses-wl-heur S*) *C*]$_a$
isasat-bounded-assn^k $*_a$ *uint64-nat-assn*^{k} → *bool-assn*›
⟨*proof*⟩

declare *marked-as-used-st-code.refine*[*sepref-fr-rules*]
marked-as-used-st-fast-code.refine[*sepref-fr-rules*]

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-unused arena C*) *C*›
⟨*proof*⟩

sepref-definition *mark-unused-st-code*
is ‹*uncurry* (*RETURN oo mark-unused-st-heur*)›
:: ‹[λ(*C*, *S*). *arena-act-pre* (*get-clauses-wl-heur S*) *C*]$_a$
nat-assn^k $*_a$ *isasat-unbounded-assn*^{d} → *isasat-unbounded-assn*›
⟨*proof*⟩

sepref-definition *isa-mark-unused-fast-code*
is ‹*uncurry isa-mark-unused*›
:: ‹(*arl64-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl64-assn uint32-assn*)›
⟨*proof*⟩

lemma *isa-mark-unused-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-unused-fast-code*, *uncurry* (*RETURN* ∘∘ *mark-unused*))
∈ [*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *uint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

sepref-register *mark-unused-st-heur*
sepref-definition *mark-unused-st-fast-code*
is ‹*uncurry* (*RETURN oo mark-unused-st-heur*)›
:: ‹[λ(*C*, *S*). *arena-act-pre* (*get-clauses-wl-heur S*) *C*]$_a$
uint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

declare *mark-unused-st-code.refine*[*sepref-fr-rules*]
mark-unused-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *mark-clauses-as-unused-wl-D-heur*
sepref-definition *mark-clauses-as-unused-wl-D-heur-code*
is ‹*uncurry mark-clauses-as-unused-wl-D-heur*›
:: ‹$nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *clause-not-marked-to-delete-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *mark-clauses-as-unused-wl-D-heur-fast-code*
is ‹*uncurry mark-clauses-as-unused-wl-D-heur*›
:: ‹$[\lambda(\text{-}, S).\ length\ (get\text{-}avdom\ S) \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *mark-clauses-as-unused-wl-D-heur-fast-code.refine*[*sepref-fr-rules*]
mark-clauses-as-unused-wl-D-heur-code.refine[*sepref-fr-rules*]

sepref-register *mark-to-delete-clauses-wl-D-heur*
sepref-definition *mark-to-delete-clauses-wl-D-heur-impl*
is ‹*mark-to-delete-clauses-wl-D-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *sort-vdom-heur-fast-code.refine*[*sepref-fr-rules*]
sort-vdom-heur-fast-code.refine[*sepref-fr-rules*]

declare *access-lit-in-clauses-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *mark-to-delete-clauses-wl-D-heur-fast-impl*
is ‹*mark-to-delete-clauses-wl-D-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *mark-to-delete-clauses-wl-D-heur-fast-impl.refine*[*sepref-fr-rules*]
mark-to-delete-clauses-wl-D-heur-impl.refine[*sepref-fr-rules*]

sepref-register *cdcl-twl-full-restart-wl-prog-heur*
sepref-definition *cdcl-twl-full-restart-wl-prog-heur-code*
is ‹*cdcl-twl-full-restart-wl-prog-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *cdcl-twl-full-restart-wl-prog-heur-fast-code*
is ‹*cdcl-twl-full-restart-wl-prog-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *cdcl-twl-full-restart-wl-prog-heur-fast-code.refine*[*sepref-fr-rules*]
cdcl-twl-full-restart-wl-prog-heur-code.refine[*sepref-fr-rules*]

sepref-definition *cdcl-twl-restart-wl-heur-code*
is ‹*cdcl-twl-restart-wl-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *cdcl-twl-restart-wl-heur-fast-code*
is ‹*cdcl-twl-restart-wl-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *cdcl-twl-restart-wl-heur-fast-code.refine*[*sepref-fr-rules*]
cdcl-twl-restart-wl-heur-code.refine[*sepref-fr-rules*]

sepref-definition *cdcl-twl-full-restart-wl-D-GC-heur-prog-code*
is ‹*cdcl-twl-full-restart-wl-D-GC-heur-prog*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

sepref-definition *cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code*
is ‹*cdcl-twl-full-restart-wl-D-GC-heur-prog*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *cdcl-twl-full-restart-wl-D-GC-heur-prog-code.refine*[*sepref-fr-rules*]
cdcl-twl-restart-wl-heur-fast-code.refine[*sepref-fr-rules*]
cdcl-twl-full-restart-wl-D-GC-heur-prog-code.refine[*sepref-fr-rules*]
cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code.refine[*sepref-fr-rules*]

declare *cdcl-twl-restart-wl-heur-fast-code.refine*[*sepref-fr-rules*]
cdcl-twl-restart-wl-heur-code.refine[*sepref-fr-rules*]

sepref-register *restart-required-heur cdcl-twl-restart-wl-heur*
sepref-definition *restart-wl-D-heur-slow-code*
is ‹*uncurry2 restart-prog-wl-D-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d *_a nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ nat\text{-}assn$›
⟨*proof*⟩

sepref-definition *restart-prog-wl-D-heur-fast-code*
is ‹*uncurry2* (*restart-prog-wl-D-heur*)›
:: ‹$[\lambda((S, \text{-}), \text{-}).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d *_a nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow isasat\text{-}bounded\text{-}assn *a\ nat\text{-}assn$›
⟨*proof*⟩

declare *restart-wl-D-heur-slow-code.refine*[*sepref-fr-rules*]
restart-prog-wl-D-heur-fast-code.refine[*sepref-fr-rules*]

sepref-definition *cdcl-twl-stgy-restart-prog-wl-heur-code*
is ‹*cdcl-twl-stgy-restart-prog-wl-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
⟨*proof*⟩

declare *cdcl-twl-stgy-restart-prog-wl-heur-code.refine*[*sepref-fr-rules*]

definition *isasat-fast-bound* **where**
‹$isasat\text{-}fast\text{-}bound = uint64\text{-}max - (uint32\text{-}max\ div\ 2 + 6)$›

lemma *isasat-fast-bound*[*sepref-fr-rules*]:
  ‹(*uncurry0* (*return 18446744071562067962*), *uncurry0* (*RETURN isasat-fast-bound*)) ∈
  unit-assn^k $\rightarrow_a$ *uint64-nat-assn*›
  ⟨*proof*⟩

sepref-register *isasat-fast*
sepref-definition *isasat-fast-code*
  is ‹*RETURN o isasat-fast*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool-assn*›
  ⟨*proof*⟩

declare *isasat-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *cdcl-twl-stgy-restart-prog-wl-heur-fast-code*
  is ‹*cdcl-twl-stgy-restart-prog-early-wl-heur*›
  :: ‹$[\lambda S.$ *isasat-fast* $S]_a$ *isasat-bounded-assn*^{d} → *isasat-unbounded-assn*›
  ⟨*proof*⟩

declare *cdcl-twl-stgy-restart-prog-wl-heur-fast-code.refine*[*sepref-fr-rules*]

end
theory *IsaSAT*
  imports *IsaSAT-Restart IsaSAT-Initialisation*
begin

0.2.8 Final code generation

We now combine all the previous definitions to prove correctness of the complete SAT solver:

1. We initialise the arena part of the state;

2. Then depending on the options and the number of clauses, we either use the bounded version or the unbounded version. Once have if decided which one, we initiale the watch lists;

3. After that, we can run the CDCL part of the SAT solver;

4. Finally, we extract the trail from the state.

   Remark that the statistics and the options are unchecked: the number of propagations might overflows (but they do not impact the correctness of the whole solver). Similar restriction applies on the options: setting the options might not do what you expect to happen, but the result will still be correct.

Correctness Relation

We cannot use *cdcl-twl-stgy-restart* since we do not always end in a final state for *cdcl-twl-stgy*.

definition *conclusive-TWL-run* :: ‹'*v twl-st* ⇒ '*v twl-st nres*› **where**
 ‹*conclusive-TWL-run S* =
   SPEC($\lambda T.\ \exists\, n\ n'.$ *cdcl-twl-stgy-restart-with-leftovers*** $(S,\ n)$ $(T,\ n')$ ∧ *final-twl-state* T)›

To get a full CDCL run:

- either we fully apply $cdcl_W$-restart-mset.$cdcl_W$-stgy (up to restarts)
- or we can stop early.

definition *conclusive-CDCL-run* **where**
‹*conclusive-CDCL-run CS T U* $\longleftrightarrow$
  ($\exists$ *n n'*. $cdcl_W$-*restart-mset*.$cdcl_W$-*restart-stgy*** (*T*, *n*) (*U*, *n'*) $\wedge$
      no-step $cdcl_W$-*restart-mset*.$cdcl_W$ (*U*)) $\vee$
   (*CS* $\neq$ {#} $\wedge$ *conflicting U* $\neq$ *None* $\wedge$ *count-decided* (*trail U*) = *0* $\wedge$
   unsatisfiable (*set-mset CS*))›

lemma *cdcl-twl-stgy-restart-restart-prog-spec*: ‹*twl-struct-invs S* $\Longrightarrow$
twl-stgy-invs S $\Longrightarrow$
clauses-to-update S = {#} $\Longrightarrow$
get-conflict S = *None* $\Longrightarrow$
cdcl-twl-stgy-restart-prog S $\leq$ *conclusive-TWL-run S*›
⟨*proof*⟩

lemma *cdcl-twl-stgy-restart-restart-prog-early-spec*: ‹*twl-struct-invs S* $\Longrightarrow$
twl-stgy-invs S $\Longrightarrow$
clauses-to-update S = {#} $\Longrightarrow$
get-conflict S = *None* $\Longrightarrow$
cdcl-twl-stgy-restart-prog-early S $\leq$ *conclusive-TWL-run S*›
⟨*proof*⟩

theorem *cdcl-twl-stgy-restart-prog-wl-D-spec*:
assumes ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S*›
shows ‹*cdcl-twl-stgy-restart-prog-wl-D S* $\leq$ $\Downarrow$*Id* (*cdcl-twl-stgy-restart-prog-wl S*)›
⟨*proof*⟩

theorem *cdcl-twl-stgy-restart-prog-early-wl-D-spec*:
assumes ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S*›
shows ‹*cdcl-twl-stgy-restart-prog-early-wl-D S* $\leq$ $\Downarrow$*Id* (*cdcl-twl-stgy-restart-prog-early-wl S*)›
⟨*proof*⟩

lemma *distinct-nat-of-uint32*[*iff*]:
‹*distinct-mset* (*nat-of-uint32* '# *A*) $\longleftrightarrow$ *distinct-mset A*›
‹*distinct* (*map nat-of-uint32 xs*) $\longleftrightarrow$ *distinct xs*›
⟨*proof*⟩

lemma $cdcl_W$-*ex*-$cdcl_W$-*stgy*:
‹$cdcl_W$-*restart-mset*.$cdcl_W$ *S T* $\Longrightarrow$ $\exists$ *U*. $cdcl_W$-*restart-mset*.$cdcl_W$-*stgy S U*›
⟨*proof*⟩

lemma *rtranclp*-$cdcl_W$-$cdcl_W$-*init-state*:
‹$cdcl_W$-*restart-mset*.$cdcl_W$** (*init-state* {#}) *S* $\longleftrightarrow$ *S* = *init-state* {#}›
⟨*proof*⟩

definition *init-state-l* :: ‹*'v twl-st-l-init*› **where**
‹*init-state-l* = (([], *fmempty*, *None*, {#}, {#}, {#}, {#}), {#})›

definition *to-init-state-l* :: ‹*nat twl-st-l-init* $\Rightarrow$ *nat twl-st-l-init*› **where**
‹*to-init-state-l S* = *S*›

definition *init-state0* :: ‹*'v twl-st-init*› **where**

‹init-state0 = (([], {#}, {#}, None, {#}, {#}, {#}, {#}), {#})›

definition *to-init-state0* :: ‹nat twl-st-init ⇒ nat twl-st-init› **where**
‹to-init-state0 S = S›

lemma *init-dt-pre-init*:
assumes *dist*: ‹Multiset.Ball (mset '# mset CS) distinct-mset›
shows ‹init-dt-pre CS (to-init-state-l init-state-l)›
⟨proof⟩

This is the specification of the SAT solver:

definition *SAT* :: ‹nat clauses ⇒ nat $cdcl_W$-restart-mset nres› **where**
‹SAT CS = do{
 let T = init-state CS;
 SPEC (conclusive-CDCL-run CS T)
}›

definition *init-dt-spec0* :: ‹'v clause-l list ⇒ 'v twl-st-init ⇒ 'v twl-st-init ⇒ bool› **where**
‹init-dt-spec0 CS SOC T' ⟷
 (
 twl-struct-invs-init T' ∧
 clauses-to-update-init T' = {#} ∧
 (∀ s∈set (get-trail-init T'). ¬is-decided s) ∧
 (get-conflict-init T' = None ⟶
literals-to-update-init T' = uminus '# lit-of '# mset (get-trail-init T')) ∧
 (mset '# mset CS + clause '# (get-init-clauses-init SOC) + other-clauses-init SOC +
 get-unit-init-clauses-init SOC =
 clause '# (get-init-clauses-init T') + other-clauses-init T' +
 get-unit-init-clauses-init T') ∧
 get-learned-clauses-init SOC = get-learned-clauses-init T' ∧
 get-unit-learned-clauses-init T' = get-unit-learned-clauses-init SOC ∧
 twl-stgy-invs (fst T') ∧
 (other-clauses-init T' ≠ {#} ⟶ get-conflict-init T' ≠ None) ∧
 ({#} ∈# mset '# mset CS ⟶ get-conflict-init T' ≠ None) ∧
 (get-conflict-init SOC ≠ None ⟶ get-conflict-init SOC = get-conflict-init T'))›

Refinements of the Whole SAT Solver

We do no add the refinement steps in separate files, since the form is very specific to the SAT solver we want to generate (and needs to be updated if it changes).

definition *SAT0* :: ‹nat clause-l list ⇒ nat twl-st nres› **where**
‹SAT0 CS = do{
 b ← SPEC(λ-::bool. True);
 if b then do {
 let S = init-state0;
 T ← SPEC (init-dt-spec0 CS (to-init-state0 S));
 let T = fst T;
 if get-conflict T ≠ None
 then RETURN T
 else if CS = [] then RETURN (fst init-state0)
 else do {
 ASSERT (extract-atms-clss CS {} ≠ {});
 ASSERT (clauses-to-update T = {#});
 ASSERT(clause '# (get-clauses T) + unit-clss T = mset '# mset CS);

    $ASSERT(get\text{-}learned\text{-}clss\ T = \{\#\})$;
    $cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\ T$
  }
}
else do {
  let $S = init\text{-}state0$;
  $T \leftarrow\ SPEC\ (init\text{-}dt\text{-}spec0\ CS\ (to\text{-}init\text{-}state0\ S))$;
  $failed \leftarrow SPEC\ (\lambda\text{-} :: bool.\ True)$;
  if failed then do {
    $T \leftarrow\ SPEC\ (init\text{-}dt\text{-}spec0\ CS\ (to\text{-}init\text{-}state0\ S))$;
    let $T = fst\ T$;
    if $get\text{-}conflict\ T \neq None$
    then $RETURN\ T$
    else if $CS = []$ *then* $RETURN\ (fst\ init\text{-}state0)$
    else do {
      $ASSERT\ (extract\text{-}atms\text{-}clss\ CS\ \{\} \neq \{\})$;
      $ASSERT\ (clauses\text{-}to\text{-}update\ T = \{\#\})$;
      $ASSERT(clause\ `\#\ (get\text{-}clauses\ T) + unit\text{-}clss\ T = mset\ `\#\ mset\ CS)$;
      $ASSERT(get\text{-}learned\text{-}clss\ T = \{\#\})$;
      $cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\ T$
    }
  } *else do* {
    let $T = fst\ T$;
    if $get\text{-}conflict\ T \neq None$
    then $RETURN\ T$
    else if $CS = []$ *then* $RETURN\ (fst\ init\text{-}state0)$
    else do {
      $ASSERT\ (extract\text{-}atms\text{-}clss\ CS\ \{\} \neq \{\})$;
      $ASSERT\ (clauses\text{-}to\text{-}update\ T = \{\#\})$;
      $ASSERT(clause\ `\#\ (get\text{-}clauses\ T) + unit\text{-}clss\ T = mset\ `\#\ mset\ CS)$;
      $ASSERT(get\text{-}learned\text{-}clss\ T = \{\#\})$;
      $cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\ T$
    }
  }
 }
}›

lemma *SAT0-SAT*:
assumes ‹$Multiset.Ball\ (mset\ `\#\ mset\ CS)\ distinct\text{-}mset$›
shows ‹$SAT0\ CS \leq \Downarrow \{(S,\ T).\ T = state_W\text{-}of\ S\}\ (SAT\ (mset\ `\#\ mset\ CS))$›
⟨*proof*⟩

definition *SAT-l* :: ‹*nat clause-l list* ⇒ *nat twl-st-l nres*› **where**
‹*SAT-l CS = do*{
  $b \leftarrow SPEC(\lambda\text{-}::bool.\ True)$;
  if b then do {
    let $S = init\text{-}state\text{-}l$;
    $T \leftarrow init\text{-}dt\ CS\ (to\text{-}init\text{-}state\text{-}l\ S)$;
    let $T = fst\ T$;
    if $get\text{-}conflict\text{-}l\ T \neq None$
    then $RETURN\ T$
    else if $CS = []$ *then* $RETURN\ (fst\ init\text{-}state\text{-}l)$
    else do {
      $ASSERT\ (extract\text{-}atms\text{-}clss\ CS\ \{\} \neq \{\})$;
  $ASSERT\ (clauses\text{-}to\text{-}update\text{-}l\ T = \{\#\})$;
      $ASSERT(mset\ `\#\ ran\text{-}mf\ (get\text{-}clauses\text{-}l\ T) + get\text{-}unit\text{-}clauses\text{-}l\ T = mset\ `\#\ mset\ CS)$;

$ASSERT(learned\text{-}clss\text{-}l\ (get\text{-}clauses\text{-}l\ T) = \{\#\});$
cdcl-twl-stgy-restart-prog-l T
}
}
else do {
let S = init-state-l;
$T \leftarrow$ *init-dt CS* (*to-init-state-l S*);
failed $\leftarrow$ *SPEC* (λ- :: *bool. True*);
if failed then do {
$T \leftarrow$ *init-dt CS* (*to-init-state-l S*);
let T = fst T;
if get-conflict-l T $\neq$ *None*
then RETURN T
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} $\neq$ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-l T
}
} *else do* {
let T = fst T;
if get-conflict-l T $\neq$ *None*
then RETURN T
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} $\neq$ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-early-l T
}
}
}
}›

lemma *SAT-l-SAT0*:
assumes *dist*: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*SAT-l CS* $\leq \Downarrow$ {(T,T'). (T, T') $\in$ *twl-st-l None*} (*SAT0 CS*)›
⟨*proof*⟩

definition *SAT-wl* :: ‹*nat clause-l list* $\Rightarrow$ *nat twl-st-wl nres*› **where**
‹*SAT-wl CS = do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(*distinct-mset-set* (*mset* ‘ *set CS*));
let $\mathcal{A}_{in}{}'$ = *extract-atms-clss CS* {};
$b \leftarrow$ *SPEC*(λ-::*bool. True*);
if b then do {
let S = init-state-wl;
$T \leftarrow$ *init-dt-wl′ CS* (*to-init-state S*);
$T \leftarrow$ *rewatch-st* (*from-init-state T*);
if get-conflict-wl T $\neq$ *None*
then RETURN T
else if CS = [] *then RETURN* (([], *fmempty*, *None*, {#}, {#}, {#}, λ-. *undefined*))
else do {

```
  ASSERT (extract-atms-clss CS {} ≠ {});
  ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
  ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T = mset ‘# mset CS);
  ASSERT(learned-clss-l (get-clauses-wl T) = {#});
  cdcl-twl-stgy-restart-prog-wl-D (finalise-init T)
      }
   }
   else do {
      let S = init-state-wl;
      T ← init-dt-wl' CS (to-init-state S);
      let T = from-init-state T;
      failed ← SPEC (λ- :: bool. True);
      if failed then do {
        let S = init-state-wl;
        T ← init-dt-wl' CS (to-init-state S);
        T ← rewatch-st (from-init-state T);
        if get-conflict-wl T ≠ None
        then RETURN T
        else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, λ-. undefined))
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
          ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
          ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T = mset ‘# mset CS);
          ASSERT(learned-clss-l (get-clauses-wl T) = {#});
          cdcl-twl-stgy-restart-prog-wl-D (finalise-init T)
        }
      } else do {
        if get-conflict-wl T ≠ None
        then RETURN T
        else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, λ-. undefined))
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
          ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
          ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T = mset ‘# mset CS);
          ASSERT(learned-clss-l (get-clauses-wl T) = {#});
          T ← rewatch-st (finalise-init T);
          cdcl-twl-stgy-restart-prog-early-wl-D T
        }
      }
    }
  }›
```

lemma *SAT-l-alt-def*:

```
 ‹SAT-l CS = do{
   𝒜 ← RETURN (); atoms
   b ← SPEC(λ-::bool. True);
   if b then do {
      let S = init-state-l;
      𝒜 ← RETURN (); initialisation
      T ← init-dt CS (to-init-state-l S);  rewatch
      let T = fst T;
      if get-conflict-l T ≠ None
      then RETURN T
      else if CS = [] then RETURN (fst init-state-l)
      else do {
```

ASSERT (*extract-atms-clss CS* {} $\neq$ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-l T
}
}
else do {
let S = *init-state-l*;
$\mathcal{A}$ $\leftarrow$ *RETURN* (); ~~*initialisation*~~
T $\leftarrow$ *init-dt CS* (*to-init-state-l S*);
failed $\leftarrow$ *SPEC* (λ- :: *bool*. *True*);
if failed then do {
let S = *init-state-l*;
$\mathcal{A}$ $\leftarrow$ *RETURN* (); ~~*initialisation*~~
T $\leftarrow$ *init-dt CS* (*to-init-state-l S*);
let T = *T*;
if get-conflict-l-init T $\neq$ *None*
then RETURN (*fst T*)
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} $\neq$ {});
ASSERT (*clauses-to-update-l* (*fst T*) = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l* (*fst T*)) + *get-unit-clauses-l* (*fst T*) = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l* (*fst T*)) = {#});
let T = *fst T*;
cdcl-twl-stgy-restart-prog-l T
}
} *else do* {
let T = *T*;
if get-conflict-l-init T $\neq$ *None*
then RETURN (*fst T*)
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} $\neq$ {});
ASSERT (*clauses-to-update-l* (*fst T*) = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l* (*fst T*)) + *get-unit-clauses-l* (*fst T*) = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l* (*fst T*)) = {#});
let T = *fst T*;
cdcl-twl-stgy-restart-prog-early-l T
}
}
}
}›
⟨*proof*⟩

lemma *init-dt-wl-full-init-dt-wl-spec-full*:
assumes ‹*init-dt-wl-pre CS S*› **and** ‹*init-dt-pre CS S'*› **and**
‹(*S*, *S'*) $\in$ *state-wl-l-init*› **and** ‹$\forall$ *C*$\in$*set CS*. *distinct C*›
shows ‹*init-dt-wl-full CS S* $\leq$ $\Downarrow$ {(*S*, *S'*). (*fst S*, *fst S'*) $\in$ *state-wl-l None*} (*init-dt CS S'*)›
⟨*proof*⟩

lemma *init-dt-wl-pre*:
assumes *dist*: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›

shows ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
⟨*proof*⟩

lemma *SAT-wl-SAT-l*:
assumes
dist: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*› **and**
bounded: ‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*SAT-wl CS* ≤ ⇓ {(*T*,*T′*). (*T*, *T′*) ∈ *state-wl-l None*} (*SAT-l CS*)›
⟨*proof*⟩

definition *extract-model-of-state* **where**
‹*extract-model-of-state U* = *Some* (*map lit-of* (*get-trail-wl U*))›

definition *extract-model-of-state-heur* **where**
‹*extract-model-of-state-heur U* = *Some* (*fst* (*get-trail-wl-heur U*))›

definition *extract-stats* **where**
[*simp*]: ‹*extract-stats U* = *None*›

definition *extract-stats-init* **where**
[*simp*]: ‹*extract-stats-init* = *None*›

definition *IsaSAT* :: ‹*nat clause-l list* ⇒ *nat literal list option nres*› **where**
‹*IsaSAT CS* = *do*{
S ← *SAT-wl CS*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}›

lemma *IsaSAT-alt-def*:
‹*IsaSAT CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(*distinct-mset-set* (*mset* ‘ *set CS*));
let $\mathcal{A}_{in}{}'$ = *extract-atms-clss CS* {};
- ← *RETURN* ();
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN (*extract-stats T*)
else if CS = [] *then RETURN* (*Some* [])
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}{}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
T ← *RETURN* (*finalise-init T*);
S ← *cdcl-twl-stgy-restart-prog-wl-D* (*T*);
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}
}
else do {
let S = *init-state-wl*;

T ← *init-dt-wl′ CS* (*to-init-state S*);
failed ← *SPEC* (λ- :: *bool. True*);
if failed then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN (*extract-stats T*)
else if CS = [] *then RETURN* (*Some* [])
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}$′));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
let T = *finalise-init T*;
S ← *cdcl-twl-stgy-restart-prog-wl-D T*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}
} *else do* {
let T = *from-init-state T*;
if get-conflict-wl T ≠ *None*
then RETURN (*extract-stats T*)
else if CS = [] *then RETURN* (*Some* [])
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}$′));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
T ← *rewatch-st T*;
T ← *RETURN* (*finalise-init T*);
S ← *cdcl-twl-stgy-restart-prog-early-wl-D T*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}
}
}
}› (**is** ‹*?A* = *?B*›) **for** *CS opts*
⟨*proof*⟩

definition *extract-model-of-state-stat* :: ‹*twl-st-wl-heur* ⇒ *nat literal list option* × *stats*› **where**
‹*extract-model-of-state-stat U* =
(*Some* (*fst* (*get-trail-wl-heur U*)),
(λ(*M*, -, -, -, - ,- ,- ,-, -, -, -, *stat*, -, -). *stat*) *U*)›

definition *extract-state-stat* :: ‹*twl-st-wl-heur* ⇒ *nat literal list option* × *stats*› **where**
‹*extract-state-stat U* =
(*None*,
(λ(*M*, -, -, -, - ,- ,- ,-, -, -, -, *stat*, -, -). *stat*) *U*)›

definition *empty-conflict* :: ‹*nat literal list option*› **where**
‹*empty-conflict* = *Some* []›

definition *empty-conflict-code* :: ‹(- *list option* × *stats*) *nres*› **where**
‹*empty-conflict-code* = *do*{
let M0 = [];
let M1 = *Some M0*;
RETURN (*M1*, (*zero-uint64*, *zero-uint64*, *zero-uint64*, *zero-uint64*, *zero-uint64*, *zero-uint64*,

zero-uint64,
    zero-uint64))}⟩

definition *empty-init-code* :: ⟨*- list option* × *stats*⟩ **where**
⟨*empty-init-code* = (*None*, (*zero-uint64*, *zero-uint64*, *zero-uint64*, *zero-uint64*,
zero-uint64, *zero-uint64*, *zero-uint64*, *zero-uint64*))⟩

definition *convert-state* **where**
⟨*convert-state - S* = *S*⟩

definition *IsaSAT-use-fast-mode* **where**
⟨*IsaSAT-use-fast-mode* = *True*⟩

definition *isasat-fast-init* :: ⟨*twl-st-wl-heur-init* ⇒ *bool*⟩ **where**
⟨*isasat-fast-init S* ⟷ (*length* (*get-clauses-wl-heur-init S*) ≤ *uint64-max* − (*uint32-max div 2* + *6*))⟩

definition *IsaSAT-heur* :: ⟨*opts* ⇒ *nat clause-l list* ⇒ (*nat literal list option* × *stats*) *nres*⟩ **where**
⟨*IsaSAT-heur opts CS* = *do*{
  ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
  ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint-max*);
  let $\mathcal{A}_{in}$' = *mset-set* (*extract-atms-clss CS* {});
  ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}$');
  ASSERT(*distinct-mset* $\mathcal{A}_{in}$');
  let $\mathcal{A}_{in}$'' = *virtual-copy* $\mathcal{A}_{in}$';
  let b = *opts-unbounded-mode opts*;
  if b
  then do {
    S ← *init-state-wl-heur* $\mathcal{A}_{in}$';
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur True CS S*;
T ← *rewatch-heur-st T*;
    let T = *convert-state* $\mathcal{A}_{in}$'' *T*;
    if ¬*get-conflict-wl-is-None-heur-init T*
    then RETURN (*empty-init-code*)
    else if CS = [] *then empty-conflict-code*
    else do {
      ASSERT($\mathcal{A}_{in}$'' ≠ {#});
      ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}$'');
      - ← *isasat-information-banner T*;
      ASSERT((λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As ≠ *None* ∧
        lst-As ≠ *None*) *T*);
      T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
      U ← *cdcl-twl-stgy-restart-prog-wl-heur T*;
      RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
        else extract-state-stat U)
    }
  }
  else do {
    S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}$';
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur False CS S*;
    let failed = *is-failed-heur-init T* ∨ ¬*isasat-fast-init T*;
    if failed then do {
      let $\mathcal{A}_{in}$' = *mset-set* (*extract-atms-clss CS* {});
      S ← *init-state-wl-heur* $\mathcal{A}_{in}$';

$(T::twl\text{-}st\text{-}wl\text{-}heur\text{-}init) \leftarrow init\text{-}dt\text{-}wl\text{-}heur\ True\ CS\ S$;
let $T = convert\text{-}state\ \mathcal{A}_{in}''\ T$;
$T \leftarrow rewatch\text{-}heur\text{-}st\ T$;
if $\neg get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\text{-}init\ T$
then $RETURN\ (empty\text{-}init\text{-}code)$
else if $CS = []$ *then* $empty\text{-}conflict\text{-}code$
else do {
$ASSERT(\mathcal{A}_{in}'' \neq \{\#\})$;
$ASSERT(isasat\text{-}input\text{-}bounded\text{-}nempty\ \mathcal{A}_{in}'')$;
$- \leftarrow isasat\text{-}information\text{-}banner\ T$;
$ASSERT((\lambda(M', N', D', Q', W', ((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove), \varphi, clvls).$
$fst\text{-}As \neq None \wedge$
$lst\text{-}As \neq None)\ T)$;
$T \leftarrow finalise\text{-}init\text{-}code\ opts\ (T::twl\text{-}st\text{-}wl\text{-}heur\text{-}init)$;
$U \leftarrow cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}wl\text{-}heur\ T$;
$RETURN$ (*if* $get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\ U$ *then* $extract\text{-}model\text{-}of\text{-}state\text{-}stat\ U$
else $extract\text{-}state\text{-}stat\ U$)
}
}
else do {
let $T = convert\text{-}state\ \mathcal{A}_{in}''\ T$;
if $\neg get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\text{-}init\ T$
then $RETURN\ (empty\text{-}init\text{-}code)$
else if $CS = []$ *then* $empty\text{-}conflict\text{-}code$
else do {
$ASSERT(\mathcal{A}_{in}'' \neq \{\#\})$;
$ASSERT(isasat\text{-}input\text{-}bounded\text{-}nempty\ \mathcal{A}_{in}'')$;
$- \leftarrow isasat\text{-}information\text{-}banner\ T$;
$ASSERT((\lambda(M', N', D', Q', W', ((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove), \varphi, clvls).$
$fst\text{-}As \neq None \wedge$
$lst\text{-}As \neq None)\ T)$;
$ASSERT(rewatch\text{-}heur\text{-}st\text{-}fast\text{-}pre\ T)$;
$T \leftarrow rewatch\text{-}heur\text{-}st\text{-}fast\ T$;
$ASSERT(isasat\text{-}fast\text{-}init\ T)$;
$T \leftarrow finalise\text{-}init\text{-}code\ opts\ (T::twl\text{-}st\text{-}wl\text{-}heur\text{-}init)$;
$ASSERT(isasat\text{-}fast\ T)$;
$U \leftarrow cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\text{-}wl\text{-}heur\ T$;
$RETURN$ (*if* $get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\ U$ *then* $extract\text{-}model\text{-}of\text{-}state\text{-}stat\ U$
else $extract\text{-}state\text{-}stat\ U$)
}
}
}
}›

lemma *fref-to-Down-unRET-uncurry0-SPEC*:
assumes ‹$(\lambda\text{-}.\ (f), \lambda\text{-}.\ (RETURN\ g)) \in [P]_f\ unit\text{-}rel \rightarrow \langle B\rangle nres\text{-}rel$› **and** ‹$P\ ()$›
shows ‹$f \leq SPEC\ (\lambda c.\ (c, g) \in B)$›
⟨*proof*⟩

lemma *fref-to-Down-unRET-SPEC*:
assumes ‹$(f, RETURN\ o\ g) \in [P]_f\ A \rightarrow \langle B\rangle nres\text{-}rel$› **and**
‹$P\ y$› **and**
‹$(x, y) \in A$›
shows ‹$f\ x \leq SPEC\ (\lambda c.\ (c, g\ y) \in B)$›
⟨*proof*⟩

lemma *fref-to-Down-unRET-curry-SPEC*:
  assumes ‹$(uncurry\ f,\ uncurry\ (RETURN\ oo\ g)) \in [P]_f\ A \rightarrow \langle B\rangle nres\text{-}rel$› **and**
    ‹$P\ (x,\ y)$› **and**
    ‹$((x',\ y'),\ (x,\ y)) \in A$›
  shows ‹$f\ x'\ y' \leq SPEC\ (\lambda c.\ (c,\ g\ x\ y) \in B)$›
⟨*proof*⟩

lemma *all-lits-of-mm-empty-iff*: ‹$all\text{-}lits\text{-}of\text{-}mm\ A = \{\#\} \longleftrightarrow (\forall\ C \in\#\ A.\ C = \{\#\})$›
  ⟨*proof*⟩

lemma *all-lits-of-mm-extract-atms-clss*:
  ‹$L \in\#\ (all\text{-}lits\text{-}of\text{-}mm\ (mset\ `\#\ mset\ CS)) \longleftrightarrow atm\text{-}of\ L \in extract\text{-}atms\text{-}clss\ CS\ \{\}$›
  ⟨*proof*⟩

lemma *IsaSAT-heur-alt-def*:
  ‹*IsaSAT-heur opts CS = do*{
    $ASSERT(isasat\text{-}input\text{-}bounded\ (mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\})))$;
    $ASSERT(\forall\ C{\in}set\ CS.\ \forall\ L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint\text{-}max)$;
    let $\mathcal{A}_{in}{}' = mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\})$;
    $ASSERT(isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}{}')$;
    $ASSERT(distinct\text{-}mset\ \mathcal{A}_{in}{}')$;
    let $\mathcal{A}_{in}{}'' = virtual\text{-}copy\ \mathcal{A}_{in}{}'$;
    let $b = opts\text{-}unbounded\text{-}mode\ opts$;
    if b
    then do {
        $S \leftarrow init\text{-}state\text{-}wl\text{-}heur\ \mathcal{A}_{in}{}'$;
        $(T{::}twl\text{-}st\text{-}wl\text{-}heur\text{-}init) \leftarrow\ init\text{-}dt\text{-}wl\text{-}heur\ True\ CS\ S$;
        $T \leftarrow rewatch\text{-}heur\text{-}st\ T$;
        let $T = convert\text{-}state\ \mathcal{A}_{in}{}''\ T$;
        if $\neg get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\text{-}init\ T$
        then $RETURN\ (empty\text{-}init\text{-}code)$
        else if $CS = []$ *then* $empty\text{-}conflict\text{-}code$
        else do {
          $ASSERT(\mathcal{A}_{in}{}'' \neq \{\#\})$;
          $ASSERT(isasat\text{-}input\text{-}bounded\text{-}nempty\ \mathcal{A}_{in}{}'')$;
           $ASSERT((\lambda(M',\ N',\ D',\ Q',\ W',\ ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove),\ \varphi,\ clvls).$
$fst\text{-}As \neq None\ \wedge$
            $lst\text{-}As \neq None)\ T)$;
          $T \leftarrow finalise\text{-}init\text{-}code\ opts\ (T{::}twl\text{-}st\text{-}wl\text{-}heur\text{-}init)$;
          $U \leftarrow cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}wl\text{-}heur\ T$;
          $RETURN$ (*if* $get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\ U$ *then* $extract\text{-}model\text{-}of\text{-}state\text{-}stat\ U$
            else $extract\text{-}state\text{-}stat\ U$)
        }
    }
    else do {
        $S \leftarrow init\text{-}state\text{-}wl\text{-}heur\ \mathcal{A}_{in}{}'$;
        $(T{::}twl\text{-}st\text{-}wl\text{-}heur\text{-}init) \leftarrow init\text{-}dt\text{-}wl\text{-}heur\ False\ CS\ S$;
        $failed \leftarrow RETURN\ (is\text{-}failed\text{-}heur\text{-}init\ T \vee \neg isasat\text{-}fast\text{-}init\ T)$;
        if failed then do {
          $S \leftarrow init\text{-}state\text{-}wl\text{-}heur\ \mathcal{A}_{in}{}'$;
         $(T{::}twl\text{-}st\text{-}wl\text{-}heur\text{-}init) \leftarrow init\text{-}dt\text{-}wl\text{-}heur\ True\ CS\ S$;
         $T \leftarrow rewatch\text{-}heur\text{-}st\ T$;
         let $T = convert\text{-}state\ \mathcal{A}_{in}{}''\ T$;
         if $\neg get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\text{-}init\ T$
         then $RETURN\ (empty\text{-}init\text{-}code)$

else if $CS = []$ *then empty-conflict-code*
else do {
$ASSERT(\mathcal{A}_{in}'' \neq \{\#\})$;
$ASSERT(\textit{isasat-input-bounded-nempty}\ \mathcal{A}_{in}'')$;
$ASSERT((\lambda(M', N', D', Q', W', ((ns, m, \textit{fst-As}, \textit{lst-As}, \textit{next-search}), \textit{to-remove}), \varphi, \textit{clvls}).$
$\textit{fst-As} \neq \textit{None} \wedge$
$\textit{lst-As} \neq \textit{None})\ T)$;
$T \leftarrow \textit{finalise-init-code opts}\ (T::\textit{twl-st-wl-heur-init})$;
$U \leftarrow \textit{cdcl-twl-stgy-restart-prog-wl-heur}\ T$;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else do {
let $T = \textit{convert-state}\ \mathcal{A}_{in}''\ T$;
if $\neg$*get-conflict-wl-is-None-heur-init T*
then RETURN (*empty-init-code*)
else if $CS = []$ *then empty-conflict-code*
else do {
$ASSERT(\mathcal{A}_{in}'' \neq \{\#\})$;
$ASSERT(\textit{isasat-input-bounded-nempty}\ \mathcal{A}_{in}'')$;
$ASSERT((\lambda(M', N', D', Q', W', ((ns, m, \textit{fst-As}, \textit{lst-As}, \textit{next-search}), \textit{to-remove}), \varphi, \textit{clvls}).$
$\textit{fst-As} \neq \textit{None} \wedge$
$\textit{lst-As} \neq \textit{None})\ T)$;
ASSERT(*rewatch-heur-st-fast-pre T*);
$T \leftarrow$ *rewatch-heur-st-fast T*;
ASSERT(*isasat-fast-init T*);
$T \leftarrow \textit{finalise-init-code opts}\ (T::\textit{twl-st-wl-heur-init})$;
ASSERT(*isasat-fast T*);
$U \leftarrow$ *cdcl-twl-stgy-restart-prog-early-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
}
}›
⟨*proof*⟩

lemma *rewatch-heur-st-rewatch-st*:
assumes
UV: ‹(U, V)
$\in \textit{twl-st-heur-parsing-no-WL}\ (\textit{mset-set}\ (\textit{extract-atms-clss}\ CS\ \{\}))\ \textit{True}\ O$
$\{(S, T).\ S = \textit{remove-watched}\ T \wedge \textit{get-watched-wl}\ (\textit{fst}\ T) = (\lambda\text{-}.\ [])\}$›
shows ‹$\textit{rewatch-heur-st}\ U \leq$
$\Downarrow(\{(S,T).\ (S, T) \in \textit{twl-st-heur-parsing}\ (\textit{mset-set}\ (\textit{extract-atms-clss}\ CS\ \{\}))\ \textit{True} \wedge$
$\textit{get-clauses-wl-heur-init}\ S = \textit{get-clauses-wl-heur-init}\ U \wedge$
$\textit{get-conflict-wl-heur-init}\ S = \textit{get-conflict-wl-heur-init}\ U \wedge$
$\textit{get-clauses-wl}\ (\textit{fst}\ T) = \textit{get-clauses-wl}\ (\textit{fst}\ V) \wedge$
$\textit{get-conflict-wl}\ (\textit{fst}\ T) = \textit{get-conflict-wl}\ (\textit{fst}\ V) \wedge$
$\textit{get-unit-clauses-wl}\ (\textit{fst}\ T) = \textit{get-unit-clauses-wl}\ (\textit{fst}\ V)\}\ O\ \{(S, T).\ S = (T, \{\#\})\})$
$(\textit{rewatch-st}\ (\textit{from-init-state}\ V))$›
⟨*proof*⟩

lemma *rewatch-heur-st-rewatch-st2*:
assumes

T: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
shows ‹*rewatch-heur-st-fast*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
≤ ⇓ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})})
(*rewatch-st* (*from-init-state V*))›
⟨*proof*⟩

lemma *rewatch-heur-st-rewatch-st3*:
assumes
T: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
failed: ‹¬*is-failed-heur-init U*›
shows ‹*rewatch-heur-st-fast*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
≤ ⇓ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})})
(*rewatch-st* (*from-init-state V*))›
⟨*proof*⟩

lemma *IsaSAT-heur-IsaSAT*:
‹*IsaSAT-heur b CS* ≤ ⇓{((*M*, *stats*), *M'*). *M* = *map-option rev M'*} (*IsaSAT CS*)›
⟨*proof*⟩

definition *model-stat-rel* **where**
‹*model-stat-rel* = {((*M'*, *s*), *M*). *map-option rev M* = *M'*}›

lemma *nat-of-uint32-max*:
‹*max* (*nat-of-uint32 a*) (*nat-of-uint32 b*) = *nat-of-uint32* (*max a b*)› **for** *a b*
⟨*proof*⟩

lemma *max-0L-uint32*[*simp*]: ‹*max* (*0*::*uint32*) *a* = *a*›
⟨*proof*⟩

definition *length-get-clauses-wl-heur-init* **where**
‹*length-get-clauses-wl-heur-init S* = *length* (*get-clauses-wl-heur-init S*)›

lemma *length-get-clauses-wl-heur-init-alt-def*:
‹*RETURN o length-get-clauses-wl-heur-init* = (λ(-, *N*,-). *RETURN* (*length N*))›
⟨*proof*⟩

definition *model-if-satisfiable* :: ‹*nat clauses* ⇒ *nat literal list option nres*› **where**

‹*model-if-satisfiable CS* = *SPEC* (*λM*.
if satisfiable (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*)›

definition *SAT′* :: ‹*nat clauses* ⇒ *nat literal list option nres*› **where**
‹*SAT′ CS* = *do* {
T ← *SAT CS*;
RETURN(*if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*)
}
›

lemma *SAT-model-if-satisfiable*:
‹(*SAT′*, *model-if-satisfiable*) ∈ [*λCS*. (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id*→ ⟨*Id*⟩*nres-rel*›
(**is** ‹- ∈[*λCS*. *?P CS*]$_f$ *Id* → -›)
⟨*proof*⟩

lemma *SAT-model-if-satisfiable′*:
‹(*uncurry* (λ-. *SAT′*), *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id* ×$_r$ *Id*→ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *SAT-l′* **where**
‹*SAT-l′ CS* = *do*{
S ← *SAT-l CS*;
RETURN (*if get-conflict-l S* = *None then Some* (*map lit-of* (*get-trail-l S*)) *else None*)
}›

definition *SAT0′* **where**
‹*SAT0′ CS* = *do*{
S ← *SAT0 CS*;
RETURN (*if get-conflict S* = *None then Some* (*map lit-of* (*get-trail S*)) *else None*)
}›

lemma *twl-st-l-map-lit-of*[*twl-st-l*, *simp*]:
‹(*S*, *T*) ∈ *twl-st-l b* ⟹ *map lit-of* (*get-trail-l S*) = *map lit-of* (*get-trail T*)›
⟨*proof*⟩

lemma *ISASAT-SAT-l′*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*› **and**
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT CS* ≤ ⇓ *Id* (*SAT-l′ CS*)›
⟨*proof*⟩

lemma *SAT-l′-SAT0′*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*SAT-l′ CS* ≤ ⇓ *Id* (*SAT0′ CS*)›
⟨*proof*⟩

lemma *SAT0′-SAT′*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*SAT0′ CS* ≤ ⇓ *Id* (*SAT′* (*mset* ‘# *mset CS*))›
⟨*proof*⟩

lemma *IsaSAT-heur-model-if-sat*:
  assumes ‹$\forall$ *C* $\in$# *mset* ‘# *mset CS*. *distinct-mset C*› **and**
    ‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*$\in$*set CS*. *atm-of* ‘ *set C*))›
  shows ‹*IsaSAT-heur opts CS* $\leq$ $\Downarrow$ *model-stat-rel* (*model-if-satisfiable* (*mset* ‘# *mset CS*))›
  ⟨*proof*⟩

lemma *IsaSAT-heur-model-if-sat'*: ‹(*uncurry IsaSAT-heur*, *uncurry* (λ-. *model-if-satisfiable*)) $\in$
  [λ(-, *CS*). ($\forall$ *C* $\in$# *CS*. *distinct-mset C*) $\wedge$
    ($\forall$ *C*$\in$#*CS*. $\forall$ *L*$\in$#*C*. *nat-of-lit L* $\leq$ *uint-max*)]$_f$
    Id $\times_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* $\rightarrow$ ⟨*model-stat-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *IsaSAT-bounded-heur* :: ‹*opts* $\Rightarrow$ *nat clause-l list* $\Rightarrow$ (*bool* $\times$ (*nat literal list option* $\times$ *stats*)) *nres*› **where**
  ‹*IsaSAT-bounded-heur opts CS* = *do*{
    ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
    ASSERT($\forall$ *C*$\in$*set CS*. $\forall$ *L*$\in$*set C*. *nat-of-lit L* $\leq$ *uint-max*);
    let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
    ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
    ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
    let $\mathcal{A}_{in}''$ = *virtual-copy* $\mathcal{A}_{in}'$;
    let b = *opts-unbounded-mode opts*;
    S $\leftarrow$ *init-state-wl-heur-fast* $\mathcal{A}_{in}'$;
    (*T*::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur False CS S*;
    let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
    if $\neg$*get-conflict-wl-is-None-heur-init T*
    then RETURN (*True*, *empty-init-code*)
    else if CS = [] *then do* {*stat* $\leftarrow$ *empty-conflict-code*; *RETURN* (*True*, *stat*)}
    else
    if isasat-fast-init T $\wedge$ $\neg$*is-failed-heur-init T*
    then do {
      ASSERT($\mathcal{A}_{in}''$ $\neq$ {#});
      ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
      - $\leftarrow$ *isasat-information-banner T*;
      ASSERT((λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*). *fst-As* $\neq$ *None* $\wedge$
        lst-As $\neq$ *None*) *T*);
      ASSERT(*rewatch-heur-st-fast-pre T*);
      T $\leftarrow$ *rewatch-heur-st-fast T*;
      ASSERT(*isasat-fast-init T*);
      T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
      ASSERT(*isasat-fast T*);
      (*b*, *U*) $\leftarrow$ *cdcl-twl-stgy-restart-prog-bounded-wl-heur T*;
      RETURN (*b*, *if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
        else extract-state-stat U)
    } *else RETURN* (*False*, *empty-init-code*)
  }›

end
theory *IsaSAT-SML*
  imports *Watched-Literals.WB-Word-Assn IsaSAT Version IsaSAT-Restart-SML*
    IsaSAT-Initialisation-SML Version
begin

lemma [*code*]:

‹nth-aa64-i32-u64 xs x L = do {
    x ← nth-u-code xs x;
    arl64-get x L ≫= return
  }›
⟨proof⟩

lemma [code]: ‹uint32-max-uint32 = 4294967295›
⟨proof⟩

abbreviation *model-stat-assn* **where**
‹model-stat-assn ≡ option-assn (arl-assn unat-lit-assn) *a stats-assn›

abbreviation *lits-with-max-assn* **where**
‹lits-with-max-assn ≡ hr-comp (arl-assn uint32-nat-assn *a uint32-nat-assn) lits-with-max-rel›
lemma *lits-with-max-assn-alt-def*: ‹lits-with-max-assn = hr-comp (arl-assn uint32-assn *a uint32-assn)
    (lits-with-max-rel O ⟨uint32-nat-rel⟩IsaSAT-Initialisation.mset-rel)›
⟨proof⟩

lemma *init-state-wl-D′-code-isasat*: ‹(hr-comp isasat-init-assn
  ($Id \times_f$
   ($Id \times_f$
    ($Id \times_f$
     ($nat\text{-}rel \times_f$
      (⟨⟨Id⟩list-rel⟩list-rel $\times_f$
       ($Id \times_f$ (⟨bool-rel⟩list-rel $\times_f$ ($nat\text{-}rel \times_f$ ($Id \times_f$ ($Id \times_f$ Id)))))))))) = isasat-init-assn›
⟨proof⟩

lemma *list-assn-list-mset-rel-clauses-l-assn*:
‹(hr-comp (list-assn (list-assn unat-lit-assn)) (list-mset-rel O ⟨list-mset-rel⟩IsaSAT-Initialisation.mset-rel))
xs xs′
    = clauses-l-assn xs xs′›
⟨proof⟩

definition *get-trail-wl-code* :: ‹- ⇒ uint32 array-list option × stats› **where**
‹get-trail-wl-code = (λ((M, -), -, -, -, - ,- ,- ,-, -, -, -, stat, -). (Some M, stat))›

definition *get-stats-code* :: ‹- ⇒ uint32 array-list option × stats› **where**
‹get-stats-code = (λ((M, -), -, -, -, - ,- ,- ,-, -, -, -, stat, -). (None, stat))›

definition *model-assn* **where**
‹model-assn = hr-comp model-stat-assn model-stat-rel›

lemma *extract-model-of-state-stat-hnr*[sepref-fr-rules]:
‹(return o get-trail-wl-code, RETURN o extract-model-of-state-stat) ∈ isasat-unbounded-assn^d $\rightarrow_a$
    model-stat-assn›
⟨proof⟩

lemma *get-stats-code*[sepref-fr-rules]:
‹(return o get-stats-code, RETURN o extract-state-stat) ∈ isasat-unbounded-assn^d $\rightarrow_a$
    model-stat-assn›
⟨proof⟩

lemma *convert-state-hnr*:
‹(uncurry (return oo (λ- S. S)), uncurry (RETURN oo convert-state))

$\in$ *ghost-assn*^{k} $*_a$ (*isasat-init-assn*)^d $\rightarrow_a$
isasat-init-assn›
⟨*proof*⟩

lemma *convert-state-hnr-unb*:
‹(*uncurry* (*return oo* (λ- *S*. *S*)), *uncurry* (*RETURN oo convert-state*))
$\in$ *ghost-assn*^{k} $*_a$ (*isasat-init-unbounded-assn*)^d $\rightarrow_a$
isasat-init-unbounded-assn›
⟨*proof*⟩

lemma *IsaSAT-use-fast-mode*[*sepref-fr-rules*]:
‹(*uncurry0* (*return IsaSAT-use-fast-mode*), *uncurry0* (*RETURN IsaSAT-use-fast-mode*))
$\in$ *unit-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

sepref-definition *empty-conflict-code′*
is ‹*uncurry0* (*empty-conflict-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-assn*›
⟨*proof*⟩

declare *empty-conflict-code′.refine*[*sepref-fr-rules*]

sepref-definition *empty-init-code′*
is ‹*uncurry0* (*RETURN empty-init-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-assn*›
⟨*proof*⟩

declare *empty-init-code′.refine*[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-full*

declare *extract-model-of-state-stat-hnr*[*sepref-fr-rules*]
sepref-register *to-init-state from-init-state get-conflict-wl-is-None-init extract-stats*
init-dt-wl-heur

declare
get-stats-code[*sepref-fr-rules*]

lemma *isasat-fast-init-alt-def*:
‹*RETURN o isasat-fast-init* = (λ(*M*, *N*, -). *RETURN* (*length N* ≤ *isasat-fast-bound*))›
⟨*proof*⟩

sepref-definition *isasat-fast-init-code*
is ‹*RETURN o isasat-fast-init*›
:: ‹*isasat-init-assn*^{k} $\rightarrow_a$ *bool-assn*›
⟨*proof*⟩

declare *isasat-fast-init-code.refine*[*sepref-fr-rules*]

declare *convert-state-hnr*[*sepref-fr-rules*]
convert-state-hnr-unb[*sepref-fr-rules*]

sepref-register
cdcl-twl-stgy-restart-prog-wl-heur

declare *init-state-wl-D′-code.refine*[*FCOMP init-state-wl-D′*[*unfolded convert-fref*],
unfolded lits-with-max-assn-alt-def[*symmetric*] *init-state-wl-heur-fast-def*[*symmetric*],
unfolded init-state-wl-D′-code-isasat, *sepref-fr-rules*]

lemma *init-state-wl-D′-code-isasat-unb*: ‹(*hr-comp isasat-init-unbounded-assn*
(*Id* $\times_f$
(*Id* $\times_f$
(*Id* $\times_f$
(*nat-rel* $\times_f$
(⟨⟨*Id*⟩*list-rel*⟩*list-rel* $\times_f$
(*Id* $\times_f$ (⟨*bool-rel*⟩*list-rel* $\times_f$ (*nat-rel* $\times_f$ (*Id* $\times_f$ (*Id* $\times_f$ *Id*)))))))))) = *isasat-init-unbounded-assn*›
⟨*proof*⟩

lemma *arena-assn-alt-def*: ‹*arl-assn* (*pure* (*uint32-nat-rel O arena-el-rel*)) = *arena-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]: ‹(*init-state-wl-D′-code-unb*, *init-state-wl-heur*)
∈ [λ*x*. *distinct-mset x* ∧
(∀ *L*∈#$\mathcal{L}_{all}$ *x*.
nat-of-lit L
≤ *uint-max*)]$_a$ *IsaSAT-SML.lits-with-max-assn*^{d} → *isasat-init-unbounded-assn*›
⟨*proof*⟩

sepref-definition *isasat-init-fast-slow-code*
is ‹*isasat-init-fast-slow*›
:: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
⟨*proof*⟩

declare *isasat-init-fast-slow-code.refine*[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-unb*

fun (**in** −) *is-failed-heur-init-code* :: ‹- ⇒ *bool*› **where**
‹*is-failed-heur-init-code* (-, -, -, -, -, -, -, -, -, -, -, *failed*) = *failed*›

lemma *is-failed-heur-init-code*[*sepref-fr-rules*]:
‹(*return o is-failed-heur-init-code*, *RETURN o is-failed-heur-init*) ∈ *isasat-init-assn*^{k} $\rightarrow_a$
bool-assn›
⟨*proof*⟩

declare *init-dt-wl-heur-code-unb.refine*[*sepref-fr-rules*]

sepref-definition *IsaSAT-code*
is ‹*uncurry IsaSAT-heur*›
:: ‹*opts-assn*^{d} $*_a$ (*list-assn* (*list-assn unat-lit-assn*))^k $\rightarrow_a$ *model-stat-assn*›
⟨*proof*⟩

theorem *IsaSAT-full-correctness*:
‹(*uncurry IsaSAT-code*, *uncurry* (λ-. *model-if-satisfiable*))
∈ [λ(-, *a*). *Multiset.Ball a distinct-mset* ∧
(∀ *C*∈#*a*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint-max*)]$_a$ *opts-assn*^{d} $*_a$ *clauses-l-assn*^{k} → *model-assn*›
⟨*proof*⟩

sepref-definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur-fast-code*

is ‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur*›
:: ‹$[\lambda S.\ isasat\text{-}fast\ S]_a$ *isasat-bounded-assn*^{d} → *bool-assn* ∗*a* *isasat-bounded-assn*›
⟨*proof*⟩

declare *cdcl-twl-stgy-restart-prog-bounded-wl-heur-fast-code.refine*[*sepref-fr-rules*]

definition *get-trail-wl-code-b* :: ‹- ⇒ *uint32 array-list32 option* × *stats*› **where**
‹*get-trail-wl-code-b* = (λ((*M*, -), -, -, -, - ,- ,- ,-, -, -, -, *stat*, -). (*Some M*, *stat*))›

abbreviation *model-stat-fast-assn* **where**
‹*model-stat-fast-assn* ≡ *option-assn* (*arl32-assn unat-lit-assn*) ∗*a* *stats-assn*›

lemma *extract-model-of-state-stat-bounded-hnr*[*sepref-fr-rules*]:
‹(*return o get-trail-wl-code-b*, *RETURN o extract-model-of-state-stat*) ∈ *isasat-bounded-assn*^{d} $\rightarrow_a$ *model-stat-fast-assn*›
⟨*proof*⟩

sepref-definition *empty-conflict-fast-code′*
is ‹*uncurry0* (*empty-conflict-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-fast-assn*›
⟨*proof*⟩

declare *empty-conflict-fast-code′.refine*[*sepref-fr-rules*]

sepref-definition *empty-init-fast-code′*
is ‹*uncurry0* (*RETURN empty-init-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-fast-assn*›
⟨*proof*⟩

declare *empty-init-fast-code′.refine*[*sepref-fr-rules*]

definition *get-stats-fast-code* :: ‹- ⇒ *uint32 array-list32 option* × *stats*› **where**
‹*get-stats-fast-code* = (λ((*M*, -), -, -, -, - ,- ,- ,-, -, -, -, *stat*, -). (*None*, *stat*))›

lemma *get-stats-b-code*[*sepref-fr-rules*]:
‹(*return o get-stats-fast-code*, *RETURN o extract-state-stat*) ∈ *isasat-bounded-assn*^{d} $\rightarrow_a$ *model-stat-fast-assn*›
⟨*proof*⟩

sepref-definition *IsaSAT-bounded-code*
is ‹*uncurry IsaSAT-bounded-heur*›
:: ‹*opts-assn*^{d} $*_a$ (*list-assn* (*list-assn unat-lit-assn*))^k $\rightarrow_a$ *bool-assn* ∗*a* *model-stat-fast-assn*›
⟨*proof*⟩

Code Export

definition *nth-u-code′* **where**
[*symmetric*, *code*]: ‹*nth-u-code′* = *nth-u-code*›

code-printing constant *nth-u-code′* ⇀ (*SML*) (*fn*/ ()/ =>/ *Array.sub*/ ((-),/ *Word32.toInt* (-)))

definition *nth-u64-code′* **where**
[*symmetric*, *code*]: ‹*nth-u64-code′* = *nth-u64-code*›

code-printing constant *nth-u64-code′* ⇀ (*SML*) (*fn*/ ()/ =>/ *Array.sub*/ ((-),/ *Word64.toInt* ((-))))

definition *heap-array-set′-u′* **where**
[*symmetric*, *code*]: ‹*heap-array-set′-u′* = *heap-array-set′-u*›

code-printing constant *heap-array-set′-u′* ⇀
(*SML*) (*fn*/ ()/ =>/ *Array.update*/ ((-),/ (*Word32.toInt* (-)),/ (-)))

definition *heap-array-set′-u64′* **where**
[*symmetric*, *code*]: ‹*heap-array-set′-u64′* = *heap-array-set′-u64*›

code-printing constant *heap-array-set′-u64′* ⇀
(*SML*) (*fn*/ ()/ =>/ *Array.update*/ ((-),/ (*Word64.toInt* (-)),/ (-)))

definition *length-u-code′* **where**
[*symmetric*, *code*]: ‹*length-u-code′* = *length-u-code*›

code-printing constant *length-u-code′* ⇀ (*SML-imp*) (*fn*/ ()/ =>/ *Word32.fromInt* (*Array.length* (-)))

definition *length-aa-u-code′* **where**
[*symmetric*, *code*]: ‹*length-aa-u-code′* = *length-aa-u-code*›

code-printing constant *length-aa-u-code′* ⇀ (*SML-imp*)
(*fn*/ ()/ =>/ *Word32.fromInt* (*Array.length* (*Array.sub*/ ((*fn*/ (*a*,*b*)/ =>/ *a*) (-),/ *IntInf.toInt* (*integer′-of′-nat* (-))))))

definition *nth-raa-i-u64′* **where**
[*symmetric*, *code*]: ‹*nth-raa-i-u64′* = *nth-raa-i-u64*›

code-printing constant *nth-raa-i-u64′* ⇀ (*SML-imp*)
(*fn*/ ()/ =>/ *Array.sub* (*Array.sub*/ ((*fn*/ (*a*,*b*)/ =>/ *a*) (-),/ *IntInf.toInt* (*integer′-of′-nat* (-))), *Word64.toInt* (-)))

definition *length-u64-code′* **where**
[*symmetric*, *code*]: ‹*length-u64-code′* = *length-u64-code*›

code-printing constant *length-u64-code′* ⇀ (*SML-imp*)
(*fn*/ ()/ =>/ *Uint64.fromFixedInt* (*Array.length* (-)))

code-printing constant *arl-get-u* ⇀ (*SML*) (*fn*/ ()/ =>/ *Array.sub*/ ((*fn*/ (*a*,*b*)/ =>/ *a*) ((-)),/ *Word32.toInt* ((-))))

definition *uint32-of-uint64′* **where**
[*symmetric*, *code*]: ‹*uint32-of-uint64′* = *uint32-of-uint64*›

code-printing constant *uint32-of-uint64′* ⇀ (*SML-imp*)
Word32.fromLargeWord (-)

lemma *arl-set-u64-code*[*code*]: ‹*arl-set-u64 a i x* =
Array-upd-u64 i x (*fst a*) ⤜ (*λb. return* (*b*, (*snd a*)))›
⟨*proof*⟩

lemma *arl-set-u-code*[*code*]: ‹*arl-set-u a i x* =
Array-upd-u i x (*fst a*) ⤜ (*λb. return* (*b*, (*snd a*)))›
⟨*proof*⟩

definition *arl-get-u64 ′* **where**
[*symmetric*, *code*]: ‹*arl-get-u64 ′* = *arl-get-u64*›

code-printing constant *arl-get-u64 ′* ⇀ (*SML*)
(*fn*/ ()/ =>/ *Array.sub*/ ((*fn* (*a*,*b*) => *a*) (-),/ *Word64.toInt* (-)))

code-printing code-module *Uint64* ⇀ (*SML*) ‹(∗ *Test that words can handle numbers between 0 and 63* ∗)

```
val - = if 6 <= Word.wordSize then () else raise (Fail (wordSize less than 6));

structure Uint64 : sig
  eqtype uint64;
  val zero : uint64;
  val one : uint64;
  val fromInt : IntInf.int −> uint64;
  val toInt : uint64 −> IntInf.int;
  val toFixedInt : uint64 −> Int.int;
  val toLarge : uint64 −> LargeWord.word;
  val fromLarge : LargeWord.word −> uint64
  val fromFixedInt : Int.int −> uint64
  val plus : uint64 −> uint64 −> uint64;
  val minus : uint64 −> uint64 −> uint64;
  val times : uint64 −> uint64 −> uint64;
  val divide : uint64 −> uint64 −> uint64;
  val modulus : uint64 −> uint64 −> uint64;
  val negate : uint64 −> uint64;
  val less-eq : uint64 −> uint64 −> bool;
  val less : uint64 −> uint64 −> bool;
  val notb : uint64 −> uint64;
  val andb : uint64 −> uint64 −> uint64;
  val orb : uint64 −> uint64 −> uint64;
  val xorb : uint64 −> uint64 −> uint64;
  val shiftl : uint64 −> IntInf.int −> uint64;
  val shiftr : uint64 −> IntInf.int −> uint64;
  val shiftr-signed : uint64 −> IntInf.int −> uint64;
  val set-bit : uint64 −> IntInf.int −> bool −> uint64;
  val test-bit : uint64 −> IntInf.int −> bool;
end = struct

type uint64 = Word64.word;

val zero = (0wx0 : uint64);

val one = (0wx1 : uint64);

fun fromInt x = Word64.fromLargeInt (IntInf.toLarge x);

fun toInt x = IntInf.fromLarge (Word64.toLargeInt x);

fun toFixedInt x = Word64.toInt x;

fun fromLarge x = Word64.fromLarge x;

fun fromFixedInt x = Word64.fromInt x;
```

fun toLarge x = Word64.toLarge x;

fun plus x y = Word64.+(x, y);

fun minus x y = Word64.−(x, y);

fun negate x = Word64.~(x);

fun times x y = Word64.∗(x, y);

fun divide x y = Word64.div(x, y);

fun modulus x y = Word64.mod(x, y);

fun less-eq x y = Word64.<=(x, y);

fun less x y = Word64.<(x, y);

fun set-bit x n b =
  let val mask = Word64.<< (0wx1, Word.fromLargeInt (IntInf.toLarge n))
  in if b then Word64.orb (x, mask)
    else Word64.andb (x, Word64.notb mask)
  end

fun shiftl x n =
  Word64.<< (x, Word.fromLargeInt (IntInf.toLarge n))

fun shiftr x n =
  Word64.>> (x, Word.fromLargeInt (IntInf.toLarge n))

fun shiftr-signed x n =
  Word64.~>> (x, Word.fromLargeInt (IntInf.toLarge n))

fun test-bit x n =
  Word64.andb (x, Word64.<< (0wx1, Word.fromLargeInt (IntInf.toLarge n))) <> Word64.fromInt 0

val notb = Word64.notb

fun andb x y = Word64.andb(x, y);

fun orb x y = Word64.orb(x, y);

fun xorb x y = Word64.xorb(x, y);

end (∗struct Uint64∗)
⟩
export-code *IsaSAT-code* **checking** *SML-imp*

code-printing constant — print with line break
  println-string ⇀ (*SML*) *ignore/* (*print/* ((-) ^ \n))

export-code *IsaSAT-code*
  int-of-integer
  integer-of-int
  integer-of-nat

nat-of-integer
uint32-of-nat
Version.version
in *SML-imp* **module-name** *SAT-Solver* **file-prefix** *IsaSAT-solver*

external-file ‹*code/Unsynchronized.sml*›
external-file ‹*code/IsaSAT.mlb*›
external-file ‹*code/IsaSAT.sml*›
external-file ‹*code/dimacs-parser.sml*›

compile-generated-files -
external-files
‹*code/IsaSAT.mlb*›
‹*code/Unsynchronized.sml*›
‹*code/IsaSAT.sml*›
‹*code/dimacs-parser.sml*›
where ‹*fn dir =>*
let
val exec = Generated-Files.execute (Path.append dir (Path.basic code));
val - = exec ‹rename file› mv IsaSAT-solver.ML IsaSAT-solver.sml
val - =
exec ‹Copy files›
(cp IsaSAT-solver.sml ^
((File.bash-path ***path*** *‹$ISAFOL›) ^ /Weidenbach-Book/code/IsaSAT-solver.sml));*
val - =
exec ‹Compilation›
(File.bash-path ***path*** *‹$ISABELLE-MLTON› ^*
−const 'MLton.safe false' −verbose 1 −default−type int64 −output IsaSAT ^
−codegen native −inline 700 −cc−opt −O3 IsaSAT.mlb);
val - =
exec ‹Copy binary files›
(cp IsaSAT ^
File.bash-path ***path*** *‹$ISAFOL› ^ /Weidenbach-Book/code/);*
in () end›

export-code *IsaSAT-bounded-code*
int-of-integer
integer-of-int
integer-of-nat
nat-of-integer
uint32-of-nat
Version.version
in *SML-imp* **module-name** *SAT-Solver* **file-prefix** *IsaSAT-solver-bounded*

compile-generated-files -
external-files
‹*code/IsaSAT-bounded.mlb*›
‹*code/Unsynchronized.sml*›
‹*code/IsaSAT-bounded.sml*›
‹*code/dimacs-parser.sml*›
where ‹*fn dir =>*
let
val exec = Generated-Files.execute (Path.append dir (Path.basic code));
val - = exec ‹rename file› mv IsaSAT-solver-bounded.ML IsaSAT-solver-bounded.sml

```
    val - =
      exec ‹Copy files›
        (cp IsaSAT-solver-bounded.sml ^
  ((File.bash-path path ‹$ISAFOL›) ^ /Weidenbach-Book/code/IsaSAT-solver-bounded.sml));
    val - =
      exec ‹Compilation›
        (File.bash-path path ‹$ISABELLE-MLTON› ^
           −const 'MLton.safe false' −verbose 1 −default−type int64 −output IsaSAT-bounded ^
           −codegen native −inline 700 −cc−opt −O3 IsaSAT-bounded.mlb);
    val - =
      exec ‹Copy binary files›
        (cp IsaSAT-bounded ^
          File.bash-path path ‹$ISAFOL› ^ /Weidenbach-Book/code/);
  in () end›
```

end