Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

0.0.1 Literals as Natural Numbers 4
0.0.2 Atoms with bound 6
0.1 Code Generation 13
0.1.1 Literals as Natural Numbers 13
0.1.2 State Conversion 14
0.1.3 Code Generation 14
0.1.4 The memory representation: Arenas 18
0.1.5 Declaration of some Operators and Implementation 75
0.1.6 Length of the trail 108
0.1.7 Variable-Move-to-Front 120
0.1.8 Phase saving 174
0.1.9 Code Generation 262
0.1.10 More theorems 274
0.1.11 Shared Code Equations 276
0.1.12 Rewatch 277
0.1.13 Rewatch 277
0.1.14 Fast to slow conversion 281
0.1.15 More theorems 314
0.1.16 Propagations Step 316
0.1.17 Code generation for the VMTF decision heuristic and the trail 384
0.1.18 Backtrack 418
0.2 Code for the initialisation of the Data Structure 467
0.2.1 Initialisation of the state 467
0.2.2 Parsing 475
0.2.3 Extractions of the atoms in the state 488
0.2.4 Parsing 496
0.2.5 Conversion to normal state 500
0.2.6 Printing information about progress 572
0.2.7 Print Information for IsaSAT 573
0.2.8 Final code generation 697

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-fun-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-fun-rel* D = $\{(m, f).\ \forall (i, j) \in D.\ i <$ *length* $m \wedge (m\ !\ i, f\ j) \in R\}$›
unfolding *relAPP-def map-fun-rel-def-internal* **by** *auto*

0.0.1 Literals as Natural Numbers

Definition

lemma *Pos-div2-iff*:
‹*Pos* $((bb :: nat)$ *div* $2) = b \longleftrightarrow$ *is-pos* $b \wedge (bb = 2 *$ *atm-of* $b \vee bb = 2 *$ *atm-of* $b + 1)$›
by (*cases* b) *auto*
lemma *Neg-div2-iff*:
‹*Neg* $((bb :: nat)$ *div* $2) = b \longleftrightarrow$ *is-neg* $b \wedge (bb = 2 *$ *atm-of* $b \vee bb = 2 *$ *atm-of* $b + 1)$›
by (*cases* b) *auto*

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-of-lit* (*Pos* L) = $2*L$›
| ‹*nat-of-lit* (*Neg* L) = $2*L + 1$›

lemma *nat-of-lit-def*: ‹*nat-of-lit* L = (*if is-pos* L *then* $2 *$ *atm-of* L *else* $2 *$ *atm-of* $L + 1$)›
by (*cases* L) *auto*

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

lemma *lit-of-nat-nat-of-lit*[*simp*]: ‹*literal-of-nat* (*nat-of-lit* L) = L›
by (*cases* L) *auto*

lemma *nat-of-lit-lit-of-nat*[*simp*]: ‹*nat-of-lit* (*literal-of-nat* n) = n›
by *auto*

lemma *atm-of-lit-of-nat*: ‹*atm-of* (*literal-of-nat* n) = n *div* 2›
by *auto*

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-of-nat* n) = (*if even* n *then literal-of-nat* $(n+1)$ *else literal-of-nat* $(n-1)$)›
by (*auto elim*!: *oddE*)

lemma *literal-of-nat-literal-of-nat-eq*[*iff*]: ‹*literal-of-nat* x = *literal-of-nat* $xa \longleftrightarrow x = xa$›
by *auto presburger+*

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

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

⟨*unat-lit-rel* ≡ *uint32-nat-rel O nat-lit-rel*⟩

fun *pair-of-ann-lit* :: ⟨(′*a*, ′*b*) *ann-lit* ⇒ ′*a literal* × ′*b option*⟩ **where**
⟨*pair-of-ann-lit* (*Propagated L D*) = (*L*, *Some D*)⟩
| ⟨*pair-of-ann-lit* (*Decided L*) = (*L*, *None*)⟩

fun *ann-lit-of-pair* :: ⟨′*a literal* × ′*b option* ⇒ (′*a*, ′*b*) *ann-lit*⟩ **where**
⟨*ann-lit-of-pair* (*L*, *Some D*) = *Propagated L D*⟩
| ⟨*ann-lit-of-pair* (*L*, *None*) = *Decided L*⟩

lemma *ann-lit-of-pair-alt-def*:
⟨*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))⟩
by (*cases D*) *auto*

lemma *ann-lit-of-pair-pair-of-ann-lit*: ⟨*ann-lit-of-pair* (*pair-of-ann-lit L*) = *L*⟩
by (*cases L*) *auto*

lemma *pair-of-ann-lit-ann-lit-of-pair*: ⟨*pair-of-ann-lit* (*ann-lit-of-pair L*) = *L*⟩
by (*cases L*; *cases* ⟨*snd L*⟩) *auto*

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ⟨*literal-of-nat b* = *L* ⟷ *b* = *nat-of-lit L*⟩
by (*auto simp del*: *literal-of-nat.simps*)

lemma *nat-of-lit-eq-iff*[*iff*]: ⟨*nat-of-lit xa* = *nat-of-lit x* ⟷ *x* = *xa*⟩
apply (*cases x*; *cases xa*) **by** *auto presburger*+

definition *ann-lit-rel*:: ⟨(′*a* × *nat*) *set* ⇒ (′*b* × *nat option*) *set* ⇒
((′*a* × ′*b*) × (*nat*, *nat*) *ann-lit*) *set*⟩ **where**
ann-lit-rel-internal-def:
⟨*ann-lit-rel R R*′ = {(*a*, *b*). ∃ *c d*. (*fst a*, *c*) ∈ *R* ∧ (*snd a*, *d*) ∈ *R*′ ∧
b = *ann-lit-of-pair* (*literal-of-nat c*, *d*)}⟩

type-synonym *ann-lit-wl* = ⟨*uint32* × *nat option*⟩
type-synonym *ann-lits-wl* = ⟨*ann-lit-wl list*⟩
type-synonym *ann-lit-wl-fast* = ⟨*uint32* × *uint64 option*⟩
type-synonym *ann-lits-wl-fast* = ⟨*ann-lit-wl-fast list*⟩

definition *nat-ann-lit-rel* :: ⟨(*ann-lit-wl* × (*nat*, *nat*) *ann-lit*) *set*⟩ **where**
nat-ann-lit-rel-internal-def: ⟨*nat-ann-lit-rel* = ⟨*uint32-nat-rel*, ⟨*nat-rel*⟩*option-rel*⟩*ann-lit-rel*⟩

lemma *ann-lit-rel-def*:
⟨⟨*R*, *R*′⟩*ann-lit-rel* = {(*a*, *b*). ∃ *c d*. (*fst a*, *c*) ∈ *R* ∧ (*snd a*, *d*) ∈ *R*′ ∧
b = *ann-lit-of-pair* (*literal-of-nat c*, *d*)}⟩
unfolding *nat-ann-lit-rel-internal-def ann-lit-rel-internal-def relAPP-def* **..**

lemma *nat-ann-lit-rel-def*:
⟨*nat-ann-lit-rel* = {(*a*, *b*). *b* = *ann-lit-of-pair* ((λ(*a*,*b*). (*literal-of-nat* (*nat-of-uint32 a*), *b*)) *a*)}⟩
unfolding *nat-ann-lit-rel-internal-def ann-lit-rel-def*
apply (*auto simp*: *option-rel-def ex-disj-distrib uint32-nat-rel-def br-def*)
apply (*case-tac b*)
apply *auto*
apply (*case-tac b*)
apply *auto*
done

definition *nat-ann-lits-rel* :: ‹(*ann-lits-wl* × (*nat*, *nat*) *ann-lits*) *set*› **where**
‹*nat-ann-lits-rel* = ⟨*nat-ann-lit-rel*⟩*list-rel*›

lemma *nat-ann-lits-rel-Cons*[*iff*]:
‹(*x* # *xs*, *y* # *ys*) ∈ *nat-ann-lits-rel* ⟷ (*x*, *y*) ∈ *nat-ann-lit-rel* ∧ (*xs*, *ys*) ∈ *nat-ann-lits-rel*›
by (*auto simp*: *nat-ann-lits-rel-def*)

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 ≡ (λ*L*. (*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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$]$_f$ ((⟨*Id*⟩*map-fun-rel* D_0) $\times_r$ *nat-lit-rel*) →
⟨*nat-rel*⟩ *nres-rel*›
unfolding *length-ll-def length-ll-f-def*
by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def*
nat-lit-rel-def)

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*›)
proof −
define *D'* **where** ‹*D'* = D_0›
define $\mathcal{L}_{all}$*'* **where** ‹$\mathcal{L}_{all}$*'* = $\mathcal{L}_{all}$›
define *D''* **where** ‹*D''* = *mset-set* (*snd* ‘ *D'*)›
let *?f* = ‹(λ*L a*. *a*[*nat-of-lit L*:= *W L*])›
interpret *comp-fun-commute ?f*
apply *standard*
apply (*case-tac* ‹*y* = *x*›)
apply (*solves simp*)
apply (*intro ext*)
apply (*subst* (*asm*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*subst* (*asm*)(*3*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*clarsimp simp only*: *comp-def intro*!: *list-update-swap*)
done
define *aa* **where**
‹*aa* ≡ *fold-mset ?f* (*replicate* (*1*+*Max* (*nat-of-lit* ‘ *snd* ‘ *D'*)) []) (*mset-set* (*snd* ‘ *D'*))›

have *length-fold*: ‹*length* (*fold-mset* (λL *a*. *a*[*nat-of-lit* L := W L]) l M) = *length* l› **for** l M
  by (*induction* M) *auto*
have *length-aa*: ‹*length aa* = *Suc* (*Max* (*nat-of-lit* ‘ *snd* ‘ D'))›
  unfolding *aa-def* D''*-def*[*symmetric*] **by** (*simp add*: *length-fold*)
have H: ‹$x \in\# \mathcal{L}_{all}{}' \Longrightarrow$
    length $l \geq$ *Suc* (*Max* (*nat-of-lit* ‘ *set-mset* ($\mathcal{L}_{all}{}'$))) $\Longrightarrow$
    fold-mset (λL *a*. *a*[*nat-of-lit* L := W L]) l (*remdups-mset* ($\mathcal{L}_{all}{}'$)) ! *nat-of-lit* x = W x›
  for x l $\mathcal{L}_{all}{}'$
  unfolding $\mathcal{L}_{all}{}'$*-def*[*symmetric*]
  apply (*induction* $\mathcal{L}_{all}{}'$ *arbitrary*: l)
  subgoal by *simp*
  subgoal for *xa Ls l*
    apply (*case-tac* ‹(*nat-of-lit* ‘ *set-mset Ls*) = {}›)
     apply (*solves simp*)
    apply (*auto simp*: *less-Suc-eq-le length-fold*)
    done
  done
have H': ‹*aa* ! *nat-of-lit* x = W x› **if** ‹$x \in\# \mathcal{L}_{all}$ $\mathcal{A}_{in}$› **for** x
  using *that* **unfolding** *aa-def* D'*-def*
  by (*auto simp*: D'*-def image-image remdups-mset-def*[*symmetric*]
    less-Suc-eq-le intro!: H)
have ‹*?P aa*›
  by (*auto simp*: D'*-def image-image remdups-mset-def*[*symmetric*]
    less-Suc-eq-le length-aa H')
then show *?thesis*
  by *blast*
qed

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

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

definition *isasat-input-bounded-nempty* **where**
  ‹*isasat-input-bounded-nempty* = (*isasat-input-bounded* $\wedge$ *isasat-input-nempty*)›

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

lemma *in-*$\mathcal{L}_{all}$*-less-uint-max*$'$: ‹$L \in\# \mathcal{L}_{all}$ $\mathcal{A}_{in} \Longrightarrow$ *nat-of-lit* $L \leq$ *uint-max*›
  using *in-*$\mathcal{L}_{all}$*-less-uint-max* **by** *auto*

lemma *in-*$\mathcal{A}_{in}$*-less-than-uint-max-div-2*:
  ‹$L \in\# \mathcal{A}_{in} \Longrightarrow L \leq$ *uint-max div 2*›
  using *in-*$\mathcal{L}_{all}$*-less-uint-max*$'$[*of* ‹*Neg* L›]
  unfolding *Ball-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
  by (*auto simp*: *uint-max-def*)

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: ‹$\neg$*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 −
  let ?C = ‹atm-of ‘# C›
  have ‹distinct-mset ?C›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then obtain K where ‹¬count (atm-of ‘# C) K ≤ Suc 0›
      unfolding distinct-mset-count-less-1
      by auto
    then have ‹count (atm-of ‘# C) K ≥ 2›
      by auto
    then obtain L L′ C′ where
      C: ‹C = {#L, L′#} + C′› and L-L′: ‹atm-of L = atm-of L′›
      by (auto dest!: count-image-mset-multi-member-split-2)
    then show False
      using dist tauto by (auto simp: atm-of-eq-atm-of tautology-add-mset)
  qed
  then have card: ‹size ?C = card (set-mset ?C)›
    using distinct-mset-size-eq-card by blast
  have size: ‹size ?C = size C›
    using dist tauto
    by (induction C) (auto simp: tautology-add-mset)
  have m: ‹set-mset ?C ⊆ {0..<uint-max div 2}›
  proof
    fix L
    assume ‹L ∈ set-mset ?C›
    then have ‹L ∈ atms-of (ℒ_all 𝒜_in)›
    using lits by (auto simp: literals-are-in-ℒ_in-def atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹Pos L ∈# (ℒ_all 𝒜_in)›
      using lits by (auto simp: in-ℒ_all-atm-of-in-atms-of-iff)
    then have ‹nat-of-lit (Pos L) < uint-max − 1›
      using in-ℒ_all-less-uint-max by (auto simp: atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹L < uint-max div 2›
       by (auto simp: atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff uint-max-def)
    then show ‹L ∈ {0..<uint-max div 2}›
       by (auto simp: atm-of-lit-in-atms-of uint-max-def
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
  qed
  moreover have ‹card ... =  uint-max div 2›
    by auto
  ultimately have ‹card (set-mset ?C) ≤ uint-max div 2›
    using card-mono[OF - m] by auto
  then show ?thesis
    unfolding card[symmetric] size .
qed


lemma simple-clss-size-upper-div2:
  assumes
   lits: ‹literals-are-in-ℒ_in  𝒜_in C› and
   dist: ‹distinct-mset C› and
   tauto: ‹¬tautology C›
  shows ‹size C ≤ 1 + uint-max div 2›
```

proof −
  let *?C* = ‹*atm-of* ‘# *C*›
  have ‹*distinct-mset ?C*›
  proof (*rule ccontr*)
    assume ‹¬ *?thesis*›
    then obtain *K* **where** ‹¬*count* (*atm-of* ‘# *C*) *K* ≤ *Suc 0*›
      unfolding *distinct-mset-count-less-1*
      by *auto*
    then have ‹*count* (*atm-of* ‘# *C*) *K* ≥ *2*›
      by *auto*
    then obtain *L L′ C′* **where**
      C: ‹*C* = {#*L*, *L′*#} + *C′*› **and** *L-L′*: ‹*atm-of L* = *atm-of L′*›
      by (*auto dest*!: *count-image-mset-multi-member-split-2*)
    then show *False*
      using *dist tauto* **by** (*auto simp*: *atm-of-eq-atm-of tautology-add-mset*)
  qed
  then have *card*: ‹*size ?C* = *card* (*set-mset ?C*)›
    using *distinct-mset-size-eq-card* **by** *blast*
  have *size*: ‹*size ?C* = *size C*›
    using *dist tauto*
    by (*induction C*) (*auto simp*: *tautology-add-mset*)
  have *m*: ‹*set-mset ?C* ⊆ {*0*..*uint-max div 2*}›
  proof
    fix *L*
    assume ‹*L* ∈ *set-mset ?C*›
    then have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
    using *lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹*Pos L* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
      using *lits* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
    then have ‹*nat-of-lit* (*Pos L*) ≤ *uint-max*›
      using *in-*$\mathcal{L}_{all}$*-less-uint-max* **by** (*auto simp*: *atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹*L* ≤ *uint-max div 2*›
       by (*auto simp*: *atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff uint-max-def)
    then show ‹*L* ∈ {*0* .. *uint-max div 2*}›
       by (*auto simp*: *atm-of-lit-in-atms-of uint-max-def*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
  qed
  moreover have ‹*card* . . . = *1* + *uint-max div 2*›
    by *auto*
  ultimately have ‹*card* (*set-mset ?C*) ≤ *1* + *uint-max div 2*›
    using *card-mono*[*OF* - *m*] **by** *auto*
  then show *?thesis*
    unfolding *card*[*symmetric*] *size* **.**
qed

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 −
  let *?posC* = ‹*filter-mset is-pos C*›
  let *?negC* = ‹*filter-mset is-neg C*›

  have *C*: ‹C = *?posC* + *?negC*›
    apply (*subst multiset-partition*[*of - is-pos*])
    by *auto*
  have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *?posC*›
    by (*rule literals-are-in-*$\mathcal{L}_{in}$*-mono*[*OF lits*]) *auto*
  moreover have ‹*distinct-mset ?posC*›
    by (*rule distinct-mset-mono*[*OF -dist*]) *auto*
  ultimately have *pos*: ‹*size ?posC* ≤ *1* + *uint-max div 2*›
    by (*rule simple-clss-size-upper-div2*) (*auto simp*: *tautology-decomp*)

  have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *?negC*›
    by (*rule literals-are-in-*$\mathcal{L}_{in}$*-mono*[*OF lits*]) *auto*
  moreover have ‹*distinct-mset ?negC*›
    by (*rule distinct-mset-mono*[*OF -dist*]) *auto*
  ultimately have *neg*: ‹*size ?negC* ≤ *1* + *uint-max div 2*›
    by (*rule simple-clss-size-upper-div2*) (*auto simp*: *tautology-decomp*)

  show *?thesis*
    apply (*subst C*)
    apply (*subst size-union*)
    using *pos neg* **by** *linarith*
qed

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*›
  using *clss-size-uint-max*[*OF assms*] **by** (*auto simp*: *uint32-max-def uint64-max-def*)

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*: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint-max* − *1*›
 shows ‹*size C* ≤ *uint-max*›
proof −
  let *?A* = ‹*remdups-mset* (*atm-of* ‘# *C*)›
  have [*simp*]: ‹*distinct-mset* (*poss ?A*)› ‹*distinct-mset* (*negs ?A*)›
    by (*simp-all add*: *distinct-image-mset-inj inj-on-def*)

  have ‹*C* ⊆# *poss ?A* + *negs ?A*›
    apply (*rule distinct-subseteq-iff*[*THEN iffD1*])
    subgoal by (*auto simp*: *dist distinct-mset-add disjunct-not-in*)
    subgoal by (*auto simp*: *dist distinct-mset-add disjunct-not-in*)
    subgoal
      apply *rule*
      using *literal.exhaust-sel* **by** (*auto simp*: *image-iff*)
    done
  have [*simp*]: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*poss ?A*)› ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*negs ?A*)›
    using *lits*
    by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-negs-remdups-mset literals-are-in-*$\mathcal{L}_{in}$*-poss-remdups-mset*)

  have ‹¬ *tautology* (*poss ?A*)› ‹¬ *tautology* (*negs ?A*)›
    by (*auto simp*: *tautology-decomp*)
  then have ‹*size* (*poss ?A*) ≤ *uint-max div 2*› **and** ‹*size* (*negs ?A*) ≤ *uint-max div 2*›

using *simple-clss-size-upper-div2'*[*of* ‹*poss ?A*›]
simple-clss-size-upper-div2'[*of* ‹*negs ?A*›] *in-*$\mathcal{L}_{all}$*-less-uint-max*
by *auto*
then have ‹*size C* ≤ *uint-max div 2* + *uint-max div 2*›
using ‹*C* ⊆# *poss* (*remdups-mset* (*atm-of* '# *C*)) + *negs* (*remdups-mset* (*atm-of* '# *C*))›
size-mset-mono **by** *fastforce*
then show *?thesis* **by** (*auto simp*: *uint-max-def*)
qed

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 −
have ‹*length M* = *card* (*atm-of* ' *lits-of-l M*)›
using *no-dup-length-eq-card-atm-of-lits-of-l*[*OF n-d*] **.**
moreover have ‹*atm-of* ' *lits-of-l M* ⊆ *set-mset* $\mathcal{A}_{in}$›
using *lits* **unfolding** *literals-are-in-*$\mathcal{L}_{in}$*-trail-atm-of* **by** *auto*
ultimately have ‹*length M* ≤ *card* (*set-mset* $\mathcal{A}_{in}$)›
by (*simp add*: *card-mono*)
moreover {
have ‹*set-mset* $\mathcal{A}_{in}$ ⊆ {*0* ..< (*uint-max div 2*) + *1*}›
using *in-*$\mathcal{A}_{in}$*-less-than-uint-max-div-2* **by** (*fastforce simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
Ball-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *uint-max-def*)
from *subset-eq-atLeast0-lessThan-card*[*OF this*] **have** ‹*card* (*set-mset* $\mathcal{A}_{in}$) ≤ *uint-max div 2* + *1*›
.
}
ultimately show ‹*length M* ≤ *Suc* (*uint-max div 2*)›
by *linarith*
moreover have ‹*count-decided M* ≤ *length M*›
unfolding *count-decided-def* **by** *auto*
ultimately show ‹*count-decided M* ≤ *Suc* (*uint-max div 2*)› **by** *simp*
then show ‹*get-level M L* ≤ *Suc* (*uint-max div 2*)›
using *count-decided-ge-get-level*[*of M L*]
by *simp*
qed

lemma *length-trail-uint-max-div2*:
fixes *M* :: ‹(*nat*, *'b*) *ann-lits*›
assumes
M-$\mathcal{L}_{all}$: ‹∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$› **and**
n-d: ‹*no-dup M*›
shows ‹*length M* ≤ *uint-max div 2* + *1*›
proof −
have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
have *incl*: ‹*atm-of* '# *lit-of* '# *mset M* ⊆# *remdups-mset* (*atm-of* '# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *assms dist-atm-M*

by (*auto 5 5 simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct atm-of-eq-atm-of*)
have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)))›
by (*auto simp*: *inj-on-def*)
have *H*: ‹*xa* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}_{in}$ $\Longrightarrow$ *atm-of xa* $\leq$ *uint-max div 2*› **for** *xa*
using *in-*$\mathcal{L}_{all}$*-less-uint-max*
by (*cases xa*) (*auto simp*: *uint-max-def*)
have ‹*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$) $\subseteq\#$ *mset* [*0*..< *1* + (*uint-max div 2*)]›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *H distinct-image-mset-inj*[*OF inj-on*]
by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set dest*: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))›
by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))) $\leq$ *1* + *uint-max div 2*›
by *auto*
from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* $\leq$ *size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)))›
unfolding *uint-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
with *2* **show** *?thesis*
by (*auto simp*: *uint32-max-def*)
qed

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 *standard*
obtain *f* :: ‹′*a* ⇒ *nat*› **where** *f*: ‹*inj f*›
by *blast*
then have *Hf*: ‹*f x* = *f s* ⟷ *x* = *s*› **for** *s x*
unfolding *inj-on-def Ball-def comp-def* **by** *blast*
let *?f* = ‹λ*L*. (*is-pos L*, *f* (*atm-of L*))›
have ‹*OFCLASS*(*bool* × *nat*, *heap-class*)›
by *standard*
then obtain *g* :: ‹*bool* × *nat* ⇒ *nat*› **where** *g*: ‹*inj g*›
by *blast*
then have *H*: ‹*g* (*x*, *y*) = *g* (*s*, *t*) ⟷ *x* = *s* ∧ *y* = *t*› **for** *s t x y*
unfolding *inj-on-def Ball-def comp-def* **by** *blast*
have ‹*inj* (*g o ?f*)›
using *f g* **unfolding** *inj-on-def Ball-def comp-def H Hf*
apply (*intro allI impI*)
apply (*rename-tac x y*, *case-tac x*; *case-tac y*)
by *auto*
then show ‹∃ *to-nat*:: ′*a literal* ⇒ *nat*. *inj to-nat*›
by *blast*
qed

instance *annotated-lit* :: (*heap*, *heap*, *heap*) *heap*
proof *standard*
let *?f* = ‹λ*L*:: (′*a*, ′*b*, ′*c*) *annotated-lit*.
(*if is-decided L then Some* (*lit-dec L*) *else None*,

if is-decided L then None else Some (*lit-prop L*), *if is-decided L then None else Some* (*mark-of L*))›
have *f*: ‹*inj ?f*›
unfolding *inj-on-def Ball-def*
apply (*intro allI impI*)
apply (*rename-tac x y*, *case-tac x*; *case-tac y*)
by *auto*
then have *Hf*: ‹*?f x* = *?f s* ⟷ *x* = *s*› **for** *s x*
unfolding *inj-on-def Ball-def comp-def* **by** *blast*
have ‹*OFCLASS*(*'a option* × *'b option* × *'c option*, *heap-class*)›
by *standard*
then obtain *g* :: ‹*'a option* × *'b option* × *'c option* ⇒ *nat*› **where** *g*: ‹*inj g*›
by *blast*
then have *H*: ‹*g* (*x*, *y*) = *g* (*s*, *t*) ⟷ *x* = *s* ∧ *y* = *t*› **for** *s t x y*
unfolding *inj-on-def Ball-def comp-def* **by** *blast*
have ‹*inj* (*g o ?f*)›
using *f g* **unfolding** *inj-on-def Ball-def comp-def H Hf*
apply (*intro allI impI*)
apply (*rename-tac x y*, *case-tac x*; *case-tac y*)
by *auto*
then show ‹∃ *to-nat*:: (*'a*, *'b*, *'c*) *annotated-lit* ⇒ *nat*. *inj to-nat*›
by *blast*
qed

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

definition *default-literal* **where**
‹*default-literal* = *Pos default*›
instance by *standard*

end

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

definition *default-fmap* **where**
‹*default-fmap* = *fmempty*›
instance by *standard*

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*⟩
unfolding *bitXOR-1-if-mod-2 uminus-lit-imp-def*
by (*intro frefI nres-relI*) (*auto simp*: *nat-ann-lit-rel-def uminus-lit-imp-def case-prod-beta p2rel-def br-def nat-lit-rel-def split*: *option.splits*, *presburger*)

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*⟩
by (*cases S*) (*auto simp*: *watched-app-def*)

More Operations

lemma *nat-of-uint32-shiftr*: ⟨*nat-of-uint32* (*shiftr xi n*) = *shiftr* (*nat-of-uint32 xi*) *n*⟩
by *transfer* (*auto simp*: *shiftr-div-2n unat-def shiftr-nat-def*)

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*⟩
apply (*subgoal-tac* ⟨*length b* = *length ba*⟩)
subgoal
  apply (*rotate-tac 2*)
  apply (*induction b ba rule*: *list-induct2*)
   apply (*solves simp*)
  apply (*auto simp*: *p2rel-def nat-lit-rel-def br-def*, *presburger*)[]
  done
subgoal using *list-all2-lengthD* **by** *auto*
done

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*›
by (*auto simp*: *nat-ann-lit-rel-def is-decided-wl-def is-decided-def intro*!: *frefI nres-relI*
elim: *ann-lit-of-pair.elims*)

lemma *ann-lit-of-pair-if*:
‹*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))›
by (*cases D*) *auto*

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*: ‹$a \in$ *set xs* ⟹ *list-all2 R xs ys* ⟹ ∃ *b* ∈ *set ys*. *R a b*›
apply (*subgoal-tac* ‹*length xs* = *length ys*›)
apply (*rotate-tac 2*)
apply (*induction xs ys rule*: *list-induct2*)
apply ((*solves auto*)+)[*2*]
using *list-all2-lengthD* **by** *blast*

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*)›
using *get-all-ann-decomposition-ex* **by** *fastforce*

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

lemma *count-decided-butlast*:
‹*count-decided* (*butlast xs*) = (*if is-decided* (*last xs*) *then count-decided xs* − *1 else count-decided xs*)›
by (*cases xs rule*: *rev-cases*) (*auto simp*: *count-decided-def*)

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*⟩
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-def split*: *option.splits*)

lemma *watched-by-nth-watched-app′*:
⟨*watched-by S K* = ((*snd o snd o snd o snd o snd o snd*) *S*) *K*⟩
by (*cases S*) (*auto simp*: *watched-app-def*)

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

definition (**in** −) *find-decomp-wl-imp′* :: ⟨(*nat*, *nat*) *ann-lits* ⇒ *nat clause-l list* ⇒ *nat* ⇒
nat clause ⇒ *nat clauses* ⇒ *nat clauses* ⇒ *nat lit-queue-wl* ⇒
(*nat literal* ⇒ *nat watched*) ⇒ - ⇒ (*nat*, *nat*) *ann-lits nres*⟩ **where**
⟨*find-decomp-wl-imp′* = (λ*M N U D NE UE W Q L*. *find-decomp-wl-imp M D L*)⟩

lemma *nth-ll-watched-app*:
⟨(*uncurry2* (*RETURN ooo nth-rll*), *uncurry2* (*RETURN ooo watched-app*)) ∈
[λ((*W*, *L*), *i*). *L* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)]$_f$ (((⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)) $\times_r$ *nat-lit-rel*) $\times_r$ *nat-rel* →
⟨*nat-rel* $\times_r$ *Id*⟩ *nres-rel*⟩
unfolding *watched-app-def nth-rll-def*
by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def*
nat-lit-rel-def)

lemma *ex-literal-of-nat*: ⟨∃ *bb*. *b* = *literal-of-nat bb*⟩
by (*cases b*)
(*auto simp*: *nat-of-lit-def split*: *if-splits*; *presburger*; *fail*)+

definition (**in** −) *is-pos-code* :: ⟨*uint32* ⇒ *bool*⟩ **where**
⟨*is-pos-code L* ⟷ *bitAND L 1* = *0*⟩

Unit Propagation: Step

definition *delete-index-and-swap-update* :: ⟨(′*a* ⇒ ′*b list*) ⇒ ′*a* ⇒ *nat* ⇒ ′*a* ⇒ ′*b list*⟩ **where**
⟨*delete-index-and-swap-update W K w* = *W*(*K* := *delete-index-and-swap* (*W K*) *w*)⟩

The precondition is not necessary.

lemma *delete-index-and-swap-ll-delete-index-and-swap-update*:
⟨(*uncurry2* (*RETURN ooo delete-index-and-swap-ll*), *uncurry2* (*RETURN ooo delete-index-and-swap-update*))
∈[λ((*W*, *L*), *i*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ (⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) $\times_r$ *nat-lit-rel*) $\times_r$ *nat-rel* →
⟨⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩*nres-rel*⟩
by (*auto simp*: *delete-index-and-swap-ll-def uncurry-def fref-def nres-rel-def*
delete-index-and-swap-update-def map-fun-rel-def p2rel-def nat-lit-rel-def br-def
nth-list-update′ nat-lit-rel-def
simp del: *literal-of-nat.simps*)

definition *append-update* :: ‹('a ⇒ 'b list) ⇒ 'a ⇒ 'b ⇒ 'a ⇒ 'b list› **where**
‹*append-update W L a = W(L:= W (L) @ [a])*›

lemma *append-ll-append-update*:
‹(*uncurry2 (RETURN ooo (λxs i j. append-ll xs (nat-of-uint32 i) j)), uncurry2 (RETURN ooo append-update)*)
∈ $[\lambda((W, L), i).\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$
$\langle Id\rangle$*map-fun-rel* $(D_0\ \mathcal{A}) \times_f$ *unat-lit-rel* $\times_f$ *Id* → $\langle\langle Id\rangle$*map-fun-rel* $(D_0\ \mathcal{A})\rangle$*nres-rel*›
by (*auto simp*: *append-ll-def uncurry-def fref-def nres-rel-def*
delete-index-and-swap-update-def map-fun-rel-def p2rel-def nat-lit-rel-def
nth-list-update' append-update-def nat-lit-rel-def unat-lit-rel-def br-def
uint32-nat-rel-def append-update-def
simp del: *literal-of-nat.simps*)

definition *is-decided-hd-trail-wl* **where**
‹*is-decided-hd-trail-wl S = is-decided (hd (get-trail-wl S))*›

definition *is-decided-hd-trail-wll* :: ‹*nat twl-st-wl ⇒ bool nres*› **where**
‹*is-decided-hd-trail-wll = (λ(M, N, D, NE, UE, Q, W).*
RETURN (is-decided (hd M))
)›

lemma *Propagated-eq-ann-lit-of-pair-iff*:
‹*Propagated x21 x22 = ann-lit-of-pair (a, b) ⟷ x21 = a ∧ b = Some x22*›
by (*cases b*) *auto*

definition *lit-and-ann-of-propagated-code* **where**
‹*lit-and-ann-of-propagated-code = (λL::ann-lit-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* $(\mathcal{L}_{all}\ \mathcal{A})$ ⟷ *atms-of-ms (set-mset A) = atms-of* $(\mathcal{L}_{all}\ \mathcal{A})$›
by (*force simp add*: *atms-of-s-def in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def*
atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *atms-of-def atm-of-eq-atm-of uminus-*$\mathcal{A}_{in}$*-iff*
eq-commute[*of* ‹*set-mset (all-lits-of-mm -)*› ‹*set-mset* $(\mathcal{L}_{all}$ -)›]
dest: *multi-member-split*)

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›
by (*auto simp*: *card-max-lvl-def*)

lemma *card-max-lvl-empty*[*simp*]: ‹*card-max-lvl M {#} = 0*›
by (*auto simp*: *card-max-lvl-def*)

lemma *card-max-lvl-all-poss*:
‹*card-max-lvl M C = card-max-lvl M (poss (atm-of '# C))*›
unfolding *card-max-lvl-def*
apply (*induction C*)
subgoal by *auto*
subgoal for *L C*
using *get-level-uminus*[*of M L*]

```
    by (cases L) (auto)
  done

lemma card-max-lvl-distinct-cong:
  assumes
    ‹⋀L. get-level M (Pos L) = count-decided M ⟹ (L ∈ atms-of C) ⟹ (L ∈ atms-of C′)› and
    ‹⋀L. get-level M (Pos L) = count-decided M ⟹ (L ∈ atms-of C′) ⟹ (L ∈ 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 −
  have [simp]: ‹NO-MATCH (Pos x) L ⟹ get-level M L = get-level M (Pos (atm-of L))› for x L
    by (simp add: get-level-def)
  have [simp]: ‹atm-of L ∉ atms-of C′ ⟷ L ∉# C′ ∧ −L ∉# C′› for L C′
    by (cases L) (auto simp: atm-iff-pos-or-neg-lit)
  then have [iff]: ‹atm-of L ∈ atms-of C′ ⟷ L ∈# C′ ∨ −L ∈# C′› for L C′
    by blast
  have H: ‹distinct-mset {#L ∈# poss (atm-of ‘# C). get-level M L = count-decided M#}›
    if ‹distinct-mset C› ‹¬tautology C› for C
    using that by (induction C) (auto simp: tautology-add-mset atm-of-eq-atm-of)
  show ?thesis
    apply (subst card-max-lvl-all-poss)
    apply (subst (2) card-max-lvl-all-poss)
    unfolding card-max-lvl-def
    apply (rule arg-cong[of - - size])
    apply (rule distinct-set-mset-eq)
    subgoal by (rule H) (use assms in fast)+
    subgoal by (rule H) (use assms in fast)+
    subgoal using assms by (auto simp: atms-of-def imageI image-iff) blast+
    done
qed


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 *uint-max* $+$ *1*). Therefore, we restrict the clauses to have at least length 2 and we keep *length* $C - 2$ instead of *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 *standard*
  let *?f* = ‹(λx. *case x of IRRED* ⇒ (*0*::*nat*) | *LEARNED* ⇒ *1* | *DELETED* ⇒ *2*)›
  have ‹*inj ?f*›
    by (*auto simp*: *inj-def split*: *clause-status.splits*)
  then show ‹∃ *f*. *inj* (*f*:: *clause-status* ⇒ *nat*)›
    by *blast*
qed

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

definition *default-clause-status* **where** ‹*default-clause-status* = *DELETED*›
instance by *standard*

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* $C \longleftrightarrow$ *length* $C \leq$ *MAX-LENGTH-SHORT-CLAUSE*›

abbreviation *is-long-clause* **where**
‹*is-long-clause* $C \equiv \neg$*is-short-clause* C›

definition *header-size* :: ‹*nat clause-l* $\Rightarrow$ *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*:
‹$i > 3 \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *LBD-SHIFT*›
‹$i > 3 \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *ACTIVITY-SHIFT*›
‹$i > 3 \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *STATUS-SHIFT*›
‹$i > 3 \Longrightarrow i$ − *LBD-SHIFT* $\neq i$ − *ACTIVITY-SHIFT*›
‹$i > 3 \Longrightarrow i$ − *LBD-SHIFT* $\neq i$ − *STATUS-SHIFT*›
‹$i > 3 \Longrightarrow i$ − *ACTIVITY-SHIFT* $\neq i$ − *STATUS-SHIFT*›

‹$i > 4 \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *POS-SHIFT*›
‹$i > 4 \Longrightarrow i$ − *LBD-SHIFT* $\neq i$ − *POS-SHIFT*›
‹$i > 4 \Longrightarrow i$ − *ACTIVITY-SHIFT* $\neq i$ − *POS-SHIFT*›
‹$i > 4 \Longrightarrow i$ − *STATUS-SHIFT* $\neq i$ − *POS-SHIFT*›

‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i$ − *SIZE-SHIFT* = j − *SIZE-SHIFT* $\longleftrightarrow i = j$›
‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i$ − *LBD-SHIFT* = j − *LBD-SHIFT* $\longleftrightarrow i = j$›
‹$i > 4 \Longrightarrow j > 4 \Longrightarrow i$ − *ACTIVITY-SHIFT* = j − *ACTIVITY-SHIFT* $\longleftrightarrow i = j$›
‹$i > 3 \Longrightarrow j > 3 \Longrightarrow i$ − *STATUS-SHIFT* = j − *STATUS-SHIFT* $\longleftrightarrow i = j$›
‹$i > 4 \Longrightarrow j > 4 \Longrightarrow i$ − *POS-SHIFT* = j − *POS-SHIFT* $\longleftrightarrow i = j$›

‹$i \geq$ *header-size* $C \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *LBD-SHIFT*›
‹$i \geq$ *header-size* $C \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *ACTIVITY-SHIFT*›
‹$i \geq$ *header-size* $C \Longrightarrow i$ − *SIZE-SHIFT* $\neq i$ − *STATUS-SHIFT*›
‹$i \geq$ *header-size* $C \Longrightarrow i$ − *LBD-SHIFT* $\neq i$ − *ACTIVITY-SHIFT*›
‹$i \geq$ *header-size* $C \Longrightarrow i$ − *LBD-SHIFT* $\neq i$ − *STATUS-SHIFT*›
‹$i \geq$ *header-size* $C \Longrightarrow i$ − *ACTIVITY-SHIFT* $\neq i$ − *STATUS-SHIFT*›

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

‹$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i$ − *SIZE-SHIFT* = j − *SIZE-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i$ − *LBD-SHIFT* = j − *LBD-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i$ − *ACTIVITY-SHIFT* = j − *ACTIVITY-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i$ − *STATUS-SHIFT* = j − *STATUS-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow$ *is-long-clause* $C \Longrightarrow$ *is-long-clause* $C' \Longrightarrow$
i − *POS-SHIFT* = j − *POS-SHIFT* $\longleftrightarrow i = j$›
unfolding *POS-SHIFT-def STATUS-SHIFT-def ACTIVITY-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def header-size-def*
by (*auto split*: *if-splits simp*: *is-short-clause-def*)

lemma *header-size-ge0*[*simp*]: ‹*0* < *header-size x1*›
  by (*auto simp*: *header-size-def*)

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*).
     (*length C* ≥ *2* ∧
       header-size C + *length C* = *length arena* ∧
     (*is-long-clause C* ⟶ (*is-Pos* (*arena*!(*header-size C* − *POS-SHIFT*)) ∧
       xarena-pos(*arena*!(*header-size C* − *POS-SHIFT*)) ≤ *length C* − *2*))) ∧
     is-Status(*arena*!(*header-size C* − *STATUS-SHIFT*)) ∧
        (*xarena-status*(*arena*!(*header-size C* − *STATUS-SHIFT*)) = *IRRED* ⟷ *red*) ∧
        (*xarena-status*(*arena*!(*header-size C* − *STATUS-SHIFT*)) = *LEARNED* ⟷ ¬*red*) ∧
     is-LBD(*arena*!(*header-size C* − *LBD-SHIFT*)) ∧
     is-Act(*arena*!(*header-size C* − *ACTIVITY-SHIFT*)) ∧
     is-Size(*arena*!(*header-size C* − *SIZE-SHIFT*)) ∧
     xarena-length(*arena*!(*header-size C* − *SIZE-SHIFT*)) + *2* = *length C* ∧
     drop (*header-size C*) *arena* = *map ALit C*
)›

As (*N* ∝ *i*, *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*)) ⟷ (
     (*length* (*N*∝*i*) ≥ *2* ∧
       header-size (*N*∝*i*) + *length* (*N*∝*i*) = *length arena* ∧
     (*is-long-clause* (*N*∝*i*) ⟶ (*is-Pos* (*arena*!(*header-size* (*N*∝*i*) − *POS-SHIFT*)) ∧
       xarena-pos(*arena*!(*header-size* (*N*∝*i*) − *POS-SHIFT*)) ≤ *length* (*N*∝*i*) − *2*)) ∧
     is-Status(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) ∧
        (*xarena-status*(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) = *IRRED* ⟷ *irred N i*) ∧
        (*xarena-status*(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) = *LEARNED* ⟷ ¬*irred N i*) ∧
     is-LBD(*arena*!(*header-size* (*N*∝*i*) − *LBD-SHIFT*)) ∧
     is-Act(*arena*!(*header-size* (*N*∝*i*) − *ACTIVITY-SHIFT*)) ∧
     is-Size(*arena*!(*header-size* (*N*∝*i*) − *SIZE-SHIFT*)) ∧
     xarena-length(*arena*!(*header-size* (*N*∝*i*) − *SIZE-SHIFT*)) + *2* = *length* (*N*∝*i*) ∧
     drop (*header-size* (*N*∝*i*)) *arena* = *map ALit* (*N*∝*i*)
  ))›
proof −
  have *C*: ‹*the* (*fmlookup N i*) = (*N* ∝ *i*, *irred N i*)›
    by *simp*
  show *?thesis*
    apply (*subst C*)
    unfolding *xarena-active-clause-def prod.case*
    by *meson*
qed

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* ⟷
$is\text{-}Status(arena!(4 - STATUS\text{-}SHIFT)) \wedge xarena\text{-}status(arena!(4 - STATUS\text{-}SHIFT)) = DELETED$
$\wedge$
$is\text{-}LBD(arena!(4 - LBD\text{-}SHIFT)) \wedge$
$is\text{-}Act(arena!(4 - ACTIVITY\text{-}SHIFT)) \wedge$
$is\text{-}Size(arena!(4 - SIZE\text{-}SHIFT))$
›

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*[$i - STATUS\text{-}SHIFT$:= *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* ($N \propto i$)) (i + *length*($N \propto i$)) *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* ($N \propto i$) ∧
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* {}›
unfolding *valid-arena-def*
by *auto*

definition *arena-status* **where**
‹*arena-status arena i* = *xarena-status* (*arena*!($i - STATUS\text{-}SHIFT$))›

definition *arena-used* **where**
‹*arena-used arena i* = *xarena-used* (*arena*!($i - STATUS\text{-}SHIFT$))›

definition *arena-length* **where**
‹*arena-length arena i* = *2* + *xarena-length* (*arena*!($i - SIZE\text{-}SHIFT$))›

definition *arena-lbd* **where**
‹*arena-lbd arena i* = *xarena-lbd* (*arena*!($i - LBD\text{-}SHIFT$))›

definition *arena-act* **where**
‹*arena-act arena i* = *xarena-act* (*arena*!($i - ACTIVITY\text{-}SHIFT$))›

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 (*rule ccontr*)
assume *False*: ‹¬ *?thesis*›
let *?Ci* = ‹*the* (*fmlookup N i*)›
let *?Cj* = ‹*the* (*fmlookup N j*)›
have
1: ‹*xarena-active-clause* (*clause-slice arena N i*) (*N* ∝ *i*, *irred N i*)› **and**
2: ‹*xarena-active-clause* (*clause-slice arena N j*) (*N* ∝ *j*, *irred N j*)› **and**
i-le: ‹*i* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N*∝*i*)›**and**
j-le: ‹*j* < *length arena*› **and**
j-ge: ‹*j* ≥ *header-size*(*N*∝*j*)›
using *assms*
by *auto*

have *Ci*: ‹*?Ci* = (*N* ∝ *i*, *irred N i*)› **and** *Cj*: ‹*?Cj* = (*N* ∝ *j*, *irred N j*)›
by *auto*

have
eq: ‹*Misc.slice i* (*i* + *length* (*N* ∝ *i*)) *arena* = *map ALit* (*N* ∝ *i*)› **and**
‹*length* (*N* ∝ *i*) − *Suc 0* < *length* (*N* ∝ *i*)› **and**
length-Ni: ‹*length* (*N*∝*i*) ≥ *2*›
using *1 i-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
apply *simp-all*
apply *force*
done

from *arg-cong*[*OF this*(*1*), *of* ‹λ*n*. *n* ! (*length* (*N*∝*i*) − *1*)›] *this*(*2*−)
have *lit*: ‹*is-Lit* (*arena* ! (*i* + *length*(*N*∝*i*) − *1*))›
using *i-le i-ge* **by** (*auto simp*: *map-nth slice-nth*)
have
Cj2: ‹*2* ≤ *length* (*N* ∝ *j*)›
using *2 j-le j-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
header-size-def
by *simp*
have *headerj*: ‹*header-size* (*N* ∝ *j*) ≥ *4*›
unfolding *header-size-def* **by** (*auto split*: *if-splits*)
then have [*simp*]: ‹*header-size* (*N* ∝ *j*) − *POS-SHIFT* < *length* (*N* ∝ *j*) + *header-size* (*N* ∝ *j*)›
using *Cj2*

by *linarith*
have [*simp*]:
⟨*is-long-clause* $(N \propto j) \longrightarrow j + (\textit{header-size}\ (N \propto j) - \textit{POS-SHIFT}) - \textit{header-size}\ (N \propto j) = j -$ *POS-SHIFT*⟩
⟨$j + (\textit{header-size}\ (N \propto j) - \textit{STATUS-SHIFT}) - \textit{header-size}\ (N \propto j) = j - \textit{STATUS-SHIFT}$⟩
⟨$j + (\textit{header-size}\ (N \propto j) - \textit{SIZE-SHIFT}) - \textit{header-size}\ (N \propto j) = j - \textit{SIZE-SHIFT}$⟩
⟨$j + (\textit{header-size}\ (N \propto j) - \textit{LBD-SHIFT}) - \textit{header-size}\ (N \propto j) = j - \textit{LBD-SHIFT}$⟩
⟨$j + (\textit{header-size}\ (N \propto j) - \textit{ACTIVITY-SHIFT}) - \textit{header-size}\ (N \propto j) = j - \textit{ACTIVITY-SHIFT}$⟩
using *Cj2 headerj* **unfolding** *POS-SHIFT-def STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def ACTIVITY-SHIFT-def*
by (*auto simp*: *header-size-def*)

have
pos: ⟨$\textit{is-long-clause}\ (N \propto j) \longrightarrow \textit{is-Pos}\ (\textit{arena}\ !\ (j - \textit{POS-SHIFT}))$⟩ **and**
st: ⟨$\textit{is-Status}\ (\textit{arena}\ !\ (j - \textit{STATUS-SHIFT}))$⟩ **and**
size: ⟨$\textit{is-Size}\ (\textit{arena}\ !\ (j - \textit{SIZE-SHIFT}))$⟩ **and**
lbd: ⟨$\textit{is-LBD}\ (\textit{arena}\ !\ (j - \textit{LBD-SHIFT}))$⟩ **and**
act: ⟨$\textit{is-Act}\ (\textit{arena}\ !\ (j - \textit{ACTIVITY-SHIFT}))$⟩
using *2 j-le j-ge Cj2 headerj*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
by (*simp-all add*: *slice-nth*)
have *False* **if** *ji*: ⟨$j - i \geq \textit{length}\ (N \propto i)$⟩
proof −
have *Suc3*: ⟨$3 = \textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ 0))$⟩
by *auto*
have *Suc4*: ⟨$4 = \textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ 0)))$⟩
by *auto*
have *Suc5*: ⟨$5 = \textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ (\textit{Suc}\ 0))))$⟩
by *auto*
have *j-i-1* [*iff*]:
⟨$j - 1 = i + \textit{length}\ (N \propto i) - 1 \longleftrightarrow j = i + \textit{length}\ (N \propto i)$⟩
⟨$j - 2 = i + \textit{length}\ (N \propto i) - 1 \longleftrightarrow j = i + \textit{length}\ (N \propto i) + 1$⟩
⟨$j - 3 = i + \textit{length}\ (N \propto i) - 1 \longleftrightarrow j = i + \textit{length}\ (N \propto i) + 2$⟩
⟨$j - 4 = i + \textit{length}\ (N \propto i) - 1 \longleftrightarrow j = i + \textit{length}\ (N \propto i) + 3$⟩
⟨$j - 5 = i + \textit{length}\ (N \propto i) - 1 \longleftrightarrow j = i + \textit{length}\ (N \propto i) + 4$⟩
using *False that j-ge i-ge length-Ni* **unfolding** *Suc4 Suc5 header-size-def numeral-2-eq-2*
by (*auto split*: *if-splits*)
have *H4*: ⟨$\textit{Suc}\ (j - i) \leq \textit{length}\ (N \propto i) + 4 \Longrightarrow j - i = \textit{length}\ (N \propto i) \lor$
$j - i = \textit{length}\ (N \propto i) + 1 \lor j - i = \textit{length}\ (N \propto i) + 2 \lor j - i = \textit{length}\ (N \propto i) + 3$⟩
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
have *H5*: ⟨$\textit{Suc}\ (j - i) \leq \textit{length}\ (N \propto i) + 5 \Longrightarrow j - i = \textit{length}\ (N \propto i) \lor$
$j - i = \textit{length}\ (N \propto i) + 1 \lor j - i = \textit{length}\ (N \propto i) + 2 \lor j - i = \textit{length}\ (N \propto i) + 3 \lor$
$(\textit{is-long-clause}\ (N \propto j) \land j = i+\textit{length}\ (N \propto i) + 4)$⟩
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
consider
⟨$\textit{is-long-clause}\ (N \propto j)$⟩ ⟨$j - \textit{POS-SHIFT} = i + \textit{length}(N \propto i) - 1$⟩ |
⟨$j - \textit{STATUS-SHIFT} = i + \textit{length}(N \propto i) - 1$⟩ |
⟨$j - \textit{LBD-SHIFT} = i + \textit{length}(N \propto i) - 1$⟩ |
⟨$j - \textit{ACTIVITY-SHIFT} = i + \textit{length}(N \propto i) - 1$⟩ |
⟨$j - \textit{SIZE-SHIFT} = i + \textit{length}(N \propto i) - 1$⟩
using *False ji j-ge i-ge length-Ni*
unfolding *header-size-def not-less-eq-eq STATUS-SHIFT-def SIZE-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def le-Suc-eq POS-SHIFT-def j-i-1*
apply (*cases* ⟨$\textit{is-short-clause}\ (N \propto j)$⟩)

subgoal
using *H4* **by** *auto*
subgoal
using *H5* **by** *auto*
done
then show *False*
using *lit pos st size lbd act*
by *cases auto*
qed
moreover have *False* **if** *ji*: ‹$j - i < \mathit{length}\ (N \propto i)$›
proof −
from *arg-cong*[*OF eq*, *of* ‹$\lambda xs.\ xs\ !\ (j-i-1)$›]
have ‹*is-Lit* (*arena* ! ($j-1$))›
using *that j-le i-le* ‹$j > i$›
by (*auto simp*: *slice-nth*)
then show *False*
using *size* **unfolding** *SIZE-SHIFT-def* **by** *auto*
qed
ultimately show *False*
by *linarith*
qed

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** ‹$j \in \mathit{vdom}$›
shows ‹$j - i \geq \mathit{length}\ (N \propto i) + 4$›
proof (*rule ccontr*)
assume *False*: ‹¬ *?thesis*›
let *?Ci* = ‹*the* (*fmlookup N i*)›
let *?Cj* = ‹*the* (*fmlookup N j*)›
have
1: ‹*xarena-active-clause* (*clause-slice arena N i*) ($N \propto i$, *irred N i*)› **and**
2: ‹*arena-dead-clause* (*dead-clause-slice arena N j*)› **and**
i-le: ‹$i <$ *length arena*› **and**
i-ge: ‹$i \geq$ *header-size*($N \propto i$)›**and**
j-le: ‹$j <$ *length arena*› **and**
j-ge: ‹$j \geq 4$›
using *assms* **unfolding** *valid-arena-def*
by *auto*

have *Ci*: ‹*?Ci* = ($N \propto i$, *irred N i*)› **and** *Cj*: ‹*?Cj* = ($N \propto j$, *irred N j*)›
by *auto*

have
eq: ‹*Misc.slice i* (i + *length* ($N \propto i$)) *arena* = *map ALit* ($N \propto i$)› **and**
‹*length* ($N \propto i$) − *Suc 0* < *length* ($N \propto i$)› **and**
length-Ni: ‹*length* ($N \propto i$) ≥ 2› **and**
pos: ‹*is-long-clause* ($N \propto i$) ⟶
is-Pos (*arena* ! (i − *POS-SHIFT*))› **and**
status: ‹*is-Status* (*arena* ! (i − *STATUS-SHIFT*))› **and**
lbd: ‹*is-LBD* (*arena* ! (i − *LBD-SHIFT*))› **and**
act: ‹*is-Act* (*arena* ! (i − *ACTIVITY-SHIFT*))› **and**
size: ‹*is-Size* (*arena* ! (i − *SIZE-SHIFT*))› **and**
st-init: ‹(*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *IRRED*) = (*irred N i*)› **and**
st-learned: ‹(*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *LEARNED*) = (¬ *irred N i*)›
using *1 i-ge i-le*

```
    unfolding xarena-active-clause-def extra-information-mark-to-delete-def prod.case
    unfolding STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
    apply (simp-all add: header-size-def slice-nth split: if-splits)
    apply force+
    done

  have
    st: ‹is-Status (arena ! (j − STATUS-SHIFT))› and
    del: ‹xarena-status (arena ! (j − STATUS-SHIFT)) = DELETED›
    using 2 j-le j-ge unfolding arena-dead-clause-def STATUS-SHIFT-def
    by (simp-all add: header-size-def slice-nth)
  consider
    ‹j − STATUS-SHIFT ≥ i› |
    ‹j − STATUS-SHIFT < i›
    using False ‹j ≥ i› unfolding STATUS-SHIFT-def
    by linarith
  then show False
  proof cases
    case 1
    then have ‹j − STATUS-SHIFT < i + length (N∝i)›
      using ‹j ≥ i› False j-ge
      unfolding not-less-eq-eq STATUS-SHIFT-def
      by simp
    with arg-cong[OF eq, of ‹λn. n ! (j − STATUS-SHIFT − i)›]
    have lit: ‹is-Lit (arena ! (j − STATUS-SHIFT))›
      using 1 ‹j ≥ i› i-le i-ge j-ge by (auto simp: map-nth slice-nth STATUS-SHIFT-def)
    with st
    show False by auto
  next
    case 2
    then consider
      ‹j − STATUS-SHIFT = i − STATUS-SHIFT› |
      ‹j − STATUS-SHIFT = i − LBD-SHIFT› |
      ‹j − STATUS-SHIFT = i − ACTIVITY-SHIFT› |
      ‹j − STATUS-SHIFT = i − SIZE-SHIFT› |
      ‹is-long-clause (N ∝ i)› and ‹j − STATUS-SHIFT = i − POS-SHIFT›
      using ‹j ≥ i›
    unfolding STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
      by force
    then show False
      apply cases
      subgoal using st status st-init st-learned del by auto
      subgoal using st lbd by auto
      subgoal using st act by auto
      subgoal using st size by auto
      subgoal using st pos by auto
      done
  qed
qed
```

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 −
let *?Ci* = ⟨*the* (*fmlookup N i*)⟩
let *?Cj* = ⟨*the* (*fmlookup N j*)⟩
have
1: ⟨*xarena-active-clause* (*clause-slice arena N i*) ($N \propto i$, *irred N i*)⟩ **and**
2: ⟨*arena-dead-clause* (*dead-clause-slice arena N j*)⟩ **and**
i-le: ⟨$i <$ *length arena*⟩ **and**
i-ge: ⟨$i \geq$ *header-size*($N \propto i$)⟩**and**
j-le: ⟨$j <$ *length arena*⟩ **and**
j-ge: ⟨$j \geq 4$⟩
using *assms* **unfolding** *valid-arena-def*
by *auto*

have *Ci*: ⟨*?Ci* = ($N \propto i$, *irred N i*)⟩ **and** *Cj*: ⟨*?Cj* = ($N \propto j$, *irred N j*)⟩
by *auto*

have
eq: ⟨*Misc.slice i* (i + *length* ($N \propto i$)) *arena* = *map ALit* ($N \propto i$)⟩ **and**
⟨*length* ($N \propto i$) − *Suc 0* < *length* ($N \propto i$)⟩ **and**
length-Ni: ⟨*length* ($N \propto i$) ≥ 2⟩ **and**
pos: ⟨*is-long-clause* ($N \propto i$) $\longrightarrow$
is-Pos (*arena* ! (i − *POS-SHIFT*))⟩ **and**
status: ⟨*is-Status* (*arena* ! (i − *STATUS-SHIFT*))⟩ **and**
lbd: ⟨*is-LBD* (*arena* ! (i − *LBD-SHIFT*))⟩ **and**
act: ⟨*is-Act* (*arena* ! (i − *ACTIVITY-SHIFT*))⟩ **and**
size: ⟨*is-Size* (*arena* ! (i − *SIZE-SHIFT*))⟩ **and**
st-init: ⟨(*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *IRRED*) $\longleftrightarrow$ (*irred N i*)⟩ **and**
st-learned: ⟨ (*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *LEARNED*) $\longleftrightarrow$ ¬*irred N i*⟩
using *1 i-ge i-le*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
unfolding *STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def*
apply (*simp-all add*: *header-size-def slice-nth split*: *if-splits*)
apply *force+*
done

have
st: ⟨*is-Status* (*arena* ! (j − *STATUS-SHIFT*))⟩ **and**
del: ⟨*xarena-status* (*arena* ! (j − *STATUS-SHIFT*)) = *DELETED*⟩ **and**
lbd': ⟨*is-LBD* (*arena* ! (j − *LBD-SHIFT*))⟩ **and**
act': ⟨*is-Act* (*arena* ! (j − *ACTIVITY-SHIFT*))⟩ **and**
size': ⟨*is-Size* (*arena* ! (j − *SIZE-SHIFT*))⟩
using *2 j-le j-ge* **unfolding** *arena-dead-clause-def SHIFTS-def*
by (*simp-all add*: *header-size-def slice-nth*)
have *4*: ⟨*4* = *Suc* (*Suc* (*Suc* (*Suc 0*)))⟩ **and** *5*: ⟨*5* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))⟩
by *auto*
have [*simp*]: ⟨$a < 4 \Longrightarrow j$ − *Suc a* = i − *Suc 0* $\longleftrightarrow i = j - a$⟩ **for** *a*
using ⟨$i > j$⟩ *j-ge i-ge*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
have [*simp*]: ⟨*Suc i* − j = *Suc a* $\longleftrightarrow i - j = a$⟩ **for** *a*
using ⟨$i > j$⟩ *j-ge i-ge*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)

show *1*: ‹$i - j \geq Suc\ (Suc\ (Suc\ (Suc\ 0)))$› (**is** *?A*)
proof (*rule ccontr*)
assume *False*: ‹¬*?A*›
consider
‹$i - STATUS\text{-}SHIFT = j - STATUS\text{-}SHIFT$› |
‹$i - STATUS\text{-}SHIFT = j - LBD\text{-}SHIFT$› |
‹$i - STATUS\text{-}SHIFT = j - ACTIVITY\text{-}SHIFT$› |
‹$i - STATUS\text{-}SHIFT = j - SIZE\text{-}SHIFT$›
using *False* ‹$i > j$› *j-ge i-ge* **unfolding** *SHIFTS-def header-size-def 4*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
then show *False*
apply *cases*
subgoal using *st status st-init st-learned del* **by** *auto*
subgoal using *status lbd′* **by** *auto*
subgoal using *status act′* **by** *auto*
subgoal using *status size′* **by** *auto*
done
qed

show ‹$i - j \geq Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))$› (**is** *?A*)
if *long*: ‹*is-long-clause* ($N \propto i$)›
proof (*rule ccontr*)
assume *False*: ‹¬*?A*›

have [*simp*]: ‹$a < 5 \Longrightarrow a' < 4 \Longrightarrow i - Suc\ a = j - Suc\ a' \longleftrightarrow i - a = j - a'$› **for** *a a′*
using ‹$i > j$› *j-ge i-ge long*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
have ‹$i - j = Suc\ (Suc\ (Suc\ (Suc\ 0)))$›
using *1* ‹$i > j$› *False j-ge i-ge long* **unfolding** *SHIFTS-def header-size-def 4*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
then have ‹$i - POS\text{-}SHIFT = j - SIZE\text{-}SHIFT$›
using *1* ‹$i > j$› *j-ge i-ge long* **unfolding** *SHIFTS-def header-size-def 4 5*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
then show *False*
using *pos long size′*
by *auto*
qed
qed

lemma *valid-arena-in-vdom-le-arena*:
assumes ‹*valid-arena arena N vdom*› **and** ‹$j \in vdom$›
shows ‹$j < length\ arena$› **and** ‹$j \geq 4$›
using *assms* **unfolding** *valid-arena-def*
by (*cases* ‹$j \in\# dom\text{-}m\ N$›; *auto simp*: *header-size-def*
dest!: *multi-member-split split*: *if-splits*; *fail*)+

lemma *valid-minimal-difference-between-valid-index*:
assumes ‹*valid-arena arena N vdom*› **and**
‹$i \in\# dom\text{-}m\ N$› **and** ‹$j \in\# dom\text{-}m\ N$› **and** ‹$j > i$›
shows ‹$j - i \geq length\ (N \propto i) + header\text{-}size\ (N \propto j)$›
by (*rule minimal-difference-between-valid-index*[*OF* - *assms*(*2*−*4*)])
(*use assms*(*1*) **in** ‹*auto simp*: *valid-arena-def*›)

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 −
have *ia-ge*: ‹$ia \geq$ *header-size*$(N \propto ia)$› ‹$ia <$ *length arena*› **and**
i-ge: ‹$i \geq$ *header-size*$(N \propto i)$› ‹$i <$ *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ‹$ia < i$›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
Misc.slice-def header-size-def split: *if-splits*)
qed

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 −
have *ia-ge*: ‹$ia \geq 4$› ‹$ia <$ *length arena*› **and**
i-ge: ‹$i \geq$ *header-size*$(N \propto i)$› ‹$i <$ *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹$ia < i$›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
arena-dead-clause-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-extra-information-mark-to-delete*[*simp*]:
‹*length* (*extra-information-mark-to-delete arena* i) = *length arena*›
unfolding *extra-information-mark-to-delete-def* **by** *auto*

lemma *valid-arena-mono*: ‹*valid-arena ab ar vdom1* $\Longrightarrow$ *vdom2* $\subseteq$ *vdom1* $\Longrightarrow$ *valid-arena ab ar vdom2*›
unfolding *valid-arena-def*
by *fast*

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 −
  let *?arena* = ‹*extra-information-mark-to-delete arena i*›
  have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
    ‹$\bigwedge$*ia*. *ia* $\notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ *ia* =*i* $\vee$ ($i \neq ia \wedge ia \notin\#$ *dom-m N*)›
    using *assms distinct-mset-dom*[*of N*]
    by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
  have
    dom: ‹$\forall$ *i*$\in\#$*dom-m N*.
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    dom': ‹$\bigwedge$*i*. *i*$\in\#$*dom-m N* $\Longrightarrow$
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    vdom: ‹$\bigwedge$*i*. *i*$\in$*vdom* $\longrightarrow i \notin\#$ *dom-m N* $\longrightarrow 4 \leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N i*)›
    using *assms* **unfolding** *valid-arena-def* **by** *auto*
  have ‹*ia*$\in\#$*dom-m* (*fmdrop i N*) $\Longrightarrow$
        ia $<$ *length ?arena* $\wedge$
        header-size (*fmdrop i N* $\propto$ *ia*) $\leq$ *ia* $\wedge$
        xarena-active-clause (*clause-slice ?arena* (*fmdrop i N*) *ia*) (*the* (*fmlookup* (*fmdrop i N*) *ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-extra-information-mark-to-delete*[*OF i - dom*, *of ia*]
    by *auto*
  moreover have ‹$ia \neq i \longrightarrow$ *ia*$\in$*insert i vdom* $\longrightarrow$
        ia $\notin\#$ *dom-m* (*fmdrop i N*) $\longrightarrow$
        $4 \leq$ *ia* $\wedge$ *arena-dead-clause*
         (*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *ia*)› **for** *ia*
    using *vdom*[*of ia*] *clause-slice-extra-information-mark-to-delete-dead*[*OF i - - arena*, *of ia*]
    by *auto*
  moreover have ‹$4 \leq i \wedge$ *arena-dead-clause*
         (*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *i*)›
    using *dom'*[*of i*, *OF i*]
    unfolding *arena-dead-clause-def xarena-active-clause-alt-def*
      extra-information-mark-to-delete-def **apply** −
    by (*simp-all add*: *SHIFTS-def header-size-def Misc.slice-def drop-update-swap min-def*
         split: *if-splits*)
       force+
  ultimately show *?thesis*
    using *assms* **unfolding** *valid-arena-def*
    by *auto*
qed

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*›
  using *valid-arena-extra-information-mark-to-delete*[*OF assms*]
  by (*auto intro*: *valid-arena-mono*)

Removable from addressable space **lemma** *valid-arena-remove-from-vdom*:
  assumes ‹*valid-arena arena N* (*insert i vdom*)›
  shows ‹*valid-arena arena N vdom*›

using *assms valid-arena-def*
by (*auto dest*!: *in-diffD*)

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* ∈# *dom-m N*› **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*))›
shows
‹*clause-slice* (*update-act i act arena*) *N ia* =
(*if ia* = *i then update-act* (*header-size* (*N*∝*i*)) *act* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
ACTIVITY-SHIFT-def update-act-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-update-act*[*simp*]:
‹*length* (*update-act i act arena*) = *length arena*›
by (*auto simp*: *update-act-def*)

lemma *clause-slice-update-act-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *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 −
have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
arena-dead-clause-def update-act-def ACTIVITY-SHIFT-def
Misc.slice-def header-size-def split: *if-splits*)
qed

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*))›
  using *assms*
  by (*cases* ‹*is-short-clause* ($N \propto i$)›)
    (*simp-all add*: *xarena-active-clause-alt-def update-act-def SHIFTS-def Misc.slice-def*
    header-size-def)

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 −
  let *?arena* = ‹*update-act i act arena*›
  have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
     ‹$\bigwedge ia.\ ia \notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow ia = i \vee (i \neq ia \wedge ia \notin\#$ *dom-m N*)›
    using *assms distinct-mset-dom*[*of N*]
    by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
  have
    dom: ‹$\forall i \in\#$*dom-m N*.
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    dom': ‹$\bigwedge i.\ i \in\#$*dom-m N* $\Longrightarrow$
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    vdom: ‹$\bigwedge i.\ i \in vdom \longrightarrow i \notin\#$ *dom-m N* $\longrightarrow 4 \leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N i*)›
    using *assms* **unfolding** *valid-arena-def* **by** *auto*
  have ‹*ia*$\in\#$*dom-m N* $\Longrightarrow ia \neq i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-act*[*OF i - dom*, *of ia act*]
    by *auto*
  moreover have ‹$ia = i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-act*[*OF i - dom*, *of ia act*] *i*
    by (*simp add*: *xarena-active-clause-update-act-same*)
  moreover have ‹$ia \in vdom \longrightarrow$
        $ia \notin\#$ *dom-m N* $\longrightarrow$
        $4 \leq ia \wedge$ *arena-dead-clause*
         (*dead-clause-slice* (*update-act i act arena*) (*fmdrop i N*) *ia*)› **for** *ia*
    using *vdom*[*of ia*] *clause-slice-update-act-dead*[*OF i - - arena*, *of ia*] *i*
    by *auto*
  ultimately show *?thesis*
    using *assms* **unfolding** *valid-arena-def*

by *auto*
qed

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* ∈# *dom-m N*⟩ **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*))⟩
shows
⟨*clause-slice* (*update-lbd i lbd arena*) *N ia* =
(*if ia* = *i then update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N ia*)
else clause-slice arena N ia)⟩
proof −
have *ia-ge*: ⟨*ia* ≥ *header-size*(*N* ∝ *ia*)⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-lbd-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-update-lbd*[*simp*]:
⟨*length* (*update-lbd i lbd arena*) = *length arena*⟩
by (*auto simp*: *update-lbd-def*)

lemma *clause-slice-update-lbd-dead*:
assumes
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ia: ⟨*ia* ∉# *dom-m N*⟩ ⟨*ia* ∈ *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 −
have *ia-ge*: ⟨*ia* ≥ *4*⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-lbd-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)

qed

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*))›
  using *assms*
  by (*cases* ‹*is-short-clause* ($N \propto i$)›)
    (*simp-all add*: *xarena-active-clause-alt-def update-lbd-def SHIFTS-def Misc.slice-def*
    header-size-def)

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 −
  let *?arena* = ‹*update-lbd i lbd arena*›
  have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
     ‹$\bigwedge$*ia*. $ia \notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ $ia = i \vee (i \neq ia \wedge ia \notin\#$ *dom-m N*)›
    using *assms distinct-mset-dom*[*of N*]
    by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
  have
    dom: ‹$\forall i \in\#$*dom-m N*.
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    dom': ‹$\bigwedge i$. $i \in\#$*dom-m N* $\Longrightarrow$
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    vdom: ‹$\bigwedge i$. $i \in$*vdom* $\longrightarrow i \notin\#$ *dom-m N* $\longrightarrow 4 \leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N*
i)›
    using *assms* **unfolding** *valid-arena-def* **by** *auto*
  have ‹$ia \in\#$*dom-m N* $\Longrightarrow ia \neq i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*]
    by *auto*
  moreover have ‹$ia = i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*] *i*
    by (*simp add*: *xarena-active-clause-update-lbd-same*)
  moreover have ‹$ia \in$*vdom* $\longrightarrow$
        $ia \notin\#$ *dom-m N* $\longrightarrow$
        $4 \leq ia \wedge$ *arena-dead-clause*
         (*dead-clause-slice* (*update-lbd i lbd arena*) (*fmdrop i N*) *ia*)› **for** *ia*
    using *vdom*[*of ia*] *clause-slice-update-lbd-dead*[*OF i - - arena*, *of ia*] *i*
    by *auto*
  ultimately show *?thesis*

using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

Update saved position **definition** *update-pos-direct* **where**
⟨*update-pos-direct C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos pos*]⟩

lemma *clause-slice-update-pos*:
assumes
i: ⟨*i* ∈# *dom-m N*⟩ **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**
long: ⟨*is-long-clause* (*N* ∝ *i*)⟩
shows
⟨*clause-slice* (*update-pos-direct i pos arena*) *N ia* =
(*if ia* = *i then update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N ia*)
else clause-slice arena N ia)⟩
proof −
have *ia-ge*: ⟨*ia* ≥ *header-size*(*N* ∝ *ia*)⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*
show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge long*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *clause-slice-update-pos-dead*:
assumes
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ia: ⟨*ia* ∉# *dom-m N*⟩ ⟨*ia* ∈ *vdom*⟩ **and**
dom: ⟨*valid-arena arena N vdom*⟩ **and**
long: ⟨*is-long-clause* (*N* ∝ *i*)⟩
shows
⟨*arena-dead-clause* (*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)⟩
proof −
have *ia-ge*: ⟨*ia* ≥ *4*⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i long* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge long*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *xarena-active-clause-update-pos-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*))› **and**
    long: ‹*is-long-clause* ($N \propto i$)› **and**
    ‹*pos* $\leq$ *length* ($N \propto i$) $-$ *2*›
  shows ‹*xarena-active-clause* (*update-pos-direct* (*header-size* ($N \propto i$)) *pos* (*clause-slice arena N i*))
      (*the* (*fmlookup N i*))›
  using *assms*
  by (*simp-all add*: *update-pos-direct-def SHIFTS-def Misc.slice-def*
    header-size-def xarena-active-clause-alt-def)

lemma *length-update-pos*[*simp*]:
  ‹*length* (*update-pos-direct i pos arena*) = *length arena*›
  by (*auto simp*: *update-pos-direct-def*)

lemma *valid-arena-update-pos*:
  assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*› **and**
    long: ‹*is-long-clause* ($N \propto i$)›**and**
    pos: ‹*pos* $\leq$ *length* ($N \propto i$) $-$ *2*›
  shows ‹*valid-arena* (*update-pos-direct i pos arena*) *N vdom*›
proof $-$
  let *?arena* = ‹*update-pos-direct i pos arena*›
  have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
    ‹$\bigwedge$*ia*. *ia* $\notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ *ia* =*i* $\vee$ ($i \neq$ *ia* $\wedge$ *ia* $\notin\#$ *dom-m N*)›
    using *assms distinct-mset-dom*[*of N*]
    by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
  have
    dom: ‹$\forall$ $i \in\#$*dom-m N*.
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    dom': ‹$\bigwedge$*i*. $i \in\#$*dom-m N* $\Longrightarrow$
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    vdom: ‹$\bigwedge$*i*. $i \in$*vdom* $\longrightarrow i \notin\#$ *dom-m N* $\longrightarrow$ *4* $\leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N*
i)›
    using *assms* **unfolding** *valid-arena-def* **by** *auto*
  have ‹*ia*$\in\#$*dom-m N* $\Longrightarrow$ *ia* $\neq i \Longrightarrow$
        ia $<$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq$ *ia* $\wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom*, *of ia pos*] *long*
    by *auto*
  moreover have ‹*ia* = $i \Longrightarrow$
        ia $<$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq$ *ia* $\wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom*, *of ia pos*] *i long pos*
    by (*simp add*: *xarena-active-clause-update-pos-same*)
  moreover have ‹*ia*$\in$*vdom* $\longrightarrow$
        ia $\notin\#$ *dom-m N* $\longrightarrow$
        4 $\leq$ *ia* $\wedge$ *arena-dead-clause*

    (*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*)› **for** *ia*
  using *vdom*[*of ia*] *clause-slice-update-pos-dead*[*OF i - - arena*, *of ia*] *i long*
  by *auto*
 ultimately show *?thesis*
  using *assms* **unfolding** *valid-arena-def*
  by *auto*
qed

Swap literals **definition** *swap-lits* **where**
 ‹*swap-lits C i j arena* = *swap arena* (*C* +*i*) (*C* + *j*)›

lemma *clause-slice-swap-lits*:
 assumes
  i: ‹*i* ∈# *dom-m N*› **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 −
 have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
 i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
  using *dom ia i* **unfolding** *xarena-active-clause-def*
  by *auto*

 show *?thesis*
  using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
  minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge k l*
  by (*cases* ‹*ia* < *i*›)
   (*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
    swap-lits-def SHIFTS-def swap-def ac-simps
    Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-swap-lits*[*simp*]:
 ‹*length* (*swap-lits i k l arena*) = *length arena*›
 by (*auto simp*: *swap-lits-def*)

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 −
 have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*› **and**
 i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
  using *dom ia i* **unfolding** *valid-arena-def*

by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge k l*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
Misc.slice-def header-size-def split: *if-splits*)
qed

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*))›
using *assms*
unfolding *xarena-active-clause-alt-def*
by (*cases* ‹*is-short-clause* (*N* ∝ *i*)›)
(*simp-all add*: *swap-lits-def SHIFTS-def min-def swap-nth-if map-swap swap-swap*
header-size-def ac-simps is-short-clause-def split: *if-splits*)

lemma *is-short-clause-swap*[*simp*]: ‹*is-short-clause* (*swap* (*N* ∝ *i*) *k l*) = *is-short-clause* (*N* ∝ *i*)›
by (*auto simp*: *header-size-def is-short-clause-def split*: *if-splits*)

lemma *header-size-swap*[*simp*]: ‹*header-size* (*swap* (*N* ∝ *i*) *k l*) = *header-size* (*N* ∝ *i*)›
by (*auto simp*: *header-size-def split*: *if-splits*)

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 −
let *?arena* = ‹*swap-lits i k l arena*›
have [*simp*]: ‹*i* ∉# *remove1-mset i* (*dom-m N*)›
‹⋀*ia*. *ia* ∉# *remove1-mset i* (*dom-m N*) ⟷ *ia* =*i* ∨ (*i* ≠ *ia* ∧ *ia* ∉# *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have
dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *4* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N*
i)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹

ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *k l*
by *auto*
moreover have ‹*ia* = *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*)
(*the* (*fmlookup* (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *ia*))›
for *ia*
using *dom'*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *i k l*
xarena-active-clause-swap-lits-same[*OF - - - k l*, *of arena*]
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
4 ≤ *ia* ∧ *arena-dead-clause* (*dead-clause-slice* (*swap-lits i k l arena*) (*fmdrop i N*) *ia*)›
for *ia*
using *vdom*[*of ia*] *clause-slice-swap-lits-dead*[*OF i - - arena*, *of ia*] *i k l*
by *auto*
ultimately show *?thesis*
using *i k l arena* **unfolding** *valid-arena-def*
by *auto*
qed

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* ≥ *header-size* (*N* ∝ *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 −
have ‹*drop* (*header-size* (*N* ∝ *i*)) (*clause-slice arena N i*) = *map ALit* (*N* ∝ *i*)› **and**
‹*header-size* (*N* ∝ *i*) ≤ *i*› **and**
‹*i* < *length arena*›
using *assms*
unfolding *xarena-active-clause-alt-def*
by *auto*
from *arg-cong*[*OF this*(*1*), *of length*] *this*(*2*−)
have ‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*›
unfolding *xarena-active-clause-alt-def*
by (*auto simp add*: *slice-len-min-If header-size-def is-short-clause-def split*: *if-splits*)
then have ‹*clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N i* =

clause-slice arena N i›
by (*auto simp add*: *append-clause-skeleton-def*)
then show *?thesis*
using *assms* **by** *simp*
qed

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*›
by (*auto simp*: *append-clause-skeleton-def header-size-def*
append-clause-def)

lemma *arena-active-clause-append-clause-same*: ‹$2 \le$ *length C* $\Longrightarrow$ *st* $\neq$ *DELETED* $\Longrightarrow$
pos $\le$ *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*)))›
unfolding *xarena-active-clause-alt-def append-clause-skeleton-def*
by (*cases st*)
(*auto simp*: *header-size-def slice-start0 SHIFTS-def slice-Cons split*: *if-splits*)

lemma *clause-slice-append-clause*:
assumes
ia: ‹*ia* $\notin\#$ *dom-m N*› ‹*ia* $\in$ *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 −
have *ia-ge*: ‹*ia* ≥ 4› ‹*ia* < *length arena*›
using *dom ia* **unfolding** *valid-arena-def*
by *auto*
then have ‹*dead-clause-slice* (*arena*) *N ia* =
dead-clause-slice (*append-clause-skeleton pos st used act lbd C arena*) *N ia*›
by (*auto simp add*: *extra-information-mark-to-delete-def drop-update-swap*
append-clause-skeleton-def
arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
Misc.slice-def header-size-def split: *if-splits*)
then show *?thesis*
using *assms* **by** *simp*
qed

lemma *valid-arena-append-clause-skeleton*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ 2› **and**
b: ‹*b* $\longleftrightarrow$ (*st* = *IRRED*)› **and** *st*: ‹*st* $\neq$ *DELETED*› **and**
pos: ‹*pos* $\le$ *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 −
let *?arena* = ‹*append-clause-skeleton pos st used act lbd C arena*›

let *?i*= ‹*length arena* + *header-size C*›
let *?N* = ‹(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)›
let *?vdom* = ‹*insert* (*length arena* + *header-size C*) *vdom*›
have
dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *i* ≤ *length arena* ∧ *4* ≤ *i* ∧
arena-dead-clause (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have [*simp*]: ‹*?i* ∉# *dom-m N*›
using *dom'*[*of ?i*]
by *auto*
have ‹*ia*∈#*dom-m N* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *arena-active-clause-append-clause*[*of N ia arena*]
by *auto*
moreover have ‹*ia* = *?i* ⟹
ia < *length ?arena* ∧
header-size (*?N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena ?N ia*) (*the* (*fmlookup ?N ia*))› **for** *ia*
using *dom'*[*of ia*] *le-C arena-active-clause-append-clause-same*[*of C st pos b arena used*]
b st pos
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶ *ia* < *length* (*?arena*) ∧
4 ≤ *ia* ∧ *arena-dead-clause* (*Misc.slice* (*ia* − *4*) *ia* (*?arena*))› **for** *ia*
using *vdom*[*of ia*] *clause-slice-append-clause*[*of ia N vdom arena pos st used act lbd C*, *OF* - - *arena*]
le-C b st
by *auto*
ultimately show *?thesis*
unfolding *valid-arena-def*
by *auto*
qed

lemma *valid-arena-append-clause*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ *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*)›
using *valid-arena-append-clause-skeleton*[*OF assms*(*1*,*2*),
of b ‹*if b then IRRED else LEARNED*›]
by (*auto simp*: *append-clause-def*)

Refinement Relation

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

definition *bitfield-rel* **where**
⟨*bitfield-rel* $n = \{(a,\ b).\ b \longleftrightarrow a\ AND\ (2\ \hat{}\ n) > 0\}$⟩

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 −
have
dom: ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
using *valid* **unfolding** *valid-arena-def*
by *blast*+

have
i-le: ‹*i* < *length arena*› **and**
i-ge: ‹*header-size* (*N* ∝ *i*) ≤ *i*› **and**
xi: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
using *dom*[*OF i*] **by** *fast*+

have
ge2: ‹*2* ≤ *length* (*N* ∝ *i*)› **and**
‹*header-size* (*N* ∝ *i*) + *length* (*N* ∝ *i*) = *length* (*clause-slice arena N i*)› **and**
pos: ‹*is-long-clause* (*N* ∝ *i*) ⟶
is-Pos (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *POS-SHIFT*)) ∧
xarena-pos (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *POS-SHIFT*))
≤ *length* (*N* ∝ *i*) − *2*› **and**
status: ‹*is-Status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*))› **and**
init: ‹(*xarena-status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*)) =
IRRED) =
irred N i› **and**
learned: ‹(*xarena-status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*)) =
LEARNED) =
(¬ *irred N i*)› **and**
lbd: ‹*is-LBD* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *LBD-SHIFT*))› **and**
act: ‹*is-Act* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *ACTIVITY-SHIFT*))› **and**
size: ‹*is-Size* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *SIZE-SHIFT*))› **and**
size': ‹*Suc* (*Suc* (*xarena-length*
(*clause-slice arena N i* !
(*header-size* (*N* ∝ *i*) − *SIZE-SHIFT*)))) =
length (*N* ∝ *i*)› **and**
clause: ‹*Misc.slice i* (*i* + *length* (*N* ∝ *i*)) *arena* = *map ALit* (*N* ∝ *i*)›
using *xi i-le i-ge* **unfolding** *xarena-active-clause-alt-def arena-length-def*
by *simp-all*
have [*simp*]:
‹*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *LBD-SHIFT*) = *ALBD* (*arena-lbd arena i*)›
‹*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*) =
AStatus (*arena-status arena i*) (*arena-used arena i*)›
using *size size' i-le i-ge ge2 lbd status size'*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def arena-used-def*
by (*auto simp*: *SHIFTS-def slice-nth*)
have *HH*:
‹*arena-length arena i* = *length* (*N* ∝ *i*)› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›

using *size size′ i-le i-ge ge2 lbd status size′ ge2*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def*
by (*cases* ‹*arena* ! (*i* − *Suc 0*)›; *auto simp*: *SHIFTS-def slice-nth*; *fail*)+
then show ‹*length* ($N \propto i$) = *arena-length arena i*› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
using *i-le i-ge size′ size ge2 HH* **unfolding** *numeral-2-eq-2*
by (*simp-all split*:)
show ‹*arena-length arena i* ≥ *2*›
‹*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*›
using *ge2* **unfolding** *HH* **by** *auto*
show
‹*i* ≥ *header-size* ($N \propto i$)› **and**
‹*i* < *length arena*›
using *i-le i-ge* **by** *auto*
show *is-lit*: ‹*is-Lit* (*arena* ! (*i*+*j*))› ‹$N \propto i$! *j* = *arena-lit arena* (*i* + *j*)›
if ‹*j* < *length* ($N \propto i$)›
for *j*
using *arg-cong*[*OF clause*, *of* ‹λ*xs*. *xs* ! *j*›] *i-le i-ge that*
by (*auto simp*: *slice-nth arena-lit-def*)

show *i-le-arena*: ‹*i* + *length* ($N \propto i$) ≤ *length arena*›
using *arg-cong*[*OF clause*, *of length*] *i-le i-ge*
by (*auto simp*: *arena-lit-def slice-len-min-If*)
show ‹*is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› **and**
‹*arena-pos arena i* ≤ *arena-length arena i*›
if ‹*is-long-clause* ($N \propto i$)›
using *pos ge2 i-le i-ge that* **unfolding** *arena-pos-def HH*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› **and**
‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› **and**
‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))›
using *lbd act ge2 i-le i-ge status* **unfolding** *arena-pos-def*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*SIZE-SHIFT* ≤ *i*› **and** ‹*LBD-SHIFT* ≤ *i*› **and**
‹*ACTIVITY-SHIFT* ≤ *i*›
using *i-ge* **unfolding** *header-size-def SHIFTS-def* **by** (*auto split*: *if-splits*)
show ‹*j* < *length* ($N \propto i$) ⟹ *i* + *j* < *length arena*›
using *i-le-arena* **by** *linarith*
show
‹$N \propto i$! *0* = *arena-lit arena i*› **and**
‹*is-Lit* (*arena* ! *i*)›
using *is-lit*[*of 0*] *ge2* **by** *fastforce*+
show
‹*arena-status arena i* = *LEARNED* ⟷ ¬*irred N i* ›**and**
‹*arena-status arena i* = *IRRED* ⟷ *irred N i*› **and**
‹*arena-status arena i* ≠ *DELETED*›
using *learned init* **unfolding** *arena-status-def*
by (*auto simp*: *arena-status-def*)
show
‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* ($N \propto i$)›
apply (*subst list-eq-iff-nth-eq*, *intro conjI allI*)
subgoal
using *HH i-le-arena i-le*
by (*auto simp*: *slice-nth slice-len-min-If*)

subgoal for *j*
using *HH i-le-arena i-le is-lit*[*of j*]
by (*cases* ‹*arena* ! (*i* + *j*)›)
(*auto simp*: *slice-nth slice-len-min-If*
arena-lit-def)
done
qed

lemma *arena-dom-status-iff*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹$i \in vdom$›
shows
‹$i \in\#$ *dom-m N* $\longleftrightarrow$ *arena-status arena i* $\neq$ *DELETED*› (**is** ‹*?eq*› **is** ‹*?A* $\longleftrightarrow$ *?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**
‹$4 \leq i$› (**is** *?ge*)
proof −
have *H1*: *?eq ?lbd ?act ?stat ?ge*
if ‹*?A*›
proof −
have
‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
i-ge: ‹*header-size* ($N \propto i$) $\leq i$› **and**
i-le: ‹$i <$ *length arena*›
using *assms that* **unfolding** *valid-arena-def* **by** *blast+*
then have ‹*is-Status* (*clause-slice arena N i* ! (*header-size* ($N \propto i$) − *STATUS-SHIFT*))› **and**
‹(*xarena-status* (*clause-slice arena N i* ! (*header-size* ($N \propto i$) − *STATUS-SHIFT*)) = *IRRED*) =
irred N i› **and**
‹(*xarena-status* (*clause-slice arena N i* ! (*header-size* ($N \propto i$) − *STATUS-SHIFT*)) = *LEARNED*)
=
(¬ *irred N i*)› **and**
‹*is-LBD* (*clause-slice arena N i* ! (*header-size* ($N \propto i$) − *LBD-SHIFT*))› **and**
‹*is-Act* (*clause-slice arena N i* ! (*header-size* ($N \propto i$) − *ACTIVITY-SHIFT*)) ›
unfolding *xarena-active-clause-alt-def arena-status-def*
by *blast+*
then show *?eq* **and** *?lbd* **and** *?act* **and** *?stat* **and** *?ge*
using *i-ge i-le that*
unfolding *xarena-active-clause-alt-def arena-status-def*
by (*auto simp*: *SHIFTS-def header-size-def slice-nth split*: *if-splits*)
qed
moreover have *H2*: *?eq*
if ‹*?B*›
proof −
have *?A*
proof (*rule ccontr*)
assume ‹$i \notin\#$ *dom-m N*›
then have
‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i arena*)› **and**
i-ge: ‹$4 \leq i$› **and**
i-le: ‹$i <$ *length arena*›
using *assms* **unfolding** *valid-arena-def* **by** *blast+*
then show *False*
using ‹*?B*›
unfolding *arena-dead-clause-def*

by (*auto simp*: *arena-status-def slice-nth SHIFTS-def*)
qed
then show *?eq*
using *arena-lifting*[*OF valid*, *of i*] *that*
by *auto*
qed
moreover have *?lbd ?act ?stat ?ge* **if** ‹¬*?A*›
proof −
have
‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
i-ge: ‹*4* ≤ *i*› **and**
i-le: ‹*i* < *length arena*›
using *assms that* **unfolding** *valid-arena-def* **by** *blast*+
then show *?lbd ?act ?stat ?ge*
unfolding *arena-dead-clause-def*
by (*auto simp*: *SHIFTS-def slice-nth*)
qed
ultimately show *?eq* **and** *?lbd* **and** *?act* **and** *?stat* **and** *?ge*
by *blast*+
qed

lemma *valid-arena-one-notin-vdomD*:
‹*valid-arena M N vdom* ⟹ *Suc 0* ∉ *vdom*›
using *arena-dom-status-iff*[*of M N vdom 1*]
by *auto*

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*›
by *transfer* (*auto simp*: *unat-div*)

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*›
by (*auto simp*: *SHIFTS-def*)

Code Generation

Length **definition** *isa-arena-length* **where**
‹*isa-arena-length arena i* = *do* {

  ASSERT(*i* ≥ *SIZE-SHIFT* ∧ *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 −
have *ba-le*: ‹*ba* < *length aa*› **and**
size: ‹*is-Size* (*aa* ! (*ba* − *SIZE-SHIFT*))› **and**
length: ‹*length* (*N* ∝ *ba*) = *arena-length aa ba*›
using *ba valid* **by** (*auto simp*: *arena-lifting*)
have ‹(*a* ! (*ba* − *SIZE-SHIFT*), *aa* ! (*ba* − *SIZE-SHIFT*))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*, *of* ‹*ba* − *SIZE-SHIFT*› ‹*ba* − *SIZE-SHIFT*›])
(*use ba-le* **in** *auto*)
then have ‹*aa* ! (*ba* − *SIZE-SHIFT*) = *ASize* (*nat-of-uint32* (*a* ! (*ba* − *SIZE-SHIFT*)))›
using *size* **unfolding** *arena-el-rel-def*
by (*auto split*: *arena-el.splits simp*: *uint32-nat-rel-def br-def*)
moreover have ‹*Suc* (*Suc* (*nat-of-uint32* (*a* ! (*ba* − *SIZE-SHIFT*)))) ≤ *uint64-max*›
using *nat-of-uint32-le-uint32-max*[*of* ‹*a* ! (*ba* − *SIZE-SHIFT*)›]
by (*auto simp*: *uint64-max-def uint32-max-def*)
ultimately show *?thesis* **by** (*simp add*: *nat-of-uint64-add nat-of-uint64-uint64-of-uint32*)
qed

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*›
unfolding *isa-arena-length-def arena-length-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
intro!: *ASSERT-refine-left*)

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 −
  have *j-le*: ‹$j < length\ arena$› **and**
    length: ‹$length\ (N \propto j) = arena\text{-}length\ arena\ j$› **and**
    k1:‹$\bigwedge k.\ k < length\ (N \propto j) \Longrightarrow N \propto j\ !\ k = arena\text{-}lit\ arena\ (j + k)$› **and**
    k2:‹$\bigwedge k.\ k < length\ (N \propto j) \Longrightarrow is\text{-}Lit\ (arena\ !\ (j{+}k))$› **and**
    le: ‹$j + length\ (N \propto j) \le length\ arena$› **and**
    j-ge: ‹$header\text{-}size\ (N \propto j) \le j$›
    using *arena-lifting*[*OF valid j*] **by** (*auto simp*:)
  show *le'*: *?le*
     using *le ba-le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
     by (*auto split*: *if-splits*)

  have ‹(a ! ba, $arena$! ba)
      ∈ *uint32-nat-rel O arena-el-rel*›
    by (*rule param-nth*[*OF - - a*, *of* ‹*ba*› ‹*ba*›])
      (*use ba-le le'* **in** *auto*)
  then show *?unat*
     using *k1*[*of* ‹$ba - j$›] *k2*[*of* ‹$ba - j$›] *ba-le length ba-j*
     by (*cases* ‹*arena* ! *ba*›)
      (*auto simp*: *arena-el-rel-def unat-lit-rel-def arena-lit-def*
       split: *arena-el.splits*)
qed

definition *arena-is-valid-clause-idx-and-access* :: ‹$arena \Rightarrow nat \Rightarrow nat \Rightarrow bool$› **where**
‹*arena-is-valid-clause-idx-and-access arena i j* ⟷
  ($\exists N\ vdom.\ valid\text{-}arena\ arena\ N\ vdom \land i \in\#\ dom\text{-}m\ N \land j < length\ (N \propto i)$)›

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* ⟷
  ($\exists j.\ i \ge j \land arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\text{-}and\text{-}access\ arena\ j\ (i - j)$)›

lemma *isa-arena-lit-arena-lit*:
  ‹(*uncurry isa-arena-lit*, *uncurry* (*RETURN oo arena-lit*)) ∈
    [*uncurry arena-lit-pre*]$_f$
     ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_r$ *nat-rel* → ⟨*unat-lit-rel*⟩*nres-rel*›
  unfolding *isa-arena-lit-def arena-lit-def*
  by (*intro frefI nres-relI*)
    (*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
        list-rel-imp-same-length arena-length-uint64-conv arena-lifting
        arena-is-valid-clause-idx-and-access-def arena-length-literal-conv
        arena-lit-pre-def
      intro!: *ASSERT-refine-left*)

Status of the clause **definition** *isa-arena-status* **where**
  ‹*isa-arena-status arena i* = *do* {
      ASSERT($i < length\ arena$);
      ASSERT($i \ge STATUS\text{-}SHIFT$);
      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*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*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*)))
∈ *uint32-nat-rel O status-rel*› (**is** *?rel*)
proof −
show *le*: *?le* **and** *i4*: ‹$4 \leq j$› **and** ‹$j \geq$ *STATUS-SHIFT*›
using *valid j* **unfolding** *valid-arena-def*
by (*cases* ‹*j* ∈# *dom-m N*›; *auto simp*: *header-size-def SHIFTS-def split*: *if-splits*; *fail*)+
have [*intro*]: ‹⋀*a y*. (*a*, *y*) ∈ *uint32-nat-rel* ⟹
(*a AND 3*, *y AND 3*) ∈ *uint32-nat-rel*›
apply (*auto simp*: *uint32-nat-rel-def br-def nat-of-uint32-ao*)
by (*metis nat-of-uint32-3 nat-of-uint32-ao*(*1*))
have [*dest*]: ‹(*y*, *AStatus x61 x62*) ∈ *arena-el-rel* ⟹ (*y AND 3*, *x61*) ∈ *status-rel*› **for** *y x61 x62*
by (*auto simp*: *status-rel-def arena-el-rel-def*)
have ‹(*a* ! (*j* − *STATUS-SHIFT*), *arena* ! (*j* − *STATUS-SHIFT*)) ∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*]) (*use le* **in** ‹*auto simp*: *list-rel-imp-same-length*›)
then have ‹(*a* ! (*j* − *STATUS-SHIFT*) *AND 0b11*, *xarena-status* (*arena* ! (*j* − *STATUS-SHIFT*)))
∈ *uint32-nat-rel O status-rel*›
using *arena-dom-status-iff*[*OF valid j*]
by (*cases* ‹*arena* ! (*j* − *STATUS-SHIFT*)›)
(*auto intro*!: *relcomp.relcompI*)
then show *?rel*
using *arena-dom-status-iff*[*OF valid j*]
by (*cases* ‹*arena* ! (*j* − *STATUS-SHIFT*)›)
(*auto simp*: *arena-el-rel-def*)
qed

lemma *isa-arena-status-arena-status*:
‹(*uncurry isa-arena-status*, *uncurry* (*RETURN oo arena-status*)) ∈
[*uncurry arena-is-valid-clause-vdom*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_r$ *nat-rel* → ⟨*uint32-nat-rel O status-rel*⟩*nres-rel*›
unfolding *isa-arena-status-def arena-status-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
arena-is-valid-clause-idx-and-access-def arena-length-literal-conv
arena-is-valid-clause-vdom-def arena-status-literal-conv
intro!: *ASSERT-refine-left*)

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

definition *swap-lits-pre* **where**
‹*swap-lits-pre C i j arena* ⟷ *C* + *i* < *length arena* ∧ *C* + *j* < *length arena*›

lemma *isa-arena-swap*:
‹(*uncurry3 isa-arena-swap*, *uncurry3* (*RETURN oooo swap-lits*)) ∈

[*uncurry3 swap-lits-pre*]$_f$
nat-rel $\times_f$ *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*›
unfolding *isa-arena-status-def arena-status-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
arena-is-valid-clause-idx-and-access-def arena-length-literal-conv
arena-is-valid-clause-vdom-def arena-status-literal-conv
isa-arena-swap-def swap-lits-def swap-lits-pre-def
intro!: *ASSERT-refine-left swap-param*)

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

lemma *arena-lbd-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**
b: ‹(*b*, *bb*) ∈ *uint32-nat-rel*›
shows
‹*j* − *LBD-SHIFT* < *length arena*› (**is** *?le*) **and**
‹(*a*[*j* − *LBD-SHIFT* := *b*], *update-lbd j bb arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
(**is** *?unat*)
proof −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*›
using *arena-lifting*[*OF valid j*] **by** (*auto simp*:)
show *le'*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *LBD-SHIFT-def*)

show *?unat*
using *length a b*
by
(*auto simp*: *arena-el-rel-def unat-lit-rel-def arena-lit-def update-lbd-def*
uint32-nat-rel-def br-def Collect-eq-comp
split: *arena-el.splits*
intro!: *list-rel-update'*)
qed

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*)) ∈
[*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*⟩
unfolding *isa-arena-status-def arena-status-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
arena-is-valid-clause-idx-and-access-def arena-lbd-conv
arena-is-valid-clause-vdom-def arena-status-literal-conv
update-lbd-pre-def
isa-arena-swap-def swap-lits-def swap-lits-pre-def isa-update-lbd-def
intro!: *ASSERT-refine-left*)

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* ∈# *dom-m N*› **and**
a: ‹(*a*, *arena*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*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 −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*› **and**
lbd: ‹*is-LBD* (*arena* ! (*j* − *LBD-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** (*auto simp*:)
show *le'*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *LBD-SHIFT-def*)
show *?ge*
using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have
‹(*a* ! (*j* − *LBD-SHIFT*),
(*arena* ! (*j* − *LBD-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
then show ‹(*a* ! (*j* − *LBD-SHIFT*),
xarena-lbd (*arena* ! (*j* − *LBD-SHIFT*)))
∈ *uint32-nat-rel*›

using *lbd* **by** (*cases* ‹*arena* ! (*j* − *LBD-SHIFT*)›) (*auto simp*: *arena-el-rel-def*)
qed

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-get-clause-LBD-def get-clause-LBD-def arena-get-lbd-conv*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def
list-rel-imp-same-length
intro!: *ASSERT-leI*)

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* {
ASSERT(*C* − *POS-SHIFT* < *length arena* ∧ *C* ≥ *POS-SHIFT*);
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 −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*› **and**
lbd: ‹*is-Pos* (*arena* ! (*j* − *POS-SHIFT*))› **and**
ge: ‹*length* (*N* ∝ *j*) > *MAX-LENGTH-SHORT-CLAUSE*›
using *arena-lifting*[*OF valid j*] *length* **by** (*auto simp*: *is-short-clause-def*)
show *le'*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *POS-SHIFT-def*)
show *?ge*

using *j-ge ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have
‹(*a* ! (*j* − *POS-SHIFT*),
(*arena* ! (*j* − *POS-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF* - - *a*]) (*use j-le* **in** *auto*)
moreover have ‹*Suc* (*Suc* (*nat-of-uint32* (*a* ! (*j* − *POS-SHIFT*)))) ≤ *uint64-max*›
using *nat-of-uint32-le-uint32-max*[*of* ‹*a* ! (*j* − *POS-SHIFT*)›]
unfolding *uint64-max-def uint32-max-def*
by *auto*
ultimately show *?rel*
using *lbd* **apply** (*cases* ‹*arena* ! (*j* − *POS-SHIFT*)›)
by (*auto simp*: *arena-el-rel-def*
uint64-nat-rel-def br-def two-uint64-def uint32-nat-rel-def nat-of-uint64-add
uint64-of-uint32-def nat-of-uint64-add arena-pos-def
nat-of-uint64-uint64-of-nat-id nat-of-uint32-le-uint64-max)
then show *?eq′*
using *lbd* ‹*?rel*› **apply** (*cases* ‹*arena* ! (*j* − *POS-SHIFT*)›)
by (*auto simp*: *arena-el-rel-def*
uint64-nat-rel-def br-def two-uint64-def uint32-nat-rel-def nat-of-uint64-add
uint64-of-uint32-def nat-of-uint64-add arena-pos-def
nat-of-uint64-uint64-of-nat-id nat-of-uint32-le-uint64-max)
qed

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-get-saved-pos-def arena-get-lbd-conv*
arena-is-valid-clause-idx-def arena-get-pos-conv
list-rel-imp-same-length get-saved-pos-pre-def
intro!: *ASSERT-leI*)

definition *isa-get-saved-pos′* :: ‹*uint32 list* ⇒ *nat* ⇒ *nat nres*› **where**
‹*isa-get-saved-pos′ arena C* = *do* {
pos ← *isa-get-saved-pos arena C*;
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*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-get-saved-pos-def arena-pos-def*
arena-is-valid-clause-idx-def arena-get-pos-conv isa-get-saved-pos′-def
list-rel-imp-same-length get-saved-pos-pre-def
intro!: *ASSERT-leI*)

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

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*⟩
by (*auto simp*: *arena-update-pos-def update-pos-direct-def*)

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 −
have *j-le*: ⟨*j* < *length arena*⟩ **and**
length: ⟨*length* (*N* ∝ *j*) = *arena-length arena j*⟩ **and**
k1:⟨⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)⟩ **and**
k2:⟨⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))⟩ **and**
le: ⟨*j* + *length* (*N* ∝ *j*) ≤ *length arena*⟩ **and**
j-ge: ⟨*header-size* (*N* ∝ *j*) ≤ *j*⟩ **and**
long: ⟨*is-long-clause* (*N* ∝ *j*)⟩ **and**
pos: ⟨*is-Pos* (*arena* ! (*j* − *POS-SHIFT*))⟩ **and**
is-size: ⟨*is-Size* (*arena* ! (*j* − *SIZE-SHIFT*))⟩
using *arena-lifting*[*OF valid j*] *length* **by** (*auto simp*: *is-short-clause-def header-size-def*)
show *le′*: *?le* **and** *?ge*
using *le j-ge long* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *POS-SHIFT-def*)
have
⟨(*a* ! (*j* − *SIZE-SHIFT*),
(*arena* ! (*j* − *SIZE-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*⟩
by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
then show ⟨*pos* − *2* ≤ *uint-max*⟩
using *b′ length is-size pos-le nat-of-uint32-le-uint32-max*[*of* ⟨*a* ! (*j* − *SIZE-SHIFT*)⟩]
by (*cases* ⟨*arena* ! (*j* − *SIZE-SHIFT*)⟩)
(*auto simp*: *uint32-nat-rel-def br-def arena-el-rel-def arena-length-def*)
then show *?unat*
using *length a pos b′*
valid-arena-update-pos[*OF valid j* ⟨*is-long-clause* (*N* ∝ *j*)⟩]
by (*auto simp*: *arena-el-rel-def unat-lit-rel-def arena-lit-def arena-update-pos-def*
uint32-nat-rel-def br-def Collect-eq-comp nat-of-uint32-notle-minus
nat-of-uint32-uint32-of-nat-id
split: *arena-el.splits*
intro!: *list-rel-update′*)
qed

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*)) ∈
$[isa\text{-}update\text{-}pos\text{-}pre]_f$
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*›
unfolding *isa-arena-status-def arena-status-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
arena-is-valid-clause-idx-and-access-def arena-lbd-conv
arena-is-valid-clause-vdom-def arena-status-literal-conv
update-lbd-pre-def isa-update-pos-def update-pos-direct-def isa-update-pos-pre-def
isa-arena-swap-def swap-lits-def swap-lits-pre-def isa-update-lbd-def
arena-update-pos-conv
intro!: *ASSERT-refine-left*)

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 −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*›
using *arena-lifting*[*OF valid j*] **by** (*auto simp*:)
show *le'*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *SHIFTS-def*)

show *?rel*
apply (*rule list-rel-update'*[*OF arena*])

using *length*
by
(*auto simp*: *arena-el-rel-def unat-lit-rel-def arena-lit-def update-lbd-def*
uint32-nat-rel-def br-def Collect-eq-comp status-rel-def bitfield-rel-def
split: *arena-el.splits*
intro!:)
show *?ge*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *SHIFTS-def*)
qed

lemma *isa-mark-garbage*:
‹(*uncurry mark-garbage*, *uncurry* (*RETURN oo extra-information-mark-to-delete*)) $\in$
[*mark-garbage-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* $\rightarrow$
⟨⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*⟩*nres-rel*›
unfolding *isa-arena-status-def arena-status-def*
by (*intro frefI nres-relI*)
(*auto simp*: *arena-is-valid-clause-idx-def uint64-nat-rel-def br-def two-uint64-def*
list-rel-imp-same-length arena-length-uint64-conv arena-lifting
arena-is-valid-clause-idx-and-access-def arena-lbd-conv
arena-is-valid-clause-vdom-def arena-status-literal-conv mark-garbage-pre
mark-garbage-def mark-garbage-pre-def extra-information-mark-to-delete-def
isa-arena-swap-def swap-lits-def swap-lits-pre-def isa-update-lbd-def
intro!: *ASSERT-refine-left*)

Activity **definition** *arena-act-pre* **where**
‹*arena-act-pre* = *arena-is-valid-clause-idx*›

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

lemma *arena-act-conv*:
assumes
valid: ‹*valid-arena arena N x*› **and**
j: ‹*j* $\in\#$ *dom-m N*› **and**
a: ‹(*a*, *arena*) $\in$ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
shows
‹*j* − *ACTIVITY-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*ACTIVITY-SHIFT* $\leq$ *j*› (**is** *?ge*) **and**
‹(*a* ! (*j* − *ACTIVITY-SHIFT*),
xarena-act (*arena* ! (*j* − *ACTIVITY-SHIFT*)))
$\in$ *uint32-nat-rel*›
proof −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* $\propto$ *j*) = *arena-length arena j*› **and**
k1:‹$\bigwedge$*k*. *k* < *length* (*N* $\propto$ *j*) $\Longrightarrow$ *N* $\propto$ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹$\bigwedge$*k*. *k* < *length* (*N* $\propto$ *j*) $\Longrightarrow$ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* $\propto$ *j*) $\leq$ *length arena*› **and**
j-ge: ‹*header-size* (*N* $\propto$ *j*) $\leq$ *j*› **and**
lbd: ‹*is-Act* (*arena* ! (*j* − *ACTIVITY-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** *auto*
show *le*′: *?le*

using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *ACTIVITY-SHIFT-def*)
show *?ge*
using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have
‹(*a* ! (*j* − *ACTIVITY-SHIFT*),
(*arena* ! (*j* − *ACTIVITY-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF* - - *a*]) (*use j-le* **in** *auto*)
then show ‹(*a* ! (*j* − *ACTIVITY-SHIFT*),
xarena-act (*arena* ! (*j* − *ACTIVITY-SHIFT*)))
∈ *uint32-nat-rel*›
using *lbd* **by** (*cases* ‹*arena* ! (*j* − *ACTIVITY-SHIFT*)›) (*auto simp*: *arena-el-rel-def*)
qed

lemma *isa-arena-act-arena-act*:
‹(*uncurry isa-arena-act*, *uncurry* (*RETURN oo arena-act*)) ∈
[*uncurry arena-act-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨*uint32-nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-arena-act-def arena-act-def arena-get-lbd-conv*
arena-act-pre-def arena-is-valid-clause-idx-def
list-rel-imp-same-length arena-act-conv
intro!: *ASSERT-leI*)

Increment Activity **definition** *isa-arena-incr-act* :: ‹*uint32 list* ⇒ *nat* ⇒ *uint32 list nres*› **where**
‹*isa-arena-incr-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-incr-act* **where**
‹*arena-incr-act arena i* = *arena*[*i* − *ACTIVITY-SHIFT* := *AActivity* (*sum-mod-uint32-max 1* (*xarena-act* (*arena*!(*i* − *ACTIVITY-SHIFT*))))]›

lemma *arena-incr-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*) + *1*], *arena-incr-act arena j*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
proof −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*› **and**
lbd: ‹*is-Act* (*arena* ! (*j* − *ACTIVITY-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** *auto*

show *le'*: *?le*
  using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
  by (*auto split*: *if-splits simp*: *ACTIVITY-SHIFT-def*)
show *?ge*
  using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have *b*:
  ‹(*a* ! (*j* − *ACTIVITY-SHIFT*),
    (*arena* ! (*j* − *ACTIVITY-SHIFT*)))
   ∈ *uint32-nat-rel O arena-el-rel*›
  by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
show ‹(*a*[*j* − *ACTIVITY-SHIFT* := *a* ! (*j* − *ACTIVITY-SHIFT*) + *1*], *arena-incr-act arena j*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
  unfolding *arena-incr-act-def*
  by (*rule list-rel-update'*[*OF a*])
   (*cases* ‹*arena* ! (*j* − *ACTIVITY-SHIFT*)›;
   use lbd b **in** ‹*auto simp add*: *uint32-nat-rel-def br-def arena-el-rel-def*
    Collect-eq-comp sum-mod-uint32-max-def nat-of-uint32-plus›)
qed

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*›
by (*intro frefI nres-relI*)
 (*auto simp*: *isa-arena-incr-act-def arena-act-def arena-get-lbd-conv*
  arena-act-pre-def arena-is-valid-clause-idx-def arena-incr-act-conv
  list-rel-imp-same-length arena-act-conv
  intro!: *ASSERT-leI*)

lemma *length-clause-slice-list-update*[*simp*]:
‹*length* (*clause-slice* (*arena*[*i* := *x*]) *a b*) = *length* (*clause-slice arena a b*)›
by (*auto simp*: *Misc.slice-def*)

lemma *length-arena-incr-act*[*simp*]:
‹*length* (*arena-incr-act arena C*) = *length arena*›
by (*auto simp*: *arena-incr-act-def*)

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 −
let *?arena* = ‹*arena-incr-act arena C*›
have *act*: ‹∀ *i*∈#*dom-m N*.
  i < *length* (*arena*) ∧
  header-size (*N* ∝ *i*) ≤ *i* ∧
  xarena-active-clause (*clause-slice arena N i*)
   (*the* (*fmlookup N i*))› **and**
 dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
   4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
 C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
 C-le: ‹*C* < *length arena*› **and**
 C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
  (*the* (*fmlookup N C*))›

using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*is-Act* (*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*))› **and**
[*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def arena-incr-act-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
xarena-active-clause (*clause-slice* (*arena-incr-act arena C*) *N i*) (*the* (*fmlookup N i*))›
if ‹*i* ∈# *dom-m N*›
for *i*
using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *arena-incr-act-def header-size-def ACTIVITY-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i ?arena*)›
if ‹*i* ∈ *vdom*›‹*i* ∉# *dom-m N*›‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)›
for *i*
proof −
have *i-ge*: ‹*i* ≥ *4*› ‹*i* < *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *arena-incr-act-def header-size-def ACTIVITY-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

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 −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*› **and**
lbd: ‹*is-Act* (*arena* ! (*j* − *ACTIVITY-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** *auto*
show *le′*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *ACTIVITY-SHIFT-def*)
show *?ge*
using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have *b*:
‹(*a* ! (*j* − *ACTIVITY-SHIFT*),
(*arena* ! (*j* − *ACTIVITY-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
show ‹(*a*[*j* − *ACTIVITY-SHIFT* := *a* ! (*j* − *ACTIVITY-SHIFT*) >> *Suc 0*], *arena-decr-act arena j*)
∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
unfolding *arena-decr-act-def*
by (*rule list-rel-update′*[*OF a*])
(*cases* ‹*arena* ! (*j* − *ACTIVITY-SHIFT*)›;
use lbd b **in** ‹*auto simp add*: *uint32-nat-rel-def br-def arena-el-rel-def*
Collect-eq-comp sum-mod-uint32-max-def nat-of-uint32-plus
nat-of-uint32-shiftr›)
qed

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*⟩
by (*intro frefI nres-relI*)
(*auto simp*: *isa-arena-decr-act-def arena-act-def arena-get-lbd-conv*
arena-act-pre-def arena-is-valid-clause-idx-def arena-decr-act-conv
list-rel-imp-same-length arena-act-conv
intro!: *ASSERT-leI*)

lemma *length-arena-decr-act*[*simp*]:
‹*length* (*arena-decr-act arena C*) = *length arena*›
by (*auto simp*: *arena-decr-act-def*)

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 −
let *?arena* = ‹*arena-decr-act arena C*›
have *act*: ‹∀ *i*∈#*dom-m N*.
i < *length* (*arena*) ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
C-le: ‹*C* < *length arena*› **and**
C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))›
using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*is-Act* (*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*))› **and**
[*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def arena-decr-act-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
xarena-active-clause (*clause-slice* (*arena-decr-act arena C*) *N i*) (*the* (*fmlookup N i*))›
if ‹*i* ∈# *dom-m N*›
for *i*

using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *arena-decr-act-def header-size-def ACTIVITY-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i ?arena*)›
if ‹$i \in vdom$›‹$i \notin\# $ *dom-m N*›‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i arena*)›
for *i*
proof −
have *i-ge*: ‹$i \geq 4$› ‹$i <$ *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *arena-decr-act-def header-size-def ACTIVITY-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

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* ∧ $C \geq$ *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: ‹$j \in\#$ *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* $\leq j$› (**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 −
have *j-le*: ‹$j <$ *length arena*› **and**
length: ‹*length* ($N \propto j$) = *arena-length arena j*› **and**
k1:‹$\bigwedge k$. $k <$ *length* ($N \propto j$) $\Longrightarrow$ $N \propto j$! k = *arena-lit arena* ($j + k$)› **and**
k2:‹$\bigwedge k$. $k <$ *length* ($N \propto j$) $\Longrightarrow$ *is-Lit* (*arena* ! ($j+k$))› **and**
le: ‹$j +$ *length* ($N \propto j$) $\leq$ *length arena*› **and**
j-ge: ‹*header-size* ($N \propto j$) $\leq j$› **and**

lbd: ‹*is-Status* (*arena* ! (*j* − *STATUS-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** *auto*
show *le′*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *STATUS-SHIFT-def*)
show *?ge*
using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have *b*:
‹(*a* ! (*j* − *STATUS-SHIFT*),
(*arena* ! (*j* − *STATUS-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
have [*simp*]: ‹(*a OR 4*) *AND 3* = *a AND 3*› **for** *a* :: *int*
supply [[*show-types*]]
by (*smt BIT-special-simps*(*1*) *BIT-special-simps*(*4*) *One-nat-def bbw-ao-dist expand-BIT*(*2*)
int-and-comm int-and-numerals(*17*) *int-and-numerals*(*3*) *int-or-extra-simps*(*1*)
numeral-One numeral-plus-one of-nat-numeral semiring-1-class.of-nat-simps(*1*)
semiring-1-class.of-nat-simps(*2*) *semiring-norm*(*2*) *semiring-norm*(*8*))
have *Pos*: ‹*b* ≥*0* ⟹ *a* ≤ *a OR b*› **for** *a b* :: *int*
by (*rule le-int-or*)
(*auto simp*: *bin-sign-def*)
have [*simp*]: ‹(*0*::*int*) ≤ *int a OR* (*4*::*int*)› **for** *a* :: *nat*
by (*rule order-trans*[*OF - Pos*])
auto
then have [*simp*]: ‹(*a OR 4*) *AND 3* = *a AND 3*› **for** *a* :: *nat*
supply [[*show-types*]]
unfolding *bitAND-nat-def bitOR-nat-def*
by *auto*

have [*simp*]: ‹(*a OR 4*) *AND 4* = *4*› **for** *a* :: *nat*
supply [[*show-types*]]
unfolding *bitAND-nat-def bitOR-nat-def*
by *auto*
have *nat-of-uint32-4*: ‹*4* = *nat-of-uint32 4*›
by *auto*
have [*simp*]: ‹*nat-of-uint32* (*a OR 4*) = *nat-of-uint32 a OR 4*› **for** *a*
by (*subst nat-of-uint32-4*, *subst nat-of-uint32-ao*) *simp*

show ‹(*a*[*j* − *STATUS-SHIFT* := *a* ! (*j* − *STATUS-SHIFT*) *OR 0b100*],
mark-used arena j) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
unfolding *mark-used-def*
by (*rule list-rel-update′*[*OF a*])
(*cases* ‹*arena* ! (*j* − *STATUS-SHIFT*)›;
use lbd b **in** ‹*auto simp add*: *uint32-nat-rel-def br-def arena-el-rel-def*
status-rel-def bitfield-rel-def
Collect-eq-comp sum-mod-uint32-max-def nat-of-uint32-plus›)
qed

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-mark-used-def arena-get-lbd-conv*

arena-act-pre-def arena-is-valid-clause-idx-def arena-incr-act-conv
list-rel-imp-same-length isa-mark-used-conv
intro!: *ASSERT-leI*)

lemma *length-mark-used*[*simp*]: ‹*length* (*mark-used arena C*) = *length arena*›
by (*auto simp*: *mark-used-def*)

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 −
let *?arena* = ‹*mark-used arena C*›
have *act*: ‹∀ *i*∈#*dom-m N*.
i < *length* (*arena*) ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
C-le: ‹*C* < *length arena*› **and**
C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))›
using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
AStatus (*xarena-status* (*clause-slice arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)))
True› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*)› **and**
[*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def mark-used-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
xarena-active-clause (*clause-slice* (*mark-used arena C*) *N i*) (*the* (*fmlookup N i*))›
if ‹*i* ∈# *dom-m N*›
for *i*
using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge

by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *mark-used-def header-size-def STATUS-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i ?arena*)›
if ‹$i \in$ *vdom*›‹$i \notin\#$ *dom-m N*›‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i arena*)›
for *i*
proof −
have *i-ge*: ‹$i \geq 4$› ‹$i <$ *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *mark-used-def header-size-def STATUS-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

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-SHIFT* := *AStatus* (*xarena-status* (*arena*!(*i* − *STATUS-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-SHIFT* < *length arena*› (**is** *?le*) **and**
‹*STATUS-SHIFT* ≤ *j*› (**is** *?ge*) **and**
‹(*a*[*j* − *STATUS-SHIFT* := *a* ! (*j* − *STATUS-SHIFT*) *AND 3*], *mark-unused arena j*) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
proof −
have *j-le*: ‹$j <$ *length arena*› **and**
length: ‹*length* ($N \propto j$) = *arena-length arena j*› **and**
k1:‹⋀*k*. *k* < *length* ($N \propto j$) ⟹ $N \propto j$! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2:‹⋀*k*. *k* < *length* ($N \propto j$) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* ($N \propto j$) ≤ *length arena*› **and**
j-ge: ‹*header-size* ($N \propto j$) ≤ *j*› **and**
lbd: ‹*is-Status* (*arena* ! (*j* − *STATUS-SHIFT*))›
using *arena-lifting*[*OF valid j*] **by** *auto*
show *le'*: *?le*

    using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
    by (*auto split*: *if-splits simp*: *STATUS-SHIFT-def*)
  show *?ge*
    using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
  have *b*:
    ‹(*a* ! (*j* − *STATUS-SHIFT*),
        (*arena* ! (*j* − *STATUS-SHIFT*)))
      ∈ *uint32-nat-rel O arena-el-rel*›
    by (*rule param-nth*[*OF* - - *a*]) (*use j-le* **in** *auto*)
  have [*simp*]: ‹(*a OR 4*) *AND 3* = *a AND 3*› **for** *a* :: *int*
    supply [[*show-types*]]
    by (*smt BIT-special-simps*(*1*) *BIT-special-simps*(*4*) *One-nat-def bbw-ao-dist expand-BIT*(*2*)
      int-and-comm int-and-numerals(*17*) *int-and-numerals*(*3*) *int-or-extra-simps*(*1*)
      numeral-One numeral-plus-one of-nat-numeral semiring-1-class.of-nat-simps(*1*)
      semiring-1-class.of-nat-simps(*2*) *semiring-norm*(*2*) *semiring-norm*(*8*))
  have *Pos*: ‹*b* ≥*0* ⟹ *a* ≤ *a OR b*› **for** *a b* :: *int*
    by (*rule le-int-or*)
      (*auto simp*: *bin-sign-def*)
  have [*simp*]: ‹(*0*::*int*) ≤ *int a OR* (*4*::*int*)› **for** *a* :: *nat*
    by (*rule order-trans*[*OF* - *Pos*])
      auto
  then have [*simp*]: ‹(*a OR 4*) *AND 3* = *a AND 3*› **for** *a* :: *nat*
    supply [[*show-types*]]
    unfolding *bitAND-nat-def bitOR-nat-def*
    by *auto*

  have [*simp*]: ‹(*a OR 4*) *AND 4* = *4*› **for** *a* :: *nat*
    supply [[*show-types*]]
    unfolding *bitAND-nat-def bitOR-nat-def*
    by *auto*
  have *nat-of-uint32-4*: ‹*3* = *nat-of-uint32 3*›
    by *auto*
  have [*simp*]: ‹*nat-of-uint32* (*a AND 3*) = *nat-of-uint32 a AND 3*› **for** *a*
    by (*subst nat-of-uint32-4*, *subst nat-of-uint32-ao*) *simp*

  show ‹(*a*[*j* − *STATUS-SHIFT* := *a* ! (*j* − *STATUS-SHIFT*) *AND 3*],
      mark-unused arena j) ∈ ⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel*›
    unfolding *mark-unused-def*
    supply [[*show-types*]]
    by (*rule list-rel-update′*[*OF a*])
      (*cases* ‹*arena* ! (*j* − *STATUS-SHIFT*)›;
      use lbd b **in** ‹*auto simp add*: *uint32-nat-rel-def br-def arena-el-rel-def*
          status-rel-def bitfield-rel-def nat-0-AND
          Collect-eq-comp sum-mod-uint32-max-def nat-of-uint32-plus›)
qed

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*›
  by (*intro frefI nres-relI*)
   (*auto simp*: *isa-mark-unused-def arena-get-lbd-conv*
      arena-act-pre-def arena-is-valid-clause-idx-def arena-incr-act-conv
      list-rel-imp-same-length isa-mark-unused-conv

intro!: *ASSERT-leI*)

lemma *length-mark-unused*[*simp*]: ‹*length* (*mark-unused arena C*) = *length arena*›
  by (*auto simp*: *mark-unused-def*)

lemma *valid-arena-mark-unused*:
  assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
  shows
   ‹*valid-arena* (*mark-unused arena C*) *N vdom*›
proof −
  let *?arena* = ‹*mark-unused arena C*›
  have *act*: ‹∀ *i*∈#*dom-m N*.
     i < *length* (*arena*) ∧
     header-size (*N* ∝ *i*) ≤ *i* ∧
     xarena-active-clause (*clause-slice arena N i*)
      (*the* (*fmlookup N i*))› **and**
    dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
         4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
    C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
    C-le: ‹*C* < *length arena*› **and**
    C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
      (*the* (*fmlookup N C*))›
    using *assms*
    by (*auto simp*: *valid-arena-def*)
  have
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
         clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
         AStatus (*xarena-status* (*clause-slice arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)))
           False› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
         clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
   [*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
         clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
   [*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*) =
         clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*)› **and**
   [*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
     Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
    using *C-le C-ge* **unfolding** *SHIFTS-def mark-unused-def header-size-def*
    by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

  have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
    using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
    by *simp*

  then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
     xarena-active-clause (*clause-slice* (*mark-unused arena C*) *N i*) (*the* (*fmlookup N i*))›
    if ‹*i* ∈# *dom-m N*›
    for *i*
    using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
      minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
      that C-ge
    by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)

(*auto simp*: *mark-unused-def header-size-def STATUS-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i* *?arena*)›
if ‹*i* ∈ *vdom*›‹*i* ∉# *dom-m N*›‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i* *arena*)›
for *i*
proof −
have *i-ge*: ‹*i* ≥ *4*› ‹*i* < *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *mark-unused-def header-size-def STATUS-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

Marked as used? **definition** *marked-as-used* :: ‹*arena* ⇒ *nat* ⇒ *bool*› **where**
‹*marked-as-used arena C* = *xarena-used* (*arena* ! (*C* − *STATUS-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-SHIFT* < *length arena* ∧ *C* ≥ *STATUS-SHIFT*);
RETURN (*arena* ! (*C* − *STATUS-SHIFT*) *AND 4* ≠ *0*)
}›

lemma *arena-marked-as-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*) *AND 4* ≠ *0* ⟷
marked-as-used arena j›
proof −
have *j-le*: ‹*j* < *length arena*› **and**
length: ‹*length* (*N* ∝ *j*) = *arena-length arena j*› **and**
k1: ‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *N* ∝ *j* ! *k* = *arena-lit arena* (*j* + *k*)› **and**
k2: ‹⋀*k*. *k* < *length* (*N* ∝ *j*) ⟹ *is-Lit* (*arena* ! (*j*+*k*))› **and**
le: ‹*j* + *length* (*N* ∝ *j*) ≤ *length arena*› **and**
j-ge: ‹*header-size* (*N* ∝ *j*) ≤ *j*› **and**
lbd: ‹*is-Status* (*arena* ! (*j* − *STATUS-SHIFT*))›

using *arena-lifting*[*OF valid j*] **by** (*auto simp*:)
show *le'*: *?le*
using *le j-ge* **unfolding** *length*[*symmetric*] *header-size-def*
by (*auto split*: *if-splits simp*: *STATUS-SHIFT-def*)
show *?ge*
using *j-ge* **by** (*auto simp*: *SHIFTS-def header-size-def split*: *if-splits*)
have [*simp*]: ‹$a \neq 0 \longleftrightarrow$ *nat-of-uint32* $a \neq 0$› **for** *a*:: *uint32*
by (*simp add*: *nat-of-uint32-0-iff*)
have
‹(*a* ! (*j* − *STATUS-SHIFT*),
(*arena* ! (*j* − *STATUS-SHIFT*)))
∈ *uint32-nat-rel O arena-el-rel*›
by (*rule param-nth*[*OF - - a*]) (*use j-le* **in** *auto*)
then show ‹*a* ! (*j* − *STATUS-SHIFT*) *AND 4* ≠ *0* ⟷
marked-as-used arena j›
using *lbd* **by** (*cases* ‹*arena* ! (*j* − *STATUS-SHIFT*)›)
(*auto simp*: *arena-el-rel-def bitfield-rel-def nat-of-uint32-ao*[*symmetric*]
marked-as-used-def uint32-nat-rel-def br-def)
qed

lemma *isa-marked-as-used-marked-as-used*:
‹(*uncurry isa-marked-as-used*, *uncurry* (*RETURN oo marked-as-used*)) ∈
[*uncurry marked-as-used-pre*]$_f$
⟨*uint32-nat-rel O arena-el-rel*⟩*list-rel* $\times_f$ *nat-rel* →
⟨*bool-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *marked-as-used-pre-def arena-marked-as-used-conv*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def
list-rel-imp-same-length isa-marked-as-used-def
intro!: *ASSERT-leI*)

lemma *valid-arena-vdom-le*:
assumes ‹*valid-arena arena N ovdm*›
shows ‹*finite ovdm*› **and** ‹*card ovdm* ≤ *length arena*›
proof −
have *incl*: ‹*ovdm* ⊆ {*4*..< *length arena*}›
apply *auto*
using *assms valid-arena-in-vdom-le-arena* **by** *blast*+
from *card-mono*[*OF - this*] **show** ‹*card ovdm* ≤ *length arena*› **by** *auto*
have ‹*length arena* ≥ *4* ∨ *ovdm* = {}›
using *incl* **by** *auto*
with *card-mono*[*OF - incl*] **have** ‹*ovdm* ≠ {} ⟹ *card ovdm* < *length arena*›
by *auto*
from *finite-subset*[*OF incl*] **show** ‹*finite ovdm*› **by** *auto*
qed

lemma *valid-arena-vdom-subset*:
assumes ‹*valid-arena arena N* (*set vdom*)› **and** ‹*distinct vdom*›
shows ‹*length vdom* ≤ *length arena*›
proof −
have ‹*set vdom* ⊆ {*0* ..< *length arena*}›
using *assms* **by** (*auto simp*: *valid-arena-def*)
from *card-mono*[*OF - this*] **show** *?thesis* **using** *assms* **by** (*auto simp*: *distinct-card*)

qed

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* ⇒ *nat*› ::
‹(*Id* :: (*nat literal* × -) *set*) → (*Id* :: (*nat* × -) *set*)› **.**

lemma [*def-pat-rules*]:
‹*atm-of* ≡ *op-atm-of*›
by *auto*

sepref-decl-op *lit-of*: ‹*lit-of* :: (*nat*, *nat*) *ann-lit* ⇒ *nat literal*› ::
‹(*Id* :: ((*nat*, *nat*) *ann-lit* × -) *set*) → (*Id* :: (*nat literal* × -) *set*)› **.**

lemma [*def-pat-rules*]:
‹*lit-of* ≡ *op-lit-of*›
by *auto*

sepref-decl-op *watched-app*:
‹*watched-app* ::(*nat literal* ⇒ (*nat* × -) *list*) ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*›
::
‹(*Id* :: ((*nat literal* ⇒ (*nat watcher*) *list*) × -) *set*) → (*Id* :: (*nat literal* × -) *set*) → *nat-rel* →
nat-rel $\times_r$ (*Id* :: (*nat literal* × -) *set*) $\times_r$ *bool-rel*›
.

lemma (**in** −) *safe-minus-nat-assn*:
‹(*uncurry* (*return oo* (−)), *uncurry* (*RETURN oo fast-minus*)) ∈
$[\lambda(m, n).\ m \geq n]_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} → *nat-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *uint32-nat-rel-def br-def nat-of-uint32-le-minus*
nat-of-uint32-notle-minus nat-of-uint32-le-iff)

definition *map-fun-rel-assn*
:: ‹(*nat* × *nat literal*) *set* ⇒ (′*a* ⇒ ′*b* ⇒ *assn*) ⇒ (*nat literal* ⇒ ′*a*) ⇒ ′*b list* ⇒ *assn*›
where
‹*map-fun-rel-assn D R* = *pure* (⟨*the-pure R*⟩*map-fun-rel D*)›

lemma [*safe-constraint-rules*]: ‹*is-pure* (*map-fun-rel-assn D R*)›
unfolding *map-fun-rel-assn-def* **by** *auto*
abbreviation *nat-lit-assn* :: ‹*nat literal* ⇒ *nat* ⇒ *assn*› **where**
‹*nat-lit-assn* ≡ *pure nat-lit-rel*›

abbreviation *unat-lit-assn* :: ‹*nat literal* ⇒ *uint32* ⇒ *assn*› **where**
‹*unat-lit-assn* ≡ *pure unat-lit-rel*›

lemma *hr-comp-uint32-nat-assn-nat-lit-rel*[*simp*]:
‹*hr-comp uint32-nat-assn nat-lit-rel* = *unat-lit-assn*›
by (*auto simp*: *hrp-comp-def hr-comp-def uint32-nat-rel-def*
hr-comp-pure br-def unat-lit-rel-def)

abbreviation *pair-nat-ann-lit-assn* :: ‹(*nat*, *nat*) *ann-lit* ⇒ *ann-lit-wl* ⇒ *assn*› **where**
‹*pair-nat-ann-lit-assn* ≡ *pure nat-ann-lit-rel*›

abbreviation *pair-nat-ann-lits-assn* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *ann-lits-wl* ⇒ *assn*› **where**
‹*pair-nat-ann-lits-assn* ≡ *list-assn pair-nat-ann-lit-assn*›

abbreviation *pair-nat-ann-lit-fast-assn* :: ‹(*nat*, *nat*) *ann-lit* ⇒ *ann-lit-wl-fast* ⇒ *assn*› **where**
‹*pair-nat-ann-lit-fast-assn* ≡ *hr-comp* (*uint32-assn* ∗*a option-assn uint64-nat-assn*) *nat-ann-lit-rel*›

abbreviation *pair-nat-ann-lits-fast-assn* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *ann-lits-wl-fast* ⇒ *assn*› **where**
‹*pair-nat-ann-lits-fast-assn* ≡ *list-assn pair-nat-ann-lit-fast-assn*›

Code

lemma [*sepref-fr-rules*]: ‹(*return o id*, *RETURN o nat-of-lit*) ∈ $unat\text{-}lit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
by *sepref-to-hoare*
(*sep-auto simp*: *uint32-nat-rel-def br-def unat-lit-rel-def nat-lit-rel-def*)

lemma ‹(*return o* (λ*n*. *shiftr n 1*), *RETURN o shiftr1*) ∈ $word\text{-}nat\text{-}assn^k \rightarrow_a word\text{-}nat\text{-}assn$›
by *sepref-to-hoare* (*sep-auto simp*: *shiftr1-def word-nat-rel-def unat-shiftr br-def*)

lemma *propagated-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo propagated*), *uncurry* (*RETURN oo Propagated*)) ∈
$unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn$›
by *sepref-to-hoare* (*sep-auto simp*: *nat-ann-lit-rel-def propagated-def case-prod-beta p2rel-def*
br-def uint32-nat-rel-def unat-lit-rel-def nat-lit-rel-def
split: *option.splits*)

lemma *decided-hnr*[*sepref-fr-rules*]:
‹(*return o decided*, *RETURN o Decided*) ∈
$unat\text{-}lit\text{-}assn^k \rightarrow_a pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn$›
by *sepref-to-hoare* (*sep-auto simp*: *nat-ann-lit-rel-def decided-def case-prod-beta p2rel-def*
br-def uint32-nat-rel-def unat-lit-rel-def nat-lit-rel-def
split: *option.splits*)

lemma *uminus-lit-hnr*[*sepref-fr-rules*]:
‹(*return o uminus-code*, *RETURN o uminus*) ∈
$unat\text{-}lit\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
proof −
have [*simp*]: ‹*nat-of-uint32 1* = *1*›
by *transfer auto*
have [*simp*]: ‹*0 XOR Suc 0* = *Suc 0*›
unfolding *bitXOR-nat-def*
by *auto*
have [*simp*]: ‹*bin-last* (*2* + *n*) = *bin-last n*› **for** *n*
by (*auto simp*: *bin-last-def*)
have [*simp*]: ‹*bin-rest* (*2* + *n*) = *1* + (*bin-rest n*)› **for** *n*
by (*auto simp*: *bin-rest-def*)
have ‹*bin-nth* (*2* + *n XOR 1*) *na* ⟷ *bin-nth* (*2* + (*n XOR 1*)) *na*› **for** *n na*
by (*induction na*) *auto*

then have [*simp*]: ‹((*2* + *n*) *XOR 1*) = *2* + (((*n XOR 1*)))› **for** *n* :: *int*
by (*auto simp*: *bin-eq-iff*)

have [*simp*]: ‹*Suc* (*Suc n*) *XOR Suc 0* = *Suc* (*Suc* (*n XOR Suc 0*))› **for** *n* :: *nat*
unfolding *bitXOR-nat-def*
by (*auto simp*: *nat-add-distrib*)

have [*simp*]: ‹*2 ∗ q XOR Suc 0 = Suc (2 ∗ q)*› **for** *q* :: *nat*
by (*induction q*) *auto*

have *1*: ‹(*return o* (*λL. bitXOR L 1*), *RETURN o uminus-lit-imp*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
unfolding *bitXOR-1-if-mod-2 uminus-lit-imp-def*
apply *sepref-to-hoare*
apply (*sep-auto simp*: *nat-ann-lit-rel-def uminus-lit-imp-def case-prod-beta p2rel-def uint32-nat-rel-def br-def nat-of-uint32-XOR bitXOR-1-if-mod-2 split*: *option.splits*)
using *One-nat-def bitXOR-1-if-mod-2* **by** *presburger*
show *?thesis*
using *1* [*FCOMP uminus-lit-imp-uminus*[*unfolded convert-fref*]] **unfolding** *unat-lit-rel-def uminus-code-def*
.
qed

abbreviation *ann-lit-wl-assn* :: ‹*ann-lit-wl ⇒ ann-lit-wl ⇒ 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*) ∈ $unat\text{-}lit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
by *sepref-to-hoare* (*sep-auto simp*: *p2rel-def uint32-nat-rel-def br-def Collect-eq-comp unat-lit-rel-def nat-of-uint32-shiftr nat-lit-rel-def atm-of-code-def*)

lemma *lit-of-hnr*[*sepref-fr-rules*]:
‹(*return o fst*, *RETURN o op-lit-of*) ∈ $pair\text{-}nat\text{-}ann\text{-}lit\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
by *sepref-to-hoare*
(*sep-auto simp*: *p2rel-def nat-ann-lit-rel-def uint32-nat-rel-def*

Collect-eq-comp br-def unat-lit-rel-def nat-lit-rel-def ann-lit-of-pair-alt-def
split: *if-splits*)

lemma *lit-of-fast-hnr*[*sepref-fr-rules*]:
‹(*return o fst*, *RETURN o op-lit-of*) $\in$ *pair-nat-ann-lit-fast-assn*^{k} $\rightarrow_a$ *unat-lit-assn*›
apply *sepref-to-hoare*
apply (*sep-auto simp*: *unat-lit-rel-def uint32-nat-rel-def*)
apply (*sep-auto simp*: *p2rel-def nat-ann-lit-rel-def uint32-nat-rel-def*
Collect-eq-comp br-def unat-lit-rel-def nat-lit-rel-def ann-lit-of-pair-alt-def
hr-comp-def prod.splits case-prod-beta
split: *if-splits*)
done

lemma *op-eq-op-nat-lit-eq*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) $\in$
(*pure unat-lit-rel*)^k $*_a$ (*pure unat-lit-rel*)^k $\rightarrow_a$ *bool-assn*›
proof −
have [*simp*]: ‹*even bi* $\Longrightarrow$ *even ai* $\Longrightarrow$ *ai div 2* = *bi div 2* $\Longrightarrow$ *ai* = *bi*› **for** *ai bi* :: *nat*
by *presburger*
have [*dest*]: ‹*odd bi* $\Longrightarrow$ *odd ai* $\Longrightarrow$ *ai div 2* = *bi div 2* $\Longrightarrow$ *ai* = *bi*› **for** *ai bi* :: *nat*
by *presburger*
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *p2rel-def nat-ann-lit-rel-def nat-lit-rel-def*
br-def Collect-eq-comp uint32-nat-rel-def dvd-div-eq-iff unat-lit-rel-def
split: *if-splits*)+
qed

lemma (**in** −) *is-pos-hnr*[*sepref-fr-rules*]:
‹(*return o is-pos-code*, *RETURN o is-pos*) $\in$ *unat-lit-assn*^{k} $\rightarrow_a$ *bool-assn*›
proof −
have *1*: ‹(*RETURN o* (λ*L*. *bitAND L 1* = *0*), *RETURN o is-pos*) $\in$ *nat-lit-rel* $\rightarrow_f$ $\langle$*bool-rel*$\rangle$*nres-rel*›
unfolding *bitAND-1-mod-2 is-pos-code-def*
by (*intro nres-relI frefI*) (*auto simp*: *nat-lit-rel-def br-def split*: *if-splits*, *presburger*)
have *2*: ‹(*return o is-pos-code*, *RETURN o* (λ*L*. *bitAND L 1* = *0*)) $\in$ *uint32-nat-assn*^{k} $\rightarrow_a$ *bool-assn*›
apply *sepref-to-hoare*
using *nat-of-uint32-ao*[*of - 1*]
by (*sep-auto simp*: *p2rel-def unat-lit-rel-def uint32-nat-rel-def*
nat-lit-rel-def br-def is-pos-code-def
nat-of-uint32-0-iff nat-0-AND uint32-0-AND
split: *if-splits*)+
show *?thesis*
using *2*[*FCOMP 1*[*unfolded convert-fref*]] **unfolding** *unat-lit-rel-def* **.**
qed

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*) $\in$
[λ*L*. ¬*is-decided L*]$_a$ *pair-nat-ann-lit-assn*^{k} $\rightarrow$ (*unat-lit-assn* $*a$ *nat-assn*)›
unfolding *lit-and-ann-of-propagated-code-def*
apply *sepref-to-hoare*
apply (*rename-tac x x'*)
apply (*case-tac x*)
by (*sep-auto simp*: *nat-ann-lit-rel-def p2rel-def nat-lit-rel-def*
Propagated-eq-ann-lit-of-pair-iff unat-lit-rel-def uint32-nat-rel-def Collect-eq-comp
br-def Collect-eq-comp nat-lit-rel-def
simp del: *literal-of-nat.simps*)+

lemma *Pos-unat-lit-assn*:
⟨(*return o* (λn. *two-uint32* $*$ *n*), *RETURN o Pos*) $\in$ [λL. *Pos L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}$]$_a$ *uint32-nat-assn*^{k} $\rightarrow$
unat-lit-assn⟩
by *sepref-to-hoare*
(*sep-auto simp*: *unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def Collect-eq-comp*
nat-of-uint32-distrib-mult2)

lemma *Neg-unat-lit-assn*:
⟨(*return o* (λn. *two-uint32* $*$ *n* +*1*), *RETURN o Neg*) $\in$ [λL. *Pos L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}$]$_a$ *uint32-nat-assn*^{k} $\rightarrow$
unat-lit-assn⟩
by *sepref-to-hoare*
(*sep-auto simp*: *unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def Collect-eq-comp*
nat-of-uint32-distrib-mult2-plus1 uint-max-def)

lemma *Pos-unat-lit-assn'*:
⟨(*return o* (λn. *two-uint32* $*$ *n*), *RETURN o Pos*) $\in$ [λL. $L \leq$ *uint-max div 2*]$_a$ *uint32-nat-assn*^{k} $\rightarrow$
unat-lit-assn⟩
by *sepref-to-hoare*
(*sep-auto simp*: *unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def Collect-eq-comp*
nat-of-uint32-distrib-mult2 uint-max-def)

lemma *Neg-unat-lit-assn'*:
⟨(*return o* (λn. *two-uint32* $*$ *n* + *1*), *RETURN o Neg*) $\in$ [λL. $L \leq$ *uint-max div 2*]$_a$ *uint32-nat-assn*^{k}
$\rightarrow$
unat-lit-assn⟩
by *sepref-to-hoare*
(*sep-auto simp*: *unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def Collect-eq-comp*
nat-of-uint32-distrib-mult2 uint-max-def nat-of-uint32-add)

0.1.5 Declaration of some Operators and Implementation

sepref-register ⟨*watched-by* :: *nat twl-st-wl* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat watched*⟩
:: ⟨*nat twl-st-wl* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat watched*⟩

lemma [*def-pat-rules*]:
⟨*watched-app* \$ *M* \$ *L* \$ *i* $\equiv$ *op-watched-app* \$ *M* \$ *L* \$ *i*⟩
by (*auto simp*: *watched-app-def*)

sepref-definition *is-decided-wl-code*
is ⟨(*RETURN o is-decided-wl*)⟩
:: ⟨*ann-lit-wl-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *is-decided-wl-def*[*abs-def*]
by *sepref*

sepref-definition *is-decided-wl-fast-code*
is ⟨(*RETURN o is-decided-wl*)⟩
:: ⟨*ann-lit-wl-fast-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *is-decided-wl-def*[*abs-def*]
by *sepref*

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 −
have *1*: ‹*hr-comp ann-lit-wl-assn nat-ann-lit-rel* = *pair-nat-ann-lit-assn*›
by (*fastforce simp*: *case-prod-beta hr-comp-def*[*abs-def*] *pure-def nat-ann-lit-rel-def*
prod-assn-def ann-lit-of-pair-if ex-assn-def imp-ex Abs-assn-eqI(*1*) *ex-simps*(*1*)[*symmetric*]
simp del: *pair-of-ann-lit.simps literal-of-nat.simps ex-simps*(*1*)
split: *if-splits*)
show *?slow*
using *is-decided-wl-code.refine*[*FCOMP is-decided-wl-is-decided*]
unfolding *1* **.**
show *?fast*
using *is-decided-wl-fast-code.refine*[*FCOMP is-decided-wl-is-decided*]
unfolding *1* **.**
qed

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*)) ∈ *unit-assn*^{k} →$_a$ *clause-status-assn*›
by *sepref-to-hoare sep-auto*

lemma *LEARNED-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return LEARNED*), *uncurry0* (*RETURN LEARNED*)) ∈ *unit-assn*^{k} →$_a$ *clause-status-assn*›
by *sepref-to-hoare sep-auto*

lemma *DELETED-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return DELETED*), *uncurry0* (*RETURN DELETED*)) ∈ *unit-assn*^{k} →$_a$ *clause-status-assn*›
by *sepref-to-hoare sep-auto*

lemma *ACTIVITY-SHIFT-hnr*:
‹(*uncurry0* (*return 3*), *uncurry0* (*RETURN ACTIVITY-SHIFT*)) ∈ *unit-assn*^{k} →$_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *ACTIVITY-SHIFT-def uint64-nat-rel-def br-def*)
lemma *STATUS-SHIFT-hnr*:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN STATUS-SHIFT*)) ∈ *unit-assn*^{k} →$_a$ *uint64-nat-assn*›

by *sepref-to-hoare* (*sep-auto simp*: *STATUS-SHIFT-def uint64-nat-rel-def br-def*)

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN SIZE-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *SIZE-SHIFT-def*)

lemma [*sepref-fr-rules*]:
‹(*return o id*, *RETURN o xarena-length*) $\in$ [*is-Size*]$_a$ *arena-el-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *SIZE-SHIFT-def uint32-nat-rel-def arena-el-rel-def br-def hr-comp-def split*: *arena-el.splits*)

lemma (**in** −) *POS-SHIFT-uint64-hnr*:
‹(*uncurry0* (*return 5*), *uncurry0* (*RETURN POS-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *POS-SHIFT-def uint64-nat-rel-def br-def*)

lemma *nat-of-uint64-eq-2-iff*[*simp*]: ‹*nat-of-uint64 c* = *2* $\longleftrightarrow$ *c* = *2*›
using *word-nat-of-uint64-Rep-inject* **by** *fastforce*

lemma *arena-el-assn-alt-def*:
‹*arena-el-assn* = *hr-comp uint32-assn* (*uint32-nat-rel O arena-el-rel*)›
by (*auto simp*: *hr-comp-assoc*[*symmetric*])

lemma *arena-el-comp*: ‹*hn-val* (*uint32-nat-rel O arena-el-rel*) = *hn-ctxt arena-el-assn*›
by (*auto simp*: *hn-ctxt-def arena-el-assn-alt-def*)

lemma *status-assn-hnr-eq*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) $\in$ *status-assn*^{k} $*_a$ *status-assn*^{k} $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *status-rel-def hr-comp-def uint32-nat-rel-def br-def nat-of-uint32-0-iff nat-of-uint32-Suc03-iff nat-of-uint32-013-neq*)

lemma *IRRED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 0*), *uncurry0* (*RETURN IRRED*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *status-assn*›
by (*sepref-to-hoare*) (*sep-auto simp*: *status-rel-def hr-comp-def uint32-nat-rel-def br-def*)

lemma *LEARNED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN LEARNED*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *status-assn*›
by (*sepref-to-hoare*) (*sep-auto simp*: *status-rel-def hr-comp-def uint32-nat-rel-def br-def*)

lemma *DELETED-status-assn*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 3*), *uncurry0* (*RETURN DELETED*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *status-assn*›
by (*sepref-to-hoare*) (*sep-auto simp*: *status-rel-def hr-comp-def uint32-nat-rel-def br-def nat-of-uint32-Suc03-iff*)

lemma *status-assn-alt-def*:
‹*status-assn* = *pure* (*uint32-nat-rel O status-rel*)›
unfolding *hr-comp-pure* **by** *simp*

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 2*), *uncurry0* (*RETURN LBD-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *LBD-SHIFT-def*)

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN STATUS-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *STATUS-SHIFT-def*)

lemma (**in** −) *LBD-SHIFT-hnr*:
‹(*uncurry0* (*return 2*), *uncurry0* (*RETURN LBD-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *LBD-SHIFT-def uint64-nat-rel-def br-def*)

lemma *MAX-LENGTH-SHORT-CLAUSE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN MAX-LENGTH-SHORT-CLAUSE*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def*)

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

lemma *four-uint32-hnr*:
‹(*uncurry0* (*return 4*), *uncurry0* (*RETURN* (*four-uint32* :: *uint32*))) $\in$ *unit-assn*^{k} $\rightarrow_a$ *uint32-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint32-nat-rel-def br-def four-uint32-def*)

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 5*), *uncurry0* (*RETURN POS-SHIFT*)) $\in$ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *SHIFTS-def*)

lemma [*sepref-fr-rules*]:
‹(*return o id*, *RETURN o xarena-lit*) $\in$ [*is-Lit*]$_a$ *arena-el-assn*^{k} $\rightarrow$ *unat-lit-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *SIZE-SHIFT-def uint32-nat-rel-def unat-lit-rel-def arena-el-rel-def br-def hr-comp-def split*: *arena-el.splits*)

sepref-definition *isa-arena-length-code*
is ‹*uncurry isa-arena-length*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-length-def*
by *sepref*

lemma *isa-arena-length-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-length*))
$\in$ [*uncurry arena-is-valid-clause-idx*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
using *isa-arena-length-code.refine*[*FCOMP isa-arena-length-arena-length*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl-assn-comp*)

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*›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
minus-uint64-nat-assn[*sepref-fr-rules*]
unfolding *isa-arena-length-def SIZE-SHIFT-def fast-minus-def one-uint64-nat-def*[*symmetric*]
by *sepref*

lemma *isa-arena-length-fast-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-fast-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-length*))
$\in$ [*uncurry arena-is-valid-clause-idx*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
using *isa-arena-length-fast-code.refine*[*FCOMP isa-arena-length-arena-length*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl64-assn-comp*)

sepref-definition *isa-arena-length-fast-code2*

is ‹*uncurry isa-arena-length*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
minus-uint64-nat-assn[*sepref-fr-rules*]
unfolding *isa-arena-length-def SIZE-SHIFT-def fast-minus-def one-uint64-nat-def*[*symmetric*]
by *sepref*

lemma *isa-arena-length-fast-code2-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-length-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-length*))
$\in$ [*uncurry arena-is-valid-clause-idx*]$_a$
$arena\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint64\text{-}nat\text{-}assn$›
using *isa-arena-length-fast-code2.refine*[*FCOMP isa-arena-length-arena-length*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl-assn-comp*)

sepref-definition *isa-arena-lit-code*
is ‹*uncurry isa-arena-lit*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-lit-def*
by *sepref*

lemma *isa-arena-lit-code-refine*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-lit-code*, *uncurry* (*RETURN* ∘∘ *arena-lit*))
$\in$ [*uncurry arena-lit-pre*]$_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
using *isa-arena-lit-code.refine*[*FCOMP isa-arena-lit-arena-lit*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl-assn-comp*)

sepref-definition (**in**−) *isa-arena-lit-fast-code*
is ‹*uncurry isa-arena-lit*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-lit-def*
by *sepref*

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*))
$\in$ [*uncurry arena-lit-pre*]$_a$
$arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$›
using *isa-arena-lit-fast-code.refine*[*FCOMP isa-arena-lit-arena-lit*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl64-assn-comp*)

sepref-definition (**in**−) *isa-arena-lit-fast-code2*
is ‹*uncurry isa-arena-lit*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-lit-def*
by *sepref*

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*))
∈ [*uncurry arena-lit-pre*]$_a$
arena-assn^k $*_a$ *uint64-nat-assn*^{k} → *unat-lit-assn*⟩
using *isa-arena-lit-fast-code2.refine*[*FCOMP isa-arena-lit-arena-lit*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl-assn-comp*)

sepref-definition *arena-status-code*
is ⟨*uncurry isa-arena-status*⟩
:: ⟨(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*⟩
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-status-def*
by *sepref*

lemma *isa-arena-status-refine*[*sepref-fr-rules*]:
⟨(*uncurry arena-status-code*, *uncurry* (*RETURN* ∘∘ *arena-status*))
∈ [*uncurry arena-is-valid-clause-vdom*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} → *status-assn*⟩
using *arena-status-code.refine*[*FCOMP isa-arena-status-arena-status*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp status-assn-alt-def*
by (*simp add*: *arl-assn-comp*)

sepref-definition *swap-lits-code*
is ⟨*Sepref-Misc.uncurry3 isa-arena-swap*⟩
:: ⟨*nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ (*arl-assn uint32-assn*)^d $\rightarrow_a$ *arl-assn uint32-assn*⟩
unfolding *isa-arena-swap-def WB-More-Refinement-List.swap-def IICF-List.swap-def*[*symmetric*]
by *sepref*

lemma *swap-lits-refine*[*sepref-fr-rules*]:
⟨(*uncurry3 swap-lits-code*, *uncurry3* (*RETURN oooo swap-lits*))
∈ [*uncurry3 swap-lits-pre*]$_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *arena-assn*^{d} → *arena-assn*⟩
using *swap-lits-code.refine*[*FCOMP isa-arena-swap*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp*)

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

lemma *update-lbd-hnr*[*sepref-fr-rules*]:
⟨(*uncurry2 isa-update-lbd-code*, *uncurry2* (*RETURN ooo update-lbd*))
∈ [*update-lbd-pre*]$_a$ *nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-assn*^{d} → *arena-assn*⟩
using *isa-update-lbd-code.refine*[*FCOMP isa-update-lbd*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

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

supply *LBD-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-update-lbd-def*
by *sepref*

lemma *update-lbd-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-lbd-fast-code*, *uncurry2* (*RETURN ooo update-lbd*))
$\in$ [*update-lbd-pre*]$_a$ *uint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow$ *arena-fast-assn*›
using *isa-update-lbd-fast-code.refine*[*FCOMP isa-update-lbd*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

sepref-definition (**in** $-$)*isa-update-lbd-fast-code2*
is ‹*uncurry2 isa-update-lbd*›
:: ‹*uint64-nat-assn*^{k} $*_a$ *uint32-assn*^{k} $*_a$ (*arl-assn uint32-assn*)^d $\rightarrow_a$ *arl-assn uint32-assn*›
supply *LBD-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-update-lbd-def*
by *sepref*

lemma *update-lbd-fast-hnr2*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-lbd-fast-code2*, *uncurry2* (*RETURN ooo update-lbd*))
$\in$ [*update-lbd-pre*]$_a$ *uint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-assn*^{d} $\rightarrow$ *arena-assn*›
using *isa-update-lbd-fast-code2.refine*[*FCOMP isa-update-lbd*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-get-clause-LBD-code*
is ‹*uncurry isa-get-clause-LBD*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
unfolding *isa-get-clause-LBD-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-get-clause-LBD-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-clause-LBD-code*, *uncurry* (*RETURN* $\circ\circ$ *get-clause-LBD*))
$\in$ [*uncurry get-clause-LBD-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
using *isa-get-clause-LBD-code.refine*[*FCOMP isa-get-clause-LBD-get-clause-LBD*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-get-saved-pos-fast-code*
is ‹*uncurry isa-get-saved-pos*›
:: ‹(*arl64-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
supply *sum-uint64-assn*[*sepref-fr-rules*] *POS-SHIFT-uint64-hnr*[*sepref-fr-rules*]
unfolding *isa-get-saved-pos-def*
by *sepref*

lemma *get-saved-pos-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-fast-code*, *uncurry* (*RETURN* $\circ\circ$ *arena-pos*))
$\in$ [*uncurry get-saved-pos-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
using *isa-get-saved-pos-fast-code.refine*[*FCOMP isa-get-saved-pos-get-saved-pos*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

sepref-definition *isa-get-saved-pos-code*
is ‹*uncurry isa-get-saved-pos*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
supply *sum-uint64-assn*[*sepref-fr-rules*]

unfolding *isa-get-saved-pos-def POS-SHIFT-def*
by *sepref*

lemma *get-saved-pos-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-code*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
∈ [*uncurry get-saved-pos-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} → *uint64-nat-assn*›
using *isa-get-saved-pos-code.refine*[*FCOMP isa-get-saved-pos-get-saved-pos*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-get-saved-pos-code′*
is ‹*uncurry isa-get-saved-pos′*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *nat-assn*›
supply *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-get-saved-pos-def isa-get-saved-pos′-def*
by *sepref*

lemma *get-saved-pos-code′*:
‹(*uncurry isa-get-saved-pos-code′*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
∈ [*uncurry get-saved-pos-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} → *nat-assn*›
using *isa-get-saved-pos-code′.refine*[*FCOMP isa-get-saved-pos-get-saved-pos′*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-get-saved-pos-fast-code2*
is ‹*uncurry isa-get-saved-pos*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-assn*›
supply *sum-uint64-assn*[*sepref-fr-rules*] *POS-SHIFT-uint64-hnr*[*sepref-fr-rules*]
unfolding *isa-get-saved-pos-def*
by *sepref*

lemma *get-saved-pos-code2*[*sepref-fr-rules*]:
‹(*uncurry isa-get-saved-pos-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-pos*))
∈ [*uncurry get-saved-pos-pre*]$_a$ *arena-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *uint64-nat-assn*›
using *isa-get-saved-pos-fast-code2.refine*[*FCOMP isa-get-saved-pos-get-saved-pos*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

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*›
supply *minus-uint32-assn*[*sepref-fr-rules*]
unfolding *isa-update-pos-def*
by *sepref*

lemma *isa-update-pos-code-hnr*[*sepref-fr-rules*]:
‹(*uncurry2 isa-update-pos-code*, *uncurry2* (*RETURN ooo arena-update-pos*))
∈ [*isa-update-pos-pre*]$_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *arena-assn*^{d} → *arena-assn*›
using *isa-update-pos-code.refine*[*FCOMP isa-update-pos*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp isa-update-pos-pre-def*)

sepref-definition *mark-garbage-code*
is ‹*uncurry mark-garbage*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ *arl-assn uint32-assn*›
unfolding *mark-garbage-def fast-minus-def*[*symmetric*]

by *sepref*

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*›
using *mark-garbage-code.refine*[*FCOMP isa-mark-garbage*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-arena-act-code*
is ‹*uncurry isa-arena-act*›
:: ‹(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
unfolding *isa-arena-act-def ACTIVITY-SHIFT-def fast-minus-def*[*symmetric*]
by *sepref*

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*›
using *isa-arena-act-code.refine*[*FCOMP isa-arena-act-arena-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

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*)›
unfolding *isa-arena-incr-act-def ACTIVITY-SHIFT-def fast-minus-def*
by *sepref*

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*›
using *isa-arena-incr-act-code.refine*[*FCOMP isa-arena-incr-act-arena-incr-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

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*)›
unfolding *isa-arena-decr-act-def ACTIVITY-SHIFT-def fast-minus-def*
by *sepref*

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*›
using *isa-arena-decr-act-code.refine*[*FCOMP isa-arena-decr-act-arena-decr-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-arena-decr-act-fast-code*
is ‹*uncurry isa-arena-decr-act*›
:: ‹(*arl64-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl64-assn uint32-assn*)›
unfolding *isa-arena-decr-act-def*
three-uint32-def[*symmetric*] *ACTIVITY-SHIFT-hnr*[*sepref-fr-rules*]
by *sepref*

lemma *isa-arena-decr-act-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-arena-decr-act-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-decr-act*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *arena-fast-assn*›
using *isa-arena-decr-act-fast-code.refine*[*FCOMP isa-arena-decr-act-arena-decr-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

sepref-definition *isa-mark-used-code*
is ‹*uncurry isa-mark-used*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
unfolding *isa-mark-used-def ACTIVITY-SHIFT-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-mark-used-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-used-code*, *uncurry* (*RETURN* ∘∘ *mark-used*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *nat-assn*^{k} $\rightarrow$ *arena-assn*›
using *isa-mark-used-code.refine*[*FCOMP isa-mark-used-mark-used*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-mark-used-fast-code*
is ‹*uncurry isa-mark-used*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
supply *four-uint32-hnr*[*sepref-fr-rules*] *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-mark-used-def four-uint32-def*[*symmetric*]
by *sepref*

lemma *isa-mark-used-fast-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-used-fast-code*, *uncurry* (*RETURN* ∘∘ *mark-used*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *uint64-nat-assn*^{k} $\rightarrow$ *arena-assn*›
using *isa-mark-used-fast-code.refine*[*FCOMP isa-mark-used-mark-used*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-mark-unused-code*
is ‹*uncurry isa-mark-unused*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
unfolding *isa-mark-unused-def ACTIVITY-SHIFT-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-mark-unused-code*[*sepref-fr-rules*]:
‹(*uncurry isa-mark-unused-code*, *uncurry* (*RETURN* ∘∘ *mark-unused*))
$\in$ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *nat-assn*^{k} $\rightarrow$ *arena-assn*›
using *isa-mark-unused-code.refine*[*FCOMP isa-mark-unused-mark-unused*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-mark-unused-fast-code*
is ‹*uncurry isa-mark-unused*›
:: ‹(*arl-assn uint32-assn*)^d $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ (*arl-assn uint32-assn*)›
supply *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-mark-unused-def ACTIVITY-SHIFT-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-mark-unused-fast-code*[*sepref-fr-rules*]:
⟨(*uncurry isa-mark-unused-fast-code*, *uncurry* (*RETURN* ∘∘ *mark-unused*))
∈ [*uncurry arena-act-pre*]$_a$ *arena-assn*^{d} $*_a$ *uint64-nat-assn*^{k} → *arena-assn*⟩
using *isa-mark-unused-fast-code.refine*[*FCOMP isa-mark-unused-mark-unused*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

sepref-definition *isa-marked-as-used-code*
is ⟨*uncurry isa-marked-as-used*⟩
:: ⟨(*arl-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
supply *op-eq-uint32*[*sepref-fr-rules*]
unfolding *isa-marked-as-used-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-marked-as-used-code*[*sepref-fr-rules*]:
⟨(*uncurry isa-marked-as-used-code*, *uncurry* (*RETURN* ∘∘ *marked-as-used*))
∈ [*uncurry marked-as-used-pre*]$_a$ *arena-assn*^{k} $*_a$ *nat-assn*^{k} → *bool-assn*⟩
using *isa-marked-as-used-code.refine*[*FCOMP isa-marked-as-used-marked-as-used*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl-assn-comp update-lbd-pre-def*)

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*)⟩
supply *ACTIVITY-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-arena-incr-act-def*
by *sepref*

lemma *isa-arena-incr-act-fast-code*[*sepref-fr-rules*]:
⟨(*uncurry isa-arena-incr-act-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-incr-act*))
∈ [*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *uint64-nat-assn*^{k} → *arena-fast-assn*⟩
using *isa-arena-incr-act-fast-code.refine*[*FCOMP isa-arena-incr-act-arena-incr-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

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*⟩
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
three-uint32-hnr[*sepref-fr-rules*] *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-arena-status-def three-uint32-def*[*symmetric*]
by *sepref*

lemma *isa-arena-status-fast-hnr*[*sepref-fr-rules*]:
⟨(*uncurry arena-status-fast-code*, *uncurry* (*RETURN* ∘∘ *arena-status*))
∈ [*uncurry arena-is-valid-clause-vdom*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} → *status-assn*⟩
using *arena-status-fast-code.refine*[*FCOMP isa-arena-status-arena-status*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp status-assn-alt-def*
by (*simp add*: *arl64-assn-comp*)

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*)) ∈ [λ(*l*,*i*). *i*<*length l*]$_a$ (*is-array-list64*^{k} $*_a$ *nat-assn*^{k}) → *id-assn*
by *sepref-to-hoare* (*sep-auto simp*: *arl64-get2-def is-array-list64-def*)

sepref-decl-impl *arl64-get2*: *arl64-get2-hnr-aux* **.**
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*⟩
supply *arena-el-assn-alt-def* [*symmetric*, *simp*] *sum-uint64-assn* [*sepref-fr-rules*]
three-uint32-hnr [*sepref-fr-rules*]
unfolding *isa-arena-status-def STATUS-SHIFT-def fast-minus-def*
by *sepref*

lemma *isa-arena-status-fast-hnr2* [*sepref-fr-rules*]:
⟨(*uncurry arena-status-fast-code2*, *uncurry* (*RETURN* ∘∘ *arena-status*))
∈ [*uncurry arena-is-valid-clause-vdom*]$_a$
arena-fast-assn^k $*_a$ *nat-assn*^{k} → *status-assn*⟩
using *arena-status-fast-code2.refine* [*FCOMP isa-arena-status-arena-status* [*unfolded convert-fref*]]
unfolding *hr-comp-assoc* [*symmetric*] *uncurry-def list-rel-compp status-assn-alt-def*
by (*simp add*: *arl64-assn-comp*)

sepref-definition *isa-update-pos-fast-code*
is ⟨*uncurry2 isa-update-pos*⟩
:: ⟨*uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ (*arl64-assn uint32-assn*)^d $\rightarrow_a$ *arl64-assn uint32-assn*⟩
supply *minus-uint32-assn* [*sepref-fr-rules*] *POS-SHIFT-uint64-hnr* [*sepref-fr-rules*] *minus-uint64-assn* [*sepref-fr-rules*]
unfolding *isa-update-pos-def uint32-nat-assn-minus* [*sepref-fr-rules*] *two-uint64-nat-def* [*symmetric*]
by *sepref*

lemma *isa-update-pos-code-fast-hnr* [*sepref-fr-rules*]:
⟨(*uncurry2 isa-update-pos-fast-code*, *uncurry2* (*RETURN ooo arena-update-pos*))
∈ [*isa-update-pos-pre*]$_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*⟩
using *isa-update-pos-fast-code.refine* [*FCOMP isa-update-pos* [*unfolded convert-fref*]]
unfolding *hr-comp-assoc* [*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp isa-update-pos-pre-def*)

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*)))
∈ [λ((*N*, *i*), *j*). *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*)]$_f$
list-fmap-rel vdom $\times_f$ *nat-rel* $\times_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *list-fmap-rel-def arena-lifting*)

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)]);
$(i, N) \leftarrow WHILE_T$ *λ(i, N). i < length C ⟶ length N < header-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)))))›
unfolding *header-size-def*
by *auto*

lemma *nth-append-clause*:
‹*a < length C ⟹ append-clause b C N ! (length N + header-size C + a) = ALit (C ! a)*›
unfolding *append-clause-def header-size-Suc-def append-clause-skeleton-def*
by (*auto simp*: *nth-Cons nth-append*)

lemma *fm-add-new-append-clause*:
‹*fm-add-new b C N ≤ RETURN (append-clause b C N, length N + header-size C)*›
unfolding *fm-add-new-def*
apply (*rewrite at* ‹*let - = length - in -*› *Let-def*)
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure (λ(i, -). Suc (length C) − i)*› **and**
I′ = ‹*λ(i, N′). N′ = take (length N + header-size C + i) (append-clause b C N) ∧*
i ≤ length C›])
subgoal by *auto*
subgoal by (*auto simp*: *append-clause-def header-size-def*
append-clause-skeleton-def split: *if-splits*)
subgoal by (*auto simp*: *append-clause-def header-size-def*
append-clause-skeleton-def split: *if-splits*)
subgoal by *simp*
subgoal by *simp*
subgoal by *auto*
subgoal by (*auto simp*: *take-Suc-conv-app-nth nth-append-clause*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

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*›
by (*auto simp*: *uint64-max-def uint32-max-def header-size-def*)

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-uint64-nat*, *N*);

RETURN (N, l + header-size C)
}⟩
unfolding *fm-add-new-def Let-def AStatus-LEARNED2-def AStatus-IRRED2-def AStatus-LEARNED-def AStatus-IRRED-def*
by *auto*

definition *fmap-swap-ll-u64* **where**
[*simp*]: ⟨*fmap-swap-ll-u64 = fmap-swap-ll*⟩

lemma *slice-Suc-nth*:
⟨*a < b ⟹ a < length xs ⟹ Suc a < b ⟹ Misc.slice a b xs = xs ! a # Misc.slice (Suc a) b xs*⟩
by (*metis Cons-nth-drop-Suc Misc.slice-def Suc-diff-Suc take-Suc-Cons*)

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 ≥* (*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*) *≤ 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 ⟶ 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 ∈# dom-m N*⟩
shows ⟨*valid-arena* (*new-arena* @ *clause-slice old-arena N C*)
(*fmupd* (*length new-arena + header-size* (*N ∝ C*)) (*N ∝ C*, *irred N C*) *N′*)
(*insert* (*length new-arena + header-size* (*N ∝ C*)) *vd′*)⟩
proof −
define *pos st lbd act used* **where**
⟨*pos =* (*if is-long-clause* (*N ∝ C*) *then arena-pos old-arena C − 2 else 0*)⟩ **and**
⟨*st = arena-status old-arena C*⟩ **and**
⟨*lbd = arena-lbd old-arena C*⟩ **and**
⟨*act = arena-act old-arena C*⟩ **and**
⟨*used = arena-used old-arena C*⟩
have ⟨*2 ≤ length* (*N ∝ C*)⟩
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def
arena-lbd-def xarena-lbd-def
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def

xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def pos-def arena-pos-def
xarena-pos-def
arena-lbd-def xarena-lbd-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def is-LBD-def*
is-Act-def)
have
45: ‹*4* = (*Suc* (*Suc* (*Suc* (*Suc* *0*))))›
‹*5* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))›
by *auto*
have *sl*: ‹*clause-slice old-arena N C* =
(*if is-long-clause* ($N \propto C$) *then* [*APos pos*]
else []) @
[*AStatus st used*, *AActivity act*, *ALBD lbd*, *ASize* (*length* ($N \propto C$) − *2*)] @
map ALit ($N \propto C$) ›
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def pos-def arena-pos-def
xarena-pos-def
arena-lbd-def xarena-lbd-def
arena-length-def xarena-length-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def is-LBD-def*
is-Act-def header-size-def 45
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))›]
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* (*Suc* (*Suc* *0*)))›]
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* (*Suc* *0*))›]
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* *0*)›]
slice-Suc-nth[*of* ‹*C* − *Suc* *0*›]
SHIFTS-alt-def arena-length-def
arena-pos-def xarena-pos-def
arena-status-def xarena-status-def)

have ‹*2* ≤ *length* ($N \propto C$)› **and**
‹*pos* ≤ *length* ($N \propto C$) − *2*› **and**
‹*st* = *IRRED* ⟷ *irred N C*› **and**
‹*st* ≠ *DELETED*›
unfolding *st-def used-def act-def lbd-def pos-def*
append-clause-skeleton-def st-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*cases* ‹*is-short-clause* ($N \propto C$)›;
auto split: *arena-el.splits if-splits*
simp: *header-size-def arena-pos-def*; *fail*)+

then have ‹*valid-arena* (*append-clause-skeleton pos st used act lbd* ($N \propto C$) *new-arena*)
(*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′*)›
apply −
by (*rule valid-arena-append-clause-skeleton*[*OF assms*(*2*), *of* ‹$N \propto C$› - *st*
pos used act lbd]) *auto*
moreover have
‹*append-clause-skeleton pos st used act lbd* ($N \propto C$) *new-arena* =

```
      new-arena @ clause-slice old-arena N C›
    by (auto simp: append-clause-skeleton-def sl)
  ultimately show ?thesis
    by auto
qed

lemma fm-mv-clause-to-new-arena:
  assumes ‹valid-arena old-arena N vd› and
    ‹valid-arena new-arena N' vd'› and
    ‹C ∈# dom-m N›
  shows ‹fm-mv-clause-to-new-arena C old-arena new-arena ≤
    SPEC(λnew-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 ∝ C)) (N ∝ C, irred N C) N')
        (insert (length new-arena + header-size (N ∝ C)) vd'))›
proof −
  define st and en where
    ‹st = C − (if arena-length old-arena C ≤ 4 then 4 else 5)› and
    ‹en = C + arena-length old-arena C›
  have st:
    ‹st = C − header-size (N ∝ C)›
    using assms
    unfolding st-def
    by (auto simp: st-def header-size-def
        arena-lifting)
  show ?thesis
    using assms
    unfolding fm-mv-clause-to-new-arena-def st-def[symmetric]
      en-def[symmetric] Let-def nat-of-uint64-conv-def
    apply (refine-vcg
     WHILEIT-rule-stronger-inv[where R = ‹measure (λ(i, N). en − i)› and
      I' = ‹λ(i, new-arena'). i ≤ C + length (N∝C) ∧ i ≥ st ∧
        new-arena' = new-arena @
    Misc.slice (C − header-size (N∝C)) i old-arena›])
    subgoal
      unfolding arena-is-valid-clause-idx-def
      by auto
    subgoal using arena-lifting(4)[OF assms(1)] by (auto
        dest!: arena-lifting(1)[of - N - C] simp: header-size-def split: if-splits)
    subgoal using arena-lifting(10, 4) en-def by auto
    subgoal
      by auto
    subgoal by auto
    subgoal
      using arena-lifting[OF assms(1,3)]
      by (auto simp: st)
    subgoal
      by (auto simp: st arena-lifting)
    subgoal
      using arena-lifting[OF assms(1,3)]
      by (auto simp: st en-def)
    subgoal
      using arena-lifting[OF assms(1,3)]
      by (auto simp: st en-def)
    subgoal by auto
```

subgoal using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *slice-len-min-If en-def st-def header-size-def*)
subgoal
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *st en-def*)
subgoal
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *st*)
subgoal
by (*auto simp*: *st en-def arena-lifting*[*OF assms*(*1*,*3*)]
slice-append-nth)
subgoal by *auto*
subgoal by (*auto simp*: *en-def arena-lifting*)
subgoal
using *valid-arena-append-clause-slice*[*OF assms*]
by *auto*
done
qed

lemma *size-learned-clss-dom-m*: ‹*size* (*learned-clss-l N*) $\leq$ *size* (*dom-m N*)›
unfolding *ran-m-def*
apply (*rule order-trans*[*OF size-filter-mset-lesseq*])
by (*auto simp*: *ran-m-def*)

lemma *distinct-sum-mset-sum*:
‹*distinct-mset As* $\Longrightarrow$ ($\sum$ *A* $\in$# *As*. (*f* :: ′*a* $\Rightarrow$ *nat*) *A*) = ($\sum$ *A* $\in$ *set-mset As*. *f A*)›
by (*subst sum-mset-sum-count*) (*auto intro*!: *sum.cong simp*: *distinct-mset-def*)

lemma *distinct-sorted-append*: ‹*distinct* (*xs* @ [*x*]) $\Longrightarrow$ *sorted* (*xs* @ [*x*]) $\longleftrightarrow$ *sorted xs* $\wedge$ ($\forall$ *y* $\in$ *set xs*. *y* < *x*)›
using *not-distinct-conv-prefix sorted-append* **by** *fastforce*

lemma (**in** *linordered-ab-semigroup-add*) *Max-add-commute2*:
fixes *k*
assumes *finite S* **and** *S* $\neq$ {}
shows *Max* ((λ*x*. *x* + *k*) ‘ *S*) = *Max S* + *k*
proof −
have *m*: $\bigwedge$*x y*. *max x y* + *k* = *max* (*x*+*k*) (*y*+*k*)
by(*simp add*: *max-def antisym add-right-mono*)
have (λ*x*. *x* + *k*) ‘ *S* = (λ*y*. *y* + *k*) ‘ (*S*) **by** *auto*
have *Max* ... = *Max* (*S*) + *k*
using *assms hom-Max-commute* [*of* λ*y*. *y*+*k S*, *OF m*, *symmetric*] **by** *simp*
then show *?thesis* **by** *simp*
qed

lemma *valid-arena-ge-length-clauses*:
assumes ‹*valid-arena arena N vdom*›
shows ‹*length arena* $\geq$ ($\sum$ *C* $\in$# *dom-m N*. *length* (*N* $\propto$ *C*) + *header-size* (*N* $\propto$ *C*))›
proof −
obtain *xs* **where**
mset-xs: ‹*mset xs* = *dom-m N*› **and** *sorted*: ‹*sorted xs*› **and** *dist*[*simp*]: ‹*distinct xs*› **and** *set-xs*: ‹*set xs* = *set-mset* (*dom-m N*)›
using *distinct-mset-dom distinct-mset-mset-distinct mset-sorted-list-of-multiset* **by** *fastforce*
then have *1*: ‹*set-mset* (*mset xs*) = *set xs*› **by** (*meson set-mset-mset*)

have *diff*: ‹*xs* $\neq$ [] $\Longrightarrow$ *a* $\in$ *set xs* $\Longrightarrow$ *a* < *last xs* $\Longrightarrow$ *a* + *length* (*N* $\propto$ *a*) $\leq$ *last xs*› **for** *a*

using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a* ‹*last xs*›]
mset-xs[*symmetric*] *sorted* **by** (*cases xs rule*: *rev-cases*; *auto simp*: *sorted-append*)
have ‹*set xs* ⊆ *set-mset* (*dom-m N*)›
using *mset-xs*[*symmetric*] **by** *auto*
then have ‹($\sum$ *A*∈*set xs*. *length* (*N* ∝ *A*) + *header-size* (*N* ∝ *A*)) ≤ *Max* (*insert 0* ((λ*A*. *A* + *length* (*N* ∝ *A*)) ‘ (*set xs*)))›
(**is** ‹*?P xs* ≤ *?Q xs*›)
using *sorted dist*
proof (*induction xs rule*: *rev-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*snoc x xs*)
then have *IH*: ‹($\sum$ *A*∈*set xs*. *length* (*N* ∝ *A*) + *header-size* (*N* ∝ *A*))
≤ *Max* (*insert 0* ((λ*A*. *A* + *length* (*N* ∝ *A*)) ‘ *set xs*))› **and**
x-dom: ‹*x* ∈# *dom-m N*› **and**
x-max: ‹⋀*a*. *a* ∈ *set xs* ⟹ *x* > *a*› **and**
xs-N: ‹*set xs* ⊆ *set-mset* (*dom-m N*)›
by (*auto simp*: *sorted-append order*.*order-iff-strict dest*!: *bspec*)
have *x-ge*: ‹*header-size* (*N* ∝ *x*) ≤ *x*›
using *assms* ‹*x* ∈# *dom-m N*› *arena-lifting*(*1*) **by** *blast*
have *diff*: ‹*a* ∈ *set xs* ⟹ *a* + *length* (*N* ∝ *a*) + *header-size* (*N* ∝ *x*) ≤ *x*›
‹*a* ∈ *set xs* ⟹ *a* + *length* (*N* ∝ *a*) ≤ *x*› **for** *a*
using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a x*]
x-max[*of a*] *xs-N x-dom* **by** *auto*

have ‹*?P* (*xs* @ [*x*]) ≤ *?P xs* + *length* (*N* ∝ *x*) + *header-size* (*N* ∝ *x*)›
using *snoc* **by** *auto*
also have ‹... ≤ *?Q xs* + (*length* (*N* ∝ *x*) + *header-size* (*N* ∝ *x*))›
using *IH* **by** *auto*
also have ‹... ≤ (*length* (*N* ∝ *x*) + *x*)›
by (*subst linordered-ab-semigroup-add-class*.*Max-add-commute2*[*symmetric*]; *auto intro*: *diff x-ge*)
also have ‹... = *Max* (*insert* (*x* + *length* (*N* ∝ *x*)) ((λ*x*. *x* + *length* (*N* ∝ *x*)) ‘ *set xs*))›
by (*subst eq-commute*)
(*auto intro*!: *linorder-class*.*Max-eqI intro*: *order-trans*[*OF diff*(*2*)])
finally show *?case* **by** *auto*
qed
also have ‹... ≤ (*if xs* = [] *then 0 else last xs* + *length* (*N* ∝ *last xs*))›
using *sorted distinct-sorted-append*[*of* ‹*butlast xs*› ‹*last xs*›] *dist*
by (*cases* ‹*xs*› *rule*: *rev-cases*)
(*auto intro*: *order-trans*[*OF diff*])
also have ‹... ≤ *length arena*›
using *arena-lifting*(*7*)[*OF assms*, *of* ‹*last xs*› ‹*length* (*N* ∝ *last xs*) − *1*›] *mset-xs*[*symmetric*] *assms*
by (*cases* ‹*xs*› *rule*: *rev-cases*) (*auto simp*: *arena-lifting*)
finally show *?thesis*
unfolding *mset-xs*[*symmetric*]
by (*subst distinct-sum-mset-sum*) *auto*
qed

lemma *valid-arena-size-dom-m-le-arena*: ‹*valid-arena arena N vdom* ⟹ *size* (*dom-m N*) ≤ *length arena*›
using *valid-arena-ge-length-clauses*[*of arena N vdom*]
ordered-comm-monoid-add-class.*sum-mset-mono*[*of* ‹*dom-m N*› ‹λ-. *1*›
‹λ*C*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*)›]
by (*fastforce simp*: *header-size-def split*: *if-splits*)

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

abbreviation *isasat-clauses-assn* **where**
  ‹*isasat-clauses-assn* ≡ *arlO-assn clause-ll-assn* ∗*a* *arl-assn* (*clause-status-assn* ∗*a* *uint32-nat-assn* ∗*a* *uint32-nat-assn*)›

lemma *AStatus-IRRED* [*sepref-fr-rules*]:
  ‹(*uncurry0* (*return 0*), *uncurry0* (*RETURN AStatus-IRRED*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-IRRED-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
      status-rel-def bitfield-rel-def nat-0-AND)

lemma *AStatus-IRRED2* [*sepref-fr-rules*]:
  ‹(*uncurry0* (*return 0b100*), *uncurry0* (*RETURN AStatus-IRRED2*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-IRRED2-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
      status-rel-def bitfield-rel-def nat-0-AND)

lemma *AStatus-LEARNED* [*sepref-fr-rules*]:
  ‹(*uncurry0* (*return 0b101*), *uncurry0* (*RETURN AStatus-LEARNED*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-LEARNED-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
      status-rel-def bitfield-rel-def)

lemma *AStatus-LEARNED2* [*sepref-fr-rules*]:
  ‹(*uncurry0* (*return 0b001*), *uncurry0* (*RETURN AStatus-LEARNED2*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-LEARNED2-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
    status-rel-def bitfield-rel-def)

lemma *AActivity-hnr*[*sepref-fr-rules*]:
  ‹(*return o id*, *RETURN o AActivity*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-LEARNED-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
    status-rel-def)

lemma *ALBD-hnr*[*sepref-fr-rules*]:
  ‹(*return o id*, *RETURN o ALBD*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-LEARNED-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
    status-rel-def)

lemma *ASize-hnr*[*sepref-fr-rules*]:
  ‹(*return o id*, *RETURN o ASize*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *AStatus-LEARNED-def arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
    status-rel-def)

lemma *APos-hnr*[*sepref-fr-rules*]:
  ‹(*return o id*, *RETURN o APos*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *arena-el-assn*›
  by *sepref-to-hoare*
    (*sep-auto simp*: *arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def*
    status-rel-def)

lemma *ALit-hnr*[*sepref-fr-rules*]:
‹(*return o id*, *RETURN o ALit*) ∈ *unat-lit-assn*^{k} $\rightarrow_a$ *arena-el-assn*›
apply *sepref-to-hoare*
by *sep-auto*
(*sep-auto simp*: *arena-el-rel-def hr-comp-def uint32-nat-rel-def br-def unat-lit-rel-def*)
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*›
by (*sepref-to-hoare*; *sep-auto simp*: *uint64-nat-rel-def br-def*)+

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*))) ∈
[λ(((*l*, -),*i*),*j*). *i* < *length l* ∧ *j* < *length-rll l i*]$_a$
(*arlO-assn* (*array-assn R*) ∗*a GG*)^k $*_a$ *nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *R*›
unfolding *nth-raa-i-u64-def*
supply *nth-aa-hnr*[*to-hnr*, *sep-heap-rules*]
using *assms*
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def*)

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*))) ∈
[λ(((*l*, -),*i*),*j*). *i* < *length l* ∧ *j* < *length-rll l i*]$_a$
(*arlO-assn* (*array-assn R*) ∗*a GG*)^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} → *R*›
using *assms*
by *sepref-to-hoare sep-auto*

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

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

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* ∧ *length* (*xs* ! *i*) ≤ *uint-max*]$_a$
isasat-clauses-assn^k $*_a$ *nat-assn*^{k} → *uint32-nat-assn*›
by *sepref*

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* ∧ *length* (*xs* ! *i*) ≤ *uint-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint64-nat-assn*^{k} → *uint32-nat-assn*›
by *sepref*

sepref-definition *length-raa-u64-clss*
is ‹*uncurry* ((*RETURN* ∘∘∘ *case-prod*) (λ*N* -. *length-rll-n-uint64 N*))›
:: ‹[λ((*xs*, -), *i*). *i* < *length xs* ∧ *length* (*xs* ! *i*) ≤ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *nat-assn*^{k} → *uint64-nat-assn*›
by *sepref*

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* ∧ *length* (*xs* ! *i*) ≤ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint32-nat-assn*^{k} → *uint64-nat-assn*›
by *sepref*

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* ∧ *length* (*xs* ! *i*) ≤ *uint64-max*]$_a$
isasat-clauses-assn^k $*_a$ *uint64-nat-assn*^{k} → *uint64-nat-assn*›
by *sepref*

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} → *nat-assn*›
by *sepref*

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} → *nat-assn*›
by *sepref*

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* ∧ *i* < *length-rll xs k* ∧ *j* < *length-rll xs k*]$_a$
isasat-clauses-assn^d $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} → *isasat-clauses-assn*›
by *sepref*

sepref-definition *is-short-clause-code*
is ‹*RETURN o is-short-clause*›
:: ‹*clause-ll-assn*^{k} $\to_a$ *bool-assn*›
unfolding *is-short-clause-def MAX-LENGTH-SHORT-CLAUSE-def*
by *sepref*
declare *is-short-clause-code.refine*[*sepref-fr-rules*]

sepref-definition *header-size-code*
is ⟨*RETURN o header-size*⟩
:: ⟨*clause-ll-assn*^{k} $\rightarrow_a$ *nat-assn*⟩
unfolding *header-size-def*
by *sepref*

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-max*+*2* $\wedge$ *length* $C \geq 2$]$_a$ *bool-assn*^{k} $*_a$ *clause-ll-assn*^{d} $*_a$ (*arena-assn*)^d $\rightarrow$
arena-assn $*a$ *nat-assn*⟩
supply [[*goals-limit*=*1*]] *le-uint32-max-le-uint64-max*[*intro*]
unfolding *fm-add-new-def AStatus-IRRED-def*[*symmetric*] *AStatus-IRRED2-def*[*symmetric*]
AStatus-LEARNED-def[*symmetric*] *AStatus-LEARNED2-def*[*symmetric*]
two-uint64-nat-def[*symmetric*]
apply (*rewrite* **in** ⟨*let* - = - − - *in* -⟩ *length-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨*let* - = - *in let* - = - *in let* - = ¤ *in* -⟩ *uint32-of-uint64-conv-def*[*symmetric*])
apply (*rewrite at* ⟨*WHILEIT* - (λ(-, -).- < ¤)⟩ *length-uint64-nat-def*[*symmetric*])
by *sepref*

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

sepref-definition (**in** −) *header-size-fast-code*
is ⟨*RETURN o header-size*⟩
:: ⟨*clause-ll-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*⟩
supply *uint64-max-def*[*simp*]
unfolding *header-size-def five-uint64-nat-def*[*symmetric*] *four-uint64-nat-def*[*symmetric*]

by *sepref*

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-and-length-fast-code-pre*]$_a$
bool-assn^k $*_a$ *clause-ll-assn*^{d} $*_a$ (*arena-fast-assn*)^d $\rightarrow$
arena-fast-assn $*a$ *uint64-nat-assn*⟩
supply [[*goals-limit*=*1*]] *le-uint32-max-le-uint64-max*[*intro*] *append-and-length-code-fast*[*intro*]
header-size-def[*simp*] *if-splits*[*split*] *header-size-fast-code.refine*[*sepref-fr-rules*]
unfolding *fm-add-new-def AStatus-IRRED-def*[*symmetric*] *append-and-length-fast-code-pre-def*
AStatus-LEARNED-def[*symmetric*] *AStatus-LEARNED2-def*[*symmetric*]
AStatus-IRRED2-def[*symmetric*] *four-uint64-nat-def*[*symmetric*]
two-uint64-nat-def[*symmetric*] *fm-add-new-fast-def*
apply (*rewrite* **in** ⟨*let* - = - − - *in* -⟩ *length-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨*let* - = - *in let* - = - *in let* - = ¤ *in* -⟩ *uint32-of-uint64-conv-def*[*symmetric*])
apply (*rewrite at* ⟨*WHILEIT* - (λ(-, -).- < ¤)⟩ *length-uint64-nat-def*[*symmetric*])
by *sepref*

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

sepref-definition *fmap-swap-ll-u64-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* ∧ *i* < *length-rll xs k* ∧ *j* < *length-rll xs k*]$_a$
    (*isasat-clauses-assn*^{d} $*_a$ *nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k}) →
      isasat-clauses-assn›
  by *sepref*

sepref-definition *fmap-rll-u-clss*
  is ‹*uncurry2* (*RETURN ooo* (λ(*N*, -) *i*. *nth-rll N i*))›
  :: ‹[λ(((*l*, -), *i*), *j*). *i* < *length l* ∧ *j* < *length-rll l i*]$_a$
    isasat-clauses-assn^k $*_a$ *nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} →
     unat-lit-assn›
  by *sepref*

sepref-definition *fmap-rll-u32-clss*
  is ‹*uncurry2* (*RETURN ooo* (λ(*N*, -) *i*. *nth-rll N i*))›
  :: ‹[λ(((*l*, -), *i*), *j*). *i* < *length l* ∧ *j* < *length-rll l i*]$_a$
    isasat-clauses-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} →
     unat-lit-assn›
  supply *length-rll-def*[*simp*]
  by *sepref*

sepref-definition *swap-lits-code*
  is ‹*uncurry3 isa-arena-swap*›
  :: ‹*nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ (*arl-assn uint32-assn*)^d $\rightarrow_a$ *arl-assn uint32-assn*›
  unfolding *isa-arena-swap-def WB-More-Refinement-List.swap-def IICF-List.swap-def*[*symmetric*]
  by *sepref*

lemma *swap-lits-refine*[*sepref-fr-rules*]:
  ‹(*uncurry3 swap-lits-code*, *uncurry3* (*RETURN oooo swap-lits*))
  ∈ [*uncurry3 swap-lits-pre*]$_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *arena-assn*^{d} → *arena-assn*›
  using *swap-lits-code.refine*[*FCOMP isa-arena-swap*[*unfolded convert-fref*]]
  unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
  by (*auto simp add*: *arl-assn-comp*)

sepref-definition (**in** −) *swap-lits-fast-code*
  is ‹*uncurry3 isa-arena-swap*›
  :: ‹[λ(((-, -), -), *N*). *length N* ≤ *uint64-max*]$_a$
    uint64-nat-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ (*arl64-assn uint32-assn*)^d →
      arl64-assn uint32-assn›
  unfolding *isa-arena-swap-def WB-More-Refinement-List.swap-def IICF-List.swap-def*[*symmetric*]
  by *sepref*

lemma *swap-lits-fast-refine*[*sepref-fr-rules*]:
  ‹(*uncurry3 swap-lits-fast-code*, *uncurry3* (*RETURN oooo swap-lits*))
  ∈ [λ(((*C*, *i*), *j*), *arena*). *swap-lits-pre C i j arena* ∧ *length arena* ≤ *uint64-max*]$_a$
    uint64-nat-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
  using *swap-lits-fast-code.refine*[*FCOMP isa-arena-swap*[*unfolded convert-fref*]]
  unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
  by (*auto simp add*: *arl64-assn-comp*)

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*›

:: ‹$nat\text{-}assn^k *_a arena\text{-}assn^k *_a arena\text{-}assn^d \rightarrow_a arena\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *fm-mv-clause-to-new-arena-def*
by *sepref*

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*›
:: ‹$[\lambda((n, arena_o), arena).\ length\ arena_o \leq uint64\text{-}max \wedge length\ arena + arena\text{-}length\ arena_o\ n + (if\ arena\text{-}length\ arena_o\ \ n \leq 4\ then\ 4\ else\ 5) \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
supply [[*goals-limit=1*]] *if-splits*[*split*]
unfolding *fm-mv-clause-to-new-arena-def four-uint64-nat-def*[*symmetric*] *five-uint64-nat-def*[*symmetric*] *one-uint64-nat-def*[*symmetric*] *nat-of-uint64-conv-def*
by *sepref*

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*›
by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-le-length-M*:
  ‹*control-stack cs M* $\Longrightarrow$ *c*∈*set cs* $\Longrightarrow$ *c* < *length M*›
  by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-propa*[*simp*]:
  ‹*control-stack cs* (*Propagated x21 x22* # *list*) $\longleftrightarrow$ *control-stack cs list*›
  by (*auto simp*: *control-stack.intros elim*: *control-stackE*)

lemma *control-stack-filter-map-nth*:
  ‹*control-stack cs M* $\Longrightarrow$ *filter is-decided* (*rev M*) = *map* (*nth* (*rev M*)) *cs*›
  apply (*induction rule*: *control-stack.induct*)
  subgoal by *auto*
  subgoal for *cs M L C*
    using *control-stack-le-length-M*[*of cs M*]
    by (*auto simp*: *nth-append*)
  subgoal for *cs M L*
    using *control-stack-le-length-M*[*of cs M*]
    by (*auto simp*: *nth-append*)
  done

lemma *control-stack-empty-cs*[*simp*]: ‹*control-stack* [] *M* $\longleftrightarrow$ *count-decided M* = *0*›
  by (*induction M rule*:*ann-lit-list-induct*)
    (*auto simp*: *control-stack.empty control-stack.cons-prop elim*: *control-stackE*)

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* $\longleftrightarrow$
    (*length cs* = *count-decided M* ∧
      (∀ *L*∈*set M*. *is-decided L* $\longrightarrow$ (*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* $\Longrightarrow$
    no-dup M $\Longrightarrow$ *L*∈*set M* $\Longrightarrow$ *is-decided L* $\Longrightarrow$ *rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
  apply (*induction arbitrary*: *L rule*: *control-stack.induct*)
  subgoal by *auto*
  subgoal for *cs M L C La*
    using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
      count-decided-ge-get-level[*of M* ‹*lit-of La*›]
    apply (*auto simp*: *get-level-cons-if nth-append atm-of-eq-atm-of undefined-notin*)
    by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
        le-SucI le-refl neq0-conv nth-mem)
  subgoal for *cs M L*
    using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
    apply (*auto simp*: *nth-append get-level-cons-if atm-of-eq-atm-of undefined-notin*)
    by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
        le-SucI le-refl neq0-conv)+
  done

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

length cs = *count-decided M* ⟹
control-stack cs M›

proof (*induction M arbitrary*: *cs rule*:*ann-lit-list-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Decided L M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* = *this*(*4*)
from *length* **obtain** *cs′ n* **where** *cs*[*simp*]: ‹*cs* = *cs′* @ [*n*]›
using *length* **by** (*cases cs rule*: *rev-cases*) *auto*
have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*›
by (*auto simp*: *count-decided-0-iff*)
have *dec′*: ‹*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′*› **for** *L′*
using *dec*[*of L′*] *n-d length*
count-decided-ge-get-level[*of M* ‹*lit-of L′*›]
apply (*auto simp*: *get-level-cons-if atm-of-eq-atm-of undefined-notin*
split: *if-splits*)
apply (*auto simp*: *nth-append split*: *if-splits*)
done
have *le*: ‹*length cs′* = *count-decided M*›
using *length* **by** *auto*
have [*simp*]: ‹*n* = *length M*›
using *n-d dec*[*of* ‹*Decided L*›] *le undefined-notin*[*of M* ‹*rev M* ! *n*›] *nth-mem*[*of n* ‹*rev M*›]
by (*auto simp*: *nth-append split*: *if-splits*)
show *?case*
unfolding *cs*
apply (*rule control-stack.cons-dec*)
subgoal
apply (*rule IH*)
using *n-d dec′ le* **by** *auto*
subgoal by *auto*
done
next
case (*Propagated L m M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* = *this*(*4*)
have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*› **for** *n*
by (*auto simp*: *count-decided-0-iff*)
have *dec′*: ‹*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′*› **for** *L′*
using *dec*[*of L′*] *n-d length*
count-decided-ge-get-level[*of M* ‹*lit-of L′*›]
apply (*cases L′*)
apply (*auto simp*: *get-level-cons-if atm-of-eq-atm-of undefined-notin*
split: *if-splits*)
apply (*auto simp*: *nth-append split*: *if-splits*)
done
show *?case*
apply (*rule control-stack.cons-prop*)
apply (*rule IH*)
subgoal using *n-d* **by** *auto*
subgoal using *dec′* **by** *auto*
subgoal using *length* **by** *auto*
done
qed

lemma *control-stack-alt-def*: ‹*no-dup M* ⟹ *control-stack′ cs M* ⟷ *control-stack cs M*›

using *control-stack-alt-def-imp*[*of M cs*] *control-stack-rev-get-lev*[*of cs M*]
control-stack-length-count-dec[*of cs M*] *control-stack-le-length-M*[*of cs M*]
unfolding *control-stack'-def* **apply** −
apply (*rule iffI*)
subgoal by *blast*
subgoal
using *count-decided-ge-get-level*[*of M*]
by (*metis One-nat-def Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff less-imp-diff-less neq0-conv nth-mem*)
done

lemma *control-stack-decomp*:
assumes
decomp: ⟨(*Decided L* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)⟩ **and**
cs: ⟨*control-stack cs M*⟩ **and**
n-d: ⟨*no-dup M*⟩
shows ⟨*control-stack* (*take* (*count-decided M1*) *cs*) *M1*⟩
proof −
obtain *M3* **where** *M*: ⟨*M* = *M3* @ *M2* @ *Decided L* # *M1*⟩
using *decomp* **by** *auto*
define *M2'* **where** ⟨*M2'* = *M3* @ *M2*⟩
have *M*: ⟨*M* = *M2'* @ *Decided L* # *M1*⟩
unfolding *M M2'-def* **by** *auto*
have *n-d1*: ⟨*no-dup M1*⟩
using *n-d no-dup-appendD* **unfolding** *M* **by** *auto*
have ⟨*control-stack' cs M*⟩
using *cs*
apply (*subst* (*asm*) *control-stack-alt-def*[*symmetric*])
apply (*rule n-d*)
apply *assumption*
done
then have
cs-M: ⟨*length cs* = *count-decided M*⟩ **and**
L: ⟨⋀*L*. *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*⟩
unfolding *control-stack'-def* **by** *auto*
have *H*: ⟨*L'* ∈ *set M1* ⟹ *undefined-lit M2'* (*lit-of L'*) ∧ *atm-of* (*lit-of L'*) ≠ *atm-of L*⟩ **for** *L'*
using *n-d* **unfolding** *M*
by (*metis atm-of-eq-atm-of defined-lit-no-dupD*(*1*) *defined-lit-uminus lit-of.simps*(*1*) *no-dup-appendD no-dup-append-cons no-dup-cons undefined-notin*)
have ⟨*distinct M*⟩
using *no-dup-imp-distinct*[*OF n-d*] **.**
then have *K*: ⟨*L'* ∈ *set M1* ⟹ *x* < *length M* ⟹ *rev M* ! *x* = *L'* ⟹ *x* < *length M1*⟩ **for** *x L'*
unfolding *M* **apply** (*auto simp*: *nth-append nth-Cons split*: *if-splits nat.splits*)
by (*metis length-rev less-diff-conv local.H not-less-eq nth-mem set-rev undefined-notin*)
have *I*: ⟨*L* ∈ *set M1* ⟹ *is-decided L* ⟹ *get-level M1* (*lit-of L*) > *0*⟩ **for** *L*
using *n-d* **unfolding** *M* **by** (*auto dest*!: *split-list*)
have *cs'*: ⟨*control-stack'* (*take* (*count-decided M1*) *cs*) *M1*⟩
unfolding *control-stack'-def*
apply (*intro conjI ballI impI*)
subgoal using *cs-M* **unfolding** *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ⟨*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)⟩]
count-decided-ge-get-level[*of* ⟨*M1*⟩ ⟨*lit-of L*⟩] *I*[*of L*]
unfolding *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ⟨*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)⟩]
count-decided-ge-get-level[*of* ⟨*M1*⟩ ⟨*lit-of L*⟩] *I*[*of L*]

unfolding *M* **by** *auto*
done
show *?thesis*
apply (*subst control-stack-alt-def*[*symmetric*])
apply (*rule n-d1*)
apply (*rule cs'*)
done
qed

Encoding of the reasons **definition** *DECISION-REASON* :: *nat* **where**
‹*DECISION-REASON* = *1*›

definition *ann-lits-split-reasons* **where**
‹*ann-lits-split-reasons* $\mathcal{A}$ = {((*M*, *reasons*), *M'*). *M* = *map lit-of* (*rev M'*) ∧
(∀ *L* ∈ *set M'*. *is-proped L* ⟶
reasons ! (*atm-of* (*lit-of L*)) = *mark-of L* ∧ *mark-of L* ≠ *DECISION-REASON*) ∧
(∀ *L* ∈ *set M'*. *is-decided L* ⟶ *reasons* ! (*atm-of* (*lit-of L*)) = *DECISION-REASON*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length reasons*)
}›

definition *trail-pol* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*› **where**
‹*trail-pol* $\mathcal{A}$ =
{((*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*) ∧
k = *count-decided M* ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack cs M ∧
isasat-input-bounded $\mathcal{A}$}›

Definition of the full trail **lemma** *trail-pol-alt-def*:
‹*trail-pol* $\mathcal{A}$ = {((*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*) ∧
k = *count-decided M* ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack 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* ∧
M' = *map lit-of* (*rev M*) ∧
isasat-input-bounded $\mathcal{A}$
}›
proof −
have [*intro!*]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint-max-def count-decided-def*
simp del: *length-filter-le*
dest: *length-trail-uint-max-div2*)
show *?thesis*
unfolding *trail-pol-def*

by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-trail-def uint-max-def ann-lits-split-reasons-def*
dest: *length-trail-uint-max-div2*
simp del: *isasat-input-bounded-def*)
qed

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*›
by (*cases M*)
(*auto simp*: *trail-pol-slow-of-fast-def array-nat-of-uint64-conv-def*)

lemma *trail-pol-fast-of-slow-trail-fast-of-slow*:
‹(*RETURN o trail-pol-fast-of-slow*, *RETURN o trail-fast-of-slow*)
∈ $[\lambda M.\ (\forall C\ L.\ Propagated\ L\ C \in set\ M \longrightarrow C < uint64\text{-}max)]_f$
trail-pol $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def array-nat-of-uint64-conv-def*
trail-fast-of-slow-def array-uint64-of-nat-conv-def)

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def array-nat-of-uint64-conv-def*
trail-fast-of-slow-def array-uint64-of-nat-conv-def trail-slow-of-fast-def
trail-pol-slow-of-fast-def)

lemma *trail-pol-same-length*[*simp*]: ‹(M', *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *length* (*fst* M') = *length M*›
by (*auto simp*: *trail-pol-alt-def*)

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*)›
by (*auto simp*: *counts-maximum-level-def*)

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*)⟩
using *assms*
by (*auto 5 5 simp*: *trail-pol-def unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def get-level-atm-pol-pre-def intro*!: *ext*)

lemma (**in** −) *get-level-get-level-atm*: ⟨*get-level M L* = *get-level-atm M* (*atm-of L*)⟩
unfolding *get-level-atm-def*
by (*cases L*) (*auto simp*: *get-level-Neg-Pos*)

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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩ **and**
⟨(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$⟩
shows ⟨*get-level-pol-pre* (*M'*, *L*)⟩
using *assms*
by (*auto 5 5 simp*: *trail-pol-def unat-lit-rel-def nat-lit-rel-def uint32-nat-rel-def br-def get-level-pol-pre-def intro*!: *ext*)

lemma *get-level-get-level-pol*:
assumes
⟨(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$⟩ **and** ⟨*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩
shows ⟨*get-level M L* = *get-level-pol M' L*⟩
using *assms*
by (*auto simp*: *get-level-pol-def get-level-atm-pol-def trail-pol-def*)

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*) ∈ *trail-pol* $\mathcal{A}$ $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*⟩
by (*intro frefI nres-relI*) (*auto simp*: *trail-pol-def count-decided-pol-def*)

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*)) ∈
  $[\lambda(M, L).\ L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_f$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
by (*intro nres-relI frefI*)
(*auto simp*: *trail-pol-def polarity-def polarity-pol-def*
  dest!: *multi-member-split*)

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›
by (*auto simp*: *trail-pol-def polarity-def polarity-pol-def polarity-pol-pre-def*
  dest!: *multi-member-split*)

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* $M' \leq$ *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›
by (*auto simp*: *isa-length-trail-def trail-pol-alt-def isa-length-trail-pre-def*)

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*›
by (*intro frefI nres-relI*)
  (*auto simp*: *isa-length-trail-def trail-pol-alt-def*
  intro!: *ASSERT-leI*)

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* = (λL 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*›
by (*metis Neg-atm-of-iff Pos-atm-of-iff lits-of-def rev-image-eqI*)

lemma *cons-trail-Propagated-tr*:
‹(*uncurry2* (*RETURN ooo cons-trail-Propagated-tr*), *uncurry2* (*RETURN ooo cons-trail-Propagated*)) ∈
  $[\lambda((L, C), M).$ *undefined-lit* M $L \wedge L \in\# \mathcal{L}_{all}\ \mathcal{A} \wedge C \neq$ *DECISION-REASON*$]_f$
  Id $\times_f$ *nat-rel* $\times_f$ *trail-pol* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ‹*fst* (*fst x*)›)
  (*auto simp add*: *trail-pol-def polarity-def cons-trail-Propagated-def uminus-lit-swap*
  cons-trail-Propagated-tr-def Decided-Propagated-in-iff-in-lits-of-l nth-list-update'
  ann-lits-split-reasons-def *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$
  dest!: *in-list-pos-neg-notD multi-member-split dest*: *pos-lit-in-atms-of neg-lit-in-atms-of*
  simp del: *nat-of-lit.simps*)

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 –
have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
have *incl*: ‹*atm-of* ‘# *lit-of* ‘# *mset* (*Decided L* # *M*) ⊆# *remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$)›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *assms dist-atm-M*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*
atm-of-eq-atm-of)
from *size-mset-mono*[*OF this*] **have** *1*: ‹*count-decided M* + *1* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
using *length-filter-le*[*of is-decided M*] **unfolding** *uint-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)))›
by (*auto simp*: *inj-on-def*)
have *H*: ‹*xa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of xa* ≤ *uint-max div 2*› **for** *xa*
using *bounded*
by (*cases xa*) (*auto simp*: *uint-max-def*)
have ‹*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$) ⊆# *mset* [*0*..< *1* + (*uint-max div 2*)]›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *H distinct-image-mset-inj*[*OF inj-on*]
by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}$))) ≤ *1* + *uint-max div 2*›
by *auto*

show *?thesis*
using *1 2* **by** (*auto simp*: *uint-max-def*)

from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* + *1* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
unfolding *uint-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
with *2* **have** ‹*length M* ≤ *uint32-max*›
by *auto*
qed

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 –
have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
have *incl*: ‹*atm-of* ‘# *lit-of* ‘# *mset M* ⊆# *remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$)›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *assms dist-atm-M*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*

atm-of-eq-atm-of)

have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)))›
  by (*auto simp*: *inj-on-def*)
have *H*: ‹*xa* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A} \Longrightarrow$ *atm-of xa* $\leq$ *uint-max div 2*› **for** *xa*
  using *bounded*
  by (*cases xa*) (*auto simp*: *uint-max-def*)
have ‹*remdups-mset* (*atm-of* '# $\mathcal{L}_{all}$ $\mathcal{A}$) $\subseteq\#$ *mset* [*0*..< *1* + (*uint-max div 2*)]›
  apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
  using *H distinct-image-mset-inj*[*OF inj-on*]
  by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
    dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* '# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
  by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* '# $\mathcal{L}_{all}$ $\mathcal{A}$)) $\leq$ *1* + *uint-max div 2*›
  by *auto*
from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* $\leq$ *size* (*remdups-mset* (*atm-of* '# $\mathcal{L}_{all}$ $\mathcal{A}$))›
  unfolding *uint-max-def count-decided-def*
  by (*auto simp del*: *length-filter-le*)
with *2* **show** *?thesis*
  by (*auto simp*: *uint32-max-def*)
qed

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* $\wedge$ *nat-of-lit* ($-L$) < *length xs* $\wedge$ *atm-of L* < *length lvls* $\wedge$ *atm-of L* < *length reasons* $\wedge$ *length M* < *uint32-max*)›

lemma *cons-trail-Propagated-tr-pre*:
assumes ‹(M', *M*) $\in$ *trail-pol* $\mathcal{A}$› **and**
  ‹*undefined-lit M L*› **and**
  ‹*L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
  ‹*C* $\neq$ *DECISION-REASON*›
shows ‹*cons-trail-Propagated-tr-pre* ((*L*, *C*), M')›
using *assms*
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
    cons-trail-Propagated-tr-pre-def
  intro!: *ext*)

lemma *cons-trail-Propagated-tr2*:
‹(M', *M*) $\in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A} \Longrightarrow$ *undefined-lit M L* $\Longrightarrow$ *C* $\neq$ *DECISION-REASON* $\Longrightarrow$
(*cons-trail-Propagated-tr L C* M', *Propagated L C* # *M*) $\in$ *trail-pol* $\mathcal{A}$›
using *cons-trail-Propagated-tr*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *L C M L C* M']
by (*auto simp*: *cons-trail-Propagated-def*)

definition *last-trail-pol-pre* **where**
‹*last-trail-pol-pre* = (λ(*M*, *xs*, *lvls*, *reasons*, *k*). *atm-of* (*last M*) < *length reasons* $\wedge$ *M* $\neq$ [])›

definition (**in** $-$) *last-trail-pol* :: ‹*trail-pol* $\Rightarrow$ (*nat literal* $\times$ *nat option*)› **where**
‹*last-trail-pol* = (λ(*M*, *xs*, *lvls*, *reasons*, *k*).
  let r = *reasons* ! (*atm-of* (*last M*)) *in*
  (*last M*, *if r* = *DECISION-REASON then None else Some r*))›

lemma (**in** $-$) *nat-ann-lit-rel-alt-def*: ‹*nat-ann-lit-rel* = (*unat-lit-rel* $\times_r$ $\langle$*nat-rel*$\rangle$ *option-rel*) *O*

```
    {((L, C), L').
      (C = None ⟶ L' = Decided L) ∧
      (C ≠ None ⟶ L' = Propagated L (the C))}›
  apply (rule; rule)
  subgoal for x
    by (cases x; cases ‹fst x›)
      (auto simp: nat-ann-lit-rel-def ann-lit-of-pair-alt-def
        unat-lit-rel-def uint32-nat-rel-def br-def nat-lit-rel-def
        Collect-eq-comp case-prod-beta relcomp.simps
        split: if-splits)
  subgoal for x
    by (cases x; cases ‹fst x›)
      (auto simp: nat-ann-lit-rel-def ann-lit-of-pair-alt-def
        unat-lit-rel-def uint32-nat-rel-def br-def nat-lit-rel-def
        Collect-eq-comp case-prod-beta relcomp.simps
        split: if-splits)
  done

definition tl-trailt-tr :: ‹trail-pol ⇒ trail-pol› where
  ‹tl-trailt-tr = (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, if reasons ! atm-of L = DECISION-REASON then k−one-uint32-nat else k,
      if reasons ! atm-of L = DECISION-REASON then butlast cs else cs))›

definition tl-trailt-tr-pre where
  ‹tl-trailt-tr-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 ∧
        (reason ! atm-of (last M) = DECISION-REASON ⟶ k ≥ 1 ∧ cs ≠ []))›

lemma ann-lits-split-reasons-map-lit-of:
  ‹((M, reasons), M') ∈ ann-lits-split-reasons 𝒜 ⟹ M = map lit-of (rev M')›
  by (auto simp: ann-lits-split-reasons-def)

lemma control-stack-dec-butlast:
  ‹control-stack b (Decided x1 # M's) ⟹ control-stack (butlast b) M's›
  by (cases b rule: rev-cases) (auto dest: control-stackE)

lemma tl-trail-tr:
  ‹((RETURN o tl-trailt-tr), (RETURN o tl)) ∈
    [λM. M ≠ []]_f trail-pol 𝒜 → ⟨trail-pol 𝒜⟩nres-rel›
proof −
  show ?thesis
    apply (intro frefI nres-relI, rename-tac x y, case-tac ‹y›)
    subgoal by fast
    subgoal for M M' L M's
      unfolding trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def
      apply clarify
      apply (intro conjI; clarify?; (intro conjI)?)
      subgoal
        by (auto simp: trail-pol-def polarity-atm-def tl-trailt-tr-def
            ann-lits-split-reasons-def Let-def)
      subgoal by (auto simp: trail-pol-def polarity-atm-def tl-trailt-tr-def)
```

subgoal by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*)
subgoal
by (*cases* ‹*lit-of L*›)
(*auto simp*: *polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l*
uminus-lit-swap Let-def
dest: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*
atm-of-eq-atm-of get-level-cons-if)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
dest: *no-dup-consistentD*)
subgoal
by (*auto simp*: *tl-trailt-tr-def*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
done
done
qed

lemma *tl-trailt-tr-pre*:
assumes ‹$M \neq []$›
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$›
shows ‹*tl-trailt-tr-pre* M'›
proof −
have [*simp*]: ‹$x \neq [] \Longrightarrow$ *is-decided* (*last* x) $\Longrightarrow$ *Suc 0* $\leq$ *count-decided* x› **for** x
by (*cases x rule*: *rev-cases*) *auto*
show *?thesis*
using *assms*
by (*cases M*; *cases* ‹*hd M*›)
(*auto simp*: *trail-pol-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map tl-trailt-tr-pre-def simp del*: *rev-map*
intro!: *ext*)
qed

definition *tl-trail-propedt-tr* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trail-propedt-tr* = (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-trail-propedt-tr-pre* **where**
‹*tl-trail-propedt-tr-pre* =
(λ(M, *xs*, *lvls*, *reason*, *k*, *cs*). $M \neq [] \land$ *nat-of-lit*(*last M*) < *length xs* $\land$
nat-of-lit($-$*last M*) < *length xs* $\land$ *atm-of* (*last M*) < *length lvls* $\land$
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'›
  using *assms*
  unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def*
    tl-trail-propedt-tr-def tl-trail-propedt-tr-pre-def
  apply *clarify*
  apply (*cases M*; *intro conjI*; *clarify?*; (*intro conjI*)*?*)
  subgoal
    by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
 ann-lits-split-reasons-def Let-def)
  subgoal
    by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
       atm-of-eq-atm-of get-level-cons-if Let-def
 dest!: *ann-lits-split-reasons-map-lit-of*)
  subgoal
    by (*cases* ‹*hd M*›)
      (*auto simp*: *tl-trailt-tr-def* *in*-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
 dest: *no-dup-consistentD*)
  subgoal
    by (*cases* ‹*hd M*›)
      (*auto simp*: *tl-trailt-tr-def* *in*-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
 control-stack-dec-butlast
 dest: *no-dup-consistentD*)
  subgoal
    by (*cases* ‹*hd M*›)
      (*auto simp*: *tl-trailt-tr-def* *in*-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
 control-stack-dec-butlast
 dest: *no-dup-consistentD*)
  done

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*) $\in$
         $[\lambda S.\ S \neq []]_f$ *trail-pol* $\mathcal{A}$ $\rightarrow$ $\langle Id\rangle$*nres-rel*›
  by (*auto simp*: *lit-of-hd-trail-def trail-pol-def lit-of-last-trail-pol-def*
     ann-lits-split-reasons-def hd-map rev-map[*symmetric*]
      intro!: *frefI nres-relI*)

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

definition *cons-trail-Decided-tr* :: ‹*nat literal* ⇒ *trail-pol* ⇒ *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* =
$(\lambda(L, (M, xs, lvls, reason, k, cs)).\ \textit{nat-of-lit}\ L < \textit{length}\ xs \wedge \textit{nat-of-lit}\ (-L) < \textit{length}\ xs \wedge$
$\textit{atm-of}\ L < \textit{length}\ lvls \wedge \textit{atm-of}\ L < \textit{length reason}\ \wedge \textit{length}\ cs < \textit{uint32-max} \wedge$
$\textit{Suc}\ k \leq \textit{uint-max} \wedge \textit{length}\ M < \textit{uint32-max})$⟩

lemma *length-cons-trail-Decided*[*simp*]:
⟨*length* (*cons-trail-Decided L M*) = *Suc* (*length M*)⟩
by (*auto simp*: *cons-trail-Decided-def*)

lemma *cons-trail-Decided-tr*:
⟨(*uncurry* (*RETURN oo cons-trail-Decided-tr*), *uncurry* (*RETURN oo cons-trail-Decided*)) ∈
$[\lambda(L, M).\ \textit{undefined-lit}\ M\ L \wedge L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f\ \textit{Id} \times_f \textit{trail-pol}\ \mathcal{A} \rightarrow \langle \textit{trail-pol}\ \mathcal{A}\rangle \textit{nres-rel}$⟩
by (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ⟨*fst x*⟩)
(*auto simp*: *trail-pol-def polarity-def cons-trail-Decided-def uminus-lit-swap*
Decided-Propagated-in-iff-in-lits-of-l
cons-trail-Decided-tr-def nth-list-update′ ann-lits-split-reasons-def
dest!: *in-list-pos-neg-notD multi-member-split*
intro: *control-stack.cons-dec*
simp del: *nat-of-lit.simps*)

lemma *cons-trail-Decided-tr-pre*:
assumes ⟨$(M', M) \in \textit{trail-pol}\ \mathcal{A}$⟩ **and**
⟨$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$⟩ **and** ⟨*undefined-lit M L*⟩
shows ⟨*cons-trail-Decided-tr-pre* (*L*, *M′*)⟩
using *assms*
by (*auto simp*: *trail-pol-alt-def image-image ann-lits-split-reasons-def* $\textit{uminus-}\mathcal{A}_{in}\textit{-iff}$
cons-trail-Decided-tr-pre-def control-stack-length-count-dec
intro!: *ext undefined-lit-count-decided-uint-max length-trail-uint-max*)

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

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

lemma *undefined-atm-code*:
⟨(*uncurry* (*RETURN oo defined-atm-pol*), *uncurry* (*RETURN oo defined-atm*)) ∈
$[\lambda(M, L).\ \textit{Pos}\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f\ \textit{trail-pol}\ \mathcal{A} \times_r \textit{Id} \rightarrow \langle \textit{bool-rel}\rangle\ \textit{nres-rel}$⟩ (**is** *?A*) **and**
defined-atm-pol-pre:
⟨$(M', M) \in \textit{trail-pol}\ \mathcal{A} \Longrightarrow L \in\#\ \ \mathcal{A} \Longrightarrow \textit{defined-atm-pol-pre}\ M'\ L$⟩
proof −
have *H*: ⟨$2{*}L < \textit{length}\ xs$⟩ (**is** ⟨*?length*⟩) **and**
none: ⟨$\textit{defined-atm}\ M\ L \longleftrightarrow xs\ !\ (2{*}L) \neq \textit{None}$⟩ (**is** *?undef*) **and**
le: ⟨$2{*}L \leq \textit{uint-max}$⟩ (**is** *?le*)
if *L-N*: ⟨$\textit{Pos}\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$⟩ **and** *tr*: ⟨$((M', xs, lvls, k), M) \in \textit{trail-pol}\ \mathcal{A}$⟩
for *M xs lvls k M′ L*
proof −
have
⟨*M′* = *map lit-of* (*rev M*)⟩ **and**
⟨$\forall L{\in}\#\mathcal{L}_{all}\ \mathcal{A}.\ \textit{nat-of-lit}\ L < \textit{length}\ xs \wedge xs\ !\ \textit{nat-of-lit}\ L = \textit{polarity}\ M\ L$⟩
using *tr* **unfolding** *trail-pol-def ann-lits-split-reasons-def* **by** *fast+*
then have *L*: ⟨*nat-of-lit* (*Pos L*) < *length xs*⟩ **and**

xsL: ‹*xs* ! (*nat-of-lit* (*Pos L*)) = *polarity M* (*Pos L*)›
using *L-N* by (*auto dest*!: *multi-member-split*)
show *?length*
using *L* by *simp*
show *?undef*
using *xsL* by (*auto simp*: *polarity-def defined-atm-def*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits*)
show ‹*2*∗*L* ≤ *uint-max*›
using *tr L-N* unfolding *trail-pol-def* by *auto*
qed
show *?A*
unfolding *defined-atm-pol-def*
by (*intro frefI nres-relI*) (*auto 5 5 simp*: *none H le intro*!: *ASSERT-leI*)
show ‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{A} \Longrightarrow$ *defined-atm-pol-pre* M' L›
using *H le* by (*auto simp*: *defined-atm-pol-pre-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
qed

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*) ∈
$[\lambda(M, L).\ L \in$ *lits-of-l* $M]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *lits-of-def*
apply *clarify*
apply (*rename-tac a aa ab ac b ba ad bb x*, *case-tac x*)
by (*auto simp*: *get-propagation-reason-def get-propagation-reason-pol-def*
trail-pol-def ann-lits-split-reasons-def lits-of-def assert-bind-spec-conv)

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* ≠ *None* ⟷ *Propagated L* (*the C*) ∈ *set M*) ∧
(*C* = *None* ⟷ *Decided L* ∈ *set M* ∨ *L* ∉ *lits-of-l M*))›

lemma *no-dup-Decided-PropedD*:
‹*no-dup ad* ⟹ *Decided L* ∈ *set ad* ⟹ *Propagated L C* ∈ *set ad* ⟹ *False*›
by (*metis annotated-lit.distinct*(*1*) *in-set-conv-decomp lit-of.simps*(*1*) *lit-of.simps*(*2*)
no-dup-appendD no-dup-cons undefined-notin xy-in-set-cases)

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*) ∈
[λ(*M*, *L*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ $\times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
proof −
have [*dest*]: ‹*no-dup bb* ⟹
Some True = *polarity bb* (*Pos x1*) ⟹ *Pos x1* ∈ *lits-of-l bb* ∧ *Neg x1* ∉ *lits-of-l bb*› **for** *bb x1*
by (*auto simp*: *polarity-def split*: *if-splits dest*: *no-dup-consistentD*)
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *lits-of-def get-the-propagation-reason-def uncurry-def get-the-propagation-reason-pol-def*
apply *clarify*
apply (*refine-vcg*)
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv
dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv
dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal for *a aa ab ac ad b ba ae bb*
apply (*cases* ‹*aa* ! *nat-of-lit ba* ≠ *SET-TRUE*›)
apply (*subgoal-tac* ‹*ba* ∉ *lits-of-l ae*›)
prefer *2*
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec′(*2*)
dest: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
by (*auto simp*: *lits-of-def dest*: *imageI*[*of* - - *lit-of*])

apply (*subgoal-tac* ‹*ba* ∈ *lits-of-l ae*›)
prefer *2*
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec′(*2*)
dest: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
apply (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv lits-of-def
dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
apply (*case-tac x*; *auto*)
apply (*case-tac x*; *auto*)
done
done
done
qed

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›
by (intro frefI nres-relI)
  (auto simp: isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def
  intro!: ASSERT-leI)

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 ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. 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 ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
  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›
using control-stack-length-count-dec[of ‹take (Suc (count-decided M′s)) cs› ‹Decided x1 # M′s›]
by (auto simp: min-def take-Suc-conv-app-nth split: if-splits elim: control-stackE)

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 −
have [simp]: ‹x ≠ [] ⟹ is-decided (last x) ⟹ Suc 0 ≤ count-decided x› **for** x
  by (cases x rule: rev-cases) auto
show ?thesis
  using assms
  unfolding trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def
    tl-trailt-tr-no-CS-def tl-trailt-tr-no-CS-pre-def
  by (cases M; cases ‹hd M›)

(*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map simp del*: *rev-map*
intro!: *ext*)
qed

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* $\mathcal{A} \rightarrow$ ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ‹*y*›)
subgoal by *fast*
subgoal for *M M′ L M′s*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def*
tl-trailt-tr-no-CS-def
apply *clarify*
apply (*intro conjI*; *clarify?*; (*intro conjI*)*?*)
subgoal
by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
ann-lits-split-reasons-def Let-def)
subgoal by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*)
subgoal by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*)
subgoal
by (*cases* ‹*lit-of L*›)
(*auto simp*: *polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l*
uminus-lit-swap Let-def
dest: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*
atm-of-eq-atm-of get-level-cons-if)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast control-stack-take-Suc-count-dec-unstack
dest: *no-dup-consistentD ann-lits-split-reasons-map-lit-of*)
done
done

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def*

control-stack-length-count-dec trail-pol-conv-to-no-CS-def
intro: *ext*)

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*))
∈ [λ(*k*, *M*). *k* = *count-decided M*]$_f$ *nat-rel* $\times_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def*
control-stack-length-count-dec trail-conv-back-def trail-conv-back-imp-def
intro: *ext intro*!: *ASSERT-refine-left*
dest: *control-stack-length-count-dec multi-member-split*)

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*)) ∈
[λ(*M*, *i*). *i* < *length M*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ $\times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def*
trail-pol-no-CS-def
intro!: *ASSERT-leI*)

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 −
have [*intro*!]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint-max-def count-decided-def*
simp del: *length-filter-le*
dest: *length-trail-uint-max-div2*)
show *?thesis*
unfolding *trail-pol-no-CS-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint-max-def ann-lits-split-reasons-def*
dest: *length-trail-uint-max-div2*

simp del: *isasat-input-bounded-def*)
qed

lemma *isa-length-trail-length-u-no-CS*:
‹(*RETURN o isa-length-trail*, *RETURN o length-uint32-nat*) $\in$ *trail-pol-no-CS* $\mathcal{A} \rightarrow_f \langle$*nat-rel*$\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-length-trail-def trail-pol-no-CS-alt-def ann-lits-split-reasons-def*
intro!: *ASSERT-leI*)

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-hd xs* = (*if xs* = [] *then None else Some* (*hd xs*))›

lemma *option-hd-None-iff*[*iff*]: ‹*option-hd zs* = *None* $\longleftrightarrow$ *zs* = []› ‹*None* = *option-hd zs* $\longleftrightarrow$ *zs* = []›
by (*auto simp*: *option-hd-def*)

lemma *option-hd-Some-iff*[*iff*]: ‹*option-hd zs* = *Some y* $\longleftrightarrow$ (*zs* $\neq$ [] $\wedge$ *y* = *hd zs*)›
‹*Some y* = *option-hd zs* $\longleftrightarrow$ (*zs* $\neq$ [] $\wedge$ *y* = *hd zs*)›
by (*auto simp*: *option-hd-def*)

lemma *option-hd-Some-hd*[*simp*]: ‹*zs* $\neq$ [] $\Longrightarrow$ *option-hd zs* = *Some* (*hd zs*)›
by (*auto simp*: *option-hd-def*)

lemma *option-hd-Nil*[*simp*]: ‹*option-hd* [] = *None*›
by (*auto simp*: *option-hd-def*)

definition *option-last* **where**
‹*option-last l* = (*if l* = [] *then None else Some* (*last l*))›

lemma
option-last-None-iff[*iff*]: ‹*option-last l* = *None* $\longleftrightarrow$ *l* = []› ‹*None* = *option-last l* $\longleftrightarrow$ *l* = []› **and**
option-last-Some-iff[*iff*]:
‹*option-last l* = *Some a* $\longleftrightarrow$ *l* $\neq$ [] $\wedge$ *a* = *last l*›
‹*Some a* = *option-last l* $\longleftrightarrow$ *l* $\neq$ [] $\wedge$ *a* = *last l*›
by (*auto simp*: *option-last-def*)

lemma *option-last-Some*[*simp*]: ‹*l* $\neq$ [] $\Longrightarrow$ *option-last l* = *Some* (*last l*)›
by (*auto simp*: *option-last-def*)

lemma *option-last-Nil*[*simp*]: ‹*option-last* [] = *None*›
by (*auto simp*: *option-last-def*)

lemma *option-last-remove1-not-last*:
‹*x* $\neq$ *last xs* $\Longrightarrow$ *option-last xs* = *option-last* (*remove1 x xs*)›
by (*cases xs rule*: *rev-cases*)
(*auto simp*: *option-last-def remove1-Nil-iff remove1-append*)

lemma *option-hd-rev*: ‹*option-hd* (*rev xs*) = *option-last xs*›
by (*cases xs rule*: *rev-cases*) *auto*

lemma *map-option-option-last*:
‹*map-option f* (*option-last xs*) = *option-last* (*map f xs*)›
by (*cases xs rule*: *rev-cases*) *auto*

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: ‹$a < length\ xs \Longrightarrow m \geq n \Longrightarrow xs\ !\ a = VMTF\text{-}Node\ (n::nat)\ None\ None \Longrightarrow vmtf\text{-}ns\ [a]\ m\ xs$›
|
Cons: ‹$vmtf\text{-}ns\ (b\ \#\ l)\ m\ xs \Longrightarrow a < length\ xs \Longrightarrow xs\ !\ a = VMTF\text{-}Node\ n\ None\ (Some\ b) \Longrightarrow$
$a \neq b \Longrightarrow a \notin set\ l \Longrightarrow n > m \Longrightarrow$
$xs' = xs[b := VMTF\text{-}Node\ (stamp\ (xs!b))\ (Some\ a)\ (get\text{-}next\ (xs!b))] \Longrightarrow n' \geq n \Longrightarrow$
$vmtf\text{-}ns\ (a\ \#\ b\ \#\ l)\ n'\ xs'$›

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

lemma *vmtf-ns-le-length*: ‹$vmtf\text{-}ns\ l\ m\ xs \Longrightarrow i \in set\ l \Longrightarrow i < length\ xs$›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

lemma *vmtf-ns-distinct*: ‹$vmtf\text{-}ns\ l\ m\ xs \Longrightarrow distinct\ l$›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

lemma *vmtf-ns-eq-iff*:
assumes
‹$\forall i \in set\ l.\ xs\ !\ i = zs\ !\ i$› **and**
‹$\forall i \in set\ l.\ i < length\ xs \land i < length\ zs$›
shows ‹$vmtf\text{-}ns\ l\ m\ zs \longleftrightarrow vmtf\text{-}ns\ l\ m\ xs$› (**is** ‹$?A \longleftrightarrow ?B$›)
proof −
have ‹*vmtf-ns l m xs*›
if
‹*vmtf-ns l m zs*› **and**
‹($\forall i \in set\ l.\ xs\ !\ i = zs\ !\ i$)› **and**
‹($\forall i \in set\ l.\ i < length\ xs \land i < length\ zs$)›
for *xs zs*
using *that*
proof (*induction arbitrary*: *xs rule*: *vmtf-ns.induct*)
case (*Nil st xs zs*)

then show *?case* **by** (*auto intro*: *vmtf-ns.intros*)
next
case (*Cons1 a xs n zs*)
show *?case* **by** (*rule vmtf-ns.Cons1*) (*use Cons1* **in** ‹*auto intro*: *vmtf-ns.intros*›)
next
case (*Cons b l m xs c n zs n' zs'*) **note** *vmtf-ns* = *this*(*1*) **and** *a-le-y* = *this*(*2*) **and** *zs-a* = *this*(*3*) **and** *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and** *nn'* = *this*(*8*) **and** *IH* = *this*(*9*) **and** *H* = *this*(*10−*)
have ‹*vmtf-ns* (*c* # *b* # *l*) *n' zs*›
by (*rule vmtf-ns.Cons*[*OF Cons.hyps*])
have [*simp*]: ‹*b* < *length xs*› ‹*b* < *length zs*›
using *H xs'* **by** *auto*
have [*simp*]: ‹*b* ∉ *set l*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*
then have *K*: ‹(∀ *i*∈*set l*. *zs* ! *i* = (*if b* = *i then x else xs* ! *i*) =
(∀ *i*∈*set l*. *zs* ! *i* = *xs* ! *i*)› **for** *x*
using *H*(*2*)
by (*simp add*: *H*(*1*) *xs'*)
have *next-xs-b*: ‹*get-next* (*xs* ! *b*) = *None*› **if** ‹*l* = []›
using *vmtf-ns* **unfolding** *that* **by** (*auto simp*: *elim*!: *vmtf-nsE*)
have *prev-xs-b*: ‹*get-prev* (*xs* ! *b*) = *None*›
using *vmtf-ns* **by** (*auto elim*: *vmtf-nsE*)
have *vmtf-ns-zs*: ‹*vmtf-ns* (*b* # *l*) *m* (*zs'*[*b* := *xs*!*b*])›
apply (*rule IH*)
subgoal using *H*(*1*) *ab next-xs-b prev-xs-b H* **unfolding** *xs'* **by** (*auto simp*: *K*)
subgoal using *H*(*2*) *ab next-xs-b prev-xs-b* **unfolding** *xs'* **by** (*auto simp*: *K*)
done
have ‹*zs'* ! *b* = *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some c*) (*get-next* (*xs* ! *b*))›
using *H*(*1*) **unfolding** *xs'* **by** *auto*
show *?case*
apply (*rule vmtf-ns.Cons*[*OF vmtf-ns-zs*, *of - n*])
subgoal using *a-le-y xs' H*(*2*) **by** *auto*
subgoal using *ab zs-a xs' H*(*1*) **by** (*auto simp*: *K*)
subgoal using *ab* **.**
subgoal using *a-l* **.**
subgoal using *mn* **.**
subgoal using *ab xs' H*(*1*) **by** (*metis H*(*2*) *insert-iff list.set*(*2*) *list-update-id list-update-overwrite nth-list-update-eq*)
subgoal using *nn'* **.**
done
qed
then show *?thesis*
using *assms* **by** *metis*
qed

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*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*rule vmtf-ns.Cons1*) (*auto intro*!: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

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*)›

```
  by (auto 5 5 elim!: vmtf-nsE intro: vmtf-ns.intros)

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))›
  using assms
proof (induction ‹axs @ [ax, ay] @ azs› an ns arbitrary: axs ax ay azs rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by simp
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10−)
  have b-le-xs: ‹b < length xs›
    using vmtf-ns by (auto intro: vmtf-ns-le-length simp: xs')
  show ?case
  proof (cases ‹axs›)
    case [simp]: Nil
    then have [simp]: ‹ax = a› ‹ay = b› ‹azs = l›
      using decomp by auto
    show ?thesis
    proof (cases l)
      case Nil
      then show ?thesis
        using vmtf-ns xs' a-le-y zs-a ab a-l mn nn' by (cases ‹xs ! b›)
          (auto simp: vmtf-ns-single-iff)
    next
      case (Cons al als) note l = this
       have vmtf-ns-b: ‹vmtf-ns [b] m (xs[b := VMTF-Node (stamp (xs ! b)) (get-prev (xs ! b)) None])›
and
        vmtf-ns-l: ‹vmtf-ns (al # als) (stamp (xs ! al))
           (xs[al := VMTF-Node (stamp (xs ! al)) None (get-next (xs ! al))])› and
        stamp-al-b: ‹stamp (xs ! al) < stamp (xs ! b)›
        using IH[of Nil b al als] unfolding l by auto
      have ‹vmtf-ns [a] n' (xs'[a := VMTF-Node (stamp (xs' ! a)) (get-prev (xs' ! a)) None])›
          using a-le-y xs' ab mn nn' zs-a by (auto simp: vmtf-ns-single-iff)
      have al-b[simp]: ‹al ≠ b› and b-als: ‹b ∉ set als›
        using vmtf-ns unfolding l by (auto dest: vmtf-ns-distinct)
      have al-le-xs: ‹al < length xs›
        using vmtf-ns vmtf-ns-l by (auto intro: vmtf-ns-le-length simp: l xs')
      have xs-al: ‹xs ! al = VMTF-Node (stamp (xs ! al)) (Some b) (get-next (xs ! al))›
        using vmtf-ns unfolding l by (auto 5 5 elim: vmtf-nsE)
      have xs-b: ‹xs ! b = VMTF-Node (stamp (xs ! b)) None (get-next (xs ! b))›
        using vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›) (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)

      have ‹vmtf-ns (b # al # als) (stamp (xs' ! b))
          (xs'[b := VMTF-Node (stamp (xs' ! b)) None (get-next (xs' ! b))])›
        apply (rule vmtf-ns.Cons[OF vmtf-ns-l, of - ‹stamp (xs' ! b)›])
        subgoal using b-le-xs by auto
        subgoal using xs-b vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›)
```

    (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)
  subgoal using *al-b* **by** *blast*
  subgoal using *b-als* **.**
  subgoal using *xs′ b-le-xs stamp-al-b* **by** (*simp add*:)
  subgoal using *ab* **unfolding** *xs′* **by** (*simp add*: *b-le-xs al-le-xs xs-al*[*symmetric*]
    xs-b[*symmetric*])
  subgoal by *simp*
  done
moreover have ‹*vmtf-ns* [*a*] *n′*
  (*xs′*[*a* := *VMTF-Node* (*stamp* (*xs′* ! *a*)) (*get-prev* (*xs′* ! *a*)) *None*])›
  using *ab a-le-y mn nn′ zs-a* **by** (*auto simp*: *vmtf-ns-single-iff xs′*)
moreover have ‹*stamp* (*xs′* ! *b*) < *stamp* (*xs′* ! *a*)›
  using *b-le-xs ab mn vmtf-ns-b zs-a* **by** (*auto simp add*: *xs′ vmtf-ns-single-iff*)
ultimately show *?thesis*
  unfolding *l* **by** (*simp add*: *l*)
qed
next
case (*Cons aaxs axs′*) **note** *axs* = *this*
have [*simp*]: ‹*aaxs* = *a*› **and** *bl*: ‹*b* # *l* = *axs′* @ [*ax*, *ay*] @ *azs*›
  using *decomp* **unfolding** *axs* **by** *simp-all*
have
  vmtf-ns-axs′: ‹*vmtf-ns* (*axs′* @ [*ax*]) *m*
    (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) *None*])› **and**
  vmtf-ns-ay: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs* ! *ay*))
    (*xs*[*ay* := *VMTF-Node* (*stamp* (*xs* ! *ay*)) *None* (*get-next* (*xs* ! *ay*))])› **and**
  stamp: ‹*stamp* (*xs* ! *ay*) < *stamp* (*xs* ! *ax*)›
  using *IH*[*OF bl*] **by** *fast+*
have *b-ay*: ‹*b* ≠ *ay*›
  using *bl vmtf-ns-distinct*[*OF vmtf-ns*] **by** (*cases axs′*) *auto*
have *vmtf-ns-ay′*: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs′* ! *ay*))
    (*xs*[*ay* := *VMTF-Node* (*stamp* (*xs* ! *ay*)) *None* (*get-next* (*xs* ! *ay*))])›
  using *vmtf-ns-ay xs′ b-ay* **by** (*auto*)
have [*simp*]: ‹*ay* < *length xs*›
    using *vmtf-ns* **by** (*auto intro*: *vmtf-ns-le-length simp*: *bl xs′*)
have *in-azs-noteq-b*: ‹*i* ∈ *set azs* ⟹ *i* ≠ *b*› **for** *i*
  using *vmtf-ns-distinct*[*OF vmtf-ns*] *bl* **by** (*cases axs′*) (*auto simp*: *xs′ b-ay*)
have *a-ax*[*simp*]: ‹*a* ≠ *ax*›
  using *ab a-l bl* **by** (*cases axs′*) (*auto simp*: *xs′ b-ay*)
have ‹*vmtf-ns* (*axs* @ [*ax*]) *n′*
   (*xs′*[*ax* := *VMTF-Node* (*stamp* (*xs′* ! *ax*)) (*get-prev* (*xs′* ! *ax*)) *None*])›
proof (*cases axs′*)
  case *Nil*
  then have [*simp*]: ‹*ax* = *b*›
    using *bl* **by** *auto*
  have ‹*vmtf-ns* [*ax*] *m* (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) *None*])›
    using *vmtf-ns-axs′* **unfolding** *axs Nil* **by** *simp*
  then have ‹*vmtf-ns* (*aaxs* # *ax* # []) *n′*
    (*xs′*[*ax* := *VMTF-Node* (*stamp* (*xs′* ! *ax*)) (*get-prev* (*xs′* ! *ax*)) *None*])›
    apply (*rule vmtf-ns.Cons*[*of* - - - - - *n*])
    subgoal using *a-le-y* **by** *auto*
    subgoal using *zs-a a-le-y ab* **by** *auto*
    subgoal using *ab* **by** *auto*
    subgoal by *simp*
    subgoal using *mn* **.**
    subgoal using *zs-a a-le-y ab xs′ b-le-xs* **by** *auto*
    subgoal using *nn′* **.**

        done
      then show *?thesis*
        using *vmtf-ns-axs′* **unfolding** *axs Nil* **by** *simp*
    next
      case (*Cons aaaxs′ axs″*)
      have [*simp*]: ‹*aaaxs′* = *b*›
        using *bl* **unfolding** *Cons* **by** *auto*
      have ‹*vmtf-ns* (*aaaxs′* # *axs″* @ [*ax*]) *m*
          (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) *None*])›
        using *vmtf-ns-axs′* **unfolding** *axs Cons* **by** *simp*
      then have ‹*vmtf-ns* (*a* # *aaaxs′* # *axs″* @ [*ax*]) *n′*
          (*xs′*[*ax* := *VMTF-Node* (*stamp* (*xs′* ! *ax*)) (*get-prev* (*xs′* ! *ax*)) *None*])›
        apply (*rule vmtf-ns.Cons*[*of - - - - - n*])
        subgoal using *a-le-y* **by** *auto*
        subgoal using *zs-a a-le-y a-ax ab* **by** (*auto simp del*: ‹*a* ≠ *ax*›)
        subgoal using *ab* **by** *auto*
        subgoal using *a-l bl* **unfolding** *Cons* **by** *simp*
        subgoal using *mn* **.**
        subgoal using *zs-a a-le-y ab xs′ b-le-xs* **by** (*auto simp*: *list-update-swap*)
        subgoal using *nn′* **.**
        done
      then show *?thesis*
        unfolding *axs Cons* **by** *simp*
    qed
    moreover have ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs′* ! *ay*))
        (*xs′*[*ay* := *VMTF-Node* (*stamp* (*xs′* ! *ay*)) *None* (*get-next* (*xs′* ! *ay*))])›
      apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-ay′*])
      subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *bl b-le-xs in-azs-noteq-b* **by** (*auto simp*: *xs′ b-ay*)
      subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] *bl* **unfolding** *xs′* **by** *auto*
      done
    moreover have ‹*stamp* (*xs′* ! *ay*) < *stamp* (*xs′* ! *ax*)›
      using *stamp* **unfolding** *axs xs′* **by** (*auto simp*: *b-le-xs b-ay*)
    ultimately show *?thesis*
      unfolding *axs xs′* **by** *fast*
  qed
qed

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-ns* (*axs* @ [*ax*, *ay*] @ *azs*) *an*
    (*ns*[*ax* := *VMTF-Node* (*stamp* (*ns*!*ax*)) (*get-prev* (*ns*!*ax*)) (*Some ay*) ,
      ay := *VMTF-Node* (*stamp* (*ns*!*ay*)) (*Some ax*) (*get-next* (*ns*!*ay*))])›
  using *assms*
proof (*induction* ‹*axs* @ [*ax*]› *an ns arbitrary*: *axs ax ay azs rule*: *vmtf-ns.induct*)
  case (*Nil st xs*)
  then show *?case* **by** *simp*
next
  case (*Cons1 a xs m n*) **note** *a-le-xs* = *this*(*1*) **and** *nm* = *this*(*2*) **and** *xs-a* = *this*(*3*) **and** *a* = *this*(*4*)
    and *vmtf-ns* = *this*(*5*) **and** *stamp* = *this*(*6*) **and** *dist* = *this*(*7*)
  have *a-ax*: ‹*ax* = *a*›
    using *a* **by** *simp*

have *vmtf-ns-ay'*: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs* ! *ay*)) (*xs*[*ax* := *VMTF-Node n None* (*Some ay*)])›
  apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns*])
  subgoal using *dist a-ax a-le-xs* **by** *auto*
  subgoal using *vmtf-ns vmtf-ns-le-length* **by** *auto*
  done

then have ‹*vmtf-ns* (*ax* # *ay* # *azs*) *m* (*xs*[*ax* := *VMTF-Node n None* (*Some ay*),
    ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
  apply (*rule vmtf-ns.Cons*[*of - - - - -* ‹*stamp* (*xs* ! *a*)›])
  subgoal using *a-le-xs* **unfolding** *a-ax* **by** *auto*
  subgoal using *xs-a a-ax a-le-xs* **by** *auto*
  subgoal using *dist* **by** *auto*
  subgoal using *dist* **by** *auto*
  subgoal using *stamp* **by** (*simp add*: *a-ax*)
  subgoal using *a-ax a-le-xs dist* **by** *auto*
  subgoal by (*simp add*: *nm xs-a*)
  done
then show *?case*
  using *a-ax a xs-a* **by** *auto*
next
case (*Cons b l m xs a n xs' n'*) **note** *vmtf-ns* = *this*(*1*) **and** *IH* = *this*(*2*) **and** *a-le-y* = *this*(*3*) **and** *zs-a* = *this*(*4*) **and** *ab* = *this*(*5*) **and** *a-l* = *this*(*6*) **and** *mn* = *this*(*7*) **and** *xs'* = *this*(*8*) **and** *nn'* = *this*(*9*) **and** *decomp* = *this*(*10*) **and** *vmtf-ns-ay* = *this*(*11*) **and** *stamp* = *this*(*12*) **and** *dist* = *this*(*13*)

have *dist-b*: ‹*distinct* ((*a* # *b* # *l*) @ *ay* # *azs*)›
  using *dist* **unfolding** *decomp* **by** *auto*
then have *b-ay*: ‹*b* ≠ *ay*›
  by *auto*
have *b-le-xs*: ‹*b* < *length xs*›
  using *vmtf-ns vmtf-ns-le-length* **by** *auto*
have *a-ax*: ‹*a* ≠ *ax*› **and** *a-ay*: ‹*a* ≠ *ay*›
  using *dist-b decomp dist* **by** (*cases axs*; *auto*)+
have *vmtf-ns-ay'*: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs* ! *ay*)) *xs*›
  apply (*rule vmtf-ns-eq-iffI*[*of - - xs'*])
  subgoal using *xs' b-ay dist-b b-le-xs* **by** *auto*
  subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-ay*] *xs'* **by** *auto*
  subgoal using *xs' b-ay dist-b b-le-xs vmtf-ns-ay xs'* **by** *auto*
  done

have ‹*vmtf-ns* (*tl axs* @ [*ax*, *ay*] @ *azs*) *m*
    (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) (*Some ay*),
      ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
  apply (*rule IH*)
  subgoal using *decomp* **by** (*cases axs*) *auto*
  subgoal using *vmtf-ns-ay'* **.**
  subgoal using *stamp xs' b-ay b-le-xs* **by** (*cases* ‹*ax* = *b*›) *auto*
  subgoal using *dist* **by** (*cases axs*) *auto*
  done
moreover have ‹*tl axs* @ [*ax*, *ay*] @ *azs* = *b* # *l* @ *ay* # *azs*›
  using *decomp* **by** (*cases axs*) *auto*
ultimately have *vmtf-ns-tl-axs*: ‹*vmtf-ns* (*b* # *l* @ *ay* # *azs*) *m*
    (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) (*Some ay*),
      ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
  by *metis*

then have ‹*vmtf-ns* (*a* # *b* # *l* @ *ay* # *azs*) *n'*
  (*xs'*[*ax* := *VMTF-Node* (*stamp* (*xs'* ! *ax*)) (*get-prev* (*xs'* ! *ax*)) (*Some ay*),
    ay := *VMTF-Node* (*stamp* (*xs'* ! *ay*)) (*Some ax*) (*get-next* (*xs'* ! *ay*))])›
 apply (*rule vmtf-ns.Cons*[*of* - - - - - ‹*stamp* (*xs* ! *a*)›])
 subgoal using *a-le-y* **by** *simp*
 subgoal using *zs-a a-le-y a-ax a-ay* **by** *auto*
 subgoal using *ab* **.**
 subgoal using *dist-b* **by** *auto*
 subgoal using *mn* **by** (*simp add*: *zs-a*)
 subgoal using *zs-a a-le-y a-ax a-ay b-ay b-le-xs* **unfolding** *xs'*
  by (*auto simp*: *list-update-swap*)
 subgoal using *stamp xs' nn' b-ay b-le-xs zs-a* **by** *auto*
 done
then show *?case*
 by (*metis append.assoc append-Cons append-Nil decomp*)
qed

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

lemma *vmtf-ns-different-same-neq*: ‹*vmtf-ns* (*b* # *c* # *l'*) *m xs* ⟹ *vmtf-ns* (*c* # *l'*) *m xs* ⟹ *False*›
 apply (*cases l'*)
 subgoal by (*force elim*: *vmtf-nsE*)
 subgoal for *x xs*
  apply (*subst* (*asm*) *vmtf-ns.simps*)
  apply (*subst* (*asm*)(*2*) *vmtf-ns.simps*)
  by (*metis* (*no-types*, *lifting*) *vmtf-node.inject length-list-update list.discI list-tail-coinc*
   nth-list-update-eq nth-list-update-neq option.discI)
 done

lemma *vmtf-ns-last-next*:
 ‹*vmtf-ns* (*xs* @ [*x*]) *m ns* ⟹ *get-next* (*ns* ! *x*) = *None*›
 apply (*induction xs* @ [*x*] *m ns arbitrary*: *xs x rule*: *vmtf-ns.induct*)
 subgoal by *auto*
 subgoal by *auto*
 subgoal for *b l m xs a n xs' n' xsa x*
  by (*cases* ‹*xs* ! *b*›; *cases* ‹*x* = *b*›; *cases xsa*)
   (*force simp*: *vmtf-ns-le-length*)+
 done

lemma *vmtf-ns-hd-prev*:

```
  ‹vmtf-ns (x # xs) m ns ⟹ get-prev (ns ! x) = None›
  apply (induction x # xs m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  done

lemma vmtf-ns-last-mid-get-next:
  ‹vmtf-ns (xs @ [x, y] @ zs) m ns ⟹ get-next (ns ! x) = Some y›
  apply (induction xs @ [x, y] @ zs m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  subgoal for b l m xs a n xs' n' xsa x
    by (cases ‹xs ! b›; cases ‹x = b›; cases xsa)
      (force simp: vmtf-ns-le-length)+
  done

lemma vmtf-ns-last-mid-get-next-option-hd:
  ‹vmtf-ns (xs @ x # zs) m ns ⟹ get-next (ns ! x) = option-hd zs›
  using vmtf-ns-last-mid-get-next[of xs x ‹hd zs› ‹tl zs› m ns]
  vmtf-ns-last-next[of xs x]
  by (cases zs) auto

lemma vmtf-ns-last-mid-get-prev:
  assumes ‹vmtf-ns (xs @ [x, y] @ zs) m ns›
  shows ‹get-prev (ns ! y) = Some x›
    using assms
proof (induction xs @ [x, y] @ zs m ns arbitrary: xs x rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by auto
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xxs a n xxs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10)
  show ?case
  proof (cases xs)
    case Nil
    then show ?thesis using Cons vmtf-ns-le-length by auto
  next
    case (Cons aaxs axs')
    then have b-l: ‹b # l = tl xs @ [x, y] @ zs›
      using decomp by auto
    then have ‹get-prev (xxs ! y) = Some x›
      by (rule IH)
    moreover have ‹x ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] b-l by auto
    moreover have ‹b ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] decomp by (cases axs') (auto simp add: Cons)
    moreover have ‹y < length xxs› ‹b < length xxs›
      using vmtf-ns-le-length[OF vmtf-ns, unfolded b-l] vmtf-ns-le-length[OF vmtf-ns] by auto
    ultimately show ?thesis
      unfolding xs' by auto
  qed
qed
```

lemma *vmtf-ns-last-mid-get-prev-option-last*:
‹*vmtf-ns* (*xs* @ *x* # *zs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *option-last xs*›
using *vmtf-ns-last-mid-get-prev*[*of* ‹*butlast xs*› ‹*last xs*› ‹*x*› ‹*zs*› *m ns*]
by (*cases xs rule*: *rev-cases*) (*auto elim*: *vmtf-nsE*)

lemma *length-ns-vmtf-dequeue*[*simp*]: ‹*length* (*ns-vmtf-dequeue x ns*) = *length ns*›
unfolding *ns-vmtf-dequeue-def* **by** (*auto simp*: *Let-def split*: *option.splits*)

lemma *vmtf-ns-skip-fst*:
assumes *vmtf-ns*: ‹*vmtf-ns* (*x* # *y'* # *zs'*) *m ns*›
shows ‹∃ *n*. *vmtf-ns* (*y'* # *zs'*) *n* (*ns*[*y'* := *VMTF-Node* (*stamp* (*ns* ! *y'*)) *None* (*get-next* (*ns* ! *y'*))]) ∧ *m* ≥ *n*›
using *assms*
proof (*rule vmtf-nsE*, *goal-cases*)
case *1*
then show *?case* **by** *simp*
next
case (*2 a n*)
then show *?case* **by** *simp*
next
case (*3 b l m xs a n*)
moreover have ‹*get-prev* (*xs* ! *b*) = *None*›
using *3*(*3*) **by** (*fastforce elim*: *vmtf-nsE*)
moreover have ‹*b* < *length xs*›
using *3*(*3*) *vmtf-ns-le-length* **by** *auto*
ultimately show *?case*
by (*cases* ‹*xs* ! *b*›) (*auto simp*: *eq-commute*[*of* ‹*xs* ! *b*›])
qed

definition *vmtf-ns-notin* **where**
‹*vmtf-ns-notin l m xs* ⟷ (∀ *i*<*length xs*. *i*∉*set l* ⟶ (*get-prev* (*xs* ! *i*) = *None* ∧ *get-next* (*xs* ! *i*) = *None*))›

lemma *vmtf-ns-notinI*:
‹(⋀*i*. *i* <*length xs* ⟹ *i*∉*set l* ⟹ *get-prev* (*xs* ! *i*) = *None* ∧ *get-next* (*xs* ! *i*) = *None*) ⟹ *vmtf-ns-notin l m xs*›
by (*auto simp*: *vmtf-ns-notin-def*)

lemma *stamp-ns-vmtf-dequeue*:
‹*axs* < *length zs* ⟹ *stamp* (*ns-vmtf-dequeue x zs* ! *axs*) = *stamp* (*zs* ! *axs*)›
by (*cases* ‹*zs* ! (*the* (*get-next* (*zs* ! *x*)))›; *cases* ‹(*the* (*get-next* (*zs* ! *x*))) = *axs*›; *cases* ‹(*the* (*get-prev* (*zs* ! *x*))) = *axs*›; *cases* ‹*zs* ! *x*›)
(*auto simp*: *nth-list-update'* *ns-vmtf-dequeue-def Let-def split*: *option.splits*)

lemma *sorted-many-eq-append*: ‹*sorted* (*xs* @ [*x*, *y*]) ⟷ *sorted* (*xs* @ [*x*]) ∧ *x* ≤ *y*›
using *sorted-append*[*of* ‹*xs* @ [*x*]› ‹[*y*]›] *sorted-append*[*of xs* ‹[*x*]›]
by *force*

lemma *vmtf-ns-stamp-sorted*:
assumes ‹*vmtf-ns l m ns*›
shows ‹*sorted* (*map* (λ*a*. *stamp* (*ns*!*a*)) (*rev l*)) ∧ (∀ *a* ∈ *set l*. *stamp* (*ns*!*a*) ≤ *m*)›
using *assms*
proof (*induction rule*: *vmtf-ns.induct*)
case (*Cons b l m xs a n xs' n'*) **note** *vmtf-ns* = *this*(*1*) **and** *IH* = *this*(*9*) **and** *a-le-y* = *this*(*2*) **and** *zs-a* = *this*(*3*) **and** *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and**

$nn' = this(8)$
have *H*:
‹*map* (*λaa*. *stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *aa*)) (*rev l*) =
map (*λa*. *stamp* (*xs* ! *a*)) (*rev l*)›
apply (*rule map-cong*)
subgoal by *auto*
subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **by** *auto*
done
have [*simp*]: ‹*stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *b*) =
stamp (*xs* ! *b*)›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **by** (*cases* ‹*xs* ! *b*›) *auto*
have ‹*stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *aa*) ≤ *n'*›
if ‹*aa* ∈ *set l*› **for** *aa*
apply (*cases* ‹*aa* = *b*›)
subgoal using *Cons* **by** *auto*
subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] *IH nn' mn that* **by** *auto*
done
then show *?case*
using *Cons* **by** (*auto simp*: *H sorted-many-eq-append*)
qed *auto*

lemma *vmtf-ns-ns-vmtf-dequeue*:
assumes *vmtf-ns*: ‹*vmtf-ns l m ns*› **and** *notin*: ‹*vmtf-ns-notin l m ns*› **and** *valid*: ‹*x* < *length ns*›
shows ‹*vmtf-ns* (*remove1 x l*) *m* (*ns-vmtf-dequeue x ns*)›
proof (*cases* ‹*x* ∈ *set l*›)
case *False*
then have *H*: ‹*remove1 x l* = *l*›
by (*simp add*: *remove1-idem*)
have *simp-is-stupid*[*simp*]: ‹*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *a* ≠ *x*› ‹*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *x* ≠ *a*›
for *a x*
by *auto*
have
‹*get-prev* (*ns* ! *x*) = *None* › **and**
‹*get-next* (*ns* ! *x*) = *None*›
using *notin False valid* **unfolding** *vmtf-ns-notin-def* **by** *auto*
then have *vmtf-ns-eq*: ‹(*ns-vmtf-dequeue x ns*) ! *a* = *ns* ! *a*› **if** ‹*a* ∈ *set l*› **for** *a*
using *that False valid* **unfolding** *vmtf-ns-notin-def ns-vmtf-dequeue-def*
by (*cases* ‹*ns* ! (*the* (*get-prev* (*ns* ! *x*)))›; *cases* ‹*ns* ! (*the* (*get-next* (*ns* ! *x*)))›)
(*auto simp*: *Let-def split*: *option.splits*)
show *?thesis*
unfolding *H*
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns*])
subgoal using *vmtf-ns-eq* **by** *blast*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **by** *auto*
done
next
case *True*
then obtain *xs zs* **where**
l: ‹*l* = *xs* @ *x* # *zs*›
by (*meson split-list*)
have *r-l*: ‹*remove1 x l* = *xs* @ *zs*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *l* **by** (*simp add*: *remove1-append*)
have *dist*: ‹*distinct l*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **.**
have *le-length*: ‹*i* ∈ *set l* ⟹ *i* < *length ns*› **for** *i*
using *vmtf-ns-le-length*[*OF vmtf-ns*] **.**

consider
  (*xs-zs-empty*) ‹$xs = []$› **and** ‹$zs = []$› |
  (*xs-nempty-zs-empty*) x' xs' **where** ‹$xs = xs'$ @ $[x']$› **and** ‹$zs = []$› |
  (*xs-empty-zs-nempty*) y' zs' **where** ‹$xs = []$› **and** ‹$zs = y'$ # zs'› |
  (*xs-zs-nempty*) x' y' xs' zs' **where** ‹$xs = xs'$ @ $[x']$› **and** ‹$zs = y'$ # zs'›
  by (*cases xs rule*: *rev-cases*; *cases zs*)
then show *?thesis*
proof *cases*
  case *xs-zs-empty*
  then show *?thesis*
    using *vmtf-ns* **by** (*auto simp*: *r-l intro*: *vmtf-ns.intros*)
next
  case *xs-empty-zs-nempty* **note** $xs = this(1)$ **and** $zs = this(2)$
  have [*simp*]: ‹$x \neq y'$› ‹$y' \neq x$› ‹$x \notin set\ zs'$›
    using *dist* **unfolding** *l xs zs* **by** *auto*
  have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *None*› ‹*get-next* (*ns* ! *x*) = *option-hd zs*›
    using *vmtf-ns* **unfolding** *l xs zs*
    by (*cases zs*; *auto 5 5 simp*: *option-hd-def elim*: *vmtf-nsE*; *fail*)+
  then have *vmtf'*: ‹*vmtf-ns* (y' # zs') *m* (*ns*[y' := *VMTF-Node* (*stamp* (*ns* ! y')) *None* (*get-next* (*ns* ! y'))])›
    using *vmtf-ns* **unfolding** *r-l* **unfolding** *l xs zs*
    by (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update' zs*
        split: *option.splits*
        intro: *vmtf-ns.intros vmtf-ns-stamp-increase dest*: *vmtf-ns-skip-fst*)
  show *?thesis*
    apply (*rule vmtf-ns-eq-iffI*[*of - -*
        ‹(*ns*[y' := *VMTF-Node* (*stamp* (*ns* ! y')) *None* (*get-next* (*ns* ! y'))])› *m*])
    subgoal
      using *prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
      by (*cases* ‹*ns* ! *x*›) (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update'*)
    subgoal
      using *prev-next le-length* **unfolding** *r-l* **unfolding** *l xs zs*
      by (*cases* ‹*ns* ! *x*›) *auto*
    subgoal
      using *vmtf'* **unfolding** *r-l* **unfolding** *l xs zs* **by** *auto*
    done
next
  case *xs-nempty-zs-empty* **note** $xs = this(1)$ **and** $zs = this(2)$
  have [*simp*]: ‹$x \neq x'$› ‹$x' \neq x$› ‹$x' \notin set\ xs'$› ‹$x \notin set\ xs'$›
    using *dist* **unfolding** *l xs zs* **by** *auto*
  have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *Some* x'› ‹*get-next* (*ns* ! *x*) = *None*›
    using *vmtf-ns vmtf-ns-append-decomp*[*of* xs' x' *x zs m ns*] **unfolding** *l xs zs*
    by (*auto simp*: *vmtf-ns-single-iff intro*: *vmtf-ns-last-mid-get-prev*)
  then have *vmtf'*: ‹*vmtf-ns* (xs' @ [x']) *m* (*ns*[x' := *VMTF-Node* (*stamp* (*ns* ! x')) (*get-prev* (*ns* ! x')) *None*])›
    using *vmtf-ns* **unfolding** *r-l* **unfolding** *l xs zs*
    by (*auto simp*: *ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp split*: *option.splits*
        intro: *vmtf-ns.intros*)
  show *?thesis*
    apply (*rule vmtf-ns-eq-iffI*[*of - -*
        ‹(*ns*[x' := *VMTF-Node* (*stamp* (*ns* ! x')) (*get-prev* (*ns* ! x')) *None*])› *m*])
    subgoal
      using *prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
      by (*cases* ‹*ns* ! x'›) (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update'*)
    subgoal
      using *prev-next le-length* **unfolding** *r-l* **unfolding** *l xs zs*

by (*cases* ‹*ns* ! *x*›) *auto*
subgoal
using *vmtf′* **unfolding** *r-l* **unfolding** *l xs zs* **by** *auto*
done
next
case *xs-zs-nempty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*2*)
have *vmtf-ns-x′-x*: ‹*vmtf-ns* (*xs′* @ [*x′*, *x*] @ (*y′* # *zs′*)) *m ns*› **and**
vmtf-ns-x-y: ‹*vmtf-ns* ((*xs′* @ [*x′*]) @ [*x*, *y′*] @ *zs′*) *m ns*›
using *vmtf-ns* **unfolding** *l xs zs* **by** *simp-all*
from *vmtf-ns-append-decomp*[*OF vmtf-ns-x′-x*] **have**
vmtf-ns-xs: ‹*vmtf-ns* (*xs′* @ [*x′*]) *m* (*ns*[*x′* := *VMTF-Node* (*stamp* (*ns* ! *x′*)) (*get-prev* (*ns* ! *x′*)) *None*])› **and**
vmtf-ns-zs: ‹*vmtf-ns* (*x* # *y′* # *zs′*) (*stamp* (*ns* ! *x*)) (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))])› **and**
stamp: ‹*stamp* (*ns* ! *x*) < *stamp* (*ns* ! *x′*)›
by *fast*+
have [*simp*]: ‹*y′* < *length ns*› ‹*x* < *length ns*› ‹*x* ≠ *y′*› ‹*x′* ≠ *y′*› ‹*x′* < *length ns*› ‹*y′* ≠ *x′*›
‹*x′* ≠ *x*› ‹*x* ≠ *x′*› ‹*y′* ≠ *x*›
and *x-zs′*: ‹*x* ∉ *set zs′*› ‹*x* ∉ *set xs′*› **and** *x′-zs′*: ‹*x′* ∉ *set zs′*› **and** *y′-xs′*: ‹*y′* ∉ *set xs′*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l xs zs*
by *auto*
obtain *n* **where**
vmtf-ns-zs′: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*)),
y′ := *VMTF-Node* (*stamp* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))] ! *y′*)) *None*
(*get-next* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))] ! *y′*))])› **and**
‹*n* ≤ *stamp* (*ns* ! *x*)›
using *vmtf-ns-skip-fst*[*OF vmtf-ns-zs*] **by** *blast*
then have *vmtf-ns-y′-zs′-x-y′*: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*)),
y′ := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
by *auto*

define *ns′* **where** ‹*ns′* = *ns*[*x′* := *VMTF-Node* (*stamp* (*ns* ! *x′*)) (*get-prev* (*ns* ! *x′*)) *None*,
y′ := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))]›
have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*y′* := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-x-y′*])
subgoal using *x-zs′* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l xs zs* **by** *auto*
done
moreover have *stamp-y′-n*: ‹*stamp* (*ns*[*x′* := *VMTF-Node* (*stamp* (*ns* ! *x′*)) (*get-prev* (*ns* ! *x′*)) *None*] ! *y′*) ≤ *n*›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns-y′-zs′-y′*] *stamp* **unfolding** *l xs zs*
by (*auto simp*: *sorted-append*)
ultimately have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) (*stamp* (*ns′* ! *y′*))
(*ns*[*y′* := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
using *l vmtf-ns vmtf-ns-append-decomp xs-zs-nempty*(*2*) *ns′-def* **by** *auto*
have *vmtf-ns-y′-zs′-x′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) (*stamp* (*ns′* ! *y′*)) *ns′*›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-y′*])
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** *l xs zs* **by** *auto*
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** *l xs zs* **by** *auto*
done
have *vmtf-ns-xs′*: ‹*vmtf-ns* (*xs′* @ [*x′*]) *m ns′*›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-xs*])
subgoal using *y′-xs′ ns′-def* **by** *auto*

subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-xs*] *ns′-def* **by** *auto*
done
have *vmtf-x′-y′*: ‹*vmtf-ns* (*xs′* @ [*x′*, *y′*] @ *zs′*) *m*
(*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*))])›
apply (*rule vmtf-ns-append-rebuild*[*OF vmtf-ns-xs′ vmtf-ns-y′-zs′-x′-y′*])
subgoal using *stamp-y′-n vmtf-ns-xs vmtf-ns-zs stamp* ‹$n \leq$ *stamp* (*ns* ! *x*)›
unfolding *ns′-def* **by** *auto*
subgoal by (*metis append.assoc append-Cons distinct-remove1 r-l self-append-conv2 vmtf-ns vmtf-ns-distinct xs zs*)
done
have ‹*vmtf-ns* (*xs′* @ [*x′*, *y′*] @ *zs′*) *m*
(*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*)),
x := *VMTF-Node* (*stamp* (*ns′* ! *x*)) *None None*])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-x′-y′*])
subgoal
using *vmtf-ns-last-mid-get-next*[*OF vmtf-ns-x-y*] *vmtf-ns-last-mid-get-prev*[*OF vmtf-ns-x′-x*] *x-zs′*
by (*cases* ‹*ns*!*x*›; *auto simp*: *nth-list-update′ ns′-def*)
subgoal using *le-length* **unfolding** *l xs zs ns′-def* **by** *auto*
done
moreover have ‹*xs′* @ [*x′*, *y′*] @ *zs′* = *remove1 x l*›
unfolding *r-l xs zs* **by** *auto*
moreover have ‹*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*)),
x := *VMTF-Node* (*stamp* (*ns′* ! *x*)) *None None*] = *ns-vmtf-dequeue x ns*›
using *vmtf-ns-last-mid-get-next*[*OF vmtf-ns-x-y*] *vmtf-ns-last-mid-get-prev*[*OF vmtf-ns-x′-x*]
list-update-swap[*of x′ y′* - ‹- :: *nat-vmtf-node*›]
unfolding *ns′-def ns-vmtf-dequeue-def*
by (*auto simp*: *Let-def*)
ultimately show *?thesis*
by *force*
qed
qed

lemma *vmtf-ns-hd-next*:
‹*vmtf-ns* (*x* # *a* # *list*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *Some a*›
by (*auto 5 5 elim*: *vmtf-nsE*)

lemma *vmtf-ns-notin-dequeue*:
assumes *vmtf-ns*: ‹*vmtf-ns l m ns*› **and** *notin*: ‹*vmtf-ns-notin l m ns*› **and** *valid*: ‹$x <$ *length ns*›
shows ‹*vmtf-ns-notin* (*remove1 x l*) *m* (*ns-vmtf-dequeue x ns*)›
proof (*cases* ‹$x \in$ *set l*›)
case *False*
then have *H*: ‹*remove1 x l* = *l*›
by (*simp add*: *remove1-idem*)
have *simp-is-stupid*[*simp*]: ‹$a \in$ *set l* ⟹ $x \notin$ *set l* ⟹ $a \neq x$› ‹$a \in$ *set l* ⟹ $x \notin$ *set l* ⟹ $x \neq a$›
for *a x*
by *auto*
have
‹*get-prev* (*ns* ! *x*) = *None*› **and**
‹*get-next* (*ns* ! *x*) = *None*›
using *notin False valid* **unfolding** *vmtf-ns-notin-def* **by** *auto*
show *?thesis*
using *notin valid False* **unfolding** *vmtf-ns-notin-def*
by (*auto simp*: *vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def H split*: *option.splits*)

next
  case *True*
  then obtain *xs zs* **where**
    l: ‹$l = xs @ x \# zs$›
    by (*meson split-list*)
  have *r-l*: ‹*remove1 x l* = *xs* @ *zs*›
    using *vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *l* **by** (*simp add*: *remove1-append*)

  consider
    (*xs-zs-empty*) ‹*xs* = []› **and** ‹*zs* = []› |
    (*xs-nempty-zs-empty*) *x′ xs′* **where** ‹*xs* = *xs′* @ [*x′*]› **and** ‹*zs* = []› |
    (*xs-empty-zs-nempty*) *y′ zs′* **where** ‹*xs* = []› **and** ‹*zs* = *y′* # *zs′*› |
    (*xs-zs-nempty*) *x′ y′ xs′ zs′* **where** ‹*xs* = *xs′* @ [*x′*]› **and** ‹*zs* = *y′* # *zs′*›
    by (*cases xs rule*: *rev-cases*; *cases zs*)
  then show *?thesis*
  proof *cases*
    case *xs-zs-empty*
    then show *?thesis*
      using *notin vmtf-ns* **unfolding** *l* **apply** (*cases* ‹*ns* ! *x*›)
        by (*auto simp*: *vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def vmtf-ns-single-iff*
          split: *option.splits*)
  next
    case *xs-empty-zs-nempty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*1*)
    have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *None*› ‹*get-next* (*ns* ! *x*) = *option-hd zs*›
      using *vmtf-ns* **unfolding** *l xs zs*
      by (*cases zs*; *auto simp*: *option-hd-def elim*: *vmtf-nsE dest*: *vmtf-ns-hd-next*)+
    show *?thesis*
      apply (*rule vmtf-ns-notinI*)
      apply (*case-tac* ‹*i* = *x*›)
      subgoal
        using *vmtf-ns prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
        by (*cases zs*) (*auto simp*: *ns-vmtf-dequeue-def Let-def*
            vmtf-ns-notin-def vmtf-ns-single-iff
            split: *option.splits*)
      subgoal
        using *vmtf-ns notin prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
        by (*auto simp*: *ns-vmtf-dequeue-def Let-def*
            vmtf-ns-notin-def vmtf-ns-single-iff
            split: *option.splits*
            intro: *vmtf-ns.intros vmtf-ns-stamp-increase dest*: *vmtf-ns-skip-fst*)
      done
  next
    case *xs-nempty-zs-empty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*2*)
    have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *Some x′*› ‹*get-next* (*ns* ! *x*) = *None*›
      using *vmtf-ns vmtf-ns-append-decomp*[*of xs′ x′ x zs m ns*] **unfolding** *l xs zs*
      by (*auto simp*: *vmtf-ns-single-iff intro*: *vmtf-ns-last-mid-get-prev*)
    then show *?thesis*
      using *vmtf-ns notin* **unfolding** *r-l* **unfolding** *l xs zs*
      by (*auto simp*: *ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp vmtf-ns-notin-def*
          split: *option.splits*
          intro: *vmtf-ns.intros*)
  next
    case *xs-zs-nempty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*2*)
    have *vmtf-ns-x′-x*: ‹*vmtf-ns* (*xs′* @ [*x′*, *x*] @ (*y′* # *zs′*)) *m ns*› **and**
      vmtf-ns-x-y: ‹*vmtf-ns* ((*xs′* @ [*x′*]) @ [*x*, *y′*] @ *zs′*) *m ns*›
      using *vmtf-ns* **unfolding** *l xs zs* **by** *simp-all*

```
    have [simp]: ‹y' < length ns› ‹x < length ns› ‹x ≠ y'› ‹x' ≠ y'› ‹x' < length ns› ‹y' ≠ x'›
      ‹y' ≠ x› ‹y' ∉ set xs›  ‹y' ∉ set zs'›
      and x-zs': ‹x ∉ set zs'› and x'-zs': ‹x' ∉ set zs'› and y'-xs': ‹y' ∉ set xs'›
      using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] unfolding l xs zs
      by auto
    have ‹get-next (ns!x) = Some y'› ‹get-prev (ns!x) = Some x'›
      using vmtf-ns-last-mid-get-prev[OF vmtf-ns-x'-x] vmtf-ns-last-mid-get-next[OF vmtf-ns-x-y]
      by fast+
    then show ?thesis
      using notin x-zs' x'-zs' y'-xs' unfolding l xs zs
      by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def)
  qed
qed

lemma vmtf-ns-stamp-distinct:
  assumes ‹vmtf-ns l m ns›
  shows ‹distinct (map (λa. stamp (ns!a)) l)›
  using assms
proof (induction rule: vmtf-ns.induct)
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(9) and a-le-y = this(2) and
    zs-a = this(3) and ab = this(4) and a-l = this(5) and mn = this(6) and xs' = this(7) and
    nn' = this(8)
  have [simp]: ‹map (λaa. stamp
               (if b = aa
                then VMTF-Node (stamp (xs ! aa)) (Some a) (get-next (xs ! aa))
                else xs ! aa)) l =
         map (λaa. stamp (xs ! aa)) l
       › for a
    apply (rule map-cong)
    subgoal ..
    subgoal using vmtf-ns-distinct[OF vmtf-ns] by auto
    done
  show ?case
    using Cons vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns]
    by (auto simp: sorted-many-eq-append leD vmtf-ns-stamp-sorted cong: if-cong)
qed auto

lemma vmtf-ns-thighten-stamp:
  assumes vmtf-ns: ‹vmtf-ns l m xs› and n: ‹∀ a∈set l. stamp (xs ! a) ≤ n›
  shows ‹vmtf-ns l n xs›
proof −
  consider
    (empty) ‹l = []› |
    (single) x where ‹l = [x]› |
    (more-than-two) x y ys where ‹l = x # y # ys›
    by (cases l; cases ‹tl l›) auto
  then show ?thesis
  proof cases
    case empty
    then show ?thesis by (auto intro: vmtf-ns.intros)
  next
    case (single x)
    then show ?thesis using n vmtf-ns by (auto simp: vmtf-ns-single-iff)
  next
    case (more-than-two x y ys) note l = this
    then have vmtf-ns': ‹vmtf-ns ([] @ [x, y] @ ys) m xs›
```

using *vmtf-ns* **by** *auto*
from *vmtf-ns-append-decomp*[*OF this*] **have**
‹*vmtf-ns* ([*x*]) *m* (*xs*[*x* := *VMTF-Node* (*stamp* (*xs* ! *x*)) (*get-prev* (*xs* ! *x*)) *None*])› **and**
vmtf-ns-y-ys: ‹*vmtf-ns* (*y* # *ys*) (*stamp* (*xs* ! *y*))
(*xs*[*y* := *VMTF-Node* (*stamp* (*xs* ! *y*)) *None* (*get-next* (*xs* ! *y*))])› **and**
‹*stamp* (*xs* ! *y*) < *stamp* (*xs* ! *x*)›
by *auto*
have [*simp*]: ‹$x \neq y$› ‹$x \notin$ *set ys*› ‹*x* < *length xs*› ‹*y* < *length xs*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l* **by** *auto*
show *?thesis*
unfolding *l*
apply (*rule vmtf-ns.Cons*[*OF vmtf-ns-y-ys*, *of* - ‹*stamp* (*xs* ! *x*)›])
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l* **by** *auto*
subgoal using *vmtf-ns* **unfolding** *l* **by** (*cases* ‹*xs* ! *x*›) (*auto elim*: *vmtf-nsE*)
subgoal by *simp*
subgoal by *simp*
subgoal using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *vmtf-ns-stamp-distinct*[*OF vmtf-ns*]
by (*auto simp*: *l sorted-many-eq-append*)
subgoal
using *vmtf-ns vmtf-ns-last-mid-get-prev*[*OF vmtf-ns'*]
apply (*cases* ‹*xs* ! *y*›)
by *simp* (*auto simp*: *l eq-commute*[*of* ‹*xs* ! *y*›])
subgoal using *n* **unfolding** *l* **by** *auto*
done
qed
qed

lemma *vmtf-ns-rescale*:
assumes
‹*vmtf-ns l m xs*› **and**
‹*sorted* (*map* (λa. *st* ! *a*) (*rev l*))› **and** ‹*distinct* (*map* (λa. *st* ! *a*) *l*)›
‹$\forall$ *a* $\in$ *set l*. *get-prev* (*zs* ! *a*) = *get-prev* (*xs* ! *a*)› **and**
‹$\forall$ *a* $\in$ *set l*. *get-next* (*zs* ! *a*) = *get-next* (*xs* ! *a*)› **and**
‹$\forall$ *a* $\in$ *set l*. *stamp* (*zs* ! *a*) = *st* ! *a*› **and**
‹*length xs* $\leq$ *length zs*› **and**
‹$\forall$ *a*$\in$*set l*. *a* < *length st*› **and**
m': ‹$\forall$ *a* $\in$ *set l*. *st* ! *a* < *m'*›
shows ‹*vmtf-ns l m' zs*›
using *assms*
proof (*induction arbitrary*: *zs m' rule*: *vmtf-ns.induct*)
case (*Nil st xs*)
then show *?case* **by** (*auto intro*: *vmtf-ns.intros*)
next
case (*Cons1 a xs m n*)
then show *?case* **by** (*cases* ‹*zs* ! *a*›) (*auto simp*: *vmtf-ns-single-iff intro*!: *Max-ge nth-mem*)
next
case (*Cons b l m xs a n xs' n' zs m'*) **note** *vmtf-ns* = *this*(*1*) **and** *a-le-y* = *this*(*2*) **and**
zs-a = *this*(*3*) **and** *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and**
nn' = *this*(*8*) **and** *IH* = *this*(*9*) **and** *H* = *this*(*10*−)
have [*simp*]: ‹*b* < *length xs*› ‹$b \neq a$› ‹$a \neq b$› ‹$b \notin$ *set l*› ‹*b* < *length zs*› ‹*a* < *length zs*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] *ab H*(*6*) *a-le-y* **unfolding** *xs'*
by *force+*

have *simp-is-stupid*[*simp*]: ‹*a* $\in$ *set l* $\Longrightarrow$ *x* $\notin$ *set l* $\Longrightarrow$ $a \neq x$› ‹*a* $\in$ *set l* $\Longrightarrow$ *x* $\notin$ *set l* $\Longrightarrow$ $x \neq a$›
for *a x*
by *auto*

define *zs'* **where** ‹*zs'* ≡ (*zs*[*b* := *VMTF-Node* (*st* ! *b*) (*get-prev* (*xs* ! *b*)) (*get-next* (*xs* ! *b*)),
    a := *VMTF-Node* (*st* ! *a*) *None* (*Some b*)])›
  have *zs-upd-zs*: ‹*zs* = *zs*
    [*b* := *VMTF-Node* (*st* ! *b*) (*get-prev* (*xs* ! *b*)) (*get-next* (*xs* ! *b*)),
     a := *VMTF-Node* (*st* ! *a*) *None* (*Some b*),
     b := *VMTF-Node* (*st* ! *b*) (*Some a*) (*get-next* (*xs* ! *b*))]
    ›
    using *H*(*2*−*5*) *xs'* *zs-a* ‹*b* < *length xs*›
    by (*metis list.set-intros*(*1*) *list.set-intros*(*2*) *list-update-id list-update-overwrite*
      nth-list-update-eq nth-list-update-neq vmtf-node.collapse vmtf-node.sel(*2*,*3*))

  have *vtmf-b-l*: ‹*vmtf-ns* (*b* # *l*) *m'* *zs'*›
    unfolding *zs'-def*
    apply (*rule IH*)
    subgoal using *H*(*1*) **by** (*simp add*: *sorted-many-eq-append*)
    subgoal using *H*(*2*) **by** *auto*
    subgoal using *H*(*3*,*4*,*5*) *xs'* *zs-a a-l ab* **by** (*auto split*: *if-splits*)
    subgoal using *H*(*4*) *xs'* *zs-a a-l ab* **by** *auto*
    subgoal using *H*(*5*) *xs'* *a-l ab* **by** *auto*
    subgoal using *H*(*6*) *xs'* **by** *auto*
    subgoal using *H*(*7*) *xs'* **by** *auto*
    subgoal using *H*(*8*) **by** *auto*
    done
  then have ‹*vmtf-ns* (*b* # *l*) (*stamp* (*zs'* ! *b*)) *zs'*›
    by (*rule vmtf-ns-thighten-stamp*)
      (*use vmtf-ns-stamp-sorted*[*OF vtmf-b-l*] **in** ‹*auto simp*: *sorted-append*›)

  then show *?case*
    apply (*rule vmtf-ns.Cons*[*of* - - - - - ‹*st* ! *a*›])
    unfolding *zs'-def*
    subgoal using *a-le-y H*(*6*) *xs'* **by** *auto*
    subgoal using *a-le-y* **by** *auto*
    subgoal using *ab*.
    subgoal using *a-l* .
    subgoal using *nn'* *mn H*(*1*,*2*) **by** (*auto simp*: *sorted-many-eq-append*)
    subgoal using *zs-upd-zs* **by** *auto*
    subgoal using *H* **by** (*auto intro*!: *Max-ge nth-mem*)
    done
qed

lemma *vmtf-ns-last-prev*:
  assumes *vmtf*: ‹*vmtf-ns* (*xs* @ [*x*]) *m ns*›
  shows ‹*get-prev* (*ns* ! *x*) = *option-last xs*›
proof (*cases xs rule*: *rev-cases*)
  case *Nil*
  then show *?thesis* **using** *vmtf* **by** (*cases* ‹*ns*!*x*›) (*auto simp*: *vmtf-ns-single-iff*)
next
  case (*snoc xs'* *y'*)
  then show *?thesis*
    using *vmtf-ns-last-mid-get-prev*[*of xs'* *y'* *x* ‹[]› *m ns*] *vmtf* **by** *auto*
qed

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 −
  obtain *xs′ ys′* **where**
    vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
    fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
    lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
    next-search: ‹*next-search* = *option-hd xs′*› **and**
    abs-vmtf: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *remove*)› **and**
    notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
    atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
    ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
    using *vmtf* **unfolding** *vmtf-def* **by** *fast*
  moreover have ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ (*L* # *M*) ((*set xs′*, *set ys′*), *remove*)›
    using *abs-vmtf* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
  ultimately have ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns* ∧
      fst-As = *hd* (*ys′* @ *xs′*) ∧
      lst-As = *last* (*ys′* @ *xs′*) ∧
      next-search = *option-hd xs′* ∧
      vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*L* # *M*) ((*set xs′*, *set ys′*), *remove*) ∧
      vmtf-ns-notin (*ys′* @ *xs′*) *m ns* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧

($\forall$ L∈*set* (*ys'* @ *xs'*). $L \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show *?thesis*
unfolding *vmtf-def* **by** *fast*
qed

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* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
unfolding *vmtf-def*
proof −
have *ns'*: ‹*ns'* = *ns-vmtf-dequeue L ns*› **and**
fst-As': ‹*fst-As'* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*)› **and**
lst-As': ‹*lst-As'* = (*if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As*)› **and**
m'm: ‹*m'* = *m*› **and**
next-search-L-next:
‹*next-search'* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)›
using *dequeue* **unfolding** *vmtf-dequeue-def* **by** *auto*
obtain *xs ys* **where**
vmtf: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**

```
  next-search: ‹next-search = option-hd xs› and
  abs-inv: ‹vmtf-ℒall 𝒜 M ((set xs, set ys), to-remove)› and
  fst-As: ‹fst-As = hd (ys @ xs)› and
  lst-As: ‹lst-As = last (ys @ xs)› and
  atm-A: ‹∀L∈atms-of (ℒall 𝒜). L < length ns› and
  L-ys-xs: ‹∀L∈set (ys @ xs). L ∈ atms-of (ℒall 𝒜)›
  using vmtf unfolding vmtf-def by auto
have [dest]: ‹xs = [] ⟹ ys = [] ⟹ False›
  using abs-inv 𝒜in-nempty unfolding atms-of-ℒall-𝒜in vmtf-ℒall-def
  by auto
let ?ys = ‹ys›
let ?xs = ‹xs›
have dist: ‹distinct (xs @ ys)›
  using vmtf-ns-distinct[OF vmtf] by auto
have xs-ys: ‹remove1 L ys @ remove1 L xs = remove1 L (ys @ xs)›
  using dist by (auto simp: remove1-append remove1-idem disjoint-iff-not-equal
      intro!: remove1-idem)
have atm-L-A: ‹L < length ns›
  using atm-A L by blast

have ‹vmtf-ns (remove1 L ys @ remove1 L xs) m' ns'›
  using vmtf-ns-ns-vmtf-dequeue[OF vmtf notin, of L] dequeue dist atm-L-A
  unfolding vmtf-dequeue-def by (auto split: if-splits simp: xs-ys)
moreover have next-search': ‹next-search' = option-hd (remove1 L xs)›
proof −
  have ‹[hd xs, hd (tl xs)] @ tl (tl xs) = xs›
    if ‹xs ≠ []› ‹tl xs ≠ []›
    apply (cases xs; cases ‹tl xs›)
     using that by (auto simp: tl-append split: list.splits)
  then have [simp]: ‹get-next (ns ! hd xs) = option-hd (remove1 (hd xs) xs)› if ‹xs ≠ []›
    using vmtf-ns-last-mid-get-next[of ‹?ys› ‹hd ?xs›
        ‹hd (tl ?xs)› ‹tl (tl ?xs)› m ns] vmtf vmtf-ns-distinct[OF vmtf] that
      distinct-remove1-last-butlast[of xs]
    by (cases xs; cases ‹tl xs›)
      (auto simp: tl-append vmtf-ns-last-next split: list.splits elim: vmtf-nsE)
  have ‹xs ≠ [] ⟹ xs ≠ [L] ⟹  L ≠ hd xs ⟹ hd xs = hd (remove1 L xs)›
    by (induction xs) (auto simp: remove1-Nil-iff)
  then have [simp]: ‹option-hd xs = option-hd (remove1 L xs)› if ‹L ≠ hd xs›
    using that vmtf-ns-distinct[OF vmtf]
    by (auto simp: option-hd-def remove1-Nil-iff)
  show ?thesis
    using dequeue dist atm-L-A next-search next-search unfolding vmtf-dequeue-def
    by (auto split: if-splits simp: xs-ys dest: last-in-set)
  qed
moreover {
  have ‹[hd ys, hd (tl ys)] @ tl (tl ys) = ys›
    if ‹ys ≠ []› ‹tl ys ≠ []›
     using that by (auto simp: tl-append split: list.splits)
  then have ‹get-next (ns ! hd (ys @ xs)) = option-hd (remove1 (hd (ys @ xs)) (ys @ xs))›
    if ‹ys @ xs ≠ []›
    using vmtf-ns-last-next[of ‹?xs @ butlast ?ys› ‹last ?ys›] that
    using vmtf-ns-last-next[of ‹butlast ?xs› ‹last ?xs›]  vmtf dist
      distinct-remove1-last-butlast[of ‹?ys @ ?xs›]
    by (cases ys; cases ‹tl ys›)
     (auto simp: tl-append vmtf-ns-last-prev remove1-append hd-append remove1-Nil-iff
        split: list.splits if-splits elim: vmtf-nsE)
```

moreover have ‹*hd ys* ∉ *set xs*› **if** ‹*ys* ≠ []›
using *vmtf-ns-distinct*[*OF vmtf*] *that* **by** (*cases ys*) *auto*
ultimately have ‹*fst-As'* = *option-hd* (*remove1 L* (*ys* @ *xs*))›
using *dequeue dist atm-L-A fst-As vmtf-ns-distinct*[*OF vmtf*] *vmtf*
unfolding *vmtf-dequeue-def*
apply (*cases ys*)
subgoal by (*cases xs*) (*auto simp*: *remove1-append option-hd-def remove1-Nil-iff split*: *if-splits*)
subgoal by (*auto simp*: *remove1-append option-hd-def remove1-Nil-iff*)
done
}
moreover have ‹*lst-As'* = *option-last* (*remove1 L* (*ys* @ *xs*))›
apply (*cases* ‹*ys* @ *xs*› *rule*: *rev-cases*)
using *lst-As vmtf-ns-distinct*[*OF vmtf*] *vmtf-ns-last-prev vmtf*
by (*auto simp*: *lst-As' remove1-append simp del*: *distinct-append*) *auto*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set* (*remove1 L xs*)), *set* (*remove1 L ys*)),
to-remove)›
using *abs-inv L dist*
unfolding *vmtf*-$\mathcal{L}_{all}$-*def* **by** (*auto dest*: *in-set-remove1D*)
moreover have ‹*vmtf-ns-notin* (*remove1 L ?ys* @ *remove1 L ?xs*) *m' ns'*›
unfolding *xs-ys ns'*
apply (*rule vmtf-ns-notin-dequeue*)
subgoal using *vmtf* **unfolding** *m'm* **.**
subgoal using *notin* **unfolding** *m'm* **.**
subgoal using *atm-L-A* **.**
done
moreover have ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns'*›
using *atm-A* **unfolding** *ns'* **by** *auto*
moreover have ‹*L* ∉ *set* (*remove1 L ys* @ *remove1 L xs*)›
using *dist* **by** *auto*
moreover have ‹∀ *L*∈*set* (*remove1 L* (*ys* @ *xs*)). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L-ys-xs* **by** (*auto dest*: *in-set-remove1D*)
ultimately show *?thesis*
using *next-search-L-next*
apply −
apply (*rule exI*[*of* - ‹*remove1 L xs*›])
apply (*rule exI*[*of* - ‹*remove1 L ys*›])
unfolding *xs-ys*
by *blast*
qed

lemma *vmtf-ns-get-prev-not-itself*:
‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-prev* (*ns* ! *L*) ≠ *Some L*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by *auto*
subgoal by (*auto simp*: *vmtf-ns-single-iff*)
subgoal by *auto*
done

lemma *vmtf-ns-get-next-not-itself*:
‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-next* (*ns* ! *L*) ≠ *Some L*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by *auto*
subgoal by (*auto simp*: *vmtf-ns-single-iff*)
subgoal by *auto*
done

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*›
  unfolding *vmtf-def*
proof *clarify*
  fix *xxs yys zzs ns*′ *m*′ *fst-As*′ *lst-As*′ *next-search*′
  assume *dequeue*: ‹(*ns*′, *m*′, *fst-As*′, *lst-As*′, *next-search*′) =
    vmtf-en-dequeue M L (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›
  obtain *xs ys* **where**
    vmtf-ns: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
    notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**
    next-search: ‹*next-search* = *option-hd xs*› **and**
    abs-inv: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs*, *set ys*), *to-remove*)› **and**
    fst-As: ‹*fst-As* = *hd* (*ys* @ *xs*)› **and**
    lst-As: ‹*lst-As* = *last* (*ys* @ *xs*)› **and**
    atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
    ys-xs-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys* @ *xs*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
    using *assms* **unfolding** *vmtf-def* **by** *auto*
  have *atm-L-A*: ‹*L* < *length ns*›
    using *atm-A L* **by** *blast*

d stands for dequeue

  obtain *nsd md fst-Asd lst-Asd next-searchd* **where**
    de: ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) = (*nsd*, *md*, *fst-Asd*, *lst-Asd*, *next-searchd*) ›
    by (*cases* ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›)
  obtain *xs*′ *ys*′ **where**
    vmtf-ns′: ‹*vmtf-ns* (*ys*′ @ *xs*′) *md nsd*› **and**
    fst-Asd: ‹*fst-Asd* = *option-hd* (*ys*′ @ *xs*′)› **and**
    lst-Asd: ‹*lst-Asd* = *option-last* (*ys*′ @ *xs*′)› **and**
    next-searchd-hd: ‹*next-searchd* = *option-hd xs*′› **and**
    next-searchd-L-next:
      ‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)› **and**
    abs-l: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set xs*′), *set ys*′), *to-remove*)› **and**
    not-in: ‹*vmtf-ns-notin* (*ys*′ @ *xs*′) *md nsd*› **and**
    L-xs′*-ys*′: ‹*L* ∉ *set* (*ys*′ @ *xs*′)› **and**
    L-xs′*-ys*′-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys*′ @ *xs*′). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
    using *abs-vmtf-ns-bump-vmtf-dequeue*[*OF vmtf L de*[*symmetric*] *nempty*] **by** *blast*

  have [*simp*]: ‹*length ns*′ = *length ns*› ‹*length nsd* = *length ns*›
    using *dequeue de* **unfolding** *vmtf-en-dequeue-def comp-def vmtf-dequeue-def*
    by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
  have *nsd*: ‹*nsd* = *ns-vmtf-dequeue L ns*›
    using *de* **unfolding** *vmtf-dequeue-def* **by** *auto*
  have [*simp*]: ‹*fst-As* = *L*› **if** ‹*ys*′ = []› **and** ‹*xs*′ = []›
    proof −
      have *1*: ‹*set-mset* $\mathcal{A}$ = {*L*}›
        using *abs-l* **unfolding** *that vmtf*-$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
      show *?thesis*
        using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs*-$\mathcal{L}_{all}$ *abs-inv*
        unfolding *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *1 fst-As vmtf*-$\mathcal{L}_{all}$*-def*
        by (*cases* ‹*ys* @ *xs*›) *auto*

qed
have *fst-As'*: ‹*fst-As'* = *L*› **and** *m'*: ‹*m'* = *md* + *1*› **and**
lst-As': ‹*fst-Asd* ≠ *None* ⟶ *lst-As'* = *the* (*lst-Asd*)›
‹*fst-Asd* = *None* ⟶ *lst-As'* = *L*›
using *dequeue* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have ‹*lst-As* = *L*› **if** ‹*ys'* = []› **and** ‹*xs'* = []›
proof −
have *1*: ‹*set-mset* $\mathcal{A}$ = {*L*}›
using *abs-l* **unfolding** *that vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
then have ‹*set* (*ys* @ *xs*) = {*L*} ›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv*
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 fst-As vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
then have ‹*ys* @ *xs* = [*L*]›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv vmtf-*$\mathcal{L}_{all}$*-def*
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 fst-As*
by (*cases* ‹*ys* @ *xs*› *rule*: *rev-cases*) (*auto simp del*: *set-append distinct-append*
simp: *set-append*[*symmetric*], *auto*)
then show *?thesis*
using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv vmtf-*$\mathcal{L}_{all}$*-def*
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 lst-As*
by (*auto simp del*: *set-append distinct-append simp*: *set-append*[*symmetric*])
qed
then have [*simp*]: ‹*lst-As'* = *L*› **if** ‹*ys'* = []› **and** ‹*xs'* = []›
using *lst-As' fst-Asd* **unfolding** *that* **by** *auto*
have [*simp*]: ‹*lst-As'* = *last* (*ys'* @ *xs'*)› **if** ‹*ys'* ≠ [] ∨ *xs'* ≠ []›
using *lst-As' fst-Asd that lst-Asd* **by** *auto*

have ‹*get-prev* (*nsd* ! *i*) ≠ *Some L*› (**is** *?prev*) **and**
‹*get-next* (*nsd* ! *i*) ≠ *Some L*› (**is** *?next*)
if
i-le-A: ‹*i* < *length ns*› **and**
i-L: ‹*i* ≠ *L*› **and**
i-ys': ‹*i* ∉ *set ys'*› **and**
i-xs': ‹*i* ∉ *set xs'*›
for *i*
proof −
have ‹*i* ∉ *set xs*› ‹*i* ∉ *set ys*› **and** *L-xs-ys*: ‹*L* ∈ *set xs* ∨ *L* ∈ *set ys*›
using *i-ys' i-xs' abs-l abs-inv i-L* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
then have
‹*get-next* (*ns* ! *i*) = *None*›
‹*get-prev* (*ns* ! *i*) = *None*›
using *notin i-le-A* **unfolding** *nsd vmtf-ns-notin-def ns-vmtf-dequeue-def*
by (*auto simp*: *Let-def split*: *option.splits*)
moreover have ‹*get-prev* (*ns* ! *L*) ≠ *Some L*› **and** ‹*get-next* (*ns* ! *L*) ≠ *Some L*›
using *vmtf-ns-get-prev-not-itself*[*OF vmtf-ns*, *of L*] *L-xs-ys atm-L-A*
vmtf-ns-get-next-not-itself[*OF vmtf-ns*, *of L*] **by** *auto*
ultimately show *?next* **and** *?prev*
using *i-le-A L-xs-ys* **unfolding** *nsd ns-vmtf-dequeue-def vmtf-ns-notin-def*
by (*auto simp*: *Let-def split*: *option.splits*)
qed
then have *vmtf-ns-notin'*: ‹*vmtf-ns-notin* (*L* # *ys'* @ *xs'*) *m' ns'*›
using *not-in dequeue fst-Asd* **unfolding** *vmtf-en-dequeue-def comp-def de vmtf-ns-notin-def*

ns-vmtf-dequeue-def
by (*auto simp add*: *vmtf-enqueue-def hd-append split*: *option.splits if-splits*)

consider
(*defined*) ‹*defined-lit M* (*Pos L*)› |
(*undef*) ‹*undefined-lit M* (*Pos L*)›
by *blast*
then show ‹$\exists$ *xs′ ys′*.
vmtf-ns (*ys′* @ *xs′*) *m′ ns′* $\wedge$
fst-As′ = *hd* (*ys′* @ *xs′*) $\wedge$
lst-As′ = *last* (*ys′* @ *xs′*) $\wedge$
next-search′ = *option-hd xs′* $\wedge$
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove′*) $\wedge$
vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* $\wedge$
($\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) $\wedge$
($\forall$ *L*$\in$*set* (*ys′* @ *xs′*). *L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof *cases*
case *defined*
have *L-in-M*: ‹*L* $\in$ *atm-of* ‘ *lits-of-l M*›
using *defined* **by** (*auto simp*: *defined-lit-map lits-of-def*)
have *next-search′*: ‹*fst-Asd* $\neq$ *None* $\longrightarrow$ *next-search′* = *next-searchd*›
‹*fst-Asd* = *None* $\longrightarrow$ *next-search′* = *None*›
using *dequeue defined* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have *next-searchd*:
‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*)›
using *de* **by** (*auto simp*: *vmtf-dequeue-def*)
have *abs′*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *insert L* (*set ys′*)), *to-remove′*)›
using *abs-l to-remove L-in-M L-xs′-ys′* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto 5 5 dest*: *in-diffD*)

have *vmtf-ns*: ‹*vmtf-ns* (*L* # (*ys′* @ *xs′*)) *m′ ns′*›
proof (*cases* ‹*ys′* @ *xs′*›)
case *Nil*
then have ‹*fst-Asd* = *None*›
using *fst-Asd* **by** *auto*
then show *?thesis*
using *atm-L-A dequeue Nil* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
next
case (*Cons z zs*)
let *?m* = ‹(*stamp* (*nsd*!*z*))›
let *?Ad* = ‹*nsd*[*L* := *VMTF-Node m′ None* (*Some z*)]›
have *L-z-zs*: ‹*L* $\notin$ *set* (*z* # *zs*)›
using *L-xs′-ys′ atm-L-A* **unfolding** *Cons*
by *simp*
have *vmtf-ns-z*: ‹*vmtf-ns* (*z* # *zs*) *md nsd*›
using *vmtf-ns′* **unfolding** *Cons* **.**

have *vmtf-ns-A*: ‹*vmtf-ns* (*z* # *zs*) *md ?Ad*›
apply (*rule vmtf-ns-eq-iffI*[*of - - nsd*])
subgoal using *L-z-zs atm-L-A* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-z*] **by** *auto*
subgoal using *vmtf-ns-z* **.**
done
have [*simp*]: ‹*fst-Asd* = *Some z*›

```
        using fst-Asd unfolding Cons by simp
      show ?thesis
        unfolding Cons
        apply (rule vmtf-ns.Cons[of - - md ?Ad - m'])
        subgoal using vmtf-ns-A .
        subgoal using atm-L-A by simp
        subgoal using atm-L-A by simp
        subgoal using L-z-zs by simp
        subgoal using L-z-zs by simp
        subgoal using m' by simp-all
        subgoal
          using atm-L-A dequeue L-z-zs unfolding Nil vmtf-en-dequeue-def comp-def de nsd
          apply (cases ‹ns-vmtf-dequeue z ns ! z›)
          by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
        subgoal by linarith
        done
    qed
    have L-xs'-ys'-ℒall': ‹∀L'∈set ((L # ys') @ xs'). L' ∈ atms-of (ℒall 𝒜)›
      using L L-xs'-ys'-ℒall by auto
    have next-search'-xs': ‹next-search' = option-hd xs'›
      using next-searchd-L-next next-search' next-searchd-hd lst-As' fst-Asd
      by (auto split: if-splits)
    show ?thesis
      apply (rule exI[of - ‹xs'›])
      apply (rule exI[of - ‹L # ys'›])
      using fst-As' next-search' abs' atm-A vmtf-ns-notin' vmtf-ns ys-xs-ℒall L-xs'-ys'-ℒall'
        next-searchd next-search'-xs'
      by simp
  next
    case undef
    have next-search': ‹next-search' = Some L›
      using dequeue undef unfolding vmtf-en-dequeue-def comp-def de
      by (auto simp add: vmtf-enqueue-def split: option.splits)
    have next-searchd:
     ‹next-searchd = (if next-search = Some L then get-next (ns ! L) else next-search)›
      using de by (auto simp: vmtf-dequeue-def)
    have abs': ‹vmtf-ℒall 𝒜 M ((insert L (set (ys' @ xs')), set []), to-remove')›
      using abs-l to-remove L-xs'-ys' unfolding vmtf-ℒall-def
      by (auto 5 5 dest: in-diffD)

    have vmtf-ns: ‹vmtf-ns (L # (ys' @ xs')) m' ns'›
    proof (cases ‹ys' @ xs'›)
      case Nil
      then have ‹fst-Asd = None›
        using fst-Asd by auto
      then show ?thesis
        using atm-L-A dequeue Nil unfolding Nil vmtf-en-dequeue-def comp-def de nsd
        by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
    next
      case (Cons z zs)
      let ?m = ‹(stamp (nsd!z))›
      let ?Ad = ‹nsd[L := VMTF-Node m' None (Some z)]›
      have L-z-zs: ‹L ∉ set (z # zs)›
        using L-xs'-ys' atm-L-A unfolding Cons
        by simp
      have vmtf-ns-z: ‹vmtf-ns (z # zs) md nsd›
```

using *vmtf-ns′* **unfolding** *Cons* **.**

have *vmtf-ns-A*: ‹*vmtf-ns* (*z* # *zs*) *md ?Ad*›
apply (*rule vmtf-ns-eq-iffI*[*of* - - *nsd*])
subgoal using *L-z-zs atm-L-A* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-z*] **by** *auto*
subgoal using *vmtf-ns-z* **.**
done
have [*simp*]: ‹*fst-Asd* = *Some z*›
using *fst-Asd* **unfolding** *Cons* **by** *simp*
show *?thesis*
unfolding *Cons*
apply (*rule vmtf-ns.Cons*[*of* - - *md ?Ad* - *m′*])
subgoal using *vmtf-ns-A* **.**
subgoal using *atm-L-A* **by** *simp*
subgoal using *atm-L-A* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *m′* **by** *simp-all*
subgoal
using *atm-L-A dequeue L-z-zs* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
apply (*cases* ‹*ns-vmtf-dequeue z ns* ! *z*›)
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
subgoal by *linarith*
done
qed
have *L-xs′-ys′-*$\mathcal{L}_{all}$*′*: ‹∀ *L′*∈*set* ((*L* # *ys′*) @ *xs′*). *L′* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L L-xs′-ys′-*$\mathcal{L}_{all}$ **by** *auto*
show *?thesis*
apply (*rule exI*[*of* - ‹(*L* # *ys′*) @ *xs′*›])
apply (*rule exI*[*of* - ‹[]›])
using *fst-As′ next-search′ abs′ atm-A vmtf-ns-notin′ vmtf-ns ys-xs-*$\mathcal{L}_{all}$ *L-xs′-ys′-*$\mathcal{L}_{all}$*′*
next-searchd
by *simp*
qed
qed

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*›
using *abs-vmtf-ns-bump-vmtf-en-dequeue assms* **by** (*cases vm*) *blast*

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 −
let *?xs'* = ‹*set xs'*›
let *?ys'* = ‹*set ys'*›
have *L-xs-ys*: ‹*L* ∈ *?xs'* ∪ *?ys'*›
using *abs-vmtf L* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
have *dist*: ‹*distinct* (*xs'* @ *ys'*)›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*

have *sorted*: ‹*sorted* (*map* (λ*a*. *stamp* (*ns* ! *a*)) (*rev xs'* @ *rev ys'*))› **and**
distinct: ‹*distinct* (*map* (λ*a*. *stamp* (*ns* ! *a*)) (*xs'* @ *ys'*))›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *vmtf-ns-stamp-distinct*[*OF vmtf-ns*]
by (*auto simp*: *rev-map*[*symmetric*])
have *next-search-xs*: ‹*?xs'* = {} ⟷ *next-search* = *None*›
using *next-search* **by** *auto*

have ‹*stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*) ⟹ *L* ∉ *?xs'*›
if ‹*xs'* ≠ []›
using *that sorted distinct L-xs-ys* **unfolding** *next-search*
by (*cases xs'*) (*auto simp*: *sorted-append*)
moreover have ‹*stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)› (**is** ‹*?n* < *?L*›)
if *xs'*: ‹*xs'* ≠ []› **and** ‹*L* ∈ *?ys'*›
proof −
have ‹*?n* ≤ *?L*›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *that last-in-set*[*OF xs'*]
by (*cases xs'*)
(*auto simp*: *rev-map*[*symmetric*] *next-search sorted-append sorted2*)
moreover have ‹*?n* ≠ *?L*›
using *vmtf-ns-stamp-distinct*[*OF vmtf-ns*] *that last-in-set*[*OF xs'*]
by (*cases xs'*) (*auto simp*: *rev-map*[*symmetric*] *next-search*)
ultimately show *?thesis*
by *arith*
qed
ultimately show *?thesis*
using *L-xs-ys next-search-xs dist* **by** *auto*
qed

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'*)›
using *assms* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *mset-subset-eqD*)

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 −
obtain *xs' ys'* **where**

vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **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**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
L-ys'-xs'-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
obtain *ns' m' fst-As' next-search' to-remove'' lst-As'* **where**
S: ‹*?S* = ((*ns'*, *m'*, *fst-As'*, *lst-As'*, *next-search'*), *to-remove''*)›
by (*cases ?S*) *auto*
have *L-ys'-iff*: ‹*L* ∈ *set ys'* ⟷ (*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
using *vmtf-atm-of-ys-iff*[*OF vmtf-ns next-search abs-vmtf L-N*] **.**
have ‹*L* ∈ *set* (*xs'* @ *ys'*)›
using *abs-vmtf L-N* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
then have *L-ys'-xs'*: ‹*L* ∈ *set ys'* ⟷ *L* ∉ *set xs'*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*
have ‹∃ *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* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof (*cases* ‹*L* ∈ *set xs'*›)
case *True*
then have *C*: ‹¬(*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
by (*subst L-ys'-iff*[*symmetric*]) (*use L-ys'-xs'* **in** *auto*)
have *abs-vmtf*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove''*)›
apply (*rule vmtf-*$\mathcal{L}_{all}$*-to-remove-mono*)
apply (*rule abs-vmtf*)
using *to-remove S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)
show *?thesis*
using *S True* **unfolding** *vmtf-unset-def L-ys'-xs'*[*symmetric*]
apply −
apply (*simp add*: *C*)
using *vmtf-ns fst-As next-search abs-vmtf notin atm-A to-remove L-ys'-xs'-*$\mathcal{L}_{all}$ *lst-As*
by *auto*
next
case *False*
then have *C*: ‹*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*)›
by (*subst L-ys'-iff*[*symmetric*]) (*use L-ys'-xs'* **in** *auto*)
have *L-ys*: ‹*L* ∈ *set ys'*›
by (*use False L-ys'-xs'* **in** *auto*)
define *y-ys* **where** ‹*y-ys* ≡ *takeWhile* ((≠) *L*) *ys'*›
define *x-ys* **where** ‹*x-ys* ≡ *drop* (*length y-ys*) *ys'*›
let *?ys'* = ‹*y-ys*›
let *?xs'* = ‹*x-ys* @ *xs'*›
have *x-ys-take-ys'*: ‹*y-ys* = *take* (*length y-ys*) *ys'*›
unfolding *y-ys-def*
by (*subst take-length-takeWhile-eq-takeWhile*[*of* ‹(≠) *L*› ‹*ys'*›, *symmetric*]) *standard*
have *ys'-y-x*: ‹*ys'* = *y-ys* @ *x-ys*›
by (*subst x-ys-take-ys'*) (*auto simp*: *x-ys-def*)

have *y-ys-le-ys'*: ‹*length y-ys* < *length ys'*›
using *L-ys* **by** (*metis* (*full-types*) *append-eq-conv-conj append-self-conv le-antisym length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys' y-ys-def*)
from *nth-length-takeWhile*[*OF this*[*unfolded y-ys-def*]] **have** [*simp*]: ‹*x-ys* ≠ []› ‹*hd x-ys* = *L*›
using *y-ys-le-ys'* **unfolding** *x-ys-def y-ys-def*
by (*auto simp*: *x-ys-def y-ys-def hd-drop-conv-nth*)
have [*simp*]: ‹*ns'* = *ns*› ‹*m'* = *m*› ‹*fst-As'* = *fst-As*› ‹*next-search'* = *Some L*› ‹*to-remove''* = *to-remove'*› ‹*lst-As'* = *lst-As*›
using *S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)

have ‹*vmtf-ns* (*?ys'* @ *?xs'*) *m ns*›
using *vmtf-ns* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*fst-As'* = *hd* (*?ys'* @ *?xs'*)›
using *fst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*lst-As'* = *last* (*?ys'* @ *?xs'*)›
using *lst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*next-search'* = *option-hd ?xs'*›
by *auto*
moreover {
have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *to-remove*)›
using *abs-vmtf vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def ys'-y-x*
by *auto*
then have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *to-remove'*)›
by (*rule vmtf-*$\mathcal{L}_{all}$*-to-remove-mono*) (*use to-remove* **in** *auto*)
}
moreover have ‹*vmtf-ns-notin* (*?ys'* @ *?xs'*) *m ns*›
using *notin* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹∀ *L*∈*set* (*?ys'* @ *?xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L-ys'-xs'-*$\mathcal{L}_{all}$ **unfolding** *ys'-y-x* **by** *auto*
ultimately show *?thesis*
using *S False atm-A* **unfolding** *vmtf-unset-def L-ys'-xs'*[*symmetric*]
by (*fastforce simp add*: *C*)
qed
then show *?thesis*
unfolding *vmtf-def S*
by *fast*
qed

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*))›
apply (*intro ext*, *rename-tac x*)
subgoal for *x*
by (*cases* ‹*get-next* ((*snd x*)!(*fst x*))›; *cases* ‹*get-prev* ((*snd x*)!(*fst x*))›)
(*auto simp*: *vmtf-dequeue-pre-def intro*!: *ext*)
done

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*)).

$L < length\ ns \wedge vmtf\text{-}dequeue\text{-}pre\ (L,\ ns) \wedge$
$fst\text{-}As < length\ ns \wedge (get\text{-}next\ (ns\ !\ fst\text{-}As) \neq None \longrightarrow get\text{-}prev\ (ns\ !\ lst\text{-}As) \neq None) \wedge$
$(get\text{-}next\ (ns\ !\ fst\text{-}As) = None \longrightarrow fst\text{-}As = lst\text{-}As) \wedge$
$m+1 \leq uint64\text{-}max \wedge$
$Pos\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A})$›

lemma (**in** −) *id-reorder-list*:
‹$(RETURN\ o\ id,\ reorder\text{-}list\ vm) \in \langle nat\text{-}rel\rangle list\text{-}rel \rightarrow_f \langle\langle nat\text{-}rel\rangle list\text{-}rel\rangle nres\text{-}rel$›
unfolding *reorder-list-def* **by** (*intro frefI nres-relI*) *auto*

lemma *vmtf-vmtf-en-dequeue-pre-to-remove*:
assumes *vmtf*: ‹$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove) \in vmtf\ \mathcal{A}\ M$› **and**
i: ‹$A \in to\text{-}remove$› **and**
m-le: ‹$m + 1 \leq uint64\text{-}max$› **and**
nempty: ‹$isasat\text{-}input\text{-}nempty\ \mathcal{A}$›
shows ‹$vmtf\text{-}en\text{-}dequeue\text{-}pre\ \mathcal{A}\ ((M,\ A),\ (ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search))$›
proof −
obtain xs' ys' **where**
vmtf-ns: ‹$vmtf\text{-}ns\ (ys'\ @\ xs')\ m\ ns$› **and**
fst-As: ‹$fst\text{-}As = hd\ (ys'\ @\ xs')$› **and**
lst-As: ‹$lst\text{-}As = last\ (ys'\ @\ xs')$› **and**
next-search: ‹$next\text{-}search = option\text{-}hd\ xs'$› **and**
abs-vmtf: ‹$vmtf\text{-}\mathcal{L}_{all}\ \mathcal{A}\ M\ ((set\ xs',\ set\ ys'),\ to\text{-}remove)$› **and**
notin: ‹$vmtf\text{-}ns\text{-}notin\ (ys'\ @\ xs')\ m\ ns$› **and**
atm-A: ‹$\forall L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}).\ L < length\ ns$› **and**
$L\text{-}ys'\text{-}xs'\text{-}\mathcal{L}_{all}$: ‹$\forall L \in set\ (ys'\ @\ xs').\ L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})$›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
have [*dest*]: *False* **if** ‹$ys' = []$› **and** ‹$xs' = []$›
proof −
have *1*: ‹$set\text{-}mset\ \mathcal{A} = \{\}$›
using *abs-vmtf* **unfolding** *that* $vmtf\text{-}\mathcal{L}_{all}\text{-}def$ **by** (*auto simp*: $atms\text{-}of\text{-}\mathcal{L}_{all}\text{-}\mathcal{A}_{in}$)
then show *?thesis*
using *nempty* **by** *auto*
qed

have ‹$A \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})$›
using *abs-vmtf i* **unfolding** $vmtf\text{-}\mathcal{L}_{all}\text{-}def$ **by** *auto*
then have *remove-i-le-A*: ‹$A < length\ ns$› **and**
i-L: ‹$Pos\ A \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›
using *atm-A* **by** (*auto simp*: $in\text{-}\mathcal{L}_{all}\text{-}atm\text{-}of\text{-}\mathcal{A}_{in}$ *atms-of-def*)
moreover have ‹$fst\text{-}As < length\ ns$›
using *fst-As atm-A* $L\text{-}ys'\text{-}xs'\text{-}\mathcal{L}_{all}$ **by** (*cases* ys'; *cases* xs') *auto*
moreover have ‹$get\text{-}prev\ (ns\ !\ lst\text{-}As) \neq None$› **if** ‹$get\text{-}next\ (ns\ !\ fst\text{-}As) \neq None$›
using *that vmtf-ns-hd-next*[*of* ‹$hd\ (ys'\ @\ xs')$› ‹$hd\ (tl\ (ys'\ @\ xs'))$› ‹$tl\ (tl\ (ys'\ @\ xs'))$›]
vmtf-ns vmtf-ns-last-prev[*of* ‹$butlast\ (ys'\ @\ xs')$› ‹$last\ (ys'\ @\ xs')$›]
vmtf-ns-last-next[*of* ‹$butlast\ (ys'\ @\ xs')$› ‹$last\ (ys'\ @\ xs')$›]
by (*cases* ‹$ys'\ @\ xs'$›; *cases* ‹$tl\ (ys'\ @\ xs')$›)
(*auto simp*: *fst-As lst-As*)
moreover have ‹$vmtf\text{-}dequeue\text{-}pre\ (A,\ ns)$›
proof −
have ‹$A < length\ ns$›
using *i abs-vmtf atm-A* **unfolding** $vmtf\text{-}\mathcal{L}_{all}\text{-}def$ **by** *auto*
moreover have ‹$y < length\ ns$› **if** *get-next*: ‹$get\text{-}next\ (ns\ !\ (A)) = Some\ y$› **for** *y*
proof (*cases* ‹$A \in set\ (ys'\ @\ xs')$›)
case *False*
then show *?thesis*

      using *notin get-next remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
    next
      case *True*
      then obtain *zs zs′* **where** *zs*: ‹*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*›
        using *split-list* **by** *fastforce*
      moreover have ‹*set* (*ys′* @ *xs′*) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
        using *abs-vmtf* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
      ultimately show *?thesis*
        using *vmtf-ns-last-mid-get-next-option-hd*[*of zs′ A zs m ns*] *vmtf-ns atm-A get-next*
          L-ys′-xs′-$\mathcal{L}_{all}$ **unfolding** *zs* **by** *force*
    qed
    moreover have ‹*y* < *length ns*› **if** *get-prev*: ‹*get-prev* (*ns* ! (*A*)) = *Some y*› **for** *y*
    proof (*cases* ‹*A* ∈ *set* (*ys′* @ *xs′*)›)
      case *False*
      then show *?thesis*
        using *notin get-prev remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
    next
      case *True*
      then obtain *zs zs′* **where** *zs*: ‹*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*›
        using *split-list* **by** *fastforce*
      moreover have ‹*set* (*ys′* @ *xs′*) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
        using *abs-vmtf* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
      ultimately show *?thesis*
        using *vmtf-ns-last-mid-get-prev-option-last*[*of zs′ A zs m ns*] *vmtf-ns atm-A get-prev*
          L-ys′-xs′-$\mathcal{L}_{all}$ **unfolding** *zs* **by** *force*
    qed
    ultimately show *?thesis*
      unfolding *vmtf-dequeue-pre-def* **by** *auto*
  qed
  moreover have ‹*get-next* (*ns* ! *fst-As*) = *None* ⟶ *fst-As* = *lst-As*›
    using *vmtf-ns-hd-next*[*of* ‹*hd* (*ys′* @ *xs′*)› ‹*hd* (*tl* (*ys′* @ *xs′*))› ‹*tl* (*tl* (*ys′* @ *xs′*))›]
      vmtf-ns vmtf-ns-last-prev[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
      vmtf-ns-last-next[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
    by (*cases* ‹*ys′* @ *xs′*›; *cases* ‹*tl* (*ys′* @ *xs′*)›)
       (*auto simp*: *fst-As lst-As*)
  ultimately show *?thesis*
    using *m-le* **unfolding** *vmtf-en-dequeue-pre-def* **by** *auto*
qed

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*)›
  using *vmtf-vmtf-en-dequeue-pre-to-remove assms*
  by (*cases vm*) *auto*

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 (*rule ccontr*)
  assume ‹¬ *?thesis*›
  then obtain *f* **where**
    f: ‹(*f* (*Suc i*), *f i*) ∈ *?R*› **for** *i*
    unfolding *wf-iff-no-infinite-down-chain* **by** *blast*

obtain xs' ys' **where**
  vmtf-ns: ‹*vmtf-ns* (ys' @ xs') *m ns*› **and**
  fst-As: ‹*fst-As* = *hd* (ys' @ xs')› **and**
  lst-As: ‹*lst-As* = *last* (ys' @ xs')› **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**
  notin: ‹*vmtf-ns-notin* (ys' @ xs') *m ns*› **and**
  atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
  using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?f0* = ‹*the* (*f 0*)›
have *f-None*: ‹*f i* ≠ *None*› **for** *i*
  using *f*[*of i*] **by** *fast*
have *f-Suc* : ‹*f* (*Suc n*) = *get-next* (*ns* ! *the* (*f n*))› **for** *n*
  using *f*[*of n*] **by** *auto*
have *f0-length*: ‹*?f0* < *length ns*›
  using *f*[*of 0*] *atm-A*
  by *auto*
have ‹*?f0* ∈ *set* (ys' @ xs')›
  apply (*rule ccontr*)
  using *notin f-Suc*[*of 0*] *f0-length* **unfolding** *vmtf-ns-notin-def*
  by (*auto simp*: *f-None*)
then obtain *i0* **where**
  i0: ‹(ys' @ xs') ! *i0* = *?f0*› ‹*i0* < *length* (ys' @ xs')›
  by (*meson in-set-conv-nth*)
define *zs* **where** ‹*zs* = ys' @ xs'›
have *H*: ‹ys' @ xs' = *take m* (ys' @ xs') @ [(ys' @ xs') ! *m*, (ys' @ xs') ! (*m*+*1*)] @
   drop (*m*+*2*) (ys' @ xs')›
  if ‹*m*+*1* < *length* (ys' @ xs')›
  for *m*
  using *that*
  unfolding *zs-def*[*symmetric*]
  apply −
  apply (*subst id-take-nth-drop*[*of m*])
  by (*auto simp*: *Cons-nth-drop-Suc simp del*: *append-take-drop-id*)

have ‹*the* (*f n*) = (ys' @ xs') ! (*i0* + *n*) ∧ *i0* + *n* < *length* (ys' @ xs')› **for** *n*
proof (*induction n*)
  case *0*
  then show *?case* **using** *i0* **by** *simp*
next
  case (*Suc* n')
  have *i0-le*: ‹*i0* + n' + *1* < *length* (ys' @ xs')›
  proof (*rule ccontr*)
    assume ‹¬ *?thesis*›
    then have ‹*i0* + n' + *1* = *length* (ys' @ xs')›
      using *Suc* **by** *auto*
    then have ‹ys' @ xs' = *butlast* (ys' @ xs') @ [*the* (*f* n')]›
      using *Suc* **by** (*metis add-diff-cancel-right' append-butlast-last-id length-0-conv*
        length-butlast less-one not-add-less2 nth-append-length)
    then show *False*
      using *vmtf-ns-last-next*[*of* ‹*butlast* (ys' @ xs')› ‹*the* (*f* n')› *m ns*] *vmtf-ns*
       f-Suc[*of* n'] **by** (*auto simp*: *f-None*)
  qed
  have *get-next*: ‹*get-next* (*ns* ! ((ys' @ xs') ! (*i0* + n'))) = *Some* ((ys' @ xs') ! (*i0* + n' + *1*))›
    apply(*rule vmtf-ns-last-mid-get-next*[*of* ‹*take* (*i0* + n') (ys' @ xs')›
      ‹(ys' @ xs') ! (*i0* + n')›

```
      ‹(ys' @ xs') ! ((i0 + n') + 1)›
      ‹drop ((i0 + n') + 2) (ys' @ xs')›
      m ns])
    apply (subst H[symmetric])
    subgoal using i0-le .
    subgoal using vmtf-ns by simp
    done
  then show ?case
    using f-Suc[of n'] Suc i0-le by auto
 qed
 then show False
  by blast
qed

lemma vmtf-next-search-take-next:
 assumes
  vmtf: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf 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 𝒜 M›
 unfolding vmtf-def
proof clarify
 obtain xs' ys' where
  vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
  fst-As: ‹fst-As = hd (ys' @ xs')› and
  lst-As: ‹lst-As = last (ys' @ xs')› and
  next-search: ‹next-search = option-hd xs'› and
  abs-vmtf: ‹vmtf-ℒall 𝒜 M ((set xs', set ys'), to-remove)› and
  notin: ‹vmtf-ns-notin (ys' @ xs') m ns› and
  atm-A: ‹∀L∈atms-of (ℒall 𝒜). L < length ns› and
  ys'-xs'-ℒall: ‹∀L∈set (ys' @ xs'). L ∈ atms-of (ℒall 𝒜)›
  using vmtf unfolding vmtf-def by fast
 let ?xs' = ‹tl xs'›
 let ?ys' = ‹ys' @ [hd xs']›
 have [simp]: ‹xs' ≠ []›
  using next-search n by auto
 have ‹vmtf-ns (?ys' @ ?xs') m ns›
  using vmtf-ns by (cases xs') auto
 moreover have ‹fst-As = hd (?ys' @ ?xs')›
  using fst-As by auto
 moreover have ‹lst-As = last (?ys' @ ?xs')›
  using lst-As by auto
 moreover have ‹get-next (ns ! the next-search) = option-hd ?xs'›
  using next-search n vmtf-ns
  by (cases xs') (auto dest: vmtf-ns-last-mid-get-next-option-hd)
 moreover {
  have [dest]: ‹defined-lit M (Pos a) ⟹ a ∈ atm-of ‘ lits-of-l M› for a
   by (auto simp: defined-lit-map lits-of-def)
  have ‹vmtf-ℒall 𝒜 M ((set ?xs', set ?ys'), to-remove)›
   using abs-vmtf def-n next-search n vmtf-ns-distinct[OF vmtf-ns]
   unfolding vmtf-ℒall-def
   by (cases xs') auto }
 moreover have ‹vmtf-ns-notin (?ys' @ ?xs') m ns›
  using notin by auto
 moreover have ‹∀L∈set (?ys' @ ?xs'). L ∈ atms-of (ℒall 𝒜)›
  using ys'-xs'-ℒall by auto
```

ultimately show ‹$\exists$ *xs′ ys′*. *vmtf-ns* (*ys′* @ *xs′*) *m ns* $\wedge$
fst-As = *hd* (*ys′* @ *xs′*) $\wedge$
lst-As = *last* (*ys′* @ *xs′*) $\wedge$
get-next (*ns* ! *the next-search*) = *option-hd xs′* $\wedge$
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*) $\wedge$
vmtf-ns-notin (*ys′* @ *xs′*) *m ns* $\wedge$
($\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) $\wedge$
($\forall$ *L*$\in$*set* (*ys′* @ *xs′*). *L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
using *atm-A* **by** *blast*
qed

definition *vmtf-find-next-undef* :: ‹*nat multiset* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$ (*nat*, *nat*) *ann-lits* $\Rightarrow$ (*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-search* $\neq$ *None* $\longrightarrow$ *Pos* (
(λ*next-search*. *next-search* $\neq$ *None* $\wedge$ *defined-lit M* (*Pos* (*the next-search*)))
(λ*next-search*. *do* {
ASSERT(*next-search* $\neq$ *None*);
let n = *the next-search*;
ASSERT(*Pos n* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$);
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 −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **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**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹$\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
have *no-next-search-all-defined*:
‹((*ns′*, *m′*, *fst-As′*, *lst-As′*, *None*), *remove*) $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *x* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$ $\Longrightarrow$ *defined-lit M x*›
for *x ns′ m′ fst-As′ lst-As′ remove*
by (*auto simp*: *vmtf-def vmtf*-$\mathcal{L}_{all}$*-def in*-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
defined-lit-map lits-of-def)
have *next-search*-$\mathcal{L}_{all}$:
‹((*ns′*, *m′*, *fst-As′*, *lst-As′*, *Some y*), *remove*) $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *y* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
for *ns′ m′ fst-As′ remove y lst-As′*
by (*auto simp*: *vmtf-def vmtf*-$\mathcal{L}_{all}$*-def in*-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
defined-lit-map lits-of-def)
have *next-search-le-A′*:

‹((ns', m', fst-As', lst-As', Some y), remove) ∈ vmtf 𝒜 M ⟹ y < length ns'›
    for *ns' m' fst-As' remove y lst-As'*
    by (*auto simp*: *vmtf-def vmtf-ℒ_all-def in-ℒ_all-atm-of-in-atms-of-iff*
        defined-lit-map lits-of-def)
  show *?thesis*
    unfolding *vmtf-find-next-undef-def*
    apply (*refine-vcg*
      WHILEIT-rule[**where** *R*=‹{(get-next (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (ℒ_all 𝒜)}›])
    subgoal using *vmtf* **by** (*rule wf-vmtf-get-next*)
    subgoal using *next-search vmtf* **by** *auto*
    subgoal using *vmtf* **by** (*auto dest*!: *next-search-ℒ_all simp*: *image-image in-ℒ_all-atm-of-in-atms-of-iff*)
    subgoal using *vmtf* **by** *auto*
    subgoal using *vmtf* **by** *auto*
    subgoal using *vmtf* **by** (*auto dest*: *next-search-le-A'*)
    subgoal by (*auto simp*: *image-image in-ℒ_all-atm-of-in-atms-of-iff*)
        (*metis next-search-ℒ_all option.distinct(1) option.sel vmtf-next-search-take-next*)
    subgoal by (*auto simp*: *image-image in-ℒ_all-atm-of-in-atms-of-iff*)
        (*metis next-search-ℒ_all option.distinct(1) option.sel vmtf-next-search-take-next*)
    subgoal by (*auto dest*: *no-next-search-all-defined next-search-ℒ_all*)
    subgoal by (*auto dest*: *next-search-le-A'*)
    subgoal for *x1 ns' x2 m' x2a fst-As' next-search' x2c s*
      by (*auto dest*: *no-next-search-all-defined next-search-ℒ_all*)
    subgoal by (*auto dest*: *vmtf-next-search-take-next*)
    subgoal by (*auto simp*: *image-image in-ℒ_all-atm-of-in-atms-of-iff*)
    done
qed

definition *vmtf-mark-to-rescore*
  :: ‹nat ⇒ vmtf-remove-int ⇒ 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 ∈atms-of (ℒ_all 𝒜)› **and**
    vmtf: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf 𝒜 M›
  shows ‹vmtf-mark-to-rescore L ((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf 𝒜 M›
proof −
  obtain *xs' ys'* **where**
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› **and**
    fst-As: ‹fst-As = hd (ys' @ xs')› **and**
    lst-As: ‹lst-As = last (ys' @ xs')› **and**
    next-search: ‹next-search = option-hd xs'› **and**
    abs-vmtf: ‹vmtf-ℒ_all 𝒜 M ((set xs', set ys'), to-remove)› **and**
    notin: ‹vmtf-ns-notin (ys' @ xs') m ns› **and**
    atm-A: ‹∀ L∈atms-of (ℒ_all 𝒜). L < length ns› **and**
    ‹∀ L∈set (ys' @ xs'). L ∈ atms-of (ℒ_all 𝒜)›
    using *vmtf* **unfolding** *vmtf-def* **by** *fast*
  moreover have ‹vmtf-ℒ_all 𝒜 M ((set xs', set ys'), insert L to-remove)›
    using *abs-vmtf L* **unfolding** *vmtf-ℒ_all-def*
    by *auto*
  ultimately show *?thesis*
    unfolding *vmtf-def vmtf-mark-to-rescore-def* **by** *fast*
qed

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 –
  obtain *xs′ ys′* **where**
    vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
    fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
    lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
    next-search: ‹*next-search* = *option-hd xs′*› **and**
    abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *remove*)› **and**
    notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
    atm-A: ‹$\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
    ys′-xs′-$\mathcal{L}_{all}$: ‹$\forall$ *L*$\in$*set* (*ys′* @ *xs′*). *L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
    using *vmtf* **unfolding** *vmtf-def* **by** *fast*
  obtain *ns′ m′ fst-As′ next-search′ remove″ lst-As′* **where**
    S: ‹*?S* = ((*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*), *remove″*)›
    by (*cases ?S*) *auto*
  have *L-ys′-iff*: ‹*L* $\in$ *set ys′* $\longleftrightarrow$ (*next-search* = *None* $\vee$ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
    using *vmtf-atm-of-ys-iff*[*OF vmtf-ns next-search abs-vmtf L-N*] **.**
  have *dist*: ‹*distinct* (*ys′* @ *xs′*)›
    using *vmtf-ns-distinct*[*OF vmtf-ns*] **.**
  have ‹*L* $\in$ *set* (*xs′* @ *ys′*)›
    using *abs-vmtf L-N* **unfolding** *vmtf*-$\mathcal{L}_{all}$*-def* **by** *auto*
  then have *L-ys′-xs′*: ‹*L* $\in$ *set ys′* $\longleftrightarrow$ *L* $\notin$ *set xs′*›
    using *dist* **by** *auto*
  have [*simp*]: ‹*remove″* = *remove*›
    using *S* **unfolding** *vmtf-unset-def* **by** (*auto split*: *if-splits*)
  have ‹$\exists$ *xs′ ys′*.
      vmtf-ns (*ys′* @ *xs′*) *m′ ns′* $\wedge$
      fst-As′ = *hd* (*ys′* @ *xs′*) $\wedge$
      lst-As′ = *last* (*ys′* @ *xs′*) $\wedge$
      next-search′ = *option-hd xs′* $\wedge$
      vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set xs′*, *set ys′*), *remove″*) $\wedge$
      vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* $\wedge$ ($\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) $\wedge$
      ($\forall$ *L*$\in$*set* (*ys′* @ *xs′*). *L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
  proof (*cases* ‹*L* $\in$ *set xs′*›)
    case *True*
    then have *C*[*unfolded L-def*]: ‹$\neg$(*next-search* = *None* $\vee$ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
      by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
    have *abs-vmtf*: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set xs′*, *set ys′*), *remove*)›
      using *S abs-vmtf dist L-ys′-xs′ True* **unfolding** *vmtf*-$\mathcal{L}_{all}$*-def vmtf-unset-def*
      by (*cases M*) (*auto simp*: *C*)
    show *?thesis*
      using *S True* **unfolding** *vmtf-unset-def L-ys′-xs′*[*symmetric*]
      apply –
      apply (*simp add*: *C*)
      using *vmtf-ns fst-As next-search abs-vmtf notin atm-A ys′-xs′*-$\mathcal{L}_{all}$ *lst-As*
      by *auto*
  next
    case *False*

then have *C*[*unfolded L-def*]: ‹*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*)›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *L-ys*: ‹*L* ∈ *set ys′*›
by (*use False L-ys′-xs′* **in** *auto*)
define *y-ys* **where** ‹*y-ys* ≡ *takeWhile* ((≠) *L*) *ys′*›
define *x-ys* **where** ‹*x-ys* ≡ *drop* (*length y-ys*) *ys′*›
let *?ys′* = ‹*y-ys*›
let *?xs′* = ‹*x-ys* @ *xs′*›
have *x-ys-take-ys′*: ‹*y-ys* = *take* (*length y-ys*) *ys′*›
unfolding *y-ys-def*
by (*subst take-length-takeWhile-eq-takeWhile*[*of* ‹(≠) *L*› ‹*ys′*›, *symmetric*]) *standard*
have *ys′-y-x*: ‹*ys′* = *y-ys* @ *x-ys*›
by (*subst x-ys-take-ys′*) (*auto simp*: *x-ys-def*)
have *y-ys-le-ys′*: ‹*length y-ys* < *length ys′*›
using *L-ys* **by** (*metis* (*full-types*) *append-eq-conv-conj append-self-conv le-antisym*
length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys′ y-ys-def)
from *nth-length-takeWhile*[*OF this*[*unfolded y-ys-def*]] **have** [*simp*]: ‹*x-ys* ≠ []› ‹*hd x-ys* = *L*›
using *y-ys-le-ys′* **unfolding** *x-ys-def y-ys-def*
by (*auto simp*: *x-ys-def y-ys-def hd-drop-conv-nth*)
have [*simp*]: ‹*ns′* = *ns*› ‹*m′* = *m*› ‹*fst-As′* = *fst-As*› ‹*next-search′* = *Some* (*atm-of* (*lit-of* (*hd M*)))›
‹*lst-As′* = *lst-As*›
using *S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)
have *L-y-ys*: ‹*L* ∉ *set y-ys*›
unfolding *y-ys-def* **by** (*metis* (*full-types*) *takeWhile-eq-all-conv takeWhile-idem*)
have ‹*vmtf-ns* (*?ys′* @ *?xs′*) *m ns*›
using *vmtf-ns* **unfolding** *ys′-y-x* **by** *simp*
moreover have ‹*fst-As′* = *hd* (*?ys′* @ *?xs′*)›
using *fst-As* **unfolding** *ys′-y-x* **by** *simp*
moreover have ‹*lst-As′* = *last* (*?ys′* @ *?xs′*)›
using *lst-As* **unfolding** *ys′-y-x* **by** *simp*
moreover have ‹*next-search′* = *option-hd ?xs′*›
by *auto*
moreover {
have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs′*, *set ?ys′*), *remove*)›
using *abs-vmtf dist* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def ys′-y-x*
by *auto*
then have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set ?xs′*, *set ?ys′*), *remove*)›
using *dist L-y-ys* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def ys′-y-x ys′-y-x*
by (*cases M*) *auto*
}
moreover have ‹*vmtf-ns-notin* (*?ys′* @ *?xs′*) *m ns*›
using *notin* **unfolding** *ys′-y-x* **by** *simp*
moreover have ‹∀ *L*∈*set* (*?ys′* @ *?xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *ys′-xs′*-$\mathcal{L}_{all}$ **unfolding** *ys′-y-x* **by** *simp*
ultimately show *?thesis*
using *S False atm-A* **unfolding** *vmtf-unset-def L-ys′-xs′*[*symmetric*]
by (*fastforce simp add*: *C*)
qed
then show *?thesis*
unfolding *vmtf-def S*
by *fast*
qed

definition *vmtf-mark-to-rescore-and-unset* :: ‹*nat* ⇒ *vmtf-remove-int* ⇒ *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*) ∈ *vmtf* $\mathcal{A}$ *M* ⟷
L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧ ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *b*) ∈ *vmtf* $\mathcal{A}$ *M*›
(**is** ‹*?A* ⟷ *?L* ∧ *?B*›)
proof
assume *vmtf*: *?A*
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *insert L b*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *b*)› **and** *L*: *?L*
using *abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def* **by** *auto*
ultimately have ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns* ∧
fst-As = *hd* (*ys′* @ *xs′*) ∧
next-search = *option-hd xs′* ∧
lst-As = *last* (*ys′* @ *xs′*) ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *b*) ∧
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* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show ‹*?L* ∧ *?B*›
using *L* **unfolding** *vmtf-def* **by** *fast*
next
assume *vmtf*: ‹*?L* ∧ *?B*›
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *b*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *insert L b*)›
using *vmtf abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def* **by** *auto*
ultimately have ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns* ∧
fst-As = *hd* (*ys′* @ *xs′*) ∧
next-search = *option-hd xs′* ∧
lst-As = *last* (*ys′* @ *xs′*) ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *insert L b*) ∧
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* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show *?A*
unfolding *vmtf-def* **by** *fast*
qed

lemma *vmtf-append-remove-iff′*:
‹(*vm*, *insert L b*) ∈ *vmtf* $\mathcal{A}$ *M* ⟷

$L \in$ *atms-of* $(\mathcal{L}_{all}\ \mathcal{A}) \wedge (vm,\ b) \in$ *vmtf* $\mathcal{A}$ M›
by (*cases vm*) (*auto simp*: *vmtf-append-remove-iff*)

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* $\in$ -›)
using *vmtf-unset-vmtf-tl*[*OF assms*(*2*$-$)[*unfolded assms*(*1*)]] *L-N*
unfolding *vmtf-mark-to-rescore-and-unset-def vmtf-mark-to-rescore-def*
by (*cases* ‹*vmtf-unset* (*atm-of* (*lit-of* (*hd* M))) ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)›)
(*auto simp*: *vmtf-append-remove-iff*)

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 $-$
obtain *ns m fst-As lst-As next-search remove* **where**
vm: ‹*vm* = ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)›
by (*cases vm*) *auto*

obtain *xs'* *ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ M ((*set xs'*, *set ys'*), *remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹$\forall L \in$*atms-of* $(\mathcal{L}_{all}\ \mathcal{A})$. $L <$ *length ns*› **and**
‹$\forall L \in$*set* (*ys'* @ *xs'*). $L \in$ *atms-of* $(\mathcal{L}_{all}\ \mathcal{A})$›
using *vmtf* **unfolding** *vmtf-def vm* **by** *fast*
show *?thesis*
using *atm-A abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *vm*)
qed

lemma *vmtf-ns-Cons*:
assumes
vmtf: ‹*vmtf-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-Node n None* (*Some b*),
b := *VMTF-Node* (*stamp* (*xs*!b)) (*Some a*) (*get-next* (*xs*!b))]› **and**
nn': ‹$n' \geq n$›
shows ‹*vmtf-ns* (a # b # l) n' *xs'*›
proof $-$
have ‹*vmtf-ns* (b # l) *m* (*xs*[a := *VMTF-Node n None* (*Some b*)])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf*])
subgoal using *ab a-l a-xs* **by** *auto*
subgoal using *a-xs vmtf-ns-le-length*[*OF vmtf*] **by** *auto*

    done
  then show *?thesis*
    apply (*rule vmtf-ns.Cons*[*of* - - - - - *n*])
    subgoal using *a-xs* **by** *simp*
    subgoal using *a-xs* **by** *simp*
    subgoal using *ab* **.**
    subgoal using *a-l* **.**
    subgoal using *nm* **.**
    subgoal using *xs′ ab a-xs* **by** (*cases* ‹*xs* ! *b*›) *auto*
    subgoal using *nn′* **.**
    done
qed

definition (**in** −) *vmtf-cons* **where**
‹*vmtf-cons ns L cnext st* =
  (*let*
    ns = *ns*[*L* := *VMTF-Node* (*Suc st*) *None cnext*];
    ns = (*case cnext of None* ⇒ *ns*
       | *Some cnext* ⇒ *ns*[*cnext* := *VMTF-Node* (*stamp* (*ns*!*cnext*)) (*Some L*) (*get-next* (*ns*!*cnext*))]) *in*
  ns)
›

lemma *vmtf-notin-vmtf-cons*:
  assumes
    vmtf-ns: ‹*vmtf-ns-notin xs m ns*› **and**
    cnext: ‹*cnext* = *option-hd xs*› **and**
    L-xs: ‹*L* ∉ *set xs*›
  shows
    ‹*vmtf-ns-notin* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
  case *Nil*
  then show *?thesis*
    using *assms* **by** (*auto simp*: *vmtf-ns-notin-def vmtf-cons-def elim*: *vmtf-nsE*)
next
  case (*Cons L′ xs′*) **note** *xs* = *this*
  show *?thesis*
    using *assms* **unfolding** *xs vmtf-ns-notin-def xs vmtf-cons-def* **by** *auto*
qed

lemma *vmtf-cons*:
  assumes
    vmtf-ns: ‹*vmtf-ns xs m ns*› **and**
    cnext: ‹*cnext* = *option-hd xs*› **and**
    L-A: ‹*L* < *length ns*› **and**
    L-xs: ‹*L* ∉ *set xs*›
  shows
    ‹*vmtf-ns* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
  case *Nil*
  then show *?thesis*
    using *assms* **by** (*auto simp*: *vmtf-ns-single-iff vmtf-cons-def elim*: *vmtf-nsE*)
next
  case (*Cons L′ xs′*) **note** *xs* = *this*
  show *?thesis*
    unfolding *xs*
    apply (*rule vmtf-ns-Cons*[*OF vmtf-ns*[*unfolded xs*], *of* - ‹*Suc m*›])

```
    subgoal using L-A .
    subgoal using L-xs unfolding xs by simp
    subgoal using L-xs unfolding xs by simp
    subgoal by simp
    subgoal using cnext L-xs
      by (auto simp: vmtf-cons-def Let-def xs)
    subgoal by linarith
    done
qed

lemma length-vmtf-cons[simp]: ‹length (vmtf-cons ns L n m) = length ns›
  by (auto simp: vmtf-cons-def Let-def split: option.splits)

lemma wf-vmtf-get-prev:
  assumes vmtf: ‹((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf 𝒜 M›
  shows ‹wf {(get-prev (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (ℒ_all 𝒜)}› (is ‹wf ?R›)
proof (rule ccontr)
  assume ‹¬ ?thesis›
  then obtain f where
    f: ‹(f (Suc i), f i) ∈ ?R› for i
    unfolding wf-iff-no-infinite-down-chain by blast

  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    lst-As: ‹lst-As = last (ys' @ xs')› and
    next-search: ‹next-search = option-hd xs'› and
    abs-vmtf: ‹vmtf-ℒ_all 𝒜 M ((set xs', set ys'), to-remove)› and
    notin: ‹vmtf-ns-notin (ys' @ xs') m ns› and
    atm-A: ‹∀ L∈atms-of (ℒ_all 𝒜). L < length ns›
    using vmtf unfolding vmtf-def by fast
  let ?f0 = ‹the (f 0)›
  have f-None: ‹f i ≠ None› for i
    using f[of i] by fast
  have f-Suc : ‹f (Suc n) = get-prev (ns ! the (f n))› for n
    using f[of n] by auto
  have f0-length: ‹?f0 < length ns›
    using f[of 0] atm-A
    by auto
  have f0-in: ‹?f0 ∈ set (ys' @ xs')›
    apply (rule ccontr)
    using notin f-Suc[of 0] f0-length unfolding vmtf-ns-notin-def
    by (auto simp: f-None)
  then obtain i0 where
    i0: ‹(ys' @ xs') ! i0 = ?f0› ‹i0 < length (ys' @ xs')›
    by (meson in-set-conv-nth)
  define zs where ‹zs = ys' @ xs'›
  have H: ‹ys' @ xs' = take m (ys' @ xs') @ [(ys' @ xs') ! m, (ys' @ xs') ! (m+1)] @
      drop (m+2) (ys' @ xs')›
    if ‹m + 1 < length (ys' @ xs')›
    for m
    using that
    unfolding zs-def[symmetric]
    apply -
    apply (subst id-take-nth-drop[of m])
    by (auto simp: take-Suc-conv-app-nth Cons-nth-drop-Suc  simp del: append-take-drop-id)
```

```
  have ‹the (f n) = (ys' @ xs') ! (i0 - n) ∧ i0 - n ≥ 0 ∧ n ≤ i0› for n
  proof (induction n)
    case 0
    then show ?case using i0 by simp
  next
    case (Suc n')
    have i0-le: ‹n' < i0›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have ‹i0 = n'›
        using Suc by auto
      then have ‹ys' @ xs' = [the (f n')] @ tl (ys' @ xs')›
        using Suc f0-in
        by (cases ‹ys' @ xs'›) auto
      then show False
        using vmtf-ns-hd-prev[of ‹the (f n')› ‹tl (ys' @ xs')› m ns] vmtf-ns
         f-Suc[of n'] by (auto simp: f-None)
    qed
    have get-prev: ‹get-prev (ns ! ((ys' @ xs') ! (i0 - (n' +1) + 1))) =
          Some ((ys' @ xs') ! ((i0 - (n' + 1))))›
      apply (rule vmtf-ns-last-mid-get-prev[of ‹take (i0 - (n' +1)) (ys' @ xs')› - -
        ‹drop ((i0 - (n' + 1)) + 2) (ys' @ xs')› m])
      apply (subst H[symmetric])
      subgoal using i0-le i0 by auto
      subgoal using vmtf-ns by simp
      done
    then show ?case
      using f-Suc[of n'] Suc i0-le by auto
  qed
  from this[of ‹Suc i0›] show False
    by auto
qed

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^λ-. 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* $\mathcal{A}$›
shows
‹*vmtf-rescale vm* ≤ *SPEC* (λ*vm*. (*vm*, *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M* ∧ *fst* (*snd vm*) ≤ *uint32-max*)›
(**is** ‹*?A* ≤ *?R*›)
proof −
obtain *ns m fst-As lst-As next-search* **where**
vm: ‹*vm* = ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))›
by (*cases vm*) *auto*

obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **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**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
in-lall: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def vm* **by** *fast*
have [*dest*]: ‹*ys′* = [] ⟹ *xs′* = [] ⟹ *False*› **and**
[*simp*]: ‹*ys′* = [] ⟶ *xs′* ≠ []›
using *abs-vmtf nempty* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
have *1*: ‹*RES* (*vmtf* $\mathcal{A}$ *M*) = *do* {
a ← *RETURN* ();
RES (*vmtf* $\mathcal{A}$ *M*)
}›
by *auto*
define *zs* **where** ‹*zs* ≡ *ys′* @ *xs′*›

define *I′* **where**
‹*I′* ≡ λ(*ns′*, *n*::*nat*, *lst*::*nat option*).
map get-prev ns = *map get-prev ns′* ∧
map get-next ns = *map get-next ns′* ∧
(∀ *i*<*n*. *stamp* (*ns′* ! (*rev zs* ! *i*)) = *i*) ∧
(*lst* ≠ *None* ⟶ *n* < *length* (*zs*) ∧ *the lst* = *zs* ! (*length zs* − *Suc n*)) ∧
(*lst* = *None* ⟶ *n* = *length zs*) ∧
n ≤ *length zs*›
have [*simp*]: ‹*zs* ≠ []›
unfolding *zs-def* **by** *auto*
have *I′0*: ‹*I′* (*ns*, *0*, *Some lst-As*)›
using *vmtf lst-As* **unfolding** *I′-def vm zs-def*[*symmetric*] **by** (*auto simp*: *last-conv-nth*)

have *lits*: ‹*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* ‘# *mset zs*)› **and**
dist: ‹*distinct zs*›
using *abs-vmtf vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *vmtf-def zs-def*
vmtf-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *literals-are-in*-$\mathcal{L}_{in}$-*alt-def inj-on-def*)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset zs*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset zs*))›
by (*auto simp*: *tautology-decomp*)

have *length-zs-le*: ‹*length zs* < *uint32-max*› **using** *vmtf-ns-distinct*[*OF vmtf-ns*]

using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)

have ‹*wf* {(*a*, *b*). (*a*, *b*) ∈ {(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧ *the a* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}}›
by (*rule wf-subset*[*OF wf-vmtf-get-prev*[*OF vmtf*[*unfolded vm*]]]) *auto*
from *wf-snd-wf-pair*[*OF wf-snd-wf-pair*[*OF this*]]
have *wf*: ‹*wf* {((-, -, *a*), (-, -, *b*)). (*a*, *b*) ∈ {(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧
the a ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}}›
by (*rule wf-subset*) *auto*
have *zs-lall*: ‹*zs* ! (*length zs* − *Suc n*) ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **for** *n*
using *abs-vmtf nth-mem*[*of* ‹*length zs* − *Suc n*› *zs*] **unfolding** *zs-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
then have *zs-le-ns*[*simp*]: ‹*zs* ! (*length zs* − *Suc n*) < *length ns*› **for** *n*
using *atm-A* **by** *auto*
have *loop-body*: ‹(*case s′ of*
(*ns*, *n*, *a*) ⇒ *do* {
ASSERT (*a* ≠ *None*);
ASSERT (*n* + *1* ≤ *uint-max*);
ASSERT(*the a* < *length ns*);
RETURN (*update-stamp ns n* (*the a*), *n* + *1*, *get-prev* (*ns* ! *the a*))
})
≤ *SPEC*
(λ*s′a*. *True* ∧
I′ s′a ∧
(*s′a*, *s′*)
∈ {((-, -, *a*), -, -, *b*).
(*a*, *b*)
∈ {(*get-prev* (*ns* ! *the a*), *a*) |*a*.
a ≠ *None* ∧ *the a* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}})›
if
I′: ‹*I′ s′*› **and**
cond: ‹*case s′ of* (*ns*, *n*, *lst-As*) ⇒ *lst-As* ≠ *None*›
for *s′*
proof −
obtain *ns′ n′ a′* **where** *s′*: ‹*s′* = (*ns′*, *n′* , *a′*)›
by (*cases s′*)
have
a[*simp*]: ‹*a′* = *Some* (*zs* ! (*length zs* − *Suc n′*))› **and**
eq-prev: ‹*map get-prev ns* = *map get-prev ns′*› **and**
eq-next: ‹*map get-next ns* = *map get-next ns′*› **and**
eq-stamps: ‹$\bigwedge$*i*. *i*<*n′* ⟹ *stamp* (*ns′* ! (*rev zs* ! *i*)) = *i*› **and**
n′-le: ‹*n′* < *length zs*›
using *I′ cond* **unfolding** *I′-def prod.simps s′*
by *auto*
have [*simp*]: ‹*length ns′* = *length ns*›
using *arg-cong*[*OF eq-prev*, *of length*] **by** *auto*
have *vmtf-as*: ‹*vmtf-ns*
(*take* (*length zs* − (*n′* + *1*)) *zs* @
zs ! (*length zs* − (*n′* + *1*)) #
drop (*Suc* (*length zs* − (*n′* + *1*))) *zs*)
m ns›
apply (*subst Cons-nth-drop-Suc*)
subgoal by *auto*
apply (*subst append-take-drop-id*)
using *vmtf-ns* **unfolding** *zs-def*[*symmetric*] **.**

have ‹*get-prev* (*ns′* ! *the a′*) ≠ *None* ⟶
n′ + *1* < *length zs* ∧
the (*get-prev* (*ns′* ! *the a′*)) = *zs* ! (*length zs* − *Suc* (*n′* + *1*))›
using *n′-le vmtf-ns arg-cong*[*OF eq-prev*, *of* ‹λ*xs*. *xs* ! (*zs* ! (*length zs* − *Suc n′*))›]
vmtf-ns-last-mid-get-prev-option-last[*OF vmtf-as*]
by (*auto simp*: *last-conv-nth*)
moreover have ‹*map get-prev ns* = *map get-prev* (*update-stamp ns′ n′* (*the a′*))›
unfolding *update-stamp.simps*
apply (*subst map-update*)
apply (*subst list-update-id′*)
subgoal by *auto*
subgoal using *eq-prev* **.**
done
moreover have ‹*map get-next ns* = *map get-next* (*update-stamp ns′ n′* (*the a′*))›
unfolding *update-stamp.simps*
apply (*subst map-update*)
apply (*subst list-update-id′*)
subgoal by *auto*
subgoal using *eq-next* **.**
done
moreover have ‹*i*<*n′* + *1* ⟹ *stamp* (*update-stamp ns′ n′* (*the a′*) ! (*rev zs* ! *i*)) = *i*› **for** *i*
using *eq-stamps*[*of i*] *vmtf-ns-distinct*[*OF vmtf-ns*] *n′-le*
unfolding *zs-def*[*symmetric*]
by (*cases* ‹*i* < *n′*›)
(*auto simp*: *rev-nth nth-eq-iff-index-eq*)
moreover have ‹*n′* + *1* ≤ *length zs*›
using *n′-le* **by** (*auto simp*: *Suc-le-eq*)
moreover have ‹*get-prev* (*ns′* ! *the a′*) = *None* ⟹ *n′* + *1* = *length zs*›
using *n′-le vmtf-ns arg-cong*[*OF eq-prev*, *of* ‹λ*xs*. *xs* ! (*zs* ! (*length zs* − *Suc n′*))›]
vmtf-ns-last-mid-get-prev-option-last[*OF vmtf-as*]
by *auto*
ultimately have *I′-f*: ‹*I′* (*update-stamp ns′ n′* (*the a′*), *n′* + *1*, *get-prev* (*ns′* ! *the a′*))›
using *cond n′-le* **unfolding** *I′-def prod.simps s′*
by *simp*

show *?thesis*
unfolding *s′ prod.case*
apply *refine-vcg*
subgoal using *cond* **by** *auto*
subgoal using *length-zs-le n′-le* **by** *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by (*rule I′-f*)
subgoal
using *arg-cong*[*OF eq-prev*, *of* ‹λ*xs*. *xs* ! (*zs* ! (*length zs* − *Suc n′*))›] *zs-lall*
by *auto*
done
qed
have *loop-final*: ‹*s* ∈ {*x*. (*case x of*
(*ns*, *m*, *uua-*) ⇒
RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)))
≤ *?R*}›
if
‹*True*› **and**
‹*I′ s*› **and**
‹¬ (*case s of* (*ns*, *n*, *lst-As*) ⇒ *lst-As* ≠ *None*)›

for *s*
proof −
obtain *ns'* *n'* *a'* **where** *s*: ‹*s* = (*ns'*, *n'* , *a'*)›
by (*cases s*)
have
[*simp*]:‹*a'* = *None*› **and**
eq-prev: ‹*map get-prev ns* = *map get-prev ns'*› **and**
eq-next: ‹*map get-next ns* = *map get-next ns'*› **and**
stamp: ‹∀ *i*<*n'*. *stamp* (*ns'* ! (*rev zs* ! *i*)) = *i*› **and**
[*simp*]: ‹*n'* = *length zs*›
using *that* **unfolding** *I'-def s prod.case* **by** *auto*
have [*simp*]: ‹*length ns'* = *length ns*›
using *arg-cong*[*OF eq-prev*, *of length*] **by** *auto*
have [*simp*]: ‹*map* ((!) (*map stamp ns'*)) (*rev zs*) = [*0*..<*length zs*]›
apply (*subst list-eq-iff-nth-eq*, *intro conjI*)
subgoal by *auto*
subgoal using *stamp* **by** (*auto simp*: *rev-nth*)
done
then have *stamps-zs*[*simp*]: ‹*map* ((!) (*map stamp ns'*)) *zs* = *rev* [*0*..<*length zs*]›
unfolding *rev-map*[*symmetric*]
using *rev-swap* **by** *blast*

have ‹*sorted* (*map* ((!) (*map stamp ns'*)) (*rev zs*))›
by *simp*
moreover have ‹*distinct* (*map* ((!) (*map stamp ns'*)) *zs*)›
by *simp*
moreover have ‹∀ *a*∈*set zs*. *get-prev* (*ns'* ! *a*) = *get-prev* (*ns* ! *a*)›
using *eq-prev map-eq-nth-eq* **by** *fastforce*
moreover have ‹∀ *a*∈*set zs*. *get-next* (*ns'* ! *a*) = *get-next* (*ns* ! *a*)›
using *eq-next map-eq-nth-eq* **by** *fastforce*
moreover have ‹∀ *a*∈*set zs*. *stamp* (*ns'* ! *a*) = *map stamp ns'* ! *a*›
using *vmtf-ns vmtf-ns-le-length zs-def* **by** *auto*
moreover have ‹*length ns* ≤ *length ns'*›
by *simp*
moreover have ‹∀ *a*∈*set zs*. *a* < *length* (*map stamp ns'*)›
using *vmtf-ns vmtf-ns-le-length zs-def* **by** *auto*
moreover have ‹∀ *a*∈*set zs*. *map stamp ns'* ! *a* < *n'*›
proof
fix *a*
assume ‹*a* ∈ *set zs*›
then have ‹*map stamp ns'* ! *a* ∈ *set* (*map* ((!) (*map stamp ns'*)) *zs*)›
by (*metis in-set-conv-nth length-map nth-map*)
then show ‹*map stamp ns'* ! *a* < *n'*›
unfolding *stamps-zs* **by** *simp*
qed
ultimately have ‹*vmtf-ns zs n' ns'*›
using *vmtf-ns-rescale*[*OF vmtf-ns*, *of* ‹*map stamp ns'*› *ns'*, *unfolded zs-def*[*symmetric*]]
by *fast*
moreover have ‹*vmtf-ns-notin zs* (*length zs*) *ns'*›
using *notin map-eq-nth-eq*[*OF eq-prev*] *map-eq-nth-eq*[*OF eq-next*]
unfolding *zs-def*[*symmetric*]
by (*auto simp*: *vmtf-ns-notin-def*)
ultimately have ‹((*ns'*, *n'*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
using *fst-As lst-As next-search abs-vmtf atm-A notin in-lall*
unfolding *vmtf-def in-pair-collect-simp prod.case* **apply** −
apply (*rule exI*[*of* - *xs'*])

apply (*rule exI*[*of* - *ys'*])
unfolding *zs-def*[*symmetric*]
by *auto*
then show *?thesis*
using *length-zs-le*
by (*auto simp*: *s*)
qed

have *H*: ‹$WHILE_T{}^{\lambda\text{-}.\ True}$ (λ(*ns*, *n*, *lst-As*). *lst-As* $\neq$ *None*)
(λ(*ns*, *n*, *a*). *do* {
- $\leftarrow$ *ASSERT* (*a* $\neq$ *None*);
- $\leftarrow$ *ASSERT* (*n* + *1* $\leq$ *uint-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*)
$\leq$ *SPEC*
(λ*x*. (*case x of*
(*ns*, *m*, *uua-*) $\Rightarrow$
RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)))
$\leq$ *?R*)
›
apply (*rule WHILEIT-rule-stronger-inv-RES*[**where** I' = I' **and**
R = ‹{((-, -, *a*), (-, -, *b*)). (*a*, *b*) $\in$
{(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* $\neq$ *None* $\wedge$ *the a* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}}›])
subgoal
by (*rule wf*)
subgoal by *fast*
subgoal by (*rule I'0*)
subgoal for *s'*
by (*rule loop-body*)
subgoal for *s*
by (*rule loop-final*)
done

show *?thesis*
unfolding *vmtf-rescale-def vm prod.case*
apply (*subst bind-rule-complete-RES*)
apply (*rule H*)
done
qed

definition *vmtf-flush*
:: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*) *ann-lits* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$ *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* $\Rightarrow$ (*bool list* $\times$ *nat set*) *set*› **where**
‹*atoms-hash-rel* $\mathcal{A}$ = {(*C*, *D*). ($\forall$ *L* $\in$ *D*. *L* < *length C*) $\wedge$ ($\forall$ *L* < *length C*. *C* ! *L* $\longleftrightarrow$ *L* $\in$ *D*) $\wedge$
($\forall$ *L* $\in\#$ $\mathcal{A}$. *L* < *length C*) $\wedge$ *D* $\subseteq$ *set-mset* $\mathcal{A}$}›

definition *distinct-hash-atoms-rel*
:: ‹*nat multiset* $\Rightarrow$ (('*v list* $\times$ '*v set*) $\times$ '*v set*) *set*›
where
‹*distinct-hash-atoms-rel* $\mathcal{A}$ = {((*C*, *h*), *D*). *set C* = *D* $\wedge$ *h* = *D* $\wedge$ *distinct C*}›

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

lemma *distinct-atoms-rel-alt-def*:
‹*distinct-atoms-rel* $\mathcal{A}$ = {((*D′*, *C*), *D*). (∀ *L* ∈ *D*. *L* < *length C*) ∧ (∀ *L* < *length C*. *C* ! *L* ⟷ *L* ∈ *D*) ∧
(∀ *L* ∈# $\mathcal{A}$. *L* < *length C*) ∧ *set D′* = *D* ∧ *distinct D′* ∧ *set D′* ⊆ *set-mset* $\mathcal{A}$}›
unfolding *distinct-atoms-rel-def atoms-hash-rel-def distinct-hash-atoms-rel-def prod-rel-def*
apply *rule*
subgoal
by (*auto simp*: *mset-set-set*)
subgoal
by (*auto simp*: *mset-set-set*)
done

lemma *distinct-atoms-rel-empty-hash-iff*:
‹(([], *h*), {}) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟷ (∀ *L* ∈# $\mathcal{A}$. *L* < *length h*) ∧ (∀ *i*∈*set h*. *i* = *False*)›
unfolding *distinct-atoms-rel-alt-def all-set-conv-nth*
by *auto*

definition *atoms-hash-del-pre* **where**
‹*atoms-hash-del-pre i xs* = (*i* < *length xs*)›

definition *atoms-hash-del* **where**
‹*atoms-hash-del i xs* = *xs*[*i* := *False*]›

definition *vmtf-flush-int* :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ - ⇒ - *nres*› **where**
‹*vmtf-flush-int* $\mathcal{A}_{in}$ = (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′* + *fst* (*snd vm*) ≥ *uint64-max*
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove′* + *fst* (*snd vm*) ≤ *uint64-max*);
(-, *vm*, *h*) ← $WHILE_T$ λ(*i*, *vm′*, *h*). *i* ≤ *length to-remove′* ∧ *fst* (*snd vm′*) = *i* + *fst* (*snd vm*) ∧ (*i* < *length to-remove*
(λ(*i*, *vm*, *h*). *i* < *length to-remove′*)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove′*);
ASSERT(*to-remove′*!*i* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove′*!*i*) *h*);
RETURN (*i*+*1*, *vmtf-en-dequeue M* (*to-remove′*!*i*) *vm*, *atoms-hash-del* (*to-remove′*!*i*) *h*)})
(*0*, *vm*, *h*);
RETURN (*vm*, (*emptied-list to-remove′*, *h*))
})›

lemma *vmtf-change-to-remove-order*:
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*› **and**
CD-rem: ‹((*C*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}_{in}$› **and**

bounded: ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
shows ‹*vmtf-flush-int* $\mathcal{A}_{in}$ *M* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), (*C*, *D*))
$\leq \Downarrow$(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$)
(*vmtf-flush* $\mathcal{A}_{in}$ *M* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*))›
proof −
let *?vm* = ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
have *vmtf-flush-alt-def*: ‹*vmtf-flush* $\mathcal{A}_{in}$ *M ?vm* = *do* {
- ← *RETURN* ();
- ← *RETURN* ();
vm ← *RES*(*vmtf* $\mathcal{A}_{in}$ *M*);
RETURN (*vm*)
}›
unfolding *vmtf-flush-def* **by** (*auto simp*: *RES-RES-RETURN-RES RES-RETURN-RES vmtf*)

have *pre-sort*: ‹∀ *x*∈*set x1a*. *x* < *length* (*fst x1*)›
if
‹*x2* = (*x1a*, *x2a*)› **and**
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)›
for *x1 x2 x1a x2a*
proof −
show *?thesis*
using *vmtf CD-rem that* **by** (*auto simp*: *vmtf-def vmtf*-$\mathcal{L}_{all}$-*def*
distinct-atoms-rel-alt-def)
qed

have *length-le*: ‹*length x1a* ≤ *uint32-max*›
if
‹*x2* = (*x1a*, *x2a*)› **and**
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)› **and**
‹∀ *x*∈*set x1a*. *x* < *length* (*fst x1*)›
for *x1 x2 x1a x2a*
proof −
have *lits*: ‹*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*Pos* ‘# *mset x1a*)› **and**
dist: ‹*distinct x1a*›
using *that vmtf CD-rem* **unfolding** *vmtf-def*
vmtf-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *literals-are-in*-$\mathcal{L}_{in}$-*alt-def distinct-atoms-rel-alt-def inj-on-def*)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset x1a*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset x1a*))›
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)
qed

have [*refine0*]:
‹*reorder-list x1 x1a* ≤ *SPEC* (λ*c*. (*c*, ()) ∈
{(*c*, *c'*). ((*c*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$ ∧ *to-remove* = *set c* ∧
length C = *length c*})›
(**is** ‹- ≤ *SPEC*(λ-. - ∈ *?reorder-list*)›)
if
‹*x2* = (*x1a*, *x2a*)› **and**

‹((ns, m, fst-As, lst-As, next-search), C, D) = (x1, x2)›
for *x1 x2 x1a x2a*
proof −
show *?thesis*
using *that assms* **by** (*force simp*: *reorder-list-def distinct-atoms-rel-alt-def*
dest: *mset-eq-setD same-mset-distinct-iff mset-eq-length*)
qed

have [*refine0*]: ‹(*if uint64-max* $\leq$ *length to-remove'* + *fst* (*snd x1*) *then vmtf-rescale x1*
else RETURN x1)
$\leq$ *SPEC* (λc. (c, ()) $\in$
{(*vm* ,*vm'*). *uint64-max* $\geq$ *length to-remove'* + *fst* (*snd vm*) $\wedge$
(*vm*, *set to-remove'*) $\in$ *vmtf* $\mathcal{A}_{in}$ *M*}) ›
(**is** ‹- $\leq$ *SPEC*(λc. (c, ()) $\in$ *?rescale*)› **is** ‹- $\leq$ *?H*›)
if
‹*x2* = (*x1a*, *x2a*)› **and**
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)› **and**
‹$\forall$ *x*$\in$*set x1a*. *x* < *length* (*fst x1*)› **and**
‹*length x1a* $\leq$ *uint-max*› **and**
‹(*to-remove'*, *uu*) $\in$ *?reorder-list*› **and**
‹*length to-remove'* $\leq$ *uint-max*›
for *x1 x2 x1a x2a to-remove' uu*
proof −
have ‹*vmtf-rescale x1* $\leq$ *?H*›
apply (*rule order-trans*)
apply (*rule vmtf-rescale-vmtf*[*of - to-remove* $\mathcal{A}_{in}$ *M*])
subgoal using *vmtf that* **by** *auto*
subgoal using *nempty* **by** *fast*
subgoal using *bounded* **by** *fast*
subgoal using *that* **by** (*auto intro*!: *RES-refine simp*: *uint64-max-def uint32-max-def*)
done
then show *?thesis*
using *that vmtf*
by (*auto intro*!: *RETURN-RES-refine*)
qed

have *loop-ref*: ‹$WHILE_T{}^{\lambda(i,\ vm',\ h).}$ $i \leq$ *length to-remove'* $\wedge$ *fst* (*snd vm'*) = *i* + *fst* (*snd x1*) $\wedge$
(λ(*i*, *vm*, *h*). *i* < *length to-remove'*)
(λ(*i*, *vm*, *h*). *do* {
ASSERT (*i* < *length to-remove'*);
ASSERT(*to-remove'*!*i* $\in$# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove'*!*i*) *h*);
RETURN
(*i* + *1*, *vmtf-en-dequeue M* (*to-remove'* ! *i*) *vm*,
atoms-hash-del (*to-remove'*!*i*) *h*)
})
(*0*, *x1*, *x2a*)
$\leq \Downarrow$ {((*i*, *vm*::*vmtf*, *h*:: -), *vm'*). (*vm*, {}) = *vm'* $\wedge$ ($\forall$ *i*$\in$*set h*. *i* = *False*) $\wedge$ *i* = *length to-remove'*
$\wedge$
((*drop i to-remove'*, *h*), *set*(*drop i to-remove'*)) $\in$ *distinct-atoms-rel* $\mathcal{A}_{in}$}
(*RES* (*vmtf* $\mathcal{A}_{in}$ *M*))›
if
x2: ‹*x2* = (*x1a*, *x2a*)› **and**
CD: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1'*, *x2*)› **and**

x1: ‹(*x1*, *u′*) ∈ *?rescale to-remove′*›
‹(*to-remove′*, *u*) ∈ *?reorder-list*›
for *x1 x2 x1a x2a to-remove′ u u′ x1′*
proof −
define *I* **where** ‹$I \equiv \lambda(i, vm'::vmtf, h::bool\ list)$.
$i \le$ *length to-remove′* ∧ *fst* (*snd vm′*) = *i* + *fst* (*snd x1*) ∧
(*i* < *length to-remove′* ⟶
vmtf-en-dequeue-pre $\mathcal{A}_{in}$ ((*M*, *to-remove′* ! *i*), *vm′*))›
define *I′* **where** ‹$I' \equiv \lambda(i, vm::vmtf, h::\ bool\ list)$.
((*drop i to-remove′*, *h*), *set*(*drop i to-remove′*)) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$ ∧
(*vm*, *set* (*drop i to-remove′*)) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
have [*simp*]:
‹*x2* = (*C*, *D*)›
‹*x1′* = (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›
‹*x1a* = *C*›
‹*x2a* = *D*› **and**
rel: ‹((*to-remove′*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
to-rem: ‹*to-remove* = *set to-remove′*›
using *that* **by** (*auto simp*:)
have *D*: ‹*set to-remove′* = *to-remove*› **and** *dist*: ‹*distinct to-remove′*›
using *rel* **unfolding** *distinct-atoms-rel-alt-def* **by** *auto*
have *in-lall*: ‹*to-remove′* ! *x1* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)› **if** *le′*: ‹*x1* < *length to-remove′*› **for** *x1*
using *vmtf to-rem nth-mem*[*OF le′*] **by** (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*)
have *bound*: ‹*fst* (*snd x1*) + *1* ≤ *uint64-max*› **if** ‹*0* < *length to-remove′*›
using *rel vmtf to-rem that x1* **by** (*cases to-remove′*) *auto*
have *I-init*: ‹*I* (*0*, *x1*, *x2a*)› (**is** *?A*)
for *x1a x2 x1aa x2aa*
proof −
have ‹*vmtf-en-dequeue-pre* $\mathcal{A}_{in}$ ((*M*, *to-remove′* ! *0*), *x1*)› **if** ‹*0* < *length to-remove′*›
apply (*rule vmtf-vmtf-en-dequeue-pre-to-remove′*[*of* - ‹*set to-remove′*›])
using *rel vmtf to-rem that x1 bound nempty* **by** *auto*
then show *?A*
unfolding *I-def* **by** *auto*
qed
have *I′-init*: ‹*I′* (*0*, *x1*, *x2a*)› (**is** *?B*)
for *x1a x2 x1aa x2aa*
proof −
show *?B*
using *rel to-rem CD-rem that vmtf* **by** (*auto simp*: *distinct-atoms-rel-def I′-def*)
qed
have *post-loop*: ‹*do* {
ASSERT (*x2* < *length to-remove′*);
ASSERT(*to-remove′* ! *x2* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove′* ! *x2*) *x2a′*);
RETURN
(*x2* + *1*, *vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove′*!*x2*) *x2a′*)
} ≤ *SPEC*
($\lambda s'$. *I s′* ∧ *I′ s′* ∧ (*s′*, *x1a*) ∈ *measure* (λ(*i*, *vm*, *h*). *Suc* (*length to-remove′*) − *i*))›
if
I: ‹*I x1a*› **and**
I′: ‹*I′ x1a*› **and**
‹*case x1a of* (*i*, *vm*, *h*) ⇒ *i* < *length to-remove′*› **and**
x1aa: ‹*x1aa* = (*x2aa*, *x2a′*)› ‹*x1a* = (*x2*, *x1aa*)›
for *s x2 x1a x2a x1a′ x2a′ x1aa x2aa*
proof −

let *?x2a′* = ‹*set* (*drop x2 to-remove′*)›
have *le*: ‹*x2* < *length to-remove′*› **and** *vm*: ‹(*x2aa*, *set* (*drop x2 to-remove′*)) ∈ *vmtf* $\mathcal{A}_{in}$ *M*› **and**
x2a′: ‹*fst* (*snd x2aa*) = *x2* + *fst* (*snd x1*)›
using *that* **unfolding** *I-def I′-def* **by** (*auto simp*: *distinct-atoms-rel-alt-def*)
have *1*: ‹(*vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*, *?x2a′*− {*to-remove′* ! *x2*}) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
by (*rule abs-vmtf-ns-bump-vmtf-en-dequeue′*[*OF vm in-lall*[*OF le*]])
(*use nempty* **in** *auto*)
have *2*: ‹*to-remove′* ! *Suc x2* ∈ *?x2a′*− {*to-remove′* ! *x2*}›
if ‹*Suc x2* < *length to-remove′*›
using *I I′ le dist that x1aa* **unfolding** *I-def I′-def*
by (*auto simp*: *distinct-atoms-rel-alt-def in-set-drop-conv-nth I′-def*
nth-eq-iff-index-eq x2 intro: *bex-geI*[*of* - ‹*Suc x2*›])
have *3*: ‹*fst* (*snd x2aa*) = *fst* (*snd x1*) + *x2*›
using *I I′ le dist that CD*[*unfolded x2*] *x2a′* **unfolding** *I-def I′-def x2 x2a′ x1aa*
by (*auto simp*: *distinct-atoms-rel-def in-set-drop-conv-nth I′-def*
nth-eq-iff-index-eq x2 intro: *bex-geI*[*of* - ‹*Suc x2*›])
then have *4*: ‹*fst* (*snd* (*vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*)) + *1* ≤ *uint64-max*›
if ‹*Suc x2* < *length to-remove′*›
using *x1 le that*
by (*cases x2aa*)
(*auto simp*: *vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def*
split: *option.splits*)
have *1*: ‹*vmtf-en-dequeue-pre* $\mathcal{A}_{in}$
((*M*, *to-remove′* ! *Suc x2*), *vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*)›
if ‹*Suc x2* < *length to-remove′*›
by (*rule vmtf-vmtf-en-dequeue-pre-to-remove′*)
(*rule 1*, *rule 2*, *rule that*, *rule 4*[*OF that*], *rule nempty*)
have *3*: ‹(*vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*, *?x2a′* − {*to-remove′* ! *x2*}) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
by (*rule abs-vmtf-ns-bump-vmtf-en-dequeue′*[*OF vm in-lall*[*OF le*]]) (*use nempty* **in** *auto*)
have *4*: ‹((*drop* (*Suc x2*) *to-remove′*, *atoms-hash-del* (*to-remove′* ! *x2*) *x2a′*),
set (*drop* (*Suc x2*) *to-remove′*))
∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
3: ‹(*vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*, *set* (*drop* (*Suc x2*) *to-remove′*))
∈ *vmtf* $\mathcal{A}_{in}$ *M*›
using *3 I′ le to-rem that* **unfolding** *I′-def distinct-atoms-rel-alt-def atoms-hash-del-def*
by (*auto simp*: *Cons-nth-drop-Suc*[*symmetric*] *intro*: *mset-le-add-mset-decr-left1*)

have *A*: ‹*to-remove′* ! *x2* ∈ *?x2a′*›
using *I I′ le dist that x1aa* **unfolding** *I-def I′-def*
by (*auto simp*: *distinct-atoms-rel-def in-set-drop-conv-nth I′-def*
nth-eq-iff-index-eq x2 x2a′ intro: *bex-geI*[*of* - ‹*x2*›])
moreover have ‹*I* (*Suc x2*, *vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove′* ! *x2*) *x2a′*)›
using *that 1* **unfolding** *I-def*
by (*cases x2aa*)
(*auto simp*: *vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def*
split: *option.splits*)
moreover have ‹*length to-remove′* − *x2* < *Suc* (*length to-remove′*) − *x2*›
using *le* **by** *auto*
moreover have ‹*I′* (*Suc x2*, *vmtf-en-dequeue M* (*to-remove′* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove′* ! *x2*) *x2a′*)›
using *that 3 4 I′* **unfolding** *I′-def*
by *auto*
moreover have ‹*atoms-hash-del-pre* (*to-remove′* ! *x2*) *x2a′*›
unfolding *atoms-hash-del-pre-def*
using *that le A* **unfolding** *I-def I′-def* **by** (*auto simp*: *distinct-atoms-rel-alt-def*)

```
      ultimately show ?thesis
        using that in-lall[OF le]
        by (auto simp: atms-of-ℒ_all-𝒜_in)
    qed
    have [simp]: ‹∀L<length ba. ¬ ba ! L ⟹  True ∉ set ba› for ba
      by (simp add: in-set-conv-nth)
    have post-rel: ‹RETURN s
        ≤ ⇓ {((i, vm, h), vm').
              (vm, {}) = vm' ∧
              (∀i∈set h. i = False) ∧
              i = length to-remove' ∧
              ((drop i to-remove', h), set (drop i to-remove'))
              ∈ distinct-atoms-rel 𝒜_in}          (RES (vmtf 𝒜_in M))›
        if
          ‹¬ (case s of (i, vm, h) ⇒ i < length to-remove')› and
          ‹I s› and
          ‹I' s›
        for s
    proof –
      obtain i vm h where s: ‹s = (i, vm, h)› by (cases s)
      have [simp]: ‹i = length (to-remove')› and [iff]: ‹True ∉ set h› and
        [simp]: ‹(([], h), {}) ∈ distinct-atoms-rel 𝒜_in›
          ‹(vm, {}) ∈ vmtf 𝒜_in M›
        using that unfolding s I-def I'-def by (auto simp: distinct-atoms-rel-empty-hash-iff)
      show ?thesis
        unfolding s
        by (rule RETURN-RES-refine) auto
    qed

    show ?thesis
      unfolding I-def[symmetric]
      apply (refine-rcg
        WHILEIT-rule-stronger-inv-RES'[where R=‹measure (λ(i, vm::vmtf, h). Suc (length to-remove')
-i)› and
            I'=‹I'›])
      subgoal by auto
      subgoal by (rule I-init)
      subgoal by (rule I'-init)
      subgoal for x1'' x2'' x1a'' x2a'' by (rule post-loop)
      subgoal by (rule post-rel)
      done
  qed


  show ?thesis
    unfolding vmtf-flush-int-def vmtf-flush-alt-def
    apply (refine-rcg)
    subgoal by (rule pre-sort)
    subgoal by (rule length-le)
    apply (assumption+)[2]
    subgoal by auto
    apply (assumption+)[5]
    subgoal by auto
    apply (rule loop-ref; assumption)
    subgoal by (auto simp: emptied-list-def)
    done
```

qed

lemma *vmtf-change-to-remove-order′*:
‹$(uncurry\ (vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in}),\ uncurry\ (vmtf\text{-}flush\ \mathcal{A}_{in})) \in$
$[\lambda(M,\ vm).\ vm \in vmtf\ \mathcal{A}_{in}\ M \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}nempty\ \mathcal{A}_{in}]_f$
$Id \times_r (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}) \to \langle(Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in})\rangle\ nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*use vmtf-change-to-remove-order* **in** *auto*)

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*]:
‹$phase\text{-}saving\ \mathcal{A}\ (save\text{-}phase\ L\ \varphi) \longleftrightarrow phase\text{-}saving\ \mathcal{A}\ \varphi$›
by (*auto simp*: *phase-saving-def save-phase-def*)

Save opposite of the phase (e.g. for literals in the conflict clause):

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

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**
⟨$lbd\text{-}ref = \{((lbd,\ n,\ m),\ lbd').\ lbd = lbd' \land n < length\ lbd \land$
$(\forall\, k > n.\ k < length\ lbd \longrightarrow \neg lbd!k) \land$
$length\ lbd \le Suc\ (Suc\ (uint\text{-}max\ div\ 2)) \land n < length\ lbd \land$
$m = length\ (filter\ id\ lbd)\}$⟩

Testing if a level is marked **definition** *level-in-lbd* :: ⟨$nat \Rightarrow lbd \Rightarrow bool$⟩ **where**
⟨$level\text{-}in\text{-}lbd\ i = (\lambda lbd.\ i < length\ lbd \land lbd!i)$⟩

definition *level-in-lbd-ref* :: ⟨$nat \Rightarrow lbd\text{-}ref \Rightarrow bool$⟩ **where**
⟨$level\text{-}in\text{-}lbd\text{-}ref = (\lambda i\ (lbd,\ \text{-}).\ i < length\text{-}uint32\text{-}nat\ lbd \land lbd!i)$⟩

lemma *level-in-lbd-ref-level-in-lbd*:
⟨$(uncurry\ (RETURN\ oo\ level\text{-}in\text{-}lbd\text{-}ref),\ uncurry\ (RETURN\ oo\ level\text{-}in\text{-}lbd)) \in$
$nat\text{-}rel \times_r lbd\text{-}ref \rightarrow_f \langle bool\text{-}rel\rangle nres\text{-}rel$⟩
by (*intro frefI nres-relI*) (*auto simp*: *level-in-lbd-ref-def level-in-lbd-def lbd-ref-def*)

Marking more levels **definition** *list-grow* **where**
⟨$list\text{-}grow\ xs\ n\ x = xs\ @\ replicate\ (n - length\ xs)\ x$⟩

definition *lbd-write* :: ⟨$lbd \Rightarrow nat \Rightarrow lbd$⟩ **where**
⟨$lbd\text{-}write = (\lambda lbd\ i.$
$(if\ i < length\ lbd\ then\ (lbd[i := True])$
$else\ ((list\text{-}grow\ lbd\ (i + 1)\ False)[i := True])))$⟩

definition *lbd-ref-write* :: ⟨$lbd\text{-}ref \Rightarrow nat \Rightarrow lbd\text{-}ref\ nres$⟩ **where**
⟨$lbd\text{-}ref\text{-}write = (\lambda(lbd,\ m,\ n)\ i.\ do\ \{$
$ASSERT(length\ lbd \le uint\text{-}max \land n + 1 \le uint\text{-}max);$
$(if\ i < length\text{-}uint32\text{-}nat\ lbd\ then$
$let\ n = if\ lbd\ !\ i\ then\ n\ else\ n{+}one\text{-}uint32\text{-}nat\ in$
$RETURN\ (lbd[i := True],\ max\ i\ m,\ n)$
$else\ do\ \{$
$ASSERT(i + 1 \le uint\text{-}max);$
$RETURN\ ((list\text{-}grow\ lbd\ (i + one\text{-}uint32\text{-}nat)\ False)[i := True],\ max\ i\ m,\ n + one\text{-}uint32\text{-}nat)$
$\})$
$\})$⟩

lemma *length-list-grow*[*simp*]:
⟨$length\ (list\text{-}grow\ xs\ n\ a) = max\ (length\ xs)\ n$⟩
by (*auto simp*: *list-grow-def*)

lemma *list-update-append2*: ⟨$i \ge length\ xs \Longrightarrow (xs\ @\ ys)[i := x] = xs\ @\ ys[i - length\ xs := x]$⟩
by (*induction xs arbitrary*: *i*) (*auto split*: *nat.splits*)

lemma *lbd-ref-write-lbd-write*:
⟨$(uncurry\ (lbd\text{-}ref\text{-}write),\ uncurry\ (RETURN\ oo\ lbd\text{-}write)) \in$
$[\lambda(lbd,\ i).\ i \le Suc\ (uint\text{-}max\ div\ 2)]_f$
$lbd\text{-}ref \times_f nat\text{-}rel \rightarrow \langle lbd\text{-}ref\rangle nres\text{-}rel$⟩
unfolding *lbd-ref-write-def lbd-write-def*
by (*intro frefI nres-relI*)
(*auto simp*: *level-in-lbd-ref-def level-in-lbd-def lbd-ref-def list-grow-def*
nth-append uint-max-def length-filter-update-true list-update-append2
length-filter-update-false
intro!: *ASSERT-leI le-trans*[*OF length-filter-le*])

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 −
have *m-xs*: ‹*m* ≤ *length xs*› **and** [*simp*]: ‹*xs* ≠ []› **and** *le-xs*: ‹*length xs* ≤ *uint-max div 2* + *2*›
using *assms* **by** (*auto simp*: *lbd-ref-def*)
have [*iff*]: ‹(∀ *j*. ¬ *j* < (*b* :: *nat*)) ⟷ *b* = *0*› **for** *b*
by *auto*
have *init*: ‹*lbd-emtpy-inv m* (*xs*, *zero-uint32-nat*)›
using *assms m-xs* **unfolding** *lbd-emtpy-inv-def*
by (*auto simp*: *lbd-ref-def*)
have *lbd-remove*: ‹*lbd-emtpy-inv m*
(*a*[*b* := *False*], *b* + *one-uint32-nat*)›
if
inv: ‹*lbd-emtpy-inv m s*› **and**
‹*case s of* (*ys*, *i*) ⇒ *length ys* = *length xs*› **and**
cond: ‹*case s of* (*xs*, *i*) ⇒ *i* ≤ *m*› **and**
s: ‹*s* = (*a*, *b*)› **and**
[*simp*]: ‹*b* < *length a*›
for *s a b*
proof −
have *1*: ‹*a*[*b* := *False*] ! *j* = *False*› **if** ‹*j*<*b*› **for** *j*
using *inv that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have *2*: ‹∀ *j*>*m*. *j* < *length* (*a*[*b* := *False*]) ⟶ *a*[*b* := *False*] ! *j* = *False*›
using *inv that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have ‹*b* + *one-uint32-nat* ≤ *Suc m*›
using *cond* **unfolding** *s* **by** *simp*
moreover have ‹*a*[*b* := *False*] ! *j* = *False*› **if** ‹*j*<*b* + *one-uint32-nat*› **for** *j*
using *1*[*of j*] *that cond* **by** (*cases* ‹*j* = *b*›) *auto*
moreover have ‹∀ *j*>*m*. *j* < *length* (*a*[*b* := *False*]) ⟶ *a*[*b* := *False*] ! *j* = *False*›
using *2* **by** *auto*
ultimately show *?thesis*
unfolding *lbd-emtpy-inv-def* **by** *auto*

qed
have *lbd-final*: ‹((*a*, *zero-uint32-nat*, *zero-uint32-nat*), *replicate* (*length xs*) *False*) ∈ *lbd-ref*›
if
lbd: ‹*lbd-emtpy-inv m s*› **and**
I': ‹*case s of* (*ys*, *i*) ⇒ *length ys* = *length xs*› **and**
cond: ‹¬ (*case s of* (*xs*, *i*) ⇒ *i* ≤ *m*)› **and**
s: ‹*s* = (*a*, *b*)›
for *s a b*
proof −
have *1*: ‹*a*[*b* := *False*] ! *j* = *False*› **if** ‹*j*<*b*› **for** *j*
using *lbd that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have *2*: ‹*j*>*m* ⟶ *j* < *length a* ⟶ *a* ! *j* = *False*› **for** *j*
using *lbd that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have [*simp*]: ‹*length a* = *length xs*›
using *I'* **unfolding** *s* **by** *auto*
have [*simp*]: ‹*b* = *Suc m*›
using *cond lbd* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have [*dest*]: ‹*i* < *length xs* ⟹ ¬*a* ! *i*› **for** *i*
using *1*[*of i*] *2*[*of i*] **by** (*cases* ‹*i* < *Suc m*›) *auto*

have [*simp*]: ‹*a* = *replicate* (*length xs*) *False*›
unfolding *list-eq-iff-nth-eq*
apply (*intro conjI*)
subgoal by *simp*
subgoal by *auto*
done
show *?thesis*
using *le-xs* **by** (*auto simp*: *lbd-ref-def*)
qed
show *?thesis*
unfolding *lbd-empty-ref-def conc-fun-RETURN*
apply *clarify*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*xs*, *i*). *Suc m* − *i*)› **and**
I' = ‹λ(*ys*, *i*). *length ys* = *length xs*›])
subgoal by *auto*
subgoal by (*rule init*)
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *lbd-ref-def*)
subgoal using *m-xs le-xs* **by** (*auto simp*: *uint-max-def*)
subgoal by (*rule lbd-remove*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule lbd-final*)
done
qed

lemma *lbd-empty-ref-lbd-empty*:
‹(*lbd-empty-ref*, *lbd-empty*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *lbd m lbd'*
using *lbd-empty-ref*[*of lbd m*]
by (*auto simp*: *lbd-empty-def lbd-ref-def*)

done

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*›
by (*intro frefI nres-relI*) (*auto simp*: *empty-lbd-def lbd-ref-def empty-lbd-ref-def*
uint-max-def nth-Cons split: *nat.splits*)

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'*)›
unfolding *get-LBD-ref-def get-LBD-def*
by (*auto split*:*prod.splits*)

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *lbd m n lbd'*
using *get-LBD-ref*[*of lbd*]
by (*auto simp*: *lbd-empty-def lbd-ref-def*)
done

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*)›
:: ‹[λ(*n*, (*lbd*, *m*)). *length lbd* ≤ *uint-max*]$_a$
uint32-nat-assn^k $*_a$ *lbd-int-assn*^{k} → *bool-assn*›
unfolding *level-in-lbd-ref-def short-circuit-conv*
by *sepref*

lemma *level-in-lbd-hnr*[*sepref-fr-rules*]:

⟨(uncurry level-in-lbd-code, uncurry (RETURN ∘∘ level-in-lbd)) ∈ uint32-nat-assn^k $*_a$
lbd-assn^k $\to_a$ bool-assn⟩
supply *lbd-ref-def*[*simp*] *uint-max-def*[*simp*]
using *level-in-lbd-code.refine*[*FCOMP level-in-lbd-ref-level-in-lbd*[*unfolded convert-fref*]]
unfolding *lbd-assn-def* **.**

Marking more levels **lemma** *list-grow-array-hnr*[*sepref-fr-rules*]:
assumes ⟨*CONSTRAINT is-pure R*⟩
shows
⟨(uncurry2 (λxs u. array-grow xs (nat-of-uint32 u)),
uncurry2 (RETURN ooo list-grow)) ∈
[λ((xs, n), x). n ≥ length xs]$_a$ (array-assn R)^d $*_a$ uint32-nat-assn^d $*_a$ R^k →
array-assn R⟩
proof −
obtain R' **where** [*simp*]:
⟨R = pure R'⟩
⟨the-pure R = R'⟩
using *assms* **by** (*metis CONSTRAINT-D pure-the-pure*)
have [*simp*]: ⟨pure R' b bi = ↑((bi, b) ∈ R')⟩ **for** *b bi*
by (*auto simp*: *pure-def*)
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *list-grow-def array-assn-def is-array-def*
hr-comp-def list-rel-pres-length list-rel-def list-all2-replicate
uint32-nat-rel-def br-def
intro!: *list-all2-appendI*)
qed

sepref-definition *lbd-write-code*
is ⟨*uncurry lbd-ref-write*⟩
:: ⟨lbd-int-assn^d $*_a$ uint32-nat-assn^k $\to_a$ lbd-int-assn⟩
unfolding *lbd-ref-write-def*
by *sepref*

lemma *lbd-write-hnr-*[*sepref-fr-rules*]:
⟨(uncurry lbd-write-code, uncurry (RETURN ∘∘ lbd-write))
∈ [λ(lbd, i). i ≤ Suc (uint-max div 2)]$_a$
lbd-assn^d $*_a$ uint32-nat-assn^k → lbd-assn⟩
using *lbd-write-code.refine*[*FCOMP lbd-ref-write-lbd-write*[*unfolded convert-fref*]]
unfolding *lbd-assn-def* **.**

sepref-definition *lbd-empty-code*
is ⟨*lbd-empty-ref*⟩
:: ⟨lbd-int-assn^d $\to_a$ lbd-int-assn⟩
unfolding *lbd-empty-ref-def*
by *sepref*

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
⟨(lbd-empty-code, lbd-empty) ∈ lbd-assn^d $\to_a$ lbd-assn⟩
using *lbd-empty-code.refine*[*FCOMP lbd-empty-ref-lbd-empty*[*unfolded convert-fref*]]
unfolding *lbd-assn-def* **.**

sepref-definition *empty-lbd-code*
is ⟨*uncurry0 (RETURN empty-lbd-ref)*⟩
:: ⟨unit-assn^k $\to_a$ lbd-int-assn⟩
unfolding *empty-lbd-ref-def array-fold-custom-replicate*

by *sepref*

lemma *empty-lbd-hnr*[*sepref-fr-rules*]:
‹(*Sepref-Misc.uncurry0 empty-lbd-code*, *Sepref-Misc.uncurry0* (*RETURN empty-lbd*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *lbd-assn*›
using *empty-lbd-code.refine*[*FCOMP empty-lbd-ref-empty-lbd*[*unfolded convert-fref uncurry0-def*[*symmetric*]]]
unfolding *lbd-assn-def* **.**

sepref-definition *get-LBD-code*
is ‹*get-LBD-ref*›
:: ‹*lbd-int-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *get-LBD-ref-def*
by *sepref*

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) ∈ *lbd-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
using *get-LBD-code.refine*[*FCOMP get-LBD-ref-get-LBD*[*unfolded convert-fref*],
unfolded lbd-assn-def[*symmetric*]] **.**

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
by(auto simp add: numeral-eq-Suc)

lemma bin-nth2-32-iff: ‹bin-nth 4294967295 na ⟷ na < 32›
  by (auto simp: bin-nth-Bit1 bin-nth-Bit0 nat-less-numeral-unfold)

lemma take-only-lower32-le-uint32-max:
  ‹nat-of-uint64 n ≤ uint32-max ⟹ take-only-lower32 n = n›
  unfolding take-only-lower32-def
  apply transfer
  by (auto intro!: and-eq-bits-eqI dest: unat-le-uint32-max-no-bit-set
    intro!: bin-nth2-32-iff[THEN iffD2])


lemma uint32-of-uint64-uint64-of-uint32[simp]: ‹uint32-of-uint64 (uint64-of-uint32 n) = n›
  by (auto simp: uint64-of-uint32-def uint32-of-uint64-def
    nat-of-uint64-uint64-of-nat-id nat-of-uint32-le-uint32-max nat-of-uint32-le-uint64-max)

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›
  unfolding take-only-lower32-def
  apply transfer
  subgoal for m n
    using ex-rbl-word64-le-uint32-max[of m] ex-rbl-word64-ge-uint32-max[of n]
    apply (auto intro: and-eq-bits-eqI simp: bin-nth2-32-iff word-add-rbl
      dest: unat-le-uint32-max-no-bit-set)
    apply (rule word-bl.Rep-inject[THEN iffD1])
    apply (auto simp del: word-bl.Rep-inject simp: bl-word-and word-add-rbl
      split!: if-splits)
    done
  done

lemma take-only-lower32-1-32: ‹take-only-lower32 (1 << 32) = 0›
  unfolding take-only-lower32-def
  by transfer (auto simp: )

lemma nat-of-uint64-1-32: ‹nat-of-uint64 (1 << 32) = uint32-max + 1›
  unfolding uint32-max-def
  by transfer auto

lemma watcher-enc-extract-blit:
  assumes ‹(n, (L, b)) ∈ watcher-enc›
  shows ‹(uint32-of-uint64 (take-only-lower32 n), L) ∈ unat-lit-rel›
```

using *assms*
by (*auto simp*: *watcher-enc-def take-only-lower32-le-uint32-max nat-of-uint64-uint64-of-uint32 nat-of-uint32-le-uint32-max nat-of-uint64-1-32 take-only-lower32-1-32 take-only-lower32-le-uint32-max-ge-uint32-max*)

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*)›
by *auto*

lemma *AND-2-32-bool*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n* + (*1* << *32*) *AND 4294967296* = *4294967296*›
apply *transfer*
subgoal for *n*
using *ex-rbl-word64-ge-uint32-max*[*of* ‹*1* << *32*›] *ex-rbl-word64-le-uint32-max*[*of n*]
apply (*auto intro*: *and-eq-bits-eqI simp*: *bin-nth2-32-iff word-add-rbl dest*: *unat-le-uint32-max-no-bit-set*)
apply (*rule word-bl.Rep-inject*[*THEN iffD1*])
apply (*auto simp del*: *word-bl.Rep-inject simp*: *bl-word-and word-add-rbl split*!: *if-splits*)
done
done

lemma *watcher-enc-extract-bool-True*:
assumes ‹(*n*, (*L*, *True*)) ∈ *watcher-enc*›
shows ‹*n AND 4294967296* = *4294967296*›
using *assms*
by (*auto simp*: *watcher-enc-def take-only-lower32-le-uint32-max nat-of-uint64-uint64-of-uint32 nat-of-uint32-le-uint32-max nat-of-uint64-1-32 take-only-lower32-1-32 AND-2-32-bool take-only-lower32-le-uint32-max-ge-uint32-max*)

lemma *le-uint32-max-AND2-32-eq0*: ‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n AND 4294967296* = *0*›
apply *transfer*
subgoal for *n*
using *ex-rbl-word64-le-uint32-max*[*of n*]
apply (*auto intro*!:)
apply (*rule word-bl.Rep-inject*[*THEN iffD1*])
apply (*auto simp del*: *word-bl.Rep-inject simp*: *bl-word-and word-add-rbl split*!: *if-splits*)
done
done

lemma *watcher-enc-extract-bool-False*:
assumes ‹(*n*, (*L*, *False*)) ∈ *watcher-enc*›
shows ‹(*n AND 4294967296* = *0*)›

using *assms*
by (*auto simp*: *watcher-enc-def take-only-lower32-le-uint32-max nat-of-uint64-uint64-of-uint32 nat-of-uint32-le-uint32-max nat-of-uint64-1-32 take-only-lower32-1-32 AND-2-32-bool take-only-lower32-le-uint32-max-ge-uint32-max le-uint32-max-AND2-32-eq0*)

lemma *watcher-enc-extract-bool*:
assumes ‹$(n, (L, b)) \in$ *watcher-enc* ›
shows ‹$b \longleftrightarrow (n$ *AND* $4294967296 \neq 0)$›
using *assms*
supply [[*show-sorts*]]
by (*cases b*)
(*auto dest*!: *watcher-enc-extract-bool-False watcher-enc-extract-bool-True*)

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*)))›
by (*auto simp*: *shiftl-integer-conv-mult-pow2 to-watcher-code-def shiftl-t2n-uint64*)

lemma *OR-int64-0*[*simp*]: ‹*A OR* (*0* :: *uint64*) = *A*›
by *transfer auto*

lemma *OR-132-is-sum*:
‹*nat-of-uint64 n* ≤ *uint32-max* ⟹ *n OR* (*1* << *32*) = *n* + (*1* << *32*)›
apply *transfer*
subgoal for *n*
using *ex-rbl-word64-le-uint32-max*[*of n*]
apply (*auto intro*: *and-eq-bits-eqI simp*: *bin-nth2-32-iff word-add-rbl dest*: *unat-le-uint32-max-no-bit-set*)
apply (*rule word-bl.Rep-inject*[*THEN iffD1*])
by (*auto simp del*: *word-bl.Rep-inject simp*: *bl-word-or word-add-rbl split*!: *if-splits*)
done

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*⟩
by (*auto simp*: *take-only-lower32-def shiftl-t2n-uint64*)

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} $\rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare*
(*sep-auto dest*: *watcher-enc-extract-bool watcher-enc-extract-bool-True*)

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*⟩
by (*auto simp*: *nat-ann-lit-rel-def pure-def*)

lemma *blit-of-code-hnr*:
⟨(*return o blit-of-code*, *RETURN o blit-of*) ∈ *watcher-assn*^{k} $\rightarrow_a$ *unat-lit-assn*⟩
by *sepref-to-hoare*
(*sep-auto simp*: *watcher-enc-extract-blit*)

lemma *watcher-of-code-hnr*[*sepref-fr-rules*]:
⟨(*return o watcher-of-code*, *RETURN o watcher-of*) ∈
watcher-assn^k $\rightarrow_a$ (*nat-assn* ∗*a unat-lit-assn* ∗*a bool-assn*)⟩
by *sepref-to-hoare*
(*sep-auto dest*: *watcher-enc-extract-bool watcher-enc-extract-bool-True watcher-enc-extract-blit*
simp: *watcher-of-code-def*)

lemma *watcher-of-fast-code-hnr*[*sepref-fr-rules*]:
⟨(*return o watcher-of-fast-code*, *RETURN o watcher-of*) ∈
watcher-fast-assn^k $\rightarrow_a$ (*uint64-nat-assn* ∗*a unat-lit-assn* ∗*a bool-assn*)⟩
by *sepref-to-hoare*
(*sep-auto dest*: *watcher-enc-extract-bool watcher-enc-extract-bool-True watcher-enc-extract-blit*
simp: *watcher-of-fast-code-def*)

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} $\rightarrow_a$ *watcher-assn*›
by *sepref-to-hoare*
(*sep-auto dest*: *watcher-enc-extract-bool watcher-enc-extract-bool-True watcher-enc-extract-blit*
simp: *to-watcher-code-def watcher-enc-def OR-132-is-sum nat-of-uint64-uint64-of-uint32*
nat-of-uint32-le-uint32-max)

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} $\rightarrow_a$ *watcher-fast-assn*›
by *sepref-to-hoare*
(*sep-auto dest*: *watcher-enc-extract-bool watcher-enc-extract-bool-True watcher-enc-extract-blit*
simp: *to-watcher-fast-code-def watcher-enc-def OR-132-is-sum nat-of-uint64-uint64-of-uint32*
nat-of-uint32-le-uint32-max)

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*›
 by (*induction rule*: *mset-as-position.induct*) *auto*

lemma *mset-as-position-atm-le-length*:
 ‹*mset-as-position xs P* ⟹ $L \in\# P$ ⟹ *atm-of L* < *length xs*›
 by (*induction rule*: *mset-as-position.induct*) (*auto simp*: *nth-list-update' atm-of-eq-atm-of*)

lemma *mset-as-position-nth*:
 ‹*mset-as-position xs P* ⟹ $L \in\# P$ ⟹ *xs* ! (*atm-of L*) = *Some* (*is-pos L*)›
 by (*induction rule*: *mset-as-position.induct*)
  (*auto simp*: *nth-list-update' atm-of-eq-atm-of dest*: *mset-as-position-atm-le-length*)

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*)›
 by (*induction rule*: *mset-as-position.induct*)
  (*auto simp*: *nth-list-update' atm-of-eq-atm-of is-pos-neg-not-is-pos*
   dest: *mset-as-position-atm-le-length*)

lemma *mset-as-position-tautology*: ‹*mset-as-position xs C* ⟹ ¬*tautology C*›
 by (*induction rule*: *mset-as-position.induct*) (*auto simp*: *tautology-add-mset*)

lemma *mset-as-position-right-unique*:
 assumes
  map: ‹*mset-as-position xs D*› **and**
  map': ‹*mset-as-position xs D'*›
 shows ‹*D* = *D'*›
proof (*rule distinct-set-mset-eq*)
 show ‹*distinct-mset D*›
  using *mset-as-position-distinct-mset*[*OF map*] **.**
 show ‹*distinct-mset D'*›
  using *mset-as-position-distinct-mset*[*OF map'*] **.**
 show ‹*set-mset D* = *set-mset D'*›
  using *mset-as-position-in-iff-nth*[*OF map*] *mset-as-position-in-iff-nth*[*OF map'*]
   mset-as-position-atm-le-length[*OF map*] *mset-as-position-atm-le-length*[*OF map'*]
  by *blast*
qed

lemma *mset-as-position-mset-union*:
 fixes *P xs*
 defines ‹*xs'* ≡ *fold* (λ*L xs*. *xs*[*atm-of L* := *Some* (*is-pos L*)]) *P xs*›

```
  assumes
    mset: ‹mset-as-position xs P'› and
    atm-P-xs: ‹∀ L ∈ set P. atm-of L < length xs› and
    uL-P: ‹∀ L ∈ set P. −L ∉# P'› and
    dist: ‹distinct P› and
    tauto: ‹¬tautology (mset P)›
  shows ‹mset-as-position xs' (mset P ∪# P')›
  using atm-P-xs uL-P dist tauto unfolding xs'-def
proof (induction P)
  case Nil
  show ?case using mset by auto
next
  case (Cons L P) note IH = this(1) and atm-P-xs = this(2) and uL-P = this(3) and dist = this(4)
    and tauto = this(5)
  then have mset:
    ‹mset-as-position (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) (mset P ∪# P')›
    by (auto simp: tautology-add-mset)
  show ?case
  proof (cases ‹L ∈# P'›)
    case True
    then show ?thesis
      using mset dist
      by (metis (no-types, lifting) add-mset-union assms(2) distinct.simps(2) fold-simps(2)
       insert-DiffM list-update-id mset.simps(2) mset-as-position-nth set-mset-mset
       sup-union-right1)
  next
    case False
    have [simp]: ‹length (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) = length xs›
      by (induction P arbitrary: xs) auto
    moreover have ‹− L ∉ set P›
      using tauto by (metis list.set-intros(1) list.set-intros(2) set-mset-mset tautology-minus)
    moreover have
      ‹fold (λL xs. xs[atm-of L := Some (is-pos L)]) P (xs[atm-of L := Some (is-pos L)]) =
       (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) [atm-of L := Some (is-pos L)]›
      using uL-P dist tauto
      apply (induction P arbitrary: xs)
      subgoal by auto
      subgoal for L' P
        by (cases ‹atm-of L = atm-of L'›)
          (auto simp: tautology-add-mset list-update-swap atm-of-eq-atm-of)
      done
    ultimately show ?thesis
      using mset atm-P-xs dist uL-P False by (auto intro!: mset-as-position.add)
  qed
qed

lemma mset-as-position-empty-iff: ‹mset-as-position xs {#} ⟷ (∃ n. xs = replicate n None)›
  apply (rule iffI)
  subgoal
    by (cases rule: mset-as-position.cases, assumption) auto
  subgoal
    by (auto intro: mset-as-position.intros)
  done

type-synonym (in −) lookup-clause-rel = ‹nat × bool option list›
```

definition *lookup-clause-rel* :: ‹*nat multiset* ⇒ (*lookup-clause-rel* × *nat literal multiset*) *set*› **where**
‹*lookup-clause-rel* $\mathcal{A}$ = {((*n*, *xs*), *C*). *n* = *size C* ∧ *mset-as-position xs C* ∧
(∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*)}›

lemma *lookup-clause-rel-empty-iff*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *n* = *0* ⟷ *C* = {#}›
by (*auto simp*: *lookup-clause-rel-def*)

lemma *conflict-atm-le-length*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
L < *length xs*›
by (*auto simp*: *lookup-clause-rel-def*)

lemma *conflict-le-length*:
assumes
c-rel: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹*atm-of L* < *length xs*›
proof −
have
size: ‹*n* = *size C*› **and**
mset-pos: ‹*mset-as-position xs C*› **and**
atms-le: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*›
using *c-rel* **unfolding** *lookup-clause-rel-def* **by** *blast+*
have ‹*atm-of L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L-*$\mathcal{L}_{all}$ **by** (*simp add*: *atms-of-def*)
then show *?thesis*
using *atms-le* **by** *blast*
qed

lemma *lookup-clause-rel-atm-in-iff*:
‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *L* ∈# *C* ⟷ *xs*!(*atm-of L*) = *Some* (*is-pos L*)›
by (*rule mset-as-position-in-iff-nth*)
(*auto simp*: *lookup-clause-rel-def atms-of-def*)

lemma
assumes
c: ‹((*n*,*xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
lookup-clause-rel-not-tautolgy: ‹¬*tautology C*› **and**
lookup-clause-rel-distinct-mset: ‹*distinct-mset C*› **and**
lookup-clause-rel-size: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹ *size C* ≤ *1* + *uint-max div 2*›
proof −
have *mset*: ‹*mset-as-position xs C*› **and** ‹*n* = *size C*› **and** ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*›
using *c* **unfolding** *lookup-clause-rel-def* **by** *fast+*
show ‹¬*tautology C*›
using *mset*
apply (*induction rule*: *mset-as-position.induct*)
subgoal by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def*)
subgoal by (*auto simp*: *tautology-add-mset*)
done
show ‹*distinct-mset C*›
using *mset mset-as-position-distinct-mset* **by** *blast*
then show ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹ *size C* ≤ *1* + *uint-max div 2*›
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*C*›] ‹¬*tautology C*› *bounded* **by** *auto*

qed

type-synonym *lookup-clause-assn* = ⟨*uint32* × *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* $\mathcal{A}$⟩
unfolding *option-lookup-clause-rel-def* **by** *auto*

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*⟩
by (*intro nres-relI frefI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-None-def split*: *option.splits*)

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*⟩
by (*intro nres-relI frefI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-empty-def lookup-clause-rel-def*
Multiset.is-empty-def split: *option.splits*)

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 (*induction rule*: *mset-as-position.induct*)
case (*empty n*)
then show *?case* **by** *auto*
next
case (*add xs P L xs'*) **note** *m-as-p* = *this*(*1*) **and** *atm-L* = *this*(*2*)
have *xs-L*: ‹*xs* ! (*atm-of L*) = *None*›
proof −
obtain *bb* :: ‹*bool option* ⇒ *bool*› **where**
f1: ‹∀ *z*. *z* = *None* ∨ *z* = *Some* (*bb z*)›
by (*metis option.exhaust*)
have *f2*: ‹*xs* ! *atm-of L* ≠ *Some* (*is-pos L*)›
using *add.hyps*(*1*) *add.hyps*(*2*) *add.hyps*(*3*) *mset-as-position-in-iff-nth* **by** *blast*
have *f3*: ‹∀ *z b*. ((*Some b* = *z* ∨ *z* = *None*) ∨ *bb z*) ∨ *b*›
using *f1* **by** *blast*
have *f4*: ‹∀ *zs*. (*zs* ! *atm-of L* ≠ *Some* (*is-pos* (− *L*)) ∨ ¬ *atm-of L* < *length zs*)
∨ ¬ *mset-as-position zs P*›
by (*metis add.hyps*(*4*) *atm-of-uminus mset-as-position-in-iff-nth*)
have ‹∀ *z b*. ((*Some b* = *z* ∨ *z* = *None*) ∨ ¬ *bb z*) ∨ ¬ *b*›
using *f1* **by** *blast*
then show *?thesis*
using *f4 f3 f2* **by** (*metis add.hyps*(*1*) *add.hyps*(*2*) *is-pos-neg-not-is-pos*)
qed
obtain *xs1 xs2* **where**
xs-xs12: ‹*xs* = *xs1* @ *None* # *xs2*› **and**
xs1: ‹*length xs1* = *atm-of L*›
using *atm-L upd-conv-take-nth-drop*[*of* ‹*atm-of L*› *xs* ‹*None*›] **apply** −
apply (*subst*(*asm*)(*2*) *xs-L*[*symmetric*])
by (*force simp del*: *append-take-drop-id*)+
then show *?case*
using *add* **by** (*auto simp*: *list-update-append*)
qed

definition (**in** −) *is-in-lookup-conflict* **where**
‹*is-in-lookup-conflict* = (λ(*n*, *xs*) *L*. ¬*is-None* (*xs* ! *atm-of L*))›

lemma *mset-as-position-remove*:
‹*mset-as-position xs D* ⟹ *L* < *length xs* ⟹
mset-as-position (*xs*[*L* := *None*]) (*remove1-mset* (*Pos L*) (*remove1-mset* (*Neg L*) *D*))›
proof (*induction rule*: *mset-as-position.induct*)
case (*empty n*)
then have [*simp*]: ‹(*replicate n None*)[*L* := *None*] = *replicate n None*›
using *list-update-id*[*of* ‹*replicate n None*› *L*] **by** *auto*
show *?case* **by** (*auto intro*: *mset-as-position.intros*)
next
case (*add xs P K xs'*)
show *?case*
proof (*cases* ‹*L* = *atm-of K*›)
case *True*
then show *?thesis*
using *add* **by** (*cases K*) *auto*

next
  case *False*
  have *map*: ‹*mset-as-position* (*xs*[*L* := *None*]) (*remove1-mset* (*Pos L*) (*remove1-mset* (*Neg L*) *P*))›
    using *add* **by** *auto*
  have ‹$K \notin\# P - \{\#Pos\ L,\ Neg\ L\#\}$› ‹$-K \notin\# P - \{\#Pos\ L,\ Neg\ L\#\}$›
    by (*auto simp*: *add.hyps dest*!: *in-diffD*)
  then show *?thesis*
    using *mset-as-position.add*[*OF map*, *of* ‹*K*› ‹*xs*[*L* := *None*, *atm-of K* := *Some* (*is-pos K*)]›]
      add False list-update-swap[*of* ‹*atm-of K*› *L xs*] **apply** *simp*
    apply (*subst diff-add-mset-swap*)
    by *auto*
qed
qed

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($n \geq 1$);
    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*)) ∈
    $[\lambda(L,\ D).\ -L \notin\# D \wedge L \in\# \mathcal{L}_{all}\ \mathcal{A} \wedge L \in\# D]_f\ Id \times_f$ *lookup-clause-rel* $\mathcal{A}$ →
    ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
  using *mset-as-position-remove*[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*atm-of* (*fst y*)›]
  apply (*cases x*; *cases y*; *cases* ‹*fst y*›)
  by (*auto simp*: *delete-from-lookup-conflict-def lookup-clause-rel-def*
      dest!: *multi-member-split*
      intro!: *ASSERT-refine-left*)
done

definition *delete-from-lookup-conflict-pre* **where**
‹*delete-from-lookup-conflict-pre* $\mathcal{A} = (\lambda(a,\ b).\ -\ a \notin\# b \wedge a \in\# \mathcal{L}_{all}\ \mathcal{A} \wedge a \in\# b)$›

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* ($N \propto i$)) ∧ *n* = *card-max-lvl M* (*mset* ($N \propto i$)) ∧
     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* ($N \propto i$)) ∪# *the D*) ∧
      n = *card-max-lvl M* (*mset* (*tl* ($N \propto i$)) ∪# *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 −
have [*simp*]: ⟨*L* ∉# *A* ⟹ *A* − *add-mset L′* (*add-mset L B*) = *A* − *add-mset L′ B*⟩
for *A B* :: ⟨*′a multiset*⟩ **and** *L L′*
by (*metis add-mset-commute minus-notin-trivial*)
have ⟨*n* = *size D*⟩ **and** *map*: ⟨*mset-as-position xs D*⟩
using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)
have *xs-None-iff*: ⟨*xs* ! *L* = *None* ⟷ *Pos L* ∉# *D* ∧ *Neg L* ∉# *D*⟩
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ⟨*Pos L*⟩]
mset-as-position-in-iff-nth[*of xs D* ⟨*Neg L*⟩]
by (*cases* ⟨*xs* ! *L*⟩) *auto*

have *1*: ⟨*xs* ! *L* = *None* ⟹ *D* − {#*Pos L*, *Neg L*#} = *D*⟩
using *assms* **by** (*auto simp*: *xs-None-iff minus-notin-trivial*)
have *2*: ⟨*xs* ! *L* = *None* ⟹ *xs*[*L* := *None*] = *xs*⟩
using *map list-update-id*[*of xs L*] **by** (*auto simp*: *1*)
have *3*: ⟨*xs* ! *L* = *Some y* ⟷ (*y* ∧ *Pos L* ∈# *D* ∧ *Neg L* ∉# *D*) ∨ (¬*y* ∧ *Pos L* ∉# *D* ∧ *Neg L* ∈#
D)⟩
for *y*
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ⟨*Pos L*⟩]
mset-as-position-in-iff-nth[*of xs D* ⟨*Neg L*⟩]
by (*cases* ⟨*xs* ! *L*⟩) *auto*

show *?thesis*
using *assms mset-as-position-remove*[*of xs D L*]
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def 1 2 3 size-remove1-mset-If*
minus-notin-trivial mset-as-position-remove)
qed

lemma *add-to-lookup-conflict-lookup-clause-rel*:
  assumes
    confl: ‹$((n, xs), C) \in$ *lookup-clause-rel* $\mathcal{A}$› **and**
    uL-C: ‹$-L \notin\# C$› **and**
    L-$\mathcal{L}_{all}$: ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$›
  shows ‹(*add-to-lookup-conflict* L (n, xs), $\{\#L\#\} \cup\# C) \in$ *lookup-clause-rel* $\mathcal{A}$›
proof −
  have
    n: ‹$n =$ *size* C› **and**
    mset: ‹*mset-as-position xs C*› **and**
    atm: ‹$\forall L \in$*atms-of* $(\mathcal{L}_{all}\ \mathcal{A})$. $L <$ *length xs*›
    using *confl* **unfolding** *lookup-clause-rel-def* **by** *blast+*
  have ‹*distinct-mset C*›
    using *mset* **by** (*blast dest*: *mset-as-position-distinct-mset*)
  have *atm-L-xs*: ‹*atm-of* $L <$ *length xs*›
    using *atm* *L-$\mathcal{L}_{all}$* **by** (*auto simp*: *atms-of-def*)
  then show *?thesis*
  proof (*cases* ‹$L \in\# C$›)
    case *True*
    with *mset* **have** *xs*: ‹*xs* ! *atm-of* $L =$ *Some* (*is-pos* L)› ‹*xs* ! *atm-of* $L \neq$ *None*›
      by (*auto dest*: *mset-as-position-nth*)
    moreover have ‹$\{\#L\#\} \cup\# C = C$›
      using *True* **by** (*simp add*: *subset-mset.sup.absorb2*)
    ultimately show *?thesis*
      using *n mset atm True*
      by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def xs*[*symmetric*])
  next
    case *False*
    with *mset* **have** ‹*xs* ! *atm-of* $L =$ *None*›
      using *mset-as-position-in-iff-nth*[*of xs C L*]
       mset-as-position-in-iff-nth[*of xs C* ‹$-L$›] *atm-L-xs uL-C*
      by (*cases L*; *cases* ‹*xs* ! *atm-of L*›) *auto*
    then show *?thesis*
      using *n mset atm False atm-L-xs uL-C*
      by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def*
          intro!: *mset-as-position.intros*)
  qed
qed

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$$^{\lambda(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*›
by (*auto simp*: *isa-outlearned-add-def outlearned-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

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*›
by (*auto simp*: *isa-clvls-add-def clvls-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

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*) ← $WHILE_T{}^{\lambda(i::nat,\ clvls\ ::\ nat,\ zs,\ lbd,\ outl).}$ *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 −
have [*refine0*]: ‹((*i* + *init*, *clvls*, *xs*, *lbd*, *outl*), *init*, *clvls*, *xs*, *lbd*, *outl*) ∈
{(*k*, *l*). *k* = *l* + *i*} $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* $\times_r$ *Id*›
by *auto*
have ‹*no-dup M*›
using *assms* **by** (*auto simp*: *trail-pol-def*)
have ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
using *M'M* **by** (*auto simp*: *trail-pol-def literals-are-in-*$\mathcal{L}_{in}$*-trail-def*)
from *literals-are-in-*$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*OF bound this* ‹*no-dup M*›]
have *lev-le*: ‹*get-level M L* ≤ *Suc* (*uint32-max div 2*)› **for** *L* **.**

show *?thesis*
unfolding *isa-lookup-conflict-merge-def lookup-conflict-merge-def prod.case*
apply *refine-vcg*
subgoal using *assms* **unfolding** *arena-is-valid-clause-idx-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal using *valid i* **by** (*auto simp*: *arena-lifting*)
subgoal using *valid i* **by** (*auto simp*: *arena-lifting ac-simps*)
subgoal using *valid i*
by (*auto simp*: *arena-lifting arena-lit-pre-def arena-is-valid-clause-idx-and-access-def intro*!: *exI*[*of - i*])
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g*
using *i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits valid M′M*
by (*auto simp*: *arena-lifting ac-simps image-image intro*!: *get-level-pol-pre*)
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g′*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M′M*, *symmetric*] *isa-outlearned-add-outlearned-add*[*OF M′M*] *isa-clvls-add-clvls-add*[*OF M′M*] *lev-le*)
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g*
using *i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits valid M′M*
using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps*)
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g′*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M′M*] *isa-outlearned-add-outlearned-add*[*OF M′M*] *isa-clvls-add-clvls-add*[*OF M′M*])
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g′*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M′M*] *isa-outlearned-add-outlearned-add*[*OF M′M*] *isa-clvls-add-clvls-add*[*OF M′M*])
subgoal using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff get-level-get-level-pol*[*OF M′M*])
done
qed

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*›
using *arena-lifting*(*10*)[*of be - - bd*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

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* ≤
⇓(*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*merge-conflict-m-g init M D* (*Some C*))⟩
(**is** ⟨- ≤ ⇓ *?Ref ?Spec*⟩)
proof −
define *lbd-upd* **where**
⟨*lbd-upd lbd i* ≡ *lbd-write lbd* (*get-level M* (*D*!*i*))⟩ **for** *lbd i*
let *?D* = ⟨*drop init D*⟩
have *le-D-le-upper*[*simp*]: ⟨*a* < *length D* ⟹ *Suc* (*Suc a*) ≤ *uint-max*⟩ **for** *a*
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ⟨*mset D*⟩] *assms* **by** (*auto simp*: *uint-max-def*)
have *Suc-N-uint-max*: ⟨*Suc n* ≤ *uint-max*⟩ **and**
size-C-uint-max: ⟨*size C* ≤ *1* + *uint-max div 2*⟩ **and**
clvls: ⟨*clvls* = *card-max-lvl M C*⟩ **and**
tauto-C: ⟨¬ *tautology C*⟩ **and**
dist-C: ⟨*distinct-mset C*⟩ **and**
atms-le-xs: ⟨∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*⟩ **and**
map: ⟨*mset-as-position xs C*⟩
using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ *C*] *mset-as-position-distinct-mset*[*of xs C*]
lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*auto simp*: *uint-max-def*)
then have *clvls-uint-max*: ⟨*clvls* ≤ *1* + *uint-max div 2*⟩
using *size-filter-mset-lesseq*[*of* ⟨λ*L*. *get-level M L* = *count-decided M*⟩ *C*]
unfolding *uint-max-def card-max-lvl-def* **by** *linarith*
have [*intro*]: ⟨((*b*, *a*, *ba*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹
Suc (*Suc a*) ≤ *uint-max*⟩ **for** *b a ba C*
using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def uint-max-def)
have [*simp*]: ⟨*remdups-mset C* = *C*⟩
using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def distinct-mset-remdups-mset-id)
have ⟨¬*tautology C*⟩
using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def)
have ⟨*distinct-mset C*⟩
using *mset-as-position-distinct-mset*[*of - C*] *o*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def* **by** *auto*
let *?C'* = ⟨λ*a*. (*mset* (*take* (*a* − *init*) (*drop init D*)) + *C*)⟩
have *tauto-C'*: ⟨¬ *tautology* (*?C' a*)⟩ **if** ⟨*a* ≥ *init*⟩ **for** *a*
using *that tauto tauto-C dist dist-C CD* **unfolding** *tautology-decomp'*
by (*force dest*: *in-set-takeD in-diffD dest*: *in-set-dropD*
simp: *distinct-mset-in-diff in-image-uminus-uminus*)

define *I* **where**
⟨*I xs* = (λ(*i*, *clvls*, *zs* :: *lookup-clause-rel*, *lbd* :: *lbd*, *outl* :: *out-learned*).
length (*snd zs*) =

length (*snd xs*) ∧
Suc i ≤ *uint-max* ∧
Suc (*fst zs*) ≤ *uint-max* ∧
Suc clvls ≤ *uint-max*)›
for *xs* :: *lookup-clause-rel*
define *I′* **where** ‹*I′* = (λ(*i*, *clvls*, *zs*, *lbd* :: *lbd*, *outl*).
lookup-conflict-merge′-step $\mathcal{A}$ *init M i clvls zs D C outl* ∧ *i* ≥ *init*)›

have *dist-D*: ‹*distinct-mset* (*mset D*)›
using *dist* **by** *auto*
have *dist-CD*: ‹*distinct-mset* (*C* − *mset D* − *uminus* ‘# *mset D*)›
using ‹*distinct-mset C*› **by** *auto*
have [*simp*]: ‹*remdups-mset* (*mset* (*drop init D*) + *C*) = *mset* (*drop init D*) ∪# *C*›
apply (*rule distinct-mset-rempdups-union-mset*[*symmetric*])
using *dist-C dist* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*)› **for** *a*
using *lits-C lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def all-lits-of-m-def*
dest!: *in-set-takeD in-set-dropD*)
then have *size-outl-le*: ‹*size* (*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*) ≤ *Suc uint32-max div 2*› **if**
‹*a* ≥ *init*› **for** *a*
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*›]
tauto-C′[*OF that*] ‹*distinct-mset C*› *dist-D*
by (*auto simp*: *uint32-max-def*)

have
if-True-I: ‹*I x2* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*, *lbd-upd lbd′ a*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I*) **and**
if-true-I′: ‹*I′* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*, *lbd-upd lbd′ a*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I′*)
if
I: ‹*I x2 s*› **and**
I′: ‹*I′ s*› **and**
cond: ‹*case s of* (*i*, *clvls*, *zs*, *lbd*, *outl*) ⇒ *i* < *length D*› **and**
s: ‹*s* = (*a*, *ba*)› ‹*ba* = (*aa*, *baa2*)› ‹*baa2* = (*baa*, *lbdL′*)› ‹(*b*, *n*, *xs*) = (*x1*, *x2*)›
‹*lbdL′* = (*lbd′*, *outl*)› **and**
a-le-D: ‹*a* < *length D*› **and**
a-uint-max: ‹*Suc a* ≤ *uint-max*›
for *x1 x2 s a ba aa baa baa2 lbd′ lbdL′ outl*
proof −
have [*simp*]:
‹*s* = (*a*, *aa*, *baa*, *lbd′*, *outl*)›
‹*ba* = (*aa*, *baa*, *lbd′*, *outl*)›
‹*x2* = (*n*, *xs*)›
using *s* **by** *auto*
obtain *ab b* **where** *baa*[*simp*]: ‹*baa* = (*ab*, *b*)› **by** (*cases baa*)

have *aa*: ‹*aa* = *card-max-lvl M* (*remdups-mset* (*?C′ a*))› **and**
ocr: ‹((*False*, *ab*, *b*), *Some* (*remdups-mset* (*?C′ a*))) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*remdups-mset* (*?C′ a*))› **and**
outl: ‹*out-learned M* (*Some* (*remdups-mset* (*?C′ a*))) *outl*›
using *I′*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def*
by *auto*
have

ab: ‹*ab* = *size* (*remdups-mset* (*?C′ a*))› **and**
map: ‹*mset-as-position b* (*remdups-mset* (*?C′ a*))› **and**
‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length b*› **and**
cr: ‹((*ab*, *b*), *remdups-mset* (*?C′ a*)) ∈ *lookup-clause-rel* $\mathcal{A}$›
using *ocr* **unfolding** *option-lookup-clause-rel-def lookup-clause-rel-def*
by *auto*
have *a-init*: ‹*a* ≥ *init*›
using *I′* **unfolding** *I′-def* **by** *auto*
have ‹*size* (*card-max-lvl M* (*remdups-mset* (*?C′ a*))) ≤ *size* (*remdups-mset* (*?C′ a*))›
unfolding *card-max-lvl-def*
by *auto*
then have [*simp*]: ‹*Suc* (*Suc aa*) ≤ *uint-max*› ‹*Suc aa* ≤ *uint-max*›
using *size-C-uint-max lits a-init*
simple-clss-size-upper-div2[*of* $\mathcal{A}$ ‹*remdups-mset* (*?C′ a*)›, *OF bounded*]
unfolding *uint-max-def aa*[*symmetric*]
by (*auto simp*: *tauto-C′*)
have [*simp*]: ‹*length b* = *length xs*›
using *I* **unfolding** *I-def* **by** *simp-all*

have *ab-upper*: ‹*Suc* (*Suc ab*) ≤ *uint-max*›
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*remdups-mset* (*?C′ a*)›]
lookup-clause-rel-not-tautolgy[*OF cr bounded*] *a-le-D lits mset-as-position-distinct-mset*[*OF map*]
unfolding *ab literals-are-in-*$\mathcal{L}_{in}$*-remdups uint-max-def* **by** *auto*
show *?I*
using *le-D-le-upper a-le-D ab-upper a-init*
unfolding *I-def add-to-lookup-conflict-def baa clvls-add-def* **by** *auto*

have *take-Suc-a*[*simp*]: ‹*take* (*Suc a* − *init*) *?D* = *take* (*a* − *init*) *?D* @ [*D* ! *a*]›
by (*smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop*
length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
have [*simp*]: ‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
using *dist tauto a-le-D* **apply** (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc a* − *init*›],
subst append-take-drop-id[*symmetric*, *of* - ‹*Suc a* − *init*›])
apply (*subst* (*asm*) *distinct-append*, *subst nth-append*)
by (*auto simp*: *in-set-distinct-take-drop-iff*)
have [*simp*]: ‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
proof
assume ‹− *D* ! *a* ∈ *set* (*take* (*a* − *init*) (*drop init D*))›
then have (*if is-pos* (*D* ! *a*) *then Neg else Pos*) (*atm-of* (*D* ! *a*)) ∈ *set D*
by (*metis* (*no-types*) *in-set-dropD in-set-takeD uminus-literal-def*)
then show *False*
using *a-le-D tauto* **by** *force*
qed

have *D-a-notin*: ‹*D* ! *a* ∉# (*mset* (*take* (*a* − *init*) *?D*) + *uminus* ‘# *mset* (*take* (*a* − *init*) *?D*))›
by (*auto simp*: *uminus-lit-swap*[*symmetric*])
have *uD-a-notin*: ‹−*D* ! *a* ∉# (*mset* (*take* (*a* − *init*) *?D*) + *uminus* ‘# *mset* (*take* (*a* − *init*) *?D*))›
by (*auto simp*: *uminus-lit-swap*[*symmetric*])

show *?I′*
proof (*cases* ‹(*get-level M* (*D* ! *a*) = *count-decided M* ∧ ¬ *is-in-lookup-conflict baa* (*D* ! *a*))›)
case *if-cond*: *True*
have [*simp*]: ‹*D* ! *a* ∉# *C*› ‹−*D* ! *a* ∉# *C*› ‹*b* ! *atm-of* (*D* ! *a*) = *None*›
using *if-cond mset-as-position-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-nth[*OF map*, *of* ‹−*D* ! *a*›] *D-a-notin uD-a-notin*
by (*auto simp*: *is-in-lookup-conflict-def split*: *option.splits bool.splits*

    dest: *in-diffD*)
  have [*simp*]: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
    using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
    by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)

  have *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)), *Some* (*remdups-mset* (*?C′* (*Suc a*))))
    ∈ *option-lookup-clause-rel* $\mathcal{A}$›
    using *ocr D-a-notin uD-a-notin*
    unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def*
    by (*auto dest*: *in-diffD simp*: *minus-notin-trivial*
      intro!: *mset-as-position.intros*)
  have ‹*out-learned M* (*Some* (*remdups-mset* (*?C′* (*Suc a*)))) (*outlearned-add M* (*D* ! *a*) (*ab*, *b*) *outl*)›
    using *D-a-notin uD-a-notin ocr lits if-cond a-init outl*
    unfolding *outlearned-add-def out-learned-def*
    by *auto*
  then show *?I′*
    using *D-a-notin uD-a-notin ocr lits if-cond a-init*
    unfolding *I′-def lookup-conflict-merge′-step-def Let-def clvls-add-def*
    by (*auto simp*: *minus-notin-trivial literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
      card-max-lvl-add-mset aa)
next
  case *if-cond*: *False*
  have *atm-D-a-le-xs*: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
    using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
    by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
  have [*simp*]: ‹*D* ! *a* ∉# *C* − *add-mset* (− *D* ! *a*)
      (*add-mset* (*D* ! *a*)
        (*mset* (*take a D*) + *uminus* ‘# *mset* (*take a D*)))›
    using *dist-C in-diffD*[*of* ‹*D* ! *a*› *C* ‹*add-mset* (− *D* ! *a*)
        (*mset* (*take a D*) + *uminus* ‘# *mset* (*take a D*))›,
      THEN multi-member-split]
    by (*meson distinct-mem-diff-mset member-add-mset*)
  have *a-init*: ‹*a* ≥ *init*›
    using *I′* **unfolding** *I′-def* **by** *auto*
  have *take-Suc-a*[*simp*]: ‹*take* (*Suc a* − *init*) *?D* = *take* (*a* − *init*) *?D* @ [*D* ! *a*]›
    by (*smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop*
      length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
  have [*iff*]: ‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
    using *dist tauto a-le-D*
    apply (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc a* − *init*›],
      subst append-take-drop-id[*symmetric*, *of* - ‹*Suc a* − *init*›])
    apply (*subst* (*asm*) *distinct-append*, *subst nth-append*)
    by (*auto simp*: *in-set-distinct-take-drop-iff*)
  have [*simp*]: ‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
  proof
    assume − *D* ! *a* ∈ *set* (*take* (*a* − *init*) (*drop init D*))
    then have (*if is-pos* (*D* ! *a*) *then Neg else Pos*) (*atm-of* (*D* ! *a*)) ∈ *set D*
      by (*metis* (*no-types*) *in-set-dropD in-set-takeD uminus-literal-def*)
    then show *False*
      using *a-le-D tauto* **by** *force*
  qed
  have ‹*D* ! *a* ∈ *set* (*drop init D*)›
    using *a-init a-le-D* **by** (*meson in-set-drop-conv-nth*)
  from *CD*[*OF this*] **have** [*simp*]: ‹−*D* ! *a* ∉# *C*› **.**
  consider
    (*None*) ‹*b* ! *atm-of* (*D* ! *a*) = *None*› |

(*Some-in*) *i* **where** ‹*b* ! *atm-of* (*D* ! *a*) = *Some i*› **and**
‹(*if i then Pos* (*atm-of* (*D* ! *a*)) *else Neg* (*atm-of* (*D* ! *a*))) ∈# *C*›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
by (*cases* ‹*b* ! *atm-of* (*D* ! *a*)›) (*auto simp*: *is-pos-neg-not-is-pos*)
then have *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)),
Some (*remdups-mset* (*?C′* (*Suc a*)))) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
proof *cases*
case [*simp*]: *None*
have [*simp*]: ‹*D* ! *a* ∉# *C*›
using *if-cond mset-as-position-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-nth[*OF map*, *of* ‹−*D* ! *a*›]
by (*auto simp*: *is-in-lookup-conflict-def split*: *option.splits bool.splits*
dest: *in-diffD*)
have [*simp*]: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)

show *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)),
Some (*remdups-mset* (*?C′* (*Suc a*)))) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
using *ocr*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def*
by (*auto dest*: *in-diffD simp*: *minus-notin-trivial*
intro!: *mset-as-position.intros*)
next
case *Some-in*
then have ‹*remdups-mset* (*?C′ a*) = *remdups-mset* (*?C′* (*Suc a*))›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›] *a-init*
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
by (*auto simp*: *is-neg-neg-not-is-neg*)
moreover
have *1*: ‹*Some i* = *Some* (*is-pos* (*D* ! *a*))›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›] *a-init Some-in*
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)› ‹−*D* ! *a* ∉# *C*›
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
by (*cases* ‹*D* ! *a*›) (*auto simp*: *is-neg-neg-not-is-neg*)
moreover have ‹*b*[*atm-of* (*D* ! *a*) := *Some i*] = *b*›
unfolding *1*[*symmetric*] *Some-in*(*1*)[*symmetric*] **by** *simp*
ultimately show *?thesis*
using *dist-C atms-le-xs Some-in*(*1*) *map*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def ab*
by (*auto simp*: *distinct-mset-in-diff minus-notin-trivial*
intro: *mset-as-position.intros*
simp del: *remdups-mset-singleton-sum*)
qed
have *notin-lo-in-C*: ‹¬*is-in-lookup-conflict* (*ab*, *b*) (*D* ! *a*) ⟹ *D* ! *a* ∉# *C*›
using *mset-as-position-in-iff-nth*[*OF map*, *of* ‹*Pos* (*atm-of* (*D*!*a*))›]
mset-as-position-in-iff-nth[*OF map*, *of* ‹*Neg* (*atm-of* (*D*!*a*))›] *atm-D-a-le-xs*(*1*)
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹−*D* ! *a* ∉# *C*› *a-init*
by (*cases* ‹*b* ! (*atm-of* (*D* ! *a*))›; *cases* ‹*D* ! *a*›)
(*auto simp*: *is-in-lookup-conflict-def dist-C distinct-mset-in-diff*
split: *option.splits bool.splits*
dest: *in-diffD*)

have *in-lo-in-C*: ‹*is-in-lookup-conflict* (*ab*, *b*) (*D* ! *a*) $\Longrightarrow$ *D* ! *a* ∈# *C*›
using *mset-as-position-in-iff-nth*[*OF map*, *of* ‹*Pos* (*atm-of* (*D*!*a*))›]
mset-as-position-in-iff-nth[*OF map*, *of* ‹*Neg* (*atm-of* (*D*!*a*))›] *atm-D-a-le-xs*(*1*)
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹−*D* ! *a* ∉# *C*› *a-init*
by (*cases* ‹*b* ! (*atm-of* (*D* ! *a*))›; *cases* ‹*D* ! *a*›)
(*auto simp*: *is-in-lookup-conflict-def dist-C distinct-mset-in-diff*
split: *option.splits bool.splits*
dest: *in-diffD*)
moreover have ‹*out-learned M* (*Some* (*remdups-mset* (*?C′* (*Suc a*))))
(*outlearned-add M* (*D* ! *a*) (*ab*, *b*) *outl*)›
using *D-a-notin uD-a-notin ocr lits if-cond a-init outl in-lo-in-C notin-lo-in-C*
unfolding *outlearned-add-def out-learned-def*
by *auto*
ultimately show *?I′*
using *ocr lits if-cond atm-D-a-le-xs a-init*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def clvls-add-def*
by (*auto simp*: *minus-notin-trivial literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
card-max-lvl-add-mset aa)
qed
qed
have *uL-C-if-L-C*: ‹−*L* ∉# *C*› **if** ‹*L* ∈# *C*› **for** *L*
using *tauto-C that* **unfolding** *tautology-decomp′* **by** *blast*

have *outl-le*: ‹*length bc* < *uint-max*›
if
‹*I x2 s*› **and**
‹*I′ s*› **and**
‹*s* = (*a*, *ba*)› **and**
‹*ba* = (*aa*, *baa*)› **and**
‹*baa* = (*ab*, *bb*)› **and**
‹*bb* = (*ac*, *bc*)› **for** *x1 x2 s a ba aa baa ab bb ac bc*
proof −
have ‹*mset* (*tl bc*) ⊆# (*remdups-mset* (*mset* (*take* (*a* −*init*) (*drop init D*)) + *C*))› **and** ‹*init* ≤ *a*›
using *that* **by** (*auto simp*: *I-def I′-def lookup-conflict-merge′-step-def Let-def out-learned-def*)
from *size-mset-mono*[*OF this*(*1*)] *this*(*2*) **show** *?thesis* **using** *size-outl-le*[*of a*] *dist-C dist-D*
by (*auto simp*: *uint32-max-def distinct-mset-rempdups-union-mset*)
qed
show *confl*: ‹*lookup-conflict-merge init M D* (*b*, *n*, *xs*) *clvls lbd outl*
≤ ⇓ *?Ref* (*merge-conflict-m-g init M D* (*Some C*))›
supply [[*goals-limit=1*]]
unfolding *resolve-lookup-conflict-aa-def lookup-conflict-merge-def*
distinct-mset-rempdups-union-mset[*OF dist-D dist-CD*] *I-def*[*symmetric*] *conc-fun-SPEC*
lbd-upd-def[*symmetric*] *Let-def length-uint32-nat-def merge-conflict-m-g-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*j*, -). *length D* − *j*)› **and**
I′ = *I′*])
subgoal by *auto*
subgoal
using *clvls-uint-max Suc-N-uint-max* ‹*Suc init* ≤ *uint-max*›
unfolding *uint-max-def I-def* **by** *auto*
subgoal using *assms*
unfolding *lookup-conflict-merge′-step-def Let-def option-lookup-clause-rel-def I′-def*
by (*auto simp add*: *uint-max-def lookup-conflict-merge′-step-def option-lookup-clause-rel-def*)
subgoal by *auto*
subgoal unfolding *I-def* **by** *fast*

subgoal for *x1 x2 s a ba aa baa ab bb ac bc* **by** (*rule outl-le*)
subgoal by (*rule if-True-I*)
subgoal by (*rule if-true-I′*)
subgoal for *b′ n′ s j zs*
using *dist lits tauto*
by (*auto simp*: *option-lookup-clause-rel-def take-Suc-conv-app-nth literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$)
subgoal using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-conflict-merge′-step-def Let-def I-def I′-def*)
done
qed

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* ∈# *dom-m N*⟩
shows ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* ($N \propto i$))⟩
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-def*
proof (*standard*)
fix *L*
assume ⟨*L* ∈# *all-lits-of-m* (*mset* ($N \propto i$))⟩
then have ⟨*atm-of L* ∈ *atms-of-mm* (*mset '# ran-mf N*)⟩
using *i* **unfolding** *ran-m-def in-all-lits-of-m-ain-atms-of-iff*
by (*auto dest*!: *multi-member-split*)
then show ⟨*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩
using *lits atm-of-notin-atms-of-iff in-all-lits-of-mm-ain-atms-of-iff*
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-mm-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
by *blast*
qed

lemma *isa-set-lookup-conflict*:
⟨(*uncurry6 isa-set-lookup-conflict-aa*, *uncurry6 set-conflict-m*) ∈
[λ((((((*M*, *N*), *i*), *xs*), *clvls*), *lbd*), *outl*). *i* ∈# *dom-m N* ∧ *xs* = *None* ∧ *distinct* ($N \propto i$) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset '# ran-mf N*) ∧
¬*tautology* (*mset* ($N \propto i$)) ∧ *clvls* = *0* ∧
out-learned M None outl ∧
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* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*⟩
proof −
have *H*: ⟨*set-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls lbd outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*set-conflict-m M N i None clvls lbd outl*)⟩
if
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ocr: ⟨((*b*, *n*, *xs*), *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
dist: ⟨*distinct* ($N \propto i$)⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset '# ran-mf N*)⟩ **and**
tauto: ⟨¬*tautology* (*mset* ($N \propto i$))⟩ **and**
⟨*clvls* = *0*⟩ **and**
out: ⟨*out-learned M None outl*⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
for *b n xs N i M clvls lbd outl*
proof −
have *lookup-conflict-merge-normalise*:

⟨*lookup-conflict-merge 0 M C* (*b*, *zs*) = *lookup-conflict-merge 0 M C* (*False*, *zs*)⟩
for *M C zs*
unfolding *lookup-conflict-merge-def* **by** *auto*
have [*simp*]: ⟨*out-learned M* (*Some* {#}) *outl*⟩
using *out* **by** (*cases outl*) (*auto simp*: *out-learned-def*)
have *T*: ⟨((*False*, *n*, *xs*), *Some* {#}) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩
using *ocr* **unfolding** *option-lookup-clause-rel-def* **by** *auto*
have ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* ($N \propto i$))⟩
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$[*OF lits i*] **.**
then show *?thesis* **unfolding** *set-lookup-conflict-aa-def set-conflict-m-def*
using *lookup-conflict-merge*′*-spec*[*of False n xs* ⟨{#}⟩ $\mathcal{A}$ ⟨$N\propto i$⟩ *0 - 0 outl lbd*] *that dist T*
by (*auto simp*: *lookup-conflict-merge-normalise uint-max-def merge-conflict-m-g-def*)
qed

have *H*: ⟨*isa-set-lookup-conflict-aa M*′ *arena i* (*b*, *n*, *xs*) *clvls lbd outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*set-conflict-m M N i None clvls lbd outl*)⟩
if
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ocr: ⟨((*b*, *n*, *xs*), *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
dist: ⟨*distinct* ($N \propto i$)⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)⟩ **and**
tauto: ⟨¬*tautology* (*mset* ($N \propto i$))⟩ **and**
⟨*clvls* = *0*⟩ **and**
out: ⟨*out-learned M None outl*⟩ **and**
valid: ⟨*valid-arena arena N vdom*⟩ **and**
M′*M*: ⟨(*M*′, *M*) ∈ *trail-pol* $\mathcal{A}$⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
for *b n xs N i M clvls lbd outl arena vdom M*′
unfolding *isa-set-lookup-conflict-aa-def*
apply (*rule order.trans*)
apply (*rule isa-lookup-conflict-merge-lookup-conflict-merge-ext*[*OF valid i lits ocr M*′*M bounded*])
unfolding *lookup-conflict-merge-def*[*symmetric*] *set-lookup-conflict-aa-def*[*symmetric*]
zero-uint32-nat-def[*symmetric*]
by (*auto intro*: *H*[*OF that*(*1*−*7*,*10*)])
show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
by (*intro nres-relI WB-More-Refinement.frefI*) (*auto intro*!: *H*)
qed

definition *merge-conflict-m-pre* **where**
⟨*merge-conflict-m-pre* $\mathcal{A}$ =
(λ((((((*M*, *N*), *i*), *xs*), *clvls*), *lbd*), *out*). *i* ∈# *dom-m N* ∧ *xs* ≠ *None* ∧ *distinct* ($N \propto i$) ∧
¬*tautology* (*mset* ($N \propto i$)) ∧
(∀ *L* ∈ *set* (*tl* ($N \propto i$)). − *L* ∉# *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 $\mathcal{A}$)⟩

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*) ∈
[*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 −
have *H1*: ⟨*resolve-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls lbd outl*
$\leq \Downarrow$ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*merge-conflict-m M N i C clvls lbd outl*)⟩
if
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ocr: ⟨((*b*, *n*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
dist: ⟨*distinct* (*N* ∝ *i*)⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)⟩ **and**
lits': ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the C*)⟩ **and**
tauto: ⟨¬*tautology* (*mset* (*N* ∝ *i*))⟩ **and**
out: ⟨*out-learned M C outl*⟩ **and**
not-neg: ⟨⋀*L*. *L* ∈ *set* (*tl* (*N* ∝ *i*)) ⟹ − *L* ∉# *the C*⟩ **and**
⟨*clvls* = *card-max-lvl M* (*the C*)⟩ **and**
C-None: ⟨*C* ≠ *None*⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
for *b n xs N i M clvls lbd outl C*
proof −
have *lookup-conflict-merge-normalise*:
⟨*lookup-conflict-merge 1 M C* (*b*, *zs*) = *lookup-conflict-merge 1 M C* (*False*, *zs*)⟩
for *M C zs*
unfolding *lookup-conflict-merge-def* **by** *auto*
have ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))⟩
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$[*OF lits i*] **.**
then show *?thesis* **unfolding** *resolve-lookup-conflict-aa-def merge-conflict-m-def*
using *lookup-conflict-merge'-spec*[*of b n xs* ⟨*the C*⟩ $\mathcal{A}$ ⟨*N*∝*i*⟩ *clvls M 1 outl lbd*] *that dist*
not-neg ocr C-None lits'
by (*auto simp*: *lookup-conflict-merge-normalise uint-max-def merge-conflict-m-g-def*
drop-Suc)
qed

have *H2*: ⟨*isa-resolve-merge-conflict-gt2 M' arena i* (*b*, *n*, *xs*) *clvls lbd outl*
$\leq \Downarrow$ (*Id* $\times_r$ *Id*)
(*resolve-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls lbd outl*)⟩
if
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ocr: ⟨((*b*, *n*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
dist: ⟨*distinct* (*N* ∝ *i*)⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)⟩ **and**
lits': ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the C*)⟩ **and**
tauto: ⟨¬*tautology* (*mset* (*N* ∝ *i*))⟩ **and**
out: ⟨*out-learned M C outl*⟩ **and**
not-neg: ⟨⋀*L*. *L* ∈ *set* (*tl* (*N* ∝ *i*)) ⟹ − *L* ∉# *the C*⟩ **and**
⟨*clvls* = *card-max-lvl M* (*the C*)⟩ **and**
C-None: ⟨*C* ≠ *None*⟩ **and**
valid: ⟨*valid-arena arena N vdom*⟩ **and**

i: ⟨*i* ∈# *dom-m N*⟩ **and**
dist: ⟨*distinct* (*N* ∝ *i*)⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)⟩ **and**
tauto: ⟨¬*tautology* (*mset* (*N* ∝ *i*))⟩ **and**
⟨*clvls* = *card-max-lvl M* (*the C*)⟩ **and**
out: ⟨*out-learned M C outl*⟩ **and**

bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
for *b n xs N i M clvls lbd outl arena vdom C M'*
unfolding *isa-resolve-merge-conflict-gt2-def*
apply (*rule order.trans*)
apply (*rule isa-lookup-conflict-merge-lookup-conflict-merge-ext*[*OF valid i lits ocr M'M*])
unfolding *resolve-lookup-conflict-aa-def*[*symmetric*] *set-lookup-conflict-aa-def*[*symmetric*]
using *bounded* **by** (*auto intro*: *H1*[*OF that*(*1−6*)])
show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
apply (*intro nres-relI frefI*)
apply *clarify*
subgoal
unfolding *merge-conflict-m-pre-def*
apply (*rule order-trans*)
apply (*rule H2*; *auto*; *auto*; *fail*)
by (*auto intro*!: *H1 simp*: *merge-conflict-m-pre-def*)
done
qed

definition (**in** −) *is-in-conflict* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *bool*› **where**
[*simp*]: ‹*is-in-conflict L C* ⟷ *L* ∈# *the C*›

definition (**in** −) *is-in-lookup-option-conflict*
:: ‹*nat literal* ⇒ (*bool* × *nat* × *bool option list*) ⇒ *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*)) ∈
[λ(*L*, *C*). *C* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_r$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*Id*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
subgoal for *Lxs LC*
using *lookup-clause-rel-atm-in-iff*[*of* - ‹*snd* (*snd* (*snd Lxs*))›]
apply (*cases Lxs*)
by (*auto simp*: *is-in-lookup-option-conflict-def option-lookup-clause-rel-def*)
done

definition *conflict-from-lookup* **where**
‹*conflict-from-lookup* = (λ(*n*, *xs*). *SPEC*(λ*D*. *mset-as-position xs D* ∧ *n* = *size D*))›

lemma *Ex-mset-as-position*:
‹*Ex* (*mset-as-position xs*)›
proof (*induction* ‹*size* {#*x* ∈# *mset xs*. *x* ≠ *None*#}› *arbitrary*: *xs*)
case *0*
then have *xs*: ‹*xs* = *replicate* (*length xs*) *None*›
by (*auto simp*: *filter-mset-empty-conv dest*: *replicate-length-same*)
show *?case*
by (*subst xs*) (*auto simp*: *mset-as-position.empty intro*!: *exI*[*of* - ‹{#}›])
next
case (*Suc x*) **note** *IH* = *this*(*1*) **and** *xs* = *this*(*2*)
obtain *i* **where**
[*simp*]: ‹*i* < *length xs*› **and**
xs-i: ‹*xs* ! *i* ≠ *None*›

using *xs*[*symmetric*]
by (*auto dest*!: *size-eq-Suc-imp-elem simp*: *in-set-conv-nth*)
let *?xs* = ‹*xs* [*i* := *None*]›
have ‹$x = size$ {#*x* ∈# *mset ?xs*. $x \neq None$#}›
using *xs*[*symmetric*] *xs-i* **by** (*auto simp*: *mset-update size-remove1-mset-If*)
from *IH*[*OF this*] **obtain** *D* **where**
map: ‹*mset-as-position ?xs D*›
by *blast*
have [*simp*]: ‹*Pos i* ∉# *D*› ‹*Neg i* ∉# *D*›
using *xs-i mset-as-position-nth*[*OF map*, *of* ‹*Pos i*›]
mset-as-position-nth[*OF map*, *of* ‹*Neg i*›]
by *auto*
have [*simp*]: ‹*xs* ! $i = a \Longrightarrow$ *xs*[*i* := *a*] = *xs*› **for** *a*
by *auto*

have ‹*mset-as-position xs* (*add-mset* (*if the* (*xs* ! *i*) *then Pos i else Neg i*) *D*))›
using *mset-as-position.add*[*OF map*, *of* ‹*if the* (*xs* ! *i*) *then Pos i else Neg i*› *xs*]
xs-i[*symmetric*]
by (*cases* ‹*xs* ! *i*›) *auto*
then show *?case* **by** *blast*
qed

lemma *id-conflict-from-lookup*:
‹(*RETURN o id*, *conflict-from-lookup*) ∈ [λ(*n*, *xs*). ∃ *D*. ((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$]$_f$ *Id* →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *lookup-clause-rel-def conflict-from-lookup-def RETURN-RES-refine-iff*)

lemma *lookup-clause-rel-exists-le-uint-max*:
assumes *ocr*: ‹((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and** ‹$n > 0$› **and**
le-i: ‹∀ *k*<*i*. *xs* ! $k = None$› **and** *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹∃ *j*. $j \geq i \wedge j <$ *length xs* ∧ $j <$ *uint-max* ∧ *xs* ! $j \neq None$›
proof −
have
n-D: ‹*n* = *size D*› **and**
map: ‹*mset-as-position xs D*› **and**
le-xs: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). $L <$ *length xs*›
using *ocr* **unfolding** *lookup-clause-rel-def* **by** *auto*
have *map-empty*: ‹*mset-as-position xs* {#} ⟷ (*xs* = [] ∨ *set xs* = {*None*})›
by (*subst mset-as-position.simps*) (*auto simp add*: *list-eq-replicate-iff*)
have *ex-not-none*: ‹∃ *j*. $j \geq i \wedge j <$ *length xs* ∧ *xs* ! $j \neq None$›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have ‹*xs* = [] ∨ *set xs* = {*None*}›
using *le-i* **by** (*fastforce simp*: *in-set-conv-nth*)
then have ‹*mset-as-position xs* {#}›
using *map-empty* **by** *auto*
then show *False*
using *mset-as-position-right-unique*[*OF map*] ‹$n > 0$› *n-D* **by** (*cases D*) *auto*
qed
then obtain *j* **where**
j: ‹$j \geq i$›‹$j <$ *length xs*›‹*xs* ! $j \neq None$›
by *blast*

let *?L* = ‹*if the* (*xs* ! *j*) *then Pos j else Neg j*›
have ‹*?L* ∈# *D*›
  using *j mset-as-position-in-iff-nth*[*OF map*, *of ?L*] **by** *auto*
then have ‹*nat-of-lit ?L* ≤ *uint-max*›
  using *lits bounded*
  by (*auto 5 5 dest*!: *multi-member-split*[*of - D*]
    simp: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset split*: *if-splits*)
then have ‹*j* < *uint-max*›
  by (*auto simp*: *uint-max-def split*: *if-splits*)
then show *?thesis*
  using *j* **by** *blast*
qed

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* ⟷
  (*L* = *None* ⟶ *C* = {#}) ∧
  (*L* ≠ *None* ⟶ *get-level M* (*fst* (*the L*)) = *snd* (*the L*) ∧
    snd (*the L*) = *get-maximum-level M C* ∧
    fst (*the L*) ∈# *C*
)›

Conflict Minimisation **definition** *iterate-over-conflict-inv* **where**
‹*iterate-over-conflict-inv M* D_0′ = (λ(*D*, *D*′). *D* ⊆# D_0′ ∧ *D*′ ⊆# *D*)›

definition *is-literal-redundant-spec* **where**
‹*is-literal-redundant-spec K NU UNE D L* = *SPEC*(λ*b*. *b* ⟶
  NU + *UNE* ⊨*pm remove1-mset L* (*add-mset K D*))›

definition *iterate-over-conflict*
:: ‹′*v literal* ⇒ (′*v*, ′*mark*) *ann-lits* ⇒ ′*v clauses* ⇒ ′*v clauses* ⇒ ′*v clause* ⇒
  ′*v clause nres*›
where
‹*iterate-over-conflict K M NU UNE* D_0′ = *do* {
  (*D*, -) ←
    $WHILE_T{}^{iterate\text{-}over\text{-}conflict\text{-}inv\ M\ D_0'}$
    (λ(*D*, *D*′). *D*′ ≠ {#})
    (λ(*D*, *D*′). *do*{
      x ← *SPEC* (λ*x*. *x* ∈# *D*′);
      red ← *is-literal-redundant-spec K NU UNE D x*;
      if ¬*red*
      then RETURN (*D*, *remove1-mset x D*′)
      else RETURN (*remove1-mset x D*, *remove1-mset x D*′)
    })
    (D_0′, D_0′);
  RETURN D
}›

definition *minimize-and-extract-highest-lookup-conflict-inv* **where**
‹*minimize-and-extract-highest-lookup-conflict-inv* = (λ(*D*, *i*, *s*, *outl*).
  length outl ≤ *uint-max* ∧ *mset* (*tl outl*) = *D* ∧ *outl* ≠ [] ∧ *i* ≥ *1*)›

type-synonym ′*v conflict-highest-conflict* = ‹(′*v literal* × *nat*) *option*›

```
definition (in −) atm-in-conflict where
  ‹atm-in-conflict L D ⟷ L ∈ atms-of D›

definition atm-in-conflict-lookup :: ‹nat ⇒ lookup-clause-rel ⇒ bool› where
  ‹atm-in-conflict-lookup = (λL (-, xs). xs ! L ≠ None)›

definition atm-in-conflict-lookup-pre :: ‹nat ⇒ lookup-clause-rel ⇒ bool› where
‹atm-in-conflict-lookup-pre L xs ⟷ L < length (snd xs)›

lemma atm-in-conflict-lookup-atm-in-conflict:
  ‹(uncurry (RETURN oo atm-in-conflict-lookup), uncurry (RETURN oo atm-in-conflict)) ∈
     [λ(L, xs). L ∈ atms-of (ℒall 𝒜)]f Id ×f lookup-clause-rel 𝒜 → ⟨bool-rel⟩nres-rel›
  apply (intro frefI nres-relI)
  subgoal for x y
    using mset-as-position-in-iff-nth[of ‹snd (snd x)› ‹snd y› ‹Pos (fst x)›]
      mset-as-position-in-iff-nth[of ‹snd (snd x)› ‹snd y› ‹Neg (fst x)›]
    by (cases x; cases y)
      (auto simp: atm-in-conflict-lookup-def atm-in-conflict-def
        lookup-clause-rel-def atm-iff-pos-or-neg-lit
        pos-lit-in-atms-of neg-lit-in-atms-of)
  done

lemma atm-in-conflict-lookup-pre:
  fixes x1 :: ‹nat› and x2 :: ‹nat›
  assumes
    ‹x1n ∈# ℒall 𝒜› and
    ‹(x2f, x2a) ∈ lookup-clause-rel 𝒜›
  shows ‹atm-in-conflict-lookup-pre (atm-of x1n) x2f›
proof −
  show ?thesis
    using assms
    by (auto simp: lookup-clause-rel-def atm-in-conflict-lookup-pre-def atms-of-def)
qed

definition is-literal-redundant-lookup-spec where
   ‹is-literal-redundant-lookup-spec 𝒜 M NU NUE D′ L s =
    SPEC(λ(s′, b). b ⟶ (∀D. (D′, D) ∈ lookup-clause-rel 𝒜 ⟶
       (mset ‘# mset (tl NU)) + NUE ⊨pm remove1-mset L D))›

type-synonym (in −) conflict-min-cach-l = ‹minimize-status list × nat list›

definition (in −) conflict-min-cach-set-removable-l
  :: ‹conflict-min-cach-l ⇒ nat ⇒ conflict-min-cach-l nres›
where
  ‹conflict-min-cach-set-removable-l = (λ(cach, sup) L. do {
     ASSERT(L < length cach);
     ASSERT(length sup ≤ 1 + uint32-max div 2);
     RETURN (cach[L := SEEN-REMOVABLE], if cach ! L = SEEN-UNKNOWN then sup @ [L] else
sup)
   })›

definition (in −) conflict-min-cach :: ‹nat conflict-min-cach ⇒ nat ⇒ minimize-status› 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*))›
unfolding *ana-lookup-rel-def*
by *auto*

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* =
$WHILE_T{}^{\textit{lit-redundant-rec-wl-inv2 M NU D}}$
(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* ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ *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
let $C = NU \propto C$;
if $i \geq \textit{len}$
then
$RETURN(\textit{cach}\ (\textit{atm-of}\ (C\ !\ k) := \textit{SEEN-REMOVABLE}),\ \textit{butlast analyse},\ \textit{True})$
else do {
let $(L,\ \textit{analyse}) = \textit{get-literal-and-remove-of-analyse-wl2}\ C\ \textit{analyse}$;
$ASSERT(L \in\#\ \mathcal{L}_{all}\ \mathcal{A})$;
let $b = \neg\textit{level-in-lbd}\ (\textit{get-level}\ M\ L)\ \textit{lbd}$;
if $(\textit{get-level}\ M\ L = \textit{zero-uint32-nat}\ \vee$
$\textit{conflict-min-cach cach}\ (\textit{atm-of}\ L) = \textit{SEEN-REMOVABLE}\ \vee$
$\textit{atm-in-conflict}\ (\textit{atm-of}\ L)\ D)$
then $RETURN\ (\textit{cach},\ \textit{analyse},\ \textit{False})$
else if $b \vee \textit{conflict-min-cach cach}\ (\textit{atm-of}\ L) = \textit{SEEN-FAILED}$
then do {
$ASSERT(\textit{mark-failed-lits-stack-inv2}\ NU\ \textit{analyse cach})$;
$\textit{cach} \leftarrow \textit{mark-failed-lits-wl}\ NU\ \textit{analyse cach}$;
$RETURN\ (\textit{cach},\ [],\ \textit{False})$
}
else do {
$ASSERT(-\ L \in \textit{lits-of-l}\ M)$;
$C \leftarrow \textit{get-propagation-reason}\ M\ (-L)$;
case C *of*
Some $C \Rightarrow$ *do* {
$ASSERT(C \in\#\ \textit{dom-m}\ NU)$;
$ASSERT(\textit{length}\ (NU \propto C) \geq 2)$;
$ASSERT(\textit{literals-are-in-}\mathcal{L}_{in}\ \mathcal{A}\ (\textit{mset}\ (NU \propto C)))$;
$ASSERT(\textit{length}\ (NU \propto C) \leq \textit{Suc}\ (\textit{uint32-max div 2}))$;
$RETURN\ (\textit{cach},\ \textit{analyse}\ @\ [\textit{lit-redundant-reason-stack2}\ (-L)\ NU\ C],\ \textit{False})$
}
| *None* $\Rightarrow$ *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*:
‹*lit-redundant-rec-wl-ref* $NU\ x \Longrightarrow$ *lit-redundant-rec-wl-ref* NU (*butlast* x)›
by (*cases* x *rule*: *rev-cases*)
(*auto simp*: *lit-redundant-rec-wl-ref-def dest*: *in-set-butlastD*)

lemma *lit-redundant-rec-wl-lookup-mark-failed-lits-stack-inv*:
assumes
‹$(x,\ x') \in \textit{Id}$› **and**
‹*case* x *of* $(\textit{cach},\ \textit{analyse},\ b) \Rightarrow \textit{analyse} \neq []$› **and**
‹*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* $(NU \propto \textit{fst}\ (\textit{last}\ x1c))\ x1c = (x1e,\ x2e)$› **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 −
show *?thesis*
using *assms*
unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def*
by (*cases* ⟨*x1a*⟩ *rule*: *rev-cases*)
(*auto simp*: *elim*!: *in-set-upd-cases*)
qed

context
fixes *M D* $\mathcal{A}$ *NU analysis analysis'*
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**
ana: ⟨(*analysis*, *analysis'*) ∈ *ana-lookups-rel NU*⟩ **and**
lits-NU: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
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 −
show *?thesis* **using** *ana assms* **by** *auto*
qed

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*⟩
using *x-x'* **unfolding** *lit-redundant-rec-wl-inv2-def*
by *auto*

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* ≠ []) = (*x1c* ≠ [])›
using *assms x-x'*
by *auto*

end

context
assumes
‹*case x of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **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* ⇒ *minimize-status*› **and**
x2 :: ‹(*nat* × *nat* × *nat* × *nat*) *list* × *bool*› **and**
x1a :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and** *x2a* :: ‹*bool*› **and**
x1b :: ‹*nat* ⇒ *minimize-status*› **and**
x2b :: ‹(*nat* × *nat* × *bool*) *list* × *bool*› **and**
x1c :: ‹(*nat* × *nat* × *bool*) *list*› **and**
x2c :: ‹*bool*›
assumes *st*:
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x2*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x* = (*x1b*, *x2b*)› **and**
x1a: ‹*x1a* ≠ []›
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* ≠ []›
using *st x-x' x1a* **by** *auto*

lemma *ccmin-nempty*:
shows ‹*x1c* ≠ []›
using *x-x' x1a*
by (*auto simp*: *st*)

context
  notes -[*simp*] = *st*
  fixes *x1d* :: ‹*nat*› **and** *x2d* :: ‹*nat* × *nat* × *nat*› **and**
    x1e :: ‹*nat*› **and** *x2e* :: ‹*nat* × *nat*› **and**
    x1f :: ‹*nat*› **and**
    x2f :: ‹*nat*› **and** *x1g* :: ‹*nat*› **and**
    x2g :: ‹*nat* × *nat* × *nat*› **and**
    x1h :: ‹*nat*› **and**
    x2h :: ‹*nat* × *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*›
  using *st2 last ana-lookup-conv x-x′ x1a last*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
    auto simp: *ana-lookup-conv-def ana-lookup-rel-def*
      list-rel-append-single-iff; *fail*)+

private 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*)›
  using *inv2 x1a last x-x′* **unfolding** *x1g-x1d st j-def b-def st2*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
   auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
    lit-redundant-rec-wl-inv-def ana-lookup-rel-def
    lit-redundant-rec-wl-ref-def)

private 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*))⟩
subgoal
using *inv2 x1a last x-x′ x1c ana-lookup-conv*
unfolding *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def ana-lookup-conv-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)

subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
done

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
using *dom* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
using *le* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
using *in-lits* **unfolding** *x1g-x1d* **.**

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›
using *lits-NU multi-member-split*[*OF x1g-dom*]
by (*auto simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)

lemma *ccmin-le-uint32-max*:
‹*length* (*NU* ∝ *x1g*) ≤ *Suc* (*uint32-max div 2*)›

using *simple-clss-size-upper-div2* [*OF bounded ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*]
dist tauto **unfolding** *x1g-x1d*
by *auto*

lemma *ccmin-in-all-lits*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1h*]
le **unfolding** *x1g-x1d* **by** *auto*

lemma *ccmin-less-length*:
shows ‹*x2i* ≤ *length* (*NU* ∝ *x1g*)›
using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *ccmin-same-cond*:
shows ‹(*x2i* ≤ *x1i*) = (*x2f* ≤ *x1f*)›
using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *list-rel-butlast*:
assumes *rel*: ‹(*xs*, *ys*) ∈ ⟨*R*⟩*list-rel*›
shows ‹(*butlast xs*, *butlast ys*) ∈ ⟨*R*⟩*list-rel*›
proof −
have ‹*length xs* = *length ys*›
using *assms list-rel-imp-same-length* **by** *blast*
then show *?thesis*
using *rel*
by (*induction xs ys rule*: *list-induct2*) (*auto split*: *nat.splits*)
qed

lemma *ccmin-set-removable*:
assumes
‹*x2i* ≤ *x1i*› **and**
‹*x2f* ≤ *x1f*› **and** ‹*lit-redundant-rec-wl-inv2 M NU D x*›
shows ‹((*x1b*(*atm-of* (*NU* ∝ *x1g* ! *x1h*) := *SEEN-REMOVABLE*), *butlast x1c*, *True*),
x1(*atm-of* (*NU* ∝ *x1d* ! *x1e*) := *SEEN-REMOVABLE*), *butlast x1a*, *True*)
∈ {((*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*)}›
using *x-x*′ **by** (*auto simp*: *x1g-x1d lit-redundant-rec-wl-ref-butlast lit-redundant-rec-wl-inv-def*
dest: *list-rel-butlast*)

context
assumes
le: ‹¬ *x2i* ≤ *x1i*› ‹¬ *x2f* ≤ *x1f*›
begin

context
notes -[*simp*]= *x1g-x1d st2 last*
fixes *x1j* :: ‹*nat literal*› **and** *x2j* :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and**
x1k :: ‹*nat literal*› **and** *x2k* :: ‹(*nat* × *nat* × *bool*) *list*›
assumes
rem: ‹*get-literal-and-remove-of-analyse-wl* (*NU* ∝ *x1d*) *x1a* = (*x1j*, *x2j*)› **and**
rem2:‹*get-literal-and-remove-of-analyse-wl2* (*NU* ∝ *x1g*) *x1c* = (*x1k*, *x2k*)› **and**
‹*fst* (*snd* (*snd* (*last x2j*))) ≠ *0*› **and**
ux1j-M: ‹− *x1j* ∈ *lits-of-l M*›
begin

private lemma *confl-min-last*: ‹(*last x1c*, *last x1a*) ∈ *ana-lookup-rel NU*›
  using *x1a x1c x-x′ rem rem2 last ana-lookup-conv* **unfolding** *x1g-x1d st2 b-def st*
  by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
    (*auto simp*: *list-rel-append-single-iff*
     get-literal-and-remove-of-analyse-wl-def
    get-literal-and-remove-of-analyse-wl2-def)

private lemma *rel*: ‹(*x1c*[*length x1c* − *Suc 0* := (*x1d*, *Suc x1f*, *b*)], *x1a*
     [*length x1a* − *Suc 0* := (*x1d*, *x1e*, *Suc x1f*, *x2f*)])
    ∈ *ana-lookups-rel NU*›
  using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
  by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
    (*auto simp*: *list-rel-append-single-iff*
      ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
      get-literal-and-remove-of-analyse-wl2-def)

private lemma *x1k-x1j*: ‹*x1k* = *x1j*› ‹*x1j* = *NU* ∝ *x1d* ! *x1f*› **and**
  x2k-x2j: ‹(*x2k*, *x2j*) ∈ *ana-lookups-rel NU*›
  subgoal
    using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
    by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
      (*auto simp*: *list-rel-append-single-iff*
 ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
 get-literal-and-remove-of-analyse-wl2-def)
  subgoal
    using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
    by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
      (*auto simp*: *list-rel-append-single-iff*
 ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
 get-literal-and-remove-of-analyse-wl2-def)
  subgoal
    using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
    by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
      (*auto simp*: *list-rel-append-single-iff*
 ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
 get-literal-and-remove-of-analyse-wl2-def)
  done

lemma *ccmin-x1k-all*:
  shows ‹*x1k* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
  unfolding *x1k-x1j*
  using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1f*]
    literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*OF lits* ‹− *x1j* ∈ *lits-of-l M*›]
  le st3 **unfolding** *x1g-x1d* **by** (*auto split*: *if-splits simp*: *x1k-x1j uminus-*$\mathcal{A}_{in}$*-iff*)

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′*
  using *M-D n-d*
  by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*

atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)

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*)›
using *in-lits ana ux1j-M*
by (*auto simp add*: *in-conflict-atm-in*)

private lemma *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j*, *False*)›
using *x-x′ last ana-lookup-conv rem rem2 x1a x1c le*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)
(*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

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*)}›
using *x2k-x2j ccmin-lit-redundant-rec-wl-inv* **by** *auto*

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*)›
using *b* **by** *auto*

context
assumes
‹¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*› **and**
‹*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-lbd*:
shows ‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
using *x1a x1c x2k-x2j rem rem2 x-x′ le last*
unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def*

lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
unfolding *mark-failed-lits-stack-inv2-def*
apply −
apply (*rule exI*[*of - x2j*])
apply (*cases* ‹*x1a*› *rule*: *rev-cases*; *cases* ‹*x1c*› *rule*: *rev-cases*)
by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-wl-lbd*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
$\le \Downarrow$ *Id*
(*mark-failed-lits-wl NU x2j x1*)›
by (*auto simp*: *mark-failed-lits-wl-def*)

lemma *ccmin-rel-lbd*:
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*)}›
using *x-x' assms* **by** (*auto simp*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*)

end

context
assumes
‹¬ (¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*)› **and**
‹¬ (*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
begin

lemma *ccmin-lit-in-trail*:
‹− *x1k* ∈ *lits-of-l M*›
using ‹− *x1j* ∈ *lits-of-l M*› *x1k-x1j*(*1*) **by** *blast*

lemma *ccmin-lit-eq*:
‹− *x1k* = − *x1j*›
by *auto*

context
fixes *xa* :: ‹*nat option*› **and** *x'a* :: ‹*nat option*›
assumes *xa-x'a*: ‹(*xa*, *x'a*) ∈ ⟨*nat-rel*⟩*option-rel*›
begin

lemma *ccmin-lit-eq2*:
‹(*xa*, *x'a*) ∈ *Id*›
using *xa-x'a* **by** *auto*

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*⟩
using *x1a x1c x2k-x2j rem rem2 x-x′ le last*
unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def*
unfolding *mark-failed-lits-stack-inv2-def*
apply −
apply (*rule exI*[*of - x2j*])
apply (*cases* ⟨*x1a*⟩ *rule*: *rev-cases*; *cases* ⟨*x1c*⟩ *rule*: *rev-cases*)
by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-stack-wl-dec*:
shows ⟨*mark-failed-lits-wl NU x2k x1b*
$\leq \Downarrow$ *Id*
(*mark-failed-lits-wl NU x2j x1*)⟩
by (*auto simp*: *mark-failed-lits-wl-def*)

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*)}⟩
using *assms* **by** (*auto simp*: *lit-redundant-rec-wl-ref-def lit-redundant-rec-wl-inv-def*)

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*)⟩
using *H* **by** *auto*

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*:
shows ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *xb*))⟩
using *lits-NU multi-member-split*[*of xb* ⟨*dom-m NU*⟩] *H*
by (*auto simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)

lemma *ccmin-le-uint32-max-xb*:
⟨*length* (*NU* ∝ *xb*) ≤ *Suc* (*uint32-max div 2*)⟩
using *simple-clss-size-upper-div2*[*OF bounded ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*]
H **unfolding** *x1g-x1d*
by *auto*

private 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*)⟩
  using *ccmin-stack-pre H x-x' last ana-lookup-conv rem rem2 x1a x1c le*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)
    (*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
    lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
    list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

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*)}⟩
  using *x2k-x2j H ccmin-lit-redundant-rec-wl-inv3*
  by (*auto simp*: *list-rel-append-single-iff ana-lookup-rel-alt-def*
    lit-redundant-reason-stack2-def lit-redundant-reason-stack-def)

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* ⊨*as CNot D*⟩ **and**
    n-d: ⟨*no-dup M*⟩ **and**
    lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*⟩ **and**
    ⟨(*analysis*, *analysis'*) ∈ *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 −
  have *M*: ⟨∀ *a* ∈ *lits-of-l M*. *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩
    using *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l lits* **by** *blast*
  have [*simp*]: ⟨− *x1e* ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e*) *D* ⟷ *x1e* ∈# *D*⟩ **for** *x1e*
    using *M-D n-d*
    by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*
      atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)
  have [*simp*, *intro*]: ⟨− *x1e* ∈ *lits-of-l M* ⟹ *atm-of x1e* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
    ⟨*x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
    ⟨− *x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩ **for** *x1e*
    using *lits atm-of-notin-atms-of-iff literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l* **apply** *blast*
    using *M uminus-*$\mathcal{A}_{in}$*-iff* **by** *auto*
  have [*refine-vcg*]: ⟨(*a*, *b*) ∈ *Id* ⟹ (*a*, *b*) ∈ ⟨*Id*⟩*option-rel*⟩ **for** *a b* **by** *auto*

have [*refine-vcg*]: ‹*get-propagation-reason M x*
$\le \Downarrow$ (⟨*nat-rel*⟩*option-rel*) (*get-propagation-reason M y*)› **if** ‹$x = y$› **for** *x y*
by (*use that* **in** *auto*)
have [*refine-vcg*]:‹*RETURN* (¬ *level-in-lbd* (*get-level M L*) *lbd*) $\le \Downarrow$ *Id* (*RES UNIV*)› **for** *L*
by *auto*
have [*refine-vcg*]: ‹*mark-failed-lits-wl NU a b*
$\le \Downarrow$ *Id*
(*mark-failed-lits-wl NU a′ b′*)› **if** ‹$a = a'$› **and** ‹$b = b'$› **for** *a a′ b b′*
unfolding *that* **by** *auto*

have *H*: ‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* $\le$
$\Downarrow$ {((*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*)}
(*lit-redundant-rec-wl M NU D cach analysis′ lbd*)›
using *assms* **apply** −
unfolding *lit-redundant-rec-wl-lookup-def lit-redundant-rec-wl-def WHILET-def*
apply (*refine-vcg*)
subgoal by (*rule ccmin-rel*)
subgoal by (*rule ccmin-lit-redundant-rec-wl-inv2*)
subgoal by (*rule ccmin-cond*)
subgoal by (*rule ccmin-nempty*)
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*rule ccmin-in-dom*)
subgoal by (*rule ccmin-in-dom-le-length*)
subgoal by (*rule ccmin-in-trail*)
subgoal by (*rule ccmin-in-all-lits*)
subgoal by (*rule ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*)
subgoal by (*rule ccmin-le-uint32-max*)
subgoal by (*rule ccmin-less-length*)
subgoal by (*rule ccmin-same-cond*)
subgoal by (*rule ccmin-set-removable*)
subgoal by (*rule ccmin-x1k-all*)
subgoal by (*rule ccmin-already-seen*)
subgoal by (*rule ccmin-already-seen-rel*)
subgoal by (*rule ccmin-already-failed*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-lbd*)
apply (*rule ccmin-mark-failed-lits-wl-lbd*; *assumption*)
subgoal by (*rule ccmin-rel-lbd*)
subgoal by (*rule ccmin-lit-in-trail*)
subgoal by (*rule ccmin-lit-eq*)
subgoal by (*rule ccmin-lit-eq2*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-dec*)
apply (*rule ccmin-mark-failed-lits-stack-wl-dec*; *assumption*)
subgoal by (*rule ccmin-rel-dec*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*)
subgoal by (*rule ccmin-le-uint32-max-xb*)
subgoal by (*rule ccmin-stack-rel*)
done
show *?thesis*
by (*rule H*[*THEN order-trans*], *rule conc-fun-R-mono*)
auto
qed

definition *literal-redundant-wl-lookup* **where**

```
‹literal-redundant-wl-lookup 𝒜 M NU D cach L lbd = do {
    ASSERT(L ∈# ℒ_all 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-ℒ_in 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 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 ⊨as CNot D› ‹no-dup M› ‹literals-are-in-ℒ_in-trail 𝒜 M›
   ‹literals-are-in-ℒ_in-mm 𝒜 ((mset ∘ fst) ‘# ran-m NU)› and
   ‹isasat-input-bounded 𝒜›
 shows
   ‹literal-redundant-wl-lookup 𝒜 M NU D cach L lbd ≤
     ⇓ (Id ×_f (ana-lookups-rel NU ×_f bool-rel)) (literal-redundant-wl M NU D cach L lbd)›
proof −
 have M: ‹∀ a ∈ lits-of-l M. a ∈# ℒ_all 𝒜›
   using literals-are-in-ℒ_in-trail-in-lits-of-l assms by blast
 have [simp, intro!]: ‹− x1e ∈ lits-of-l M ⟹ atm-of x1e ∈ atms-of (ℒ_all 𝒜)›
   ‹− x1e ∈ lits-of-l M ⟹ x1e ∈# (ℒ_all 𝒜)› for x1e
   using assms atm-of-notin-atms-of-iff literals-are-in-ℒ_in-trail-in-lits-of-l apply blast
   using M uminus-𝒜_in-iff by auto
 have [refine]: ‹(x, x′) ∈ Id ⟹ (x, x′) ∈ ⟨Id⟩option-rel› for x x′
   by auto
 have [refine-vcg]: ‹get-propagation-reason M x
   ≤ ⇓ ({(C, C′). (C, C′) ∈ ⟨nat-rel⟩option-rel})
     (get-propagation-reason M y)› if ‹x = y› and ‹y ∈ lits-of-l M› for x y
   by (use that in ‹auto simp: get-propagation-reason-def intro: RES-refine›)
 show ?thesis
   unfolding literal-redundant-wl-lookup-def literal-redundant-wl-def
   apply (refine-vcg lit-redundant-rec-wl-lookup-lit-redundant-rec-wl)
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
```

subgoal by *auto*
subgoal
using *assms* **by** (*auto dest*!: *multi-member-split simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*mset* (*NU* ∝ -)›] **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *lit-redundant-reason-stack2-def lit-redundant-reason-stack-def ana-lookup-rel-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
done
qed

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 −

have ‹*filter-to-poslev* M' *L E* ⊆# *remove1-mset L E*›
by (*induction E*)
(*auto simp add*: *filter-to-poslev-add-mset remove1-mset-add-mset-If subset-mset-trans-add-mset*
intro: *diff-subset-eq-self subset-mset.dual-order.trans*)
then have ‹*NU* ⊨*p add-mset* (− *L*) (*remove1-mset L E*)›
using *NU-uL-E*
by (*meson conflict-minimize-intermediate-step mset-subset-eqD*)
moreover have ‹*NU* ⊨*p add-mset L* (*remove1-mset L E*)›
using *NU-E L-E* **by** *auto*
ultimately show *?thesis*
using *true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or*[*of NU L* ‹*remove1-mset L E*›
‹*remove1-mset L E*›]
by (*auto simp*: *true-clss-cls-add-self*)
qed

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
‹$S''' \equiv$ *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**
bounded: ‹*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* ∧ *outl* ≠ []})
(*iterate-over-conflict K M* NU' *NUE D*)›
(**is** ‹- ≤ ⇓ *?R* -›)
proof −
let *?UE* = ‹*get-unit-learned-clss-wl S*›
let *?NE* = ‹*get-unit-init-clss-wl S*›
define *N U* **where**
‹*N* ≡ *mset* ‘# *init-clss-lf NU*› **and**
‹*U* ≡ *mset* ‘# *learned-clss-lf NU*›
obtain *E* **where**

S''': ‹$S''' = (M', N + ?NE, U + ?UE, E)$›
  using *M′ S-S′ S′-S″* **unfolding** *S‴-def N-def U-def NU*
  by (*cases S*) (*auto simp*: *state-wl-l-def twl-st-l-def*
    mset-take-mset-drop-mset′)
then have *NU-N-U*: ‹*mset ‘# ran-mf NU* = *N* + *U*›
  using *NU S-S′ S′-S″* **unfolding** *S‴-def N-def U-def*
  apply (*subst all-clss-l-ran-m*[*symmetric*])
  apply (*subst image-mset-union*[*symmetric*])
  apply (*subst image-mset-union*[*symmetric*])
  by (*auto simp*: *mset-take-mset-drop-mset′*)
let *?NU* = ‹*N* + *?NE* + *U* + *?UE*›
have *NU′-N-U*: ‹*NU′* = *N* + *U*›
  unfolding *NU′-def N-def U-def mset-append*[*symmetric*] *image-mset-union*[*symmetric*]
  by *auto*
have *NU′-NUE*: ‹*NU′* + *NUE* = *N* + *get-unit-init-clss-wl S* + *U* + *get-unit-learned-clss-wl S*›
  unfolding *NUE NU′-N-U* **by** (*auto simp*: *ac-simps*)
have *struct-inv-S‴*: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*M′*, *N* + *get-unit-init-clss-wl S*,
    U + *get-unit-learned-clss-wl S*, *E*)›
  using *struct-invs* **unfolding** *twl-struct-invs-def S‴-def*[*symmetric*] *S‴*
  by *fast*
then have *n-d*: ‹*no-dup M′*›
  unfolding $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-M-level-inv-def*
    trail.simps **by** *fast*
then have *n-d*: ‹*no-dup M*›
  using *S-S′ S′-S″* **unfolding** *M-def M′ S‴-def* **by** (*auto simp*: *twl-st-wl twl-st-l twl-st*)

define *R* **where**
  ‹*R* = {((*D′*:: *nat clause*, *i*, *cach* :: *nat* ⇒ *minimize-status*, *outl′* :: *out-learned*),
      (*F* :: *nat clause*, *E* :: *nat clause*)).
      i ≤ *length outl′* ∧
      F ⊆# *D* ∧
      E ⊆# *F* ∧
      mset (*drop i outl′*) = *E* ∧
      mset (*tl outl′*) = *F* ∧
      conflict-min-analysis-inv M′ cach (*?NU*) *F* ∧
      NU′ + *NUE* ⊨*pm add-mset K F* ∧
      mset (*tl outl′*) = *D′* ∧
      i > *0* ∧ *outl′* ≠ [] ∧
      outl′ ! *0* = *K*
    }›
have [*simp*]: ‹*add-mset K* (*mset* (*tl outl*)) = *mset outl*›
  using *D′-D K*
  by (*cases outl*) (*auto simp*: *drop-Suc outl-nempty*)
have ‹*Suc 0* < *length outl* ⟹
  highest-lit M (*mset* (*take* (*Suc 0*) (*tl outl*)))
   (*Some* (*outl* ! *Suc 0*, *get-level M* (*outl* ! *Suc 0*)))›
  using *outl-nempty*
  by (*cases outl*; *cases* ‹*tl outl*›) (*auto simp*: *highest-lit-def get-maximum-level-add-mset*)
then have *init-args-ref*: ‹((*D*, *one-uint32-nat*, *s′*, *outl*), *D*, *D*) ∈ *R*›
  using *D′-D cach-init NU-P-D dist-D tauto K*
  unfolding *R-def NUE NU′-def NU-N-U*
  by (*auto simp*: *ac-simps drop-Suc outl-nempty*)

have *init-lo-inv*: ‹*minimize-and-extract-highest-lookup-conflict-inv s′*›
  if
    ‹(*s′*, *s*) ∈ *R*› **and**

⟨*iterate-over-conflict-inv M D s*⟩
for *s′ s*
proof −
have [*dest*!]: ⟨*mset b* ⊆# *D* ⟹ *length b* ≤ *size D*⟩ **for** *b*
using *size-mset-mono* **by** *fastforce*
show *?thesis*
using *that simple-clss-size-upper-div2*[*OF bounded lits-D dist-D tauto*]
unfolding *minimize-and-extract-highest-lookup-conflict-inv-def*
by (*auto simp*: *R-def uint-max-def*)
qed
have *cond*: ⟨(*m* < *length outl′*) = (*D′* ≠ {#})⟩
if
st′-st: ⟨(*st′*, *st*) ∈ *R*⟩ **and**
⟨*minimize-and-extract-highest-lookup-conflict-inv st′*⟩ **and**
⟨*iterate-over-conflict-inv M D st*⟩ **and**
st:
⟨*x2b* = (*j*, *outl′*)⟩
⟨*x2a* = (*m*, *x2b*)⟩
⟨*st′* = (*nxs*, *x2a*)⟩
⟨*st* = (*E*, *D′*)⟩
for *st′ st nxs x2a m x2b j x2c D′ E st2 st3 outl′*
proof −
show *?thesis*
using *st′-st* **unfolding** *st R-def*
by *auto*
qed

have *redundant*: ⟨*literal-redundant-wl-lookup* 𝒜 *M NU nxs cach*
(*outl′* ! *x1d*) *lbd*
≤ ⇓ {((*s′*, *a′*, *b′*), *b*). *b* = *b′* ∧
(*b* ⟶ *NU′* + *NUE* ⊨*pm remove1-mset L* (*add-mset K E*) ∧
conflict-min-analysis-inv M′ s′ ?NU (*remove1-mset L E*)) ∧
(¬*b* ⟶ *NU′* + *NUE* ⊨*pm add-mset K E* ∧ *conflict-min-analysis-inv M′ s′ ?NU E*)}
(*is-literal-redundant-spec K NU′ NUE E L*)⟩
(**is** ⟨- ≤ ⇓ *?red* -⟩)
if
R: ⟨(*x*, *x′*) ∈ *R*⟩ **and**
⟨*case x′ of* (*D*, *D′*) ⇒ *D′* ≠ {#}⟩ **and**
⟨*minimize-and-extract-highest-lookup-conflict-inv x*⟩ **and**
⟨*iterate-over-conflict-inv M D x′*⟩ **and**
st:
⟨*x′* = (*E*, *x1a*)⟩
⟨*x2d* = (*cach*, *outl′*)⟩
⟨*x2c* = (*x1d*, *x2d*)⟩
⟨*x* = (*nxs*, *x2c*)⟩ **and**
L: ⟨(*outl′*!*x1d*, *L*) ∈ *Id*⟩
⟨*x1d* < *length outl′*⟩
for *x x′ E x2 x1a x2a nxs x2c x1d x2d x1e x2e cach highest L outl′ st3*
proof −
let *?L* = ⟨(*outl′* ! *x1d*)⟩
have
⟨*x1d* < *length outl′*⟩ **and**
⟨*x1d* ≤ *length outl′*⟩ **and**
⟨*mset* (*tl outl′*) ⊆# *D*⟩ **and**
⟨*E* = *mset* (*tl outl′*)⟩ **and**
cach: ⟨*conflict-min-analysis-inv M′ cach ?NU E*⟩ **and**

NU-P-E: ⟨$NU' + NUE \models pm$ *add-mset K* (*mset* (*tl outl'*))⟩ **and**
⟨$nxs = mset\ (tl\ outl')$⟩ **and**
⟨$0 < x1d$⟩ **and**
[*simp*]: ⟨$L = outl'!x1d$⟩ **and**
⟨$E \subseteq\# D$⟩
⟨$E = mset\ (tl\ outl')$⟩ **and**
⟨$E = nxs$⟩
using *R L* **unfolding** *R-def st*
by *auto*

have *M-x1*: ⟨$M \models as$ *CNot E*⟩
by (*metis CNot-plus M-D* ⟨$E \subseteq\# D$⟩ *subset-mset.le-iff-add true-annots-union*)
then have *M'-x1*: ⟨$M' \models as$ *CNot E*⟩
using *S-S' S'-S''* **unfolding** *M' M-def S'''-def* **by** (*auto simp*: *twl-st twl-st-wl twl-st-l*)
have ⟨*outl'* ! *x1d* $\in\#$ *E*⟩
using ⟨$E = mset\ (tl\ outl')$⟩ ⟨$x1d < length\ outl'$⟩ ⟨$0 < x1d$⟩
by (*auto simp*: *nth-in-set-tl*)

have *1*:
⟨*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs cach ?L lbd* $\leq \Downarrow$ (*Id* $\times_f$ (*ana-lookups-rel NU* $\times_f$ *bool-rel*))
(*literal-redundant-wl M NU nxs cach ?L lbd*)⟩
by (*rule literal-redundant-wl-lookup-literal-redundant-wl*)
(*use lits-NU n-d lits M-x1 struct-invs bounded add-inv* ⟨*outl'* ! *x1d* $\in\#$ *E*⟩ ⟨$E = nxs$⟩ **in** *auto*)

have *2*:
⟨*literal-redundant-wl M NU nxs cach ?L lbd* $\leq \Downarrow$
(*Id* $\times_r$ {(*analyse*, *analyse'*). *analyse'* = *convert-analysis-list NU analyse* $\wedge$
lit-redundant-rec-wl-ref NU analyse} $\times_r$ *bool-rel*)
(*literal-redundant M' NU' nxs cach ?L*)⟩
by (*rule literal-redundant-wl-literal-redundant*[*of S S' S''*,
unfolded M-def[*symmetric*] *NU*[*symmetric*] *M'*[*symmetric*] *S'''-def*[*symmetric*]
NU'-def[*symmetric*], *THEN order-trans*])
(*use bounded S-S' S'-S'' M-x1 struct-invs add-inv* ⟨*outl'* ! *x1d* $\in\#$ *E*⟩ ⟨$E = nxs$⟩ **in**
⟨*auto simp*: *NU*⟩)

have *3*:
⟨*literal-redundant M'* ($N + U$) *nxs cach ?L* $\leq$
literal-redundant-spec M' ($N + U + ?NE + ?UE$) *nxs ?L*⟩
unfolding ⟨$E = nxs$⟩[*symmetric*]
apply (*rule literal-redundant-spec*)
apply (*rule struct-inv-S'''*)
apply (*rule cach*)
apply (*rule* ⟨*outl'* ! *x1d* $\in\#$ *E*⟩)
apply (*rule M'-x1*)
done

then have *3*:
⟨*literal-redundant M'* (*NU'*) *nxs cach ?L* $\leq$ *literal-redundant-spec M' ?NU nxs ?L*⟩
by (*auto simp*: *ac-simps NU'-N-U*)

have *ent*: ⟨$NU' + NUE \models pm$ *add-mset* ($-$ *L*) (*filter-to-poslev M' L* (*add-mset K E*))⟩
if ⟨$NU' + NUE \models pm$ *add-mset* ($-$ *L*) (*filter-to-poslev M' L E*)⟩
using *that* **by** (*auto simp*: *filter-to-poslev-add-mset add-mset-commute*)
show *?thesis*
apply (*rule order.trans*)
apply (*rule 1*)

apply (*rule order.trans*)
apply (*rule ref-two-step'*)
apply (*rule 2*)
apply (*subst conc-fun-chain*)
apply (*rule order.trans*)
apply (*rule ref-two-step'*[*OF 3*])
unfolding *literal-redundant-spec-def is-literal-redundant-spec-def*
conc-fun-SPEC NU'-NUE[*symmetric*]
apply (*rule SPEC-rule*)
apply *clarify*
using *NU-P-E ent* ‹$E = nxs$› ‹$E = mset\ (tl\ outl')$›[*symmetric*] ‹$outl'$! $x1d \in\# E$›
by (*auto simp*: *intro*!: *entails-uminus-filter-to-poslev-can-remove*[*of - - M'*]
filter-to-poslev-conflict-min-analysis-inv simp del: *diff-union-swap2*)
qed

have
outl'-F: ‹$outl'$! $i \in\# F$› (**is** *?out*) **and**
outl'-$\mathcal{L}_{all}$: ‹$outl'$! $i \in\# \mathcal{L}_{all}\ \mathcal{A}$› (**is** *?out-L*)
if
R: ‹$(S, T) \in R$› **and**
‹*case S of* $(nxs, i, s, outl) \Rightarrow i < length\ outl$› **and**
‹*case T of* $(D, D') \Rightarrow D' \neq \{\#\}$› **and**
‹*minimize-and-extract-highest-lookup-conflict-inv S*› **and**
‹*iterate-over-conflict-inv M D T*› **and**
st:
‹$T = (F', F)$›
‹$S2 = (cach, outl')$›
‹$S1 = (i, S2)$›
‹$S = (D', S1)$›
‹$i < length\ outl'$›
for *S T F' T1 F highest' D' S1 i S2 cach S3 highest outl'*
proof −
have *?out* **and** ‹$F \subseteq\# D$›
using *R* ‹$i < length\ outl'$› **unfolding** *R-def st*
by (*auto simp*: *set-drop-conv*)
show *?out*
using ‹*?out*› **.**
then have ‹$outl'$! $i \in\# D$›
using ‹$F \subseteq\# D$› **by** *auto*
then show *?out-L*
using *lits-D* **by** (*auto dest*!: *multi-member-split simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
qed

have
not-red: ‹$\neg\ red \Longrightarrow ((D', i + 1, cachr, outl'), F',$
remove1-mset L F) $\in R$› (**is** ‹- $\Longrightarrow$ *?not-red*›) **and**
red: ‹$\neg\ \neg\ red \Longrightarrow$
((*remove1-mset* ($outl'$! i) D', i, *cachr*, *delete-index-and-swap* $outl'$ i),
remove1-mset L F', *remove1-mset L F*) $\in R$› (**is** ‹- $\Longrightarrow$ *?red*›) **and**
del: ‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ ($outl'$! i, D')› (**is** *?del*)
if
R: ‹$(S, T) \in R$› **and**
‹*case S of* $(nxs, i, s, outl) \Rightarrow i < length\ outl$› **and**
‹*case T of* $(D, D') \Rightarrow D' \neq \{\#\}$› **and**
‹*minimize-and-extract-highest-lookup-conflict-inv S*› **and**
‹*iterate-over-conflict-inv M D T*› **and**

st:
⟨$T = (F', F)$⟩
⟨$S2 = (cach, outl')$⟩
⟨$S1 = (i, S2)$⟩
⟨$S = (D', S1)$⟩
⟨$cachred1 = (stack, red)$⟩
⟨$cachred = (cachr, cachred1)$⟩ **and**
⟨$i < length\ outl'$⟩ **and**
L: ⟨$(outl' ! i, L) \in Id$⟩ **and**
⟨$outl' ! i \in\# \mathcal{L}_{all}\ \mathcal{A}$⟩ **and**
cach: ⟨$(cachred, red') \in (?red\ F'\ L)$⟩
for *S T F′ T1 F D′ S1 i S2 cach S3 highest outl′ L cachred red′ cachr cachred1 stack red*
proof −
have ⟨$L = outl' ! i$⟩ **and**
⟨$i \leq length\ outl'$⟩ **and**
⟨$mset\ (tl\ outl') \subseteq\# D$⟩ **and**
⟨$mset\ (drop\ i\ outl') \subseteq\# mset\ (tl\ outl')$⟩ **and**
F: ⟨$F = mset\ (drop\ i\ outl')$⟩ **and**
F′: ⟨$F' = mset\ (tl\ outl')$⟩ **and**
⟨*conflict-min-analysis-inv M′ cach ?NU (mset (tl outl′))*⟩ **and**
⟨$NU' + NUE \models pm$ *add-mset K (mset (tl outl′))*⟩ **and**
⟨$D' = mset\ (tl\ outl')$⟩ **and**
⟨$0 < i$⟩ **and**
[*simp*]: ⟨$D' = F'$⟩ **and**
F′-D: ⟨$F' \subseteq\# D$⟩ **and**
F′-F: ⟨$F \subseteq\# F'$⟩ **and**
⟨$outl' \neq []$⟩ ⟨$outl' ! 0 = K$⟩
using *R L* **unfolding** *R-def st*
by *clarify+*

have [*simp*]: ⟨$L = outl' ! i$⟩
using *L* **by** *fast*
have *L-F*: ⟨*mset (drop (Suc i) outl′) = remove1-mset L F*⟩
unfolding *F*
apply (*subst* (*2*) *Cons-nth-drop-Suc*[*symmetric*])
using ⟨$i < length\ outl'$⟩ *F′-D*
by (*auto*)
have ⟨*remove1-mset (outl′ ! i) F* $\subseteq\# F'$⟩
using ⟨$F \subseteq\# F'$⟩
by *auto*
have ⟨$red' = red$⟩ **and**
red: ⟨$red \longrightarrow NU' + NUE \models pm$ *remove1-mset L (add-mset K F′)* ∧
conflict-min-analysis-inv M′ cachr ?NU (remove1-mset L F′)⟩ **and**
not-red: ⟨$\neg\ red \longrightarrow NU' + NUE \models pm$ *add-mset K F′* ∧ *conflict-min-analysis-inv M′ cachr ?NU F′*⟩
using *cach*
unfolding *st*
by *auto*
have [*simp*]: ⟨*mset (drop (Suc i) (swap outl′ (Suc 0) i)) = mset (drop (Suc i) outl′)*⟩
by (*subst drop-swap-irrelevant*) (*use* ⟨$0 < i$⟩ **in** *auto*)
have [*simp*]: ⟨*mset (tl (swap outl′ (Suc 0) i)) = mset (tl outl′)*⟩
apply (*cases outl′*; *cases i*)
using ⟨$i > 0$⟩ ⟨$outl' \neq []$⟩ ⟨$i < length\ outl'$⟩
apply (*auto simp*: *WB-More-Refinement-List.swap-def*)
unfolding *WB-More-Refinement-List.swap-def*[*symmetric*]

```
    by (auto simp: )
  have [simp]: ‹mset (take (Suc i) (tl (swap outl' (Suc 0) i))) =  mset (take (Suc i) (tl outl'))›
    using ‹i > 0› ‹outl' ≠ []› ‹i < length outl'›
    by (auto simp: take-tl take-swap-relevant tl-swap-relevant)
  have [simp]: ‹mset (take i (tl (swap outl' (Suc 0) i))) =  mset (take i (tl outl'))›
    using ‹i > 0› ‹outl' ≠ []› ‹i < length outl'›
    by (auto simp: take-tl take-swap-relevant tl-swap-relevant)

  have [simp]: ‹¬ Suc 0 < a ⟷ a = 0 ∨ a = 1› for a :: nat
    by auto
   show ?not-red if ‹¬red›
    using ‹i < length outl'› F'-D L-F ‹remove1-mset (outl' ! i) F ⊆# F'› not-red that
      ‹i > 0› ‹outl' ! 0 = K›
    by (auto simp: R-def F[symmetric] F'[symmetric]  drop-swap-irrelevant)

  have [simp]: ‹length (delete-index-and-swap outl' i) = length outl' − 1›
    by auto
  have last: ‹¬ length outl' ≤ Suc i ⟹last outl' ∈ set (drop (Suc i) outl')›
    by (metis List.last-in-set drop-eq-Nil last-drop not-le-imp-less)
  then have H: ‹mset (drop i (delete-index-and-swap outl' i)) = mset (drop (Suc i) outl')›
    using ‹i < length outl'›
    by (cases ‹drop (Suc i) outl' = []›)
      (auto simp: butlast-list-update mset-butlast-remove1-mset)
  have H': ‹mset (tl (delete-index-and-swap outl' i)) = remove1-mset (outl' ! i) (mset (tl outl'))›
    apply (rule mset-tl-delete-index-and-swap)
    using ‹i < length outl'› ‹i > 0› by fast+
  have [simp]: ‹Suc 0 < i ⟹ delete-index-and-swap outl' i ! Suc 0 = outl' ! Suc 0›
    using ‹i < length outl'› ‹i > 0›
    by (auto simp: nth-butlast)
  have ‹remove1-mset (outl' ! i) F ⊆# remove1-mset (outl' ! i) F'›
    using ‹F ⊆# F'›
    using mset-le-subtract by blast
  have [simp]: ‹delete-index-and-swap outl' i ≠ []›
    using ‹outl' ≠ []› ‹i > 0› ‹i < length outl'›
    by (cases outl') (auto simp: butlast-update'[symmetric] split: nat.splits)
  have [simp]: ‹delete-index-and-swap outl' i ! 0 = outl' ! 0›
    using  ‹outl' ! 0 = K› ‹i < length outl'› ‹i > 0›
    by (auto simp: butlast-update'[symmetric] nth-butlast)
  have ‹(outl' ! i) ∈# F'›
    using ‹i < length outl'› ‹i > 0› unfolding F' by (auto simp: nth-in-set-tl)
  then show ?red if ‹¬¬red›
    using ‹i < length outl'› F'-D L-F ‹remove1-mset (outl' ! i) F ⊆# remove1-mset (outl' ! i) F'›
      red that ‹i > 0› ‹outl' ! 0 = K› unfolding R-def
    by (auto simp: R-def F[symmetric] F'[symmetric] H H' drop-swap-irrelevant
        simp del: delete-index-and-swap.simps)

  have ‹outl' ! i ∈# ℒ_all 𝒜› ‹outl' ! i ∈# D›
    using ‹(outl' ! i) ∈# F'› F'-D lits-D
    by (force simp: literals-are-in-ℒ_in-add-mset
        dest!: multi-member-split[of ‹outl' ! i› D])+
  then show ?del
    using ‹(outl' ! i) ∈# F'› lits-D F'-D tauto
    by (auto simp: delete-from-lookup-conflict-pre-def
        literals-are-in-ℒ_in-add-mset)
qed
show ?thesis
```

unfolding *minimize-and-extract-highest-lookup-conflict-def iterate-over-conflict-def*
apply (*refine-vcg WHILEIT-refine*[**where** $R = R$])
subgoal by (*rule init-args-ref*)
subgoal for s' s **by** (*rule init-lo-inv*)
subgoal by (*rule cond*)
subgoal by *auto*
subgoal by (*rule outl'-F*)
subgoal by (*rule outl'-*$\mathcal{L}_{all}$)
apply (*rule redundant*; *assumption*)
subgoal by *auto*
subgoal by (*rule not-red*)
subgoal by (*rule del*)
subgoal
by (*rule red*)
subgoal for x x' *x1 x2 x1a x2a x1b x2b x1c x2c*
unfolding *R-def* **by** (*cases x1b*) *auto*
done
qed

definition *cach-refinement-list*
:: ‹*nat multiset* ⇒ (*minimize-status list* × (*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* ⇒ ((*minimize-status list* × *nat list*) × *minimize-status list*) *set*›
where
‹*cach-refinement-nonull* $\mathcal{A}$ = {((*cach*, *support*), *cach'*). *cach* = *cach'* ∧
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}$}›

definition *cach-refinement*
:: ‹*nat multiset* ⇒ ((*minimize-status list* × *nat list*) × (*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'*).
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
(∀ *L* ∈# $\mathcal{A}_{in}$. *L* < *length cach* ∧ *cach* ! *L* = *cach' L*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}_{in}$}›
unfolding *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*
apply (*rule*; *rule*)
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply *blast*
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply (*rule-tac b*=*x1a* **in** *relcomp.relcompI*)
apply *blast*
apply *blast*
done

lemma *in-cach-refinement-alt-def*:
‹((*cach*, *support*), *cach'*) ∈ *cach-refinement* $\mathcal{A}_{in}$ ⟷

$(cach, cach') \in$ *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* $\mathcal{A}_{in}$›
by (*auto simp*: *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*)

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 −
show *?thesis*
using *assms* **by** (*auto simp*: *cach-refinement-alt-def* *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *conflict-min-cach-l-pre-def*)
qed

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*›
by (*intro frefI nres-relI*) (*auto simp*: *map-fun-rel-def*
in-cach-refinement-alt-def cach-refinement-list-def conflict-min-cach-l-def)

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 −
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* ‘# *mset n*)› **and**
dist: ‹*distinct n*›
using *assms*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def inj-on-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset n*)›
by (*subst distinct-image-mset-inj*)

(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset n*))›
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)
qed

lemma *conflict-min-cach-set-failed*:
‹(*uncurry conflict-min-cach-set-failed-l*, *uncurry* (*RETURN oo conflict-min-cach-set-failed*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
apply (*intro frefI nres-relI*)
apply (*auto simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*
conflict-min-cach-set-failed-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)
by (*fastforce dest*: *subset-add-mset-notin-subset-mset*)+

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*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
apply (*intro frefI nres-relI*)
apply (*auto simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*
conflict-min-cach-set-removable-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)
by (*fastforce dest*: *subset-add-mset-notin-subset-mset*)+

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* $\leq$ *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* $\geq$ *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'* $\Longrightarrow$
a1' < *length ba* $\Longrightarrow$
(*a1'a*, *a2'a*) = *ba* ! *a1'* $\Longrightarrow$
a1'a ∈# *dom-m a*›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2'* $\Longrightarrow$
a1' < *length ba* $\Longrightarrow$
(*a1'a*, *xx*, *a2'a*, *yy*) = *ba* ! *a1'* $\Longrightarrow$
a2'a − *Suc 0* < *length* (*a* $\propto$ *a1'a*)›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

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* $\leq$ *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 −
have *subset-mset-add-new*: ‹*a* ∉# *A* $\Longrightarrow$ *a* ∈# *B* $\Longrightarrow$ *add-mset a A* ⊆# *B* ⟷ *A* ⊆# *B*› **for** *a A B*
by (*metis insert-DiffM insert-subset-eq-iff subset-add-mset-notin-subset*)
have [*refine0*]: ‹((*0*, *x2c*), *0*, *x2a*) ∈ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$›
if ‹(*x2c*, *x2a*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **for** *x2c x2a*
using *that* **by** *auto*
have *le-length-arena*: ‹*x1g* + *x2g* − *1* < *length x1c*› (**is** *?le*) **and**
is-lit: ‹*arena-lit-pre x1c* (*x1g* + *x2g* − *1*)› (**is** *?lit*) **and**
isA: ‹*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *1*)) ∈# $\mathcal{A}_{in}$› (**is** *?A*) **and**
final: ‹*conflict-min-cach-set-failed-l x2e*
(*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *1*)))
$\leq$ *SPEC*

(λ*cach.*
RETURN (*x1e* + *1*, *cach*)
$\leq$ *SPEC*
(λ*c.* (*c*, *x1d* + *1*, *x2d*
(*atm-of* (*x1a* $\propto$ *x1f* ! (*x2f* − *1*)) := *SEEN-FAILED*))
$\in$ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$))› (**is** *?final*) **and**
ge1: ‹*x1g* + *x2g* $\geq$ *1*›
if
‹*case y of* (*x*, *xa*) $\Rightarrow$ (*case x of* (*N*, *ana*) $\Rightarrow$ λ*cach. length ana* $\leq$ *1* + *uint32-max div 2*) *xa*› **and**
xy: ‹(*x*, *y*) $\in$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *ana-lookups-rel NU*
$\times_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**
st:
‹*x1* = (*x1a*, *x2*)›
‹*y* = (*x1*, *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = (*x1b*, *x2c*)›
‹*x'* = (*x1d*, *x2d*)›
‹*xa* = (*x1e*, *x2e*)›
‹*x2f2* = (*x2f*, *x2f3*)›
‹*x2f0* = (*x2f1*, *x2f2*)›
‹*x2* ! *x1d* = (*x1f*, *x2f0*)›
‹*x2g0* = (*x2g*, *x2g2*)›
‹*x2b* ! *x1e* = (*x1g*, *x2g0*)› **and**
xax': ‹(*xa*, *x'*) $\in$ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**
cond: ‹*case xa of* (*i*, *cach*) $\Rightarrow$ *i* < *length x2b*› **and**
cond': ‹*case x' of* (*i*, *cach*) $\Rightarrow$ *i* < *length x2*› **and**
inv: ‹*case x' of* (-, *x*) $\Rightarrow$ *mark-failed-lits-stack-inv x1a x2 x*› **and**
le: ‹*x1d* < *length x2*› ‹*x1e* < *length x2b*› **and**
atm: ‹*atm-of* (*x1a* $\propto$ *x1f* ! (*x2f* − *1*)) $\in$# $\mathcal{A}_{in}$›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c xa x' x1d x2d x1e x2e x1f x2f x1g x2g*
x2f0 x2f1 x2f2 x2f3 x2g0 x2g1 x2g2 x2g3
proof −
obtain *i cach* **where** *x'*: ‹*x'* = (*i*, *cach*)› **by** (*cases x'*)
have [*simp*]:
‹*x1* = (*x1a*, *x2*)›
‹*y* = ((*x1a*, *x2*), *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = ((*x1c*, *x2b*), *x2c*)›
‹*x'* = (*x1d*, *x2d*)›
‹*xa* = (*x1d*, *x2e*)›
‹*x1f* = *x1g*›
‹*x1e* = *x1d*›
‹*x2f0* = (*x2f1*, *x2f*, *x2f3*)›
‹*x2g* = *x2f*›
‹*x2g0* = (*x2g*, *x2g2*)› **and**
st': ‹*x2* ! *x1d* = (*x1g*, *x2f0*)› **and**
cach:‹(*x2e*, *x2d*) $\in$ *cach-refinement* $\mathcal{A}_{in}$› **and**
‹(*x2c*, *x2a*) $\in$ *cach-refinement* $\mathcal{A}_{in}$› **and**
x2f0-x2g0: ‹((*x1g*, *x2g*, *x2g2*), (*x1f*, *x2f1*, *x2f*, *x2f3*)) $\in$ *ana-lookup-rel NU*›
using *xy st xax' param-nth*[*of x1e x2 x1d x2b* ‹*ana-lookup-rel NU*›] *le*
by (*auto intro*: *simp*: *ana-lookup-rel-alt-def*)

have *arena*: ‹*valid-arena x1c x1a vdom*›
using *xy* **unfolding** *st* **by** *auto*
have ‹*x2* ! *x1e* $\in$ *set x2*›
using *le*

by *auto*
then have ‹*x2* ! *x1d* ∈ *set x2*› and
x2f: ‹*x2f* ≤ *length* (*x1a* ∝ *x1f*)› and
x1f: ‹*x1g* ∈# *dom-m x1a*› and
x2g: ‹*x2g* > *0*› and
x2g-u1-le: ‹*x2g* − *1* < *length* (*x1a* ∝ *x1f*)›
using *inv le x2f0-x2g0 nth-mem*[*of x1d x2*]
unfolding *mark-failed-lits-stack-inv-def x′ prod.case st st′*
by (*auto simp del*: *nth-mem simp*: *st′ ana-lookup-rel-alt-def split*: *if-splits*
dest!: *bspec*[*of* ‹*set x2*› - ‹(-, -, -, -)›])

have ‹*is-Lit* (*x1c* ! (*x1g* + (*x2g* − *1*)))›
by (*rule arena-lifting*[*OF arena x1f*]) (*use x2f x2g x2g-u1-le* in *auto*)
then show *?le* and *?A*
using *arena-lifting*[*OF arena x1f*] *le x2f x1f x2g atm x2g-u1-le*
by (*auto simp*: *arena-lit-def*)
show *?lit*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of* - *x1f*])
(*use arena x1f x2f x2g x2g-u1-le* in ‹*auto intro*!: *exI*[*of* - *x1a*] *exI*[*of* - *vdom*]›)
show ‹*x1g* + *x2g* ≥ *1*›
using *x2g* by *auto*
have [*simp*]: ‹*arena-lit x1c* (*x1g* + *x2g* − *Suc 0*) = *x1a* ∝ *x1g* ! (*x2g* − *Suc 0*)›
using *that x1f x2f x2g x2g-u1-le* by (*auto simp*: *arena-lifting*[*OF arena*])
have ‹*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *Suc 0*)) < *length* (*fst x2e*)›
using ‹*?A*› *cach* by (*auto simp*: *cach-refinement-alt-def dest*: *multi-member-split*)

then show *?final*
using ‹*?le*› ‹*?A*› *cach x1f x2g-u1-le x2g assms*
apply −
apply (*rule conflict-min-cach-set-failed*[*of* $\mathcal{A}_{in}$, *THEN fref-to-Down-curry*, *THEN order-trans*])
by (*cases x2e*)
(*auto simp*: *cach-refinement-alt-def RETURN-def conc-fun-RES*
arena-lifting[*OF arena*] *subset-mset-add-new*)
qed

show *?thesis*
unfolding *isa-mark-failed-lits-stack-def mark-failed-lits-stack-def uncurry-def*
from-ana-ref-id-def
apply (*rewrite at* ‹*let* - = *length* - *in* -› *Let-def*)
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c xa x′ x1d x2d x1e x2e*
by (*auto simp*: *list-rel-imp-same-length*)
subgoal by *auto*
subgoal by (*rule ge1*)
subgoal by (*rule le-length-arena*)
subgoal
by (*rule is-lit*)
subgoal
by (*rule final*)
subgoal by *auto*
done

qed

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*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* < *uint64-max*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

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 $i \geq$ *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)›
unfolding *isa-lit-redundant-rec-wl-lookup-def from-ana-ref-id-def Let-def*
by (*auto simp*: *Let-def*)

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 ≠ [])
(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 ∈# dom-m NU);

$ASSERT(length\ (NU \propto C) > k)$; — >= 2 would work too
$ASSERT\ (NU \propto C\ !\ k \in lits\text{-}of\text{-}l\ M)$;
$ASSERT(NU \propto C\ !\ k \in\#\ \mathcal{L}_{all}\ \mathcal{A})$;
$ASSERT(literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (NU \propto C)))$;
$ASSERT(length\ (NU \propto C) \leq Suc\ (uint32\text{-}max\ div\ 2))$;
$ASSERT(len \leq length\ (NU \propto C))$; — makes the refinement easier
$let\ (C,k,\ i,\ len) = (C,k,i,len)$;
$let\ C = NU \propto C$;
$if\ i \geq len$
$then$
$RETURN(cach\ (atm\text{-}of\ (C\ !\ k) := SEEN\text{-}REMOVABLE),\ butlast\ analyse,\ True)$
$else\ do$ {
$let\ (L,\ analyse) = get\text{-}literal\text{-}and\text{-}remove\text{-}of\text{-}analyse\text{-}wl2\ C\ analyse$;
$ASSERT(L \in\#\ \mathcal{L}_{all}\ \mathcal{A})$;
$let\ b = \neg level\text{-}in\text{-}lbd\ (get\text{-}level\ M\ L)\ lbd$;
$if\ (get\text{-}level\ M\ L = zero\text{-}uint32\text{-}nat\ \vee$
$conflict\text{-}min\text{-}cach\ cach\ (atm\text{-}of\ L) = SEEN\text{-}REMOVABLE\ \vee$
$atm\text{-}in\text{-}conflict\ (atm\text{-}of\ L)\ D)$
$then\ RETURN\ (cach,\ analyse,\ False)$
$else\ if\ b \vee conflict\text{-}min\text{-}cach\ cach\ (atm\text{-}of\ L) = SEEN\text{-}FAILED$
$then\ do$ {
$ASSERT(mark\text{-}failed\text{-}lits\text{-}stack\text{-}inv2\ NU\ analyse\ cach)$;
$cach \leftarrow mark\text{-}failed\text{-}lits\text{-}wl\ NU\ analyse\ cach$;
$RETURN\ (cach,\ [],\ False)$
}
$else\ do$ {
$ASSERT(-\ L \in lits\text{-}of\text{-}l\ M)$;
$C \leftarrow get\text{-}propagation\text{-}reason\ M\ (-L)$;
$case\ C\ of$
$Some\ C \Rightarrow do$ {
$ASSERT(C \in\#\ dom\text{-}m\ NU)$;
$ASSERT(length\ (NU \propto C) \geq 2)$;
$ASSERT(literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (NU \propto C)))$;
$ASSERT(length\ (NU \propto C) \leq Suc\ (uint32\text{-}max\ div\ 2))$;
$RETURN\ (cach,\ analyse\ @\ [lit\text{-}redundant\text{-}reason\text{-}stack2\ (-L)\ NU\ C],\ False)$
}
$|\ None \Rightarrow do$ {
$ASSERT(mark\text{-}failed\text{-}lits\text{-}stack\text{-}inv2\ NU\ analyse\ cach)$;
$cach \leftarrow mark\text{-}failed\text{-}lits\text{-}wl\ NU\ analyse\ cach$;
$RETURN\ (cach,\ [],\ False)$
}
}
}
})
$(cach,\ analysis,\ False)$⟩
unfolding *lit-redundant-rec-wl-lookup-def Let-def* **by** *auto*

lemma *valid-arena-nempty*:
⟨$valid\text{-}arena\ arena\ N\ vdom \Longrightarrow i \in\#\ dom\text{-}m\ N \Longrightarrow N \propto i \neq []$⟩
using *arena-lifting(19)[of arena N vdom i]*
arena-lifting(4)[of arena N vdom i]
by *auto*

lemma *isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup*:
assumes ⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows ⟨$(uncurry5\ isa\text{-}lit\text{-}redundant\text{-}rec\text{-}wl\text{-}lookup,\ uncurry5\ (lit\text{-}redundant\text{-}rec\text{-}wl\text{-}lookup\ \mathcal{A})) \in$

$[\lambda(((((-, N), -), -), -), -).$ *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 −
have *isa-mark-failed-lits-stack*: ‹*isa-mark-failed-lits-stack x2e x2z x1l*
≤ ⇓ (*cach-refinement* $\mathcal{A}$)
(*mark-failed-lits-wl x2 x2y x1j*)›
if
‹*case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*uu-*, *N*) ⇒
λ- - - -.
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) '# *ran-m N*)) *xa*)
xa)
xa)
xa› **and**
‹(*x*, *y*)
∈ *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*› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)› **and**
‹(*xa*, *x'*) ∈ *cach-refinement* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *bool-rel*)› **and**
‹*case xa of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*lit-redundant-rec-wl-inv2 x1d x2 x2a x'*› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x'* = (*x1j*, *x2j*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*xa* = (*x1l*, *x2l*)› **and**
‹*x1k* ≠ []› **and**
‹*x1m* ≠ []› **and**
‹*x2o* = (*x1p*, *x2p*)› **and**
‹*x2n* = (*x1o*, *x2o*)› **and**
‹*ana-lookup-conv x2* (*last x1k*) = (*x1n*, *x2n*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*last x1m* = (*x1q*, *x2q*)› **and**
‹*x1n* ∈# *dom-m x2*› **and**
‹*x1o* < *length* (*x2* ∝ *x1n*)› **and**
‹*x2* ∝ *x1n* ! *x1o* ∈ *lits-of-l x1d*› **and**
‹*x2* ∝ *x1n* ! *x1o* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**

```
    ‹literals-are-in-ℒin 𝒜 (mset (x2 ∝ x1n))› and
    ‹length (x2 ∝ x1n) ≤ Suc (uint-max div 2)› and
    ‹x2p ≤ length (x2 ∝ x1n)› and
    ‹arena-is-valid-clause-idx x2e (fst (last x1m))› and
    ‹x2t = (x1u, x2u)› and
    ‹x2s = (x1t, x2t)› and
    ‹(x1n, x1o, x1p, x2p) = (x1s, x2s)› and
    ‹x2w = (x1x, x2x)› and
    ‹x2v = (x1w, x2w)› and
    ‹ana-lookup-conv-lookup x2e (x1q, x1r, x2r) = (x1v, x2v)› and
    ‹x1v < length x2e› and
    ‹arena-is-valid-clause-idx x2e x1v› and
    ‹arena-lit-pre x2e (x1v + x1w)› and
    ‹¬ nat-of-uint64-conv x2x ≤ x1x› and
    ‹¬ x2u ≤ x1u› and
    ‹isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m› and
    ‹get-literal-and-remove-of-analyse-wl2 (x2 ∝ x1s) x1k = (x1y, x2y)› and
    ‹isa-get-literal-and-remove-of-analyse-wl x2e x1m = (x1z, x2z)› and
    ‹x1y ∈# ℒall 𝒜› and    ‹get-level-pol-pre (x1i, x1z)› and
    ‹atm-in-conflict-lookup-pre (atm-of x1z) x2f› and
    ‹conflict-min-cach-l-pre (x1l, atm-of x1z)› and
    ‹¬ (get-level-pol x1i x1z = zero-uint32-nat ∨
  conflict-min-cach-l x1l (atm-of x1z) = SEEN-REMOVABLE ∨
  atm-in-conflict-lookup (atm-of x1z) x2f)› and
    ‹¬ (get-level x1d x1y = zero-uint32-nat ∨
  conflict-min-cach x1j (atm-of x1y) = SEEN-REMOVABLE ∨
  atm-in-conflict (atm-of x1y) x2a)› and
    ‹¬ level-in-lbd (get-level-pol x1i x1z) x2i ∨
     conflict-min-cach-l x1l (atm-of x1z) = SEEN-FAILED› and
    ‹¬ level-in-lbd (get-level x1d x1y) x2d ∨
     conflict-min-cach x1j (atm-of x1y) = SEEN-FAILED› and
    inv2: ‹mark-failed-lits-stack-inv2 x2 x2y x1j› and
    ‹length x1m ≤ 1+ uint32-max div 2›
   for x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g
 x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
 x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z
 x2z
 proof −
  have [simp]: ‹x2z = x2y›
    using that
    by (auto simp: isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

  obtain x2y0 where
    x2z: ‹(x2y, x2y0) ∈ ana-lookups-rel x2› and
    inv: ‹mark-failed-lits-stack-inv x2 x2y0 x1j›
    using inv2 unfolding mark-failed-lits-stack-inv2-def
    by blast
  have 1: ‹mark-failed-lits-wl x2 x2y x1j  = mark-failed-lits-wl x2 x2y0 x1j›
    unfolding mark-failed-lits-wl-def by auto
  show ?thesis
    unfolding 1
    apply (rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack[THEN
  fref-to-Down-curry2, of 𝒜 x2 x2y0 x1j x2e x2z x1l vdom x2, THEN order-trans])
    subgoal using assms by fast
    subgoal using that x2z by (auto simp: list-rel-imp-same-length[symmetric]
```

isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have *isa-mark-failed-lits-stack2*: ‹*isa-mark-failed-lits-stack x2e x2z x1l*
$\leq \Downarrow$ (*cach-refinement* $\mathcal{A}$) (*mark-failed-lits-wl x2 x2y x1j*)›
if
‹*case y of*
(*x*, *xa*) $\Rightarrow$
(*case x of*
(*x*, *xa*) $\Rightarrow$
(*case x of*
(*x*, *xa*) $\Rightarrow$
(*case x of*
(*x*, *xa*) $\Rightarrow$
(*case x of*
(*uu-*, *N*) $\Rightarrow$
λ- - - -.
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* $\circ$ *fst*) '# *ran-m N*)) *xa*)
xa)
xa)
xa› **and**
‹(*x*, *y*)
$\in$ *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› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)› **and**
‹(*xa*, *x'*) $\in$ *cach-refinement* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *bool-rel*)› **and** ‹*case xa of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse*
$\neq$ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*True*› **and**
‹*lit-redundant-rec-wl-inv2 x1d x2 x2a x'*› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x'* = (*x1j*, *x2j*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*xa* = (*x1l*, *x2l*)› **and**
‹*x1k* $\neq$ []› **and**
‹*x1m* $\neq$ []› **and**
‹*x2o* = (*x1p*, *x2p*)› **and**

⟨*x2n* = (*x1o*, *x2o*)⟩ **and**
⟨*ana-lookup-conv x2* (*last x1k*) = (*x1n*, *x2n*)⟩ **and**
⟨*x2q* = (*x1r*, *x2r*)⟩ **and**
⟨*last x1m* = (*x1q*, *x2q*)⟩ **and**
⟨*x1n* ∈# *dom-m x2*⟩ **and**
⟨*x1o* < *length* (*x2* ∝ *x1n*)⟩ **and**
⟨*x2* ∝ *x1n* ! *x1o* ∈ *lits-of-l x1d*⟩ **and**
⟨*x2* ∝ *x1n* ! *x1o* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩ **and**
⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*x2* ∝ *x1n*))⟩ **and**
⟨*length* (*x2* ∝ *x1n*) ≤ *Suc* (*uint-max div 2*)⟩ **and**
⟨*x2p* ≤ *length* (*x2* ∝ *x1n*)⟩ **and**
⟨*arena-is-valid-clause-idx x2e* (*fst* (*last x1m*))⟩ **and**
⟨*x2t* = (*x1u*, *x2u*)⟩ **and**
⟨*x2s* = (*x1t*, *x2t*)⟩ **and**
⟨(*x1n*, *x1o*, *x1p*, *x2p*) = (*x1s*, *x2s*)⟩ **and**
⟨*x2w* = (*x1x*, *x2x*)⟩ **and**
⟨*x2v* = (*x1w*, *x2w*)⟩ **and**
⟨*ana-lookup-conv-lookup x2e* (*x1q*, *x1r*, *x2r*) = (*x1v*, *x2v*)⟩ **and**
⟨*x1v* < *length x2e*⟩ **and**
⟨*arena-is-valid-clause-idx x2e x1v*⟩ **and**
⟨*arena-lit-pre x2e* (*x1v* + *x1w*)⟩ **and**
⟨¬ *nat-of-uint64-conv x2x* ≤ *x1x*⟩ **and**
⟨¬ *x2u* ≤ *x1u*⟩ **and**
⟨*isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m*⟩ **and**
⟨*get-literal-and-remove-of-analyse-wl2* (*x2* ∝ *x1s*) *x1k* = (*x1y*, *x2y*)⟩ **and**
⟨*isa-get-literal-and-remove-of-analyse-wl x2e x1m* = (*x1z*, *x2z*)⟩ **and**
⟨*x1y* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩ **and** ⟨*get-level-pol-pre* (*x1i*, *x1z*)⟩ **and**
⟨*atm-in-conflict-lookup-pre* (*atm-of x1z*) *x2f*⟩ **and**
⟨*conflict-min-cach-l-pre* (*x1l*, *atm-of x1z*)⟩ **and**
⟨¬ (*get-level-pol x1i x1z* = *zero-uint32-nat* ∨
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-REMOVABLE* ∨
atm-in-conflict-lookup (*atm-of x1z*) *x2f*)⟩ **and**
⟨¬ (*get-level x1d x1y* = *zero-uint32-nat* ∨
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1y*) *x2a*)⟩ **and**
⟨¬ (¬ *level-in-lbd* (*get-level-pol x1i x1z*) *x2i* ∨
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-FAILED*)⟩ **and**
⟨¬ (¬ *level-in-lbd* (*get-level x1d x1y*) *x2d* ∨
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-FAILED*)⟩ **and**
⟨− *x1y* ∈ *lits-of-l x1d*⟩ **and**
⟨(*xb*, *x'a*) ∈ ⟨*nat-rel*⟩*option-rel*⟩ **and**
⟨*xb* = *None*⟩ **and**
⟨*x'a* = *None*⟩ **and**
inv2: ⟨*mark-failed-lits-stack-inv2 x2 x2y x1j*⟩ **and**
⟨*length x1m* ≤ *1* + *uint-max div 2*⟩
for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z x2z xb x'a*
proof −
have [*simp*]: ⟨*x2z* = *x2y*⟩
using *that*
by (*auto simp*: *isa-get-literal-and-remove-of-analyse-wl-def*
get-literal-and-remove-of-analyse-wl2-def)

obtain *x2y0* **where**

x2z: ‹(*x2y*, *x2y0*) ∈ *ana-lookups-rel x2*› **and**
inv: ‹*mark-failed-lits-stack-inv x2 x2y0 x1j*›
using *inv2* **unfolding** *mark-failed-lits-stack-inv2-def*
by *blast*
have *1*: ‹*mark-failed-lits-wl x2 x2y x1j* = *mark-failed-lits-wl x2 x2y0 x1j*›
unfolding *mark-failed-lits-wl-def* **by** *auto*
show *?thesis*
unfolding *1*
apply (*rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *x2 x2y0 x1j x2e x2z x1l vdom x2*, *THEN order-trans*])
subgoal using *assms* **by** *fast*
subgoal using *that x2z* **by** (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have [*refine0*]: ‹*get-propagation-reason-pol M′ L′*
≤ ⇓ (⟨*Id*⟩*option-rel*)
(*get-propagation-reason M L*)›
if ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and** ‹(*L′*, *L*) ∈ *Id*› **and** ‹*L* ∈ *lits-of-l M*›
for *M M′ L L′*
using *get-propagation-reason-pol*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*, *of M L M′ L′*] *that* **by** *auto*
note [*simp*]=*get-literal-and-remove-of-analyse-wl-def isa-get-literal-and-remove-of-analyse-wl-def*
arena-lifting **and** [*split*] = *prod.splits*

show *?thesis*
supply [[*goals-limit=1*]] *ana-lookup-conv-def*[*simp*] *ana-lookup-conv-lookup-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 add*]
unfolding *isa-lit-redundant-rec-wl-lookup-alt-def lit-redundant-rec-wl-lookup-alt-def uncurry-def*
from-ana-ref-id-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m
by (*auto simp*: *arena-lifting*)
subgoal by (*auto simp*: *trail-pol-alt-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*
lit-redundant-rec-wl-inv2-def)
subgoal by (*auto simp*: *ana-lookup-conv-lookup-pre-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q

x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x
apply (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def*)
by (*rule-tac x* = ‹*x1s*› **in** *exI*; *auto simp*: *valid-arena-nempty*)+
subgoal by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def nat-of-uint64-conv-def lit-redundant-rec-wl-inv-def split*: *if-splits*)
subgoal using *assms*
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def bind-rule-complete-RES conc-fun-RETURN in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def intro*!: *conflict-min-cach-set-removable*[*of* $\mathcal{A}$,*THEN fref-to-Down-curry*, *THEN order-trans*]
dest: *List.last-in-set*)

subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x*
by (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def*
uint-max-def
arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def)
(*rule-tac x* = *x1s* **in** *exI*; *auto simp*: *uint-max-def*; *fail*)+
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*auto intro*!: *get-level-pol-pre simp*: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal by (*auto intro*!: *atm-in-conflict-lookup-pre simp*: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o*
by (*auto intro*!: *conflict-min-cach-l-pre simp*: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal
by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*] *nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*] *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
get-level-get-level-pol atms-of-def
get-literal-and-remove-of-analyse-wl2-def
split: *prod.splits*)
(*subst* (*asm*) *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *atms-of-def*; *fail*)+
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
subgoal by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*] *nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*] *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
get-level-get-level-pol atms-of-def
simp: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
apply (*rule isa-mark-failed-lits-stack*; *assumption*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def split*: *prod.splits*)
apply *assumption*
apply (*rule isa-mark-failed-lits-stack2*; *assumption*)
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z*

x2z xb x'a xc x'b
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def
simp: *valid-arena-nempty get-literal-and-remove-of-analyse-wl2-def*
lit-redundant-reason-stack-wl-lookup-def
lit-redundant-reason-stack2-def
intro!: *exI*[*of - x'b*] *bex-leI*[*of - x'b*])
subgoal premises *p* **for** *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u xb x'a xc x'b*
using *p*
by (*auto simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def get-literal-and-remove-of-analyse-wl2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why $\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *header-size* (*?N* $\propto$ *?i*) $\leq$ *?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *?i* $<$ *length ?arena*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *is-Size* (*?arena* ! (*?i* $-$ *SIZE-SHIFT*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *length* (*?N* $\propto$ *?i*) = *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$\rrbracket \Longrightarrow$ *?N* $\propto$ *?i* ! *?j* = *arena-lit ?arena* (*?i* + *?j*)
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$\rrbracket \Longrightarrow$ *is-Lit* (*?arena* ! (*?i* + *?j*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$\rrbracket \Longrightarrow$ *?i* + *?j* $<$ *length ?arena*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *?N* $\propto$ *?i* ! *0* = *arena-lit ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *is-Lit* (*?arena* ! *?i*)
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *?i* + *length* (*?N* $\propto$ *?i*) $\leq$ *length ?arena*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$\rrbracket \Longrightarrow$ *is-Pos* (*?arena* ! (*?i* $-$ *POS-SHIFT*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$\rrbracket \Longrightarrow$ *arena-pos ?arena ?i* $\leq$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *is-LBD* (*?arena* ! (*?i* $-$ *LBD-SHIFT*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *is-Act* (*?arena* ! (*?i* $-$ *ACTIVITY-SHIFT*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *is-Status* (*?arena* ! (*?i* $-$ *STATUS-SHIFT*))
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *SIZE-SHIFT* $\leq$ *?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *LBD-SHIFT* $\leq$ *?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *ACTIVITY-SHIFT* $\leq$ *?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *2* $\leq$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *Suc 0* $\leq$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *0* $\leq$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *Suc 0* $<$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *0* $<$ *arena-length ?arena ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ (*arena-status ?arena ?i* = *LEARNED*) = ($\neg$ *irred ?N ?i*)
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *arena-status ?arena ?i* $\neq$ *DELETED*
$\llbracket$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$\rrbracket \Longrightarrow$ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* $\propto$ *?i*) requires to be instantiated.
done
qed

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*(λD'. D' ⊆# D ∧
NU + *NUE* ⊨*pm add-mset K* D'))⟩
proof −
define I' **where**
⟨I' = (λ(*E*:: $'v$ *clause*, *f* :: $'v$ *clause*).
E ⊆# *D* ∧ *NU* + *NUE* ⊨*pm add-mset K E* ∧ *distinct-mset E* ∧ *distinct-mset f*)⟩

have *init-I'*: ⟨I' (*D*, *D*)⟩
using ⟨*NU* + *NUE* ⊨*pm add-mset K D*⟩ *dist* **unfolding** *I'-def highest-lit-def* **by** *auto*

have *red*: ⟨*is-literal-redundant-spec K NU NUE a x*
≤ *SPEC* (λ*red*. (*if* ¬ *red then RETURN* (*a*, *remove1-mset x aa*)
else RETURN (*remove1-mset x a*, *remove1-mset x aa*))
≤ *SPEC* (λs'. *iterate-over-conflict-inv M D* s' ∧ I' s' ∧
(s', *s*) ∈ *measure* (λ(*D*, D'). *size* D')))⟩
if
⟨*iterate-over-conflict-inv M D s*⟩ **and**
⟨I' *s*⟩ **and**
⟨*case s of* (*D*, D') ⇒ D' ≠ {#}⟩ **and**
⟨*s* = (*a*, *aa*)⟩ **and**
⟨*x* ∈# *aa*⟩
for *s a b aa x*
proof −
have ⟨*x* ∈# *a*⟩ ⟨*distinct-mset aa*⟩
using *that*
by (*auto simp*: *I'-def highest-lit-def*
eq-commute[*of* ⟨*get-level - -*⟩] *iterate-over-conflict-inv-def*
get-maximum-level-add-mset add-mset-eq-add-mset
dest!: *split*: *option.splits if-splits*)
then show *?thesis*
using *that*
by (*auto simp*: *is-literal-redundant-spec-def iterate-over-conflict-inv-def*
I'-def size-mset-remove1-mset-le-iff remove1-mset-add-mset-If
intro: *mset-le-subtract*)
qed

show *?thesis*
unfolding *iterate-over-conflict-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where**
R = ⟨*measure* (λ(*D* :: $'v$ *clause*, D':: $'v$ *clause*).
size D')⟩ **and**
I' = I'])
subgoal by *auto*
subgoal by (*auto simp*: *iterate-over-conflict-inv-def highest-lit-def*)
subgoal by (*rule init-I'*)
subgoal by (*rule red*)
subgoal unfolding *I'-def iterate-over-conflict-inv-def* **by** *auto*
subgoal unfolding *I'-def iterate-over-conflict-inv-def* **by** *auto*
done
qed

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
⟨$S''' \equiv$ *state*$_W$*-of* S''⟩
defines
⟨$M \equiv$ *get-trail-wl* S⟩ **and**
NU: ⟨$NU \equiv$ *get-clauses-wl* S⟩ **and**
NU'*-def*: ⟨$NU' \equiv$ *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: ⟨¬*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* ≠ []⟩ **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* (λD'. D' ⊆# D ∧ NU' + *NUE* $\models$*pm add-mset* K D'))⟩
proof −
show *?thesis*
apply (*rule order.trans*)
apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*OF*
assms(*7*−*22*)[*unfolded assms*(*1*−*8*)],
unfolded assms(*1*−*8*)[*symmetric*]])
apply (*rule order.trans*)
apply (*rule ref-two-step'*[*OF iterate-over-conflict-spec*[*OF NU-P-D dist-D*]])
by (*auto simp*: *conc-fun-RES*)
qed

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*]))⟩
by (*auto intro*!: *ext*)

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 {
$C \leftarrow$ *get-propagation-reason-pol M* ($-L$);
case C of
Some C $\Rightarrow$ *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* $\Rightarrow$ *do* {
RETURN (*cach*, [], *False*)
}
}
}›

lemma *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*[*intro*]: ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$ *atm-of* $L \in\# \mathcal{A}$›
using *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ **by** *blast*

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* $\mathcal{A}$)) $\in$
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* $\circ$ *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* $\rightarrow$
⟨*cach-refinement* $\mathcal{A} \times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*›
proof −
have [*intro*!]: ‹(*x2g*, *x'*) $\in$ *cach-refinement* $\mathcal{A} \Longrightarrow$
(*x2g*, *x'*) $\in$ *cach-refinement* (*fold-mset* (+) $\mathcal{A}$ {#})› **for** *x2g x'*
by *auto*
have [*refine0*]: ‹*get-propagation-reason-pol M* (− *L*)
$\leq \Downarrow$ (⟨*Id*⟩*option-rel*)
(*get-propagation-reason M'* (− *L'*))›
if ‹(*M*, *M'*) $\in$ *trail-pol* $\mathcal{A}$› **and** ‹(*L*, *L'*) $\in$ *Id*› **and** ‹$-L \in$ *lits-of-l M'*›
for *M M' L L'*
using *that get-propagation-reason-pol*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*, *of M'* ‹$-L'$› *M* ‹$-L$›] **by** *auto*

show *?thesis*
unfolding *isa-literal-redundant-wl-lookup-def literal-redundant-wl-lookup-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup[*of* $\mathcal{A}$ *vdom*, *THEN fref-to-Down-curry5*])
subgoal
by (*rule get-level-pol-pre*) *auto*
subgoal by (*rule conflict-min-cach-l-pre*) *auto*
subgoal
by (*auto simp*: *get-level-get-level-pol in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*])
(*subst* (*asm*) *nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*]; *auto*)+
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i
by (*subst nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp del: *conflict-min-cach-def*)
(*auto simp*: *get-level-get-level-pol in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*)

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ xb x′a*
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def*
simp: *valid-arena-nempty*
intro!: *exI*[*of - xb*])
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ xb x′a*
by (*simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why $[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *header-size* (*?N* $\propto$ *?i*) $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *?i* $<$ *length ?arena*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Size* (*?arena* ! (*?i* $-$ *SIZE-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *length* (*?N* $\propto$ *?i*) = *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *?N* $\propto$ *?i* ! *?j* = *arena-lit ?arena* (*?i* + *?j*)
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *is-Lit* (*?arena* ! (*?i* + *?j*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *?i* + *?j* $<$ *length ?arena*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *?N* $\propto$ *?i* ! *0* = *arena-lit ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Lit* (*?arena* ! *?i*)
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *?i* + *length* (*?N* $\propto$ *?i*) $\leq$ *length ?arena*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *is-Pos* (*?arena* ! (*?i* $-$ *POS-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *arena-pos ?arena ?i* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-LBD* (*?arena* ! (*?i* $-$ *LBD-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Act* (*?arena* ! (*?i* $-$ *ACTIVITY-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Status* (*?arena* ! (*?i* $-$ *STATUS-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *SIZE-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *LBD-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *ACTIVITY-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *2* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Suc 0* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *0* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Suc 0* $<$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *0* $<$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ (*arena-status ?arena ?i* = *LEARNED*) = ($\neg$ *irred ?N ?i*)
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *arena-status ?arena ?i* $\neq$ *DELETED*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* $\propto$ *?i*) requires to be instantiated.
done

qed

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*). $L \in\# C \wedge -L \notin\# C \wedge L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$
Id $\times_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*case-tac y*; *case-tac x*)
subgoal for *x y a b aa ab c*
using *mset-as-position-remove*[*of c b* ‹*atm-of aa*›]
by (*cases* ‹*aa*›)
(*auto simp*: *lookup-clause-rel-def lookup-conflict-remove1-def lookup-clause-rel-atm-in-iff*
minus-notin-trivial2 size-remove1-mset-If in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff minus-notin-trivial*
mset-as-position-in-iff-nth)
done

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 −
have *init*: ‹((*x2f*, *one-uint32-nat*, *x2g*, *x2i*), *x2a*::*nat literal multiset*, *one-uint32-nat*, *x2b*, *x2d*)
$\in$ *lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *cach-refinement* $\mathcal{A}$ $\times_r$ *Id* ›
if
‹(*x*, *y*)
$\in$ *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*› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)›
for *x y x1 x1a x1b x1c x1d x2 x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i **and**
x2a
proof −
show *?thesis*
using *that* **by** *auto*
qed

show *?thesis*
unfolding *isa-minimize-and-extract-highest-lookup-conflict-def uncurry-def*
minimize-and-extract-highest-lookup-conflict-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup[*of* $\mathcal{A}$ *vdom*, *THEN fref-to-Down-curry5*])
apply (*rule init*; *assumption*)
subgoal by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*auto simp*: *lookup-conflict-remove1-pre-def lookup-clause-rel-def atms-of-def*
minimize-and-extract-highest-lookup-conflict-inv-def)
subgoal
by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def*
intro!: *lookup-conflict-remove1* [*THEN fref-to-Down-unRET-uncurry*]
simp: *nth-in-set-tl delete-from-lookup-conflict-pre-def*
dest!: *in-set-takeD*)
subgoal by *auto*
done
qed

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
set-empty-conflict-to-none-def set-lookup-empty-conflict-to-none-def)

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 −
define *lbd-upd* **where**
⟨*lbd-upd lbd i* ≡ *lbd-write lbd* (*get-level M* (*D*!*i*))⟩ **for** *lbd i*
let *?D* = ⟨*remove1 L D*⟩
have *le-D-le-upper*[*simp*]: ⟨*a* < *length D* ⟹ *Suc* (*Suc a*) ≤ *uint-max*⟩ **for** *a*
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ⟨*mset D*⟩] *assms* **by** (*auto simp*: *uint-max-def*)
have *Suc-N-uint-max*: ⟨*Suc n* ≤ *uint-max*⟩ **and**
size-C-uint-max: ⟨*size C* ≤ *1* + *uint-max div 2*⟩ **and**
clvls: ⟨*clvls* = *card-max-lvl M C*⟩ **and**
tauto-C: ⟨¬ *tautology C*⟩ **and**
dist-C: ⟨*distinct-mset C*⟩ **and**
atms-le-xs: ⟨∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*⟩ **and**
map: ⟨*mset-as-position xs C*⟩
using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ *C*] *mset-as-position-distinct-mset*[*of xs C*]
lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*auto simp*: *uint-max-def*)
then have *clvls-uint-max*: ⟨*clvls* ≤ *1* + *uint-max div 2*⟩
using *size-filter-mset-lesseq*[*of* ⟨λ*L*. *get-level M L* = *count-decided M*⟩ *C*]
unfolding *uint-max-def card-max-lvl-def* **by** *linarith*
have [*intro*]: ⟨((*b*, *a*, *ba*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹
Suc (*Suc a*) ≤ *uint-max*⟩ **for** *b a ba C*
using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def uint-max-def)
have [*simp*]: ⟨*remdups-mset C* = *C*⟩
using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def distinct-mset-remdups-mset-id)
have ⟨¬*tautology C*⟩
using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def)
have ⟨*distinct-mset C*⟩
using *mset-as-position-distinct-mset*[*of - C*] *o*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def* **by** *auto*
have ⟨*mset* (*tl outl*) ⊆# *C*⟩
using *out* **by** (*auto simp*: *out-learned-def*)
from *size-mset-mono*[*OF this*] **have** *outl-le*: ⟨*length outl* < *uint32-max*⟩
using *simple-clss-size-upper-div2*[*OF bounded lits-C*] *dist-C tauto-C* **by** (*auto simp*: *uint32-max-def*)
define *L′* **where** ⟨*L′* ≡ *if D* ! *0* = *L then D* ! *1 else D* ! *0*⟩
have *L′-all*: ⟨*L′* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩
using *lits le2* **by** (*cases D*; *cases* ⟨*tl D*⟩)
(*auto simp*: *L′-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
then have *L′*: ⟨*atm-of L′* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
by (*auto simp*: *atms-of-def*)
have *DLL*: ⟨*mset D* = {#*L*, *L′*#}⟩ ⟨*set D* = {*L*, *L′*}⟩ ⟨*L* ≠ *L′*⟩ ⟨*remove1 L D* = [*L′*]⟩
using *le2 L-D dist* **by** (*cases D*; *cases* ⟨*tl D*⟩; *auto simp*: *L′-def*; *fail*)+
have ⟨− *L′* ∈# *C* ⟹ *False*⟩ **and** [*simp*]: ⟨− *L′* ∉# *C*⟩
using *dist no-tauto* **by** (*auto simp*: *DLL*)
then have *o′*: ⟨((*False*, *add-to-lookup-conflict L′* (*n*, *xs*)), *Some* ({#*L′*#} ∪# *C*))
∈ *option-lookup-clause-rel* $\mathcal{A}$⟩
using *o L′-all* **unfolding** *option-lookup-clause-rel-def*
by (*auto intro*!: *add-to-lookup-conflict-lookup-clause-rel*)
have [*iff*]: ⟨*is-in-lookup-conflict* (*n*, *xs*) *L′* ⟷ *L′* ∈# *C*⟩
using *o mset-as-position-in-iff-nth*[*of xs C L′*] *L′ no-tauto*
by (*auto simp*: *is-in-lookup-conflict-def option-lookup-clause-rel-def*

lookup-clause-rel-def DLL is-pos-neg-not-is-pos
split: *option.splits*)
(*metis* (*full-types*) ‹− L' ∉# C› *atm-of-uminus is-pos-neg-not-is-pos*
mset-as-position-in-iff-nth) +
have *clvls-add*: ‹*clvls-add* M L' (n, *xs*) *clvls* = *card-max-lvl* M ({#L'#} ∪# C)›
by (*cases* ‹L' ∈# C›)
(*auto simp*: *clvls-add-def card-max-lvl-add-mset clvls add-mset-union*
dest!: *multi-member-split*)
have *out*′: ‹*out-learned* M (*Some* ({#L'#} ∪# C)) (*outlearned-add* M L' (n, *xs*) *outl*)›
using *out*
by (*cases* ‹L' ∈# C›)
(*auto simp*: *out-learned-def outlearned-add-def add-mset-union*
dest!: *multi-member-split*)

show *?thesis*
unfolding *lookup-merge-eq2-def prod.simps* L'*-def*[*symmetric*]
apply *refine-vcg*
subgoal by (*rule le2*)
subgoal using *literals-are-in-*$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*OF bounded lits-tr n-d*] **by** *blast*
subgoal using *atms-le-xs* L' **by** *simp*
subgoal using *outl-le* **.**
subgoal using *clvls-uint-max* **by** (*auto simp*: *uint-max-def*)
subgoal using *Suc-N-uint-max* **by** *auto*
subgoal
using o' *clvls-add out*′
by (*auto simp*: *merge-conflict-m-eq2-def DLL*
intro!: *RETURN-RES-refine*)
done
qed

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 −
define *L′* **where** ‹*L′* ≡ (*if arena-lit arena i* = *L then arena-lit arena* (*i* + *1*)
else arena-lit arena i)›
define *L″* **where** ‹*L″* ≡ (*if N* ∝ *i* ! *0* = *L then N* ∝ *i* ! *1 else N* ∝ *i* ! *0*)›

have [*simp*]: ‹*L″* = *L′*›
if ‹*length* (*N* ∝ *i*) = *2*›
using *that i valid* **by** (*auto simp*: *L″-def L′-def arena-lifting*)
have *L′-all*: ‹*L′* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
if ‹*length* (*N* ∝ *i*) = *2*›
by (*use lits i valid that*
literals-are-in-$\mathcal{L}_{in}$*-mm-add-msetD*[*of* $\mathcal{A}$
‹*mset* (*N* ∝ *i*)› - ‹*arena-lit arena* (*Suc i*)›]
literals-are-in-$\mathcal{L}_{in}$*-mm-add-msetD*[*of* $\mathcal{A}$
‹*mset* (*N* ∝ *i*)› - ‹*arena-lit arena i*›]
nth-mem[*of 0* ‹*N* ∝ *i*›] *nth-mem*[*of 1* ‹*N* ∝ *i*›]
in ‹*auto simp*: *arena-lifting ran-m-def L′-def*
simp del: *nth-mem*
dest:
dest!: *multi-member-split*›)

show *?thesis*
unfolding *isasat-lookup-merge-eq2-def lookup-merge-eq2-def prod.simps*
L′-def[*symmetric*] *L″-def*[*symmetric*]
apply *refine-vcg*
subgoal
using *valid i*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*auto intro*!: *exI*[*of* - *i*] *exI*[*of* - *N*])
subgoal
using *valid i*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*auto intro*!: *exI*[*of* - *i*] *exI*[*of* - *N*])
subgoal
by (*rule get-level-pol-pre*[*OF* - *M′M*])
(*use L′-all*
in ‹*auto simp*: *arena-lifting ran-m-def*
simp del: *nth-mem*
dest:
dest!: *multi-member-split*›)
subgoal
by (*subst get-level-get-level-pol*[*OF M′M*, *symmetric*])
(*use L′-all* **in** *auto*)
subgoal by *auto*
subgoal
using *M′M L′-all*
by (*auto simp*: *isa-clvls-add-clvls-add get-level-get-level-pol*
isa-outlearned-add-outlearned-add)
done
qed

definition *merge-conflict-m-eq2-pre* **where**
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ =
(λ(((((((*L*, *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*)) $\wedge$
($\forall$ *K* $\in$ *set* (*remove1 L* (*N* $\propto$ *i*)). $-$ *K* $\notin\#$ *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 $\mathcal{A}$ $\wedge$
length (*N* $\propto$ *i*) = *2* $\wedge$
L $\in$ *set* (*N* $\propto$ *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* $\propto$ *i*) *D*›

lemma *isasat-lookup-merge-eq2*:
‹(*uncurry7 isasat-lookup-merge-eq2*, *uncurry7 merge-conflict-m-g-eq2*) $\in$
[*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* $\rightarrow$
$\langle$*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* $\rangle$*nres-rel*›
proof $-$
have *H1*: ‹*isasat-lookup-merge-eq2 a* (*aa*, *ab*, *ac*, *ad*, *ae*, *b*) *ba bb* (*af*, *ag*, *bc*) *bd be*
bf
$\leq \Downarrow$ *Id* (*lookup-merge-eq2 a bg* (*bh* $\propto$ *bb*) (*af*, *ag*, *bc*) *bd be bf*)›
if
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)› **and**
‹((((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*), *bd*), *be*), *bf*),
(((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)
$\in$ *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›
for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
proof $-$
have
bi: ‹*bi* $\in\#$ *dom-m bh*› **and**
‹(*bf*, *bm*) $\in$ *Id*› **and**
‹*bj* $\neq$ *None*› **and**
‹(*be*, *bl*) $\in$ *Id*› **and**
‹*distinct* (*bh* $\propto$ *bi*)› **and**
‹(*bd*, *bk*) $\in$ *nat-rel*› **and**
‹$\neg$ *tautology* (*mset* (*bh* $\propto$ *bi*))› **and**
o: ‹((*af*, *ag*, *bc*), *bj*) $\in$ *option-lookup-clause-rel* $\mathcal{A}$› **and**
‹$\forall$ *K*$\in$*set* (*remove1 ah* (*bh* $\propto$ *bi*)). $-$ *K* $\notin\#$ *the bj*› **and**
st: ‹*bb* = *bi*› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)› **and**
valid: ‹*valid-arena ba bh vdom*› **and**
‹*bk* = *card-max-lvl bg* (*the bj*)› **and**
‹(*a*, *ah*) $\in$ *Id*› **and**
tr: ‹((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) $\in$ *trail-pol* $\mathcal{A}$› **and**
‹*out-learned bg bj bm*› **and**
‹*no-dup bg*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf bh*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**

ah: ⟨*ah* ∈ *set* (*bh* ∝ *bi*)⟩
using *that* **unfolding** *merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff*
by *blast+*

show *?thesis*
by (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[*OF valid bi*[*unfolded st*[*symmetric*]] *lits o tr bounded*])
qed
have *H2*: ⟨*lookup-merge-eq2 a bg* (*bh* ∝ *bb*) (*af*, *ag*, *bc*) *bd be bf*
$\leq \Downarrow$ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_f$ (*nat-rel* $\times_f$ (*Id* $\times_f$ *Id*)))
(*merge-conflict-m-g-eq2 ah bg bh bi bj bk bl bm*)⟩
if
⟨*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)⟩ **and**
⟨((((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*), *bd*), *be*), *bf*),
((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)
∈ *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⟩
for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
proof −
have
bi: ⟨*bi* ∈# *dom-m bh*⟩ **and**
bj: ⟨*bj* ≠ *None*⟩ **and**
dist: ⟨*distinct* (*bh* ∝ *bi*)⟩ **and**
tauto: ⟨¬ *tautology* (*mset* (*bh* ∝ *bi*))⟩ **and**
o: ⟨((*af*, *ag*, *bc*), *bj*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
K: ⟨∀ *K*∈*set* (*remove1 ah* (*bh* ∝ *bi*)). − *K* ∉# *the bj*⟩ **and**
st: ⟨*bb* = *bi*⟩
⟨*bd* = *bk*⟩
⟨*bf* = *bm*⟩
⟨*be* = *bl*⟩
⟨*a* = *ah*⟩ **and**
lits-confl: ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)⟩ **and**
valid: ⟨*valid-arena ba bh vdom*⟩ **and**
bk: ⟨*bk* = *card-max-lvl bg* (*the bj*)⟩ **and**
tr: ⟨((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) ∈ *trail-pol* $\mathcal{A}$⟩ **and**
out: ⟨*out-learned bg bj bm*⟩ **and**
⟨*no-dup bg*⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf bh*)⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩ **and**
le2: ⟨*length* (*bh* ∝ *bi*) = *2*⟩ **and**
ah: ⟨*ah* ∈ *set* (*bh* ∝ *bi*)⟩
using *that* **unfolding** *merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff*
by *blast+*
obtain *bj'* **where** *bj'*: ⟨*bj* = *Some bj'*⟩
using *bj* **by** (*cases bj*) *auto*
have *n-d*: ⟨*no-dup bg*⟩ **and** *lits-tr*: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *bg*⟩
using *tr* **unfolding** *trail-pol-alt-def*
by *auto*
have *lits-bi*: ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*bh* ∝ *bi*))⟩
using *bi lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset ran-m-def*
dest!: *multi-member-split*)

show *?thesis*
unfolding *st merge-conflict-m-g-eq2-def*

apply (*rule lookup-merge-eq2-spec*[*THEN order-trans*, *OF o*[*unfolded bj'*]
dist lits-bi lits-tr n-d tauto lits-confl[*unfolded bj' option.sel*]
- *bk*[*unfolded bj' option.sel*] - *bounded le2 ah*])
subgoal using *K* **unfolding** *bj'* **by** *auto*
subgoal using *out* **unfolding** *bj'* **.**
subgoal unfolding *bj'* **by** *auto*
done
qed

show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
apply (*intro nres-relI frefI*)
apply *clarify*
subgoal for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
apply (*rule H1*[*THEN order-trans*]; *assumption?*)
apply (*subst Down-id-eq*)
apply (*rule H2*)
apply *assumption+*
done
done
qed

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 = (λlbd (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 = (λlvl (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 *intro-classes*
let *?to-pair* = ‹λ*x*::(′*a*, ′*b*) *vmtf-node*. (*stamp x*, *get-prev x*, *get-next x*)›
have *inj′*: ‹*inj ?to-pair*›
unfolding *inj-def* **by** (*intro allI*) (*case-tac x*; *case-tac y*; *auto*)
obtain *to-nat* :: ‹′*b* × ′*a option* × ′*a option* ⇒ *nat*› **where**
‹*inj to-nat*›
by *blast*
then have ‹*inj* (*to-nat o ?to-pair*)›
using *inj′* **by** (*blast intro*: *inj-compose*)
then show ‹∃ *to-nat* :: (′*a*, ′*b*) *vmtf-node* ⇒ *nat*. *inj to-nat*›
by *blast*
qed

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)›
by (*auto simp*: *watched-by-app-heur-def intro*!: *ext*)

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*) ←
$WHILE_T{}^{\lambda(i,W'). \ i \leq length\ xs \wedge W'!j = W!j\ @\ map\ R\ (take\ i\ xs) \wedge \qquad (\forall k.\ k \neq j \longrightarrow k < length\ W \longrightarrow W'!k = W}$
(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*])›
unfolding *op-map-to-def Let-def*
apply (*refine-vcg WHILEIT-rule*[**where** *R*=‹*measure* (λ(*i*,-). *length xs* − *i*)›])
apply (*auto simp*: *hd-conv-nth take-Suc-conv-app-nth list-update-append*
append-ll-def split: *nat.splits*)
by (*simp add*: *list-eq-iff-nth-eq*)

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*]))) ∈
[λ((*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*›
by (*intro frefI nres-relI*) (*auto simp*: *op-map-to-map*)

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-single-wl-to-nat*,
uncurry3 (*RETURN oooo convert-single-wl-to-nat-conv*)) ∈

$[\lambda(((xs, i), ys), j).\ i < length\ xs \wedge j < length\ ys \wedge ys!j = []]_f$
$\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \times_f$
$\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_f nat\text{-}rel \rightarrow$
$\langle\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel\rangle nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *convert-single-wl-to-nat-def convert-single-wl-to-nat-conv-def nat-of-uint64-conv-def*
dest: *op-map-to-map*[*of - - id*])

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* ⇒ (*nat literal* ⇒ (*nat* × -) *list*) ⇒ (*nat*, ′*b*) *fmap* ⇒ *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*]:
‹$bh \in\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (N(bh \hookrightarrow C)) = vdom\text{-}m\ \mathcal{A}\ W\ N$›
‹$bh \notin\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (N(bh \hookrightarrow C)) = insert\ bh\ (vdom\text{-}m\ \mathcal{A}\ W\ N)$›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps2*[*simp*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (W(L := W\ L\ @\ [(i, C)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
‹$bi \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L{:=}\ bp\ L\ @\ [(bi, av')]))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps3*[*simp*]:
‹$fst\ biav' \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L{:=}\ bp\ L\ @\ [biav']))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
by (*cases biav*′; *auto simp*: *dest*: *multi-member-split*[*of L*] *split*: *if-splits*)

What is the difference with the next lemma?

lemma [*simp*]:
‹$bf \in\#\ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ bj\ (ax(bf \hookrightarrow C')) = vdom\text{-}m\ \mathcal{A}\ bj\ (ax)$›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps4*[*simp*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1 := W\ L1\ @\ [(i, C1)],\ L2 := W\ L2\ @\ [(i, C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

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*]:
‹$i \in\#\ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1 := W\ L1\ @\ [(i, C1), (i, C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

We add a spurious dependency to the parameter of the locale:

definition *empty-watched* :: ‹*nat multiset* ⇒ *nat literal* ⇒ (*nat* × *nat literal* × *bool*) *list*› **where**
‹$empty\text{-}watched\ \mathcal{A} = (\lambda\text{-}.\ [])$›

lemma *vdom-m-empty-watched*[*simp*]:
‹$vdom\text{-}m\ \mathcal{A}\ (empty\text{-}watched\ \mathcal{A}')\ N = set\text{-}mset\ (dom\text{-}m\ N)$›
by (*auto simp*: *vdom-m-def empty-watched-def*)

The following rule makes the previous not applicable. Therefore, we do not mark this lemma as simp.

lemma *vdom-m-simps5*:
‹$i \notin\# \ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (fmupd\ i\ C\ N) = insert\ i\ (vdom\text{-}m\ \mathcal{A}\ W\ N)$›
by (*force simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

lemma *in-watch-list-in-vdom*:
assumes ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$w < length\ (watched\text{-}by\ S\ L)$›
shows ‹$fst\ (watched\text{-}by\ S\ L\ !\ w) \in vdom\text{-}m\ \mathcal{A}\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S)$›
using *assms*
unfolding *vdom-m-def*
by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-watch-list-in-vdom'*:
assumes ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$A \in set\ (watched\text{-}by\ S\ L)$›
shows ‹$fst\ A \in vdom\text{-}m\ \mathcal{A}\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S)$›
using *assms*
unfolding *vdom-m-def*
by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-dom-in-vdom*[*simp*]:
‹$x \in\# \ dom\text{-}m\ N \Longrightarrow x \in vdom\text{-}m\ \mathcal{A}\ W\ N$›
unfolding *vdom-m-def*
by (*auto dest*: *multi-member-split*)

lemma *in-vdom-m-upd*:
‹$x1f \in vdom\text{-}m\ \mathcal{A}\ (g(x1e := (g\ x1e)[x2 := (x1f,\ x2f)]))\ b$›
if ‹$x2 < length\ (g\ x1e)$› **and** ‹$x1e \in\# \mathcal{L}_{all}\ \mathcal{A}$›
using *that*
unfolding *vdom-m-def*
by (*auto dest*!: *multi-member-split intro*!: *set-update-memI img-fst*)

lemma *in-vdom-m-fmdropD*:
‹$x \in vdom\text{-}m\ \mathcal{A}\ ga\ (fmdrop\ C\ baa) \Longrightarrow x \in (vdom\text{-}m\ \mathcal{A}\ ga\ baa)$›
unfolding *vdom-m-def*
by (*auto dest*: *in-diffD*)

definition *cach-refinement-empty* **where**
‹$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \longleftrightarrow$
$(cach,\ \lambda\text{-}.\ SEEN\text{-}UNKNOWN) \in cach\text{-}refinement\ \mathcal{A}$›

definition *isa-vmtf* **where**
‹$isa\text{-}vmtf\ \mathcal{A}\ M =$
$((Id \times_r nat\text{-}rel \times_r nat\text{-}rel \times_r nat\text{-}rel \times_r \langle nat\text{-}rel\rangle option\text{-}rel) \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A})^{-1}$
$``\ vmtf\ \mathcal{A}\ M$›

lemma *isa-vmtfI*:
‹$(vm,\ to\text{-}remove') \in vmtf\ \mathcal{A}\ M \Longrightarrow (to\text{-}remove,\ to\text{-}remove') \in distinct\text{-}atoms\text{-}rel\ \mathcal{A} \Longrightarrow$
$(vm,\ to\text{-}remove) \in isa\text{-}vmtf\ \mathcal{A}\ M$›
by (*auto simp*: *isa-vmtf-def Image-iff intro*!: *bexI*[*of* - ‹(*vm*, *to-remove'*)›])

lemma *isa-vmtf-consD*:
‹$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ remove) \in isa\text{-}vmtf\ \mathcal{A}\ M \Longrightarrow$
$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ remove) \in isa\text{-}vmtf\ \mathcal{A}\ (L\ \#\ M)$›
by (*auto simp*: *isa-vmtf-def dest*: *vmtf-consD*)

lemma *isa-vmtf-consD2*:

$\langle f \in \textit{isa-vmtf}\ \mathcal{A}\ M \Longrightarrow$
$f \in \textit{isa-vmtf}\ \mathcal{A}\ (L\ \#\ M)\rangle$
by (*auto simp*: *isa-vmtf-def dest*: *vmtf-consD*)

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′)))›
using *assms* **unfolding** *twl-st-heur-def* **by** (*auto simp*: *map-fun-rel-def all-atms-def*)

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*),
$(M, N, D, NE, UE, Q, W))$.
$(M', M) \in$ *trail-pol* (*all-atms* N $(NE + UE)$) $\wedge$
valid-arena N' N (*set vdom*) $\wedge$
$(D', D) \in$ *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 $(NE + UE)$) φ $\wedge$
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*›
by (*auto simp*: *twl-st-heur-def twl-st-heur-conflict-ana-def*)

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›
using *assms* **unfolding** *twl-st-heur-conflict-ana-def* **by** (*auto simp*: *map-fun-rel-def all-atms-def*)

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*, φ, *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 $(NE + UE)$) φ $\wedge$
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*›
by (*cases S*) (*auto simp*: *isasat-fast-def*)

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*›
apply (*intro WB-More-Refinement.frefI nres-relI*)
apply (*rename-tac x y*, *case-tac x*, *case-tac y*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def option-lookup-clause-rel-def*
split: *option.splits*)

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*)›
unfolding *get-conflict-wl-is-None-heur-def*
by *auto*

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*›
by (*intro WB-More-Refinement.frefI nres-relI*)
(*auto simp*: *count-decided-st-def twl-st-heur-def isa-count-decided-st-def*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

lemma *count-decided-st-alt-def*: ‹*count-decided-st S* = *count-decided* (*get-trail-wl S*)›
unfolding *count-decided-st-def*
by (*cases S*) *auto*

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* = (λL (M, N, (-, (-, D)), -). $RETURN$ (D ! (*atm-of* L) $\neq$ *None*))⟩
unfolding *atm-is-in-conflict-st-heur-def* **by** (*auto intro*!: *ext simp*: *atm-in-conflict-lookup-def*)

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*)) $\in$
[$\lambda(L, S)$. $-L \notin\#$ *the* (*get-conflict-wl* S) $\wedge$ *get-conflict-wl* $S \neq$ *None* $\wedge$
$L \in\# \mathcal{L}_{all}$ (*all-atms-st* S)]$_f$
Id $\times_r$ *twl-st-heur* $\rightarrow$ ⟨*Id*⟩ *nres-rel*⟩
proof –
have *1*: ⟨*aaa* $\in\# \mathcal{L}_{all}$ $A \Longrightarrow$ *atm-of aaa* $\in$ *atms-of* ($\mathcal{L}_{all}$ A)⟩ **for** *aaa A*
by (*auto simp*: *atms-of-def*)
show *?thesis*
unfolding *atm-is-in-conflict-st-heur-def twl-st-heur-def option-lookup-clause-rel-def*
apply (*intro frefI nres-relI*)
apply (*case-tac x*, *case-tac y*)
apply *clarsimp*
apply (*subst atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*])
unfolding *prod.simps prod-rel-iff*
apply (*rule 1*; *assumption*)
apply (*auto simp*: *all-atms-def*; *fail*)
by (*auto simp*: *is-in-conflict-st-def atm-in-conflict-def atms-of-def atm-of-eq-atm-of*)
qed

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*)) $\in$
[$\lambda(L, S)$. $-L \notin\#$ *the* (*get-conflict-wl* S) $\wedge$ *get-conflict-wl* $S \neq$ *None* $\wedge$
$L \in\# \mathcal{L}_{all}$ (*all-atms-st* S)]$_f$
Id $\times_r$ *twl-st-heur-conflict-ana* $\rightarrow$ ⟨*Id*⟩ *nres-rel*⟩
proof –
have *1*: ⟨*aaa* $\in\# \mathcal{L}_{all}$ $A \Longrightarrow$ *atm-of aaa* $\in$ *atms-of* ($\mathcal{L}_{all}$ A)⟩ **for** *aaa A*
by (*auto simp*: *atms-of-def*)
show *?thesis*
unfolding *atm-is-in-conflict-st-heur-def twl-st-heur-conflict-ana-def option-lookup-clause-rel-def*
apply (*intro frefI nres-relI*)
apply (*case-tac x*, *case-tac y*)
apply *clarsimp*
apply (*subst atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*])
unfolding *prod.simps prod-rel-iff*
apply (*rule 1*; *assumption*)
apply (*auto simp*: *all-atms-def*; *fail*)
by (*auto simp*: *is-in-conflict-st-def atm-in-conflict-def atms-of-def atm-of-eq-atm-of*)
qed

definition *polarity-st-heur*
:: ⟨*twl-st-wl-heur* $\Rightarrow$ *nat literal* $\Rightarrow$ *bool option*⟩
where
⟨*polarity-st-heur* S =
polarity-pol (*get-trail-wl-heur* S)⟩

definition *polarity-st-pre* **where**
⟨*polarity-st-pre* $\equiv \lambda(S, L)$. $L \in\# \mathcal{L}_{all}$ (*all-atms-st* S)⟩

lemma *polarity-st-heur-alt-def*:

‹*polarity-st-heur* = (λ(*M*, -). *polarity-pol M*)›
by (*auto simp*: *polarity-st-heur-def*)

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*:
‹$(S', S) \in$ *twl-st-heur* $\Longrightarrow L \in\# \mathcal{L}_{all}$ (*all-atms-st S*) $\Longrightarrow$ *polarity-st-heur-pre* (S', L)›
by (*auto simp*: *twl-st-heur-def polarity-st-heur-pre-def all-atms-def*[*symmetric*]
intro!: *polarity-st-heur-pre-def polarity-pol-pre*)

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*›
using *arena-lifting*[*of arena NU vdom DECISION-REASON*]
by (*auto simp*: *DECISION-REASON-def SHIFTS-def*)

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*)›
unfolding *count-decided-st-def twl-st-heur-def trail-pol-def*
by (*cases S*) (*auto simp*: *count-decided-st-heur-def*)

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*)›
unfolding *isa-length-trail-def twl-st-heur-def trail-pol-def*
by (*cases S*) (*auto dest*: *ann-lits-split-reasons-map-lit-of*)

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈ *trail-pol* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf*-$\mathcal{L}_{all}$-*def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
atoms-hash-rel-def
by (*auto simp*:)

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf*-$\mathcal{L}_{all}$-*def*
by *auto*

lemma *isa-vmtf-cong*:

⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *isa-vmtf* $\mathcal{A}$ M $\Longrightarrow$ $L \in$ *isa-vmtf* $\mathcal{B}$ M⟩
using *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
apply (*subst* (*asm*) *isa-vmtf-def*)
apply (*cases* L)
by (*auto intro*!: *isa-vmtfI*)

lemma *option-lookup-clause-rel-cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *option-lookup-clause-rel* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *option-lookup-clause-rel* $\mathcal{B}$⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
apply (*cases* L)
by (*auto intro*!: *isa-vmtfI*)

lemma D_0-*cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ D_0 $\mathcal{A}$ = D_0 $\mathcal{B}$⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by *auto*

lemma *phase-saving-cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *phase-saving* $\mathcal{A}$ = *phase-saving* $\mathcal{B}$⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-saving-def*)

lemma *distinct-subseteq-iff2*:
assumes *dist*: *distinct-mset* M
shows *set-mset* $M \subseteq$ *set-mset* $N \longleftrightarrow M \subseteq\# N$
proof
assume *set-mset* $M \subseteq$ *set-mset* N
then show $M \subseteq\# N$
using *dist* **by** (*metis distinct-mset-set-mset-ident mset-set-subset-iff*)
next
assume $M \subseteq\# N$
then show *set-mset* $M \subseteq$ *set-mset* N
by (*metis set-mset-mono*)
qed

lemma *cach-refinement-empty-cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *cach-refinement-empty* $\mathcal{A}$ = *cach-refinement-empty* $\mathcal{B}$⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*force simp*: *cach-refinement-empty-def cach-refinement-alt-def*
distinct-subseteq-iff2[*symmetric*] *intro*!: *ext*)

lemma *vdom-m-cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *vdom-m* $\mathcal{A}$ x y = *vdom-m* $\mathcal{B}$ x y⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *vdom-m-def intro*!: *ext*)

lemma *isasat-input-bounded-cong*:
⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ = *isasat-input-bounded* $\mathcal{B}$⟩
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *intro*!: *ext*)

lemma *isasat-input-nempty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *isasat-input-nempty* $\mathcal{A}$ = *isasat-input-nempty* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *intro*!: *ext*)

0.1.11 Shared Code Equations

definition *clause-not-marked-to-delete* **where**
‹*clause-not-marked-to-delete S C* $\longleftrightarrow$ *C* ∈# *dom-m* (*get-clauses-wl S*)›

definition *clause-not-marked-to-delete-pre* **where**
‹*clause-not-marked-to-delete-pre* =
(λ(*S*, *C*). *C* ∈ *vdom-m* (*all-atms-st S*) (*get-watched-wl S*) (*get-clauses-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* $\longleftrightarrow$
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*›
by (*intro WB-More-Refinement.frefI nres-relI*)
(*use arena-dom-status-iff in-dom-in-vdom* **in**
‹*auto 5 5 simp*: *clause-not-marked-to-delete-def twl-st-heur-def*
clause-not-marked-to-delete-heur-def arena-dom-status-iff all-atms-def[*symmetric*]
clause-not-marked-to-delete-pre-def›)

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*))›
by (*auto simp*: *access-lit-in-clauses-heur-def intro*!: *ext*)

lemma *access-lit-in-clauses-heur-fast-pre*:
‹*arena-lit-pre* (*get-clauses-wl-heur a*) (*ba* + *b*) $\Longrightarrow$
isasat-fast a $\Longrightarrow$ *ba* + *b* ≤ *uint64-max*›
by (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
dest!: *arena-lifting*(*10*)
dest!: *isasat-fast-length-leD*)[]

lemma *eq-insertD*: ‹*A* = *insert a B* $\Longrightarrow$ *a* ∈ *A* ∧ *B* ⊆ *A*›
by *auto*

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*)))›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *correct-watching-dom-watched*:
assumes ‹*correct-watching S*› **and** ‹$\bigwedge C$. *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
fix *C*
assume ‹*C* ∈ *?A*›
then obtain *D* **where**
D: ‹*D* ∈# *ran-mf* (*get-clauses-wl S*)› **and**
D′: ‹*D* = *get-clauses-wl S* ∝ *C*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›
by *auto*
have ‹*atm-of* (*hd D*) ∈# *atm-of* ‘# *all-lits-st S*›
using *D D′ assms*(*2*)[*of D*]
by (*cases S*; *cases D*)
(*auto simp*: *all-lits-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
dest!: *multi-member-split*)
then show ‹*C* ∈ *?B*›
using *assms*(*1*) *assms*(*2*)[*of D*] *D D′*
multi-member-split[*OF C*]
by (*cases S*; *cases* ‹*get-clauses-wl S* ∝ *C*›;
cases ‹*hd* (*get-clauses-wl S* ∝ *C*)›)
(*auto simp*: *correct-watching.simps clause-to-update-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
$\mathcal{L}_{all}$*-add-mset*
eq-commute[*of* ‹- # -›] *atm-of-eq-atm-of*
dest!: *multi-member-split eq-insertD*
dest!: *arg-cong*[*of* ‹*filter-mset* - -› ‹*add-mset* - -› *set-mset*])
qed

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*)
(λ*i W*. *do* {
ASSERT(*i* < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C ≠ *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* ≠ *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* ⊆ *vdom*› **and** ‹*distinct xs*› **and** ‹*set-mset* (*dom-m N*) ⊆ *set xs*› **and**
‹(*W*, *W'*) ∈ ⟨*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* ⊆ *set-mset* (*dom-m N*)›
shows
‹*rewatch-heur xs arena W* ≤ ⇓ ({(*W*, *W'*). (*W*, *W'*) ∈⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W' N* ⊆ *set-mset* (*dom-m N*)}) (*rewatch N W'*)›
proof −
have [*refine0*]: ‹(*xs*, *xsa*) ∈ *Id* ⟹
([*0*..<*length xs*], [*0*..<*length xsa*]) ∈ ⟨{(*x*, *x'*). *x* = *x'* ∧ *x* < *length xsa* ∧ *xs*!*x* ∈ *vdom*}⟩*list-rel*›
for *xsa*
using *assms* **unfolding** *list-rel-def*
by (*auto simp*: *list-all2-same*)
show *?thesis*
unfolding *rewatch-heur-def rewatch-def*
apply (*subst* (*2*) *nfoldli-nfoldli-list-nth*)
apply (*refine-vcg*)
subgoal
using *assms* **by** *fast*
subgoal
using *assms* **by** *fast*
subgoal
using *assms* **by** *fast*
subgoal by *fast*
subgoal by *auto*
subgoal
using *assms*
unfolding *arena-is-valid-clause-vdom-def*
by *blast*
subgoal
using *assms*
by (*auto simp*: *arena-dom-status-iff*)
subgoal for *xsa xi x si s*
using *assms*
unfolding *arena-lit-pre-def*
by (*rule-tac j*=‹*xs* ! *xi*› **in** *bex-leI*)
(*auto simp*: *arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *N*] *exI*[*of* - *vdom*])
subgoal for *xsa xi x si s*

using *assms*
unfolding *arena-lit-pre-def*
by (*rule-tac j=‹xs ! xi›* **in** *bex-leI*)
(*auto simp*: *arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of - N*] *exI*[*of - vdom*])
subgoal for *xsa xi x si s*
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of ‹xs ! xi› 0*] *assms*
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
subgoal for *xsa xi x si s*
using *assms*
unfolding *arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def*
by (*auto simp*: *arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of - N*] *exI*[*of - vdom*])
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal for *xsa xi x si s* **using** *valid-arena-size-dom-m-le-arena*[*OF assms*(*1*)]
literals-are-in-$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of ‹xs ! xi› 0*] *assms* **by** (*auto simp*: *map-fun-rel-def arena-lifting*)
subgoal for *xsa xi x si s*
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of ‹xs ! xi› 1*] *assms*
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
subgoal for *xsa xi x si s* **using** *valid-arena-size-dom-m-le-arena*[*OF assms*(*1*)]
literals-are-in-$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of ‹xs ! xi› 1*] *assms*
by (*auto simp*: *map-fun-rel-def arena-lifting append-ll-def*)
subgoal for *xsa xi x si s*
using *assms*
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
done
qed

lemma *rewatch-heur-alt-def*:
‹*rewatch-heur vdom arena W* = *do* {
let - = *vdom*;
nfoldli [*0..<length vdom*] (λ-. *True*)
(λ*i W*. *do* {
ASSERT(*i* < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C ≠ *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* ≠ *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
}⟩
unfolding *Let-def uint64-of-nat-conv-def rewatch-heur-def*
by *auto*

lemma *arena-lit-pre-le-uint64-max*:
⟨*length ba* ≤ *uint64-max* ⟹
arena-lit-pre ba a ⟹ *a* ≤ *uint64-max*⟩
using *arena-lifting*(*10*)[*of ba - -*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def)

definition *rewatch-heur-st*
:: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩
where
⟨*rewatch-heur-st* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, *t*, *vdom*, *avdom*, *ccount*, *lcount*). *do* {
ASSERT(*length vdom* ≤ *length N0*);
W ← *rewatch-heur vdom N0 W*;
RETURN (*M*, *N0*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
stats, *fema*, *sema*, *t*, *vdom*, *avdom*, *ccount*, *lcount*)
})⟩

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 S*). *x* ≤ *uint64-max*) ∧ *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*)⟩

definition *rewatch-st* :: ⟨'*v twl-st-wl* ⇒ '*v twl-st-wl nres*⟩ **where**
⟨*rewatch-st S* = *do*{
(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) ← *RETURN S*;
W ← *rewatch N W*;
RETURN ((*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*))
}⟩

fun *remove-watched-wl* :: ⟨'*v twl-st-wl* ⇒ -⟩ **where**
⟨*remove-watched-wl* (*M*, *N*, *D*, *NE*, *UE*, *Q*, -) = (*M*, *N*, *D*, *NE*, *UE*, *Q*)⟩

lemma *rewatch-st-correctness*:
assumes ⟨*get-watched-wl S* = (λ-. [])⟩ **and**
⟨⋀*x*. *x* ∈# *dom-m* (*get-clauses-wl S*) ⟹
distinct ((*get-clauses-wl S*) ∝ *x*) ∧ *2* ≤ *length* ((*get-clauses-wl S*) ∝ *x*)⟩
shows ⟨*rewatch-st S* ≤ *SPEC* (λ*T*. *remove-watched-wl S* = *remove-watched-wl T* ∧
correct-watching-init T)⟩
apply (*rule SPEC-rule-conjI*)
subgoal
using *rewatch-correctness*[*OF assms*]
unfolding *rewatch-st-def*
apply (*cases S*, *case-tac* ⟨*rewatch b g*⟩)
by (*auto simp*: *RES-RETURN-RES*)
subgoal
using *rewatch-correctness*[*OF assms*]

unfolding *rewatch-st-def*
apply (*cases S*, *case-tac* ‹*rewatch b g*›)
by (*force simp*: *RES-RETURN-RES*)+
done

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*›
by (*cases S*)
(*auto simp*: *isasat-fast-slow-def trail-slow-of-fast-def convert-wlists-to-nat-conv-def*
trail-pol-slow-of-fast-alt-def)

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*›
by (*intro nres-relI WB-More-Refinement.frefI*)
(*auto simp*: *isasat-fast-slow-alt-def isasat-fast-slow-wl-D-def*)

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 −
have ‹*correct-watching x1*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-D-inv-def*

unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x'a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*) (*get-unit-clauses-wl x1*))›
using *x2 xb-x'a* **unfolding** *all-atms-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur'-def twl-st-heur-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def all-atms-def*[*symmetric*])
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur'-def twl-st-heur-def simp flip*: *all-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *4*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show *?thesis*
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

lemma *length-watched-le2*:
assumes
prop-inv: ‹*correct-watching-except i j L x1*› **and**
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur'' 𝒟1 r*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)› **and** *diff*: ‹*L* ≠ *x2*›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *4*›
proof −
from *prop-inv diff* **have** *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching-except.simps*)

then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
  using *xb-x'a*
  by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
  using *xb-x'a*
  by (*cases x1*)
    (*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*) (*get-unit-clauses-wl x1*))›
  using *x2 xb-x'a*
  by (*auto simp flip*: *all-atms-def*)

have
    valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
  using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
  by (*cases x1*)
   (*auto simp*: *twl-st-heur'-def twl-st-heur-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
  using *xb-x'a*
  by (*cases x1*)
    (*auto simp*: *twl-st-heur-def twl-st-heur'-def simp flip*: *all-atms-def*)
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
  using *x2* **unfolding** *vdom-m-def*
  by (*cases x1*)
    (*force simp*: *twl-st-heur'-def twl-st-heur-def simp flip*: *all-atms-def*
      dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
  by (*rule distinct-subseteq-iff*[*THEN iffD1*])
    (*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
      ‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}›
  using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *4*›
  by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
    (*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show *?thesis*
  using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
  by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

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* ›
  by (*induction C*; *auto simp*: *all-lits-of-m-add-mset*)+
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*›
  by *sepref-to-hoare* (*sep-auto simp*: *tri-bool-assn-def tri-bool-ref-def pure-def hr-comp-def*)

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*›
apply *sepref-to-hoare*
using *nat-of-uint32-012 nat-of-uint32-3*
by (*sep-auto simp*: *tri-bool-assn-def tri-bool-ref-def pure-def hr-comp-def*
tri-bool-eq-def)+

lemma *SET-TRUE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return SET-TRUE-code*), *uncurry0* (*RETURN SET-TRUE*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *tri-bool-assn-def tri-bool-ref-def pure-def hr-comp-def*)

lemma *SET-FALSE-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return SET-FALSE-code*), *uncurry0* (*RETURN SET-FALSE*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
using *nat-of-uint32-012 nat-of-uint32-3*
by *sepref-to-hoare* (*sep-auto simp*: *tri-bool-assn-def tri-bool-ref-def pure-def hr-comp-def*)+

lemma [*safe-constraint-rules*]:
‹*is-pure tri-bool-assn*›
unfolding *tri-bool-assn-def*
by *auto*

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*›
by *sepref-to-hoare* (*sep-auto simp*: *DECISION-REASON-def uint64-nat-rel-def br-def*)

lemma *DECISION-REASON'*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 1*), *uncurry0* (*RETURN DECISION-REASON*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *DECISION-REASON-def uint64-nat-rel-def br-def*)

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*›
:: ‹$trail\text{-}pol\text{-}fast\text{-}assn^d \rightarrow_a trail\text{-}pol\text{-}assn$›
unfolding *trail-pol-slow-of-fast-def*
apply (*rewrite* **in** ‹(⌑, -, -, -)› *arl32-to-arl-conv-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, -, -, *array-nat-of-uint64-conv* -, -, ⌑)› *arl32-to-arl-conv-def*[*symmetric*])
by *sepref*

lemma *count-decided-trail*[*sepref-fr-rules*]:
‹(*return o count-decided-pol*, *RETURN o count-decided-pol*) ∈ $trail\text{-}pol\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
supply [[*goals-limit* = *1*]]
by *sepref-to-hoare* (*sep-auto simp*: *count-decided-pol-def*)

lemma *count-decided-trail-fast*[*sepref-fr-rules*]:
‹(*return o count-decided-pol*, *RETURN o count-decided-pol*) ∈ $trail\text{-}pol\text{-}fast\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
supply [[*goals-limit* = *1*]]
by *sepref-to-hoare* (*sep-auto simp*: *count-decided-pol-def*)

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*)›
:: ‹[*get-level-atm-pol-pre*]$_a$
$trail\text{-}pol\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
unfolding *get-level-atm-pol-def nat-shiftr-div2*[*symmetric*] *nat-of-uint32-shiftr*[*symmetric*]
get-level-atm-pol-pre-def nth-u-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

declare *get-level-atm-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-atm-fast-code*
is ‹*uncurry* (*RETURN oo get-level-atm-pol*)›
:: ‹[*get-level-atm-pol-pre*]$_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
unfolding *get-level-atm-pol-def nat-shiftr-div2*[*symmetric*] *nat-of-uint32-shiftr*[*symmetric*]
nth-u-def[*symmetric*] *get-level-atm-pol-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

declare *get-level-atm-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-code*
is ‹*uncurry* (*RETURN oo get-level-pol*)›
:: ‹[*get-level-pol-pre*]$_a$
$trail\text{-}pol\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
unfolding *get-level-get-level-atm nat-shiftr-div2*[*symmetric*] *nat-of-uint32-shiftr*[*symmetric*]
nth-u-def[*symmetric*] *get-level-pol-pre-def get-level-pol-def*
supply [[*goals-limit* = *1*]] *image-image*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
get-level-atm-pol-pre-def[*simp*]
by *sepref*

declare *get-level-code.refine*[*sepref-fr-rules*]

sepref-definition *get-level-fast-code*
is ⟨*uncurry* (*RETURN oo get-level-pol*)⟩
:: ⟨[*get-level-pol-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *unat-lit-assn*^{k} → *uint32-nat-assn*⟩
unfolding *get-level-get-level-atm nat-shiftr-div2*[*symmetric*] *nat-of-uint32-shiftr*[*symmetric*]
nth-u-def[*symmetric*] *get-level-pol-pre-def get-level-pol-def*
supply [[*goals-limit* = *1*]] *image-image*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
get-level-atm-pol-pre-def[*simp*]
by *sepref*

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*⟩
unfolding *polarity-pol-def option.case-eq-if polarity-pol-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
unfolding *polarity-pol-def option.case-eq-if polarity-pol-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
unfolding *isa-length-trail-def isa-length-trail-pre-def*
by *sepref*

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*⟩
unfolding *isa-length-trail-def isa-length-trail-pre-def length-uint32-nat-def*
by *sepref*

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*⟩
unfolding *cons-trail-Propagated-tr-def cons-trail-Propagated-tr-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Propagated-tr-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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} $\rightarrow$ *trail-pol-fast-assn*⟩
unfolding *cons-trail-Propagated-tr-def cons-trail-Propagated-tr-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Propagated-tr-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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 $\rightarrow$ *unat-lit-assn* $*a$ *option-assn nat-assn*⟩
unfolding *last-trail-pol-def nth-u-def*[*symmetric*] *last-trail-pol-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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 $\rightarrow$ *unat-lit-assn* $*a$ *option-assn uint64-nat-assn*⟩
supply *DECISION-REASON-uint64*[*sepref-fr-rules*]
unfolding *last-trail-pol-def nth-u-def*[*symmetric*] *zero-uint64-nat-def*[*symmetric*]
last-trail-pol-pre-def
supply [[*goals-limit* = *1*]]
by *sepref*

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 $\rightarrow$ *trail-pol-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-def UNSET-def*[*symmetric*] *butlast-nonresizing-def*[*symmetric*]
tl-trailt-tr-pre-def
apply (*rewrite at* ⟨- − *one-uint32-nat*⟩ *fast-minus-def*[*symmetric*])
supply [[*goals-limit* = *1*]]
by *sepref*

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 $\rightarrow$ *trail-pol-fast-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*] *DECISION-REASON-uint64*[*sepref-fr-rules*]
unfolding *tl-trailt-tr-def UNSET-def*[*symmetric*] *zero-uint64-nat-def*[*symmetric*]
butlast-nonresizing-def[*symmetric*] *tl-trailt-tr-pre-def*
apply (*rewrite at* ⟨- − *one-uint32-nat*⟩ *fast-minus-def*[*symmetric*])
supply [[*goals-limit* = *1*]]

by *sepref*

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*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trail-propedt-tr-def UNSET-def*[*symmetric*]
butlast-nonresizing-def[*symmetric*] *tl-trail-propedt-tr-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trail-propedt-tr-def UNSET-def*[*symmetric*]
butlast-nonresizing-def[*symmetric*] *tl-trail-propedt-tr-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
unfolding *lit-of-last-trail-pol-def*
by *sepref*

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*⟩
unfolding *lit-of-last-trail-pol-def*
by *sepref*

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} → *trail-pol-assn*⟩
unfolding *cons-trail-Decided-tr-def cons-trail-Decided-tr-def one-uint32-nat-def*[*symmetric*]
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Decided-tr-pre-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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} → *trail-pol-fast-assn*⟩
unfolding *cons-trail-Decided-tr-def cons-trail-Decided-tr-def one-uint32-nat-def*[*symmetric*]
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Decided-tr-pre-def*
zero-uint64-nat-def[*symmetric*] *nat-of-uint32-spec-def*
supply [[*goals-limit* = *1*]] *DECISION-REASON-uint64*[*sepref-fr-rules*]
by *sepref*

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} → *bool-assn*⟩
unfolding *defined-atm-pol-def UNSET-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
defined-atm-pol-pre-def
supply *UNSET-def*[*simp del*] *uint32-nat-assn-mult*[*sepref-fr-rules*]
by *sepref*

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} → *bool-assn*⟩
unfolding *defined-atm-pol-def UNSET-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
defined-atm-pol-pre-def
supply *UNSET-def*[*simp del*] *uint32-nat-assn-mult*[*sepref-fr-rules*]
by *sepref*

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*⟩
unfolding *get-propagation-reason-pol-def*
by *sepref*

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*⟩
supply *DECISION-REASON-uint64*[*sepref-fr-rules*]
unfolding *get-propagation-reason-pol-def*
zero-uint64-nat-def[*symmetric*]
by *sepref*

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*⟩
unfolding *get-the-propagation-reason-pol-def*
tri-bool-eq-def[*symmetric*]

by *sepref*

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*⟩
supply *DECISION-REASON-uint64*[*sepref-fr-rules*]
unfolding *get-the-propagation-reason-pol-def*
tri-bool-eq-def[*symmetric*]
by *sepref*

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*⟩
unfolding *isa-trail-nth-def*
by *sepref*

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*⟩
unfolding *isa-trail-nth-def*
by *sepref*

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 → *trail-pol-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-no-CS-def UNSET-def*[*symmetric*] *tl-trailt-tr-no-CS-pre-def*
butlast-nonresizing-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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 → *trail-pol-fast-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-no-CS-def UNSET-def*[*symmetric*] *tl-trailt-tr-no-CS-pre-def*
butlast-nonresizing-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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*))
$\in [\lambda(j, xs).\ j \leq$ *length xs*$]_a$ *nat-assn*$^k *_a$ (*arl-assn R*)$^d \rightarrow$ *arl-assn R*›
apply *sepref-to-hoare*
apply (*sep-auto simp*: *arl-assn-def hr-comp-def take-arl-def intro*!: *list-rel-take*)
apply (*sep-auto simp*: *is-array-list-def list-rel-imp-same-length*[*symmetric*] *min-def*
split: *if-splits*)
done

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*$'$›
supply [[*goals-limit=1*]] *nat-of-uint32-conv-def*[*simp*]
unfolding *trail-conv-back-imp-def*
by *sepref*

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*$'$›
supply [[*goals-limit=1*]]
unfolding *trail-conv-back-imp-def nat-of-uint32-conv-def*
by *sepref*

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*:
‹$(\exists_A ba.\ \uparrow (ba = h) * P\ ba) = P\ h$›
by (*subst ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start'*:
‹$(\exists_A ba.\ \uparrow (h = ba) * P\ ba) = P\ h$›
by (*subst ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start2*:
‹$(\exists_A ba\ b.\ \uparrow (ba = h\ b) * P\ b\ ba) = (\exists_A b .\ \ P\ b\ (h\ b))$›
by (*subst ex-assn-def*, *subst* (*2*) *ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start3*:
‹$(\exists_A ba\ b\ c.\ \uparrow (ba = h\ b) * P\ b\ ba\ c) = (\exists_A b\ c.\ \ P\ b\ (h\ b)\ c)$›
by (*subst ex-assn-def*, *subst* (*3*) *ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start3'*:
‹$(\exists_A ba\ b\ c.\ \uparrow (bb = ba) * P\ b\ ba\ c) = (\exists_A b\ c.\ \ P\ b\ bb\ c)$›
by (*subst ex-assn-def*, *subst* (*3*) *ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start4'*:
‹$(\exists_A ba\ b\ c\ d.\ \uparrow (bb = ba) * P\ b\ ba\ c\ d) = (\exists_A b\ c\ d.\ \ P\ b\ bb\ c\ d)$›
by (*subst ex-assn-def*, *subst* (*4*) *ex-assn-def*, *auto*)+

lemma *ex-assn-def-pure-eq-start1*:
‹$(\exists_A ba.\ \uparrow (ba = h\ b) * P\ ba) = (P\ (h\ b))$›
by (*subst ex-assn-def*, *auto*)+

lemma *ex-assn-cong*:
‹$(\bigwedge x.\ P\ x = P'\ x) \Longrightarrow (\exists_A x.\ P\ x) = (\exists_A x.\ P'\ x)$›
by *auto*

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*⟩
by *sepref-to-hoare*
(*sep-auto simp*: *lookup-clause-assn-is-None-def*)

lemma *NOTIN-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return False*), *uncurry0* (*RETURN NOTIN*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *option-bool-assn*⟩
by *sepref-to-hoare* (*sep-auto simp*: *NOTIN-def option-bool-rel-def*)

lemma *POSIN-hnr*[*sepref-fr-rules*]:
⟨(*return o* (λ-. *True*), *RETURN o ISIN*) ∈ *bool-assn*^{k} $\rightarrow_a$ *option-bool-assn*⟩
by *sepref-to-hoare* (*sep-auto simp*: *ISIN-def option-bool-rel-def*)

lemma *is-NOTIN-hnr*[*sepref-fr-rules*]:
⟨(*return o Not*, *RETURN o is-NOTIN*) ∈ *option-bool-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare* (*sep-auto simp*: *is-NOTIN-def option-bool-rel-def split*: *option.splits*)

lemma (**in** $-$) *SEEN-REMOVABLE*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return SEEN-REMOVABLE*),*uncurry0* (*RETURN SEEN-REMOVABLE*)) ∈
unit-assn^k $\rightarrow_a$ *minimize-status-assn*⟩
by (*sepref-to-hoare*) *sep-auto*

lemma (**in** $-$) *SEEN-FAILED*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return SEEN-FAILED*),*uncurry0* (*RETURN SEEN-FAILED*)) ∈
unit-assn^k $\rightarrow_a$ *minimize-status-assn*⟩
by (*sepref-to-hoare*) *sep-auto*

lemma (**in** $-$) *SEEN-UNKNOWN*[*sepref-fr-rules*]:
⟨(*Sepref-Misc.uncurry0* (*return SEEN-UNKNOWN*),*Sepref-Misc.uncurry0* (*RETURN SEEN-UNKNOWN*))
∈
unit-assn^k $\rightarrow_a$ *minimize-status-assn*⟩
by (*sepref-to-hoare*) *sep-auto*

lemma *size-lookup-conflict*[*sepref-fr-rules*]:
⟨(*return o* (λ(-, *n*, -). *n*), *RETURN o size-lookup-conflict*) ∈
(*bool-assn* ∗*a lookup-clause-rel-assn*)^k $\rightarrow_a$ *uint32-nat-assn*⟩
unfolding *size-lookup-conflict-def*
apply *sep-auto*
apply *sepref-to-hoare*
subgoal for *x xi*
apply (*cases x*, *cases xi*)
apply *sep-auto*
done
done

lemma *option-bool-assn-is-None*[*sepref-fr-rules*]:
⟨(*return o Not*, *RETURN o is-None*) ∈ *option-bool-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare*
(*sep-auto simp*: *option-bool-rel-def hr-comp-def*)

sepref-definition *is-in-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-lookup-conflict*)›
:: ‹[λ((n, xs), L). *atm-of* L < *length* xs]$_a$
lookup-clause-rel-assn^k $*_a$ *unat-lit-assn*^{k} → *bool-assn*›
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
uint32-nat-assn-one[*sepref-fr-rules*] *image-image*[*simp*]
unfolding *is-in-lookup-conflict-def*
by *sepref*

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-option-rel-assn^k $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *lookup-clause-assn-is-empty-def uint32-nat-rel-def br-def nat-of-uint32-0-iff*)

lemma *to-ana-ref-id-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-ana-ref*), *uncurry2* (*RETURN ooo to-ana-ref-id*)) ∈
uint64-nat-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *bool-assn*^{k} $\rightarrow_a$
ana-refinement-fast-assn›
by *sepref-to-hoare*
(*sep-auto simp*: *to-ana-ref-def to-ana-ref-id-def uint32-nat-rel-def*
analyse-refinement-rel-def uint64-nat-rel-def br-def OR-132-is-sum
pure-def ana-refinement-fast-assn-def hr-comp-def
nat-of-uint64-uint64-of-uint32
nat-of-uint32-le-uint32-max)

lemma *to-ana-ref-id-hnr*[*sepref-fr-rules*]:
‹(*uncurry2* (*return ooo to-ana-ref*), *uncurry2* (*RETURN ooo to-ana-ref-id*)) ∈
nat-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *bool-assn*^{k} $\rightarrow_a$
ana-refinement-assn›
by *sepref-to-hoare*
(*sep-auto simp*: *to-ana-ref-def to-ana-ref-id-def uint32-nat-rel-def*
analyse-refinement-rel-def uint64-nat-rel-def br-def OR-132-is-sum
pure-def ana-refinement-assn-def hr-comp-def
nat-of-uint64-uint64-of-uint32
nat-of-uint32-le-uint32-max)

lemma [*sepref-fr-rules*]:
‹((*return o from-ana-ref*), (*RETURN o from-ana-ref-id*)) ∈
ana-refinement-fast-assn^k $\rightarrow_a$
uint64-nat-assn ∗*a* *uint32-nat-assn* ∗*a* *bool-assn*›
proof −
have ‹(*4294967296*::*uint64*) = (*0*::*uint64*) ⟷ (*0* :: *uint64*) = *4294967296*›
by *argo*
also have ‹. . . ⟷ *False*›
by *auto*
finally have [*iff*]: ‹(*4294967296*::*uint64*) ≠ (*0*::*uint64*)›
by *blast*
have *eq*: ‹(*1*::*uint64*) << (*32*::*nat*) = *4294967296*›
by (*auto simp*: *numeral-eq-Suc shiftl-t2n-uint64*)

show *?thesis*
apply *sepref-to-hoare*
apply (*case-tac xi*)

apply
(*sep-auto simp*: *from-ana-ref-def from-ana-ref-id-def*
analyse-refinement-rel-def uint64-nat-rel-def br-def
case-prod-beta ana-refinement-fast-assn-def pure-def)
apply (*auto simp*: *hr-comp-def uint32-nat-rel-def br-def*
take-only-lower32-le-uint32-max nat-of-uint64-uint64-of-uint32
nat-of-uint32-le-uint32-max nat-of-uint64-1-32 take-only-lower32-1-32
take-only-lower32-le-uint32-max-ge-uint32-max AND-2-32-bool
le-uint32-max-AND2-32-eq0)
apply (*auto simp*: *eq simp del*: *star-aci*(*2*))[]
apply (*subst norm-assertion-simps*(*17*)[*symmetric*])
apply (*subst star-aci*(*2*))
apply (*rule ent-refl-true*)
done
qed

lemma [*sepref-fr-rules*]:
‹((*return o from-ana-ref*), (*RETURN o from-ana-ref-id*)) ∈
ana-refinement-assn^k $\rightarrow_a$
nat-assn ∗*a uint32-nat-assn* ∗*a bool-assn*›
proof −
have ‹(*4294967296*::*uint64*) = (*0*::*uint64*) ⟷ (*0* :: *uint64*) = *4294967296*›
by *argo*
also have ‹. . . ⟷ *False*›
by *auto*
finally have [*iff*]: ‹(*4294967296*::*uint64*) ≠ (*0*::*uint64*)›
by *blast*
have *eq*: ‹(*1*::*uint64*) << (*32*::*nat*) = *4294967296*›
by (*auto simp*: *numeral-eq-Suc shiftl-t2n-uint64*)

show *?thesis*
apply *sepref-to-hoare*
apply (*case-tac xi*)
apply
(*sep-auto simp*: *from-ana-ref-def from-ana-ref-id-def*
analyse-refinement-rel-def uint64-nat-rel-def br-def
case-prod-beta ana-refinement-assn-def pure-def)
apply (*auto simp*: *hr-comp-def uint32-nat-rel-def br-def*
take-only-lower32-le-uint32-max nat-of-uint64-uint64-of-uint32
nat-of-uint32-le-uint32-max nat-of-uint64-1-32 take-only-lower32-1-32
take-only-lower32-le-uint32-max-ge-uint32-max AND-2-32-bool
le-uint32-max-AND2-32-eq0)
apply (*auto simp*: *eq simp del*: *star-aci*(*2*))[]
apply (*subst norm-assertion-simps*(*17*)[*symmetric*])
apply (*subst star-aci*(*2*))
apply (*rule ent-refl-true*)
done
qed

lemma *minimize-status-eq-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo* (=)), *uncurry* (*RETURN oo* (=))) ∈
minimize-status-assn^k $*_a$ *minimize-status-assn*^{k} $\rightarrow_a$ *bool-assn*›
by (*sepref-to-hoare*) (*sep-auto*)

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*⟩
  :: ⟨*unat-lit-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $\rightarrow_a$ *lookup-clause-rel-assn*⟩
  unfolding *delete-from-lookup-conflict-def NOTIN-def*[*symmetric*]
  by *sepref*

sepref-definition *resolve-lookup-conflict-merge-code*
  is ⟨*uncurry6 isa-set-lookup-conflict*⟩
  :: ⟨[λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), *out*). *i* < *length N*]$_a$
      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} →
      conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*⟩
  supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
    uint32-nat-assn-one[*sepref-fr-rules*] *image-image*[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
    literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
    Suc-uint32-nat-assn-hnr[*sepref-fr-rules*] *fmap-length-rll-u-def*[*simp*]
  unfolding *isa-lookup-conflict-merge-def lookup-conflict-merge-def add-to-lookup-conflict-def*
    PR-CONST-def nth-rll-def[*symmetric*]
    isa-outlearned-add-def isa-clvls-add-def isa-set-lookup-conflict-def
    isasat-codegen
    fmap-rll-u-def[*symmetric*]
    fmap-rll-def[*symmetric*]
    is-NOTIN-def[*symmetric*]
  apply (*rewrite at* ⟨- + ⌑⟩ *nat-of-uint64-conv-def*[*symmetric*])
  supply [[*goals-limit* = *1*]]
  by *sepref*

declare *resolve-lookup-conflict-merge-code.refine*[*sepref-fr-rules*]

sepref-definition *resolve-lookup-conflict-merge-fast-code*
  is ⟨*uncurry6 isa-set-lookup-conflict*⟩
  :: ⟨[λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), *out*). *i* < *length N* ∧
        length N ≤ *uint64-max*]$_a$
      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} →
      conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*⟩
  supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
    uint32-nat-assn-one[*sepref-fr-rules*] *image-image*[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
    literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
    Suc-uint32-nat-assn-hnr[*sepref-fr-rules*] *fmap-length-rll-u-def*[*simp*]
    arena-is-valid-clause-idx-le-uint64-max[*intro*]
  unfolding *isa-lookup-conflict-merge-def lookup-conflict-merge-def add-to-lookup-conflict-def*
    PR-CONST-def nth-rll-def[*symmetric*]
    isa-outlearned-add-def isa-clvls-add-def isa-set-lookup-conflict-def
    isa-set-lookup-conflict-def
    fmap-rll-u-def[*symmetric*]
    fmap-rll-def[*symmetric*]
    is-NOTIN-def[*symmetric*]
    zero-uint64-nat-def[*symmetric*]
  apply (*rewrite at* ⟨*RETURN* (⌑, - ,-, -)⟩ *Suc-eq-plus1*)
  apply (*rewrite at* ⟨*RETURN* (- + ⌑, - ,-, -)⟩ *one-uint64-nat-def*[*symmetric*])
  supply [[*goals-limit* = *1*]]

by *sepref*

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*›
:: ‹*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*›
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]
unfolding *set-lookup-conflict-aa-def lookup-conflict-merge-def add-to-lookup-conflict-def*
PR-CONST-def nth-rll-def[*symmetric*] *length-rll-def*[*symmetric*]
length-aa-u-def[*symmetric*] *isa-outlearned-add-def isa-clvls-add-def*
isasat-codegen isa-set-lookup-conflict-aa-def isa-lookup-conflict-merge-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *isa-set-lookup-conflict-aa-pre-def*
supply [[*goals-limit* = *1*]]
apply (*rewrite at* ‹- + ⌑› *nat-of-uint64-conv-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
by *sepref*

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*›
:: ‹[λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), -). *length N* ≤ *uint64-max*]$_a$
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} →
conflict-option-rel-assn $*a$ *uint32-nat-assn* $*a$ *lbd-assn* $*a$ *out-learned-assn*›
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]
arena-is-valid-clause-idx-le-uint64-max[*intro*]
unfolding *set-lookup-conflict-aa-def lookup-conflict-merge-def add-to-lookup-conflict-def*
PR-CONST-def nth-rll-def[*symmetric*] *length-rll-def*[*symmetric*]
length-aa-u-def[*symmetric*] *isa-outlearned-add-def isa-clvls-add-def*
isasat-codegen isa-set-lookup-conflict-aa-def isa-lookup-conflict-merge-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *isa-set-lookup-conflict-aa-pre-def zero-uint64-nat-def*[*symmetric*]
supply [[*goals-limit* = *1*]]
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite at* ‹*RETURN* (⌑, - ,-, -)› *Suc-eq-plus1*)
apply (*rewrite at* ‹*RETURN* (- + ⌑, - ,-, -)› *one-uint64-nat-def*[*symmetric*])
supply [[*goals-limit* = *1*]]
by *sepref*

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-set-lookup-conflict-aa-pre*]$_a$
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} →
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]
unfolding *set-lookup-conflict-aa-def lookup-conflict-merge-def add-to-lookup-conflict-def*
PR-CONST-def nth-rll-def[*symmetric*] *length-rll-def*[*symmetric*]
length-aa-u-def[*symmetric*] *isa-outlearned-add-def isa-clvls-add-def*
isasat-codegen isa-set-lookup-conflict-aa-def isa-lookup-conflict-merge-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *isa-set-lookup-conflict-aa-pre-def*
isa-resolve-merge-conflict-gt2-def
apply (*rewrite at* ‹- + ¤› *nat-of-uint64-conv-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
supply [[*goals-limit* = *1*]]
by *sepref*

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-max* ∧
isa-set-lookup-conflict-aa-pre ((((((*M*, *N*), *i*), (*b*, *xs*)), *clvls*), *lbd*), *outl*))]$_a$
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} →
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*›
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*] *uint-max-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]
arena-is-valid-clause-idx-le-uint64-max[*intro*]
unfolding *set-lookup-conflict-aa-def lookup-conflict-merge-def add-to-lookup-conflict-def*
PR-CONST-def nth-rll-def[*symmetric*] *length-rll-def*[*symmetric*]
length-aa-u-def[*symmetric*] *isa-outlearned-add-def isa-clvls-add-def*
isasat-codegen isa-set-lookup-conflict-aa-def isa-lookup-conflict-merge-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*] *nat-of-uint64-conv-def*
is-NOTIN-def[*symmetric*] *isa-set-lookup-conflict-aa-pre-def*
isa-resolve-merge-conflict-gt2-def
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + *1*› *one-uint64-nat-def*[*symmetric*])

apply (*rewrite* **in** ‹*RETURN* (*Suc* -, -)› *Suc-eq-plus1*)
apply (*rewrite* **in** ‹- + *1*› *one-uint64-nat-def*[*symmetric*])
supply [[*goals-limit* = *1*]]
by *sepref*

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-in-conflict-lookup-pre*]$_a$
uint32-nat-assn^k $*_a$ *lookup-clause-rel-assn*^{k} → *bool-assn*›
unfolding *atm-in-conflict-lookup-def atm-in-conflict-lookup-pre-def*
by *sepref*

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-min-cach-l-pre*]$_a$ *cach-refinement-l-assn*^{k} $*_a$ *uint32-nat-assn*^{k} → *minimize-status-assn*›
unfolding *conflict-min-cach-l-def conflict-min-cach-l-pre-def*
by *sepref*

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*)
})›
unfolding *conflict-min-cach-set-failed-l-def Let-def* **by** *auto*

lemma *le-uint32-max-div2-le-uint32-max*: ‹*a2'* ≤ *Suc* (*uint-max div 2*) ⟹ *a2'* < *uint-max*›
by (*auto simp*: *uint32-max-def*)

sepref-definition (**in** −) *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹*cach-refinement-l-assn*^{d} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *cach-refinement-l-assn*›
supply *arl-append-hnr*[*sepref-fr-rules*] *le-uint32-max-div2-le-uint32-max*[*intro*]
unfolding *conflict-min-cach-set-failed-l-alt-def*
by *sepref*

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*)
})›
unfolding *conflict-min-cach-set-removable-l-def* **by** *auto*

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*›
supply *arl-append-hnr*[*sepref-fr-rules*] *le-uint32-max-div2-le-uint32-max*[*intro*]

unfolding *conflict-min-cach-set-removable-l-alt-def*
by *sepref*

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*) $\in$ *lookup-clause-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
by *sepref-to-hoare sep-auto*

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
by *auto*
lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure ana-refinement-fast-assn*›
unfolding *CONSTRAINT-def ana-refinement-fast-assn-def*
by (*auto intro*: *hr-comp-is-pure*)

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure ana-refinement-assn*›
unfolding *CONSTRAINT-def ana-refinement-assn-def*
by (*auto intro*: *hr-comp-is-pure*)

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*) $\in$
lookup-clause-rel-assn^d $\rightarrow_a$ *conflict-option-rel-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *set-lookup-empty-conflict-to-none-def*)

lemma *isa-mark-failed-lits-stackI*:
assumes
‹*length ba* $\leq$ *Suc* (*uint-max div 2*)› **and**
‹*a1'* < *length ba*›
shows ‹*Suc a1'* $\leq$ *uint-max*›
using *assms* **by** (*auto simp*: *uint32-max-def*)

sepref-register *to-ana-ref-id*
sepref-definition *isa-mark-failed-lits-stack-code*
is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
:: ‹*arena-assn*^{k} $*_a$ *analyse-refinement-assn*^{d} $*_a$ *cach-refinement-l-assn*^{d} $\rightarrow_a$
cach-refinement-l-assn›
supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*!] *image-image*[*simp*] *length-rll-def*[*simp*]
mark-failed-lits-stack-inv-helper1[*dest*] *mark-failed-lits-stack-inv-helper2*[*dest*]

fmap-length-rll-u-def[*simp*] *isa-mark-failed-lits-stackI*[*intro*] *le-uint32-max-div2-le-uint32-max*[*intro*]
unfolding *isa-mark-failed-lits-stack-def PR-CONST-def*
conflict-min-cach-set-failed-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*]
fmap-rll-def[*symmetric*]
conflict-min-cach-set-failed-l-alt-def
apply (*rewrite at* ‹*arena-lit* - (- + ⌑ − -)› *nat-of-uint32-conv-def*[*symmetric*])
apply (*rewrite* **in** ‹- + ⌑› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹(⌑, -)› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

sepref-definition *isa-mark-failed-lits-stack-fast-code*
is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
:: ‹[λ((*N*, -), -). *length N* ≤ *uint64-max*]$_a$
arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{d} $*_a$ *cach-refinement-l-assn*^{d} →
cach-refinement-l-assn›
supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*!] *image-image*[*simp*] *length-rll-def*[*simp*]
mark-failed-lits-stack-inv-helper1[*dest*] *mark-failed-lits-stack-inv-helper2*[*dest*]
fmap-length-rll-u-def[*simp*] *isa-mark-failed-lits-stackI*[*intro*]
arena-is-valid-clause-idx-le-uint64-max[*intro*] *le-uint32-max-div2-le-uint32-max*[*intro*]
unfolding *isa-mark-failed-lits-stack-def PR-CONST-def*
conflict-min-cach-set-failed-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*]
fmap-rll-def[*symmetric*]
arena-lit-def[*symmetric*]
conflict-min-cach-set-failed-l-alt-def
apply (*rewrite at* ‹*arena-lit* - (- + ⌑ − -)› *uint64-of-uint32-conv-def*[*symmetric*])
apply (*rewrite* **in** ‹- − ⌑› *one-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- − ⌑› *one-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- − ⌑› *one-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹- + ⌑› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹(⌑, -)› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

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-get-literal-and-remove-of-analyse-wl-pre*]$_a$
arena-assn^k $*_a$ *analyse-refinement-assn*^{d} →
unat-lit-assn $*a$ *analyse-refinement-assn*›
unfolding *isa-get-literal-and-remove-of-analyse-wl-pre-def*
isa-get-literal-and-remove-of-analyse-wl-def fast-minus-def[*symmetric*]
one-uint32-nat-def[*symmetric*]
apply (*rewrite at* ‹*arena-lit* - (- + ⌑)› *nat-of-uint32-conv-def*[*symmetric*])
by *sepref*

sepref-definition *isa-get-literal-and-remove-of-analyse-wl-fast-code*
is ‹*uncurry* (*RETURN oo isa-get-literal-and-remove-of-analyse-wl*)›
:: ‹[λ(*arena*, *analyse*). *isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ∧

length arena $\leq$ *uint64-max*]$_a$
arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{d} $\rightarrow$
unat-lit-assn $*a$ *analyse-refinement-fast-assn*›
supply [[*goals-limit=1*]] *arena-lit-pre-le2*[*dest*]
unfolding *isa-get-literal-and-remove-of-analyse-wl-pre-def*
isa-get-literal-and-remove-of-analyse-wl-def fast-minus-def[*symmetric*]
one-uint32-nat-def[*symmetric*]
apply (*rewrite at* ‹*arena-lit* - (- + ¤)› *uint64-of-uint32-conv-def*[*symmetric*])
by *sepref*

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-lookup-conv-lookup-pre*]$_a$ *arena-fast-assn*^{k} $*_a$
(*uint64-nat-assn* $*a$ *uint32-nat-assn* $*a$ *bool-assn*)^k
$\rightarrow$ *uint64-nat-assn* $*a$ *uint64-nat-assn* $*a$ *uint64-nat-assn* $*a$ *uint64-nat-assn*›
unfolding *ana-lookup-conv-lookup-pre-def ana-lookup-conv-lookup-def*
zero-uint64-nat-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
apply (*rewrite at* ‹(-, -, ¤, -)› *uint64-of-uint32-conv-def*[*symmetric*])
by *sepref*

sepref-definition *ana-lookup-conv-lookup-code*
is ‹*uncurry* (*RETURN oo ana-lookup-conv-lookup*)›
:: ‹[*uncurry ana-lookup-conv-lookup-pre*]$_a$ *arena-assn*^{k} $*_a$
(*nat-assn* $*a$ *uint32-nat-assn* $*a$ *bool-assn*)^k
$\rightarrow$ *nat-assn* $*a$ *uint64-nat-assn* $*a$ *uint64-nat-assn* $*a$ *uint64-nat-assn*›
unfolding *ana-lookup-conv-lookup-pre-def ana-lookup-conv-lookup-def*
zero-uint64-nat-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
apply (*rewrite at* ‹(-, -, ¤, -)› *uint64-of-uint32-conv-def*[*symmetric*])
by *sepref*

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›
unfolding *lit-redundant-reason-stack-wl-lookup-def lit-redundant-reason-stack-wl-lookup-pre-def*
one-uint32-nat-def[*symmetric*] *zero-uint32-nat-def*[*symmetric*]
apply (*rewrite at* ‹*2* < ¤› *nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

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›
unfolding *lit-redundant-reason-stack-wl-lookup-def lit-redundant-reason-stack-wl-lookup-pre-def*
two-uint64-nat-def[*symmetric*] *zero-uint32-nat-def*[*symmetric*]
one-uint32-nat-def[*symmetric*]
by *sepref*

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*⟩
using *assms* **by** (*auto simp*: *uint32-max-def*)

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} →
cach-refinement-l-assn $*a$ *analyse-refinement-assn* $*a$ *bool-assn*⟩
supply [[*goals-limit = 1*]] *neq-Nil-revE*[*elim*] *image-image*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms*[*intro*] *length-rll-def*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *nth-rll-def*[*simp*]
fmap-length-rll-u-def[*simp*] *isa-lit-redundant-rec-wl-lookupI*[*intro*]
fmap-length-rll-def[*simp*] *isa-mark-failed-lits-stackI*[*intro*]
unfolding *isa-lit-redundant-rec-wl-lookup-def*
conflict-min-cach-set-removable-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*] *PR-CONST-def*
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
butlast-nonresizing-def[*symmetric*]
nat-of-uint64-conv-def
apply (*rewrite at* ⟨(-, ¤, -)⟩ *arl32.fold-custom-empty*)+
apply (*rewrite at* ⟨*op-arl32-empty*⟩ *annotate-assn*[**where** *A*=*analyse-refinement-assn*])
unfolding *nth-rll-def*[*symmetric*] *length-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
apply (*rewrite at* ⟨*arena-lit* - (- + ¤)⟩ *nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

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} →
cach-refinement-l-assn $*a$ *analyse-refinement-fast-assn* $*a$ *bool-assn*⟩
supply [[*goals-limit = 1*]] *neq-Nil-revE*[*elim*] *image-image*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms*[*intro*] *length-rll-def*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *nth-rll-def*[*simp*]
fmap-length-rll-u-def[*simp*]

fmap-length-rll-def[*simp*] *isa-lit-redundant-rec-wl-lookupI*[*intro*]
arena-lit-pre-le[*intro*] *isa-mark-failed-lits-stackI*[*intro*]
unfolding *isa-lit-redundant-rec-wl-lookup-def*
conflict-min-cach-set-removable-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*] *PR-CONST-def*
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
butlast-nonresizing-def[*symmetric*]
nat-of-uint64-conv-def
apply (*rewrite at* ‹(-, ¤, -)› *arl32.fold-custom-empty*)+
apply (*rewrite at* ‹*op-arl32-empty*› *annotate-assn*[**where** *A*=*analyse-refinement-fast-assn*])

unfolding *nth-rll-def*[*symmetric*] *length-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
unfolding *nth-rll-def*[*symmetric*] *length-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
zero-uint32-nat-def[*symmetric*]
fmap-rll-u-def[*symmetric*]
by *sepref*

declare *lit-redundant-rec-wl-lookup-fast-code.refine*[*sepref-fr-rules*]

definition *arl32-butlast-nonresizing* :: ‹*'a array-list32* ⇒ *'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*) ∈
$[\lambda xs.\ xs \neq []]_a$ $(arl32\text{-}assn\ R)^d \rightarrow arl32\text{-}assn\ R$›
proof −
have [*simp*]: ‹*nat-of-uint32* (*b* − *1*) = *nat-of-uint32 b* − *1*›
if
‹*x* ≠ []› **and**
‹(*take* (*nat-of-uint32 b*) *l'*, *x*) ∈ ⟨*the-pure R*⟩*list-rel*›
for *x* :: ‹*'b list*› **and** *a* :: ‹*'a array*› **and** *b* :: ‹*uint32*› **and** *l'* :: ‹*'a list*› **and** *aa* :: ‹*Heap.heap*› **and** *ba* :: ‹*nat set*›
by (*metis less-one list-rel-pres-neq-nil nat-of-uint32-012*(*3*) *nat-of-uint32-less-iff*
nat-of-uint32-notle-minus take-eq-Nil that)

show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *arl32-butlast-nonresizing-def arl32-assn-def hr-comp-def*
is-array-list32-def *butlast-take list-rel-imp-same-length nat-of-uint32-ge-minus*
dest:
list-rel-butlast[*of* ‹*take* - -›]
simp flip: *nat-of-uint32-le-iff*)
qed

find-theorems *butlast arl32-assn*
sepref-definition *delete-index-and-swap-code*
is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›

:: ⟨[λ(*xs*, *i*). *i* < *length xs*]$_a$
  (*arl32-assn unat-lit-assn*)^d $*_a$ *uint32-nat-assn*^{k} → *arl32-assn unat-lit-assn*⟩
unfolding *delete-index-and-swap.simps butlast-nonresizing-def*[*symmetric*]
by *sepref*

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*)⟩
:: ⟨[λ((*n*, *xs*), *i*). *i* < *length xs* ∧ *n* > *0*]$_a$
  lookup-clause-rel-assn^d $*_a$ *uint32-nat-assn*^{k} → *lookup-clause-rel-assn*⟩
unfolding *lookup-conflict-upd-None-RETURN-def fast-minus-def*[*symmetric*]
by *sepref*

declare *lookup-conflict-upd-None-code.refine*[*sepref-fr-rules*]

lemma *uint32-max-ge0*: ⟨*0* < *uint-max*⟩ **by** (*auto simp*: *uint32-max-def*)
sepref-definition *literal-redundant-wl-lookup-code*
is ⟨*uncurry5 isa-literal-redundant-wl-lookup*⟩
:: ⟨[λ(((((*M*, *NU*), *D*), *cach*), *L*), *lbd*). *True*]$_a$
  trail-pol-assn^k $*_a$ *arena-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{k} $*_a$
  cach-refinement-l-assn^d $*_a$ *unat-lit-assn*^{k} $*_a$ *lbd-assn*^{k} →
  cach-refinement-l-assn ∗*a analyse-refinement-assn* ∗*a bool-assn*⟩
supply [[*goals-limit=1*]] *Pos-unat-lit-assn*[*sepref-fr-rules*] *Neg-unat-lit-assn*[*sepref-fr-rules*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*] *op-arl32-empty-def*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *uint32-max-ge0*[*intro*!]
unfolding *isa-literal-redundant-wl-lookup-def zero-uint32-nat-def*[*symmetric*] *PR-CONST-def*
apply (*rewrite at* ⟨(-, ⌑, -)⟩ *arl32.fold-custom-empty*)+
unfolding *single-replicate*
unfolding *arl32.fold-custom-empty*
by *sepref*

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*⟩
:: ⟨[λ(((((*M*, *NU*), *D*), *cach*), *L*), *lbd*). *length NU* ≤ *uint64-max*]$_a$
  trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{k} $*_a$
  cach-refinement-l-assn^d $*_a$ *unat-lit-assn*^{k} $*_a$ *lbd-assn*^{k} →
  cach-refinement-l-assn ∗*a analyse-refinement-fast-assn* ∗*a bool-assn*⟩
supply [[*goals-limit=1*]] *Pos-unat-lit-assn*[*sepref-fr-rules*] *Neg-unat-lit-assn*[*sepref-fr-rules*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*] *uint32-max-ge0*[*intro*!]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *op-arl32-empty-def*[*simp*]
unfolding *isa-literal-redundant-wl-lookup-def zero-uint32-nat-def*[*symmetric*] *PR-CONST-def*
apply (*rewrite at* ⟨(-, ⌑, -)⟩ *arl32.fold-custom-empty*)+
unfolding *single-replicate one-uint64-nat-def*[*symmetric*]
unfolding *arl32.fold-custom-empty*
by *sepref*

declare *literal-redundant-wl-lookup-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *conflict-remove1-code*
is ⟨*uncurry* (*RETURN oo lookup-conflict-remove1*)⟩
:: ⟨[*lookup-conflict-remove1-pre*]$_a$ *unat-lit-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} →
  lookup-clause-rel-assn⟩
supply [[*goals-limit=2*]] *one-uint32-nat*[*sepref-fr-rules*]

unfolding *lookup-conflict-remove1-def one-uint32-nat-def*[*symmetric*] *fast-minus-def*[*symmetric*]
lookup-conflict-remove1-pre-def
by *sepref*

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*)›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *lbd*), *outl*). *True*]$_a$
trail-pol-assn^k $*_a$ *arena-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $*_a$
cach-refinement-l-assn^d $*_a$ *lbd-assn*^{k} $*_a$ *out-learned-assn*^{d} →
lookup-clause-rel-assn $*a$ *cach-refinement-l-assn* $*a$ *out-learned-assn*›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn*[*sepref-fr-rules*] *Neg-unat-lit-assn*[*sepref-fr-rules*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
minimize-and-extract-highest-lookup-conflict-inv-def[*simp*]
in-$\mathcal{L}_{all}$*-less-uint-max'*[*intro*] *length-u-hnr*[*sepref-fr-rules*]
array-set-hnr-u[*sepref-fr-rules*]
unfolding *isa-minimize-and-extract-highest-lookup-conflict-def zero-uint32-nat-def*[*symmetric*]
one-uint32-nat-def[*symmetric*] *PR-CONST-def*
minimize-and-extract-highest-lookup-conflict-inv-def
by *sepref*

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*›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *lbd*), *outl*). *length NU* ≤ *uint64-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} →
lookup-clause-rel-assn $*a$ *cach-refinement-l-assn* $*a$ *out-learned-assn*›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn*[*sepref-fr-rules*] *Neg-unat-lit-assn*[*sepref-fr-rules*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
minimize-and-extract-highest-lookup-conflict-inv-def[*simp*]
in-$\mathcal{L}_{all}$*-less-uint-max'*[*intro*] *length-u-hnr*[*sepref-fr-rules*]
array-set-hnr-u[*sepref-fr-rules*]
unfolding *isa-minimize-and-extract-highest-lookup-conflict-def zero-uint32-nat-def*[*symmetric*]
one-uint32-nat-def[*symmetric*] *PR-CONST-def*
minimize-and-extract-highest-lookup-conflict-inv-def
by *sepref*

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*›
supply [[*goals-limit* = *1*]]
unfolding *isasat-lookup-merge-eq2-def add-to-lookup-conflict-def*
isa-outlearned-add-def isa-clvls-add-def
is-NOTIN-def[*symmetric*]
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]

arena-is-valid-clause-idx-le-uint64-max[*intro*]
apply (*rewrite* **in** ⟨*if* - *then* - + ⌑ *else* -⟩ *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨*if* - *then* - + ⌑ *else* -⟩ *one-uint32-nat-def*[*symmetric*])
by *sepref*

sepref-definition *isasat-lookup-merge-eq2-fast-code*
is ⟨*uncurry7 isasat-lookup-merge-eq2*⟩
:: ⟨$[\lambda(((((((L, M), NU), -), -), -), -), -).\ length\ NU \leq uint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a$
$conflict\text{-}option\text{-}rel\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a lbd\text{-}assn^d *_a out\text{-}learned\text{-}assn^d \rightarrow$
conflict-option-rel-assn ∗*a uint32-nat-assn* ∗*a lbd-assn* ∗*a out-learned-assn*⟩
supply [[*goals-limit* = *1*]]
unfolding *isasat-lookup-merge-eq2-def add-to-lookup-conflict-def*
isa-outlearned-add-def isa-clvls-add-def
is-NOTIN-def[*symmetric*]
supply *length-rll-def*[*simp*] *nth-rll-def*[*simp*]
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint-max*[*dest*]
fmap-length-rll-u-def[*simp*]
arena-is-valid-clause-idx-le-uint64-max[*dest*]
arena-lit-pre-le2[*dest*]
apply (*rewrite* **in** ⟨*if* - *then* - + ⌑ *else* -⟩ *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨*if* - *then* - + ⌑ *else* -⟩ *one-uint32-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨*if* - *then arena-lit* - (- + ⌑) *else* -⟩ *one-uint64-nat-def*[*symmetric*])
by *sepref*

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*) ∈ $ema\text{-}assn^k \rightarrow_a uint64\text{-}assn$⟩
by *sepref-to-hoare sep-auto*

sepref-register *ema-bitshifting*

lemma *incr-propagation-hnr*[*sepref-fr-rules*]:
⟨(*return o incr-propagation*, *RETURN o incr-propagation*) ∈ $stats\text{-}assn^d \rightarrow_a stats\text{-}assn$⟩
by *sepref-to-hoare* (*sep-auto simp*: *incr-propagation-def*)

lemma *incr-conflict-hnr*[*sepref-fr-rules*]:
‹(*return o incr-conflict*, *RETURN o incr-conflict*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-conflict-def*)

lemma *incr-decision-hnr*[*sepref-fr-rules*]:
‹(*return o incr-decision*, *RETURN o incr-decision*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-decision-def*)

lemma *incr-restart-hnr*[*sepref-fr-rules*]:
‹(*return o incr-restart*, *RETURN o incr-restart*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-restart-def*)

lemma *incr-lrestart-hnr*[*sepref-fr-rules*]:
‹(*return o incr-lrestart*, *RETURN o incr-lrestart*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-lrestart-def*)

lemma *incr-uset-hnr*[*sepref-fr-rules*]:
‹(*return o incr-uset*, *RETURN o incr-uset*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-uset-def*)

lemma *incr-GC-hnr*[*sepref-fr-rules*]:
‹(*return o incr-GC*, *RETURN o incr-GC*) ∈ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-GC-def*)

lemma *add-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry* (*return oo add-lbd*), *uncurry* (*RETURN oo add-lbd*)) ∈ *uint64-assn*^{k} $*_a$ *stats-assn*^{d} →$_a$ *stats-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *add-lbd-def*)

lemma *ema-bitshifting-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4294967296*), *uncurry0* (*RETURN ema-bitshifting*)) ∈
unit-assn^k →$_a$ *uint64-nat-assn*›
proof −
have [*simp*]: ‹*Suc 0* << *32* = *4294967296*›
by *eval*
show *?thesis*
unfolding *ema-bitshifting-def*
by *sepref-to-hoare*
(*sep-auto simp*: *uint64-nat-rel-def br-def ema-bitshifting-def*
nat-of-uint64-1-32 uint32-max-def)
qed

lemma *ema-bitshifting-hnr2*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 4294967296*), *uncurry0* (*RETURN ema-bitshifting*)) ∈
unit-assn^k →$_a$ *uint64-assn*›
proof −
have [*simp*]: ‹(*1*::*uint64*) << *32* = *4294967296*›
by *eval*
show *?thesis*
unfolding *ema-bitshifting-def*
by *sepref-to-hoare*
(*sep-auto simp*: *uint64-nat-rel-def br-def ema-bitshifting-def*
nat-of-uint64-1-32 uint32-max-def)
qed

lemma (**in** −) *ema-update-hnr*[*sepref-fr-rules*]:

‹(*uncurry* (*return oo ema-update-ref*), *uncurry* (*RETURN oo ema-update*)) ∈ *uint32-nat-assn*^{k} $*_a$ *ema-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *ema-update-def ema-update-ref-def*
by *sepref-to-hoare*
(*sep-auto simp*: *uint32-nat-rel-def br-def uint64-of-uint32-def Let-def*)

lemma *ema-reinit-hnr*[*sepref-fr-rules*]:
‹(*return o ema-reinit*, *RETURN o ema-reinit*) ∈ *ema-assn*^{k} $\rightarrow_a$ *ema-assn*›
by *sepref-to-hoare sep-auto*

lemma (**in** −) *ema-init-coeff-hnr*[*sepref-fr-rules*]:
‹(*return o ema-init*, *RETURN o ema-init*) ∈ *uint64-assn*^{k} $\rightarrow_a$ *ema-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *ema-init-def uint64-nat-rel-def br-def*)

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*) ∈ *restart-info-assn*^{d} $\rightarrow_a$ *restart-info-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *incr-conflict-count-since-last-restart-def*)

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*))
∈ *uint32-assn*^{k} $*_a$ *restart-info-assn*^{d} $\rightarrow_a$ *restart-info-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *restart-info-update-lvl-avg-def*)

lemma *restart-info-init-hnr*[*sepref-fr-rules*]:
‹(*uncurry0* (*return restart-info-init*),
uncurry0 (*RETURN restart-info-init*))
∈ *unit-assn*^{k} $\rightarrow_a$ *restart-info-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *restart-info-init-def*)

lemma *restart-info-restart-done-hnr*[*sepref-fr-rules*]:
‹(*return o restart-info-restart-done*, *RETURN o restart-info-restart-done*) ∈ *restart-info-assn*^{d} $\rightarrow_a$ *restart-info-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *restart-info-restart-done-def*
uint64-nat-rel-def br-def)

type-synonym *vmtf-remove-assn* = ‹*vmtf-assn* × (*uint32 array-list32* × *bool array*)›

abbreviation (**in** −)*vmtf-node-assn* **where**
‹*vmtf-node-assn* ≡ *pure vmtf-node-rel*›

abbreviation *vmtf-conc* **where**
‹*vmtf-conc* ≡ (*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* ⇒ *bool array* ⇒ *assn*› **where**
‹*atoms-hash-assn* ≡ *array-assn bool-assn*›

abbreviation *distinct-atoms-assn* **where**
‹*distinct-atoms-assn* ≡ *arl32-assn uint32-nat-assn* ∗*a* *atoms-hash-assn*›

abbreviation *vmtf-remove-conc*
:: ⟨*isa-vmtf-remove-int* ⇒ *vmtf-remove-assn* ⇒ *assn*⟩
where
⟨*vmtf-remove-conc* ≡ *vmtf-conc* ∗*a* *distinct-atoms-assn*⟩

Options **abbreviation** *opts-assn*
:: ⟨*opts* ⇒ *opts* ⇒ *assn*⟩
where
⟨*opts-assn* ≡ *bool-assn* ∗*a* *bool-assn* ∗*a* *bool-assn*⟩

lemma *opts-restart-hnr*[*sepref-fr-rules*]:
⟨(*return o opts-restart*, *RETURN o opts-restart*) ∈ $opts\text{-}assn^k \rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare sep-auto*

lemma *opts-reduce-hnr*[*sepref-fr-rules*]:
⟨(*return o opts-reduce*, *RETURN o opts-reduce*) ∈ $opts\text{-}assn^k \rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare sep-auto*

lemma *opts-unbounded-mode-hnr*[*sepref-fr-rules*]:
⟨(*return o opts-unbounded-mode*, *RETURN o opts-unbounded-mode*) ∈ $opts\text{-}assn^k \rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare sep-auto*

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 −
have [*simp*]: ⟨(λ(*n*, *bL*). (*nat-of-uint64-conv n*, *bL*)) = *id*⟩
by (*auto intro*!: *ext simp*: *nat-of-uint64-conv-def*)
show *?thesis*
by (*auto simp*: *convert-wlists-to-nat-def*)
qed

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*)]⟩
unfolding *convert-single-wl-to-nat-conv-def*
by *auto*

lemma *convert-wlists-to-nat-convert-wlists-to-nat-conv*:
⟨(*convert-wlists-to-nat*, *RETURN o convert-wlists-to-nat-conv*) ∈
⟨⟨*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*⟩
by (*intro WB-More-Refinement.frefI nres-relI*)
(*auto simp*: *convert-wlists-to-nat-def*
convert-wlists-to-nat-conv-def
intro: *order.trans op-map-map*)

lemma *convert-wlists-to-nat-alt-def2*:
⟨*convert-wlists-to-nat xs* = *do* {
let n = *length xs*;
let zs = *init-lrl n*;
(*uu*, *zs*) ←

$WHILE_T{}^{\lambda(i,\ zs).\qquad i \le length\ xs\ \wedge \qquad take\ i\ zs = \qquad map\ (map\ (\lambda(n,\ y,\ y').\ (nat\text{-}of\text{-}uint64\text{-}c}$
  $(\lambda(i,\ zs).\ i < length\ zs)$
  $(\lambda(i,\ zs).\ do\ \{$
    $ASSERT\ (i < length\ zs);$
    $RETURN$
      $(i + 1,\ convert\text{-}single\text{-}wl\text{-}to\text{-}nat\text{-}conv\ xs\ i\ zs\ i)$
  $\})$
  $(0,\ zs);$
  RETURN zs
}›
unfolding *convert-wlists-to-nat-def*
op-map-def[*of* ‹*map* (λ(*n*, *y*, *y*′). (*nat-of-uint64-conv n*, *y*, *y*′))› ‹[]›,
unfolded convert-single-wl-to-nat-conv-alt-def[*symmetric*] *init-lrl-def*[*symmetric*]] *Let-def*
by *auto*

sepref-register *init-lrl*

abbreviation (**in** −) *watchers-assn* **where**
‹*watchers-assn* ≡ *arl-assn* (*watcher-assn*)›

abbreviation (**in** −) *watchlist-assn* **where**
‹*watchlist-assn* ≡ *arrayO-assn watchers-assn*›

abbreviation (**in** −) *watchers-fast-assn* **where**
‹*watchers-fast-assn* ≡ *arl64-assn* (*watcher-fast-assn*)›

abbreviation (**in** −) *watchlist-fast-assn* **where**
‹*watchlist-fast-assn* ≡ *arrayO-assn watchers-fast-assn*›

sepref-definition *convert-single-wl-to-nat-code*
is ‹*uncurry3 convert-single-wl-to-nat*›
:: ‹$[\lambda(((W,\ i),\ W'),\ j).\ i < length\ W \wedge j < length\ W']_a$
$(watchlist\text{-}fast\text{-}assn)^k *_a nat\text{-}assn^k *_a$
$(watchlist\text{-}assn)^d *_a nat\text{-}assn^k \rightarrow$
watchlist-assn›
supply [[*goals-limit=1*]] *length-aa64-hnr*[*sepref-fr-rules*] *nth-aa64-hnr*[*sepref-fr-rules*]
length-ll-def[*simp*]
unfolding *convert-single-wl-to-nat-def op-map-to-def nth-ll-def*[*symmetric*]
length-ll-def[*symmetric*]
by *sepref*

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*))
$\in [\lambda(((a,\ b),\ ba),\ bb).\ b < length\ a \wedge bb < length\ ba \wedge ba\ !\ bb = []]_a$
$(watchlist\text{-}fast\text{-}assn)^k *_a nat\text{-}assn^k *_a$
$(watchlist\text{-}assn)^d *_a nat\text{-}assn^k \rightarrow$
watchlist-assn›
using *convert-single-wl-to-nat-code.refine*[*FCOMP convert-single-wl-to-nat*[*unfolded convert-fref*]]
by *auto*

sepref-definition *convert-wlists-to-nat-code*
is ‹*convert-wlists-to-nat*›
:: ‹$watchlist\text{-}fast\text{-}assn^d \rightarrow_a watchlist\text{-}assn$›

supply *length-a-hnr*[*sepref-fr-rules*] [[*goals-limit=1*]] *arrayO-raa-empty-sz-init-lrl*[*sepref-fr-rules del*]
unfolding *convert-wlists-to-nat-alt-def2*
by *sepref*

lemma *convert-wlists-to-nat-conv-hnr*[*sepref-fr-rules*]:
‹(*convert-wlists-to-nat-code*, *RETURN* ∘ *convert-wlists-to-nat-conv*)
∈ (*watchlist-fast-assn*)^d $\rightarrow_a$ *watchlist-assn*›
using *convert-wlists-to-nat-code.refine*[*FCOMP convert-wlists-to-nat-convert-wlists-to-nat-conv*[*unfolded convert-fref*]]
by *simp*

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*⟩
:: ⟨[λS. *length*(*get-clauses-wl-heur* S) ≤ *uint64-max* ∧
length (*get-old-arena* S) ≤ *uint64-max*]$_a$
isasat-bounded-assn^d → *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-bounded-assn-def isasat-unbounded-assn-def isasat-fast-slow-def*
apply (*rewrite at* ⟨(-, ¤, -)⟩ *arl64-to-arl-conv-def*[*symmetric*])
apply (*rewrite at* ⟨(-, -, *nat-of-uint64-conv* -, -, ¤)⟩ *arl64-to-arl-conv-def*[*symmetric*])
apply (*rewrite at* ⟨(-, ¤, *nat-of-uint64-conv* -, -)⟩ *arl64-to-arl-conv-def*[*symmetric*])
apply (*rewrite at* ⟨(¤, -, *nat-of-uint64-conv* -, -)⟩ *arl64-to-arl-conv-def*[*symmetric*])
apply (*rewrite* **in** ⟨(-, ¤, *nat-of-uint64-conv* -, -)⟩ *arl-nat-of-uint64-conv-def*[*symmetric*])
apply (*rewrite* **in** ⟨(¤, -, *nat-of-uint64-conv* -, -)⟩ *arl-nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

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-unbounded-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *get-conflict-wl-is-None-heur-alt-def isasat-unbounded-assn-def length-ll-def*[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

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-bounded-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *get-conflict-wl-is-None-heur-alt-def isasat-bounded-assn-def length-ll-def*[*symmetric*]

supply [[*goals-limit=1*]]
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
supply [[*goals-limit=2*]]
unfolding *isa-count-decided-st-def isasat-unbounded-assn-def*
by *sepref*

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*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
supply [[*goals-limit=2*]]
unfolding *isa-count-decided-st-def isasat-bounded-assn-def*
by *sepref*

declare *isa-count-decided-st-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *polarity-st-heur-pol*
is ⟨*uncurry* (*RETURN oo polarity-st-heur*)⟩
:: ⟨$[polarity\text{-}st\text{-}heur\text{-}pre]_a\ isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$⟩
unfolding *polarity-st-heur-alt-def isasat-unbounded-assn-def polarity-st-pre-def polarity-st-heur-pre-def*
supply [[*goals-limit = 1*]]
by *sepref*

declare *polarity-st-heur-pol.refine*[*sepref-fr-rules*]

sepref-definition *polarity-st-heur-pol-fast*
is ⟨*uncurry* (*RETURN oo polarity-st-heur*)⟩
:: ⟨$[polarity\text{-}st\text{-}heur\text{-}pre]_a\ isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$⟩
unfolding *polarity-st-heur-alt-def isasat-bounded-assn-def polarity-st-pre-def polarity-st-heur-pre-def*
supply [[*goals-limit = 1*]]
by *sepref*

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*) ∈ $isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
unfolding *count-decided-st-heur-def isasat-bounded-assn-def isasat-unbounded-assn-def*
by (*sepref-to-hoare*; *sep-auto*)+

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-unbounded-assn^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *unat-lit-assn*⟩
supply *length-rll-def*[*simp*] [[*goals-limit=1*]]
unfolding *isasat-unbounded-assn-def access-lit-in-clauses-heur-alt-def*
fmap-rll-def[*symmetric*] *access-lit-in-clauses-heur-pre-def*
fmap-rll-u64-def[*symmetric*]
by *sepref*

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*⟩
supply *length-rll-def*[*simp*] [[*goals-limit=1*]] *arena-is-valid-clause-idx-le-uint64-max*[*intro*]
unfolding *isasat-bounded-assn-def access-lit-in-clauses-heur-alt-def*
fmap-rll-def[*symmetric*] *access-lit-in-clauses-heur-pre-def*
fmap-rll-u64-def[*symmetric*] *arena-lit-pre-le*[*intro*]
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-def Let-def two-uint64-nat-def*[*symmetric*] *PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]] *uint64-of-nat-conv-def*[*simp*]
arena-lit-pre-le-uint64-max[*intro*] *append-aa64-hnr*[*sepref-fr-rules*]
unfolding *rewatch-heur-alt-def Let-def two-uint64-nat-def*[*symmetric*] *PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *zero-uint64-nat-def*[*symmetric*]
one-uint64-nat-def[*symmetric*] *to-watcher-fast-def*[*symmetric*]
FOREACH-cond-def FOREACH-body-def uint64-of-nat-conv-def
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def*

isasat-unbounded-assn-def
by *sepref*

sepref-definition *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def rewatch-heur-st-fast-pre-def*
isasat-bounded-assn-def rewatch-heur-st-fast-def
by *sepref*

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 ‹$\mathcal{A}$ ≡ *all-atms-st S*›
assumes ‹*unit-prop-body-wl-D-inv S j w L*›
shows
‹*w* < *length* (*watched-by S L*)› **and**
‹*j* ≤ *w*› **and**
‹*fst* (*snd* (*watched-by-app S L w*)) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*)› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*) ≠ []› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *Suc 0* < *length* (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*))› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*) ! *0* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*) ! *Suc 0* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *fst* (*watched-by-app S L w*) > *0*› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *literals-are-*$\mathcal{L}_{in}$ $\mathcal{A}$ *S*› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *get-conflict-wl S* = *None*› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*)))› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *distinct* (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*))› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ (*get-trail-wl S*)› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹
length (*get-clauses-wl S* ∝ *fst* (*watched-by-app S L w*)) ≤ *uint64-max*› **and**
‹*fst* (*watched-by-app S L w*) ∈# *dom-m* (*get-clauses-wl S*) ⟹

$L \in$ *set* (*watched-l* (*get-clauses-wl* $S \propto$ *fst* (*watched-by-app* S L w)))›
proof −
let *?C* = ‹*fst* (*watched-by-app* S L w)›
show ‹$w <$ *length* (*watched-by* S L)› **and** ‹$j \leq w$›
using *assms* **unfolding** *unit-prop-body-wl-D-inv-def unit-prop-body-wl-inv-def watched-by-app-def unit-propagation-inner-loop-body-l-inv-def* **by** *fast+*
have ‹*blits-in-*$\mathcal{L}_{in}$ S› **and** ‹$L \in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$›
using *assms* **unfolding** *unit-prop-body-wl-D-inv-def unit-prop-body-wl-inv-def watched-by-app-def unit-propagation-inner-loop-body-l-inv-def literals-are-*$\mathcal{L}_{in}$*-def* **by** *fast+*
then show ‹*fst* (*snd* (*watched-by-app* S L w)) $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$›
using ‹$w <$ *length* (*watched-by* S L)› **and** ‹$j \leq w$› *nth-mem*[*of* ‹w› ‹*watched-by* S L›]
unfolding *blits-in-*$\mathcal{L}_{in}$*-def*
by (*fastforce simp*: *watched-by-app-def image-image* $\mathcal{A}$*-def dest*: *multi-member-split simp del*: *nth-mem*)
assume *C-dom*: ‹*fst* (*watched-by-app* S L w) $\in\#$ *dom-m* (*get-clauses-wl* S)›
obtain T T' **where**
lits: ‹*literals-are-*$\mathcal{L}_{in}$ $\mathcal{A}$ S› **and**
‹$L \in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
S-T: ‹(S, T) $\in$ *state-wl-l* (*Some* (L, w))› **and**
‹$L \in\#$ *all-lits-of-mm*
(*mset* ‘# *init-clss-lf* (*get-clauses-wl* S) + *get-unit-clauses-wl* S)› **and**
T-T': ‹(*set-clauses-to-update-l*
(*clauses-to-update-l* (*remove-one-lit-from-wq* (*?C*) T) +
{#*?C*#})
(*remove-one-lit-from-wq* (*?C*) T),
T')
$\in$ *twl-st-l* (*Some* L)› **and**
‹*correct-watching-except* j w L S› **and**
struct: ‹*twl-struct-invs* T'› **and**
‹$w <$ *length* (*watched-by* S L)› **and**
confl: ‹*get-conflict-wl* S = *None*› **and**
stgy: ‹*twl-stgy-invs* T'› **and**
‹*?C*
$\in\#$ *dom-m*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T))› **and**
watched-ge-0: ‹$0 <$ *?C*› **and**
‹$0 <$ *length*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T) $\propto$
(*?C*))› **and**
‹*no-dup* (*get-trail-l* (*remove-one-lit-from-wq* (*?C*) T))› **and**
i-le: ‹(*if get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T) $\propto$
(*?C*) !
0 =
L
then 0 *else* 1)
< *length*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T) $\propto$
(*?C*))› **and**
ui-le: ‹1 − (*if get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T) $\propto$
(*?C*) !
0 =
L
then 0 *else* 1)
< *length*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) T) $\propto$
(*?C*))› **and**

L-in-watched: ‹$L \in$ *set* (*watched-l*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) $\propto$
(*?C*)))› **and**
‹*get-conflict-l* (*remove-one-lit-from-wq* (*?C*) *T*) = *None*›
using *assms C-dom* **unfolding** *unit-prop-body-wl-D-inv-def unit-prop-body-wl-inv-alt-def*
watched-by-app-def unit-propagation-inner-loop-body-l-inv-def
apply − **apply** *normalize-goal+*
by *blast*
show *S-L-W-le-S*: ‹*?C* $\in\#$ *dom-m* (*get-clauses-wl S*)›
using *C-dom* **unfolding** *watched-by-app-def* **by** *auto*
show
‹*get-clauses-wl S* $\propto$ *?C* $\neq$ []› **and**
le: ‹*Suc 0* < *length* (*get-clauses-wl S* $\propto$ *?C*)›
using *ui-le i-le S-T*
unfolding *watched-by-app-def*
by (*auto simp*: *twl-st-wl*)
have *S-L-w-ge-0*: ‹*0* < *?C*›
using *watched-ge-0* **unfolding** *watched-by-app-def* **by** *auto*
obtain *M N D NE UE W Q* **where**
S: ‹*S* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)›
by (*cases S*)
show *lits-N*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*get-clauses-wl S* $\propto$ *?C*))›
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-nth*[*of*])
apply (*rule S-L-W-le-S*)
using *lits* **by** *auto*
then show ‹*get-clauses-wl S* $\propto$ *?C* ! *0* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$›
using *le* **apply** (*cases* ‹*get-clauses-wl S* $\propto$ *?C*›)
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def all-lits-of-m-add-mset*)

show ‹*get-clauses-wl S* $\propto$ *?C* ! *Suc 0* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$›
using *lits-N le* **apply** (*cases* ‹*get-clauses-wl S* $\propto$ *?C*›;
cases ‹*tl* (*get-clauses-wl S* $\propto$ *?C*)›;
cases ‹*tl* (*tl* (*get-clauses-wl S* $\propto$ *?C*))›)
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def all-lits-of-m-add-mset*)

show *S-L-W-ge-0*: ‹*?C* > *0*› **and**
‹*literals-are-*$\mathcal{L}_{in}$ $\mathcal{A}$ *S*› **and**
‹*get-conflict-wl S* = *None*›
using *confl watched-ge-0 lits* **unfolding** *watched-by-app-def*
by *fast+*
have *all-inv*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of T'*)›
using *struct* **unfolding** *twl-struct-invs-def* **by** *fast+*
then have
learned-tauto:
‹$\forall$ *s*$\in\#$*learned-clss* (*state*$_W$*-of T'*). ¬ *tautology s*› **and**
dist:
‹*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of T'*)›
using *struct* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def* **by** *fast+*
have ‹$\forall$ *D*$\in$*mset* ‘ *set-mset* (*ran-mf* (*get-clauses-wl S*)). *distinct-mset D*›
using *dist*
using *S-T T-T'*
unfolding *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def*
by (*auto simp*: *clauses-def twl-st-wl twl-st-l twl-st*
watched-by-app-def Ball-def Collect-conv-if
distinct-mset-set-def conj-disj-distribR Collect-disj-eq image-mset-union
dest!: *multi-member-split*

split: *if-splits*)
then show ‹*distinct* (*get-clauses-wl* $S \propto ?C$)›
using *S-L-W-le-S S-L-W-ge-0*
by (*auto simp*: $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def distinct-mset-set-distinct clauses-def mset-take-mset-drop-mset watched-by-app-def*)
show ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$›
using ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **.**
have *alien*: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of* T')›
using *struct* **unfolding** *twl-struct-invs-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast*
then have *N*: ‹*atm-of* ‘ *lits-of-l* (*trail* (*state*$_W$*-of* T')) ⊆ *atms-of-mm* (*init-clss* (*state*$_W$*-of* T'))›
unfolding $cdcl_W$*-restart-mset.no-strange-atm-def*
by (*cases S*)
(*auto simp*: $cdcl_W$*-restart-mset-state mset-take-mset-drop-mset*$'$)
then have *N*: ‹*atm-of* ‘ *lits-of-l* (*trail* (*state*$_W$*-of* T')) ⊆ *atms-of-mm* ($cdcl_W$*-restart-mset.clauses* (*state*$_W$*-of* T'))›
by (*auto simp*: $cdcl_W$*-restart-mset.clauses-def*)
then show ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ (*get-trail-wl S*)›
using *in-all-lits-of-m-ain-atms-of-iff S-T T-T*$'$ *lits*
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-trail-def in-all-lits-of-mm-ain-atms-of-iff image-subset-iff*
by (*auto simp*: *trail.simps in-all-lits-of-mm-ain-atms-of-iff lits-of-def image-image init-clss.simps mset-take-mset-drop-mset*$'$ *literals-are-*$\mathcal{L}_{in}$*-def convert-lits-l-def is-*$\mathcal{L}_{all}$*-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *twl-st-l twl-st-wl twl-st get-unit-clauses-wl-alt-def* $\mathcal{A}$*-def all-lits-def*)
show ‹*length* (*get-clauses-wl* $S \propto ?C$) ≤ *uint64-max*› **if** *bounded*: ‹*isasat-input-bounded* $\mathcal{A}$›
using *clss-size-uint64-max*[*of* $\mathcal{A}$ ‹*mset* (*get-clauses-wl* $S \propto ?C$)›,
OF bounded ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*get-clauses-wl* $S \propto ?C$))›]
‹*distinct* (*get-clauses-wl* $S \propto ?C$)› **by** *auto*
show *L-in-watched*: ‹$L \in$ *set* (*watched-l* (*get-clauses-wl* $S \propto ?C$))›
using *L-in-watched S-T* **by** *auto*
qed

definition (**in** −) *find-unwatched-wl-st* :: ‹*nat twl-st-wl* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*find-unwatched-wl-st* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) *i*. *do* {
find-unwatched-l M ($N \propto i$)
})›

lemma *find-unwatched-l-find-unwatched-wl-s*:
‹*find-unwatched-l* (*get-trail-wl S*) (*get-clauses-wl* $S \propto C$) = *find-unwatched-wl-st S C*›
by (*cases S*) (*auto simp*: *find-unwatched-wl-st-def*)

definition *find-non-false-literal-between* **where**
‹*find-non-false-literal-between M a b C* =
find-in-list-between (λ*L*. *polarity M L* ≠ *Some False*) *a b C*›

definition *isa-find-unwatched-between*
:: ‹- ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat option*) *nres*› **where**
‹*isa-find-unwatched-between P M*$'$ *NU a b C* = *do* {
ASSERT($C+a \le$ *length NU*);
ASSERT($C+b \le$ *length NU*);
(*x*, -) ← $WHILE_T{}^{\lambda(found,\ i).\ True}$
(λ(*found*, *i*). *found* = *None* ∧ $i < C + b$)
(λ(-, *i*). *do* {
ASSERT($i < C$ + *nat-of-uint64-conv* (*arena-length NU C*));

ASSERT($i \geq C$);
ASSERT($i < C + b$);
ASSERT(*arena-lit-pre NU i*);
ASSERT(*polarity-pol-pre M′* (*arena-lit NU i*));
if P (*arena-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-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) ∈ *trail-pol* $\mathcal{A}$›
assumes *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* ($N \propto C$))›
shows
‹*isa-find-unwatched-between P′ M′ arena a′ b′ C′* $\leq \Downarrow$ *Id* (*find-in-list-between P a b* ($N \propto C$))›
proof −
have [*refine0*]: ‹((*None*, *x2m* + *a*), *None*, *a*) ∈ ⟨*Id*⟩*option-rel* $\times_r$ {(n', n). $n' = x2m + n$}›
for *x2m*
by *auto*
have [*simp*]: ‹*arena-lit arena* ($C + x2$) $\in\# \mathcal{L}_{all}\ \mathcal{A}$› **if** ‹$x2 <$ *length* ($N \propto C$)› **for** *x2*
using *that lits assms* **by** (*auto simp*: *arena-lifting*
dest!: *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ - *x2*])
have *arena-lit-pre*: ‹*arena-lit-pre arena x2a*›
if
‹(x, x') ∈ ⟨*nat-rel*⟩*option-rel* $\times_f$ {(n', n). $n' = C + n$}› **and**
‹*case x of* (*found*, *i*) ⇒ *found* = *None* ∧ $i < C + b$› **and**
‹*case x′ of* (*found*, *i*) ⇒ *found* = *None* ∧ $i < b$› **and**
‹*case x of* (*found*, *i*) ⇒ *True*› **and**
‹*case x′ of*
(*found*, *i*) ⇒
$a \leq i$ ∧
$i \leq$ *length* ($N \propto C$) ∧
$i \leq b$ ∧
($\forall j \in \{a..<i\}$. ¬ *P* ($N \propto C$! *j*)) ∧
(∀ *j*. *found* = *Some j* ⟶ *i* = *j* ∧ *P* ($N \propto C$! *j*) ∧ $j < b$ ∧ $a \leq j$)› **and**
‹x' = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹$x2 <$ *length* ($N \propto C$)› **and**
‹$x2a < C$ + *nat-of-uint64-conv* (*arena-length arena C*)› **and**
‹$C \leq x2a$›
for *x x′ x1 x2 x1a x2a*
proof −
show *?thesis*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
apply (*rule bex-leI*[*of* - *C*])
apply (*rule exI*[*of* - *N*])
apply (*rule exI*[*of* - *vdom*])
using *assms that* **by** *auto*
qed

show *?thesis*

unfolding *isa-find-unwatched-between-def find-in-list-between-def eq*
apply *refine-vcg*
subgoal using *assms* **by** (*auto dest*!: *arena-lifting*(*10*))
subgoal using *assms* **by** (*auto dest*!: *arena-lifting*(*10*))
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by (*rule arena-lit-pre*)
subgoal
by (*rule polarity-pol-pre*[*OF M′M*]) *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *isa-find-non-false-literal-between* **where**
‹*isa-find-non-false-literal-between M arena a b C* =
isa-find-unwatched-between (λL. *polarity-pol M L* ≠ *Some False*) *M arena a b C*›

definition *find-unwatched*
:: ‹(*nat literal* ⇒ *bool*) ⇒ *nat clause-l* ⇒ (*nat option*) *nres*› **where**
‹*find-unwatched M C* = *do* {
b ← *SPEC*(λb::*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 ← *SPEC* (λi. $i \leq$ *length C* ∧ $i \geq 2$);
n ← *find-in-list-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* =
(λ(*S*, *i*). *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *i*)›

definition *find-unwatched-wl-st′*
:: ‹*nat twl-st-wl* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*find-unwatched-wl-st′* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ) *i*. *do* {
find-unwatched (λL. *polarity M L* ≠ *Some False*) (*N* ∝ *i*)
})›

definition *isa-find-unwatched*
:: ‹(*nat literal* ⇒ *bool*) ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ (*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 {
  $ASSERT(get\text{-}saved\text{-}pos\text{-}pre\ arena\ C)$;
  $pos \leftarrow RETURN\ (nat\text{-}of\text{-}uint64\text{-}conv\ (arena\text{-}pos\ arena\ C))$;
  $n \leftarrow isa\text{-}find\text{-}unwatched\text{-}between\ P\ M'\ arena\ pos\ l\ C$;
  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: ⟨$2 \leq length\ (N \propto C)$⟩ **and**
  C: ⟨$C \in\#\ dom\text{-}m\ N$⟩ **and**
  M′M: ⟨$(M', M) \in trail\text{-}pol\ \mathcal{A}$⟩
 shows ⟨$isa\text{-}find\text{-}unwatched\ P\ M'\ arena\ C \leq \Downarrow Id\ (find\text{-}unwatched\ P\ (N \propto C))$⟩
proof −
 have [*refine0*]:
  ⟨$RETURN(arena\text{-}length\ arena\ C \leq MAX\text{-}LENGTH\text{-}SHORT\text{-}CLAUSE) \leq$
   $\Downarrow \{(b,b').\ b = b' \wedge (b \longleftrightarrow is\text{-}short\text{-}clause\ (N{\propto}C))\}$
    $(SPEC\ (\lambda\text{-}.\ True))$⟩
  using *assms*
  by (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def arena-lifting*)
 show *?thesis*
  unfolding *isa-find-unwatched-def find-unwatched-def nat-of-uint64-conv-def Let-def*
  apply (*refine-vcg isa-find-unwatched-between-find-in-list-between-spec*[*of - - - - - vdom - - -* $\mathcal{A}$ *- -*])
  subgoal by *auto*
  subgoal using *ge2* **.**
  subgoal by *auto*
  subgoal using *ge2* **.**
  subgoal using *valid* **.**
  subgoal using *C* **.**
  subgoal using *assms* **by** *auto*
  subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
  subgoal using *assms* **by** *auto*
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  apply (*rule M′M*)
  subgoal using *assms* **by** *auto*
  subgoal using *assms* **unfolding** *get-saved-pos-pre-def arena-is-valid-clause-idx-def*
   by (*auto simp*: *arena-lifting*)
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  subgoal by (*auto simp*: *arena-pos-def*)
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  subgoal using *assms* **by** *auto*
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  subgoal using *assms* **by** *auto*
  subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
  subgoal using *assms* **by** *auto*
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  subgoal using *assms* **by** *auto*
  subgoal using *assms arena-lifting*[*OF valid C*] **by** *auto*
  apply (*rule M′M*)
  subgoal using *assms* **by** *auto*

```
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal using assms arena-lifting[OF valid C] by auto
    subgoal by (auto simp: arena-pos-def)
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal using assms by auto
    apply (rule M'M)
    subgoal using assms by auto
    done
qed

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 (λL. 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-ℒin 𝒜 (mset C)›
  shows ‹find-unwatched (λL. polarity M L ≠ Some False) C ≤ ⇓ Id (find-unwatched-l M C)›
proof –
  have [refine0]: ‹find-in-list-between (λL. polarity M L ≠ Some False) 2 (length C) C
      ≤ SPEC
        (λfound.
            (found = None) = (∀L∈set (unwatched-l C). – L ∈ lits-of-l M) ∧
            (∀j. found = Some j ⟶
                  j < length C ∧
                  (undefined-lit M (C ! j) ∨ C ! j ∈ lits-of-l M) ∧ 2 ≤ j))›
  proof –
    show ?thesis
      apply (rule order-trans)
      apply (rule find-in-list-between-spec)
      subgoal using assms by auto
      subgoal using assms by auto
      subgoal using assms by auto
      subgoal
        using n-d
        by (auto simp add: polarity-def in-set-drop-conv-nth Ball-def
          Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD)
      done
  qed
  have [refine0]: ‹find-in-list-between (λL. polarity M L ≠ Some False) xa (length C) C
      ≤ SPEC
        (λn. (if n = None
              then find-in-list-between (λL. polarity M L ≠ Some False) 2 xa C
              else RETURN n)
              ≤ SPEC
                (λfound.
                    (found = None) =
                    (∀L∈set (unwatched-l C). – L ∈ lits-of-l M) ∧
```

$(\forall j.\ found = Some\ j \longrightarrow$
$j < length\ C \wedge$
$(undefined\text{-}lit\ M\ (C\ !\ j) \vee C\ !\ j \in lits\text{-}of\text{-}l\ M) \wedge$
$2 \leq j)))$›

if
‹$xa \leq length\ C \wedge 2 \leq xa$›
for *xa*
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule find-in-list-between-spec*)
subgoal using *that* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *that* **by** *auto*
subgoal
apply (*rule SPEC-rule*)
subgoal for *x*
apply (*cases* ‹*x = None*›; *simp only*: *if-True if-False refl*)
subgoal
apply (*rule order-trans*)
apply (*rule find-in-list-between-spec*)
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal
apply (*rule SPEC-rule*)
apply (*intro impI conjI iffI ballI*)
unfolding *in-set-drop-conv-nth Ball-def*
apply *normalize-goal*
subgoal for *x L xaa*
apply (*cases* ‹*xaa ≥ xa*›)
subgoal
using *n-d*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
subgoal
using *n-d*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
done
subgoal for *x*
using *n-d that assms*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*,
force)
(*metis diff-is-0-eq' le-neq-implies-less le-trans less-imp-le-nat*
no-dup-consistentD zero-less-diff)
subgoal
using *n-d assms that*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l
split: *if-splits dest*: *no-dup-consistentD*)
done
done
subgoal
using *n-d that assms le-trans*

by (*auto simp add*: *polarity-def Ball-def all-conj-distrib in-set-drop-conv-nth Decided-Propagated-in-iff-in-lits-of-l split*: *if-splits dest*: *no-dup-consistentD*)
(*use le-trans no-dup-consistentD* **in** *blast*)+
done
done
done
qed

show *?thesis*
unfolding *find-unwatched-def find-unwatched-l-def*
apply (*refine-vcg*)
subgoal by *blast*
subgoal by *blast*
done
qed

definition *find-unwatched-wl-st-pre* **where**
⟨*find-unwatched-wl-st-pre* = $(\lambda(S, i).$
$i \in\#$ *dom-m* (*get-clauses-wl S*) $\wedge$
literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) $S \wedge 2 \leq$ *length* (*get-clauses-wl S* $\propto i$) $\wedge$
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* $\propto i$))
)⟩

theorem *find-unwatched-wl-st-heur-find-unwatched-wl-s*:
⟨(*uncurry isa-find-unwatched-wl-st-heur*, *uncurry find-unwatched-wl-st'*)
$\in$ [*find-unwatched-wl-st-pre*]$_f$
twl-st-heur $\times_f$ *nat-rel* $\rightarrow \langle Id\rangle$*nres-rel*⟩
proof −
have [*refine0*]: ⟨((*None*, *x2m* + *2*), *None*, *2*) $\in \langle Id\rangle$*option-rel* $\times_r$ $\{(n', n).\ n' = x2m + n\}$⟩
for *x2m*
by *auto*
have [*refine0*]:
⟨(*polarity M* (*arena-lit arena i'*), *polarity M'* ($N \propto C'$! *j*)) $\in \langle Id\rangle$*option-rel*⟩
if ⟨∃ *vdom*. *valid-arena arena N vdom*⟩ **and**
⟨$C' \in\#$ *dom-m N*⟩ **and**
⟨$i' = C' + j \wedge j <$ *length* ($N \propto C'$)⟩ **and**
⟨$M = M'$⟩
for *M arena i i' N j M' C'*
using *that* **by** (*auto simp*: *arena-lifting*)
have [*refine0*]: ⟨*RETURN* (*arena-pos arena C*) $\leq \Downarrow$ {(*pos*, *pos'*). *pos* = *pos'* $\wedge$ *pos* $\geq 2 \wedge$ *pos* $\leq$ *length* ($N \propto C$)}
(*SPEC* ($\lambda i.\ i \leq$ *length* ($N \propto C'$) $\wedge\ 2 \leq i$))⟩
if *valid*: ⟨*valid-arena arena N vdom*⟩ **and** *C*: ⟨$C \in\#$ *dom-m N*⟩ **and** ⟨$C = C'$⟩ **and**
⟨*is-long-clause* ($N \propto C'$)⟩
for *arena N vdom C C'*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff arena-pos-def*)
have [*refine0*]:
⟨*RETURN* (*arena-length arena C* $\leq$ *MAX-LENGTH-SHORT-CLAUSE*) $\leq \Downarrow$ {(*b*, *b'*). *b* = *b'* $\wedge$ (*b* $\longleftrightarrow$ *is-short-clause* ($N \propto C$))}
(*SPEC*(λ-. *True*))⟩
if *valid*: ⟨*valid-arena arena N vdom*⟩ **and** *C*: ⟨$C \in\#$ *dom-m N*⟩
for *arena N vdom C*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def*)

have *H*: ⟨*isa-find-unwatched P M' arena C* $\leq \Downarrow$ *Id* (*find-unwatched P'* ($N \propto C'$))⟩

if ⟨*valid-arena arena N vdom*⟩ **and** ⟨$C \in\# \ dom\text{-}m \ N$⟩ **and**
⟨$\bigwedge L. \ L \in\# \ \mathcal{L}_{all} \ \mathcal{A} \Longrightarrow P \ L = P' \ L$⟩ **and**
⟨$C = C'$⟩ **and**
⟨$2 \leq length \ (N \propto C')$⟩ **and** ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* $(N \propto C')$)⟩ **and**
⟨$(M', M) \in trail\text{-}pol \ \mathcal{A}$⟩
for *arena P N C vdom P' C' $\mathcal{A}$ M' M*
unfolding *isa-find-unwatched-def find-unwatched-def nat-of-uint64-conv-def Let-def*
apply (*refine-vcg isa-find-unwatched-between-find-in-list-between-spec*[*of - - - - - vdom*])
using *that* **apply** − **apply** *assumption*
using *that* **apply** − **apply** *assumption*
subgoal by *auto*
subgoal using *that* **by** (*simp add*: *arena-lifting*)
subgoal using *that* **by** *auto*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
apply *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting get-saved-pos-pre-def*
arena-is-valid-clause-idx-def)
using *that* **apply** − **apply** *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
apply *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
apply *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
done

show *?thesis*
unfolding *isa-find-unwatched-wl-st-heur-def find-unwatched-wl-st'-def*
uncurry-def twl-st-heur-def

find-unwatched-wl-st-pre-def
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal for *x y*
apply (*case-tac x*, *case-tac y*)
by (*rule H*[**where** $\mathcal{A}2$ = ‹*all-atms-st* (*fst y*)›, *of* - - ‹*set* (*get-vdom* (*fst x*))›])
(*auto simp*: *polarity-pol-polarity*[*of* ‹*all-atms-st* (*fst y*)›,
unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*]
all-atms-def[*symmetric*])
done
qed

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*) ∈ *twl-st-heur*›
‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
‹*is-long-clause* (*get-clauses-wl T* ∝ *C*)› **and**
‹*i* ≤ *length* (*get-clauses-wl T* ∝ *C*)› **and**
‹*i* ≥ *2*›
shows ‹*isa-save-pos C i S* ≤ ⇓ *twl-st-heur* (*RETURN T*)›
proof −
have ‹*isa-update-pos-pre* ((*C*, *i*), *get-clauses-wl-heur S*)›
unfolding *isa-update-pos-pre-def*
using *assms*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting)
then show *?thesis*
using *assms*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting
intro!: *valid-arena-update-pos*)
qed

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'* =
(λ*C* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). (*M*, *N*, *Some* (*mset* (*N*∝*C*)), *NE*, *UE*, {#}, *W*))›

lemma *set-conflict-wl′-alt-def*:
⟨*set-conflict-wl′ i S = set-conflict-wl (get-clauses-wl S ∝ i) S*⟩
by (*cases S*) (*auto simp*: *set-conflict-wl′-def set-conflict-wl-def*)

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 ≤ 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-clause-wl-heur = (λ(L::nat literal) C b j w i f (M, N, D, Q, W, vm). do {*
  ASSERT(arena-lit-pre N (C+f));
  let K′ = arena-lit N (C + f);
  ASSERT(swap-lits-pre C i f N);
  ASSERT(w < length N);
  let N′ = swap-lits C i f N;
  ASSERT(length (W ! nat-of-lit K′) < length N);
  let W = W[nat-of-lit K′:= W ! (nat-of-lit K′) @ [to-watcher C L b]];
  RETURN (j, w+1, (M, N′, D, Q, W, vm))
})⟩

definition *update-clause-wl-pre* **where**
⟨*update-clause-wl-pre K r = (λ(((((((L, C), b), j), w), i), f), S). C ∈# dom-m(get-clauses-wl S) ∧*
  L∈# $\mathcal{L}_{all}$ (all-atms-st S) ∧ i < length (get-clauses-wl S ∝ C) ∧
  f < length (get-clauses-wl S ∝ C) ∧
  L ≠ get-clauses-wl S ∝ C ! f ∧
  length (watched-by S (get-clauses-wl S ∝ C ! f)) < r ∧
  w < r ∧
  L = K)⟩

lemma *update-clause-wl-pre-alt-def*:
⟨*update-clause-wl-pre K r = (λ(((((((L, C), b), j), w), i), f), S). C ∈# dom-m(get-clauses-wl S) ∧*

$L \in\# \mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$ $i <$ *length* (*get-clauses-wl S* $\propto$ *C*) $\wedge$
$f <$ *length* (*get-clauses-wl S* $\propto$ *C*) $\wedge$
$L \neq$ *get-clauses-wl S* $\propto$ *C* ! *f* $\wedge$
length (*watched-by S* (*get-clauses-wl S* $\propto$ *C* ! *f*)) $< r \wedge$
$w < r \wedge$
get-clauses-wl S $\propto$ *C* ! *f* $\in\# \mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$
$L = K$)›
by (*auto intro*!: *ext simp*: *update-clause-wl-pre-def all-atms-def all-lits-def*
in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ ran-m-def
dest!: *multi-member-split*[*of* - ‹*dom-m* -›] *simp*: *all-lits-of-mm-add-mset atm-of-all-lits-of-m image-Un*
simp del: *all-atms-def*[*symmetric*])

lemma *arena-lit-pre*:
‹*valid-arena NU N vdom* $\Longrightarrow$ $C \in\#$ *dom-m N* $\Longrightarrow$ $i <$ *length* ($N \propto C$) $\Longrightarrow$ *arena-lit-pre NU* (C + i)›
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of* - *C*], *rule exI*[*of* - *N*], *rule exI*[*of* - *vdom*]) *auto*

lemma *all-atms-swap*[*simp*]:
‹$C \in\#$ *dom-m N* $\Longrightarrow$ $i <$ *length* ($N \propto C$) $\Longrightarrow$ $j <$ *length* ($N \propto C$) $\Longrightarrow$
all-atms (N($C \hookrightarrow$ *swap* ($N \propto C$) *i j*)) = *all-atms N*›
by (*auto simp flip*: *all-atms-def*[*symmetric*])

lemma *update-clause-wl-heur-update-clause-wl*:
‹(*uncurry7 update-clause-wl-heur*, *uncurry7* (*update-clause-wl*)) $\in$
[*update-clause-wl-pre K r*]$_f$
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* $\rightarrow$
$\langle$*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*$\rangle$*nres-rel*›
unfolding *update-clause-wl-heur-def update-clause-wl-def uncurry-def Let-def*
update-clause-wl-pre-alt-def
apply (*intro frefI nres-relI*)
apply *clarify*
apply *refine-rcg*
subgoal
by (*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *arena-lit-pre*)
subgoal
by (*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *bex-leI exI*)
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*)
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-alt-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def

intro!: *ASSERT-refine-left valid-arena-swap-lits*)
done

definition (**in** −) *access-lit-in-clauses* **where**
‹*access-lit-in-clauses S i j* = (*get-clauses-wl S*) ∝ *i* ! *j*›

lemma *twl-st-heur-get-clauses-access-lit*[*simp*]:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *C* ∈# *dom-m* (*get-clauses-wl T*) ⟹
i < *length* (*get-clauses-wl T* ∝ *C*) ⟹
get-clauses-wl T ∝ *C* ! *i* = *access-lit-in-clauses-heur S C i*›
for *S T C i*
by (*cases S*; *cases T*)
(*auto simp*: *arena-lifting twl-st-heur-def access-lit-in-clauses-heur-def*)

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 −
obtain *M N D NE UE WS Q* **where**
S: ‹*S* = (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*)›
by (*cases S*)
let *?C* = ‹*fst* (*watched-by S L* ! *C*)›
let *?𝒜* = ‹*all-atms-st S*›
obtain *T T′* **where**
‹*literals-are-*$\mathcal{L}_{in}$ *?𝒜 S*› **and**
‹*L* ∈# $\mathcal{L}_{all}$ *?𝒜*› **and**
S-T: ‹(*S*, *T*) ∈ *state-wl-l* (*Some* (*L*, *C*))› **and**
‹*L* ∈# *all-lits-of-mm*
(*mset* ‘# *init-clss-lf* (*get-clauses-wl S*) + *get-unit-clauses-wl S*)› **and**
T-T′: ‹(*set-clauses-to-update-l*
(*clauses-to-update-l* (*remove-one-lit-from-wq* (*?C*) *T*) +
{#*?C*#})
(*remove-one-lit-from-wq* (*?C*) *T*),
T′)
∈ *twl-st-l* (*Some L*)› **and**
inv: ‹*twl-struct-invs T′*› **and**
C-le: ‹*C* < *length* (*watched-by S L*)› **and**
confl: ‹*get-conflict-wl S* = *None*› **and**
stgy-invs: ‹*twl-stgy-invs T′*› **and**
‹*?C* ∈# *dom-m*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*))› **and**
‹*0* < *?C*› **and**
‹*0* < *length*
(*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
(*?C*))› **and**
‹*no-dup* (*get-trail-l* (*remove-one-lit-from-wq* (*?C*) *T*))› **and**

i-le: ‹(*if get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
    (*?C*) !
    0 =
    L
  then 0 else 1)
 < *length*
    (*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
    (*?C*))› **and**
ui-le: ‹*1* − (*if get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
      (*?C*) !
      0 =
      L
    then 0 else 1)
 < *length*
    (*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
    (*?C*))› **and**
‹*L* ∈ *set* (*watched-l*
      (*get-clauses-l* (*remove-one-lit-from-wq* (*?C*) *T*) ∝
      (*?C*)))› **and**
‹*get-conflict-l* (*remove-one-lit-from-wq* (*?C*) *T*) = *None*›
using *inv dom* **unfolding** *unit-prop-body-wl-D-inv-def unit-prop-body-wl-inv-alt-def*
unit-propagation-inner-loop-body-l-inv-def watched-by-app-def
apply (*simp only*: *dom simp-thms*)
apply *normalize-goal+*
by *blast*

have *L-WS*: ‹(*L*, *twl-clause-of* (*get-clauses-wl S* ∝ *?C*)) ∈# *clauses-to-update T′*›
  using *S-T T-T′ confl* **by** (*cases T′*) (*auto simp*: *twl-st-l-def state-wl-l-def S*)
have ‹*consistent-interp* (*lits-of-l* (*trail* (*state*$_W$*-of T′*)))› **and**
  n-d: ‹*no-dup* (*trail* (*state*$_W$*-of T′*))› **and**
  valid: ‹*valid-enqueued T′*› **and**
  n-d-q: ‹*no-duplicate-queued T′*›
  using *inv* **unfolding** *unit-prop-body-wl-D-inv-def unit-prop-body-wl-inv-def twl-struct-invs-def*
    cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
  by *blast+*
then have *cons*: ‹*consistent-interp* (*lits-of-l* (*get-trail-wl S*))›
  using *S-T T-T′* **by** (*auto simp*: *twl-st-l twl-st twl-st-wl*)

have ‹∀ *L*∈#*mset* (*unwatched-l* (*get-clauses-wl S* ∝ (*?C*))).
    − *L* ∈ *lits-of-l* (*get-trail-wl S*)›
  using *find-unw* **unfolding** *unit-prop-body-wl-D-find-unwatched-inv-def*
    unit-prop-body-wl-find-unwatched-inv-def watched-by-app-def
  by *auto*
moreover {
  have ‹*add-mset L* (*literals-to-update T′*) ⊆#
    {#− *lit-of x*. *x* ∈# *mset* (*get-trail T′*)#}›
   using *n-d-q S-T T-T′ L-WS*
   by (*cases* ‹*clauses-to-update T′*›)
    (*auto simp add*: *no-duplicate-queued-alt-def twl-st-wl twl-st-l twl-st*)
  note *mset-subset-eq-insertD*[*OF this*]
  moreover have ‹*xa* ∈ *set x* ⟹
    (*M*, *x*) ∈ *convert-lits-l N* (*NE* + *UE*) ⟹
    lit-of xa ∈ *lit-of* ‘ *set M*› **for** *xa x*
   using *imageI*[*of xa* ‹*set x*› *lit-of*]
   by (*auto simp*: *twl-st-wl twl-st-l twl-st S state-wl-l-def twl-st-l-def lits-of-def*
     dest: *imageI*[*of -* ‹*set -*› ‹*lit-of*›])

ultimately have ‹$-$ $L \in$ *lits-of-l M*›
using *S-T T-T'*
by (*auto simp*: *twl-st-wl twl-st-l twl-st S state-wl-l-def twl-st-l-def lits-of-def* *dest*: *imageI*[*of* - ‹*set* -› ‹*lit-of*›])
}
moreover have ‹$-$ *get-clauses-wl S* $\propto$ *?C* ! *?i* $\in$ *lits-of-l M*›
using *val* **by** (*auto simp*: *S polarity-st-def watched-by-app-def polarity-def access-lit-in-clauses-def Decided-Propagated-in-iff-in-lits-of-l split*: *if-splits*)
moreover have *length-C*: ‹*length* (*get-clauses-wl S* $\propto$ *?C*) $\geq$ *2*›
using *i-le ui-le S-T T-T'*
by (*auto simp*: *watched-by-app-def twl-st-wl twl-st-l twl-st S*)
moreover {
have *QL*: ‹*Q L* ! *C* = *hd* (*drop C* (*Q L*))›
using *confl C-le length-C* **by** (*auto simp*: *S hd-drop-conv-nth split*:)
have ‹*L* $\in\#$ *mset* (*watched-l* (*get-clauses-wl S* $\propto$ *?C*))›
using *valid confl C-le length-C S-T T-T'* **by** (*auto simp*: *QL S take-2-if watched-by-app-def twl-st-wl twl-st-l twl-st S*)
then have ‹*N* $\propto$ (*fst* (*Q L* ! *C*)) ! *0* = *L* $\vee$ *N* $\propto$ (*fst* (*Q L* ! *C*)) ! *1* = *L*›
using *confl C-le length-C* **by** (*auto simp*: *S take-2-if watched-by-app-def QL split*: *if-splits*)
}
ultimately have *Not*: ‹*lits-of-l* (*get-trail-wl S*) $\models$*s CNot ?D*›
unfolding *true-clss-def-iff-negation-in-model polarity-def polarity-st-def*
unfolding *mset-append watched-by-app-def access-lit-in-clauses-def*
by (*subst* (*1*) *append-take-drop-id*[*symmetric*, *of* - *2*])
(*auto simp*: *S take-2-if hd-conv-nth uminus-lit-swap twl-st-wl twl-st-l twl-st S split*: *if-splits*)
show *?thesis*
using *consistent-CNot-not-tautology*[*OF cons Not*] **.**
qed

definition *propagate-lit-wl-heur-pre* **where**
‹*propagate-lit-wl-heur-pre* =
(λ(((*L*, *C*), *i*), *S*). *i* $\leq$ *1* $\wedge$ *C* $\neq$ *DECISION-REASON*)›

definition *propagate-lit-wl-heur*
:: ‹*nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *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* $\wedge$ *get-conflict-wl S* = *None* $\wedge$
C $\in\#$ *dom-m* (*get-clauses-wl S*) $\wedge$ *L* $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$
1 $-$ *i* < *length* (*get-clauses-wl S* $\propto$ *C*) $\wedge$
0 < *length* (*get-clauses-wl S* $\propto$ *C*))›

lemma *isa-vmtf-consD*:
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf* $\mathcal{A}$ *M*›
shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf* $\mathcal{A}$ (*L* # *M*)›
using *vmtf-consD*[*of ns m fst-As lst-As next-search* - $\mathcal{A}$ *M L*] *assms*
by (*auto simp*: *isa-vmtf-def*)

lemma *propagate-lit-wl-heur-propagate-lit-wl*:
‹(*uncurry3 propagate-lit-wl-heur*, *uncurry3* (*RETURN oooo propagate-lit-wl*)) ∈
$[propagate\text{-}lit\text{-}wl\text{-}pre]_f$
$Id \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}up'' \mathcal{D}\ r\ s\ K \rightarrow \langle twl\text{-}st\text{-}heur\text{-}up'' \mathcal{D}\ r\ s\ K\rangle nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)

definition *propagate-lit-wl-bin-pre* **where**
‹*propagate-lit-wl-bin-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*))›

definition *propagate-lit-wl-bin-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-bin-heur* = (λ*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*)) ∈
$[propagate\text{-}lit\text{-}wl\text{-}bin\text{-}pre]_f$
$Id \times_f nat\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}up'' \mathcal{D}\ r\ s\ K \rightarrow \langle twl\text{-}st\text{-}heur\text{-}up'' \mathcal{D}\ r\ s\ K\rangle nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-bin-heur-def propagate-lit-wl-bin-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-bin-pre-def swap-lits-pre-def
arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)

lemma *undefined-lit-polarity-st-iff*:
‹*undefined-lit* (*get-trail-wl S*) *L* ⟷
polarity-st S L ≠ *Some True* ∧ *polarity-st S L* ≠ *Some False*›
by (*auto simp*: *polarity-st-def polarity-def*)

lemma *find-unwatched-le-length*:
‹*xj* < *length* (*get-clauses-wl S* ∝ *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*
using *that* **unfolding** *find-unwatched-wl-st-def IsaSAT-Inner-Propagation.find-unwatched-wl-st-def*
find-unwatched-l-def
by (*cases S*) *auto*

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 −
let *?C*= ‹*fst* (*watched-by-app S L C*)›
have ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S*›
using *inv dom* **by** (*blast intro*: *unit-prop-body-wl-D-invD*)
moreover {
have *xj*: ‹*xj* < *length* (*get-clauses-wl S* $\propto$ *?C*)›
using *find-unw* **by** (*cases S*) (*auto simp*: *IsaSAT-Inner-Propagation.find-unwatched-wl-st-def*
find-unwatched-l-def)
have ‹*?C* > *0*›
using *inv dom* **by** (*blast intro*: *unit-prop-body-wl-D-invD*)+
then have ‹*get-clauses-wl S* $\propto$ *?C* ! *xj* $\in\#$
all-lits-of-mm (*mset* ‘# *ran-mf* (*get-clauses-wl S*))›
using *xj dom*
by (*cases S*)
(*auto simp*: *clauses-def watched-by-app-def mset-take-mset-drop-mset*
in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def nth-in-set-tl
intro!: *bexI*[*of* - ‹*the* (*fmlookup* (*get-clauses-wl S*)(*?C*))›])
then have ‹*get-clauses-wl S* $\propto$ *?C* ! *xj* $\in\#$
all-lits-of-mm (*mset* ‘# *ran-mf* (*get-clauses-wl S*))›
unfolding *mset-append*
by (*cases S*)
(*auto simp*: *clauses-def watched-by-app-def mset-take-mset-drop-mset'*
all-lits-of-mm-union drop-Suc) **}**
ultimately show *?thesis*
unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-*$\mathcal{L}_{in}$*-def all-lits-def*
by (*auto simp*: *all-lits-of-mm-union*)
qed

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′* =
(λ*C* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *do* {
let D = *Some* (*mset* (*N* ∝ *C*));
RETURN (*M*, *N*, *D*, *NE*, *UE*, {#}, *W*) })
›
unfolding *set-conflict-wl′-def*
by (*auto intro*!: *ext*)

declare *RETURN-as-SPEC-refine*[*refine2 del*]

definition *set-conflict-wl′-pre* **where**
⟨*set-conflict-wl′-pre i S* $\longleftrightarrow$
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 −
have *H*:
⟨*isa-set-lookup-conflict-aa x y z a b c d*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_f$ (*nat-rel* $\times_f$ (*Id* $\times_f$ *Id*)))
(*set-conflict-m x′ y′ z′ a′ b′ c′ d′*)⟩
if
⟨(((((((*x*, *y*), *z*), *a*), *b*), *c*), *d*), (((((((*x′*, *y′*), *z′*), *a′*), *b′*), *c′*), *d′*)
∈ *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⟩ **and**
⟨*z′* ∈# *dom-m y′* ∧ *a′* = *None* ∧ *distinct* (*y′* ∝ *z′*) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf y′*) ∧
¬ *tautology* (*mset* (*y′* ∝ *z′*)) ∧ *b′* = *0* ∧ *out-learned x′ None d′* ∧
isasat-input-bounded $\mathcal{A}$⟩
for *x x′ y y′ z z′ a a′ b b′ c c′ d d′ vdom* $\mathcal{A}$
by (*rule isa-set-lookup-conflict*[*THEN fref-to-Down-curry6*,
unfolded prod.case, *OF that*(*2*,*1*)])
have [*refine0*]: ⟨*isa-set-lookup-conflict-aa x1h x1i x1g x1j zero-uint32-nat x1q x1r*
≤ ⇓ {((*C*, *n*, *lbd*, *outl*), *D*). (*C*, *D*) ∈ *option-lookup-clause-rel* (*all-atms-st x2*) ∧
n = *card-max-lvl x1a* (*the D*) ∧ *out-learned x1a D outl*}
(*RETURN* (*Some* (*mset* (*x1b* ∝ *x1*))))⟩
if
⟨(*x*, *y*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩ **and**
⟨*x2e* = (*x1f*, *x2f*)⟩ **and**
⟨*x2d* = (*x1e*, *x2e*)⟩ **and**
⟨*x2c* = (*x1d*, *x2d*)⟩ **and**
⟨*x2b* = (*x1c*, *x2c*)⟩ **and**
⟨*x2a* = (*x1b*, *x2b*)⟩ **and**
⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*y* = (*x1*, *x2*)⟩ **and**
⟨*x2s* = (*x1t*, *x2t*)⟩ **and**
⟨*x2r* = (*x1s*, *x2s*)⟩ **and**
⟨*x2q* = (*x1r*, *x2r*)⟩ **and**
⟨*x2p* = (*x1q*, *x2q*)⟩ **and**
⟨*x2o* = (*x1p*, *x2p*)⟩ **and**
⟨*x2n* = (*x1o*, *x2o*)⟩ **and**
⟨*x2m* = (*x1n*, *x2n*)⟩ **and**
⟨*x2l* = (*x1m*, *x2m*)⟩ **and**
⟨*x2k* = (*x1l*, *x2l*)⟩ **and**
⟨*x2j* = (*x1k*, *x2k*)⟩ **and**

⟨*x2i* = (*x1j*, *x2j*)⟩ **and**
⟨*x2h* = (*x1i*, *x2i*)⟩ **and**
⟨*x2g* = (*x1h*, *x2h*)⟩ **and**
⟨*x* = (*x1g*, *x2g*)⟩ **and**
⟨*case y of* (*x*, *xa*) ⇒ *set-conflict-wl′-pre x xa*⟩
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t*
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule H*[*of - - - - - - - x1a x1b x1g x1c zero-uint32-nat x1q x1r* ⟨*all-atms-st x2*⟩ ⟨*set* (*get-vdom* (*snd x*))⟩])
subgoal
using *that*
by (*auto simp*: *twl-st-heur′-def twl-st-heur-def*)
subgoal
using *that*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur′-def twl-st-heur-def set-conflict-wl′-pre-def*)
subgoal
using *that*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur′-def twl-st-heur-def*)
done
qed
have *isa-set-lookup-conflict-aa-pre*:
⟨*curry6 isa-set-lookup-conflict-aa-pre x1h x1i x1g x1j zero-uint32-nat x1q x1r*⟩
if
⟨*case y of* (*x*, *xa*) ⇒ *set-conflict-wl′-pre x xa*⟩ **and**
⟨(*x*, *y*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩ **and**
⟨*x2e* = (*x1f*, *x2f*)⟩ **and**
⟨*x2d* = (*x1e*, *x2e*)⟩ **and**
⟨*x2c* = (*x1d*, *x2d*)⟩ **and**
⟨*x2b* = (*x1c*, *x2c*)⟩ **and**
⟨*x2a* = (*x1b*, *x2b*)⟩ **and**
⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*y* = (*x1*, *x2*)⟩ **and**
⟨*x2s* = (*x1t*, *x2t*)⟩ **and**
⟨*x2r* = (*x1s*, *x2s*)⟩ **and**
⟨*x2q* = (*x1r*, *x2r*)⟩ **and**
⟨*x2p* = (*x1q*, *x2q*)⟩ **and**
⟨*x2o* = (*x1p*, *x2p*)⟩ **and**
⟨*x2n* = (*x1o*, *x2o*)⟩ **and**
⟨*x2m* = (*x1n*, *x2n*)⟩ **and**
⟨*x2l* = (*x1m*, *x2m*)⟩ **and**
⟨*x2k* = (*x1l*, *x2l*)⟩ **and**
⟨*x2j* = (*x1k*, *x2k*)⟩ **and**
⟨*x2i* = (*x1j*, *x2j*)⟩ **and**
⟨*x2h* = (*x1i*, *x2i*)⟩ **and**
⟨*x2g* = (*x1h*, *x2h*)⟩ **and**
⟨*x* = (*x1g*, *x2g*)⟩
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t*
proof −

show *?thesis*
using *that* **unfolding** *isa-set-lookup-conflict-aa-pre-def set-conflict-wl′-pre-def twl-st-heur′-def twl-st-heur-def*
by (*auto simp*: *arena-lifting*)
qed

show *?thesis*
supply [[*goals-limit=1*]]
apply (*intro nres-relI frefI*)
unfolding *uncurry-def RES-RETURN-RES4 set-conflict-wl′-alt-def2 set-conflict-wl-heur-def*
apply (*rewrite at* ‹*let - = zero-uint32-nat in -*› *Let-def*)
apply (*refine-vcg*)
subgoal by (*rule isa-set-lookup-conflict-aa-pre*)
apply *assumption+*
subgoal for *x y*
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
by (*rule isa-length-trail-pre*)
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*)
subgoal for *x y*
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl* (*snd y*)›], *rule exI*[*of -* ‹*set* (*get-vdom* (*snd x*))›])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def set-conflict-wl′-pre-def*)
subgoal
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def counts-maximum-level-def set-conflict-wl′-pre-def all-atms-def*[*symmetric*]
intro!: *valid-arena-arena-incr-act valid-arena-mark-used*)
done
qed

lemma *in-Id-in-Id-option-rel*[*refine*]:
‹$(f, f') \in Id \Longrightarrow (f, f') \in \langle Id\rangle$ *option-rel*›
by *auto*

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* =
($\lambda(((L, j), w), S)$. $j <$ *length* (*watched-by S L*) $\wedge$ $w <$ *length* (*watched-by S L*) $\wedge$
$L \in\# \mathcal{L}_{all}$ (*all-atms-st S*))›

lemma *vdom-m-update-subset′*:
‹*fst C* $\in$ *vdom-m* $\mathcal{A}$ *bh N* $\Longrightarrow$ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *C*])) *N* $\subseteq$ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *vdom-m-update-subset*:
‹*bg* $<$ *length* (*bh ap*) $\Longrightarrow$ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *bh ap* ! *bg*])) *N* $\subseteq$ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *keep-watch-heur-keep-watch*:
‹(*uncurry3 keep-watch-heur*, *uncurry3* (*RETURN oooo keep-watch*)) $\in$
[*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* $\rightarrow$ $\langle$*twl-st-heur-up″* $\mathcal{D}$ *r s K*$\rangle$ *nres-rel*›

by (*intro frefI nres-relI*)
(*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*]
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)

This is a slightly stronger version of the previous lemma:

lemma *keep-watch-heur-keep-watch'*:
⟨*keep-watch-heur-pre* (((*L*, *j*), *w*), *S*) $\Longrightarrow$
((((*L'*, *j'*), *w'*), *S'*), ((*L*, *j*), *w*), *S*)
$\in$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\Longrightarrow$
keep-watch-heur L' j' w' S' $\le \Downarrow$ {(*T*, *T'*). *get-vdom T* = *get-vdom S'* $\wedge$
(*T*, *T'*) $\in$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*}
(*RETURN* (*keep-watch L j w S*))⟩
by (*force simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*]
intro!: *ASSERT-leI dest*: *vdom-m-update-subset*)

definition *update-blit-wl-heur-pre* **where**
⟨*update-blit-wl-heur-pre r* = (λ((((((*L*, *C*), *b*), *j*), *w*), *K*), *S*). *L* $\in$# $\mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$
w < *length* (*watched-by S L*) $\wedge$ *w* < *r* $\wedge$ *j* < *r* $\wedge$
j < *length* (*watched-by S L*) $\wedge$ *C* $\in$ *vdom-m* (*all-atms-st S*) (*get-watched-wl S*) (*get-clauses-wl S*))⟩

lemma *update-blit-wl-heur-update-blit-wl*:
⟨(*uncurry6 update-blit-wl-heur*, *uncurry6 update-blit-wl*) $\in$
[*update-blit-wl-heur-pre r*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$
twl-st-heur-up'' $\mathcal{D}$ *r s K*$\rightarrow$
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩ *nres-rel*⟩
apply (*intro frefI nres-relI*) — TODO proof
apply (*auto simp*: *update-blit-wl-heur-def update-blit-wl-def twl-st-heur'-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def update-blit-wl-heur-pre-def all-atms-def*[*symmetric*]
intro!: *ASSERT-leI*
simp: *vdom-m-update-subset*)
subgoal for *aa ab ac ad ae be af ag ah bf aj ak al am an bg ao bh ap aq ar bi at bu bv cb cc cd ce cf cg ch cj ck cm y x*
apply (*subgoal-tac* ⟨*vdom-m* (*all-atms ch* (*cj* + *ck*)) (*cm*(*K* := (*cm K*)[*cd* := (*cb*, *cf*, *cc*)])) *ch* $\subseteq$
vdom-m (*all-atms ch* (*cj* + *ck*)) *cm ch*⟩)
apply *fast*
apply (*rule vdom-m-update-subset'*)
apply *auto*
done
subgoal for *aa ab ac ad ae be af ag ah bf aj ak al am an bg ao bh ap aq ar bi at bu bv ca cb cc cd ce cf cg ch cj ck cm y x*
apply (*subgoal-tac* ⟨*vdom-m* (*all-atms ch* (*cj* + *ck*)) (*cm*(*ca* := (*cm ca*)[*cd* := (*cb*, *cf*, *cc*)]))
ch $\subseteq$
vdom-m (*all-atms ch* (*cj* + *ck*)) *cm ch*⟩)
apply *fast*
apply (*rule vdom-m-update-subset'*)
apply *auto*
done
subgoal for *aa ab ac ad ae be af ag ah bf ai aj ak al am an bg ao bh ap aq ar bi at bu bv cb cc cd ce cf cg ch ci cj ck cm x*
apply (*subgoal-tac* ⟨*vdom-m* (*all-atms ch* (*cj* + *ck*)) (*cm*(*K* := (*cm K*)[*cd* := (*cb*, *cf*, *cc*)])) *ch* $\subseteq$
vdom-m (*all-atms ch* (*cj* + *ck*)) *cm ch*⟩)
apply *fast*

apply (*rule vdom-m-update-subset'*)
apply *auto*
done
subgoal for *aa ab ac ad ae be af ag ah bf ai aj ak al am an bg ao bh ap aq ar bi at bu bv ca cb cc cd ce cf cg ch ci cj ck cm x*
apply (*subgoal-tac* ‹*vdom-m* (*all-atms ch* (*cj* + *ck*)) (*cm*(*ca* := (*cm ca*)[*cd* := (*cb*, *cf*, *cc*)])) *ch* ⊆ *vdom-m* (*all-atms ch* (*cj* + *ck*)) *cm ch*›)
apply *fast*
apply (*rule vdom-m-update-subset'*)
apply *auto*
done
done

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* ∝ *C*) *S in*
RETURN
(*j* + *1*, *w* + *1*, *S*)
else
let i = ((*if get-clauses-wl S* ∝ *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 ∉# *dom-m* (*get-clauses-wl S*)
then RETURN (*j*, *w+1*, *S*)
else do {
let i = (*if* ((*get-clauses-wl S*)∝*C*) ! *0* = *L then 0 else 1*);
let L' = ((*get-clauses-wl S*)∝*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* ∝ *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* ∝ *C*) *S*;
RETURN (*j+1*, *w+1*, *S*)
}
else do {

$S \leftarrow RETURN\ (propagate\text{-}lit\text{-}wl\ L'\ C\ i\ S);$
$RETURN\ (j{+}1,\ w{+}1,\ S)$
}
}
| *Some f* $\Rightarrow$ *do* {
$S \leftarrow RETURN\ S;$
let $K = get\text{-}clauses\text{-}wl\ S \propto C\ !\ f;$
let $val\text{-}L' = polarity\ (get\text{-}trail\text{-}wl\ S)\ K;$
if $val\text{-}L' = Some\ True$
then update-blit-wl L C b j w K S
else update-clause-wl L C b j w i f S
}
}
}
}
}›
unfolding *unit-propagation-inner-loop-body-wl-D-def let-to-bind-conv*[*symmetric*] *Let-def*
by (*intro bind-cong-nres case-prod-cong if-cong*[*OF refl*] *refl*)

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* $\mathcal{D}$ *r s K*
assumes
xy: ‹$(x,\ y) \in nat\text{-}lit\text{-}lit\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f$
twl-st-heur-up″ $\mathcal{D}$ *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*) $\Rightarrow$
(*case x of*
(*x*, *xa*) $\Rightarrow$
(*case x of*
(*L*, *i*) $\Rightarrow$
$\lambda j\ S.\ length\ (watched\text{-}by\ S\ L) \le r - 4\ \wedge$
$L = K \wedge length\ (watched\text{-}by\ S\ L) = s)$
xa)
xa›
begin

private lemma *L-K*: ‹$L = K$›
using *L-K0* **unfolding** *st*
by *auto*

private 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: ⟨$(S, T) \in$ *twl-st-heur*⟩
using *xy st* **unfolding** *twl-st-heur′-def* **by** *auto*

private lemma *length-clss-Sr*: ⟨*length* (*get-clauses-wl-heur S*) = *r*⟩
using *xy* **unfolding** *state-simp-ST* **by** *auto*

private 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*)⟩
using *pre* **unfolding** *unit-propagation-inner-loop-wl-loop-D-pre-def*
unit-propagation-inner-loop-wl-loop-D-inv-def prod.simps image-image
by *simp-all*

private 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: ⟨$x2 \leq x2a$⟩ **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*⟩
using *pre pre-inv0 st x1b*
unfolding *watched-by-app-heur-pre-def prod.simps*
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def*
twl-st-heur′-def
unit-propagation-inner-loop-wl-loop-D-pre-def twl-st-heur-def map-fun-rel-def
unit-propagation-inner-loop-wl-loop-pre-def prod.simps
unit-propagation-inner-loop-wl-loop-inv-def **apply** −
by (*auto simp*: *state-simp-ST x1b x2b*)

lemma *watched-by-app-heur-pre*: ⟨*watched-by-app-heur-pre* ((*S*, *L′*), *x2d*)⟩
using *pre pre-inv0 st x2d-le x1e-le*
unfolding *watched-by-app-heur-pre-def prod.simps*
by (*simp add*: *state-simp-ST*)

lemma *keep-watch-heur-pre*: ⟨*keep-watch-heur-pre* (((*L*, *x2*), *x2a*), *T*)⟩
using *x2-x2a x2a-le x1b* **unfolding** *keep-watch-heur-pre-def*
by (*auto simp*: *x1b x2b*)

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'' 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*›
using *U* **unfolding** *twl-st-heur'-def* **by** *auto*

private lemma *eq*: ‹*watched-by T L* = *watched-by-int S L*› ‹*x1f* = *x1g*› ‹*x2f'* = *x2g'*› ‹*x2f* = *x2g*›
‹*x2f''* = *x2h*›
using *xf xg st x2f' x2g' xf x1b*
by (*auto simp*: *twl-st-heur-state-simp-watched*)

lemma *xg-S*: ‹*watched-by-int S L* ! *x2a* = (*x1g*, *x2g'*)›
using *xg* **by** *auto*

lemma *xg-T*: ‹*watched-by T L* ! *x2a* = (*x1g*, *x2g'*)›
using *U eq xf xg* **by** (*cases T*)
(*auto simp add*: *image-image*
twl-st-heur-state-simp-watched twl-st-heur'-def
twl-st-heur-def keep-watch-def)

context
notes -[*simp*]= *eq xg-S xg-T x2g'*
begin

lemma *in-D0*:
shows ‹*polarity-st-heur-pre* (*U*, *x2g*)›
using *U' unit-prop-body-wl-D-invD*[*OF prop-inv*] *xf xg x1b*
apply (*cases T*; *cases U*)
unfolding *find-unwatched-wl-st-heur-pre-def watched-by-app-def polarity-st-heur-pre-def*
by (*auto simp add*: *image-image twl-st-heur-state-simp-watched twl-st-heur'-def keep-watch-def*
twl-st-heur-def
intro!: *polarity-pol-pre*)

private lemma *x2g*: ‹*x2g* ∈# $\mathcal{L}_{all}$ (*all-atms-st T*)›
using *U' unit-prop-body-wl-D-invD*[*OF prop-inv*] *xf xg x1b*
apply (*cases T*; *cases U*)
unfolding *find-unwatched-wl-st-heur-pre-def watched-by-app-def polarity-st-heur-pre-def*
by (*auto simp add*: *image-image twl-st-heur-state-simp-watched twl-st-heur'-def keep-watch-def*
twl-st-heur-def
intro!: *polarity-pol-pre*)

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*)›
using *U' x1b x2g* **apply** (*cases U*; *cases T*)
apply (*subst polarity-pol-polarity*[*of* ‹*all-atms-st T*›,
unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*])
by (*auto intro*!: *twl-st-heur-state-simp simp*: *twl-st-heur-def*
keep-watch-def)

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*) $\subseteq$ *set* (*get-vdom* *U*)⟩
using *U′* **apply** (*cases* *T*; *auto* *simp*: *twl-st-heur-def* *keep-watch-def*; *fail*)
using *U′* **by** (*cases* *T*; *auto* *simp*: *twl-st-heur-def* *keep-watch-def*)

private lemma *x1g-vdom*: ⟨*x1f* $\in$ *vdom-m* (*all-atms-st* *T*) (*get-watched-wl* (*keep-watch* *L* *x2* *x2a* *T*))
(*get-clauses-wl* (*keep-watch* *L* *x2* *x2a* *T*))⟩
using *in-vdom-m-upd*[*of* *x2* ⟨*get-watched-wl* *T*⟩ *L* ⟨(*all-atms-st* *T*)⟩ *x1g* *x2g′*] *x2-x2a* *x2a-le* *eq* *x1b*
by (*cases* *T*)
(*auto* *simp*: *keep-watch-def* *simp* *del*: *eq*)

lemma *clause-not-marked-to-delete-heur-pre*:
⟨*clause-not-marked-to-delete-heur-pre* (*U*, *x1g*)⟩
using *x1g-vdom* *valid-UT* *vdom-m-UT*
unfolding *clause-not-marked-to-delete-heur-pre-def* *prod.simps* *arena-is-valid-clause-vdom-def*
by *auto*

private lemma *clause-not-marked-to-delete-pre*:
⟨*clause-not-marked-to-delete-pre* (*keep-watch* *L* *x2* *x2a* *T*, *x1f*)⟩
using *x1g-vdom*
unfolding *clause-not-marked-to-delete-pre-def* *prod.case* **by** *auto*

lemma *clause-not-marked-to-delete-heur-clause-not-marked-to-delete-iff*:
⟨($\neg$ *clause-not-marked-to-delete-heur* *U* *x1g*) $\longleftrightarrow$
($\neg$ *clause-not-marked-to-delete* (*keep-watch* *L* *x2* *x2a* *T*) *x1f*)⟩
apply (*subst* *Not-eq-iff*)
apply (*rule* *clause-not-marked-to-delete-rel*[*THEN* *fref-to-Down-unRET-uncurry-Id*])
apply (*rule* *clause-not-marked-to-delete-pre*)
using *U* **by** (*auto* *simp*: *twl-st-heur′-def*)

private 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 −
obtain *x* *xa* **where**
lits: ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* *T*) *T*⟩ **and**
⟨(*U*, *keep-watch* *L* *x2* *x2a* *T*) $\in$ *twl-st-heur*⟩ **and**
Tx: ⟨(*T*, *x*) $\in$ *state-wl-l* (*Some* (*L*, *x2a*))⟩ **and**
⟨*x2* $\leq$ *x2a*⟩ **and**
corr: ⟨*correct-watching-except* *x2* *x2a* *L* *T*⟩ **and**
⟨*x2a* $\leq$ *length* (*watched-by* *T* *L*)⟩ **and**
xxa: ⟨(*x*, *xa*) $\in$ *twl-st-l* (*Some* *L*)⟩ **and**
struct: ⟨*twl-struct-invs* *xa*⟩ **and**
⟨*twl-stgy-invs* *xa*⟩ **and**
⟨*twl-list-invs* *x*⟩ **and**
clss: ⟨*clauses-to-update* *xa* $\neq$ {#} $\vee$ *0* < *remaining-nondom-wl* *x2a* *L* *T* $\longrightarrow$
get-conflict *xa* = *None*⟩ **and**
uL: ⟨− *L* $\in$ *lits-of-l* (*get-trail-l* *x*)⟩
using *x2b* *U′* *loop-inv-T* **unfolding** *unit-propagation-inner-loop-wl-loop-inv-def* *prod.simps*
unit-propagation-inner-loop-l-inv-def
by *metis*

from *Tx struct xxa lits*
show ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st T*) (*get-trail-wl T*)›
by (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*)
have ‹*no-dup* (*trail* (*state*$_W$*-of xa*))›
using *struct* **unfolding** *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by *blast*
then show ‹*no-dup* (*get-trail-wl T*)›
using *Tx xxa* **by** (*auto simp*: *twl-st*)
then show ‹*polarity* (*get-trail-wl T*) *L* = *Some False*›
using *uL Tx* **unfolding** *polarity-def*
by (*auto dest*: *no-dup-consistentD in-lits-of-l-defined-litD*)
show ‹*correct-watching-except x2 x2a L T*›
using *corr* **.**
qed

lemma *prop-fast-le*:
assumes *fast*: ‹*length* (*get-clauses-wl-heur S*) ≤ *uint64-max*›
shows ‹*x2c* < *uint64-max*› ‹*x2d* < *uint64-max*›
proof −
obtain *x xa* **where**
Sx: ‹(*S*, *x*) ∈ *twl-st-heur*› **and**
‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st x*) *x*› **and**
L': ‹*L'* ∈# $\mathcal{L}_{all}$ (*all-atms-st x*)› **and**
xxa: ‹(*x*, *xa*) ∈ *state-wl-l* (*Some* (*L'*, *x2d*))› **and**
le: ‹*x2c* ≤ *x2d*› **and**
‹*unit-propagation-inner-loop-l-inv L'*
(*xa*, *remaining-nondom-wl x2d L' x*)› **and**
corr: ‹*correct-watching-except x2c x2d L' x*› **and**
le': ‹*x2d* ≤ *length* (*watched-by x L'*)› **and**
le-wb: ‹*length* (*watched-by x L'*) ≤ *length* (*get-clauses-wl-heur S*) − *4*›
using *pre-inv0*
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def*
unit-propagation-inner-loop-wl-loop-D-inv-def
unit-propagation-inner-loop-wl-loop-inv-def
prod.simps
apply −
apply *normalize-goal+*
by *blast*
show ‹*x2c* < *uint64-max*› ‹*x2d* < *uint64-max*›
using *fast le-wb le le'*
by (*auto simp*: *isasat-fast-def uint64-max-def*)
qed

context
fixes *x1i x2i x1i' x2i'*
assumes *x2h*: ‹*x2f'* = (*x1i'*, *x2i'*)› **and**
x2h': ‹*x2g'* = (*x1i*, *x2i*)›
begin

lemma *bin-last-eq*: ‹*x2i* = *x2i'*›
using *x2h x2h'*
by *auto*

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*›
  using *unit-prop-body-wl-D-invD*[*OF prop-inv*] *proper*
  by (*auto simp*: *eq watched-by-app-def propagate-proper-bin-case-def*)

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*)›
  using *U′ x2g*
  apply (*cases T*; *cases U*)
  by (*subst polarity-pol-polarity*[*of* ‹*all-atms-st T*›,
    unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*])
    (*auto simp add*: *twl-st-heur-def keep-watch-def*)

lemma *bin-set-conflict-wl-heur-pre*:
  ‹*set-conflict-wl-heur-pre* (*x1g*, *U*)›
proof −
  have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st T*) (*mset* ‘# *ran-mf* (*get-clauses-wl* (*keep-watch L x2 x2a T*)))›
    using *x2b* **unfolding** *literals-are-*$\mathcal{L}_{in}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def is-*$\mathcal{L}_{all}$*-def all-lits-def*
    by (*simp add*: *all-lits-of-mm-union*)
  then show *?thesis*
    using *proper no-dup-T U′ lits-in-trail*
    apply (*cases T*; *cases U*)
    unfolding *propagate-proper-bin-case-def set-conflict-wl-heur-pre-def*
    by (*auto simp*: *twl-st-heur-def keep-watch-def*)
qed

lemma *polarity-st-keep-watch*:
  ‹*polarity-st* (*keep-watch L x2 x2a T*) = *polarity-st T*›
  by (*intro ext*, *cases T*) (*auto simp*: *keep-watch-def polarity-st-def*)

lemma *access-lit-in-clauses-keep-watch*:
  ‹*access-lit-in-clauses* (*keep-watch L x2 x2a T*) = *access-lit-in-clauses T*›
  by (*intro ext*, *cases T*) (*auto simp*: *keep-watch-def access-lit-in-clauses-def*)

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 −
  have *x2b′*: ‹*x1f* = *fst* (*watched-by-app* (*keep-watch L x2 x2a T*) *L x2a*)›
    using *x2-x2a x2a-le*
    by (*auto simp*: *watched-by-app-def*)
  have ‹*length* (*get-clauses-wl T* ∝ *fst* (*watched-by-app* (*keep-watch L x2 x2a T*) *L x2a*)) = *2*›
    using *proper* **unfolding** *propagate-proper-bin-case-def x2b′*
    by *auto*
  then have *unw*: ‹*unit-prop-body-wl-D-find-unwatched-inv None*
    (*fst* (*watched-by-app* (*keep-watch L x2 x2a T*) *L x2a*))

(*keep-watch L x2 x2a T*)›
by (*auto simp*: *unit-prop-body-wl-D-find-unwatched-inv-def*
unit-prop-body-wl-find-unwatched-inv-def)
have *not-tauto*: ‹¬ *tautology* (*mset* (*get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f*))›
apply (*subst x2b′*)
apply (*rule find-unwatched-not-tauto*[*of - - - x2*])
subgoal by (*rule unw*)
subgoal using *prop-inv* **.**
subgoal using *proper pol pol-L U′* **unfolding** *propagate-proper-bin-case-def*
length-list-2
unfolding *watched-by-app-def*[*symmetric*] *x2b′*[*symmetric*]
by (*auto simp add*: *bin-polarity-eq polarity-st-def doubleton-eq-iff*)
subgoal using *in-dom* **unfolding** *watched-by-app-def*[*symmetric*] *x2b′*[*symmetric*] **.**
done
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st T*)
(*mset* ‘# *ran-mf* (*get-clauses-wl* (*keep-watch L x2 x2a T*)))›
using *x2b* **unfolding** *literals-are-*$\mathcal{L}_{in}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def is-*$\mathcal{L}_{all}$*-def all-atms-def*
all-lits-def
by (*simp add*: *all-lits-of-mm-union*)
show *?thesis*
using *not-tauto bin-confl-T bin-dist-Tx1g lits lits-in-trail proper*
unfolding *set-conflict-wl′-pre-def uncurry-def prod.simps propagate-proper-bin-case-def*
by *auto*
qed

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*›
using *length-clss-Sr U* **by** *auto*

lemma *bin-access-lit-in-clauses-heur-pre*:
‹*access-lit-in-clauses-heur-pre* ((*U*, *x1g*), *0*)›
using *U′ in-dom length-clss-2* **apply** −
unfolding *access-lit-in-clauses-heur-pre-def prod.case arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def
apply (*rule bex-leI*[*of - x1f*])
apply (*rule exI*[*of -* ‹*get-clauses-wl T*›])
apply (*rule exI*[*of -* ‹*set* (*get-vdom U*)›])
by (*cases T*)
(*auto simp*: *twl-st-heur-def keep-watch-def*)

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 −
have [*dest!*]: ‹*A* ≠ *Some True* ⟹ *A* ≠ *Some False* ⟹ *A* = *None*› **for** *A*
by (*cases A*) *auto*
have ‹*polarity-pol* (*get-trail-wl-heur U*) *x2g* = *None*›
using *pol pol′ U′ x1b x2g*
apply (*cases T*)
apply (*subst* (*asm*) *polarity-pol-polarity*[*of* ‹*all-atms-st T*›,
unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*, *symmetric*,
of - - ‹*get-trail-wl-heur U*› *x2g*])
by (*auto simp*: *twl-st-heur-def keep-watch-def*)

moreover have ‹*x1g* ≠ *DECISION-REASON*›
  using *arena-lifting*(*1*)[*OF valid*, *of x1f*] *in-dom*
  by (*auto simp*: *header-size-def DECISION-REASON-def split*: *if-splits*)
ultimately show *?thesis*
  unfolding *propagate-lit-wl-heur-pre-def*
  by *auto*
qed

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 −
have [*dest*!]: ‹*A* ≠ *Some True* ⟹ *A* ≠ *Some False* ⟹ *A* = *None*› **for** *A*
  by (*cases A*) *auto*
have ‹*polarity* (*get-trail-wl* (*keep-watch L x2 x2a T*)) *x2g* = *None*›
  using *pol pol′ U′ x1b x2g*
  apply (*cases T*)
  by (*subst* (*asm*) *polarity-pol-polarity*[*of* ‹*all-atms-st T*›,
   unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*,
    of - x2f ‹*get-trail-wl-heur U*› *x2g*])
   (*auto simp*: *twl-st-heur-def keep-watch-def*)
then have ‹*undefined-lit* (*get-trail-wl* (*keep-watch L x2 x2a T*)) *x2g*›
  by (*simp add*: *no-dup-T polarity-spec′*(*1*))
then have ‹*undefined-lit* (*get-trail-wl T*) *x2g*›
  using *U′ twl-st-heur-state-simp*(*1*) **by** *auto*
then show *?thesis*
  using *length-clss-2 in-dom U bin-confl-T x2g*
  unfolding *propagate-lit-wl-bin-pre-def*
  by *auto*
qed

private lemma *bin-arena-lit-eq*:
 ‹*i* < *2* ⟹ *arena-lit* (*get-clauses-wl-heur U*) (*x1g* + *i*) = *get-clauses-wl T* ∝ *x1g* ! *i*›
using *U′ in-dom length-clss-2*
by (*cases U*; *cases T*; *cases i*)
 (*auto simp*: *keep-watch-def twl-st-heur-def arena-lifting*)

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″ $\mathcal{D}$ *r s K*›
using *U bin-arena-lit-eq*[*of 0*] *bin-arena-lit-eq*[*of 1*] *length-clss-Sr* **by** *auto*

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)›
using *not-del* **unfolding** *clause-not-marked-to-delete-def*
by *auto*

private lemma *Tx1g-le2*:
‹*length* (*get-clauses-wl* T ∝ $x1g$) ≥ *2*›
using *arena-lifting*[*OF valid-UT*, *of x1g*]
by (*auto simp*: *x1g*)

lemma *access-lit-in-clauses-heur-pre0*:
‹*access-lit-in-clauses-heur-pre* ((*U*, *x1g*), *0*)›
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def
by (*rule bex-leI*[*of - x1g*], *rule exI*[*of -* ‹*get-clauses-wl T*›],
rule exI[*of -* ‹*set* (*get-vdom U*)›])
(*use valid-UT Tx1g-le2 x1g* **in** *auto*)

private 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*))›
unfolding *i-def* **by** *auto*

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*)))›
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def i-def
by (*rule bex-leI*[*of - x1g*], *rule exI*[*of -* ‹*get-clauses-wl T*›],
rule exI[*of -* ‹*set* (*get-vdom U*)›])
(*use valid-UT Tx1g-le2 x1g* **in** *auto*)

private lemma *trail-UT*:
‹(*get-trail-wl-heur U*, *get-trail-wl T*) ∈ *trail-pol* (*all-atms-st T*)›
using *U′* **by** (*cases U*; *cases T*; *auto simp*: *keep-watch-def twl-st-heur-def*)

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*))))›
unfolding *polarity-st-heur-pre-def prod.case*

unfolding *find-unwatched-wl-st-heur-pre-def watched-by-app-def polarity-st-heur-pre-def*
apply (*rule polarity-pol-pre*[*OF trail-UT*])
apply (*cases* ‹*get-clauses-wl T* ∝ *x1g*›)
using *arena-lifting*(*5*)[*OF valid-UT x1g*, *of 0*] *arena-lifting*(*5*)[*OF valid-UT x1g*, *of 1*] *x1b Tx1g-le2*
unit-prop-body-wl-D-invD[*OF prop-inv*]
by (*auto simp add*: *image-image x1g watched-by-app-def*
split: *if-splits*)

private 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*)›
using *arena-lifting*[*OF* *valid-UT* *x1g*]
unfolding *i-def*
by *auto*

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*)›
using *U*′ *trail-UT* *unit-prop-body-wl-D-invD*[*OF* *prop-inv*] *x1b* *Tx1g-le2*
unfolding *i-def*[*symmetric*] *i-alt-def*[*symmetric*] *i-alt-def-L*′[*symmetric*]
unfolding *access-x1g* *access-x1g1i*
apply (*subst* *polarity-pol-polarity*[*of* ‹*all-atms-st* *T*›,
unfolded *option-rel-id-simp*, *THEN* *fref-to-Down-unRET-uncurry-Id*,
of ‹*get-trail-wl* (*keep-watch* *L* *x2* *x2a* *T*)›
‹*get-clauses-wl* (*keep-watch* *L* *x2* *x2a* *T*) ∝ *x1f* ! (*1* − *i*)› ‹*get-trail-wl-heur* *U*›])
by (*auto* *simp*: *i-def* *watched-by-app-def* *x1g*)

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*)›
using *x2-x2a* *x2a-le* *x1g* *x1b* *L-K0* *x2a-le* **unfolding** *st*
unfolding *i-def*[*symmetric*] *i-alt-def*[*symmetric*] *update-blit-wl-heur-pre-def* *prod.simps*
by *auto*

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*›
using *U* *length-clss-Sr*
unfolding *i-def*[*symmetric*] *i-alt-def*[*symmetric*] *i-alt-def-L*′[*symmetric*]
unfolding *access-x1g* *access-x1g1i*
by *auto*

lemma *find-unwatched-wl-st-pre*:

⟨*find-unwatched-wl-st-pre* (*keep-watch L x2 x2a T*, *x1f*)⟩
using *x2-x2a x2a-le Tx1g-le2 unit-prop-body-wl-D-invD*[*OF prop-inv*]
unfolding *find-unwatched-wl-st-pre-def prod.simps*
unfolding *access-x1g access-x1g1i*
by (*auto simp*: *xf xg x1g watched-by-app-def*)

lemma *find-unwatched-wl-st-heur-pre*:
⟨*find-unwatched-wl-st-heur-pre* (*U*, *x1g*)⟩
unfolding *find-unwatched-wl-st-heur-pre-def access-lit-in-clauses-heur-pre-def arena-is-valid-clause-idx-def arena-lit-pre-def prod.simps*
by (*rule exI*[*of* - ⟨*get-clauses-wl T*⟩],
rule exI[*of* - ⟨*set* (*get-vdom U*)⟩])
(*use valid-UT Tx1g-le2 x1g* **in** *auto*)

lemma *isa-find-unwatched-wl-st-heur-pre*:
⟨((*U*, *x1g*), *keep-watch L x2 x2a T*, *x1f*) $\in$ *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*)⟩
using *U′ x2-x2a x2a-le x2a-le x2b* **by** *auto*

context — Now we try to find another literal to watch
notes - [*simp*] = *x1g*
fixes *f f′*
assumes *ff*: ⟨(*f*, *f′*) $\in$ *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′*⟩
using *ff* **by** *auto*

lemma *unit-prop-body-wl-D-find-unwatched-heur-inv*:
⟨*unit-prop-body-wl-D-find-unwatched-heur-inv f x1g U*⟩
using *U′ find-unw-pre*
unfolding
unit-prop-body-wl-D-find-unwatched-heur-inv-def
apply −
by (*rule exI*[*of* - ⟨*keep-watch L x2 x2a T*⟩]) (*auto simp*: *ff*)

private lemma *confl-T*: ⟨*get-conflict-wl T* = *None*⟩ **and**
dist-Tx1g: ⟨*distinct* (*get-clauses-wl T* $\propto$ *x1g*)⟩ **and**
L-in-watched: ⟨*L* $\in$ *set* (*watched-l* (*get-clauses-wl T* $\propto$ *x1g*))⟩
using *unit-prop-body-wl-D-invD*[*OF prop-inv*]
by (*auto simp*: *eq watched-by-app-def*)

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) ∝ x1f !
(1 −
(if get-clauses-wl (keep-watch L x2 x2a T) ∝ x1f ! 0 = L then 0
else 1))) =
Some False)›
apply (*subst polarity-pol-polarity*[*of* ‹*all-atms-st T*›,
unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*,
of ‹*get-trail-wl* (*keep-watch L x2 x2a T*)›
‹*get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! (*1* − *i*)› ‹*get-trail-wl-heur U*›])
unfolding *i-def*[*symmetric*] *i-alt-def*[*symmetric*] *i-alt-def-L′*[*symmetric*] *access-x1g1i*
using *U′ trail-UT unit-prop-body-wl-D-invD*[*OF prop-inv*] *x1b Tx1g-le2*
by (*auto simp*: *x1g i-def watched-by-app-def*)

lemma *set-conflict-wl-heur-pre*: ‹*set-conflict-wl-heur-pre* (*x1g*, *U*)›
using *lits-in-trail U′ no-dup-T*
unfolding *set-conflict-wl-heur-pre-def prod.simps*
by (*auto simp*: *twl-st-heur-state-simp*)

lemma *i-alt-def2*:
‹*i* = (*if access-lit-in-clauses* (*keep-watch L x2 x2a T*) *x1f 0* = *L then 0*
else 1)›
using *U′ access-x1g access-x1g1i* **unfolding** *i-def*
by (*auto simp*: *twl-st-heur-state-simp access-lit-in-clauses-def*)

lemma *x2da-eq*: ‹(*x2d*, *x2a*) ∈ *nat-rel*›
by *auto*

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*) ∝ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *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 −
have *x2b′*: ‹*x1f* = *fst* (*watched-by-app* (*keep-watch L x2 x2a T*) *L x2a*)›
using *x2-x2a x2a-le*
by (*auto simp*: *watched-by-app-def*)
have *not-tauto*: ‹¬ *tautology* (*mset* (*get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f*))›
apply (*subst x2b′*)
apply (*rule find-unwatched-not-tauto*[*of - - - x2*])

subgoal using *find-unw-pre* **unfolding** *f′ x2b′ watched-by-app-def* **by** *auto*
subgoal using *prop-inv* **.**
subgoal
using *pol-false*
unfolding *x2b′*[*symmetric*] *i-def*[*symmetric*] *i-alt-def2*[*symmetric*] *i-alt-def*[*symmetric*]
polarity-st-def **by** *blast*
subgoal unfolding *watched-by-app-def*[*symmetric*] *x2b′*[*symmetric*]
by *auto*
done
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st T*)
(*mset ‘# ran-mf* (*get-clauses-wl* (*keep-watch L x2 x2a T*)))›
using *x2b* **unfolding** *literals-are-*$\mathcal{L}_{in}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def is-*$\mathcal{L}_{all}$*-def all-lits-def*
by (*simp add*: *all-lits-of-mm-union*)
show *?thesis*
using *not-tauto confl-T dist-Tx1g lits lits-in-trail*
unfolding *set-conflict-wl′-pre-def uncurry-def prod.simps*
by *auto*
qed

lemma *set-conflict-keep-watch-rel*:
‹((*x1g*, *U*), *x1f*, *keep-watch L x2 x2a T*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*›
using *U length-clss-Sr* **by** *auto*

lemma *set-conflict-keep-watch-rel2*:
‹⋀*r*. (*W*, *W′*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* ⟹
((*x2c* + *1*, *W*), *x2* + *1*, *W′*) ∈ *nat-rel* $\times_f$ (*nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*)›
by *auto*

end

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*) ∝ *x1f* !
(*1* −
(*if get-clauses-wl* (*keep-watch L x2 x2a T*) ∝ *x1f* ! *0* = *L then 0*
else 1))) ≠
Some False› **and**
‹*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› **and**
pol-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⟩
begin

private lemma *undef-lit1i*:
⟨*undefined-lit* (*get-trail-wl T*) (*get-clauses-wl T* ∝ *x1g* ! (*Suc 0* − *i*))⟩
using *pol-True pol-False U′*
unfolding *i-def*[*symmetric*] *i-alt-def-L′*[*symmetric*]
i-alt-def[*symmetric*] *watched-by-app-def*
by (*auto simp*: *polarity-def twl-st-heur-state-simp split*: *if-splits*)

lemma *propagate-lit-wl-heur-pre*:
⟨*propagate-lit-wl-heur-pre*
(((*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*:: *nat*)),
U)⟩ (**is** *?A*)
proof −
have ⟨*i* = *0* ∨ *i* = *1*⟩
unfolding *i-def* **by** *auto*
moreover have ⟨*x1g* ≠ *DECISION-REASON*⟩
using *arena-lifting*(*1*)[*OF valid x1g*]
by (*auto simp*: *header-size-def DECISION-REASON-def split*: *if-splits*)
ultimately show *?A*
using *unit-prop-body-wl-D-invD*[*OF prop-inv*] *undef-lit1i*
unfolding *propagate-lit-wl-heur-pre-def prod.simps i-def*[*symmetric*] *i-alt-def-L′*[*symmetric*]
i-alt-def[*symmetric*] *watched-by-app-def*
unfolding *access-x1g1i access-x1g*
by (*auto simp*: *image-image*)
qed

private lemma *propagate-lit-wl-i-0-1*: ⟨*i* = *0* ∨ *i* = *1*⟩
unfolding *i-def* **by** *auto*

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)⟩
using *unit-prop-body-wl-D-invD*[*OF prop-inv*] *undef-lit1i propagate-lit-wl-i-0-1*
unfolding *propagate-lit-wl-pre-def prod.simps i-def*[*symmetric*] *i-alt-def-L′*[*symmetric*]
i-alt-def[*symmetric*] *watched-by-app-def*
unfolding *access-x1g1i access-x1g*
by (*auto simp*: *image-image twl-st-heur-state-simp*)

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*›
using *unit-prop-body-wl-D-invD*[*OF prop-inv*] *undef-lit1i U length-clss-Sr*
unfolding *propagate-lit-wl-pre-def prod.simps i-def*[*symmetric*] *i-alt-def-L′*[*symmetric*]
i-alt-def[*symmetric*] *watched-by-app-def*
unfolding *access-x1g1i access-x1g*
by (*auto simp*: *image-image twl-st-heur-state-simp*)

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*›
using *ff* **unfolding** *f f′* **by** *auto*

private 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*›
using *find-unw-pre* **unfolding** *unit-prop-body-wl-D-find-unwatched-inv-def f′*
by *auto*

private 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*)›
using *j-ge2 valid-UT j-le*
unfolding *isa-update-pos-pre-def access-lit-in-clauses-heur-pre-def*
arena-lit-pre-def arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def
by (*auto simp*: *arena-lifting*)

private abbreviation *isa-save-pos-rel* **where**
‹*isa-save-pos-rel* ≡ {(*V*, *V′*). *get-vdom V* = *get-vdom S* ∧ (*V*, *V′*) ∈ *twl-st-heur′ 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*))›
using *j-ge2 isa-update-pos-pre U x1g j-le*
by (*cases U*; *cases T*)
(*auto 5 5 simp*: *isa-save-pos-def twl-st-heur-def keep-watch-def twl-st-heur'-def arena-update-pos-alt-def arena-lifting ij arena-is-valid-clause-idx-def intro*!: *ASSERT-leI valid-arena-update-pos*)

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'' 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*))›
using *VV' U length-clss-Sr*
apply ((*auto*; *fail*)+)[*6*]
using *VV' U length-clss-Sr*
apply *auto*
apply (*cases T*; *auto simp*: *twl-st-heur'-def twl-st-heur-def keep-watch-def*)

using *VV' U length-clss-Sr*
apply (*cases T*; *auto simp*: *twl-st-heur'-def twl-st-heur-def keep-watch-def*)
done

lemma *access-lit-in-clauses-heur-pre3*: ‹*access-lit-in-clauses-heur-pre* ((*V*, *x1g*), *i*)›
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of - x1g*], *rule exI*[*of -* ‹*get-clauses-wl V'*›],
rule exI[*of -* ‹*set* (*get-vdom U*)›])
(*use valid-VT j-le* **in** ‹*auto simp*: *VV'*›)

private lemma *arena-lit-x1g-j*:
‹*arena-lit* (*get-clauses-wl-heur V*) (*x1g + j*) = *get-clauses-wl T* ∝ *x1g* ! *j*›
using *arena-lifting*[*OF valid-VT*, *of x1g*] *j-le*
by *auto*

lemma *polarity-st-pre-unwatched*: ‹*polarity-st-heur-pre* (*V*, *arena-lit* (*get-clauses-wl-heur V*) (*x1g + i*))›
unfolding *polarity-st-heur-pre-def arena-lit-x1g-j prod.simps*
by (*rule polarity-pol-pre*[*OF trail-VT*])
(*use x2b* **in** ‹*simp add*: *image-iff j-le literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *literals-are-in-*$\mathcal{L}_{in}$*-nth arena-lit-x1g-j*›)

private lemma *j-Lall*: ‹*get-clauses-wl* V' ∝ *x1g* ! *j* ∈# $\mathcal{L}_{all}$ (*all-atms-st* T)›
  using *x2b* **by** (*auto simp*: *image-iff j-le VV' intro*!: *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *literals-are-in-*$\mathcal{L}_{in}$*-nth*)

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)›
  apply (*subst polarity-pol-polarity*[*of* ‹*all-atms-st* V'›,
      unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*,
        of ‹*get-trail-wl* V'›
    ‹*get-clauses-wl* V' ∝ *x1f* ! *j*› ‹*get-trail-wl-heur* V›])
  using U' *VV' trail-UT j-Lall* **unfolding** *arena-lit-x1g-j*
  by (*auto simp*: *arena-lit-x1g-j*)

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')›
  using *x2-x2a x2a-le j-Lall x1b L-K0 x2a-le*
  unfolding *update-blit-wl-heur-pre-def st*
  by *auto*

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' ∝ *x1f* ! *j*), V')
    ∈ *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*›
  using *U V-T-rel length-clss-Sr* **by** *auto*

end

context
  notes - [*simp*] = *VV'*
  assumes ‹*polarity* (*get-trail-wl* V') (*get-clauses-wl* V' ∝ *x1f* ! *j*) ≠ *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*›
  unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
     arena-is-valid-clause-idx-and-access-def
  by (*rule exI*[*of* - ‹*get-clauses-wl* T›],
     rule exI[*of* - ‹*set* (*get-vdom* U)›])
    (*use valid-VT j-le* **in** *auto*)

private lemma *L-le*:

⟨*nat-of-lit* L < *length* (*get-watched-wl-heur* V)⟩
⟨*nat-of-lit* (*get-clauses-wl* V' ∝ *x1g* ! *j*) < *length* (*get-watched-wl-heur* V)⟩
using U' *j-Lall x1b*
by (*cases* T; *cases* U; *auto simp*: *twl-st-heur-def keep-watch-def map-fun-rel-def*; *fail*)+

private lemma *length-get-watched-wl-heur-U-T*:
⟨*length* (*get-watched-wl-heur* U ! *nat-of-lit* L) = *length* (*get-watched-wl* T L)⟩
using U' *j-Lall x1b*
by (*cases* T; *cases* U; *auto simp*: *twl-st-heur-def keep-watch-def map-fun-rel-def*)

private lemma *length-get-watched-wl-heur-S-T*:
⟨*length* (*watched-by-int* S L) = *length* (*get-watched-wl* T L)⟩
using *st j-Lall x1b*
by (*cases* T; *auto simp*: *twl-st-heur-def keep-watch-def map-fun-rel-def*; *fail*)+

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)⟩
using *x2a-le x2-x2a arena-is-valid-clause-idx-and-access-x1g-j x1e-le* U' *x1b L-le*
length-get-watched-wl-heur-U-T length-get-watched-wl-heur-S-T valid-VT j-le
unfolding *update-clause-wl-code-pre-def*
by (*auto simp*: *arena-lifting*)

private lemma *L-neq-j*:
⟨$L \neq$ *get-clauses-wl* T ∝ *x1g* ! *j*⟩
using *dist-Tx1g L-in-watched Tx1g-le2 j-le j-ge2*
by (*cases* ⟨*get-clauses-wl* T ∝ *x1g*⟩; *cases* ⟨*tl* (*get-clauses-wl* T ∝ *x1g*)⟩)
auto

thm *corr-T*
find-theorems S T
find-theorems *correct-watching-except keep-watch*

private lemma *in-lall*: ⟨*get-clauses-wl* T ∝ *x1g* ! *j*
∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl* T) (*get-unit-clauses-wl* T))⟩
using *multi-member-split*[*OF x1g*] *j-le* **by** (*auto simp*: *all-atms-def all-lits-def ran-m-def*
all-lits-of-mm-add-mset atm-of-all-lits-of-m in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *image-Un*
simp del: *all-atms-def*[*symmetric*])

private lemma *length-le*: ⟨*length* (*watched-by* T (*get-clauses-wl* T ∝ *x1g* ! *j*))
≤ *length* (*get-clauses-wl-heur* S) − *4*⟩
using *xy length-watched-le2*[*OF corr-T*, *of S* $\mathcal{D}$ *r* ⟨(*get-clauses-wl* T ∝ *x1g* ! *j*)⟩]
L-neq-j in-lall
by (*simp add*: *correct-watching-except.simps keep-watch-def*)

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) ∝ *x1f* ! *0* = L *then 0 else 1*),
j),
V')⟩
using *Tx1g-le2 j-le x1b T-x1g-j-neq0 T-x1g-j-neq1 L-neq-j L-K0 x2a-le length-le xy*
unfolding *update-clause-wl-pre-def st*
by (*auto simp*: *i-alt-def L-K*)

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*›
using *U V-T-rel* **unfolding** *access-x1g1i access-x1g* **by** *auto*

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 −
have [*simp*]: ‹*unit-prop-body-wl-D-inv T i C L* ⟹ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st T*)› **for** *T i L C*
unfolding *unit-prop-body-wl-D-inv-def image-image* **by** *auto*
have *pol-undef*: ‹*polarity M L* ≠ *Some True* ⟹ *polarity M L* ≠ *Some False* ⟹ *defined-lit M L* ⟹ *False*›
for *M* :: ‹(*nat*, *nat*) *ann-lits*› **and** *L* :: ‹*nat literal*›
by (*auto simp*: *polarity-def split*: *if-splits*)
have *1*: ‹*RETURN* (*w* + *1*, *f S'*) = *do* {*S* ← *RETURN* (*f S'*); *RETURN* (*w* + *1*, *S*)}›
for *w* :: *nat* **and** *S'* **and** *f*
by *auto*
have *keep-watch-skip*: ‹((*x2d* + *1*, *U*), *x2a* + *1*, *keep-watch L x2 x2a T*)
∈ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*›
if ‹(*x2d* + *1*, *x2a* + *1*) ∈ *nat-rel*› **and**
‹(*U*, *keep-watch L x2 x2a T*) ∈ *twl-st-heur-up''* $\mathcal{D}$ *r s K*›
for *x2d U x2a x2 L T*
using *that*
by *auto*

have *isa-find-unwatched-wl-st-heur-find-unwatched-wl-st*:
‹*isa-find-unwatched-wl-st-heur x' y'*

$\leq \Downarrow$ *Id* (*IsaSAT-Inner-Propagation.find-unwatched-wl-st x y*)›
if
find-unw: ‹*find-unwatched-wl-st-pre* (*x*, *y*)› **and**
xy: ‹((*x'*, *y'*), *x*, *y*) $\in$ *twl-st-heur* $\times_f$ *nat-rel*› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st x*) *x*›
for *x y x' y'*
proof −
have *n-d*: ‹*no-dup* (*get-trail-wl x*)›
using *xy* **unfolding** *twl-st-heur-def*
by *auto*
have *lits-xy*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st x*) (*mset* (*get-clauses-wl x* $\propto$ *y*))›
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-nth*)
subgoal
using *find-unw* **unfolding** *find-unwatched-wl-st-pre-def prod.simps*
by *auto*
subgoal using *lits* **.**
done

have *K*: ‹*find-unwatched-wl-st' x y* $\leq$ *IsaSAT-Inner-Propagation.find-unwatched-wl-st x y*›
unfolding *find-unwatched-wl-st'-def IsaSAT-Inner-Propagation.find-unwatched-wl-st-def*
apply (*cases x*)
apply *clarify*
apply (*rule order-trans*)
apply (*rule find-unwatched*[*of* - - ‹*all-atms-st x*›])
subgoal
using *n-d* **by** *simp*
subgoal
using *find-unw* **unfolding** *find-unwatched-wl-st-pre-def prod.simps*
by *auto*
subgoal
using *lits-xy* **by** *simp*
subgoal by *auto*
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule find-unwatched-wl-st-heur-find-unwatched-wl-s*[*THEN fref-to-Down-curry*,
OF that(*1*,*2*)])
by (*simp add*: *K*)
qed

have *set-conflict-wl'-rel*:
‹(*V*, *set-conflict-wl' x1f* (*keep-watch L x2 x2a T*)) $\in$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\Longrightarrow$
(*x2d*, *x2a*) $\in$ *nat-rel* $\Longrightarrow$
((*x2d* + *1*, *V*), *x2a* + *1*, *set-conflict-wl' x1f* (*keep-watch L x2 x2a T*))
$\in$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*›
for *V x1f L x2 x2a T x2d*
by *auto*

have *propagate-lit-wl-heur-final-rel*: ‹(*Sa*, *Sb*) $\in$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\Longrightarrow$
(*x2d*, *x2a*) $\in$ *nat-rel* $\Longrightarrow$
((*x2d* + *1*, *Sa*), *x2a* + *1*, *Sb*) $\in$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*›
for *V x1f L x2 x2a T x2d U x1g L' Sa Sb*
by *auto*

note *find-unw* = *find-unwatched-wl-st-heur-find-unwatched-wl-s*[*THEN fref-to-Down-curry*]
set-conflict-wl-heur-set-conflict-wl'[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry*, *unfolded comp-def*]

propagate-lit-wl-heur-propagate-lit-wl[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry3*, *unfolded comp-def*]
propagate-lit-wl-bin-heur-propagate-lit-wl-bin
[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry3*, *unfolded comp-def*]
update-clause-wl-heur-update-clause-wl[*of K r* $\mathcal{D}$ *s*, *THEN fref-to-Down-curry7*]
keep-watch-heur-keep-watch'[*of - - - - - - - -* $\mathcal{D}$ *r K s*]
update-blit-wl-heur-update-blit-wl[*of r* $\mathcal{D}$ *K s*, *THEN fref-to-Down-curry6*]
clause-not-marked-to-delete-rel[*THEN fref-to-Down-curry*]
keep-watch-skip
isa-find-unwatched-wl-st-heur-find-unwatched-wl-st
set-conflict-wl'-rel propagate-lit-wl-heur-final-rel

show *?thesis*
supply [[*goals-limit=1*]] *twl-st-heur'-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
unfolding *unit-propagation-inner-loop-body-wl-heur-def*
unit-propagation-inner-loop-body-wl-D-alt-def
uncurry-def find-unwatched-l-find-unwatched-wl-s 1 polarity-st-heur-def
watched-by-app-heur-def access-lit-in-clauses-heur-def
unfolding *set-conflict-wl'-alt-def*[*symmetric*]
clause-not-marked-to-delete-def[*symmetric*]
to-watcher-def watcher-of-def id-def

apply (*refine-rcg find-unw isa-save-pos*)
subgoal unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def twl-st-heur'-def*
unit-propagation-inner-loop-wl-loop-D-pre-def
by *fastforce*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d*
by (*rule watched-by-app-heur-pre*)
subgoal by (*rule keep-watch-heur-pre*)
subgoal by (*auto simp del*: *keep-watch-st-wl simp*: *twl-st-heur-state-simp*)
subgoal by *auto*
subgoal unfolding *unit-prop-body-wl-heur-inv-def* **by** *fastforce*
subgoal
by (*rule in-D0*)
subgoal by (*rule prop-fast-le*(*1*))
subgoal by (*rule prop-fast-le*(*2*))
subgoal
by (*rule polarity-eq*)
subgoal
by *simp*
subgoal
by *simp*
subgoal
by *simp*
subgoal
by (*rule bin-last-eq*)
subgoal by (*rule bin-polarity-eq*)
subgoal
by (*rule bin-set-conflict-wl-heur-pre*)
subgoal by (*rule bin-set-conflict-wl'-pre*)
subgoal by (*rule bin-conflict-rel*)
subgoal by *simp*
subgoal by *simp*
subgoal by (*rule bin-access-lit-in-clauses-heur-pre*)
subgoal

  by (*rule bin-propagate-lit-wl-heur-pre*)
subgoal by (*rule bin-propagate-lit-wl-pre*)
subgoal by (*rule bin-final-rel*)
subgoal by *simp*
subgoal by *simp*
subgoal
  by (*rule clause-not-marked-to-delete-heur-pre*)
subgoal
  by (*rule clause-not-marked-to-delete-heur-clause-not-marked-to-delete-iff*)
subgoal by *auto*
subgoal
  by (*rule access-lit-in-clauses-heur-pre0*)
subgoal
  by (*rule access-lit-in-clauses-heur-pre1i*)
subgoal
  by (*rule polarity-st-pre1i*)
subgoal
  by (*rule polarity-other-watched-lit*)
subgoal
  by (*rule update-blit-wl-heur-pre*)
subgoal
  by (*rule update-blit-wl-rel*)
subgoal
  by (*rule find-unwatched-wl-st-heur-pre*)
subgoal
  by (*rule find-unwatched-wl-st-pre*)
subgoal
  by (*rule isa-find-unwatched-wl-st-heur-pre*)
subgoal
  by (*rule isa-find-unwatched-wl-st-heur-lits*)
subgoal
  by (*rule unit-prop-body-wl-D-find-unwatched-heur-inv*)
subgoal
  by (*rule pol-other-lit-false*)
subgoal
  by (*rule set-conflict-wl-heur-pre*)
subgoal
  by (*rule unc-set-conflict-wl′-pre*)
subgoal
  by (*rule set-conflict-keep-watch-rel*)
subgoal
  by (*rule x2da-eq*)
subgoal
  by (*rule set-conflict-keep-watch-rel2*)
subgoal by (*rule propagate-lit-wl-heur-pre*)
subgoal by (*rule propagate-lit-wl-pre*)
subgoal by (*rule propagate-lit-wl-rel*)
subgoal
  by (*rule x2da-eq*)
subgoal
  by *force*
        apply *assumption+*
subgoal by *simp*
subgoal by (*rule access-lit-in-clauses-heur-pre3*)
subgoal
  by (*rule polarity-st-pre-unwatched*)

subgoal
by (*rule polarity-eq-unwatched*)
subgoal
by (*rule update-blit-wl-heur-pre-unw*)
subgoal
by (*rule update-blit-unw-rel*)
subgoal
by (*rule update-clause-wl-code-pre-unw*)
subgoal
by (*rule update-clause-wl-pre-unw*)
subgoal
by (*rule update-watched-unw-rel*)
done
qed

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'). $\exists$ $S_0{}'$ S. (S_0, $S_0{}'$) $\in$ *twl-st-heur* $\wedge$ (S', S) $\in$ *twl-st-heur* $\wedge$ *unit-propagation-inner-loop-wl-loop-D-inv* L (j, w, S) $\wedge$
L $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S) $\wedge$ *dom-m* (*get-clauses-wl* S) = *dom-m* (*get-clauses-wl* $S_0{}'$) $\wedge$
length (*get-clauses-wl-heur* S_0) = *length* (*get-clauses-wl-heur* S'))›

definition *unit-propagation-inner-loop-wl-loop-D-heur*
:: ‹*nat literal* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ (*nat* $\times$ *nat* $\times$ *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) $\leq$ *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 $\wedge$ *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)
$\in$ [λ(L, S). *length* (*watched-by* S L) $\leq$ r − *4* $\wedge$ L = K $\wedge$ *length* (*watched-by* S L) = s $\wedge$
length (*watched-by* S L) $\leq$ r]$_f$
nat-lit-lit-rel $\times_f$ *twl-st-heur-up*$''$ $\mathcal{D}$ r s K $\rightarrow$
$\langle$*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up*$''$ $\mathcal{D}$ r s $K\rangle$*nres-rel*›
proof −
have *unit-propagation-inner-loop-wl-loop-D-heur-inv*:
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv x2a x1a xa*›
if
‹(x, y) $\in$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-up*$''$ $\mathcal{D}$ r s K› **and**
‹y = (*x1*, *x2*)› **and**
‹x = (*x1a*, *x2a*)› **and**
‹(*xa*, x') $\in$ *nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up*$''$ $\mathcal{D}$ r s K› **and**
H: ‹*unit-propagation-inner-loop-wl-loop-D-inv x1* x'›
for x y *x1 x2 x1a x2a xa* x'
proof −

obtain *w S w′ S′ j j′* **where**
xa: ‹$xa = (j, w, S)$› **and** *x′*: ‹$x' = (j', w', S')$›
by (*cases xa*; *cases x′*) *auto*
show *?thesis*
unfolding *xa unit-propagation-inner-loop-wl-loop-D-heur-inv-def prod.case*
apply (*rule exI*[*of - x2*])
apply (*rule exI*[*of - S′*])
using *that xa x′ that*
unfolding *unit-propagation-inner-loop-wl-loop-D-inv-def twl-st-heur′-def*
by *auto*
qed

have *cond-eq*: ‹(*x1c* < *length* (*watched-by-int x2a x1a*) ∧ *get-conflict-wl-is-None-heur x2c*) =
(*x1e* < *length* (*watched-by x2 x1*) ∧ *get-conflict-wl x2e* = *None*)›
if
‹(*x*, *y*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹*x1* ∈# $\mathcal{L}_{all}$ (*all-atms-st x2*)› **and**
‹(*xa*, *x′*) ∈ *nat-rel* $\times_f$ (*nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*)› **and**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv*
x2a x1a xa› **and**
‹*unit-propagation-inner-loop-wl-loop-D-inv x1*
x′› **and**
st:
‹*x2b* = (*x1c*, *x2c*)›
‹*xa* = (*x1b*, *x2b*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x′* = (*x1d*, *x2d*)›
for *x y x1 x2 x1a x2a xa x′ x1b x2b x1c x2c x1d x2d*
x1e x2e
proof −
have ‹*get-conflict-wl-is-None-heur x2c* ⟷ *get-conflict-wl x2e* = *None*›
apply (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*,
of x2c x2e])
subgoal by *auto*
subgoal using *that* **unfolding** *twl-st-heur′-def* **by** *auto*
subgoal by (*auto simp*: *get-conflict-wl-is-None-def split*: *option.splits*)
done
moreover have
‹(*x1c* < *length* (*watched-by-int x2a x1a*)) ⟷
(*x1e* < *length* (*watched-by x2 x1*))›
using *that*(*1*−*5*) *st* **unfolding** *get-conflict-wl-is-None-heur-def*
by (*cases x2a*)
(*auto simp add*: *twl-st-heur′-def twl-st-heur-def map-fun-rel-def*
dest!: *multi-member-split*)
ultimately show *?thesis* **by** *blast*
qed
have *get-watched-wl-heur-pre*: ‹*nat-of-lit x1a* < *length* (*get-watched-wl-heur x2a*)›
if
‹(*x*, *y*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹*x1* ∈# $\mathcal{L}_{all}$ (*all-atms-st x2*)›
for *x y x1 x2 x1a x2a*
proof −

show *?thesis*
using *that*
by (*cases x2a*)
(*auto simp add*: *twl-st-heur′-def twl-st-heur-def map-fun-rel-def*
dest!: *multi-member-split*)
qed
note *H*[*refine*] = *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D*
[*THEN fref-to-Down-curry3*] *init*
show *?thesis*
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def*
unit-propagation-inner-loop-wl-loop-D-def uncurry-def
unit-propagation-inner-loop-wl-loop-D-inv-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*)
subgoal by (*rule get-watched-wl-heur-pre*)
subgoal by (*auto simp*: *twl-st-heur′-def twl-st-heur-state-simp-watched*)
subgoal by *auto*
subgoal by (*rule unit-propagation-inner-loop-wl-loop-D-heur-inv*)
subgoal by (*rule cond-eq*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

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* ≤ *length* (*W*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
RETURN (*M*, *N*, *D*, *Q*,
W[*nat-of-lit L* := *take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-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* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let n = *length* (*W*!(*nat-of-lit L*));
(*j*, *w*, *W*) ← $WHILE_T{}^{\lambda(j,\ w,\ W).\ j \le w \wedge w \le n \wedge nat\text{-}of\text{-}lit\ L < length\ W}$
(λ(*j*, *w*, *W*). *w* < *n*)
(λ(*j*, *w*, *W*). *do* {
ASSERT(*w* < *length* (*W*!(*nat-of-lit L*)));
RETURN (*j+1*, *w+1*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*j* := *W*!(*nat-of-lit L*)!*w*]])
})
(*j*, *w*, *W*);
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length W*);
let W = *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-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 −
obtain *M N D Q W oth* **where** *S*: ‹*S* = (*M*, *N*, *D*, *Q*, *W*, *oth*)›
by (*cases S*)
define *n* **where** *n*: ‹*n* = *length* (*W* ! *nat-of-lit L*)›
let *?R* = ‹*measure* (λ(*j'*::*nat*, *w'* :: *nat*, - :: (*nat* × *nat literal* × *bool*) *list list*). *length* (*W*!*nat-of-lit L*) − *w'*)›
define *I'* **where**
‹*I'* ≡ λ(*j'*, *w'*, *W'*). *length* (*W'* ! (*nat-of-lit L*)) = *length* (*W* ! (*nat-of-lit L*)) ∧ *j'* ≤ *w'* ∧ *w'* ≥ *w* ∧
w' − *w* = *j'* − *j* ∧ *j'* ≥ *j* ∧
drop w' (*W'* ! (*nat-of-lit L*)) = *drop w'* (*W* ! (*nat-of-lit L*)) ∧
w' ≤ *length* (*W'* ! (*nat-of-lit L*)) ∧
W'[*nat-of-lit L* := *take* (*j* + *w'* − *w*) (*W'* ! *nat-of-lit L*)] =
W[*nat-of-lit L* := *take* (*j* + *w'* − *w*) ((*take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))))]›

have *cut-watch-list-heur-alt-def*:
‹*cut-watch-list-heur j w L* =(λ(*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
ASSERT(*j* ≤ *length* (*W*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let W = *W*[*nat-of-lit L* := *take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))];
RETURN (*M*, *N*, *D*, *Q*, *W*, *oth*)
})›
unfolding *cut-watch-list-heur-def* **by** *auto*
have *REC*: ‹*ASSERT* (*x1k* < *length* (*x2k* ! *nat-of-lit L*)) ≫=
(λ-. *RETURN* (*x1j* + *1*, *x1k* + *1*, *x2k* [*nat-of-lit L* := (*x2k* ! *nat-of-lit L*) [*x1j* :=
x2k ! *nat-of-lit L* !
x1k]]))
≤ *SPEC* (λ*s'*. ∀ *x1 x2 x1a x2a*. *x2* = (*x1a*, *x2a*) ⟶ *s'* = (*x1*, *x2*) ⟶
(*x1* ≤ *x1a* ∧ *nat-of-lit L* < *length x2a*) ∧ *I' s'* ∧
(*s'*, *s*) ∈ *measure* (λ(*j'*, *w'*, -). *length* (*W* ! *nat-of-lit L*) − *w'*))›
if
‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
pre: ‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
I: ‹*case s of* (*j*, *w*, *W*) ⇒ *j* ≤ *w* ∧ *nat-of-lit L* < *length W*› **and**
I': ‹*I' s*› **and**
cond: ‹*case s of* (*j*, *w*, *W*) ⇒ *w* < *length* (*W* ! *nat-of-lit L*)› **and**
[*simp*]: ‹*x2* = (*x1k*, *x2k*)› **and**
[*simp*]: ‹*s* = (*x1j*, *x2*)›
for *s x1j x2 x1k x2k*
proof −
have [*simp*]: ‹*x1k* < *length* (*x2k* ! *nat-of-lit L*)› **and**
‹*length* (*W* ! *nat-of-lit L*) − *Suc x1k* < *length* (*W* ! *nat-of-lit L*) − *x1k*›
using *cond I I'* **unfolding** *I'-def* **by** *auto*
moreover have ‹*x1j* ≤ *x1k*› ‹*nat-of-lit L* < *length x2k*›
using *I I'* **unfolding** *I'-def* **by** *auto*
moreover have ‹*I'* (*Suc x1j*, *Suc x1k*, *x2k*
[*nat-of-lit L* := (*x2k* ! *nat-of-lit L*)[*x1j* := *x2k* ! *nat-of-lit L* ! *x1k*]])›
proof −
have *ball-leI*: ‹(⋀*x*. *x* < *A* ⟹ *P x*) ⟹ (∀ *x* < *A*. *P x*)› **for** *A P*
by *auto*
have *H*: ‹⋀*i*. *x2k*[*nat-of-lit L* := *take* (*j* + *x1k* − *w*) (*x2k* ! *nat-of-lit L*)] ! *i* = *W*

```
[nat-of-lit L :=
   take (min (j + x1k − w) j) (W ! nat-of-lit L) @
   take (j + x1k − (w + min (length (W ! nat-of-lit L)) j))
    (drop w (W ! nat-of-lit L))] ! i› and
     H': ‹x2k[nat-of-lit L := take (j + x1k − w) (x2k ! nat-of-lit L)] = W
     [nat-of-lit L :=
   take (min (j + x1k − w) j) (W ! nat-of-lit L) @
   take (j + x1k − (w + min (length (W ! nat-of-lit L)) j))
    (drop w (W ! nat-of-lit L))]› and
     ‹j < length (W ! nat-of-lit L)› and
     ‹(length (W ! nat-of-lit L) − w) ≥ (Suc x1k − w)› and
     ‹x1k ≥ w›
     ‹nat-of-lit L < length W› and
     ‹j + x1k − w = x1j› and
     ‹x1j − j = x1k − w› and
     ‹x1j < length (W ! nat-of-lit L)› and
     ‹length (x2k ! nat-of-lit L) = length (W ! nat-of-lit L)› and
     ‹drop x1k (x2k ! (nat-of-lit L)) = drop x1k (W ! (nat-of-lit L))›
     ‹x1j ≥ j›  and
     ‹w + x1j − j = x1k›
     using I I' pre cond unfolding I'-def by auto
    have
     [simp]: ‹min x1j j = j›
     using ‹x1j ≥ j› unfolding min-def by auto
    have ‹x2k[nat-of-lit L := take (Suc (j + x1k) − w) (x2k[nat-of-lit L := (x2k ! nat-of-lit L)
           [x1j := x2k ! nat-of-lit L ! x1k]] ! nat-of-lit L)] =
       W[nat-of-lit L := take j (W ! nat-of-lit L) @ take (Suc (j + x1k) − (w + min (length (W !
nat-of-lit L)) j))
          (drop w (W ! nat-of-lit L))]›
     using cond I ‹j < length (W ! nat-of-lit L)› and
      ‹(length (W ! nat-of-lit L) − w) ≥ (Suc x1k − w)› and
      ‹x1k ≥ w›
      ‹nat-of-lit L < length W›
      ‹j + x1k − w = x1j› ‹x1j < length (W ! nat-of-lit L)›
     apply (subst list-eq-iff-nth-eq)
     apply −
     apply (intro conjI ball-leI)
     subgoal using arg-cong[OF H', of length] by auto
     subgoal for k
      apply (cases ‹k ≠ nat-of-lit L›)
      subgoal using H[of k] by auto
      subgoal
        using H[of k] ‹x1j < length (W ! nat-of-lit L)›
         ‹length (x2k ! nat-of-lit L) = length (W ! nat-of-lit L)›
         arg-cong[OF ‹drop x1k (x2k ! (nat-of-lit L)) = drop x1k (W ! (nat-of-lit L))›,
            of ‹λxs. xs ! 0›] ‹x1j ≥ j›
        apply (cases ‹Suc x1j = length (W ! nat-of-lit L)›)
        apply (auto simp add: Suc-diff-le take-Suc-conv-app-nth ‹j + x1k − w = x1j›
           ‹x1j − j = x1k − w›[symmetric] ‹w + x1j − j = x1k›)
           apply (metis append.assoc le-neq-implies-less list-update-id nat-in-between-eq(1)
            not-less-eq take-Suc-conv-app-nth take-all)
         by (metis (no-types, lifting) ‹x1j < length (W ! nat-of-lit L)› append.assoc
           take-Suc-conv-app-nth take-update-last)
      done
     done
    then show ?thesis
```

unfolding *I'-def prod.case*
using *I I' cond* **unfolding** *I'-def* **by** (*auto simp*: *Cons-nth-drop-Suc*[*symmetric*])
qed
ultimately show *?thesis*
by *auto*
qed

have *step*: ‹(*s*, *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*) @ *drop w* (*W* ! *nat-of-lit L*)])
∈ {((*i*, *j*, *W'*), *W*). (*W'*[*nat-of-lit L* := *take i* (*W'* ! *nat-of-lit L*)], *W*) ∈ *Id* ∧
i ≤ *length* (*W'* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length W'* ∧
n = *length* (*W'* ! *nat-of-lit L*)}›
if
pre: ‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
‹*case s of* (*j*, *w*, *W*) ⇒ *j* ≤ *w* ∧ *nat-of-lit L* < *length W*› **and**
‹*I' s*› **and**
‹¬ (*case s of* (*j*, *w*, *W*) ⇒ *w* < *length* (*W* ! *nat-of-lit L*))›
for *s*
proof −
obtain *j' w' W'* **where** *s*: ‹*s* = (*j'*, *w'*, *W'*)› **by** (*cases s*)
have
‹¬ *w'* < *length* (*W'* ! *nat-of-lit L*)› **and**
‹*j* ≤ *length* (*W* ! *nat-of-lit L*)› **and**
‹*j'* ≤ *w'*› **and**
‹*nat-of-lit L* < *length W'*› **and**
[*simp*]: ‹*length* (*W'* ! *nat-of-lit L*) = *length* (*W* ! *nat-of-lit L*)› **and**
‹*j* ≤ *w*› **and**
‹*j'* ≤ *w'*› **and**
‹*nat-of-lit L* < *length W*› **and**
‹*w* ≤ *length* (*W* ! *nat-of-lit L*)› **and**
‹*w* ≤ *w'*› **and**
‹*w'* − *w* = *j'* − *j*› **and**
‹*j* ≤ *j'*› **and**
‹*drop w'* (*W'* ! *nat-of-lit L*) = *drop w'* (*W* ! *nat-of-lit L*)› **and**
‹*w'* ≤ *length* (*W'* ! *nat-of-lit L*)› **and**
L-le-W: ‹*nat-of-lit L* < *length W*› **and**
eq: ‹*W'*[*nat-of-lit L* := *take* (*j* + *w'* − *w*) (*W'* ! *nat-of-lit L*)] =
W[*nat-of-lit L* := *take* (*j* + *w'* − *w*) (*take j* (*W* ! *nat-of-lit L*) @ *drop w* (*W* ! *nat-of-lit L*))]›
using *that* **unfolding** *I'-def that prod.case s*
by *blast*+
then have
j-j': ‹*j* + *w'* − *w* = *j'*› **and**
j-le: ‹*j* + *w'* − *w* = *length* (*take j* (*W* ! *nat-of-lit L*) @ *drop w* (*W* ! *nat-of-lit L*))› **and**
w': ‹*w'* = *length* (*W* ! *nat-of-lit L*)›
by *auto*
have [*simp*]: ‹*length W* = *length W'*›
using *arg-cong*[*OF eq*, *of length*] **by** *auto*
show *?thesis*
using *eq* ‹*j* ≤ *w*› ‹*w* ≤ *length* (*W* ! *nat-of-lit L*)› ‹*j* ≤ *j'*› ‹*w'* − *w* = *j'* − *j*›
‹*w* ≤ *w'*› *w' L-le-W*
unfolding *j-j' j-le s S n*
by (*auto simp*: *min-def split*: *if-splits*)
qed

have *HHH*: ‹$X \leq RES\ (R^{-1}$ “ $\{S\}) \Longrightarrow X \leq \Downarrow R\ (RETURN\ S)$› **for** *X S R*
  by (*auto simp*: *RETURN-def conc-fun-RES*)

  show *?thesis*
    unfolding *cut-watch-list-heur2-def cut-watch-list-heur-alt-def prod.case S n*[*symmetric*]
    apply (*rewrite at* ‹*let - = n in -*› *Let-def*)
    apply (*refine-vcg WHILEIT-rule-stronger-inv-RES*[**where** $R = ?R$ **and**
      $I' = I'$ **and** Φ = ‹$\{((i, j, W'), W).\ (W'[nat\text{-}of\text{-}lit\ L := take\ i\ (W'\ !\ nat\text{-}of\text{-}lit\ L)],\ W) \in Id \wedge$
        $i \leq length\ (W'\ !\ nat\text{-}of\text{-}lit\ L) \wedge nat\text{-}of\text{-}lit\ L < length\ W' \wedge$
  $n = length\ (W'\ !\ nat\text{-}of\text{-}lit\ L)\}^{-1}$ “ -›] *HHH*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by (*auto simp*: *S*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
    subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
    subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
    subgoal unfolding *I′-def* **by** *auto*
    subgoal unfolding *I′-def* **by** *auto*
    subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
    subgoal using *REC* **by** (*auto simp*: *n*)
    subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
    subgoal for *s* **using** *step*[*of* ‹*s*›] **unfolding** *I′-def* **by** (*auto simp*: *n*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    done
qed

lemma *vdom-m-cut-watch-list*:
  ‹$set\ xs \subseteq set\ (W\ L) \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (W(L := xs))\ d \subseteq vdom\text{-}m\ \mathcal{A}\ W\ d$›
  by (*cases* ‹$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›)
    (*force simp*: *vdom-m-def split*: *if-splits*)+

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*:
  ‹$x \in vdom\text{-}m\ \mathcal{A}\ (W(L := xs))\ d \Longrightarrow set\ xs \subseteq set\ (W\ L) \Longrightarrow x \in vdom\text{-}m\ \mathcal{A}\ W\ d$›
  using *vdom-m-cut-watch-list*[*of xs W L*] **by** *auto*

lemma *cut-watch-list-heur-cut-watch-list-heur*:
  ‹(*uncurry3 cut-watch-list-heur*, *uncurry3 cut-watch-list*) ∈
  $[\lambda(((j, w), L), S).\ L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge j \leq length\ (watched\text{-}by\ S\ L)]_f$
    $nat\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}lit\text{-}lit\text{-}rel \times_f twl\text{-}st\text{-}heur''\ \mathcal{D}\ r \rightarrow \langle twl\text{-}st\text{-}heur''\ \mathcal{D}\ r\rangle nres\text{-}rel$›
    unfolding *cut-watch-list-heur-def cut-watch-list-def uncurry-def*
  apply (*intro frefI nres-relI*)
  apply *refine-vcg*
  subgoal
    by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
      twl-st-heur-def map-fun-rel-def)
  subgoal
    by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
      twl-st-heur-def map-fun-rel-def)

subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def)
subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def)
subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def vdom-m-cut-watch-list set-take-subset
      set-drop-subset dest!: *vdom-m-cut-watch-listD*
      dest!: *in-set-dropD in-set-takeD*)
done

definition *unit-propagation-inner-loop-wl-D-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*unit-propagation-inner-loop-wl-D-heur L* S_0 = *do* {
  (*j*, *w*, *S*) ← *unit-propagation-inner-loop-wl-loop-D-heur L* S_0;
  ASSERT(*length* (*watched-by-int S L*) ≤ *length* (*get-clauses-wl-heur* S_0) − *4*);
  S ← *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*) ∈
  [λ(*L*, *S*). *length*(*watched-by S L*) ≤ *r*−*4*]$_f$
  nat-lit-lit-rel $\times_f$ *twl-st-heur″* 𝒟 *r* → ⟨*twl-st-heur″* 𝒟 *r*⟩ *nres-rel*›
proof −
  have *length-le*: ‹*length* (*watched-by x2b x1b*) ≤ *r* − *4*› **and**
    length-eq: ‹*length* (*watched-by x2b x1b*) = *length* (*watched-by* (*snd y*) (*fst y*))› **and**
    eq: ‹*x1b* = *fst y*›
    if
      ‹*case y of* (*L*, *S*) ⇒ *length* (*watched-by S L*) ≤ *r*−*4*› **and**
      ‹(*x*, *y*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur″* 𝒟 *r*› **and**
      ‹*y* = (*x1*, *x2*)› **and**
      ‹*x* = (*x1a*, *x2a*)› **and**
      ‹(*x1*, *x2*) = (*x1b*, *x2b*)›
    for *x y x1 x2 x1a x2a x1b x2b*
      using *that* **by** *auto*
  show *?thesis*
    unfolding *unit-propagation-inner-loop-wl-D-heur-def*
      unit-propagation-inner-loop-wl-D-def uncurry-def
      apply (*intro frefI nres-relI*)
    apply (*refine-vcg cut-watch-list-heur-cut-watch-list-heur*[*of* 𝒟 *r*, *THEN fref-to-Down-curry3*]
unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D[*of r - -* 𝒟,
    THEN fref-to-Down-curry])

    apply (*rule length-le*; *assumption*)
    apply (*rule eq*; *assumption*)
    apply (*rule length-eq*; *assumption*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by (*auto simp*: *twl-st-heur′-def twl-st-heur-state-simp-watched*)
    apply (*rule order.trans*)
    apply (*rule cut-watch-list-heur2-cut-watch-list-heur*)
    apply (*subst Down-id-eq*)
    apply (*rule cut-watch-list-heur-cut-watch-list-heur*[*of* 𝒟, *THEN fref-to-Down-curry3*])

by *auto*
qed

definition *select-and-remove-from-literals-to-update-wl-heur*
:: ⟨*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × *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* ≤ *uint32-max*);
L ← *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* ⇒ *twl-st-wl-heur* ⇒ *bool*⟩
where
⟨*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S' ⟷
(∃ $S_0{}'$ *S*. (S_0, $S_0{}'$) ∈ *twl-st-heur* ∧ (S', *S*) ∈ *twl-st-heur* ∧
unit-propagation-outer-loop-wl-D-inv S ∧
dom-m (*get-clauses-wl S*) = *dom-m* (*get-clauses-wl* $S_0{}'$) ∧
length (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* S_0) ∧
isa-length-trail-pre (*get-trail-wl-heur* S'))⟩

definition *unit-propagation-outer-loop-wl-D-heur*
:: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*unit-propagation-outer-loop-wl-D-heur* S_0 =
$WHILE_T{}^{unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
(λ*S*. *literals-to-update-wl-heur S* < *isa-length-trail* (*get-trail-wl-heur S*))
(λ*S*. *do* {
ASSERT(*literals-to-update-wl-heur S* < *isa-length-trail* (*get-trail-wl-heur S*));
(S', *L*) ← *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* ≠ {#} ∧ *length* (*get-trail-wl y*) < *uint-max* ⟹
(*x*, *y*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r1* ⟹
select-and-remove-from-literals-to-update-wl-heur x
≤ ⇓{((*S*, *L*), (S', L')). ((*S*, *L*), (S', L')) ∈ *twl-st-heur''* $\mathcal{D}$*1 r1* $\times_f$ *nat-lit-lit-rel* ∧
S' = *set-literals-to-update-wl* (*literals-to-update-wl y* − {#*L*#}) *y* ∧
get-clauses-wl-heur S = *get-clauses-wl-heur x*}
(*select-and-remove-from-literals-to-update-wl y*)⟩
supply *RETURN-as-SPEC-refine*[*refine2*]
unfolding *select-and-remove-from-literals-to-update-wl-heur-def*
select-and-remove-from-literals-to-update-wl-def
apply (*refine-vcg*)
subgoal
by (*subst trail-pol-same-length*[*of* ⟨*get-trail-wl-heur x*⟩ ⟨*get-trail-wl y*⟩ ⟨*all-atms-st y*⟩])
(*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff*)
subgoal

apply (*subst* (*asm*) *trail-pol-same-length*[*of* ‹*get-trail-wl-heur x*› ‹*get-trail-wl y*› ‹*all-atms-st y*›])
apply (*auto simp*: *twl-st-heur-def twl-st-heur'-def*; *fail*)[]
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule isa-trail-nth-rev-trail-nth*[*THEN fref-to-Down-curry*, *unfolded comp-def RETURN-def*, *unfolded conc-fun-RES*, *of* ‹*get-trail-wl y*› - - - ‹*all-atms-st y*›])
apply ((*auto simp*: *twl-st-heur-def twl-st-heur'-def*; *fail*)+)[*2*]
subgoal for *z*
apply (*cases x*; *cases y*)
by (*simp-all add*: *Cons-nth-drop-Suc*[*symmetric*] *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff rev-trail-nth-def*)
done
done

lemma *unit-propagation-outer-loop-wl-D-heur-inv-length-trail-le*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur'' 𝒟 r*›
‹(*U*, *V*) ∈ *twl-st-heur'' 𝒟 r*› **and**
‹*literals-to-update-wl-heur U* < *isa-length-trail* (*get-trail-wl-heur U*)› **and**
‹*literals-to-update-wl V* ≠ {#}› **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* ≠ {#}› **and**
‹*literals-to-update-wl-heur U* < *isa-length-trail* (*get-trail-wl-heur U*)›
shows ‹*length* (*get-trail-wl V*) < *uint-max*›
proof −
have *bounded*: ‹*isasat-input-bounded* (*all-atms-st V*)›
using ‹(*U*, *V*) ∈ *twl-st-heur'' 𝒟 r*›
unfolding *twl-st-heur'-def twl-st-heur-def*
by (*auto simp del*: *isasat-input-bounded-def*)

obtain *T U b b'* **where**
VT: ‹(*V*, *T*) ∈ *state-wl-l b*› **and**
struct: ‹*twl-struct-invs U*› **and**
TU: ‹(*T*, *U*) ∈ *twl-st-l b'*› **and**
trail: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st V*) *V*›
using *assms*
unfolding *unit-propagation-outer-loop-wl-D-inv-def unit-propagation-outer-loop-wl-D-heur-inv-def unit-propagation-outer-loop-wl-inv-def unit-propagation-outer-loop-l-inv-def* **apply** −
apply *normalize-goal+*
by *blast*
then have ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st V*) (*get-trail-wl V*)›
by (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*)
moreover have ‹*no-dup* (*get-trail-wl V*)›
using *VT TU struct*
unfolding *twl-struct-invs-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by (*simp add*: *twl-st*)
ultimately show *?thesis*
using *literals-are-in-*$\mathcal{L}_{in}$*-trail-length-le-uint32-max*[*of* ‹*all-atms-st V*› ‹*get-trail-wl V*›, *OF bounded*]
by (*auto simp*: *uint32-max-def*)
qed

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 −
have ‹*correct-watching x'*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-D-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x' x2*)›
using *x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x'*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x'a* **unfolding** *st*
by (*cases x'*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x'*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def st*)
have *x2*: ‹$x2 \in\# \mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*) (*get-unit-clauses-wl x1*))›
using *x2 xb-x'a* **unfolding** *st*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur xa*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def st*
by (*cases x'*)
(*auto simp*: *twl-st-heur'-def twl-st-heur-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) $\subseteq$ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x'*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def st*)
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) $\subseteq$ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def st*
by (*cases x1*)
(*force simp*: *twl-st-heur'-def twl-st-heur-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) $\subseteq\#$ *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) $\subseteq \{4..<$ *length* (*get-clauses-wl-heur xa*)$\}$›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) $\leq$ *length* (*get-clauses-wl-heur xa*) $- 4$›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show *?thesis*

using *size-mset-mono*[*OF watched-incl*] *xb-x'a st'*
by *auto*
qed

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'' $\mathcal{D}$ *r* $\rightarrow_f$ ⟨*twl-st-heur''* $\mathcal{D}$ *r*⟩ *nres-rel*›
unfolding *unit-propagation-outer-loop-wl-D-heur-def*
unit-propagation-outer-loop-wl-D-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D[*of r* $\mathcal{D}$, *THEN fref-to-Down-curry*]
select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl
[*of* - - $\mathcal{D}$ *r*])
subgoal for *x y S T*
using *isa-length-trail-pre*[*of* ‹*get-trail-wl-heur S*› ‹*get-trail-wl T*› ‹*all-atms-st T*›] **apply** −
unfolding *unit-propagation-outer-loop-wl-D-heur-inv-def twl-st-heur'-def*
apply (*rule-tac x=y* **in** *exI*)
apply (*rule-tac x=T* **in** *exI*)
by (*auto 5 2 simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal for - - *x y*
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*, *of* - ‹*get-trail-wl y*› ‹*all-atms-st y*›])
(*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal by (*rule unit-propagation-outer-loop-wl-D-heur-inv-length-trail-le*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
subgoal for *x y xa x' xb x'a x1 x2 x1a x2a x1b x2b*
by (*rule-tac x=x* **and** *xa=xa* **and** $\mathcal{D}$=$\mathcal{D}$ **in** *outer-loop-length-watched-le-length-arena*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
done

lemma *twl-st-heur'D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge\mathcal{D}$. *f* ∈ *twl-st-heur'* $\mathcal{D}$ $\rightarrow_f$ ⟨*twl-st-heur'* $\mathcal{D}$⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
proof −
obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
using *assms* **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
apply (*rule weaken-⇓'*[*of* - ‹*twl-st-heur'* (*dom-m* (*get-clauses-wl y*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *watched-by-app-watched-by-app-heur*:
‹(*uncurry2* (*RETURN ooo watched-by-app-heur*), *uncurry2* (*RETURN ooo watched-by-app*)) ∈
[λ((*S*, *L*), *K*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧ *K* < *length* (*get-watched-wl S L*)]$_f$
twl-st-heur $\times_f$ *Id* $\times_f$ *Id* → ⟨*Id*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *watched-by-app-heur-def watched-by-app-def twl-st-heur-def map-fun-rel-def*)

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*)›
by (*auto split*: *option.splits*)

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*›
using *assms*
by (*auto simp*: *arena-is-valid-clause-idx-and-access-def isasat-fast-def*
dest!: *arena-lifting*(*10*))

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*))›
unfolding *clause-not-marked-to-delete-heur-def* **by** (*auto intro*!: *ext*)

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*›
:: ‹$nat\text{-}assn^k *_a nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
supply
[[*goals-limit*=*1*]]
if-splits[*split*]
length-rll-def[*simp*]
unfolding *isa-save-pos-def PR-CONST-def isasat-unbounded-assn-def*
by *sepref*

declare *isa-save-pos-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-save-pos-fast-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
supply
[[*goals-limit*=*1*]]
if-splits[*split*]
length-rll-def[*simp*]
unfolding *isa-save-pos-def PR-CONST-def isasat-bounded-assn-def*

by *sepref*

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-by-app-heur-pre*]$_a$
isasat-unbounded-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *nat-assn*^{k} $\rightarrow$ *watcher-assn*›
supply [[*goals-limit=1*]] *length-rll-def*[*simp*]
unfolding *watched-by-app-heur-alt-def isasat-unbounded-assn-def nth-rll-def*[*symmetric*]
watched-by-app-heur-pre-def
by *sepref*

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*›
supply [[*goals-limit=1*]] *length-rll-def*[*simp*]
unfolding *watched-by-app-heur-alt-def isasat-bounded-assn-def nth-rll-def*[*symmetric*]
watched-by-app-heur-pre-def
by *sepref*

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›
supply [[*goals-limit* = *1*]]
unfolding *isa-find-unset-lit-def isa-find-unwatched-between-def SET-FALSE-def*[*symmetric*]
apply (*rewrite* **in** ‹(*None*, -)› *annotate-assn*[**where** *A* = ‹*option-assn nat-assn*›])
apply (*rewrite* **in** ‹(*None*, -)› *annotate-assn*[**where** *A* = ‹*option-assn nat-assn*›])
apply (*rewrite* **in** ‹*if* ⌑ *then* - *else* -› *tri-bool-eq-def*[*symmetric*])
by *sepref*

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*›
supply [[*goals-limit* = *1*]]
fmap-length-rll-def[*simp*] *fmap-length-rll-u64-def*[*simp*]
get-saved-pos-code[*sepref-fr-rules*]
unfolding *isa-find-unwatched-wl-st-heur-def isasat-unbounded-assn-def PR-CONST-def*
find-unwatched-def fmap-rll-def[*symmetric*]
length-uint32-nat-def[*symmetric*] *isa-find-unwatched-def*

case-tri-bool-If find-unwatched-wl-st-heur-pre-def
fmap-rll-u64-def [*symmetric*]
MAX-LENGTH-SHORT-CLAUSE-def [*symmetric*]
apply (*subst isa-find-unset-lit-def* [*symmetric*])+
by *sepref*

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*›
:: ‹[λ((((*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›
supply [[*goals-limit* = *1*]]
unfolding *isa-find-unset-lit-def isa-find-unwatched-between-def SET-FALSE-def* [*symmetric*]
PR-CONST-def one-uint64-nat-def [*symmetric*]
apply (*rewrite* **in** ‹(*None*, -)› *annotate-assn*[**where** *A* = ‹*option-assn uint64-nat-assn*›])
apply (*rewrite* **in** ‹(*None*, -)› *annotate-assn*[**where** *A* = ‹*option-assn uint64-nat-assn*›])
apply (*rewrite* **in** ‹*if* ⌑ *then* - *else* -› *tri-bool-eq-def* [*symmetric*])
by *sepref*

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*›
:: ‹[(λ(*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*›
supply [[*goals-limit* = *1*]]
fmap-length-rll-def [*simp*]
uint64-of-uint32-conv-hnr[*sepref-fr-rules*] *isasat-fast-def* [*simp*]
unfolding *isa-find-unwatched-wl-st-heur-def PR-CONST-def*
find-unwatched-def fmap-rll-def [*symmetric*] *isasat-bounded-assn-def*
length-uint32-nat-def [*symmetric*] *isa-find-unwatched-def*
case-tri-bool-If find-unwatched-wl-st-heur-pre-def
fmap-rll-u64-def [*symmetric*]
MAX-LENGTH-SHORT-CLAUSE-def [*symmetric*]
two-uint64-nat-def [*symmetric*]
nat-of-uint64-conv-def
apply (*subst isa-find-unset-lit-def* [*symmetric*])+
by *sepref*

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*›
supply [[*goals-limit=1*]] *length-rll-def* [*simp*] *length-ll-def* [*simp*]
update-clause-wl-heur-pre-le-uint64 [*intro*!]

unfolding *update-clause-wl-heur-def isasat-unbounded-assn-def Array-List-Array.swap-ll-def*[*symmetric*]
fmap-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*] *fmap-swap-ll-def*[*symmetric*]
append-ll-def[*symmetric*] *update-clause-wl-code-pre-def*
fmap-rll-u64-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
fmap-swap-ll-def[*symmetric*]
PR-CONST-def
by *sepref*

declare *update-clause-wl-code.refine*[*sepref-fr-rules*]

sepref-definition *update-clause-wl-fast-code*
is ⟨*uncurry7 update-clause-wl-heur*⟩
:: ⟨[λ(((((((*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*⟩
supply [[*goals-limit=1*]] *length-rll-def*[*simp*] *length-ll-def*[*simp*]
update-clause-wl-heur-pre-le-uint64[*intro*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def Array-List-Array.swap-ll-def*[*symmetric*]
fmap-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*] *fmap-swap-ll-def*[*symmetric*]
append-ll-def[*symmetric*] *update-clause-wl-code-pre-def*
fmap-rll-u64-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
fmap-swap-ll-def[*symmetric*]
PR-CONST-def
to-watcher-fast-def[*symmetric*]
one-uint64-nat-def[*symmetric*]
by *sepref*

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*⟩
unfolding *PR-CONST-def propagate-lit-wl-heur-def isasat-unbounded-assn-def*
cons-trail-Propagated-def[*symmetric*]
supply [[*goals-limit=1*]]*length-rll-def*[*simp*] *length-ll-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-unbounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
save-phase-def
apply (*rewrite at* ⟨*count-decided-pol - =* ⌑⟩ *zero-uint32-nat-def*[*symmetric*])
by *sepref*

declare *propagate-lit-wl-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-lit-wl-fast-code*
is ⟨*uncurry3 propagate-lit-wl-heur*⟩
:: ⟨[λ(((*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*⟩

unfolding *PR-CONST-def propagate-lit-wl-heur-def isasat-unbounded-assn-def*
cons-trail-Propagated-def[*symmetric*]
supply [[*goals-limit=1*]] *length-rll-def*[*simp*] *length-ll-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
zero-uint64-nat-def[*symmetric*]
one-uint64-nat-def[*symmetric*]
save-phase-def
apply (*rewrite at* ‹*count-decided-pol* - = ¤› *zero-uint64-nat-def*)
apply (*rewrite at* ‹*count-decided-pol* - = ¤› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

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*›
unfolding *PR-CONST-def propagate-lit-wl-heur-def isasat-unbounded-assn-def*
cons-trail-Propagated-def[*symmetric*]
supply [[*goals-limit=1*]]*length-rll-def*[*simp*] *length-ll-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-unbounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
save-phase-def propagate-lit-wl-bin-heur-def
apply (*rewrite at* ‹*count-decided-pol* - = ¤› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

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›
unfolding *PR-CONST-def propagate-lit-wl-heur-def isasat-unbounded-assn-def*
cons-trail-Propagated-def[*symmetric*]
supply [[*goals-limit=1*]] *length-rll-def*[*simp*] *length-ll-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
zero-uint64-nat-def[*symmetric*]
one-uint64-nat-def[*symmetric*]
save-phase-def propagate-lit-wl-bin-heur-def
apply (*rewrite at* ‹*count-decided-pol* - = ¤› *zero-uint64-nat-def*)
apply (*rewrite at* ‹*count-decided-pol* - = ¤› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

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} $\rightarrow$ *bool-assn*›

supply [[*goals-limit=1*]]
unfolding *clause-not-marked-to-delete-heur-alt-def isasat-unbounded-assn-def*
clause-not-marked-to-delete-heur-pre-def
by *sepref*

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*)⟩
:: ⟨$[clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur\text{-}pre]_a$ $isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow bool\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *clause-not-marked-to-delete-heur-alt-def isasat-bounded-assn-def*
clause-not-marked-to-delete-heur-pre-def
by *sepref*

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*⟩
:: ⟨
$unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k *_a bool\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d$
$\rightarrow_a$
nat-assn ∗*a nat-assn* ∗*a isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
unfolding *update-blit-wl-heur-def isasat-unbounded-assn-def update-ll-def*[*symmetric*]
by *sepref*

declare *update-blit-wl-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *update-blit-wl-heur-fast-code*
is ⟨*uncurry6 update-blit-wl-heur*⟩
:: ⟨$[\lambda((((((\text{-}, \text{-}), \text{-}), \text{-}), C), i), S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a bool\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a$
$unat\text{-}lit\text{-}assn^k *_a$
$isasat\text{-}bounded\text{-}assn^d \rightarrow$
uint64-nat-assn ∗*a uint64-nat-assn* ∗*a isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
unfolding *update-blit-wl-heur-def isasat-bounded-assn-def update-ll-def*[*symmetric*]
to-watcher-fast-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
if-splits[*split*]
length-rll-def[*simp*] *length-ll-def*[*simp*]
supply *undefined-lit-polarity-st-iff*[*iff*]
unit-prop-body-wl-D-find-unwatched-heur-inv-def[*simp*]
update-raa-hnr[*sepref-fr-rules*]
unfolding *keep-watch-heur-def length-rll-def*[*symmetric*] *PR-CONST-def*
unfolding *fmap-rll-def*[*symmetric*] *isasat-unbounded-assn-def*

unfolding *fast-minus-def*[*symmetric*]
nth-rll-def[*symmetric*]
SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*]
update-ll-def[*symmetric*]
by *sepref*

declare *keep-watch-heur-code.refine*[*sepref-fr-rules*]

sepref-definition *keep-watch-heur-fast-code*
is ⟨*uncurry3 keep-watch-heur*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$⟩
supply
[[*goals-limit=1*]]
if-splits[*split*]
length-rll-def[*simp*] *length-ll-def*[*simp*]
supply *undefined-lit-polarity-st-iff*[*iff*]
unit-prop-body-wl-D-find-unwatched-heur-inv-def[*simp*]
update-raa-hnr[*sepref-fr-rules*]
unfolding *keep-watch-heur-def length-rll-def*[*symmetric*] *PR-CONST-def*
unfolding *fmap-rll-def*[*symmetric*] *isasat-bounded-assn-def*
unfolding *fast-minus-def*[*symmetric*]
nth-rll-def[*symmetric*]
SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*]
update-ll-def[*symmetric*]
by *sepref*

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*⟩
:: ⟨$[set\text{-}conflict\text{-}wl\text{-}heur\text{-}pre]_a$
$nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *set-conflict-wl-heur-def isasat-unbounded-assn-def*
set-conflict-wl-heur-pre-def PR-CONST-def
by *sepref*

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*⟩
:: ⟨$[\lambda(C, S).\ set\text{-}conflict\text{-}wl\text{-}heur\text{-}pre\ (C, S) \wedge$
$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *set-conflict-wl-heur-def isasat-bounded-assn-def*
set-conflict-wl-heur-pre-def PR-CONST-def
by *sepref*

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*›
  :: ‹$unat\text{-}lit\text{-}assn^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a nat\text{-}assn *a\ nat\text{-}assn *a$
isasat-unbounded-assn›
  supply
    [[*goals-limit=1*]]
    if-splits[*split*]
    length-rll-def[*simp*]
  supply *undefined-lit-polarity-st-iff*[*iff*]
    unit-prop-body-wl-D-find-unwatched-heur-inv-def[*simp*]
    unit-propagation-inner-loop-wl-loop-D-heur-inv0-def[*simp*]
    unit-propagation-inner-loop-wl-loop-D-inv-def[*simp*]
    image-image[*simp*]
  unfolding *unit-propagation-inner-loop-body-wl-heur-def length-rll-def*[*symmetric*] *PR-CONST-def*
  unfolding *fmap-rll-def*[*symmetric*]
  unfolding *fast-minus-def*[*symmetric*]
    SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
  by *sepref*

sepref-definition *unit-propagation-inner-loop-body-wl-fast-heur-code*
  is ‹*uncurry3 unit-propagation-inner-loop-body-wl-heur*›
  :: ‹$[\lambda((L, w), S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
      $unat\text{-}lit\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow$
      uint64-nat-assn ∗a uint64-nat-assn ∗a isasat-bounded-assn›
  supply [[*goals-limit=1*]]
    if-splits[*split*]
    length-rll-def[*simp*]
  supply *undefined-lit-polarity-st-iff*[*iff*]
    unit-prop-body-wl-D-find-unwatched-heur-inv-def[*simp*]
  unfolding *unit-propagation-inner-loop-body-wl-heur-def length-rll-def*[*symmetric*] *PR-CONST-def*
  unfolding *fmap-rll-def*[*symmetric*]
  unfolding *fast-minus-def*[*symmetric*]
    SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
  apply (*rewrite* **in** ‹*access-lit-in-clauses-heur - - ⌑*› *zero-uint64-nat-def*[*symmetric*])+
  apply (*rewrite* **in** ‹*If - ⌑ 1*› *zero-uint64-nat-def*[*symmetric*])
  apply (*rewrite* **in** ‹*If - zero-uint64-nat ⌑*› *one-uint64-nat-def*[*symmetric*])
  apply (*rewrite* **in** ‹*If - ⌑ 1*› *zero-uint64-nat-def*[*symmetric*])
  apply (*rewrite* **in** ‹*If - zero-uint64-nat ⌑*› *one-uint64-nat-def*[*symmetric*])
  apply (*rewrite* **in** ‹*fast-minus ⌑ -*› *one-uint64-nat-def*[*symmetric*])
  apply (*rewrite* **in** ‹*fast-minus ⌑ -*› *one-uint64-nat-def*[*symmetric*])
  unfolding *one-uint64-nat-def*[*symmetric*]
  by *sepref*

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* ⇒ *nat*› **where**
‹*size-conflict-wl S* = *size* (*the* (*get-conflict-wl S*))›

definition *size-conflict* :: ‹*nat clause option* ⇒ *nat*› **where**
‹*size-conflict D* = *size* (*the D*)›

definition *size-conflict-int* :: ‹*conflict-option-rel* ⇒ *nat*› **where**
‹*size-conflict-int* = (λ(-, *n*, -). *n*)›

definition *update-next-search* **where**
‹*update-next-search L* = (λ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
((*ns*, *m*, *fst-As*, *lst-As*, *L*), *to-remove*))›

definition *vmtf-enqueue-pre* **where**
‹*vmtf-enqueue-pre* =
(λ((*M*, *L*),(*ns*,*m*,*fst-As*,*lst-As*, *next-search*)). *L* < *length ns* ∧
(*fst-As* ≠ *None* ⟶ *the fst-As* < *length ns*) ∧
(*fst-As* ≠ *None* ⟶ *lst-As* ≠ *None*) ∧
m+*1* ≤ *uint64-max*)›

definition *isa-vmtf-enqueue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
ASSERT(*defined-atm-pol-pre M L*);
de ← *RETURN* (*defined-atm-pol M L*);
RETURN (*case fst-As of*
None ⇒(*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
(*if de 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 de then next-search else Some L*)))})›

lemma *vmtf-enqueue-alt-def*:
‹*RETURN ooo vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
let de = *defined-lit M* (*Pos L*);
RETURN (*case fst-As of*
None ⇒ (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
(*if de 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 de then next-search else Some L*)))})›
unfolding *vmtf-enqueue-def Let-def*
by (*auto intro*!: *ext*)

lemma *isa-vmtf-enqueue*:
‹(*uncurry2 isa-vmtf-enqueue*, *uncurry2* (*RETURN ooo vmtf-enqueue*)) ∈
$[\lambda((M, L), \text{-}).\ L \in\# \mathcal{A}]_f$ (*trail-pol* $\mathcal{A}$) $\times_f$ *nat-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
proof −
have *defined-atm-pol*: ‹(*defined-atm-pol x1g x2f*, *defined-lit x1a* (*Pos x2*)) ∈ *Id*›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*M*, *L*) ⇒ λ-. *L* ∈# $\mathcal{A}$) *xa*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ *Id*› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**

⟨$x2c = (x1d, x2d)$⟩ **and**
⟨$x2b = (x1c, x2c)$⟩ **and**
⟨$x2a = (x1b, x2b)$⟩ **and**
⟨$y = (x1, x2a)$⟩ **and**
⟨$x1f = (x1g, x2f)$⟩ **and**
⟨$x2j = (x1k, x2k)$⟩ **and**
⟨$x2i = (x1j, x2j)$⟩ **and**
⟨$x2h = (x1i, x2i)$⟩ **and**
⟨$x2g = (x1h, x2h)$⟩ **and**
⟨$x = (x1f, x2g)$⟩
for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h x1i x2i x1j x2j x1k x2k*
proof −
have [*simp*]: ⟨*defined-lit x1a* (*Pos x2*) ⟷ *defined-atm x1a x2*⟩
using *that* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *trail-pol-def defined-atm-def*)

show *?thesis*
using *undefined-atm-code*[*THEN fref-to-Down*, *unfolded uncurry-def*, *of* $\mathcal{A}$ ⟨(*x1a*, *x2*)⟩ ⟨(*x1g*, *x2f*)⟩]
that **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *RETURN-def*)
qed

show *?thesis*
unfolding *isa-vmtf-enqueue-def vmtf-enqueue-alt-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by (*rule defined-atm-pol-pre*) *auto*
apply (*rule defined-atm-pol*; *assumption*)
subgoal for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h x1i x2i x1j x2j x1k x2k*
by *auto*
done
qed

definition *partition-vmtf-nth* :: ⟨*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*⟩
where
⟨*partition-vmtf-nth ns* = *partition-main* (≤) (λ*n*. *stamp* (*ns* ! *n*))⟩

definition *partition-between-ref-vmtf* :: ⟨*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*⟩ **where**
⟨*partition-between-ref-vmtf ns* = *partition-between-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*))⟩

definition *quicksort-vmtf-nth* :: ⟨*nat-vmtf-node list* × *'c* ⇒ *nat list* ⇒ *nat list nres*⟩ **where**
⟨*quicksort-vmtf-nth* = (λ(*ns*, -). *full-quicksort-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*)))⟩

definition *quicksort-vmtf-nth-ref*:: ⟨*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat list nres*⟩ **where**
⟨*quicksort-vmtf-nth-ref ns a b c* =
quicksort-ref (≤) (λ*n*. *stamp* (*ns* ! *n*)) (*a*, *b*, *c*)⟩

lemma (**in** −) *partition-vmtf-nth-code-helper*:
assumes ⟨∀ *x*∈*set ba*. *x* < *length a*⟩ **and**
⟨*b* < *length ba*⟩ **and**
mset: ⟨*mset ba* = *mset a2'*⟩ **and**
⟨*a1'* < *length a2'*⟩
shows ⟨*a2'* ! *b* < *length a*⟩
using *nth-mem*[*of b a2'*] *mset-eq-setD*[*OF mset*] *mset-eq-length*[*OF mset*] *assms*
by (*auto simp del*: *nth-mem*)

lemma *partition-vmtf-nth-code-helper2*:
‹$ba < length\ b \Longrightarrow (bia,\ ba) \in uint32\text{-}nat\text{-}rel \Longrightarrow$
$(aa,\ (ba - bb)\ div\ 2) \in uint32\text{-}nat\text{-}rel \Longrightarrow$
$(ab,\ bb) \in uint32\text{-}nat\text{-}rel \Longrightarrow bb + (ba - bb)\ div\ 2 \le uint\text{-}max$›
apply (*auto simp*: *uint32-nat-rel-def br-def*)
by (*metis Nat.le-diff-conv2 ab-semigroup-add-class.add.commute diff-le-mono div-le-dividend le-trans nat-of-uint32-le-uint32-max*)

lemma *partition-vmtf-nth-code-helper3*:
‹$\forall x \in set\ b.\ x < length\ a \Longrightarrow$
$x'e < length\ a2' \Longrightarrow$
$mset\ a2' = mset\ b \Longrightarrow$
$a2'\ !\ x'e < length\ a$›
using *mset-eq-setD nth-mem* **by** *blast*

definition (**in** −) *isa-vmtf-en-dequeue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-en-dequeue* = (λ*M L vm*. *isa-vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *isa-vmtf-en-dequeue*:
‹(*uncurry2 isa-vmtf-en-dequeue*, *uncurry2* (*RETURN ooo vmtf-en-dequeue*)) ∈
$[\lambda((M,\ L),\ \text{-}).\ L \in\#\ \mathcal{A}]_f\ (trail\text{-}pol\ \mathcal{A}) \times_f nat\text{-}rel \times_f Id \rightarrow \langle Id\rangle nres\text{-}rel$›
unfolding *isa-vmtf-en-dequeue-def vmtf-en-dequeue-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b ba ae af ag ah bb ai bc aj ak al am bd*
by (*rule order.trans*,
rule isa-vmtf-enqueue[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*,
of ai bc ‹*vmtf-dequeue bc* (*aj*, *ak*, *al*, *am*, *bd*)›])
auto
done

definition *isa-vmtf-en-dequeue-pre* :: ‹(*trail-pol* × *nat*) × *vmtf* ⇒ *bool*› **where**
‹*isa-vmtf-en-dequeue-pre* = (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*)›

lemma *isa-vmtf-en-dequeue-preD*:
assumes ‹*isa-vmtf-en-dequeue-pre* ((*M*, *ah*), *a*, *aa*, *ab*, *ac*, *b*)›
shows ‹*ah* < *length a*› **and** ‹*vmtf-dequeue-pre* (*ah*, *a*)›
using *assms* **by** (*auto simp*: *isa-vmtf-en-dequeue-pre-def*)

lemma *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*:
‹*isa-vmtf-en-dequeue-pre* ((*M*, *L*), *a*, *st*, *fst-As*, *lst-As*, *next-search*) ⟹
vmtf-enqueue-pre ((*M*, *L*), *vmtf-dequeue L* (*a*, *st*, *fst-As*, *lst-As*, *next-search*))›
unfolding *vmtf-enqueue-pre-def*
apply *clarify*
apply (*intro conjI*)
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def ns-vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits*)[]
subgoal

by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
done

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-quicksort-ref R h*, *reorder-list vm*) $\in$ ⟨*Id*⟩*list-rel* $\to_f$ ⟨*Id*⟩ *nres-rel*›
proof −
show *?thesis*
apply (*intro frefI nres-relI*)
apply (*rule full-quicksort-ref-full-quicksort*[*THEN fref-to-Down*, *THEN order-trans*])
using *assms* **apply** *fast*
using *assms* **apply** *fast*
apply *fast*
apply *assumption*
using *assms*
apply (*auto 5 5 simp*: *reorder-list-def intro*!: *full-quicksort-correct*[*THEN order-trans*])
done
qed

lemma *quicksort-vmtf-nth-reorder*:
‹(*uncurry quicksort-vmtf-nth*, *uncurry reorder-list*) $\in$
Id $\times_r$ ⟨*Id*⟩*list-rel* $\to_f$ ⟨*Id*⟩ *nres-rel*›
apply (*intro WB-More-Refinement.frefI nres-relI*)
subgoal for *x y*
using *insert-sort-reorder-list*[*unfolded fref-def nres-rel-def*, *of*
‹($\le$)› ‹(λ*n*. *stamp* (*fst* (*fst y*) ! *n*) :: *nat*)› ‹*fst y*›]
by (*cases x*, *cases y*)
(*fastforce simp*: *quicksort-vmtf-nth-def uncurry-def WB-More-Refinement.fref-def*)
done

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) $\in$
nat-rel $\times_r$ *atoms-hash-rel* $\mathcal{A}$ $\to_f$ ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *atoms-hash-del-def atoms-hash-rel-def*)

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›
by (*auto simp*: *current-stamp-def intro*!: *ext*)

lemma *vmtf-rescale-alt-def*:
‹*vmtf-rescale* = (λ(*ns*, *m*, *fst-As*, *lst-As* :: *nat*, *next-search*). *do* {
(*ns*, *m*, -) $\leftarrow$ $WHILE_T{}^{\lambda\text{-}.\ True}$

(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))
})›
unfolding update-stamp.simps Let-def vmtf-rescale-def **by** auto

definition isa-vmtf-flush-int :: ‹trail-pol ⇒ - ⇒ - nres› **where**
‹isa-vmtf-flush-int = (λM (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$$^{λ(i, vm′, h). i ≤ length to-remove′ ∧ fst (snd vm′) = i + fst (snd vm) ∧ (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-list to-remove′, h))
})›

lemma isa-vmtf-flush-int:
‹(uncurry isa-vmtf-flush-int, uncurry (vmtf-flush-int 𝒜)) ∈ trail-pol 𝒜 $\times_f$ Id $\rightarrow_f$ ⟨Id⟩nres-rel›
proof −
have vmtf-flush-int-alt-def:
‹vmtf-flush-int $\mathcal{A}_{in}$ = (λM (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′ + fst (snd vm) ≥ uint64-max
then vmtf-rescale vm else RETURN vm);
ASSERT(length to-remove′ + fst (snd vm) ≤ uint64-max);
(-, vm, h) ← WHILE$_T$$^{λ(i, vm′, h). i ≤ length to-remove′ ∧ fst (snd vm′) = i + fst (snd vm) ∧ (i < length to-remove′ −}$
(λ(i, vm, h). i < length to-remove′)
(λ(i, vm, h). do {
ASSERT(i < length to-remove′);

$ASSERT(to\text{-}remove'!i \in\# \mathcal{A}_{in})$;
$ASSERT(atoms\text{-}hash\text{-}del\text{-}pre\ (to\text{-}remove'!i)\ h)$;
$vm \leftarrow RETURN(vmtf\text{-}en\text{-}dequeue\ M\ (to\text{-}remove'!i)\ vm)$;
$RETURN\ (i+1,\ vm,\ atoms\text{-}hash\text{-}del\ (to\text{-}remove'!i)\ h)\})$
$(0,\ vm,\ h)$;
$RETURN\ (vm,\ (emptied\text{-}list\ to\text{-}remove',\ h))$
})› **for** $\mathcal{A}_{in}$
unfolding *vmtf-flush-int-def*
by *auto*

have *reorder-list*: ‹*reorder-list x1d x1e*
$\leq \Downarrow Id$
(*reorder-list x1a x1b*)›
if
‹$(x,\ y) \in trail\text{-}pol\ \mathcal{A} \times_f Id$› **and** ‹$x2a = (x1b,\ x2b)$› **and**
‹$x2 = (x1a,\ x2a)$› **and**
‹$y = (x1,\ x2)$› **and**
‹$x2d = (x1e,\ x2e)$› **and**
‹$x2c = (x1d,\ x2d)$› **and**
‹$x = (x1c,\ x2c)$› **and**
‹$\forall x \in set\ x1b.\ x < length\ (fst\ x1a)$› **and**
‹$length\ x1b \leq uint\text{-}max$› **and**
‹$\forall x \in set\ x1e.\ x < length\ (fst\ x1d)$› **and**
‹$length\ x1e \leq uint\text{-}max$›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *that* **by** *auto*

have *vmtf-rescale*: ‹*vmtf-rescale x1d*
$\leq \Downarrow Id$
(*vmtf-rescale x1a*)›
if
‹*True*› **and**
‹$(x,\ y) \in trail\text{-}pol\ \mathcal{A} \times_f Id$› **and** ‹$x2a = (x1b,\ x2b)$› **and**
‹$x2 = (x1a,\ x2a)$› **and**
‹$y = (x1,\ x2)$› **and**
‹$x2d = (x1e,\ x2e)$› **and**
‹$x2c = (x1d,\ x2d)$› **and**
‹$x = (x1c,\ x2c)$› **and**
‹$\forall x \in set\ x1b.\ x < length\ (fst\ x1a)$› **and**
‹$length\ x1b \leq uint\text{-}max$› **and**
‹$\forall x \in set\ x1e.\ x < length\ (fst\ x1d)$› **and**
‹$length\ x1e \leq uint\text{-}max$› **and**
‹$(to\text{-}remove',\ to\text{-}remove'a) \in Id$› **and**
‹$length\ to\text{-}remove'a \leq uint\text{-}max$› **and**
‹$length\ to\text{-}remove' \leq uint\text{-}max$› **and**
‹$uint64\text{-}max \leq length\ to\text{-}remove'a + fst\ (snd\ x1a)$›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove′ to-remove′a*
using *that* **by** *auto*

have *loop-rel*: ‹$((0,\ vm,\ x2e),\ 0,\ vma,\ x2b) \in Id$›
if
‹$(x,\ y) \in trail\text{-}pol\ \mathcal{A} \times_f Id$› **and**
‹$x2a = (x1b,\ x2b)$› **and**
‹$x2 = (x1a,\ x2a)$› **and**
‹$y = (x1,\ x2)$› **and**
‹$x2d = (x1e,\ x2e)$› **and**

⟨*x2c* = (*x1d*, *x2d*)⟩ **and**
⟨*x* = (*x1c*, *x2c*)⟩ **and**
⟨∀ *x*∈*set x1b*. *x* < *length* (*fst x1a*)⟩ **and**
⟨*length x1b* ≤ *uint-max*⟩ **and**
⟨∀ *x*∈*set x1e*. *x* < *length* (*fst x1d*)⟩ **and**
⟨*length x1e* ≤ *uint-max*⟩ **and**
⟨(*to-remove'*, *to-remove'a*) ∈ *Id*⟩ **and**
⟨*length to-remove'a* ≤ *uint-max*⟩ **and**
⟨*length to-remove'* ≤ *uint-max*⟩ **and**
⟨(*vm*, *vma*) ∈ *Id*⟩ **and**
⟨*length to-remove'a* + *fst* (*snd vma*) ≤ *uint64-max*⟩
⟨*case* (*0*, *vma*, *x2b*) *of*
(*i*, *vm'*, *h*) ⇒
i ≤ *length to-remove'a* ∧
fst (*snd vm'*) = *i* + *fst* (*snd vma*) ∧
(*i* < *length to-remove'a* ⟶
vmtf-en-dequeue-pre 𝒜 ((*x1*, *to-remove'a* ! *i*), *vm'*))⟩
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove' to-remove'a vm vma*
using *that* **by** *auto*
have *isa-vmtf-en-dequeue-pre*:
⟨*vmtf-en-dequeue-pre* 𝒜 ((*M*, *L*), *x*) ⟹ *isa-vmtf-en-dequeue-pre* ((*M'*, *L*), *x*)⟩ **for** *x M M' L*
unfolding *vmtf-en-dequeue-pre-def isa-vmtf-en-dequeue-pre-def*
by *auto*
have *isa-vmtf-en-dequeue*: ⟨*isa-vmtf-en-dequeue x1c* (*to-remove'* ! *x1h*) *x1i*
≤ *SPEC*
(λ*c*. (*c*, *vmtf-en-dequeue x1* (*to-remove'a* ! *x1f*) *x1g*)
∈ *Id*)⟩
if
⟨(*x*, *y*) ∈ *trail-pol* 𝒜 $\times_f$ *Id*⟩ **and**
⟨∀ *x*∈*set x1b*. *x* < *length* (*fst x1a*)⟩ **and**
⟨*length x1b* ≤ *uint-max*⟩ **and**
⟨∀ *x*∈*set x1e*. *x* < *length* (*fst x1d*)⟩ **and**
⟨*length x1e* ≤ *uint-max*⟩ **and**
⟨*length to-remove'a* ≤ *uint-max*⟩ **and**
⟨*length to-remove'* ≤ *uint-max*⟩ **and**
⟨*length to-remove'a* + *fst* (*snd vma*) ≤ *uint64-max*⟩ **and**
⟨*case xa of* (*i*, *vm*, *h*) ⇒ *i* < *length to-remove'*⟩ **and**
⟨*case x' of* (*i*, *vm*, *h*) ⇒ *i* < *length to-remove'a*⟩ **and**
⟨*case xa of*
(*i*, *vm'*, *h*) ⇒
i ≤ *length to-remove'* ∧
fst (*snd vm'*) = *i* + *fst* (*snd vm*) ∧
(*i* < *length to-remove'* ⟶
isa-vmtf-en-dequeue-pre ((*x1c*, *to-remove'* ! *i*), *vm'*))⟩ **and**
⟨*case x' of*
(*i*, *vm'*, *h*) ⇒
i ≤ *length to-remove'a* ∧
fst (*snd vm'*) = *i* + *fst* (*snd vma*) ∧
(*i* < *length to-remove'a* ⟶
vmtf-en-dequeue-pre 𝒜 ((*x1*, *to-remove'a* ! *i*), *vm'*))⟩ **and**
⟨*isa-vmtf-en-dequeue-pre* ((*x1c*, *to-remove'* ! *x1h*), *x1i*)⟩ **and**
⟨*x1f* < *length to-remove'a*⟩ **and**
⟨*to-remove'a* ! *x1f* ∈# 𝒜⟩ **and**
⟨*x1h* < *length to-remove'*⟩ **and**
⟨*x2a* = (*x1b*, *x2b*)⟩ **and**

⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*y* = (*x1*, *x2*)⟩ **and**
⟨*x* = (*x1c*, *x2c*)⟩ **and**
⟨*x2d* = (*x1e*, *x2e*)⟩ **and**
⟨*x2c* = (*x1d*, *x2d*)⟩ **and**
⟨*x2f* = (*x1g*, *x2g*)⟩ **and**
⟨*x′* = (*x1f*, *x2f*)⟩ **and**
⟨*x2h* = (*x1i*, *x2i*)⟩ **and**
⟨*xa* = (*x1h*, *x2h*)⟩ **and**
⟨(*to-remove′*, *to-remove′a*) ∈ *Id*⟩ **and**
⟨(*xa*, *x′*) ∈ *Id*⟩ **and**
⟨(*vm*, *vma*) ∈ *Id*⟩
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove′ to-remove′a vm vma xa x′ x1f x2f x1g x2g x1h x2h x1i* **and** *x2i* :: ⟨*bool list*⟩
using *isa-vmtf-en-dequeue*[*of* 𝒜, *THEN fref-to-Down-curry2*, *of x1* ⟨*to-remove′a* ! *x1f*⟩ *x1g x1c* ⟨*to-remove′* ! *x1h*⟩ *x1i*] *that*
by (*auto simp*: *RETURN-def*)

show *?thesis*
unfolding *isa-vmtf-flush-int-def uncurry-def vmtf-flush-int-alt-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule reorder-list*; *assumption*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule vmtf-rescale*; *assumption*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule loop-rel*; *assumption*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto intro*!: *isa-vmtf-en-dequeue-pre*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by *auto*
apply (*rule isa-vmtf-en-dequeue*; *assumption*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove′ to-remove′a vm vma xa x′ x1f x2f x1g x2g x1h x2h x1i x2i vmb vmc*
by *auto*
subgoal
by *auto*
subgoal

by *auto*
done
qed

definition *atms-hash-insert-pre* :: ⟨*nat* ⇒ *nat list* × *bool list* ⇒ *bool*⟩ **where**
⟨*atms-hash-insert-pre i* = $(\lambda(n, xs).\ i < length\ xs \wedge (\neg xs!i \longrightarrow length\ n < uint32\text{-}max))$⟩

definition *atoms-hash-insert* :: ⟨*nat* ⇒ *nat list* × *bool list* ⇒ (*nat list* × *bool list*)⟩ **where**
⟨*atoms-hash-insert i* = $(\lambda(n, xs).$ *if xs* ! *i then* (n, xs) *else* $(n\ @\ [i], xs[i := True]))$⟩

lemma *bounded-included-le*:
assumes *bounded*: ⟨*isasat-input-bounded* $\mathcal{A}$⟩ **and** ⟨*distinct n*⟩ **and** ⟨*set n* ⊆ *set-mset* $\mathcal{A}$ ⟩ **shows** ⟨*length n* < *uint32-max*⟩
proof −
have *lits*: ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* '# *mset n*)⟩ **and**
dist: ⟨*distinct n*⟩
using *assms*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def distinct-atoms-rel-alt-def inj-on-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
have *dist*: ⟨*distinct-mset* (*Pos* '# *mset n*)⟩
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ⟨*auto simp*: *inj-on-def*⟩)
have *tauto*: ⟨¬ *tautology* (*poss* (*mset n*))⟩
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)
qed

lemma *atms-hash-insert-pre*:
assumes ⟨$L \in\#\ \mathcal{A}$⟩ **and** ⟨$(x, x') \in$ *distinct-atoms-rel* $\mathcal{A}$⟩ **and** ⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows ⟨*atms-hash-insert-pre L x*⟩
using *assms bounded-included-le*[*OF assms*(*3*), *of* ⟨*L* # *fst x*⟩]
by (*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def atms-hash-insert-pre-def*)

lemma *atoms-hash-del-op-set-insert*:
⟨(*uncurry* (*RETURN oo atoms-hash-insert*),
uncurry (*RETURN oo insert*)) ∈
$[\lambda(i, xs).\ i \in\#\ \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}]_f$
nat-rel $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$ → ⟨*distinct-atoms-rel* $\mathcal{A}_{in}$⟩*nres-rel*⟩
by (*intro frefI nres-relI*)
(*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def intro*!: *ASSERT-leI*)

definition (**in** −) *atoms-hash-set-member* **where**
⟨*atoms-hash-set-member i xs* = *do* {*ASSERT*(*i* < *length xs*); *RETURN* (*xs* ! *i*)}⟩

definition *isa-vmtf-mark-to-rescore*
:: ⟨*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*⟩
where
⟨*isa-vmtf-mark-to-rescore L* = (λ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).

((*ns*, *m*, *fst-As*, *next-search*), *atoms-hash-insert L to-remove*))›

definition *isa-vmtf-mark-to-rescore-pre* **where**
‹*isa-vmtf-mark-to-rescore-pre* = (λ*L* ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
atms-hash-insert-pre L to-remove)›

lemma *isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*:
‹(*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore*), *uncurry* (*RETURN oo vmtf-mark-to-rescore*)) $\in$
[λ(*L*, *vm*). *L*$\in$# $\mathcal{A}_{in}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *Id* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$) $\rightarrow$
$\langle$*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}\rangle$*nres-rel*›
unfolding *isa-vmtf-mark-to-rescore-def vmtf-mark-to-rescore-def*
by (*intro frefI nres-relI*)
(*auto intro*!: *atoms-hash-del-op-set-insert*[*THEN fref-to-Down-unRET-uncurry*])

definition (**in** $-$) *isa-vmtf-unset* :: ‹*nat* $\Rightarrow$ *isa-vmtf-remove-int* $\Rightarrow$ *isa-vmtf-remove-int*› **where**
‹*isa-vmtf-unset* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(*if next-search* = *None* $\vee$ *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* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
L < *length ns* $\wedge$ (*next-search* $\neq$ *None* $\longrightarrow$ *the next-search* < *length ns*))›

lemma *vmtf-unset-pre-vmtf*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in$# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
using *assms*
by (*auto simp*: *vmtf-def vmtf-unset-pre-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *vmtf-unset-pre*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in$# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
using *assms vmtf-unset-pre-vmtf*[*of ns m fst-As lst-As next-search - $\mathcal{A}$ M L*]
unfolding *isa-vmtf-def vmtf-unset-pre-def*
by *auto*

lemma *vmtf-unset-pre′*:
assumes
‹*vm* $\in$ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in$# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L vm*›
using *assms* **by** (*cases vm*) (*auto dest*: *vmtf-unset-pre*)

definition *isa-vmtf-mark-to-rescore-and-unset* :: ‹*nat* $\Rightarrow$ *isa-vmtf-remove-int* $\Rightarrow$ *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*) $\wedge$

atms-hash-insert-pre L tor)›

definition *get-pos-of-level-in-trail* **where**
‹*get-pos-of-level-in-trail* M_0 *lev* =
SPEC(λ*i*. *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*)›
by (*induction arbitrary*: *c rule*: *control-stack.induct*) (*auto simp*: *nth-append*
dest: *control-stack-le-length-M*)

lemma *control-stack-distinct*:
‹*control-stack cs M* ⟹ *distinct cs*›
by (*induction rule*: *control-stack.induct*) (*auto simp*: *nth-append*
dest: *control-stack-le-length-M*)

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-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) = *Suc i*›
proof −
define *L* **where** ‹*L* = *rev M* ! (*cs* ! *i*)›
have *csi*: ‹*cs* ! *i* < *length M*›
using *cs i* **by** (*auto intro*: *control-stack-le-length-M*)
then have *L-M*: ‹*L* ∈ *set M*›
using *nth-mem*[*of* ‹*cs* !*i*› ‹*rev M*›] **unfolding** *L-def* **by** (*auto simp del*: *nth-mem*)
have *dec-L*: ‹*is-decided L*›
using *control-stack-is-decided*[*OF cs*] *i* **unfolding** *L-def* **by** *auto*
then have ‹*rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
using *control-stack-rev-get-lev*[*OF cs n-d L-M*] **by** *auto*
moreover have ‹*distinct M*›
using *no-dup-distinct*[*OF n-d*] **unfolding** *mset-map*[*symmetric*] *distinct-mset-mset-distinct*
by (*rule distinct-mapI*)

moreover have *lev0*: ‹*get-level M* (*lit-of L*) ≥ *1*›
using *split-list*[*OF L-M*] *n-d dec-L* **by** (*auto simp*: *get-level-append-if*)
moreover have ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*) < *length M*›
using *control-stack-le-length-M*[*OF cs*,
of ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*)›, *OF nth-mem*]
control-stack-length-count-dec[*OF cs*] *count-decided-ge-get-level*[*of M*
‹*lit-of* (*rev M* ! (*cs* ! *i*))›] *lev0*
by (*auto simp*: *L-def*)
ultimately have ‹*cs* ! (*get-level M* (*lit-of L*) − *1*) = *cs* ! *i*›
using *nth-eq-iff-index-eq*[*of* ‹*rev M*›] *csi* **unfolding** *L-def* **by** *auto*
then have ‹*i* = *get-level M* (*lit-of L*) − *1*›
using *nth-eq-iff-index-eq*[*OF control-stack-distinct*[*OF cs*], *of i* ‹*get-level M* (*lit-of L*) − *1*›]
i lev0 count-decided-ge-get-level[*of M* ‹*lit-of* (*rev M* ! (*cs* ! *i*))›]
control-stack-length-count-dec[*OF cs*]

by (*auto simp*: *L-def*)
then show *?thesis* **using** *lev0* **unfolding** *L-def*[*symmetric*] **by** *auto*
qed

definition *get-pos-of-level-in-trail-pre* **where**
‹*get-pos-of-level-in-trail-pre* = (λ(*M*, *lev*). *lev* < *count-decided M*)›

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
[*get-pos-of-level-in-trail-pre*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ $\times_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-no-CS-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-no-CS-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
[*get-pos-of-level-in-trail-pre*]$_f$ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *lit-of-last-trail-pol-lit-of-last-trail-no-CS*:
‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) ∈
[λ*S*. *S* ≠ []]$_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
by (*auto simp*: *lit-of-hd-trail-def trail-pol-no-CS-def lit-of-last-trail-pol-def*
ann-lits-split-reasons-def hd-map rev-map[*symmetric*]
intro!: *frefI nres-relI*)

lemma *size-conflict-int-size-conflict*:
‹(*RETURN o size-conflict-int*, *RETURN o size-conflict*) ∈ [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel*
$\mathcal{A}$ →
⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *size-conflict-int-def size-conflict-def option-lookup-clause-rel-def*
lookup-clause-rel-def)

definition *rescore-clause*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *vmtf-remove-int* ⇒ *phase-saver* ⇒

$(vmtf\text{-}remove\text{-}int \times phase\text{-}saver)\ nres$›
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
‹$find\text{-}decomp\text{-}wl\text{-}imp\ \mathcal{A} = (\lambda M_0\ lev\ vm.\ do\ \{$
$let\ k = count\text{-}decided\ M_0;$
$let\ M_0 = trail\text{-}conv\text{-}to\text{-}no\text{-}CS\ M_0;$
$let\ n = length\ M_0;$
$pos \leftarrow get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\ M_0\ lev;$
$ASSERT((n - pos) \leq uint32\text{-}max);$
$let\ target = n - pos;$
$(\text{-}, M, vm') \leftarrow$
$WHILE_T{}^{\lambda(j, M, vm').\ j \leq target \wedge\quad M = drop\ j\ M_0 \wedge target \leq length\ M_0 \wedge\quad vm' \in vmtf\ \mathcal{A}\ M \wedge literals\text{-}ar}$
$(\lambda(j, M, vm).\ j < target)$
$(\lambda(j, M, vm).\ do\ \{$
$ASSERT(M \neq []);$
$ASSERT(Suc\ j \leq uint32\text{-}max);$
$let\ L = atm\text{-}of\ (lit\text{-}of\text{-}hd\text{-}trail\ M);$
$ASSERT(L \in\#\ \mathcal{A});$
$RETURN\ (j + one\text{-}uint32\text{-}nat, tl\ M, vmtf\text{-}unset\ L\ vm)$
$\})$
$(zero\text{-}uint32\text{-}nat, M_0, vm);$
$ASSERT(lev = count\text{-}decided\ M);$
$let\ M = trail\text{-}conv\text{-}back\ lev\ M;$
$RETURN\ (M, vm')$
$\})$›

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* ≤ *uint32-max*);
ASSERT(*case M of* (*M*, -) ⇒ *M* ≠ []);
ASSERT(*tl-trailt-tr-no-CS-pre M*);
let L = *atm-of* (*lit-of-last-trail-pol M*);
ASSERT(*vmtf-unset-pre L vm*);
RETURN (*j* + *one-uint32-nat*, *tl-trailt-tr-no-CS M*, *isa-vmtf-unset L vm*)
})
(*zero-uint32-nat*, M_0, *vm*);
M ← *trail-conv-back-imp lev M*;
RETURN (*M*, *vm'*)
})›

lemma *isa-vmtf-unset-vmtf-unset*:
‹(*uncurry* (*RETURN oo isa-vmtf-unset*), *uncurry* (*RETURN oo vmtf-unset*)) ∈
nat-rel $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\rightarrow_f$
⟨(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)⟩*nres-rel*›
unfolding *vmtf-unset-def isa-vmtf-unset-def uncurry-def*
by (*intro frefI nres-relI*) *auto*

lemma *isa-vmtf-unset-isa-vmtf*:
assumes ‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and** ‹*L* ∈# $\mathcal{A}$›
shows ‹*isa-vmtf-unset L vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*›
proof −
obtain *vm0 to-remove to-remove'* **where**
vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
vm0: ‹(*vm0*, *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹(*to-remove*, *to-remove'*) ∈ *distinct-atoms-rel* $\mathcal{A}$›
using *assms* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def*)

then show *?thesis*
using *assms*
isa-vmtf-unset-vmtf-unset[*of* $\mathcal{A}$, *THEN fref-to-Down-unRET-uncurry*, *of L vm L* ‹(*vm0*, *to-remove'*)›]
abs-vmtf-ns-unset-vmtf-unset[*of* ‹*fst vm0*› ‹*fst* (*snd vm0*)› ‹*fst* (*snd* (*snd vm0*))›
‹*fst* (*snd* (*snd* (*snd vm0*)))› ‹*snd* (*snd* (*snd* (*snd vm0*)))› *to-remove'* $\mathcal{A}$ *M L to-remove'*]
by (*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *intro*: *isa-vmtfI elim!*: *prod-relE*)
qed

lemma *isa-vmtf-tl-isa-vmtf*:
assumes ‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and** ‹*M* ≠ []› **and** ‹*lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*L* = (*atm-of* (*lit-of* (*hd M*)))›
shows ‹*isa-vmtf-unset L vm* ∈ *isa-vmtf* $\mathcal{A}$ (*tl M*)›
proof −
let *?L* = ‹*atm-of* (*lit-of* (*hd M*))›
obtain *vm0 to-remove to-remove'* **where**
vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
vm0: ‹(*vm0*, *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹(*to-remove*, *to-remove'*) ∈ *distinct-atoms-rel* $\mathcal{A}$›
using *assms* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def*)

then show *?thesis*
using *assms*
isa-vmtf-unset-vmtf-unset[*of* $\mathcal{A}$, *THEN fref-to-Down-unRET-uncurry*, *of ?L vm ?L* ‹(*vm0*, *to-remove'*)›]
vmtf-unset-vmtf-tl[*of* ‹*fst vm0*› ‹*fst* (*snd vm0*)› ‹*fst* (*snd* (*snd vm0*))›
‹*fst* (*snd* (*snd* (*snd vm0*)))› ‹*snd* (*snd* (*snd* (*snd vm0*)))› *to-remove'* $\mathcal{A}$ *M*]
by (*cases M*)

(*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *intro*: *isa-vmtfI elim*!: *prod-relE*)
qed

lemma *isa-find-decomp-wl-imp-find-decomp-wl-imp*:
‹(*uncurry2 isa-find-decomp-wl-imp*, *uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$)) ∈
[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* $\mathcal{A}$)
→
⟨*trail-pol* $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)⟩*nres-rel*›
proof −
have [*intro*]: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ (*M'*, *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$› **for** *M' M*
by (*auto simp*: *trail-pol-def trail-pol-no-CS-def control-stack-length-count-dec*[*symmetric*])

have [*refine0*]: ‹((*zero-uint32-nat*, *trail-pol-conv-to-no-CS x1c*, *x2c*),
zero-uint32-nat, *trail-conv-to-no-CS x1a*, *x2a*)
∈ *nat-rel* $\times_r$ *trail-pol-no-CS* $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)›
if
‹*case y of*
(*x*, *xa*) ⇒ (*case x of* (*M*, *lev*) ⇒ λ-. *lev* < *count-decided M*) *xa*› **and**
‹(*x*, *y*)
∈ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1b* = (*x1c*, *x2b*)› **and**
‹*x* = (*x1b*, *x2c*)› **and**
‹*isa-length-trail-pre* (*trail-pol-conv-to-no-CS x1c*)› **and**
‹(*pos*, *posa*) ∈ *nat-rel*› **and**
‹*length* (*trail-conv-to-no-CS x1a*) − *posa* ≤ *uint-max*› **and**
‹*isa-length-trail* (*trail-pol-conv-to-no-CS x1c*) − *pos* ≤ *uint-max*› **and**
‹*case* (*zero-uint32-nat*, *trail-conv-to-no-CS x1a*, *x2a*) *of*
(*j*, *M*, *vm'*) ⇒
j ≤ *length* (*trail-conv-to-no-CS x1a*) − *posa* ∧
M = *drop j* (*trail-conv-to-no-CS x1a*) ∧
length (*trail-conv-to-no-CS x1a*) − *posa*
≤ *length* (*trail-conv-to-no-CS x1a*) ∧
vm' ∈ *vmtf* $\mathcal{A}$ *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* '# *mset M*)›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa*
proof −
show *?thesis*
supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
using *that* **by** *auto*
qed
have *trail-pol-empty*: ‹((([], *x2g*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ ⟹ *M* = []› **for** *M x2g*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def*)

have *isa-vmtf*: ‹(*x2c*, *x2a*) ∈ *Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$ ⟹
(((*aa*, *ab*, *ac*, *ad*, *ba*), *baa*, *ca*), *x2e*) ∈ *Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$ ⟹
x2e ∈ *vmtf* $\mathcal{A}$ (*drop x1d x1a*) ⟹
((*aa*, *ab*, *ac*, *ad*, *ba*), *baa*, *ca*) ∈ *isa-vmtf* $\mathcal{A}$ (*drop x1d x1a*)›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f*
x1g x1h x2g x2h aa ab ac ad ba baa ca
by (*cases x2e*)
(*auto 6 6 simp*: *isa-vmtf-def Image-iff converse-iff prod-rel-iff*
intro!: *bexI*[*of - x2e*])
have *trail-pol-no-CS-last-hd*:
‹((*x1h*, *t*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ ⟹ *M* ≠ [] ⟹ (*last x1h*) = *lit-of* (*hd M*)›
for *x1h t M*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def last-map*)

have *trail-conv-back*: ‹*trail-conv-back-imp x2b x1g*
  $\leq$ *SPEC*
   ($\lambda c.$ (*c*, *trail-conv-back x2 x1e*)
     $\in$ *trail-pol* $\mathcal{A}$)›
 if
  ‹*case y of* (*x*, *xa*) $\Rightarrow$ (*case x of* (*M*, *lev*) $\Rightarrow$ $\lambda vm.$ *lev* $<$ *count-decided M*) *xa*› **and**
  ‹(*x*, *y*) $\in$ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)› **and**
  ‹*x1* = (*x1a*, *x2*)› **and**
  ‹*y* = (*x1*, *x2a*)› **and**
  ‹*x1b* = (*x1c*, *x2b*)› **and**
  ‹*x* = (*x1b*, *x2c*)› **and**
  ‹*isa-length-trail-pre* (*trail-pol-conv-to-no-CS x1c*)› **and**
  ‹(*pos*, *posa*) $\in$ *nat-rel*› **and**
  ‹*length* (*trail-conv-to-no-CS x1a*) $-$ *posa* $\leq$ *uint-max*› **and**
  ‹*isa-length-trail* (*trail-pol-conv-to-no-CS x1c*) $-$ *pos* $\leq$ *uint-max*› **and**
  ‹(*xa*, *x'*) $\in$ *nat-rel* $\times_f$ (*trail-pol-no-CS* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$))› **and**
  ‹*x2d* = (*x1e*, *x2e*)› **and**
  ‹*x'* = (*x1d*, *x2d*)› **and**
  ‹*x2f* = (*x1g*, *x2g*)› **and**
  ‹*xa* = (*x1f*, *x2f*)› **and**
  ‹*x2* = *count-decided x1e*›
 for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f*
  x1g x2g
 apply (*rule trail-conv-back*[*THEN fref-to-Down-curry*, *THEN order-trans*])
 using *that* **by** (*auto simp*: *conc-fun-RETURN*)

show *?thesis*
 supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
 unfolding *isa-find-decomp-wl-imp-def find-decomp-wl-imp-def uncurry-def*
 apply (*intro frefI nres-relI*)
 apply (*refine-vcg*
  id-trail-conv-to-no-CS[*THEN fref-to-Down*, *unfolded comp-def*]
  get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*])
 subgoal
  by (*rule isa-length-trail-pre*) *auto*
 subgoal
  by (*auto simp*: *get-pos-of-level-in-trail-pre-def*)
 subgoal
  by *auto*
 subgoal
  by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
 apply (*assumption*+)[*10*]
 subgoal
  by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
 subgoal
  by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
 subgoal
  by *auto*
 subgoal
  by (*auto dest*!: *trail-pol-empty*)
 subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa*
  by (*rule tl-trailt-tr-no-CS-pre*) *auto*
 subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f*
  x1g x1h x2g x2h

by (*cases x1g*, *cases x2h*)
(*auto intro*!: *vmtf-unset-pre*[*of - - - - - -* $\mathcal{A}$ ‹*drop x1d x1a*›] *isa-vmtf*
simp: *lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def*)
subgoal
by (*auto simp*: *lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def*
intro!: *tl-trail-tr-no-CS*[*THEN fref-to-Down-unRET*]
isa-vmtf-unset-vmtf-unset[*THEN fref-to-Down-unRET-uncurry*])
apply (*rule trail-conv-back*; *assumption*)
subgoal
by *auto*
done
qed

abbreviation *find-decomp-w-ns-prop* **where**
‹*find-decomp-w-ns-prop* $\mathcal{A}$ ≡
(λ(*M*::(*nat*, *nat*) *ann-lits*) *highest* -.
(λ(*M1*, *vm*). ∃ *K M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
get-level M K = *Suc highest* ∧ *vm* ∈ *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* = (λ*L* (*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* {
$WHILE_T$λ(*i*, *vm*, φ). *i* ≤ *length C* ∧ (∀ *c* ∈ *set C*. *atm-of c* < *length* φ ∧ *atm-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*) ∈# $\mathcal{A}_{in}$);
let vm′ = *vmtf-mark-to-rescore* (*atm-of* (*C*!*i*)) *vm*;
RETURN(*i*+*1*, *vm′*, φ)
})
(*0*, *vm*, φ)
}›

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 {
$WHILE_T{}^{\lambda(i,\ vm,\ \varphi).\ i \leq length\ C\ \wedge}$ (∀ c ∈ set C. atm-of c < length φ ∧ atm-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 $\mathcal{A}$)) ∈
[λ(((C, M), vm), φ). literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (mset C) ∧ vm ∈ vmtf $\mathcal{A}$ M ∧ phase-saving $\mathcal{A}$ φ]$_f$
(⟨Id⟩list-rel $\times_f$ Id $\times_f$ Id $\times_f$ Id) → ⟨Id $\times_f$ Id⟩ nres-rel›
proof −
have H: ‹vmtf-rescore-body $\mathcal{A}$ C M vm φ ≤
SPEC (λ(n :: nat, vm', φ' :: bool list). phase-saving $\mathcal{A}$ φ' ∧ vm' ∈ vmtf $\mathcal{A}$ M)›
if M: ‹vm ∈ vmtf $\mathcal{A}$ M›‹phase-saving $\mathcal{A}$ φ› **and** C: ‹∀ c∈set C. atm-of c ∈ atms-of ($\mathcal{L}_{all}$ $\mathcal{A}$)›
for C vm φ M
unfolding *vmtf-rescore-body-def vmtf-mark-to-rescore-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** R = ‹measure (λ(i, -). length C − i)› **and**
I' = ‹λ(i, vm', φ'). phase-saving $\mathcal{A}$ φ' ∧ vm' ∈ vmtf $\mathcal{A}$ M›])
subgoal by *auto*
subgoal by *auto*
subgoal using C M **by** (*auto simp*: *vmtf-def phase-saving-def*)
subgoal using C M **by** *auto*
subgoal using M **by** *auto*
subgoal by *auto*
subgoal using C **by** (*auto simp*: *atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$*)
subgoal using C **unfolding** *phase-saving-def* **by** *auto*
subgoal unfolding *phase-saving-def* **by** *auto*

subgoal using *C* **unfolding** *phase-saving-def* **by** *auto*
subgoal using *C* **by** (*auto simp*: *vmtf-append-remove-iff'*)
subgoal by *auto*
done
have *K*: ‹$((a,\ b),(a',\ b')) \in A \times_f B \longleftrightarrow (a,\ a') \in A \wedge (b,\ b') \in B$› **for** *a b a' b' A B*
by *auto*
show *?thesis*
unfolding *vmtf-rescore-def rescore-clause-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule bind-refine-spec*)
prefer *2*
apply (*subst* (*asm*) *K*)
apply (*rule H*; *auto*)
subgoal
by (*meson atm-of-lit-in-atms-of contra-subsetD in-all-lits-of-m-ain-atms-of-iff in-multiset-in-set literals-are-in-*$\mathcal{L}_{in}$*-def*)
subgoal by *auto*
done
qed

lemma *isa-vmtf-rescore-body*:
‹(*uncurry3* (*isa-vmtf-rescore-body*), *uncurry3* (*vmtf-rescore-body* $\mathcal{A}$)) $\in$ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
(*Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ *Id*) → ⟨*Id* $\times_r$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$) $\times_r$ *Id*⟩ *nres-rel*›
proof −
show *?thesis*
unfolding *isa-vmtf-rescore-body-def vmtf-rescore-body-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xa x' x1f x2f x1g x2g*
by (*cases x1g*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xa x' x1f x2f x1g x2g*
unfolding *isa-vmtf-mark-to-rescore-pre-def*
by (*cases x1g*)
(*auto intro*!: *atms-hash-insert-pre*)
subgoal
by (*auto intro*!: *isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
done
qed

lemma *isa-vmtf-rescore*:
‹(*uncurry3* (*isa-vmtf-rescore*), *uncurry3* (*vmtf-rescore* $\mathcal{A}$)) $\in$ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
(*Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ *Id*) → ⟨(*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ *Id*⟩ *nres-rel*›
proof −
show *?thesis*
unfolding *isa-vmtf-rescore-def vmtf-rescore-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg isa-vmtf-rescore-body*[*THEN fref-to-Down-curry3*])
subgoal by *auto*
subgoal by *auto*

done
qed

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 −
have *length-M0*: ‹$length\ M_0 \le uint32\text{-}max\ div\ 2 + 1$›
using *length-trail-uint-max-div2*[*of* $\mathcal{A}$ M_0, *OF bounded*]
n-d $literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\text{-}in\text{-}lits\text{-}of\text{-}l$[*of* $\mathcal{A}$, *OF lits*]
by (*auto simp*: *lits-of-def*)
have *1*: ‹((*count-decided x1g*, *x1g*), *count-decided x1*, *x1*) ∈ *Id*›
if ‹*x1g* = *x1*› **for** *x1g x1* :: ‹(*nat*, *nat*) *ann-lits*›
using *that* **by** *auto*
have [*simp*]: ‹∃ *M′a*. *M′* @ *x2g* = *M′a* @ *tl x2g*› **for** *M′ x2g* :: ‹(*nat*, *nat*) *ann-lits*›
by (*rule exI*[*of* - ‹*M′* @ (*if x2g* = [] *then* [] *else* [*hd x2g*])›]) *auto*
have *butlast-nil-iff*: ‹*butlast xs* = [] ⟷ *xs* = [] ∨ (∃ *a*. *xs* = [*a*])› **for** *xs* :: ‹(*nat*, *nat*) *ann-lits*›
by (*cases xs*) *auto*
have *butlast1*: ‹*tl x2g* = *drop* (*Suc* (*length x1*) − *length x2g*) *x1*› (**is** ‹*?G1*›)
if ‹*x2g* = *drop* (*length x1* − *length x2g*) *x1*› **for** *x2g x1* :: ‹′*a list*›
proof −
have [*simp*]: ‹*Suc* (*length x1* − *length x2g*) = *Suc* (*length x1*) − *length x2g*›
by (*metis Suc-diff-le diff-le-mono2 diff-zero length-drop that zero-le*)
show *?G1*
by (*subst that*) (*auto simp*: *butlast-conv-take tl-drop-def*)[]
qed
have *butlast2*: ‹*tl x2g* = *drop* (*length x1* − (*length x2g* − *Suc 0*)) *x1*› (**is** ‹*?G2*›)
if ‹*x2g* = *drop* (*length x1* − *length x2g*) *x1*› **and** *x2g*: ‹*x2g* ≠ []› **for** *x2g x1* :: ‹′*a list*›
proof −
have [*simp*]: ‹*Suc* (*length x1* − *length x2g*) = *Suc* (*length x1*) − *length x2g*›
by (*metis Suc-diff-le diff-le-mono2 diff-zero length-drop that*(*1*) *zero-le*)
have [*simp*]: ‹*Suc* (*length x1*) − *length x2g* = *length x1* − (*length x2g* − *Suc 0*)›
using *x2g* **by** *auto*
show *?G2*
by (*subst that*) (*auto simp*: *butlast-conv-take tl-drop-def*)[]
qed
note *butlast* = *butlast1 butlast2*

have *count-decided-not-Nil*[*simp*]: ‹*0* < *count-decided M* ⟹ *M* ≠ []› **for** *M* :: ‹(*nat*, *nat*) *ann-lits*›
by *auto*
have *get-lev-last*: ‹*get-level* (*M′* @ *M*) (*lit-of* (*last M′*)) = *Suc* (*count-decided M*)›
if ‹M_0 = *M′* @ *M*› **and** ‹*M′* ≠ []› **and** ‹*is-decided* (*last M′*)› **for** *M′ M*
apply (*cases M′ rule*: *rev-cases*)
using *that* **apply** (*solves simp*)
using *n-d that* **by** *auto*

have *atm-of-N*:
‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}$ (*lit-of* ‘# *mset aa*) ⟹ *aa* ≠ [] ⟹ *atm-of* (*lit-of* (*hd aa*)) ∈ *atms-of* ($\mathcal{L}_{all}\ \mathcal{A}$)›

for *aa*
by (*cases aa*) (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-add-mset in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
have *Lin-drop-tl*: ‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*lit-of* ‘# *mset* (*drop b* M_0)) $\Longrightarrow$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset* (*tl* (*drop b* M_0)))› **for** *b*
apply (*rule literals-are-in-$\mathcal{L}_{in}$-mono*)
apply *assumption*
by (*cases* ‹*drop b* M_0›) *auto*

have *highest*: ‹*highest* = *count-decided M*› **and**
ex-decomp: ‹$\exists$ *K M2*.
(*Decided K* # *M*, *M2*)
$\in$ *set* (*get-all-ann-decomposition* M_0) $\wedge$
get-level M_0 *K* = *Suc highest* $\wedge$ *vm* $\in$ *vmtf* $\mathcal{A}$ *M*›
if
pos: ‹*pos* < *length* M_0 $\wedge$ *is-decided* (*rev* M_0 ! *pos*) $\wedge$ *get-level* M_0 (*lit-of* (*rev* M_0 ! *pos*)) =
highest + *1*› **and**
‹*length* M_0 − *pos* $\leq$ *uint-max*› **and**
inv: ‹*case s of* (*j*, *M*, *vm'*) $\Rightarrow$
j $\leq$ *length* M_0 − *pos* $\wedge$
M = *drop j* M_0 $\wedge$
length M_0 − *pos* $\leq$ *length* M_0 $\wedge$
vm' $\in$ *vmtf* $\mathcal{A}$ *M* $\wedge$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset M*)› **and**
cond: ‹$\neg$ (*case s of*
(*j*, *M*, *vm*) $\Rightarrow$ *j* < *length* M_0 − *pos*)› **and**
s: ‹*s* = (*j*, *s'*)› ‹*s'* = (*M*, *vm*)›
for *pos s j s' M vm*
proof −
have
‹*j* = *length* M_0 − *pos*› **and**
M: ‹*M* = *drop* (*length* M_0 − *pos*) M_0› **and**
vm: ‹*vm* $\in$ *vmtf* $\mathcal{A}$ (*drop* (*length* M_0 − *pos*) M_0)› **and**
‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*lit-of* ‘# *mset* (*drop* (*length* M_0 − *pos*) M_0))›
using *cond inv* **unfolding** *s*
by *auto*
define *M2* **and** *L* **where** ‹*M2* = *take* (*length* M_0 − *Suc pos*) M_0› **and** ‹*L* = *rev* M_0 ! *pos*›
have *le-Suc-pos*: ‹*length* M_0 − *pos* = *Suc* (*length* M_0 − *Suc pos*)›
using *pos* **by** *auto*
have *1*: ‹*take* (*length* M_0 − *pos*) M_0 = *take* (*length* M_0 − *Suc pos*) M_0 @ [*rev* M_0 ! *pos*]›
unfolding *le-Suc-pos*
apply (*subst take-Suc-conv-app-nth*)
using *pos* **by** (*auto simp*: *rev-nth*)
have M_0: ‹M_0 = *M2* @ *L* # *M*›
apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* M_0 − *pos*›])
unfolding *M L-def M2-def 1*
by *auto*
have *L'*: ‹*Decided* (*lit-of L*) = *L*›
using *pos* **unfolding** *L-def*[*symmetric*] **by** (*cases L*) *auto*
then have M_0*'*: ‹M_0 = *M2* @ *Decided* (*lit-of L*) # *M*›
unfolding M_0 **by** *auto*

have ‹*highest* = *count-decided M*› **and** ‹*get-level* M_0 (*lit-of L*) = *Suc highest*› **and** ‹*is-decided L*›
using *n-d pos* **unfolding** *L-def*[*symmetric*] **unfolding** M_0
by (*auto simp*: *get-level-append-if split*: *if-splits*)
then show
‹$\exists$ *K M2*.

(*Decided K* # *M*, *M2*)
∈ *set* (*get-all-ann-decomposition* M_0) ∧
get-level M_0 *K* = *Suc highest* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*›
using *get-all-ann-decomposition-ex*[*of* ‹*lit-of L*› *M M2*] *vm* **unfolding** $M_0{}'$[*symmetric*] *M*[*symmetric*]
by *blast*
show ‹*highest* = *count-decided M*›
using ‹*highest* = *count-decided M*› **.**
qed
show *?decomp*
unfolding *find-decomp-wl-imp-def Let-def find-decomp-w-ns-def trail-conv-to-no-CS-def*
get-pos-of-level-in-trail-def trail-conv-back-def
apply (*refine-vcg 1 WHILEIT-rule*[**where** *R*=‹*measure* (λ(-, *M*, -). *length M*)›])
subgoal using *length-M0* **unfolding** *uint32-max-def* **by** *simp*
subgoal by *auto*
subgoal using *target* **by** (*auto simp*: *count-decided-ge-get-maximum-level*)
subgoal by *auto*
subgoal by *auto*
subgoal using *vm* **by** *auto*
subgoal using *lits* **unfolding** *literals-are-in-*$\mathcal{L}_{in}$*-trail-lit-of-mset* **by** *auto*
subgoal for *target s j b M vm* **by** *simp*
subgoal using *length-M0* **unfolding** *uint32-max-def* **by** *simp*
subgoal for *x s a ab aa bb*
by (*cases* ‹*drop a* M_0›)
(*auto simp*: *lit-of-hd-trail-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
subgoal by *auto*
subgoal by (*auto simp*: *drop-Suc drop-tl*)
subgoal by *auto*
subgoal for *s a b aa ba vm x2 x1a x2a*
by (*cases vm*)
(*auto intro*!: *vmtf-unset-vmtf-tl atm-of-N drop-tl simp*: *lit-of-hd-trail-def*)
subgoal for *s a b aa ba x1 x2 x1a x2a*
using *lits* **by** (*auto intro*: *Lin-drop-tl*)
subgoal by *auto*
subgoal by (*rule highest*)
subgoal by (*rule ex-decomp*) (*assumption*+, *auto*)
done
qed

lemma *find-decomp-wl-imp-find-decomp-wl'*:
‹(*uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$), *uncurry2* (*find-decomp-w-ns* $\mathcal{A}$)) ∈
[*find-decomp-w-ns-pre* $\mathcal{A}$]$_f$ *Id* $\times_f$ *Id* $\times_f$ *Id* → ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *find-decomp-w-ns-pre-def simp del*: *twl-st-of-wl.simps*
intro!: *find-decomp-wl-imp-le-find-decomp-wl'*)

lemma *find-decomp-wl-imp-code-conbine-cond*:
‹(λ((*b*, *a*), *c*). *find-decomp-w-ns-pre* $\mathcal{A}$ ((*b*, *a*), *c*) ∧ *a* < *count-decided b*) = (λ((*b*, *a*), *c*).
find-decomp-w-ns-pre $\mathcal{A}$ ((*b*, *a*), *c*))›
by (*auto intro*!: *ext simp*: *find-decomp-w-ns-pre-def*)

definition *vmtf-mark-to-rescore-clause* **where**
‹*vmtf-mark-to-rescore-clause* $\mathcal{A}_{in}$ *arena C vm* = *do* {
  ASSERT(*arena-is-valid-clause-idx arena C*);
  nfoldli
   ([*C*..<*C* + *nat-of-uint64-conv* (*arena-length arena C*)])
   (λ-. *True*)
   (λ*i vm*. *do* {
    ASSERT(*i* < *length arena*);
    ASSERT(*arena-lit-pre arena i*);
    ASSERT(*atm-of* (*arena-lit arena i*) ∈# $\mathcal{A}_{in}$);
    RETURN (*vmtf-mark-to-rescore* (*atm-of* (*arena-lit arena i*)) *vm*)
   })
   vm
 }›

definition *isa-vmtf-mark-to-rescore-clause* **where**
‹*isa-vmtf-mark-to-rescore-clause arena C vm* = *do* {
  ASSERT(*arena-is-valid-clause-idx arena C*);
  nfoldli
   ([*C*..<*C* + *nat-of-uint64-conv* (*arena-length arena C*)])
   (λ-. *True*)
   (λ*i vm*. *do* {
    ASSERT(*i* < *length arena*);
    ASSERT(*arena-lit-pre arena i*);
    ASSERT(*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*arena-lit arena i*)) *vm*);
    RETURN (*isa-vmtf-mark-to-rescore* (*atm-of* (*arena-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* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
  Id $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) → ⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
 unfolding *isa-vmtf-mark-to-rescore-clause-def vmtf-mark-to-rescore-clause-def*
  uncurry-def
 apply (*intro frefI nres-relI*)
 apply (*refine-rcg nfoldli-refine*[**where** *R* = ‹*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$› **and** *S* = *Id*])
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c xi xa si s*
  by (*cases s*)
   (*auto simp*: *isa-vmtf-mark-to-rescore-pre-def*
    intro!: *atms-hash-insert-pre*)
 subgoal
  by (*rule isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
   auto
 done

lemma *vmtf-mark-to-rescore-clause-spec*:
 ‹*vm* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹

$(\forall\, C \in set\ [C..<C + arena\text{-}length\ arena\ C].\ arena\text{-}lit\ arena\ C \in\#\ \mathcal{L}_{all}\ \mathcal{A}) \Longrightarrow$
vmtf-mark-to-rescore-clause $\mathcal{A}$ *arena C vm* $\leq$ *RES* (*vmtf* $\mathcal{A}$ *M*)›
unfolding *vmtf-mark-to-rescore-clause-def*
apply (*subst RES-SPEC-conv*)
apply (*refine-vcg nfoldli-rule*[**where** *I* = ‹λ- - *vm*. *vm* ∈ *vmtf* $\mathcal{A}$ *M*›])
subgoal
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-def*
apply (*rule exI*[*of - N*])
apply (*rule exI*[*of - vdom*])
apply (*fastforce simp*: *arena-lifting*)
done
subgoal for *x it σ*
using *arena-lifting*(*7*)[*of arena N vdom C* ‹*x* − *C*›]
by (*auto simp*: *arena-lifting*(*1*−*6*) *dest*!: *in-list-in-setD*)
subgoal for *x it σ*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
apply (*rule exI*[*of - C*])
apply (*intro conjI*)
apply (*solves* ‹*auto dest*: *in-list-in-setD*›)
apply (*rule exI*[*of - N*])
apply (*rule exI*[*of - vdom*])
apply (*fastforce simp*: *arena-lifting dest*: *in-list-in-setD*)
done
subgoal for *x it σ*
by *fastforce*
subgoal for *x it - σ*
by (*cases σ*)
(*auto intro*!: *vmtf-mark-to-rescore simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
dest: *in-list-in-setD*)
done

definition *vmtf-mark-to-rescore-also-reasons*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *arena* ⇒ *nat literal list* ⇒ - ⇒-› **where**
‹*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$ *M arena outl vm* = *do* {
ASSERT(*length outl* ≤ *uint32-max*);
nfoldli
([*0*..<*length outl*])
(λ-. *True*)
(λ*i vm*. *do* {
ASSERT(*i* < *length outl*); *ASSERT*(*length outl* ≤ *uint32-max*);
ASSERT(−*outl* ! *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
C ← *get-the-propagation-reason M* (−(*outl* ! *i*));
case C of
None ⇒ *RETURN* (*vmtf-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*)
| *Some C* ⇒ *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* ⇒ *arena* ⇒ *nat literal list* ⇒ - ⇒-› **where**
‹*isa-vmtf-mark-to-rescore-also-reasons M arena outl vm* = *do* {
ASSERT(*length outl* ≤ *uint32-max*);
nfoldli
([*0*..<*length outl*])
(λ-. *True*)

$(\lambda i\ vm.\ do\ \{$
$ASSERT(i < length\ outl);\ ASSERT(length\ outl \leq uint32\text{-}max);$
$C \leftarrow get\text{-}the\text{-}propagation\text{-}reason\text{-}pol\ M\ (-(outl\ !\ i));$
$case\ C\ of$
$None \Rightarrow do\ \{$
$ASSERT\ (isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}pre\ (atm\text{-}of\ (outl\ !\ i))\ vm);$
$RETURN\ (isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\ (atm\text{-}of\ (outl\ !\ i))\ vm)$
$\}$
$|\ Some\ C \Rightarrow if\ C = 0\ then\ RETURN\ vm\ else\ isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}clause\ arena\ C\ vm$
$\})$
vm
$\}\rangle$

lemma *isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons*:
$\langle(uncurry3\ isa\text{-}vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}also\text{-}reasons,\ uncurry3\ (vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}also\text{-}reasons\ \mathcal{A})) \in$
$[\lambda\text{-}.\ isasat\text{-}input\text{-}bounded\ \mathcal{A}]_f$
$trail\text{-}pol\ \mathcal{A} \times_f Id \times_f Id \times_f (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}) \to \langle Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}\rangle nres\text{-}rel\rangle$
unfolding *isa-vmtf-mark-to-rescore-also-reasons-def vmtf-mark-to-rescore-also-reasons-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg nfoldli-refine*[**where** $R = \langle Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}\rangle$ **and** $S = Id$]
get-the-propagation-reason-pol[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*]
isa-vmtf-mark-to-rescore-clause-vmtf-mark-to-rescore-clause[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xi xa si s xb x'*
by (*cases s*)
(*auto simp*: *isa-vmtf-mark-to-rescore-pre-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
intro!: *atms-hash-insert-pre*[*of* - $\mathcal{A}$])
subgoal
by (*rule isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
(*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
subgoal by *auto*
subgoal by *auto*
done

lemma *vmtf-mark-to-rescore'*:
$\langle L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}) \Longrightarrow vm \in vmtf\ \mathcal{A}\ M \Longrightarrow vmtf\text{-}mark\text{-}to\text{-}rescore\ L\ vm \in vmtf\ \mathcal{A}\ M\rangle$
by (*cases vm*) (*auto intro*: *vmtf-mark-to-rescore*)

lemma *vmtf-mark-to-rescore-also-reasons-spec*:
$\langle vm \in vmtf\ \mathcal{A}\ M \Longrightarrow valid\text{-}arena\ arena\ N\ vdom \Longrightarrow length\ outl \leq uint32\text{-}max \Longrightarrow$
$(\forall L \in set\ outl.\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}) \Longrightarrow$
$(\forall L \in set\ outl.\ \forall C.\ (Propagated\ (-L)\ C \in set\ M \longrightarrow C \neq 0 \longrightarrow (C \in\#\ dom\text{-}m\ N\ \wedge$
$(\forall C \in set\ [C..<C + arena\text{-}length\ arena\ C].\ arena\text{-}lit\ arena\ C \in\#\ \mathcal{L}_{all}\ \mathcal{A})))) \Longrightarrow$
$vmtf\text{-}mark\text{-}to\text{-}rescore\text{-}also\text{-}reasons\ \mathcal{A}\ M\ arena\ outl\ vm \leq RES\ (vmtf\ \mathcal{A}\ M)\rangle$
unfolding *vmtf-mark-to-rescore-also-reasons-def*
apply (*subst RES-SPEC-conv*)
apply (*refine-vcg nfoldli-rule*[**where** $I = \langle\lambda\text{-}\ \text{-}\ vm.\ vm \in vmtf\ \mathcal{A}\ M\rangle$])
subgoal by (*auto dest*: *in-list-in-setD*)

subgoal for *x l1 l2 σ*
unfolding *all-set-conv-nth*
by (*auto simp*: *uminus-*$\mathcal{A}_{in}$*-iff dest*!: *in-list-in-setD*)
subgoal for *x l1 l2 σ*
unfolding *get-the-propagation-reason-def*
apply (*rule SPEC-rule*)
apply (*rename-tac reason*, *case-tac reason*; *simp only*: *option.simps RES-SPEC-conv*[*symmetric*])
subgoal
by (*auto simp*: *vmtf-mark-to-rescore′*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff* [*symmetric*])
apply (*rename-tac D*, *case-tac* ‹*D* = *0*›; *simp*)
subgoal
by (*rule vmtf-mark-to-rescore-clause-spec*, *assumption*, *assumption*)
fastforce+
done
done

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* ›
unfolding *isa-vmtf-find-next-undef-def vmtf-find-next-undef-def uncurry-def*
defined-atm-def [*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by (*rule defined-atm-pol-pre*) (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
subgoal
by (*auto simp*: *undefined-atm-code*[*THEN fref-to-Down-unRET-uncurry-Id*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

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*›
by *sepref-to-hoare* (*sep-auto simp*: *size-conflict-int-def*)

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›
  by *sepref-to-hoare*
  (*sep-auto simp*: *vmtf-node-rel-def uint32-nat-rel-def br-def option-assn-alt-def*
    split: *option.splits*)

lemma *stamp-ref*[*sepref-fr-rules*]:
  ‹(*return o stamp*, *RETURN o stamp*) ∈ *vmtf-node-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
  by *sepref-to-hoare*
    (*auto simp*: *ex-assn-move-out*(*2*)[*symmetric*] *return-cons-rule ent-ex-up-swap vmtf-node-rel-def*
      simp del: *ex-assn-move-out*)

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›
  unfolding *option-assn-pure-conv*
  by *sepref-to-hoare* (*sep-auto simp*: *return-cons-rule vmtf-node-rel-def*)

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›
  unfolding *option-assn-pure-conv*
  by *sepref-to-hoare* (*sep-auto simp*: *return-cons-rule vmtf-node-rel-def*)

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*›
  unfolding *atoms-hash-del-def atoms-hash-del-pre-def*
  by *sepref*

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*›
  unfolding *atoms-hash-insert-def atms-hash-insert-pre-def*
  by *sepref*

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*›
  unfolding *get-pos-of-level-in-trail-imp-def*

by *sepref*

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*›
:: ‹$trail\text{-}pol\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
unfolding *get-pos-of-level-in-trail-imp-def*
by *sepref*

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*)) ∈
$(option\text{-}assn\ uint32\text{-}nat\text{-}assn)^k *_a vmtf\text{-}remove\text{-}conc^d \rightarrow_a vmtf\text{-}remove\text{-}conc$›
unfolding *option-assn-pure-conv*
by *sepref-to-hoare* (*sep-auto simp*: *update-next-search-def*)

sepref-definition (**in** −)*ns-vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo ns-vmtf-dequeue*)›
:: ‹$[vmtf\text{-}dequeue\text{-}pre]_a$
$uint32\text{-}nat\text{-}assn^k *_a (array\text{-}assn\ vmtf\text{-}node\text{-}assn)^d \rightarrow array\text{-}assn\ vmtf\text{-}node\text{-}assn$›
supply [[*goals-limit* = *1*]]
supply *option.splits*[*split*]
unfolding *ns-vmtf-dequeue-def vmtf-dequeue-pre-alt-def*
by *sepref*

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*)›
:: ‹$[\lambda(L,(ns,m,fst\text{-}As,next\text{-}search)).\ L < length\ ns \wedge vmtf\text{-}dequeue\text{-}pre\ (L,\ ns)]_a$
$uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}conc^d \rightarrow vmtf\text{-}conc\text{-}option\text{-}fst\text{-}As$›
supply [[*goals-limit* = *1*]]
unfolding *vmtf-dequeue-def*
by *sepref*

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*›
supply [[*goals-limit* = *1*]]
unfolding *isa-vmtf-enqueue-def vmtf-enqueue-pre-def defined-atm-def*[*symmetric*]
one-uint64-nat-def[*symmetric*]
by *sepref*

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*›
supply [[*goals-limit* = *1*]]
unfolding *isa-vmtf-enqueue-def vmtf-enqueue-pre-def defined-atm-def*[*symmetric*]
one-uint64-nat-def[*symmetric*]
by *sepref*

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*›
unfolding *partition-vmtf-nth-def insert-sort-inner-def fast-minus-def*[*symmetric*]
partition-main-def choose-pivot3-def one-uint32-nat-def[*symmetric*]
WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
supply [[*goals-limit* = *1*]]
supply *partition-vmtf-nth-code-helper3*[*intro*] *partition-main-inv-def*[*simp*]
by *sepref*

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* (−))) ∈
[λ(*a*, *b*). *a* ≥ *b*]$_a$ *uint32-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} → *uint32-nat-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *uint32-nat-rel-def nat-of-uint32-le-minus*
br-def uint32-safe-minus-def nat-of-uint32-notle-minus
nat-of-uint32-ge-minus nat-of-uint32-le-iff)

sepref-definition (**in** −) *partition-between-ref-vmtf-code*
is ‹*uncurry3 partition-between-ref-vmtf*›
:: ‹[λ(((*vm*), -), *remove*). (∀ *x*∈#*mset remove*. *x* < *length* (*fst vm*)) ∧ *length remove* ≤ *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*›
supply [[*goals-limit=1*]] *uint32-nat-assn-minus-fast*[*sepref-fr-rules*]

unfolding *quicksort-vmtf-nth-def insert-sort-def partition-vmtf-nth-def*[*symmetric*]
quicksort-vmtf-nth-ref-def List.null-def quicksort-ref-def
length-0-conv[*symmetric*] *length-uint32-nat-def*[*symmetric*]
zero-uint32-nat-def[*symmetric*] *partition-between-ref-vmtf-def*
partition-between-ref-def two-uint32-nat-def[*symmetric*]
partition-vmtf-nth-def[*symmetric*] *choose-pivot3-def*
WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
by *sepref*

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-assn uint32-nat-assn›
unfolding *quicksort-vmtf-nth-def insert-sort-def partition-vmtf-nth-def*[*symmetric*]
quicksort-vmtf-nth-ref-def List.null-def quicksort-ref-def
length-0-conv[*symmetric*] *length-uint32-nat-def*[*symmetric*]
zero-uint32-nat-def[*symmetric*] *one-uint32-nat-def*[*symmetric*]
partition-vmtf-nth-def[*symmetric*]
partition-between-ref-vmtf-def[*symmetric*]
partition-vmtf-nth-def[*symmetric*]
supply [[*goals-limit* = *1*]]
supply *mset-eq-setD*[*dest*] *mset-eq-length*[*dest*]
arl-length-hnr[*sepref-fr-rules*] *uint32-nat-assn-minus*[*sepref-fr-rules*]
by *sepref*

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-assn uint32-nat-assn›
unfolding *quicksort-vmtf-nth-def insert-sort-def partition-vmtf-nth-def*[*symmetric*]
full-quicksort-ref-def List.null-def one-uint32-nat-def[*symmetric*]
length-0-conv[*symmetric*] *zero-uint32-nat-def*[*symmetric*]
quicksort-vmtf-nth-ref-def[*symmetric*]
supply [[*goals-limit* = *1*]]
supply *mset-eq-setD*[*dest*] *mset-eq-length*[*dest*]
arl-length-hnr[*sepref-fr-rules*] *uint32-nat-assn-minus*[*sepref-fr-rules*]
by *sepref*

declare *quicksort-vmtf-nth-code.refine*[*sepref-fr-rules*]

lemma *quicksort-vmtf-nth-code-reorder-list*[*sepref-fr-rules*]:
‹(*uncurry quicksort-vmtf-nth-code*, *uncurry reorder-list*) ∈
$[\lambda((a, \text{-}), b).\ (\forall x\in set\ b.\ x < length\ a) \wedge length\ b \leq uint32\text{-}max]_a$
$vmtf\text{-}conc^k *_a (arl32\text{-}assn\ uint32\text{-}nat\text{-}assn)^d \rightarrow arl32\text{-}assn\ uint32\text{-}nat\text{-}assn$›
supply [[*show-types*]]
using *quicksort-vmtf-nth-code.refine*[*FCOMP quicksort-vmtf-nth-reorder*[*unfolded convert-fref*]]
by *auto*

sepref-register *isa-vmtf-enqueue*

lemma *current-stamp-hnr*[*sepref-fr-rules*]:
‹(*return o current-stamp*, *RETURN o current-stamp*) ∈ *vmtf-conc*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *vmtf-node-rel-def current-stamp-alt-def*)

sepref-definition *vmtf-en-dequeue-code*
is ‹*uncurry2 isa-vmtf-en-dequeue*›
:: ‹[*isa-vmtf-en-dequeue-pre*]$_a$
trail-pol-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-conc*^{d} → *vmtf-conc*›
supply [[*goals-limit* = *1*]]
supply *isa-vmtf-en-dequeue-preD*[*dest*] *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*[*dest*]
unfolding *isa-vmtf-en-dequeue-def*
by *sepref*

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*]$_a$
trail-pol-fast-assn^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-conc*^{d} → *vmtf-conc*›
supply [[*goals-limit* = *1*]]
supply *isa-vmtf-en-dequeue-preD*[*dest*] *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*[*dest*]
unfolding *isa-vmtf-en-dequeue-def*
by *sepref*

declare *vmtf-en-dequeue-fast-code.refine*[*sepref-fr-rules*]

sepref-register *vmtf-rescale*
sepref-definition *vmtf-rescale-code*
is ‹*vmtf-rescale*›
:: ‹*vmtf-conc*^{d} $\rightarrow_a$ *vmtf-conc*›
supply [[*goals-limit* = *1*]]
supply *vmtf-en-dequeue-pre-def*[*simp*] *le-uint32-max-le-uint64-max*[*intro*]
unfolding *vmtf-rescale-alt-def zero-uint64-nat-def*[*symmetric*] *PR-CONST-def update-stamp.simps*
one-uint64-nat-def[*symmetric*]
by *sepref*

declare *vmtf-rescale-code.refine*[*sepref-fr-rules*]
lemma *uint64-nal-rel-le-uint64-max*: ‹(*a*, *b*) ∈ *uint64-nat-rel* ⟹ *b* ≤ *uint64-max*›
by (*auto simp*: *uint64-nat-rel-def br-def nat-of-uint64-le-uint64-max*)

This functions deletes all elements of a resizable array, without resizing it.

definition *emptied-arl* :: ‹*'a array-list32* ⇒ *'a array-list32*› **where**
‹*emptied-arl* = (λ(*a*, *n*). (*a*, *0*))›

lemma *emptied-arl-refine*[*sepref-fr-rules*]:
‹(*return o emptied-arl*, *RETURN o emptied-list*) ∈ (*arl32-assn R*)^d $\rightarrow_a$ *arl32-assn R*›
unfolding *emptied-arl-def emptied-list-def*
by *sepref-to-hoare* (*sep-auto simp*: *arl32-assn-def hr-comp-def is-array-list32-def*)

sepref-register *isa-vmtf-en-dequeue*
sepref-definition *isa-vmtf-flush-code*
is ‹*uncurry isa-vmtf-flush-int*›
:: ‹*trail-pol-assn*^{k} $*_a$ (*vmtf-conc* ∗*a* (*arl32-assn uint32-nat-assn* ∗*a atoms-hash-assn*))^d $\rightarrow_a$
(*vmtf-conc* ∗*a* (*arl32-assn uint32-nat-assn* ∗*a atoms-hash-assn*))›

supply [[*goals-limit* = *1*]] *minus-uint64-nat-assn*[*sepref-fr-rules*] *uint64-max-uint64-nat-assn*[*sepref-fr-rules*] *uint64-nal-rel-le-uint64-max*[*intro*]
unfolding *vmtf-flush-def PR-CONST-def isa-vmtf-flush-int-def zero-uint32-nat-def*[*symmetric*] *current-stamp-def*[*symmetric*] *one-uint32-nat-def*[*symmetric*] *uint64-max-uint64-def*[*symmetric*]
apply (*rewrite at* ‹*If* (⌑ ≥ -)› *uint64-of-uint32-conv-def*[*symmetric*])
apply (*rewrite at* ‹*length* - + ⌑› *nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

declare *isa-vmtf-flush-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-vmtf-flush-fast-code*
is ‹*uncurry isa-vmtf-flush-int*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ (*vmtf-conc* ∗*a* (*arl32-assn uint32-nat-assn* ∗*a atoms-hash-assn*))^d $\rightarrow_a$ (*vmtf-conc* ∗*a* (*arl32-assn uint32-nat-assn* ∗*a atoms-hash-assn*))›
supply [[*goals-limit* = *1*]] *minus-uint64-nat-assn*[*sepref-fr-rules*] *uint64-max-uint64-nat-assn*[*sepref-fr-rules*] *uint64-nal-rel-le-uint64-max*[*intro*]
unfolding *vmtf-flush-def PR-CONST-def isa-vmtf-flush-int-def zero-uint32-nat-def*[*symmetric*] *current-stamp-def*[*symmetric*] *one-uint32-nat-def*[*symmetric*] *uint64-max-uint64-def*[*symmetric*]
apply (*rewrite at* ‹*If* (⌑ ≥ -)› *uint64-of-uint32-conv-def*[*symmetric*])
apply (*rewrite at* ‹*length* - + ⌑› *nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

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*]$_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-remove-conc*^{d} → *vmtf-remove-conc*›
supply [[*goals-limit=1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*] *neq-NilE*[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
unfolding *isa-vmtf-mark-to-rescore-pre-def isa-vmtf-mark-to-rescore-def*
by *sepref*

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-unset-pre*]$_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-remove-conc*^{d} → *vmtf-remove-conc*›
supply [[*goals-limit=1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*] *neq-NilE*[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
unfolding *isa-vmtf-unset-def vmtf-unset-pre-def*
apply (*rewrite* **in** ‹*If* (- ∨ -)› *short-circuit-conv*)
by *sepref*

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-vmtf-mark-to-rescore-and-unset-pre*]$_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-remove-conc*^{d} → *vmtf-remove-conc*›
supply *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*iff*] *in-diffD*[*dest*] *option.splits*[*split*] *if-splits*[*split*] *isa-vmtf-unset-def*[*simp*]

supply [[*goals-limit=1*]]
unfolding *isa-vmtf-mark-to-rescore-and-unset-def isa-vmtf-mark-to-rescore-def save-phase-def isa-vmtf-mark-to-rescore-and-unset-pre-def*
by *sepref*

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*)›
:: ‹$[\lambda((M, lev), vm).\ True]_a$ $trail\text{-}pol\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d$
$\rightarrow$ *trail-pol-assn* $*a$ *vmtf-remove-conc*›
unfolding *isa-find-decomp-wl-imp-def get-maximum-level-remove-def*[*symmetric*] *PR-CONST-def trail-pol-conv-to-no-CS-def*
supply [[*goals-limit=1*]] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*] *trail-conv-to-no-CS-def*[*simp*] *lit-of-hd-trail-def*[*simp*]
supply *uint32-nat-assn-one*[*sepref-fr-rules*] *vmtf-unset-pre-def*[*simp*]
supply *uint32-nat-assn-minus*[*sepref-fr-rules*]
by *sepref*

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*)›
:: ‹$[\lambda((M, lev), vm).\ True]_a$ $trail\text{-}pol\text{-}fast\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d$
$\rightarrow$ *trail-pol-fast-assn* $*a$ *vmtf-remove-conc*›
unfolding *isa-find-decomp-wl-imp-def get-maximum-level-remove-def*[*symmetric*] *PR-CONST-def trail-pol-conv-to-no-CS-def*
supply *trail-conv-to-no-CS-def*[*simp*] *lit-of-hd-trail-def*[*simp*]
supply [[*goals-limit=1*]] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
supply *uint32-nat-assn-one*[*sepref-fr-rules*] *vmtf-unset-pre-def*[*simp*]
supply *uint32-nat-assn-minus*[*sepref-fr-rules*]
by *sepref*

declare *find-decomp-wl-imp-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *vmtf-rescore-code*
is ‹*uncurry3 isa-vmtf-rescore*›
:: ‹$(array\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a trail\text{-}pol\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d *_a phase\text{-}saver\text{-}conc^d \rightarrow_a$
vmtf-remove-conc $*a$ *phase-saver-conc*›
unfolding *isa-vmtf-rescore-body-def*[*abs-def*] *PR-CONST-def isa-vmtf-rescore-def*
supply [[*goals-limit* = *1*]] *fold-is-None*[*simp*]
by *sepref*

sepref-definition *vmtf-rescore-fast-code*
is ‹*uncurry3 isa-vmtf-rescore*›
:: ‹$(array\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a trail\text{-}pol\text{-}fast\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d *_a phase\text{-}saver\text{-}conc^d \rightarrow_a$
vmtf-remove-conc $*a$ *phase-saver-conc*›
unfolding *isa-vmtf-rescore-body-def*[*abs-def*] *PR-CONST-def isa-vmtf-rescore-def*
supply [[*goals-limit* = *1*]] *fold-is-None*[*simp*]
by *sepref*

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-nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
unfolding *find-decomp-wl-st-int-def PR-CONST-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$
isasat-bounded-assn›
unfolding *find-decomp-wl-st-int-def PR-CONST-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *vmtf-remove-conc*^{d} $\rightarrow_a$ *vmtf-remove-conc*›
supply [[*goals-limit=1*]]
unfolding *isa-vmtf-mark-to-rescore-clause-def PR-CONST-def*
by *sepref*

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-pol-assn*^{k} $*_a$ *arena-assn*^{k} $*_a$ (*arl32-assn unat-lit-assn*)^k $*_a$ *vmtf-remove-conc*^{d} $\rightarrow_a$ *vmtf-remove-conc*›
supply *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*iff*] *in-diffD*[*dest*] *option.splits*[*split*]
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$[*simp*]
supply [[*goals-limit=1*]]
unfolding *isa-vmtf-mark-to-rescore-also-reasons-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*rewrite at* ‹(⌑, -)› *zero-uint32-nat-def*[*symmetric*])
unfolding *one-uint32-nat-def*[*symmetric*] *nres-monad3*
by *sepref*

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-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint32-assn*›
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
unfolding *isa-arena-lit-def*
by *sepref*

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-fast-assn^k $*_a$ *nat-assn*^{k} → *unat-lit-assn*›
using *isa-arena-lit-fast-code2.refine*[*FCOMP isa-arena-lit-arena-lit*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*

by (*simp add*: *arl64-assn-comp*)

sepref-definition *vmtf-mark-to-rescore-clause-fast-code*
is ‹*uncurry2* (*isa-vmtf-mark-to-rescore-clause*)›
:: ‹$[\lambda((N, -), -).\ length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a vmtf\text{-}remove\text{-}conc^d \to vmtf\text{-}remove\text{-}conc$›
supply [[*goals-limit=1*]] *arena-is-valid-clause-idx-le-uint64-max*[*intro*]
unfolding *isa-vmtf-mark-to-rescore-clause-def PR-CONST-def nat-of-uint64-conv-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *one-uint64-nat-def*[*symmetric*] *nres-monad3 zero-uint64-nat-def*[*symmetric*]
by *sepref*

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*)›
:: ‹$[\lambda(((-, N), -), -).\ length\ N \leq uint64\text{-}max]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a (arl32\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a vmtf\text{-}remove\text{-}conc^d \to$
$vmtf\text{-}remove\text{-}conc$›
supply *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*iff*] *in-diffD*[*dest*] *option.splits*[*split*]
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$[*simp*]
supply [[*goals-limit=1*]]
unfolding *isa-vmtf-mark-to-rescore-also-reasons-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*rewrite at* ‹(⌑, -)› *zero-uint32-nat-def*[*symmetric*])
unfolding *one-uint32-nat-def*[*symmetric*] *nres-monad3 zero-uint64-nat-def*[*symmetric*]
by *sepref*

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(λ(*D*, *C*, *n*). *D* = *None* ∧ *mset C* = *mset outl* ∧ *C*!*0* = *outl*!*0* ∧
(*length C* > *1* ⟶ *highest-lit M* (*mset* (*tl C*)) (*Some* (*C*!*1*, *get-level M* (*C*!*1*)))) ∧
(*length C* > *1* ⟶ *n* = *get-level M* (*C*!*1*)) ∧
(*length C* = *1* ⟶ *n* = *0*)
)›

definition *empty-conflict-and-extract-clause-heur-inv* **where**
‹*empty-conflict-and-extract-clause-heur-inv M outl* =
(λ(*E*, *C*, *i*). *mset* (*take i C*) = *mset* (*take i outl*) ∧

length C = length outl ∧ *C ! 0 = outl ! 0* ∧ *i* ≥ *1* ∧ *i* ≤ *length outl* ∧
(*1* < *length* (*take i C*) ⟶
highest-lit M (*mset* (*tl* (*take i C*)))
(*Some* (*C ! 1*, *get-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*, -) ← $WHILE_T{}^{empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\text{-}heur\text{-}inv\ M\ outl}$
(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(*C*!*i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *C*!*1* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *1* < *length C*);
let C = (*if get-level M* (*C*!*i*) > *get-level 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* ⟶ *C*!*1* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
RETURN ((*True*, *D*), *C*, *if length outl = 1 then zero-uint32-nat else get-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* ≠ []› **and**
dist: ‹*distinct outl*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset outl*)› **and**
DD': ‹(*D'*, *D*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
consistent: ‹¬ *tautology* (*mset outl*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*empty-conflict-and-extract-clause-heur* $\mathcal{A}$ *M D' outl* ≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*empty-conflict-and-extract-clause M D outl*)›
proof −
have *size-out*: ‹*size* (*mset outl*) ≤ *1* + *uint-max div 2*›
using *simple-clss-size-upper-div2*[*OF bounded lits - consistent*]
‹*distinct outl*› **by** *auto*
have *empty-conflict-and-extract-clause-alt-def*:
‹*empty-conflict-and-extract-clause M D outl = do* {
(*D'*, *outl'*) ← *SPEC* (λ(*E*, *F*). *E* = {#} ∧ *mset F = D*);
SPEC
(λ(*D*, *C*, *n*).
D = None ∧
mset C = mset outl ∧
C ! 0 = outl ! 0 ∧

    (*1* < *length C* ⟶
      highest-lit M (*mset* (*tl C*)) (*Some* (*C* ! *1*, *get-level M* (*C* ! *1*)))) ∧
    (*1* < *length C* ⟶ *n* = *get-level M* (*C* ! *1*)) ∧ (*length C* = *1* ⟶ *n* = *0*))
  }› **for** *M D outl*
  unfolding *empty-conflict-and-extract-clause-def RES-RES2-RETURN-RES*
  by (*auto simp*: *ex-mset*)
define *I* **where**
  ‹*I* ≡ λ(*E*, *C*, *i*). *mset* (*take i C*) = *mset* (*take i outl*) ∧
    (*E*, *D* − *mset* (*take i outl*)) ∈ *lookup-clause-rel* $\mathcal{A}$ ∧
      length C = *length outl* ∧ *C* ! *0* = *outl* ! *0* ∧ *i* ≥ *1* ∧ *i* ≤ *length outl* ∧
      (*1* < *length* (*take i C*) ⟶
        highest-lit M (*mset* (*tl* (*take i C*)))
        (*Some* (*C* ! *1*, *get-level M* (*C* ! *1*))))›
have *I0*: ‹*I* (*D′*, *replicate* (*length outl*) (*outl* ! *0*), *one-uint32-nat*)›
  using *assms* **by** (*cases outl*) (*auto simp*: *I-def*)

have [*simp*]: ‹*ba* ≥ *1* ⟹ *mset* (*tl outl*) − *mset* (*take ba outl*) = *mset* ((*drop ba outl*))›
  for *ba*
  apply (*subst append-take-drop-id*[*of* ‹*ba* − *1*›, *symmetric*])
  using *dist*
  unfolding *mset-append*
  by (*cases outl*; *cases ba*)
    (*auto simp*: *take-tl drop-Suc*[*symmetric*] *remove-1-mset-id-iff-notin dest*: *in-set-dropD*)
have *empty-conflict-and-extract-clause-heur-inv*:
  ‹*empty-conflict-and-extract-clause-heur-inv M outl*
   (*D′*, *replicate* (*length outl*) (*outl* ! *0*), *one-uint32-nat*)›
  using *assms*
  unfolding *empty-conflict-and-extract-clause-heur-inv-def*
  by (*cases outl*) *auto*
have *I0*: ‹*I* (*D′*, *replicate* (*length outl*) (*outl* ! *0*), *one-uint32-nat*)›
  using *assms*
  unfolding *I-def*
  by (*cases outl*) *auto*
have
  C1-L: ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› (**is** *?A1inL*) **and**
  ba-le: ‹*ba* + *1* ≤ *uint-max*› (**is** *?ba-le*) **and**
  I-rec: ‹*I* (*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
      if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *one-uint32-nat*)
        < *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
      then swap (*aa*[*ba* := *outl* ! *ba*]) *one-uint32-nat ba*
      else aa[*ba* := *outl* ! *ba*],
      ba + *one-uint32-nat*)› (**is** *?I*) **and**
  inv: ‹*empty-conflict-and-extract-clause-heur-inv M outl*
     (*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
      if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *one-uint32-nat*)
        < *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
      then swap (*aa*[*ba* := *outl* ! *ba*]) *one-uint32-nat ba*
      else aa[*ba* := *outl* ! *ba*],
      ba + *one-uint32-nat*)› (**is** *?inv*)
  if
   ‹*empty-conflict-and-extract-clause-heur-inv M outl s*› **and**
   I: ‹*I s*› **and**
   ‹*case s of* (*D*, *C*, *i*) ⇒ *i* < *length-uint32-nat outl*› **and**
   st:
   ‹*s* = (*a*, *b*)›
   ‹*b* = (*aa*, *ba*)› **and**

ba-le: ‹*ba* < *length outl*› **and**
‹*ba* < *length aa*› **and**
‹*lookup-conflict-remove1-pre* (*outl* ! *ba*, *a*)›
for *s a b aa ba*
proof −
have
mset-aa: ‹*mset* (*take ba aa*) = *mset* (*take ba outl*)› **and**
aD: ‹(*a*, *D* − *mset* (*take ba outl*)) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
l-aa-outl: ‹*length aa* = *length outl*› **and**
aa0: ‹*aa* ! *0* = *outl* ! *0*› **and**
ba-ge1: ‹*1* ≤ *ba*› **and**
ba-lt: ‹*ba* ≤ *length outl*› **and**
highest: ‹*1* < *length* (*take ba aa*) ⟶
highest-lit M (*mset* (*tl* (*take ba aa*)))
(*Some* (*aa* ! *1*, *get-level M* (*aa* ! *1*)))›
using *I* **unfolding** *st I-def prod.case*
by *auto*
have *set-aa-outl*: ‹*set* (*take ba aa*) = *set* (*take ba outl*)›
using *mset-aa* **by** (*blast dest*: *mset-eq-setD*)
show *?ba-le*
using *ba-le assms size-out*
by (*auto simp*: *uint32-max-def*)
have *ba-ge1-aa-ge*: ‹*ba* > *1* ⟹ *aa* ! *1* ∈ *set* (*take ba aa*)›
using *ba-ge1 ba-le l-aa-outl*
by (*auto simp*: *in-set-take-conv-nth intro*!: *bex-lessI*[*of* - ‹*Suc 0*›])
then have ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈ *set outl*›
using *ba-le l-aa-outl ba-ge1*
unfolding *mset-aa in-multiset-in-set*[*symmetric*]
by (*cases* ‹*ba* > *1*›)
(*auto simp*: *mset-aa dest*: *in-set-takeD*)
then show *?A1inL*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$ *lits* **by** *auto*

define *aa2* **where** ‹*aa2* ≡ *tl* (*tl* (*take ba aa*))›
have *tl-take-nth-con*: ‹*tl* (*take ba aa*) = *aa* ! *Suc 0* # *aa2*› **if** ‹*ba* > *Suc 0*›
using *ba-le ba-ge1 that l-aa-outl* **unfolding** *aa2-def*
by (*cases aa*; *cases* ‹*tl aa*›; *cases ba*; *cases* ‹*ba* − *1*›)
auto
have *no-tauto-nth*: ‹ *i* < *length outl* ⟹ − *outl* ! *ba* = *outl* ! *i* ⟹ *False*› **for** *i*
using *consistent ba-le nth-mem*
by (*force simp*: *tautology-decomp′ uminus-lit-swap*)
have *outl-ba--L*: ‹*outl* ! *ba* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *lits ba-le literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$ **by** *auto*
have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
remove1-mset (*outl* ! *ba*) (*D* −(*mset* (*take ba outl*)))) ∈ *lookup-clause-rel* $\mathcal{A}$›
by (*rule lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*])
(*use ba-ge1 ba-le aD outl-ba--L* **in**
‹*auto simp*: *D in-set-drop-conv-nth image-image dest*: *no-tauto-nth*
intro!: *bex-geI*[*of* - *ba*]›)
then have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
D − *mset* (*take* (*Suc ba*) *outl*))
∈ *lookup-clause-rel* $\mathcal{A}$›
using *aD ba-le ba-ge1 ba-ge1-aa-ge aa0*
by (*auto simp*: *take-Suc-conv-app-nth*)
moreover have ‹*1* < *length*
(*take* (*ba* + *one-uint32-nat*)

```
          (if get-level M (aa[ba := outl ! ba] ! one-uint32-nat)
              < get-level M (aa[ba := outl ! ba] ! ba)
           then swap (aa[ba := outl ! ba]) one-uint32-nat ba
           else aa[ba := outl ! ba])) ⟶
    highest-lit M
    (mset
      (tl (take (ba + one-uint32-nat)
            (if get-level M (aa[ba := outl ! ba] ! one-uint32-nat)
                < get-level M (aa[ba := outl ! ba] ! ba)
             then swap (aa[ba := outl ! ba]) one-uint32-nat ba
             else aa[ba := outl ! ba]))))
    (Some
      ((if get-level M (aa[ba := outl ! ba] ! one-uint32-nat)
           < get-level M (aa[ba := outl ! ba] ! ba)
        then swap (aa[ba := outl ! ba]) one-uint32-nat ba
        else aa[ba := outl ! ba]) !
       1,
       get-level M
        ((if get-level M (aa[ba := outl ! ba] ! one-uint32-nat)
             < get-level M (aa[ba := outl ! ba] ! ba)
          then swap (aa[ba := outl ! ba]) one-uint32-nat ba
          else aa[ba := outl ! ba]) !
         1)))›
    using highest ba-le ba-ge1
    by (cases ‹ba = Suc 0›)
      (auto simp: highest-lit-def take-Suc-conv-app-nth l-aa-outl
        get-maximum-level-add-mset swap-nth-relevant max-def take-update-swap
        swap-only-first-relevant tl-update-swap mset-update nth-tl
        get-maximum-level-remove-non-max-lvl tl-take-nth-con
        aa2-def[symmetric])
  moreover have ‹mset
    (take (ba + one-uint32-nat)
      (if get-level M (aa[ba := outl ! ba] ! one-uint32-nat)
          < get-level M (aa[ba := outl ! ba] ! ba)
       then swap (aa[ba := outl ! ba]) one-uint32-nat ba
       else aa[ba := outl ! ba])) =
    mset (take (ba + one-uint32-nat) outl)›
    using ba-le ba-ge1 ba-ge1-aa-ge aa0
    unfolding mset-aa
    by (cases ‹ba = 1›)
      (auto simp: take-Suc-conv-app-nth l-aa-outl
        take-swap-relevant swap-only-first-relevant mset-aa set-aa-outl
        mset-update add-mset-remove-trivial-If)
  ultimately show ?I
    using ba-ge1 ba-le
    unfolding I-def prod.simps
    by (auto simp: l-aa-outl aa0)

  then show ?inv
    unfolding empty-conflict-and-extract-clause-heur-inv-def I-def
    by (auto simp: l-aa-outl aa0)
qed
have mset-tl-out: ‹mset (tl outl) − mset outl = {#}›
  by (cases outl) auto
have H1: ‹WHILE_T^empty-conflict-and-extract-clause-heur-inv M outl
  (λ(D, C, i). i < length-uint32-nat outl)
```

(λ(*D*, *C*, *i*). *do* {
  - $\leftarrow$ *ASSERT* (*i* < *length outl*);
  - $\leftarrow$ *ASSERT* (*i* < *length C*);
  - $\leftarrow$ *ASSERT* (*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
  - $\leftarrow$ *ASSERT*
    (*C*[*i* := *outl* ! *i*] ! *i* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$
     C[*i* := *outl* ! *i*] ! *1* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$
    1 < *length* (*C*[*i* := *outl* ! *i*]));
  - $\leftarrow$ *ASSERT* (*i* + *1* $\leq$ *uint-max*);
  RETURN
   (*lookup-conflict-remove1* (*outl* ! *i*) *D*,
    if get-level M (*C*[*i* := *outl* ! *i*] ! *one-uint32-nat*)
      < *get-level M* (*C*[*i* := *outl* ! *i*] ! *i*)
    then swap (*C*[*i* := *outl* ! *i*]) *one-uint32-nat i*
    else C[*i* := *outl* ! *i*],
    i + *one-uint32-nat*)
 })
 (D', *replicate* (*length outl*) (*outl* ! *0*), *one-uint32-nat*)
$\leq \Downarrow$ {((*E*, *C*, *n*), (E', F')). (*E*, E') $\in$ *lookup-clause-rel* $\mathcal{A}$ $\wedge$ *mset C* = *mset outl* $\wedge$
   C ! *0* = *outl* ! *0* $\wedge$
  (*1* < *length C* $\longrightarrow$
   highest-lit M (*mset* (*tl C*)) (*Some* (*C* ! *1*, *get-level M* (*C* ! *1*)))) $\wedge$
  n = *length outl* $\wedge$
  I (*E*, *C*, *n*)}
 (*SPEC* (λ(*E*, *F*). *E* = {#} $\wedge$ *mset F* = *D*))›
unfolding *conc-fun-RES*
apply (*refine-vcg WHILEIT-rule-stronger-inv-RES*[**where** *R* = ‹*measure* (λ(-, -, *i*). *length outl* − *i*)› **and**
  I' = ‹*I*›])
subgoal by *auto*
subgoal by (*rule empty-conflict-and-extract-clause-heur-inv*)
subgoal by (*rule I0*)
subgoal using *assms* **by** (*cases outl*; *auto*)
subgoal
 by (*auto simp*: *I-def*)
subgoal for *s a b aa ba*
 using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
 unfolding *lookup-conflict-remove1-pre-def prod.simps*
 by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
   lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
 using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
 unfolding *lookup-conflict-remove1-pre-def prod.simps*
 by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
   lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
 by (*rule C1-L*)
subgoal for *s a b aa ba*
 using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
 unfolding *lookup-conflict-remove1-pre-def prod.simps*
 by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
   lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
 by (*rule ba-le*)
subgoal
 by (*rule inv*)

```
  subgoal
    by (rule I-rec)
  subgoal
    by auto
  subgoal for s
    unfolding prod.simps
    apply (cases s)
    apply clarsimp
    apply (intro conjI)
    subgoal
      by (rule ex-mset)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    done
  done
have x1b-lall: ‹x1b ! 1 ∈# ℒall 𝒜›
  if
    inv: ‹(x, x')
    ∈ {((E, C, n), E', F').
        (E, E') ∈ lookup-clause-rel 𝒜 ∧
        mset C = mset outl ∧
        C ! 0 = outl ! 0 ∧
        (1 < length C ⟶
        highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
          n = length outl ∧
        I (E, C, n)}› and
    ‹x' ∈ {(E, F). E = {#} ∧ mset F = D}› and
    st:
    ‹x' = (x1, x2)›
    ‹x2a = (x1b, x2b)›
    ‹x = (x1a, x2a)› and
    ‹length outl ≠ 1 ⟶ 1 < length x1b› and
    ‹length outl ≠ 1›
  for x x' x1 x2 x1a x2a x1b x2b
proof -
  have
    ‹(x1a, x1) ∈ lookup-clause-rel 𝒜› and
    ‹mset x1b = mset outl› and
    ‹x1b ! 0 = outl ! 0› and
    ‹Suc 0 < length x1b ⟶
    highest-lit M (mset (tl x1b))
      (Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))› and
    mset-aa: ‹mset (take x2b x1b) = mset (take x2b outl)› and
```

⟨(x1a, D − mset (take x2b outl)) ∈ lookup-clause-rel 𝒜⟩ **and**
l-aa-outl: ⟨length x1b = length outl⟩ **and**
⟨x1b ! 0 = outl ! 0⟩ **and**
ba-ge1: ⟨Suc 0 ≤ x2b⟩ **and**
ba-le: ⟨x2b ≤ length outl⟩ **and**
⟨Suc 0 < length x1b ∧ Suc 0 < x2b ⟶
highest-lit M (mset (tl (take x2b x1b)))
(Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))⟩
using inv **unfolding** st I-def prod.case st
by auto

have set-aa-outl: ⟨set (take x2b x1b) = set (take x2b outl)⟩
using mset-aa **by** (blast dest: mset-eq-setD)
have ba-ge1-aa-ge: ⟨x2b > 1 ⟹ x1b ! 1 ∈ set (take x2b x1b)⟩
using ba-ge1 ba-le l-aa-outl
by (auto simp: in-set-take-conv-nth intro!: bex-lessI[of - ⟨Suc 0⟩])
then have ⟨x1b ! 1 ∈ set outl⟩
using ba-le l-aa-outl ba-ge1 that
unfolding mset-aa in-multiset-in-set[symmetric]
by (cases ⟨x2b > 1⟩)
(auto simp: mset-aa dest: in-set-takeD)
then show ?thesis
using literals-are-in-$\mathcal{L}_{in}$-in-mset-$\mathcal{L}_{all}$ lits **by** auto
qed

show ?thesis
unfolding empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-alt-def
Let-def I-def[symmetric]
apply (subst empty-conflict-and-extract-clause-alt-def)
unfolding conc-fun-RES
apply (refine-vcg WHILEIT-rule-stronger-inv[**where** R = ⟨measure (λ(-, -, i). length outl − i)⟩ **and**
I′ = ⟨I⟩] H1)
subgoal using assms **by** (auto simp: I-def)
subgoal by (rule x1b-lall)
subgoal using assms
by (auto intro!: RETURN-RES-refine simp: option-lookup-clause-rel-def I-def)
done
qed

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, -) ← $WHILE_T$
(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 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* $\to_f$ ⟨*Id*⟩*nres-rel* ›
proof −
have [*refine0*]: ‹((*x2b*, *replicate* (*length x2c*) (*x2c* ! *0*), *one-uint32-nat*), *x2*,
replicate (*length x2a*) (*x2a* ! *0*), *one-uint32-nat*)
∈ *Id* $\times_f$ *Id* $\times_f$ *Id*›
 if
 ‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id*› **and** ‹*x1* = (*x1a*, *x2*)› **and**
 ‹*y* = (*x1*, *x2a*)› **and**
 ‹*x1b* = (*x1c*, *x2b*)› **and**
 ‹*x* = (*x1b*, *x2c*)›
 for *x y x1 x1a x2 x2a x1b x1c x2b x2c*
 using *that* **by** *auto*

show *?thesis*
 supply [[*goals-limit*=*1*]]
 unfolding *isa-empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-heur-def uncurry-def*
 apply (*intro frefI nres-relI*)
 apply (*refine-rcg*)
  apply (*assumption*+)[*5*]
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal
 by (*rule get-level-pol-pre*) *auto*
 subgoal
 by (*rule get-level-pol-pre*) *auto*
 subgoal by *auto*
 subgoal by *auto*
 subgoal
 by (*auto simp*: *get-level-get-level-pol*[*of* - - $\mathcal{A}$])
 subgoal by *auto*
 subgoal
 by (*rule get-level-pol-pre*) *auto*
 subgoal by (*auto simp*: *get-level-get-level-pol*[*of* - - $\mathcal{A}$])
 done
qed

definition *extract-shorter-conflict-wl-nlit* **where**
‹*extract-shorter-conflict-wl-nlit K M NU D NE UE* =
 SPEC(λ*D*′. *D*′ ≠ *None* ∧ *the D*′ ⊆# *the D* ∧ *K* ∈# *the D*′ ∧

mset ‘# ran-mf NU + NE + UE $\models$*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* =
(λ(*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* ⇒ (′*v twl-st-wl* × *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* ⟷ (∃ *S′*. (*S*, *S′*) ∈ *twl-st-heur-conflict-ana* ∧ *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 ← *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* × *nat clause*) × *nat clause-l*) ⇒ *bool*› **where**
‹*empty-conflict-and-extract-clause-pre* =
(λ((*M*, *D*), *outl*). *D* = *mset* (*tl outl*) ∧ *outl* ≠ [] ∧ *distinct outl* ∧
¬*tautology* (*mset outl*) ∧ *length outl* ≤ *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* ∧
(∀ *L* ∈ *set* (*drop i support*). *L* < *length cach*) ∧
(∀ *L* ∈ *set* (*take i support*). *cach* ! *L* = *SEEN-UNKNOWN*) ∧
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set* (*drop i support*)))›

definition *empty-cach-ref-set* **where**
‹*empty-cach-ref-set* = (λ(*cach0*, *support*). *do* {
let n = *length support*;
ASSERT(*n* ≤ *Suc* (*uint32-max div 2*));
(-, *cach*) ← $WHILE_T{}^{empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support}$

$(\lambda(i,\ cach).\ i < length\ support)$
$(\lambda(i,\ cach).\ do\ \{$
$ASSERT(i < length\ support);$
$ASSERT(support\ !\ i < length\ cach);$
$RETURN(i+1,\ cach[support\ !\ i := SEEN\text{-}UNKNOWN])$
$\})$
$(0,\ cach0);$
$RETURN\ (cach,\ emptied\text{-}list\ support)$
$\})$›

lemma *empty-cach-ref-set-empty-cach-ref*:
‹$(empty\text{-}cach\text{-}ref\text{-}set,\ RETURN\ o\ empty\text{-}cach\text{-}ref) \in$
$[\lambda(cach,\ supp).\ (\forall L \in set\ supp.\ L < length\ cach) \wedge length\ supp \leq Suc\ (uint32\text{-}max\ div\ 2) \wedge$
$(\forall L < length\ cach.\ cach\ !\ L \neq SEEN\text{-}UNKNOWN \longrightarrow L \in set\ supp)]_f$
$Id \rightarrow \langle Id\rangle\ nres\text{-}rel$›
proof –
have H: ‹$WHILE_T{}^{empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support'}\ (\lambda(i,\ cach).\ i < length\ support')$
$(\lambda(i,\ cach).$
$ASSERT\ (i < length\ support') \gg=$
$(\lambda\text{-}.\ ASSERT\ (support'\ !\ i < length\ cach) \gg=$
$(\lambda\text{-}.\ RETURN\ (i + 1,\ cach[support'\ !\ i := SEEN\text{-}UNKNOWN]))))$
$(0,\ cach0) \gg=$
$(\lambda(\text{-},\ cach).\ RETURN\ (cach,\ emptied\text{-}list\ support'))$
$\leq \Downarrow Id\ (RETURN\ (replicate\ (length\ cach0)\ SEEN\text{-}UNKNOWN,\ []))$›
if
‹$\forall L \in set\ support'.\ L < length\ cach0$› **and**
‹$\forall L < length\ cach0.\ cach0\ !\ L \neq SEEN\text{-}UNKNOWN \longrightarrow L \in set\ support'$›
for *cach support cach0 support'*
proof –
have *init*: ‹$empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support'\ (0,\ cach0)$›
using *that* **unfolding** *empty-cach-ref-set-inv-def*
by *auto*
have *valid-length*:
‹$empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support'\ s \Longrightarrow case\ s\ of\ (i,\ cach) \Rightarrow i < length\ support' \Longrightarrow$
$s = (cach',\ sup') \Longrightarrow support'\ !\ cach' < length\ sup'$› **for** *s cach' sup'*
using *that* **unfolding** *empty-cach-ref-set-inv-def*
by *auto*
have *set-next*: ‹$empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support'\ (i + 1,\ cach'[support'\ !\ i := SEEN\text{-}UNKNOWN])$›
if
inv: ‹$empty\text{-}cach\text{-}ref\text{-}set\text{-}inv\ cach0\ support'\ s$› **and**
cond: ‹$case\ s\ of\ (i,\ cach) \Rightarrow i < length\ support'$› **and**
s: ‹$s = (i,\ cach')$› **and**
valid[*simp*]: ‹$support'\ !\ i < length\ cach'$›
for *s i cach'*
proof –
have
le-cach-cach0: ‹$length\ cach' = length\ cach0$› **and**
le-length: ‹$\forall L \in set\ (drop\ i\ support').\ L < length\ cach'$› **and**
UNKNOWN: ‹$\forall L \in set\ (take\ i\ support').\ cach'\ !\ L = SEEN\text{-}UNKNOWN$› **and**
support: ‹$\forall L < length\ cach'.\ cach'\ !\ L \neq SEEN\text{-}UNKNOWN \longrightarrow L \in set\ (drop\ i\ support')$› **and**
[*simp*]: ‹$i < length\ support'$›
using *inv cond* **unfolding** *empty-cach-ref-set-inv-def s prod.case*
by *auto*

show *?thesis*
unfolding *empty-cach-ref-set-inv-def*

unfolding *prod.case*
apply (*intro conjI*)
subgoal by (*simp add*: *le-cach-cach0*)
subgoal using *le-length* **by** (*simp add*: *Cons-nth-drop-Suc*[*symmetric*])
subgoal using *UNKNOWN* **by** (*auto simp add*: *take-Suc-conv-app-nth*)
subgoal using *support* **by** (*auto simp add*: *Cons-nth-drop-Suc*[*symmetric*])
done
qed
have *final*: ‹((*cach'*, *emptied-list support'*), *replicate* (*length cach0*) *SEEN-UNKNOWN*, []) ∈ *Id*›
if
inv: ‹*empty-cach-ref-set-inv cach0 support' s*› **and**
cond: ‹¬ (*case s of* (*i*, *cach*) ⇒ *i* < *length support'*)› **and**
s: ‹*s* = (*i*, *cach'*)›
for *s cach' i*
proof −
have
le-cach-cach0: ‹*length cach'* = *length cach0*› **and**
le-length: ‹∀ *L*∈*set* (*drop i support'*). *L* < *length cach'*› **and**
UNKNOWN: ‹∀ *L*∈*set* (*take i support'*). *cach'* ! *L* = *SEEN-UNKNOWN*› **and**
support: ‹∀ *L*<*length cach'*. *cach'* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set* (*drop i support'*)› **and**
i: ‹¬*i* < *length support'*›
using *inv cond* **unfolding** *empty-cach-ref-set-inv-def s prod.case*
by *auto*
have ‹∀ *L*<*length cach'*. *cach'* ! *L* = *SEEN-UNKNOWN*›
using *support i* **by** *auto*
then have [*dest*]: ‹*L* ∈ *set cach'* ⟹ *L* = *SEEN-UNKNOWN*› **for** *L*
by (*metis in-set-conv-nth*)
then have [*dest*]: ‹*SEEN-UNKNOWN* ∉ *set cach'* ⟹ *cach0* = [] ∧ *cach'* = []›
using *le-cach-cach0* **by** (*cases cach'*) *auto*
show *?thesis*
by (*auto simp*: *emptied-list-def list-eq-replicate-iff le-cach-cach0*)
qed
show *?thesis*
unfolding *conc-Id id-def*
apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*, -). *length support'* − *i*)›])
subgoal by *auto*
subgoal by (*rule init*)
subgoal by *auto*
subgoal by (*rule valid-length*)
subgoal by (*rule set-next*)
subgoal by *auto*
subgoal using *final* **by** *simp*
done
qed
show *?thesis*
unfolding *empty-cach-ref-set-def empty-cach-ref-def Let-def comp-def*
by (*intro frefI nres-relI ASSERT-leI*) (*clarify intro*!: *H ASSERT-leI*)

qed

lemma *empty-cach-ref-empty-cach*:
‹*isasat-input-bounded* $\mathcal{A}$ ⟹ (*RETURN o empty-cach-ref*, *RETURN o empty-cach*) ∈ *cach-refinement* $\mathcal{A}$ $\to_f$ ⟨*cach-refinement* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *empty-cach-def empty-cach-ref-def cach-refinement-alt-def cach-refinement-list-def*

map-fun-rel-def intro: *bounded-included-le*)

definition *empty-cach-ref-pre* **where**
‹*empty-cach-ref-pre* = (λ(*cach* :: *minimize-status list*, *supp* :: *nat list*).
($\forall$ *L*$\in$*set supp*. *L* < *length cach*) $\wedge$
length supp $\leq$ *Suc* (*uint-max div 2*) $\wedge$
($\forall$ *L*<*length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longrightarrow$ *L* $\in$ *set supp*))›

Minimisation of the conflict **definition** *extract-shorter-conflict-list-heur-st*
:: ‹*twl-st-wl-heur* $\Rightarrow$ (*twl-st-wl-heur* $\times$ - $\times$ -) *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* $\neq$ []);
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 $\leftarrow$ *isa-vmtf-mark-to-rescore-also-reasons M N outl vm*;
(*D*, *cach*, *outl*) $\leftarrow$ *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* $\neq$ [] $\wedge$ *length outl* $\leq$ *uint-max*);
(*D*, *C*, *n*) $\leftarrow$ *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*) $\in$ [λ*D*. *D* $\neq$ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ $\rightarrow$ $\langle$*lookup-clause-rel* $\mathcal{A}\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def*)

definition *propagate-bt-wl-D-heur*
:: ‹*nat literal* $\Rightarrow$ *nat clause-l* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *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* $\leq$ *length N0*);
ASSERT(*length avdom* $\leq$ *length N0*);
ASSERT(*nat-of-lit* (*C*!*1*) < *length W0* $\wedge$ *nat-of-lit* ($-L$) < *length W0*);
ASSERT(*length C* > *1*);
let L' = *C*!*1*;
ASSERT(*length C* $\leq$ *uint32-max div 2* + *1*);
(*vm*, φ) $\leftarrow$ *isa-vmtf-rescore C M vm0* φ*0*;
glue $\leftarrow$ *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*) $\longrightarrow$ *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*) $\longrightarrow$ *lcount* < *uint64-max*);
(*N*, *i*) $\leftarrow$ *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);
$S \leftarrow$ *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 −
obtain *M3* **where** *M3*: ‹M' = *M3* @ *M2* @ *Decided* K # *a*›
using *decomp* **by** *blast*
from *lev-K* **have** *lev-L*: ‹*get-level* M' L < *get-level* M' K›
using *get-maximum-level-ge-get-level*[*OF L*, *of* M'] **unfolding** L'[*symmetric*] **by** *auto*
have [*simp*]: ‹*get-level* (*M3* @ *M2* @ *Decided* K # *a*) K = *Suc* (*count-decided a*)›
using *nd* **unfolding** *M3* **by** *auto*
have *undef*:‹*undefined-lit* (*M3* @ *M2*) L›
using *lev-L get-level-skip-end*[*of* ‹*M3* @ *M2*› L ‹*Decided* K # *a*›] **unfolding** *M3*
by *auto*
then have ‹*atm-of* L ≠ *atm-of* K›
using *lev-L* **unfolding** *M3* **by** *auto*
then show *?thesis*
using *undef* **unfolding** *M3* **by** (*auto simp*: *get-level-cons-if*)
qed

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*›
by (*cases S*) (*auto simp*: *del-conflict-wl-def*)

lemma *lcount-add-clause*[*simp*]: ‹*i* ∉# *dom-m N* ⟹
size (*learned-clss-l* (*fmupd i* (*C*, *False*) *N*)) = *Suc* (*size* (*learned-clss-l N*))›
by (*simp add*: *learned-clss-l-mapsto-upd-notin*)

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 −
have ‹*correct-watching x1*›

using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-D-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x'a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*) (*get-unit-clauses-wl x1*))›
using *x2 xb-x'a* **unfolding** *all-atms-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur'-def twl-st-heur-conflict-ana-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def all-atms-def*[*symmetric*])
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur'-def twl-st-heur-def simp flip*: *all-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4* ..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *4*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show *?thesis*
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

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 −
have *backtrack-wl-D-nlit-heur-alt-def*: ‹*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 {
let - = C ! *1*;
propagate-bt-wl-D-heur L C S
}
else do {
propagate-unit-bt-wl-D-int L S
}
}› **for** S_0
unfolding *backtrack-wl-D-nlit-heur-def Let-def*
by *auto*
have *inv*: ‹*backtrack-wl-D-heur-inv* S'›
if
‹*backtrack-wl-D-inv* S› **and**
‹(S', S) ∈ *?R*›
for S S'
using *that* **unfolding** *backtrack-wl-D-heur-inv-def*
by (*cases* S; *cases* S') (*blast intro*: *exI*[*of* - S'])
have *shorter*:
‹*extract-shorter-conflict-list-heur-st* S'
≤ ⇓ {((T', n, C), T). (T', *del-conflict-wl* T) ∈ *twl-st-heur-bt* ∧
n = *get-maximum-level* (*get-trail-wl* T)
(*remove1-mset* (−*lit-of*(*hd* (*get-trail-wl* T))) (*the* (*get-conflict-wl* T))) ∧
mset C = *the* (*get-conflict-wl* T) ∧
get-conflict-wl T ≠ *None*∧
equality-except-conflict-wl T S ∧
get-clauses-wl-heur T' = *get-clauses-wl-heur* S' ∧
(*1* < *length* C ⟶
highest-lit (*get-trail-wl* T) (*mset* (*tl* C))
(*Some* (C ! *1*, *get-level* (*get-trail-wl* T) (C ! *1*)))) ∧
C ≠ [] ∧ *hd* C = −*lit-of*(*hd* (*get-trail-wl* T)) ∧
mset C ⊆# *the* (*get-conflict-wl* S) ∧
distinct-mset (*the* (*get-conflict-wl* S)) ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st* S) (*the* (*get-conflict-wl* S)) ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st* T) (*get-trail-wl* T) ∧
get-conflict-wl S ≠ *None* ∧
− *lit-of* (*hd* (*get-trail-wl* S)) ∈# $\mathcal{L}_{all}$ (*all-atms-st* S) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* T) T ∧
n < *count-decided* (*get-trail-wl* T) ∧
get-trail-wl T ≠ [] ∧
¬ *tautology* (*mset* C) ∧
correct-watching S ∧ *length* (*get-clauses-wl-heur* T') = *length* (*get-clauses-wl-heur* S')}
(*extract-shorter-conflict-wl* S)›
(**is** ‹- ≤ ⇓ *?shorter* -›)
if
inv: ‹*backtrack-wl-D-inv* S› **and**
S'-S: ‹(S', S) ∈ *?R*›
for S S'
proof −

obtain *M N D NE UE Q W* **where**
S: ‹*S* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)›
by (*cases S*)
obtain *M′ W′ vm φ clvls cach lbd outl stats cc cc2 cc3 avdom vdom lcount D′ arena b Q′ opts* **where**
S′: ‹*S′* = (*M′*, *arena*, (*b*, *D′*), *Q′*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *cc*, *cc2*, *cc3*, *vdom*, *avdom*, *lcount*, *opts*)›
using *S′-S* **by** (*cases S′*) (*auto simp*: *twl-st-heur-conflict-ana-def S*)
have
M′-M: ‹(*M′*, *M*) ∈ *trail-pol* (*all-atms-st S*)› **and**
‹(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st S*))› **and**
vm: ‹*vm* ∈ *isa-vmtf* (*all-atms-st S*) *M*› **and**
‹*phase-saving* (*all-atms-st S*) *φ*› **and**
n-d: ‹*no-dup M*› **and**
‹*clvls* ∈ *counts-maximum-level M D*› **and**
cach-empty: ‹*cach-refinement-empty* (*all-atms-st S*) *cach*› **and**
outl: ‹*out-learned M D outl*› **and**
lcount: ‹*lcount* = *size* (*learned-clss-l N*)› **and**
‹*vdom-m* (*all-atms-st S*) *W N* ⊆ *set vdom*› **and**
D′: ‹((*b*, *D′*), *D*) ∈ *option-lookup-clause-rel* (*all-atms-st S*)› **and**
arena: ‹*valid-arena arena N* (*set vdom*)› **and**
avdom: ‹*mset avdom* ⊆# *mset vdom*› **and**
bounded: ‹*isasat-input-bounded* (*all-atms N* (*NE* + *UE*))›
using *S′-S* **unfolding** *S S′ twl-st-heur-conflict-ana-def*
by (*auto simp*: *S all-atms-def*[*symmetric*])
obtain *T U* **where**
$\mathcal{L}_{in}$:‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S*› **and**
S-T: ‹(*S*, *T*) ∈ *state-wl-l None*› **and**
corr: ‹*correct-watching S*› **and**
T-U: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
trail-nempty: ‹*get-trail-l T* ≠ []› **and**
nss: ‹∀ *S′*. ¬ *cdcl*$_W$*-restart-mset.skip* (*state*$_W$*-of U*) *S′*› **and**
nsr: ‹∀ *S′*. ¬ *cdcl*$_W$*-restart-mset.resolve* (*state*$_W$*-of U*) *S′*› **and**
not-none: ‹*get-conflict-l T* ≠ *None*› **and**
struct-invs: ‹*twl-struct-invs U*› **and**
stgy-invs: ‹*twl-stgy-invs U*› **and**
list-invs: ‹*twl-list-invs T*› **and**
not-empty: ‹*get-conflict-l T* ≠ *Some* {#}›
using *inv* **unfolding** *backtrack-wl-D-inv-def backtrack-wl-inv-def backtrack-l-inv-def*
apply −
apply *normalize-goal*+
by *blast*
have *D-none*: ‹*D* ≠ *None*›
using *S-T not-none* **by** (*auto simp*: *S*)
have *b*: ‹¬*b*›
using *D′ not-none S-T* **by** (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def*)
have *all-struct*:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of U*)›
using *struct-invs*
by (*auto simp*: *twl-struct-invs-def*)
then have *uL-D*: ‹− *lit-of* (*hd* (*get-trail-wl S*)) ∈# *the* (*get-conflict-wl S*)›
using *cdcl*$_W$*-restart-mset.no-step-skip-hd-in-conflicting*[*of*
‹*state*$_W$*-of U*›] *nss not-none not-empty stgy-invs trail-nempty S-T T-U*
by (*auto simp*: *twl-st-wl twl-st twl-stgy-invs-def*)
have
‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)› **and**
lev-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of U*)› **and**

$\langle\forall$ *s*$\in$#*learned-clss* (*state*$_W$*-of U*). $\neg$ *tautology s*$\rangle$ **and**
dist: $\langle$*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of U*)$\rangle$ **and**
confl: $\langle$*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of U*)$\rangle$ **and**
$\langle$*all-decomposition-implies-m* (*cdcl*$_W$*-restart-mset.clauses* (*state*$_W$*-of U*))
(*get-all-ann-decomposition* (*trail* (*state*$_W$*-of U*)))$\rangle$ **and**
learned: $\langle$*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*state*$_W$*-of U*)$\rangle$
using *all-struct* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast+*
have *n-d*: $\langle$*no-dup M*$\rangle$
using *lev-inv S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *twl-st S*)
have *M-*$\mathcal{L}_{in}$: $\langle$*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)$\rangle$
apply (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*[*OF S-T struct-invs T-U* $\mathcal{L}_{in}$]) **.**
have *dist-D*: $\langle$*distinct-mset* (*the* (*get-conflict-wl S*))$\rangle$
using *dist not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def S*
by (*auto simp*: *twl-st*)
have $\langle$*the* (*conflicting* (*state*$_W$*-of U*)) =
add-mset ($-$ *lit-of* (*cdcl*$_W$*-restart-mset.hd-trail* (*state*$_W$*-of U*)))
{#*L* $\in$# *the* (*conflicting* (*state*$_W$*-of U*)). *get-level* (*trail* (*state*$_W$*-of U*)) *L*
< *backtrack-lvl* (*state*$_W$*-of U*)#}$\rangle$
apply (*rule cdcl*$_W$*-restart-mset.no-skip-no-resolve-single-highest-level*)
subgoal using *nss* **unfolding** *S* **by** *simp*
subgoal using *nsr* **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *twl-struct-invs-def S* **by** *simp*
subgoal using *stgy-invs* **unfolding** *twl-stgy-invs-def S* **by** *simp*
subgoal using *not-none S-T T-U* **by** (*simp add*: *twl-st*)
subgoal using *not-empty not-none S-T T-U* **by** (*auto simp add*: *twl-st*)
done
then have *D-filter*: $\langle$*the D* = *add-mset* ($-$ *lit-of* (*hd M*)) {#*L* $\in$# *the D*. *get-level M L* < *count-decided M*#}$\rangle$
using *trail-nempty S-T T-U* **by** (*simp add*: *twl-st S*)
have *tl-outl-D*: $\langle$*mset* (*tl* (*outl*[*0* := $-$ *lit-of* (*hd M*)])) = *remove1-mset* (*outl*[*0* := $-$ *lit-of* (*hd M*)] ! *0*) (*the D*)$\rangle$
using *outl S-T T-U not-none*
apply (*subst D-filter*)
by (*cases outl*) (*auto simp*: *out-learned-def S*)
let *?D* = $\langle$*remove1-mset* ($-$ *lit-of* (*hd M*)) (*the D*)$\rangle$
have $\mathcal{L}_{in}$*-S*: $\langle$*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*))$\rangle$
apply (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-in-*$\mathcal{L}_{in}$*-conflict*[*OF S-T - T-U*])
using $\mathcal{L}_{in}$ *not-none struct-invs not-none S-T T-U* **by** (*auto simp*: *S*)
then have $\mathcal{L}_{in}$*-D*: $\langle$*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) *?D*$\rangle$
unfolding *S* **by** (*auto intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)
have $\mathcal{L}_{in}$*-NU*: $\langle$*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* '# *ran-mf* (*get-clauses-wl S*))$\rangle$

by (*auto simp*: *all-atms-def all-lits-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*)
(*simp add*: *all-lits-of-mm-union*)
have *tauto-confl*: $\langle\neg$ *tautology* (*the* (*get-conflict-wl S*))$\rangle$
apply (*rule conflict-not-tautology*[*OF S-T - T-U*])
using *struct-invs not-none S-T T-U* **by** (*auto simp*: *twl-st*)
from *not-tautology-mono*[*OF - this*, *of ?D*] **have** *tauto-D*: $\langle\neg$ *tautology ?D*$\rangle$
by (*auto simp*: *S*)
have *entailed*:
$\langle$*mset* '# *ran-mf* (*get-clauses-wl S*) + (*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) $\models$*pm*
add-mset ($-$ *lit-of* (*hd* (*get-trail-wl S*)))
(*remove1-mset* ($-$ *lit-of* (*hd* (*get-trail-wl S*))) (*the* (*get-conflict-wl S*)))$\rangle$

using *uL-D learned not-none S-T T-U* **unfolding** *$cdcl_W$-restart-mset.$cdcl_W$-learned-clause-alt-def*
by (*auto simp*: *ac-simps twl-st get-unit-clauses-wl-alt-def*)
define *cach′* **where** ⟨*cach′* = (λ-::*nat*. *SEEN-UNKNOWN*)⟩
have *mini*: ⟨*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) (*get-trail-wl S*) (*get-clauses-wl S*)
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf* (*get-clauses-wl S*))
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) *?D*)⟩
apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*of S T U*
⟨*outl* [*0* := − *lit-of* (*hd M*)]⟩
⟨*remove1-mset* - (*the D*)⟩ ⟨*all-atms-st S*⟩ *cach′* ⟨−*lit-of* (*hd M*)⟩ *lbd*])
subgoal using *S-T* **.**
subgoal using *T-U* **.**
subgoal using ⟨*out-learned M D outl*⟩ *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *confl not-none S-T T-U* **unfolding** *$cdcl_W$-restart-mset.$cdcl_W$-conflicting-def*
by (*auto simp*: *true-annot-CNot-diff twl-st S*)
subgoal
using *dist not-none S-T T-U* **unfolding** *$cdcl_W$-restart-mset.distinct-$cdcl_W$-state-def*
by (*auto simp*: *twl-st S*)
subgoal using *tauto-D* **.**
subgoal using *M-$\mathcal{L}_{in}$* **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *S* **by** *simp*
subgoal using *list-invs* **unfolding** *S* **by** *simp*
subgoal using *M-$\mathcal{L}_{in}$ cach-empty S-T T-U*
unfolding *cach-refinement-empty-def conflict-min-analysis-inv-def*
by (*auto dest*: *literals-are-in-$\mathcal{L}_{in}$-trail-in-lits-of-l-atms simp*: *cach′-def twl-st S*)
subgoal using *entailed* **unfolding** *S* **by** *simp*
subgoal using *$\mathcal{L}_{in}$-D* **.**
subgoal using *$\mathcal{L}_{in}$-NU* **.**
subgoal using ⟨*out-learned M D outl*⟩ *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using ⟨*out-learned M D outl*⟩ *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *bounded* **unfolding** *all-atms-def* **by** (*simp add*: *S*)
done
then have *mini*: ⟨*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf N*)
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) *?D*)⟩
unfolding *S* **by** *auto*
have *mini*: ⟨*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*SPEC* (λ*D′*. *D′* ⊆# *?D* ∧ *mset* ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) ⊨*pm add-mset* (− *lit-of* (*hd M*))

D'))›
apply (*rule order.trans*)
apply (*rule mini*)
apply (*rule ref-two-step'*)
apply (*rule order.trans*)
apply (*rule iterate-over-conflict-spec*)
subgoal using *entailed* **by** (*auto simp*: *S*)
subgoal
using *dist not-none S-T T-U* **unfolding** *S* $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
by (*auto simp*: *twl-st*)
subgoal by *auto*
done
have *uM-*$\mathcal{L}_{all}$: ‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
using *M-*$\mathcal{L}_{in}$ *trail-nempty S-T T-U* **by** (*cases M*)
(*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons uminus-*$\mathcal{A}_{in}$*-iff twl-st S*)

have *L-D*: ‹*lit-of* (*hd M*) ∉# *the D*› **and**
tauto-confl': ‹¬*tautology* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*))›
using *uL-D tauto-confl*
by (*auto dest*!: *multi-member-split simp*: *S add-mset-eq-add-mset tautology-add-mset*)
then have *pre1*: ‹*D* ≠ *None* ∧ *delete-from-lookup-conflict-pre* (*all-atms-st S*) (− *lit-of* (*hd M*), *the D*)›
using *not-none uL-D uM-*$\mathcal{L}_{all}$ *S-T T-U* **unfolding** *delete-from-lookup-conflict-pre-def*
by (*auto simp*: *twl-st S*)
have *pre2*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*) ≡ *True*›
and *lits-N*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*)›
using *M-*$\mathcal{L}_{in}$ *S-T T-U not-none* $\mathcal{L}_{in}$
unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*-def all-atms-def all-lits-def*
by (*auto simp*: *twl-st S all-lits-of-mm-union*)
have ‹*0* < *length outl*›
using ‹*out-learned M D outl*›
by (*auto simp*: *out-learned-def*)
have *trail-nempty*: ‹*M* ≠ []›
using *trail-nempty S-T T-U*
by (*auto simp*: *twl-st S*)

have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of* (*hd M*), *D'*)›
using *D' not-none not-empty S-T uM-*$\mathcal{L}_{all}$
unfolding *lookup-conflict-remove1-pre-def*
by (*cases* ‹*the D*›)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def S*
state-wl-l-def atms-of-def)
then have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of-last-trail-pol M'*, *D'*)›
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of M M'*])
(*use M'-M trail-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def*›)

have ‹− *lit-of* (*hd M*) ∈# (*the D*)›
using *uL-D* **by** (*auto simp*: *S*)
then have *extract-shorter-conflict-wl-alt-def*:
‹*extract-shorter-conflict-wl* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) = *do* {
let K = *lit-of* (*hd M*);
let D = (*remove1-mset* (−*K*) (*the D*));
- ← *RETURN* (); ~~*vmtf-rescoring*~~
E' ← (*SPEC*
(λ(*E'*). *E'* ⊆# *add-mset* (−*K*) *D* ∧ − *lit-of* (*hd M*) :# *E'* ∧

mset ‘# *ran-mf* N +
(*get-unit-learned-clss-wl* S + *get-unit-init-clss-wl* S) ⊨*pm* E'));
D ← *RETURN* (*Some* E');
RETURN (M, N, D, *NE*, *UE*, Q, W)
}›
unfolding *extract-shorter-conflict-wl-def*
by (*auto simp*: *RES-RETURN-RES image-iff mset-take-mset-drop-mset′ S union-assoc Un-commute Let-def*)
have *lookup-clause-rel-unique*: ‹(D', a) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ (D', b) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ $a = b$›
for a b $\mathcal{A}$
by (*auto simp*: *lookup-clause-rel-def mset-as-position-right-unique*)
have *isa-minimize-and-extract-highest-lookup-conflict*:
‹*isa-minimize-and-extract-highest-lookup-conflict* M' *arena*
(*lookup-conflict-remove1* (−*lit-of* (*hd* M)) D') *cach lbd* (*outl*[*0* := − *lit-of* (*hd* M)])
≤ ⇓ {((E, s, *outl*), E').
(E, *mset* (*tl outl*)) ∈ *lookup-clause-rel* (*all-atms-st* S) ∧
mset outl = E' ∧
outl ! *0* = − *lit-of* (*hd* M) ∧
E' ⊆# *the* D ∧ *outl* ≠ [] ∧ *distinct outl* ∧ *literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* S) (*mset outl*) ∧
¬*tautology* (*mset outl*) ∧
(∃ *cach′*. (s, *cach′*) ∈ *cach-refinement* (*all-atms-st* S))}
(*SPEC* (λE'. E' ⊆# *add-mset* (− *lit-of* (*hd* M)) (*remove1-mset* (− *lit-of* (*hd* M)) (*the* D)) ∧
− *lit-of* (*hd* M) ∈# E' ∧
mset ‘# *ran-mf* N +
(*get-unit-learned-clss-wl* S + *get-unit-init-clss-wl* S) ⊨*pm*
E'))›
(**is** ‹- ≤ ⇓ *?minimize* (*RES ?E*)›)
apply (*rule order-trans*)
apply (*rule*
isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict
[*THEN fref-to-Down-curry5*,
of ‹*all-atms-st* S› M N ‹*remove1-mset* (−*lit-of* (*hd* M)) (*the* D)› *cach′ lbd* ‹*outl*[*0* := − *lit-of* (*hd* M)]›
- - - - - - ‹*set vdom*›])
subgoal using *bounded* **by** (*auto simp*: *S all-atms-def*)
subgoal using *tauto-confl′ pre2* **by** *auto*
subgoal using D' *not-none arena S-T uL-D uM-*$\mathcal{L}_{all}$ *not-empty* D' *L-D b cach-empty M′-M* **unfolding** *all-atms-def*
by (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def image-image cach-refinement-empty-def cach′-def*
intro!: *lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*]
dest: *multi-member-split lookup-clause-rel-unique*)
apply (*rule order-trans*)
apply (*rule mini*[*THEN ref-two-step′*])
subgoal
using *uL-D dist-D tauto-D* $\mathcal{L}_{in}$*-S* $\mathcal{L}_{in}$*-D tauto-D L-D*
by (*fastforce simp*: *conc-fun-chain conc-fun-RES image-iff S union-assoc insert-subset-eq-iff neq-Nil-conv literals-are-in-*$\mathcal{L}_{in}$*-add-mset tautology-add-mset*
intro: *literals-are-in-*$\mathcal{L}_{in}$*-mono*
dest: *distinct-mset-mono not-tautology-mono*
dest!: *multi-member-split*)
done

have *empty-conflict-and-extract-clause-heur*: ‹*isa-empty-conflict-and-extract-clause-heur* M' *x1 x2a*
≤ ⇓ ({((E, *outl*, n), E').

$(E, None) \in$ *option-lookup-clause-rel* (*all-atms-st S*) $\wedge$
mset outl = *the E'* $\wedge$
outl ! *0* = − *lit-of* (*hd M*) $\wedge$
the E' ⊆# *the D* $\wedge$ *outl* ≠ [] $\wedge$ *E'* ≠ *None* $\wedge$
(*1* < *length outl* ⟶
highest-lit M (*mset* (*tl outl*)) (*Some* (*outl* ! *1*, *get-level M* (*outl* ! *1*)))) $\wedge$
(*1* < *length outl* ⟶ *n* = *get-level M* (*outl* ! *1*)) $\wedge$ (*length outl* = *1* ⟶ *n* = *0*)}) (*RETURN* (*Some E'*))›
(**is** ‹- ≤ ⇓ *?empty-conflict* -›)
if
‹*M* ≠ []› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)› **and**
‹*0* < *length outl*› **and**
‹*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), *D'*)› **and**
‹(*x*, *E'*) ∈ *?minimize*› **and**
‹*E'* ∈ *?E*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x* = (*x1*, *x2*)›
for *x* :: ‹(*nat* × *bool option list*) × (*minimize-status list* × *nat list*) × *nat literal list*› **and**
E' :: ‹*nat literal multiset*› **and**
x1 :: ‹*nat* × *bool option list*› **and**
x2 :: ‹(*minimize-status list* × *nat list*) × *nat literal list*› **and**
x1a :: ‹*minimize-status list* × *nat list*› **and**
x2a :: ‹*nat literal list*›
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur* [*THEN fref-to-Down-curry2*, *of - - - M x1 x2a* ‹*all-atms-st S*›])
apply *fast*
subgoal using *M'-M* **by** *auto*
apply (*subst Down-id-eq*)
apply (*rule order.trans*)
apply (*rule empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause*[*of* ‹*mset* (*tl x2a*)›])
subgoal by *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *bounded* **unfolding** *S all-atms-def* **by** *simp*
subgoal unfolding *empty-conflict-and-extract-clause-def*
using *that*
by (*auto simp*: *conc-fun-RES RETURN-def*)
done
qed

have *final*: ‹(((*M'*, *arena*, *x1b*, *Q'*, *W'*, *vm'*, *φ*, *clvls*, *empty-cach-ref x1a*, *lbd*, *take 1 x2a*,
stats, *cc*, *cc2*, *cc3*, *vdom*, *avdom*, *lcount*, *opts*),
x2c, *x1c*),
M, *N*, *Da*, *NE*, *UE*, *Q*, *W*)
∈ *?shorter*›
if
‹*M* ≠ []› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)› **and**

‹0 < length outl› **and**
‹lookup-conflict-remove1-pre (− lit-of (hd M), D′)› **and**
mini: ‹(x, E′) ∈ ?minimize› **and**
‹E′ ∈ ?E› **and**
‹(xa, Da) ∈ ?empty-conflict› **and**
st[simp]:
‹x2b = (x1c, x2c)›
‹x2 = (x1a, x2a)›
‹x = (x1, x2)›
‹xa = (x1b, x2b)› **and**
vm′: ‹(vm′, uu) ∈ {(c, uu). c ∈ isa-vmtf (all-atms N (NE + UE)) M}›
for x E′ x1 x2 x1a x2a xa Da x1b x2b x1c x2c vm′ uu
proof −
have x1b-None: ‹(x1b, None) ∈ option-lookup-clause-rel (all-atms N (NE + UE))›
using that **apply** (auto simp: S simp flip: all-atms-def)
done
have [simp]: ‹cach-refinement-empty (all-atms N (NE + UE)) (empty-cach-ref x1a)›
using empty-cach-ref-empty-cach[of ‹all-atms-st S›, THEN fref-to-Down-unRET, of x1a]
mini bounded
by (auto simp add: cach-refinement-empty-def empty-cach-def cach′-def S
simp flip: all-atms-def)

have cach: ‹cach-refinement-empty (all-atms-st S) (empty-cach-ref x1a)› **and**
out: ‹out-learned M None (take (Suc 0) x2a)› **and**
x1c-Da: ‹mset x1c = the Da› **and**
Da-None: ‹Da ≠ None› **and**
Da-D: ‹the Da ⊆# the D› **and**
x1c-D: ‹mset x1c ⊆# the D› **and**
x1c: ‹x1c ≠ []› **and**
hd-x1c: ‹hd x1c = − lit-of (hd M)› **and**
highest: ‹Suc 0 < length x1c ⟹ x2c = get-level M (x1c ! 1) ∧
highest-lit M (mset (tl x1c))
(Some (x1c ! Suc 0, get-level M (x1c ! Suc 0)))› **and**
highest2: ‹length x1c = Suc 0 ⟹ x2c = 0› **and**
‹E′ = mset x2a› **and**
‹− lit-of (M ! 0) ∈ set x2a› **and**
‹(λx. mset (fst x)) ‘ set-mset (ran-m N) ∪
(set-mset (get-unit-learned-clss-wl S) ∪
set-mset (get-unit-init-clss-wl S)) ⊨p
mset x2a› **and**
‹x2a ! 0 = − lit-of (M ! 0)› **and**
‹x1c ! 0 = − lit-of (M ! 0)› **and**
‹mset x2a ⊆# the D› **and**
‹mset x1c ⊆# the D› **and**
‹x2a ≠ []› **and**
x1c-nempty: ‹x1c ≠ []› **and**
‹distinct x2a› **and**
Da: ‹Da = Some (mset x1c)› **and**
‹literals-are-in-$\mathcal{L}_{in}$ (all-atms-st S) (mset x2a)› **and**
‹¬ tautology (mset x2a)›
using that
unfolding out-learned-def
by (auto simp add: hd-conv-nth S simp flip: all-atms-def)
have Da-D′: ‹remove1-mset (− lit-of (hd M)) (the Da) ⊆# remove1-mset (− lit-of (hd M)) (the D)›
using Da-D mset-le-subtract **by** blast

have K: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*state*$_W$*-of* U)›
using *stgy-invs* **unfolding** *twl-stgy-invs-def* **by** *fast*
have ‹*get-maximum-level* M {#L ∈# *the D. get-level* M L < *count-decided* M#}
< *count-decided* M›
using *cdcl*$_W$*-restart-mset.no-skip-no-resolve-level-get-maximum-lvl-le*[*OF nss nsr all-struct K*]
not-none not-empty confl trail-nempty S-T T-U
unfolding *get-maximum-level-def* **by** (*auto simp*: *twl-st S*)
then have
‹*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*)) < *count-decided M*›
by (*subst D-filter*) *auto*
then have *max-lvl-le*:
‹*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*)) < *count-decided M*›
using *get-maximum-level-mono*[*OF Da-D'*, *of M*] **by** *auto*
have ‹((*M'*, *arena*, *x1b*, *Q'*, *W'*, *vm'*, φ, *clvls*, *empty-cach-ref x1a*, *lbd*, *take* (*Suc 0*) *x2a*,
stats, *cc*, *cc2*, *cc3*, *vdom*, *avdom*, *lcount*, *opts*),
del-conflict-wl (*M*, *N*, *Da*, *NE*, *UE*, *Q*, *W*))
∈ *twl-st-heur-bt*›
using *S'-S x1b-None cach out vm'* **unfolding** *twl-st-heur-bt-def*
by (*auto simp*: *twl-st-heur-def del-conflict-wl-def S S' twl-st-heur-bt-def*
twl-st-heur-conflict-ana-def S simp flip: *all-atms-def*)
moreover have *x2c*: ‹*x2c* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*))›
using *highest highest2 x1c-nempty hd-x1c*
by (*cases* ‹*length x1c* = *Suc 0*›; *cases x1c*)
(*auto simp*: *highest-lit-def Da mset-tl*)
moreover have ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms N* (*NE* + *UE*)) (*M*, *N*, *Some* (*mset x1c*), *NE*, *UE*, *Q*,
W)›
using $\mathcal{L}_{in}$
by (*auto simp*: *S x2c literals-are-*$\mathcal{L}_{in}$*-def blits-in-*$\mathcal{L}_{in}$*-def simp flip*: *all-atms-def*)
moreover have ‹¬*tautology* (*mset x1c*)›
using *tauto-confl not-tautology-mono*[*OF x1c-D*]
by (*auto simp*: *S x2c S'*)
ultimately show *?thesis*
using $\mathcal{L}_{in}$*-S x1c-Da Da-None dist-D D-none x1c-D x1c hd-x1c highest uM-*$\mathcal{L}_{all}$ *vm' M-*$\mathcal{L}_{in}$
max-lvl-le corr
by (*auto simp*: *S x2c S'*)
qed
have *hd-M'-M*: ‹*lit-of-last-trail-pol M'* = *lit-of* (*hd M*)›
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of M M'*])
(*use M'-M trail-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def*›)

have *vmtf-mark-to-rescore-also-reasons*:
‹*isa-vmtf-mark-to-rescore-also-reasons M' arena* (*outl*[*0* := − *lit-of* (*hd M*)]) *vm*
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) *M*})›
if
‹*M* ≠ []› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms N* (*NE* + *UE*)) *M*› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*))› **and**
‹*0* < *length outl*› **and**
‹*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), *D'*)›
proof −
have *outl-hd-tl*: ‹*outl*[*0* := − *lit-of* (*hd M*)] = − *lit-of* (*hd M*) # *tl* (*outl*[*0* := − *lit-of* (*hd M*)])›
and
[*simp*]: ‹*outl* ≠ []›
using *outl* **unfolding** *out-learned-def*
by (*cases outl*; *auto*; *fail*)+

have *uM-D*: ‹− *lit-of* (*hd M*) ∈# *the D*›
  by (*subst D-filter*) *auto*
have *mset-outl-D*: ‹*mset* (*outl*[*0* := − *lit-of* (*hd M*)]) = (*the D*)›
  by (*subst outl-hd-tl*, *subst mset.simps*, *subst tl-outl-D*, *subst D-filter*)
    (*use uM-D D-filter*[*symmetric*] **in** *auto*)
from *arg-cong*[*OF this*, *of set-mset*] **have** *set-outl-D*: ‹*set* (*outl*[*0* := − *lit-of* (*hd M*)]) = *set-mset* (*the D*)›
  by *auto*

have *outl-Lall*: ‹∀ *L*∈*set* (*outl*[*0* := − *lit-of* (*hd M*)]). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
  using $\mathcal{L}_{in}$*-S* **unfolding** *set-outl-D*
  by (*auto simp*: *S all-lits-of-m-add-mset*
      all-atms-def literals-are-in-$\mathcal{L}_{in}$*-def literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$
      dest: *multi-member-split*)
have ‹*distinct* (*outl*[*0* := − *lit-of* (*hd M*)])› **using** *dist-D* **by**(*auto simp*: *S mset-outl-D*[*symmetric*])
then have *length-outl*: ‹*length outl* ≤ *uint32-max*›
  using *bounded tauto-confl* $\mathcal{L}_{in}$*-S simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*mset* (*outl*[*0* := − *lit-of* (*hd M*)])›]
  by (*auto simp*: *out-learned-def S mset-outl-D*[*symmetric*] *uint32-max-def simp flip*: *all-atms-def*)
have *lit-annots*: ‹∀ *L*∈*set* (*outl*[*0* := − *lit-of* (*hd M*)]).
  ∀ *C*. *Propagated* (− *L*) *C* ∈ *set M* ⟶
    C ≠ *0* ⟶
    C ∈# *dom-m N* ∧
    (∀ *C*∈*set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›
  unfolding *set-outl-D*
  apply (*intro ballI allI impI conjI*)
  subgoal
    using *list-invs S-T* **unfolding** *twl-list-invs-def*
    by (*auto simp*: *S*)
  subgoal for *L C i*
    using *list-invs S-T arena lits-N literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* ‹(*all-atms N* (*NE* + *UE*))› *N C* ‹*i* − *C*›]
    unfolding *twl-list-invs-def*
    by (*auto simp*: *S arena-lifting all-atms-def*[*symmetric*])
  done
obtain *vm0* **where**
  vm-vm0: ‹(*vm*, *vm0*) ∈ *Id* $\times_f$ *distinct-atoms-rel* (*all-atms-st S*)› **and**
  vm0: ‹*vm0* ∈ *vmtf* (*all-atms-st S*) *M*›
  using *vm* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def S simp flip*: *all-atms-def*)
show *?thesis*
  apply (*cases vm*)
  apply (*rule order.trans*,
    rule isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons[*of* ‹*all-atms-st S*›,
      THEN fref-to-Down-curry3,
      of - - - vm M arena ‹*outl*[*0* := − *lit-of* (*hd M*)]› *vm0*])
  subgoal using *bounded S* **by** (*auto simp*: *all-atms-def*)
  subgoal using *vm arena M′-M vm-vm0* **by** (*auto simp*: *isa-vmtf-def*)[]
  apply (*rule order.trans*, *rule ref-two-step′*)
  apply (*rule vmtf-mark-to-rescore-also-reasons-spec*[*OF vm0 arena - outl-Lall lit-annots*])
  subgoal using *length-outl* **by** *auto*
  by (*auto simp*: *isa-vmtf-def conc-fun-RES S all-atms-def*)
qed

show *?thesis*
  unfolding *extract-shorter-conflict-list-heur-st-def*
    empty-conflict-and-extract-clause-def S S′ prod.simps hd-M′-M

apply (*rewrite at* ‹*let* - = *list-update* - - - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *empty-cach-ref* - *in* - › *Let-def*)
apply (*subst extract-shorter-conflict-wl-alt-def*)
apply (*refine-vcg isa-minimize-and-extract-highest-lookup-conflict*
empty-conflict-and-extract-clause-heur)
subgoal using *trail-nempty* **using** *M′-M* **by** (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)
subgoal using ‹*0* < *length outl*› **.**
subgoal unfolding *hd-M′-M*[*symmetric*] **by** (*rule lookup-conflict-remove1-pre*)
apply (*rule vmtf-mark-to-rescore-also-reasons*; *assumption?*)
subgoal using *trail-nempty* **.**
subgoal using *pre2* **by** (*auto simp*: *S all-atms-def*)
subgoal using *uM-$\mathcal{L}_{all}$* **by** (*auto simp*: *S all-atms-def*)
subgoal premises *p*
using *bounded p*(*5*,*7*−) **by** (*auto simp*: *S empty-cach-ref-pre-def cach-refinement-alt-def*
intro!: *IsaSAT-Lookup-Conflict*.*bounded-included-le simp*: *all-atms-def simp del*: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal using *bounded pre2*
by (*auto dest*!: *simple-clss-size-upper-div2 simp*: *uint32-max-def S all-atms-def*[*symmetric*]
simp del: *isasat-input-bounded-def*)
subgoal using *trail-nempty* **by** *fast*
subgoal using *uM-$\mathcal{L}_{all}$* **.**
apply *assumption+*
subgoal
using *trail-nempty uM-$\mathcal{L}_{all}$*
unfolding *S*[*symmetric*] *S′*[*symmetric*]
by (*rule final*)
done
qed

have *find-decomp-wl-nlit*: ‹*find-decomp-wl-st-int n T*
≤ ⇓ {(*U*, *U″*). (*U*, *U″*) ∈ *twl-st-heur-bt* ∧ *equality-except-trail-wl U″ T′* ∧
(∃ *K M2*. (*Decided K* # (*get-trail-wl U″*), *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T′*))
∧
get-level (*get-trail-wl T′*) *K* = *get-maximum-level* (*get-trail-wl T′*) (*the* (*get-conflict-wl T′*) −
{#−*lit-of* (*hd* (*get-trail-wl T′*))#}) + *1* ∧
get-clauses-wl-heur U = *get-clauses-wl-heur S*) ∧
(*get-trail-wl U″*, *get-vmtf-heur U*) ∈ (*Id* $\times_f$ (*Id* $\times_f$ (*distinct-atoms-rel* (*all-atms-st T′*))$^{-1}$)) ‘‘
(*Collect* (*find-decomp-w-ns-prop* (*all-atms-st T′*) (*get-trail-wl T′*) *n* (*get-vmtf-heur T*)))}
(*find-decomp-wl* (*lit-of* (*hd* (*get-trail-wl S′*))) *T′*)›
(**is** ‹- ≤ ⇓ *?find-decomp* -›)
if
‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-D-inv S′*› **and**
‹*backtrack-wl-D-heur-inv S*› **and**
TT′: ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› **and**
[*simp*]: ‹*nC* = (*n*, *C*)› **and**
[*simp*]: ‹*TnC* = (*T*, *nC*)›
for *S S′ TnC T′ T nC n C*
proof −
obtain *M N D NE UE Q W* **where**
T′: ‹*T′* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)›
by (*cases T′*)
obtain *M′ W′ vm φ clvls cach lbd outl stats arena D′ Q′* **where**
T: ‹*T* = (*M′*, *arena*, *D′*, *Q′*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*)›
using *TT′* **by** (*cases T*) (*auto simp*: *twl-st-heur-bt-def T′ del-conflict-wl-def*)
have

vm: ‹*vm* ∈ *isa-vmtf* (*all-atms-st* *T'*) *M*› **and**
M'M: ‹(*M'*, *M*) ∈ *trail-pol* (*all-atms-st* *T'*)› **and**
lits-trail: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st* *T'*) (*get-trail-wl* *T'*)›
using *TT'* **by** (*auto simp*: *twl-st-heur-bt-def del-conflict-wl-def*
all-atms-def[*symmetric*] *T T'*)

obtain *vm0* **where**
vm: ‹(*vm*, *vm0*) ∈ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st* *T'*)› **and**
vm0: ‹*vm0* ∈ *vmtf* (*all-atms-st* *T'*) *M*›
using *vm* **unfolding** *isa-vmtf-def* **by** (*cases vm*) *auto*

have *n*: ‹*n* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*mset C*))› **and**
eq: ‹*equality-except-conflict-wl* *T'* *S'*› **and**
‹*the D* = *mset C*› ‹*D* ≠ *None*› **and**
clss-eq: ‹*get-clauses-wl-heur S* = *arena*› **and**
n: ‹*n* < *count-decided* (*get-trail-wl* *T'*)› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st* *T'*)› **and**
T-T': ‹(*T*, *del-conflict-wl* *T'*) ∈ *twl-st-heur-bt*› **and**
n2: ‹*n* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*))›
using *TT'* **by** (*auto simp*: *T T' twl-st-heur-bt-def del-conflict-wl-def simp flip*: *all-atms-def*
simp del: *isasat-input-bounded-def*)
have [*simp*]: ‹*get-trail-wl* *S'* = *M*›
using *eq* ‹*the D* = *mset C*› ‹*D* ≠ *None*› **by** (*cases S'*; *auto simp*: *T'*)
have [*simp*]: ‹*get-clauses-wl-heur S* = *arena*›
using *TT'* **by** (*auto simp*: *T T'*)

have *n-d*: ‹*no-dup M*›
using *M'M* **unfolding** *trail-pol-def* **by** *auto*

have [*simp*]: ‹*NO-MATCH* [] *M* ⟹ *out-learned M None ai* ⟷ *out-learned* [] *None ai*› **for** *M ai*
by (*auto simp*: *out-learned-def*)

show *?thesis*
unfolding *T' find-decomp-wl-st-int-def prod.case T*
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*, *of M n vm0*
- - - ‹*all-atms-st* *T'*›])
subgoal using *n* **by** (*auto simp*: *T'*)
subgoal using *M'M vm* **by** *auto*
apply (*rule order.trans*)
apply (*rule ref-two-step'*)
apply (*rule find-decomp-wl-imp-le-find-decomp-wl'*)
subgoal using *vm0* **.**
subgoal using *lits-trail* **by** (*auto simp*: *T'*)
subgoal using *n* **by** (*auto simp*: *T'*)
subgoal using *n-d* **.**
subgoal using *bounded* **.**
unfolding *find-decomp-w-ns-def conc-fun-RES*
apply (*rule order.refl*)
using *T-T' n-d*
apply (*cases* ‹*get-vmtf-heur T*›)
apply (*auto simp*: *find-decomp-wl-def twl-st-heur-bt-def T T' del-conflict-wl-def*
dest: *no-dup-appendD*
simp flip: *all-atms-def n2*

```
        intro!: RETURN-RES-refine
        intro: isa-vmtfI)
    apply (rule-tac x=an in exI)
    apply (auto dest: no-dup-appendD intro: isa-vmtfI)
    apply (auto simp: Image-iff)
    done
qed

have fst-find-lit-of-max-level-wl: ‹RETURN (C ! 1)
    ≤ ⇓ Id
        (find-lit-of-max-level-wl U'
          (lit-of (hd (get-trail-wl S'))))›
  if
    ‹(S, S') ∈ ?R› and
    ‹backtrack-wl-D-inv S'› and
    ‹backtrack-wl-D-heur-inv S› and
    ‹(TnC, T') ∈ ?shorter S' S› and
    [simp]: ‹nC = (n, C)› and
    [simp]: ‹TnC = (T, nC)› and
    find-decomp: ‹(U, U') ∈ ?find-decomp S T' n› and
    size-C: ‹1 < length C› and
    size-conflict-U': ‹1 < size (the (get-conflict-wl U'))›
  for S S' TnC T' T nC n C U U'
proof -
  obtain M N NE UE Q W where
    T': ‹T' = (M, N, Some (mset C), NE, UE, Q, W)› and
    ‹C ≠ []›
    using ‹(TnC, T') ∈ ?shorter S' S› ‹1 < length C› find-decomp
    apply (cases U'; cases T'; cases S')
    by (auto simp: find-lit-of-max-level-wl-def)

  obtain M' K M2 where
    U': ‹U' = (M', N, Some (mset C), NE, UE, Q, W)› and
    decomp: ‹(Decided K # M', M2) ∈ set (get-all-ann-decomposition M)› and
    lev-K: ‹get-level M K = Suc (get-maximum-level M (remove1-mset (- lit-of (hd M)) (the (Some
(mset C)))))›
    using ‹(TnC, T') ∈ ?shorter S' S› ‹1 < length C› find-decomp
    by (cases U'; cases S')
      (auto simp: find-lit-of-max-level-wl-def T')

  have n-d: ‹no-dup (get-trail-wl S')›
    using ‹(S, S') ∈ ?R›
    by (auto simp: twl-st-heur-conflict-ana-def trail-pol-def)

  have [simp]: ‹get-trail-wl S' = get-trail-wl T'›
    using ‹(TnC, T') ∈ ?shorter S' S› ‹1 < length C› find-decomp
    by (cases T'; cases S'; auto simp: find-lit-of-max-level-wl-def U'; fail)+
  have [simp]: ‹remove1-mset (- lit-of (hd M)) (mset C) = mset (tl C)›
    apply (subst mset-tl)
    using ‹(TnC, T') ∈ ?shorter S' S›
    by (auto simp: find-lit-of-max-level-wl-def U' highest-lit-def T')

  have n-d: ‹no-dup M›
    using ‹(TnC, T') ∈ ?shorter S' S› n-d unfolding T'
    by (cases S') auto
```

have *nempty*[*iff*]: ‹*remove1-mset* (− *lit-of* (*hd M*)) (*the* (*Some*(*mset C*))) ≠ {#}›
using *U′ T′ find-decomp size-C* **by** (*cases C*) (*auto simp*: *remove1-mset-empty-iff*)
have *H*[*simp*]: ‹*aa* ∈# *remove1-mset* (− *lit-of* (*hd M*)) (*the* (*Some*(*mset C*))) ⟹
get-level M′ aa = *get-level M aa*› **for** *aa*
apply (*rule get-all-ann-decomposition-get-level*[*of* ‹*lit-of* (*hd M*)› - *K* - *M2* ‹*the* (*Some*(*mset C*))›])
subgoal ..
subgoal by (*rule n-d*)
subgoal by (*rule decomp*)
subgoal by (*rule lev-K*)
subgoal by *simp*
done

have ‹*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*mset C*)) =
get-maximum-level M′ (*remove1-mset* (− *lit-of* (*hd M*)) (*mset C*))›
by (*rule get-maximum-level-cong*) *auto*
then show *?thesis*
using ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› ‹*1* < *length C*› *hd-conv-nth*[*OF* ‹*C* ≠ []›, *symmetric*]
by (*auto simp*: *find-lit-of-max-level-wl-def U′ highest-lit-def T′*)
qed
have *propagate-bt-wl-D-heur*: ‹*propagate-bt-wl-D-heur* (*lit-of-hd-trail-st-heur S*) *C U*
≤ ⇓ *?S* (*propagate-bt-wl-D* (*lit-of* (*hd* (*get-trail-wl S′*))) *L′ U′*)›
if
SS′: ‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-D-inv S′*› **and**
‹*backtrack-wl-D-heur-inv S*› **and**
‹(*TnC*, *T′*) ∈ *?shorter S′ S*› **and**
[*simp*]: ‹*nC* = (*n*, *C*)› **and**
[*simp*]: ‹*TnC* = (*T*, *nC*)› **and**
find-decomp: ‹(*U*, *U′*) ∈ *?find-decomp S T′ n*› **and**
le-C: ‹*1* < *length C*› **and**
‹*1* < *size* (*the* (*get-conflict-wl U′*))› **and**
C-L′: ‹(*C* ! *1*, *L′*) ∈ *nat-lit-lit-rel*›
for *S S′ TnC T′ T nC n C U U′ L′*
proof −
have
TT′: ‹(*T*, *del-conflict-wl T′*) ∈ *twl-st-heur-bt*› **and**
n: ‹*n* = *get-maximum-level* (*get-trail-wl T′*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T′*))) (*mset C*))› **and**
T-C: ‹*get-conflict-wl T′* = *Some* (*mset C*)› **and**
T′S′: ‹*equality-except-conflict-wl T′ S′*› **and**
C-nempty: ‹*C* ≠ []› **and**
hd-C: ‹*hd C* = − *lit-of* (*hd* (*get-trail-wl T′*))› **and**
highest: ‹*highest-lit* (*get-trail-wl T′*) (*mset* (*tl C*))
(*Some* (*C* ! *Suc 0*, *get-level* (*get-trail-wl T′*) (*C* ! *Suc 0*)))› **and**
incl: ‹*mset C* ⊆# *the* (*get-conflict-wl S′*)› **and**
dist-S′: ‹*distinct-mset* (*the* (*get-conflict-wl S′*))› **and**
list-confl-S′: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S′*) (*the* (*get-conflict-wl S′*))› **and**
‹*get-conflict-wl S′* ≠ *None*› **and**
uM-$\mathcal{L}_{all}$: ‹−*lit-of* (*hd* (*get-trail-wl S′*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st S′*)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st T′*) *T′*› **and**
tr-nempty: ‹*get-trail-wl T′* ≠ []› **and**
r: ‹*length* (*get-clauses-wl-heur S*) = *r*› ‹*length* (*get-clauses-wl-heur T*) = *r*› **and**
corr: ‹*correct-watching S′*›
using ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› ‹*1* < *length C*› ‹(*S*, *S′*) ∈ *?R*›
by *auto*

obtain *K M2* **where**
UU': ‹$(U, U') \in$ *twl-st-heur-bt*› **and**
U'U': ‹*equality-except-trail-wl U' T'*› **and**
lev-K: ‹*get-level* (*get-trail-wl T'*) *K* = *Suc* (*get-maximum-level* (*get-trail-wl T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T'*)))
(*the* (*get-conflict-wl T'*))))› **and**
decomp: ‹(*Decided K* # *get-trail-wl U'*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T'*))›
and
r': ‹*length* (*get-clauses-wl-heur U*) = *r*› **and**
S-arena: ‹*get-clauses-wl-heur U* = *get-clauses-wl-heur S*›
using *find-decomp r*
by *auto*

obtain *M N NE UE Q W* **where**
T': ‹*T'* = (*M*, *N*, *Some* (*mset C*), *NE*, *UE*, *Q*, *W*)› **and**
‹*C* ≠ []›
using *TT' T-C* ‹*1* < *length C*›
apply (*cases T'*; *cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def*)
obtain *D* **where**
S': ‹*S'* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)›
using *T'S'* ‹*1* < *length C*›
apply (*cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T' del-conflict-wl-def*)

obtain *M1* **where**
U': ‹*U'* = (*M1*, *N*, *Some* (*mset C*), *NE*, *UE*, *Q*, *W*)› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*mset C*)› **and**
‹*mset C* ⊆# *the* (*get-conflict-wl S'*)› **and**
tauto: ‹¬ *tautology* (*mset C*)›
using ‹(*TnC*, *T'*) ∈ *?shorter S' S*› ‹*1* < *length C*› *find-decomp*
apply (*cases U'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T' intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)
obtain *M1' vm' W' φ clvls cach lbd outl stats fema sema ccount avdom vdom lcount arena D' Q' opts*
where
U: ‹*U* = (*M1'*, *arena*, *D'*, *Q'*, *W'*, *vm'*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*, *ccount*, *vdom*, *avdom*, *lcount*, *opts*, [])›
using *UU' find-decomp* **by** (*cases U*) (*auto simp*: *U' T' twl-st-heur-bt-def all-atms-def*[*symmetric*])
have
M1'-M1: ‹(*M1'*, *M1*) ∈ *trail-pol* (*all-atms-st U'*)› **and**
W'W: ‹(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st U'*))› **and**
vmtf: ‹*vm'* ∈ *isa-vmtf* (*all-atms-st U'*) *M1*› **and**
φ: ‹*phase-saving* (*all-atms-st U'*) *φ*› **and**
n-d-M1: ‹*no-dup M1*› **and**
empty-cach: ‹*cach-refinement-empty* (*all-atms-st U'*) *cach*› **and**
‹*length outl* = *Suc 0*› **and**
outl: ‹*out-learned M1 None outl*› **and**
vdom: ‹*vdom-m* (*all-atms-st U'*) *W N* ⊆ *set vdom*› **and**
lcount: ‹*lcount* = *size* (*learned-clss-l N*)› **and**
vdom-m: ‹*vdom-m* (*all-atms-st U'*) *W N* ⊆ *set vdom*› **and**
D': ‹(*D'*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st U'*)› **and**
valid: ‹*valid-arena arena N* (*set vdom*)› **and**
avdom: ‹*mset avdom* ⊆# *mset vdom*› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st U'*)› **and**
nempty: ‹*isasat-input-nempty* (*all-atms-st U'*)› **and**

dist-vdom: ⟨*distinct vdom*⟩
using *UU'* **by** (*auto simp*: *out-learned-def twl-st-heur-bt-def U U' all-atms-def*[*symmetric*])
have [*simp*]: ⟨*C* ! *1* = *L'*⟩ ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩ **and**
n-d: ⟨*no-dup M*⟩
using *SS' C-L' hd-C* ⟨*C* ≠ []⟩ **by** (*auto simp*: *S' U' T' twl-st-heur-conflict-ana-def hd-conv-nth*)
have *undef*: ⟨*undefined-lit M1* (*lit-of* (*hd M*))⟩
using *decomp n-d*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend simp*: *T' hd-append U' neq-Nil-conv*
split: *if-splits*)
have *C-1-neq-hd*: ⟨*C* ! *Suc 0* ≠ − *lit-of* (*hd M*)⟩
using *distinct-mset-mono*[*OF incl dist-S'*] *C-L'* ⟨*1* < *length C*⟩ ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩
by (*cases C*; *cases* ⟨*tl C*⟩) (*auto simp del*: ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩)
have *H*: ⟨(*RES* ((λ*i*. (*fmupd i* (*C*, *False*) *N*, *i*)) ‘ {*i*. *0* < *i* ∧ *i* ∉# *dom-m N*}) ≫=
(λ(*N*, *i*). *ASSERT* (*i* ∈# *dom-m N*) ≫= (λ-. *f N i*))) =
(*RES* ((λ*i*. (*fmupd i* (*C*, *False*) *N*, *i*)) ‘ {*i*. *0* < *i* ∧ *i* ∉# *dom-m N*}) ≫=
(λ(*N*, *i*). *f N i*))⟩ (**is** ⟨*?A* = *?B*⟩) **for** *f C N*
proof −
have ⟨*?B* ≤ *?A*⟩
by (*force intro*: *ext complete-lattice-class.Sup-subset-mono*
simp: *intro-spec-iff bind-RES*)
moreover have ⟨*?A* ≤ *?B*⟩
by (*force intro*: *ext complete-lattice-class.Sup-subset-mono*
simp: *intro-spec-iff bind-RES*)
ultimately show *?thesis* **by** *auto*
qed

have *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 - = *C*;
let b = *False*;
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 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*) @ [(*i*, *L'*, *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*) ⟶ *length-ll W* (*nat-of-lit L'*) < *uint64-max*);
let W = *W*[*nat-of-lit L'* := *W*!*nat-of-lit L'* @ [(*i*, −*L*, *length C* = *2*)]];
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-uint64-conv i], Suc lcount, opts)
    })›
    unfolding propagate-bt-wl-D-heur-def Let-def
    by auto
  have propagate-bt-wl-D-alt-def:
    ‹propagate-bt-wl-D (lit-of (hd (get-trail-wl S′))) L′ U′ = do {
        - ← RETURN (); ~~phase saving~~
        - ← RETURN (); ~~LBD~~
        D′′ ←
          list-of-mset2 (− lit-of (hd (get-trail-wl S′))) L′
            (the (Some (mset C)));
        (N, i) ← SPEC
             (λ(N′, i). N′ = fmupd i (D′′, False) N ∧ 0 < i ∧
                  i ∉# dom-m N ∧
                  (∀ L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE)).
                      i ∉ fst ‘ set (W L)));
        - ← RETURN (); ~~lbd empty~~
        - ← RETURN (); ~~lbd empty~~
       M2 ← RETURN (cons-trail-Propagated (− lit-of (hd (get-trail-wl S′))) i M1);
        - ← RETURN (); ~~vmtf-flush~~
        RETURN
          (M2,
            N, None, NE, UE, unmark (hd (get-trail-wl S′)),
            W(− lit-of (hd (get-trail-wl S′)) :=
               W (− lit-of (hd (get-trail-wl S′))) @ [(i, L′, length D′′ = 2)],
             L′ := W L′ @ [(i, − lit-of (hd (get-trail-wl S′)), length D′′ = 2)]))
      }›
    unfolding propagate-bt-wl-D-def Let-def cons-trail-Propagated-def
      U U′ H get-fresh-index-wl-def prod.case
      propagate-bt-wl-D-heur-def propagate-bt-wl-D-def Let-def rescore-clause-def
    by (auto simp: U′ RES-RES2-RETURN-RES RES-RETURN-RES φ uminus-𝒜in-iff
        uncurry-def RES-RES-RETURN-RES
        get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 list-of-mset2-def)
  have [refine0]: ‹SPEC (λ(vm′, φ′). vm′ ∈ vmtf 𝒜 M1 ∧ phase-saving 𝒜 φ′)
    ≤ ⇓{((vm′, φ′), ()). vm′ ∈ vmtf 𝒜 M1 ∧ phase-saving 𝒜 φ′} (RETURN ())› for 𝒜
    by (auto intro!: RES-refine simp: RETURN-def)

  obtain vm0 where
    vm: ‹(vm′, vm0) ∈ Id ×r distinct-atoms-rel (all-atms-st U′)› and
    vm0: ‹vm0 ∈ vmtf (all-atms-st U′) M1›
    using vmtf unfolding isa-vmtf-def by (cases vm′) auto
  have [refine0]:
    ‹isa-vmtf-rescore C M1′ vm′ φ ≤ SPEC (λc. (c, ()) ∈ {((vm, φ), -).
       vm ∈ isa-vmtf (all-atms-st U′) M1 ∧
      phase-saving (all-atms-st U′) φ})›
    apply (rule order.trans)
     apply (rule isa-vmtf-rescore[of ‹all-atms-st U′›, THEN fref-to-Down-curry3, of - - - - C M1 vm0
φ])
```

subgoal using *bounded* **by** *auto*
subgoal using *vm M1'-M1* **by** *auto*
apply (*rule order.trans*)
apply (*rule ref-two-step'*)
apply (*rule vmtf-rescore-score-clause*[*THEN fref-to-Down-curry3*, *of* ⟨*all-atms-st U'*⟩ *C M1 vm0* φ])
subgoal using *vm0 lits-confl* φ **by** (*auto simp*: *S' U'*)
subgoal using *vm M1'-M1* **by** *auto*
subgoal by (*auto simp*: *rescore-clause-def conc-fun-RES intro*!: *isa-vmtfI*)
done

have [*refine0*]: ⟨*isa-vmtf-flush-int* (*cons-trail-Propagated-tr L i M1'*) *vm* ≤
SPEC(λ*c*. (*c*, ()) ∈ {(*vm'*, -). *vm'* ∈ *isa-vmtf* (*all-atms-st U'*) (*cons-trail-Propagated L i M1*)})⟩
if *vm*: ⟨*vm* ∈ *isa-vmtf* (*all-atms-st U'*) *M1*⟩ **and**
L: ⟨*L* ∈# $\mathcal{L}_{all}$ (*all-atms-st U'*)⟩ **and**
undef: ⟨*undefined-lit M1 L*⟩ **and**
i: ⟨*i* ≠ *DECISION-REASON*⟩
for *vm i L*
proof −
let *?M1'* = ⟨*cons-trail-Propagated-tr L i M1'*⟩
let *?M1* = ⟨*cons-trail-Propagated L i M1*⟩

have *M1'-M1*: ⟨(*?M1'*, *?M1*) ∈ *trail-pol* (*all-atms-st U'*)⟩
unfolding *cons-trail-Propagated-def*
by (*rule cons-trail-Propagated-tr2*[*OF M1'-M1 L undef i*])

have *vm*: ⟨*vm* ∈ *isa-vmtf* (*all-atms-st U'*) *?M1*⟩
using *vm* **by** (*auto simp*: *isa-vmtf-def cons-trail-Propagated-def dest*: *vmtf-consD*)
obtain *vm0* **where**
vm: ⟨(*vm*, *vm0*) ∈ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st U'*)⟩ **and**
vm0: ⟨*vm0* ∈ *vmtf* (*all-atms-st U'*) *?M1*⟩
using *vm* **unfolding** *isa-vmtf-def* **by** (*cases vm*) *auto*
show *?thesis*
apply (*rule order.trans*)
apply (*rule isa-vmtf-flush-int*[*THEN fref-to-Down-curry*, *of* - - *?M1*])
apply ((*solves* ⟨*use M1'-M1 in auto*⟩)+)[*2*]
apply (*subst Down-id-eq*)
apply (*rule order.trans*)
apply (*rule vmtf-change-to-remove-order'*[*THEN fref-to-Down-curry*, *of* ⟨*all-atms-st U'*⟩ *?M1 vm0 ?M1 vm*])
subgoal using *vm0 bounded nempty* **by** *auto*
subgoal using *vm* **by** *auto*
subgoal by (*auto simp*: *vmtf-flush-def conc-fun-RES RETURN-def intro*: *isa-vmtfI*)
done
qed

have [*refine0*]: ⟨(*isa-length-trail M1'*, ()) ∈ {(*j*, -). *j* = *length M1*}⟩
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*, *OF* - *M1'-M1*]) *auto*
have [*refine0*]: ⟨*get-LBD lbd* ≤ ⇓ {(-, -). *True*}(*RETURN* ())⟩
unfolding *get-LBD-def* **by** (*auto intro*!: *RES-refine simp*: *RETURN-def*)
have [*refine0*]: ⟨*RETURN C*
≤ ⇓ *Id*
(*list-of-mset2* (− *lit-of* (*hd* (*get-trail-wl S'*))) *L'*
(*the* (*Some* (*mset C*))))⟩
using *that*
by (*auto simp*: *list-of-mset2-def S'*)

have [*simp*]: ⟨*0* < *header-size* D''⟩ **for** D''
by (*auto simp*: *header-size-def*)
have [*simp*]: ⟨*length arena* + *header-size* D'' ∉ *set vdom*⟩
⟨*length arena* + *header-size* D'' ∉ *vdom-m* (*all-atms-st* U') *W N*⟩
⟨*length arena* + *header-size* D'' ∉# *dom-m N*⟩ **for** D''
using *valid-arena-in-vdom-le-arena*(*1*)[*OF valid*] *vdom*
by (*auto 5 1 simp*: *vdom-m-def*)
have *add-new-alt-def*: ⟨(*SPEC*
(λ(N', *i*).
N' = *fmupd i* (D'', *False*) *N* ∧
0 < *i* ∧
i ∉# *dom-m N* ∧
(∀ *L*∈#*all-lits-of-mm* (*mset* '# *ran-mf N* + (*NE* + *UE*)).
i ∉ *fst* ' *set* (*W L*)))) =
(*SPEC*
(λ(N', *i*).
N' = *fmupd i* (D'', *False*) *N* ∧
0 < *i* ∧
i ∉ *vdom-m* (*all-atms-st* U') *W N*))⟩ **for** D''
using *lits*
by (*auto simp*: T' *vdom-m-def literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* U' *all-atms-def all-lits-def*)
have [*refine0*]: ⟨*fm-add-new False C arena*
≤ ⇓ {((*arena*′, *i*), (N', *i*′)). *valid-arena arena*′ N' (*insert i* (*set vdom*)) ∧ *i* = *i*′ ∧
i ∉# *dom-m N* ∧ *i* ∉ *set vdom* ∧ *length arena*′ = *length arena* + *header-size* D'' + *length*
D''}
(*SPEC*
(λ(N', *i*).
N' = *fmupd i* (D'', *False*) *N* ∧
0 < *i* ∧
i ∉# *dom-m N* ∧
(∀ *L*∈#*all-lits-of-mm* (*mset* '# *ran-mf N* + (*NE* + *UE*)).
i ∉ *fst* ' *set* (*W L*))))⟩
if ⟨(*C*, D'') ∈ *Id*⟩ **for** D''
apply (*subst add-new-alt-def*)
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *that valid le-C vdom*
by (*auto simp*: *intro*!: *RETURN-RES-refine valid-arena-append-clause*)
have [*refine0*]:
⟨*lbd-empty lbd* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* = *replicate* (*length lbd*) *False*})⟩
by (*auto simp*: *lbd-empty-def*)
have [*of - -* ⟨*all-atms-st* U'⟩, *refine0*]: ⟨*undefined-lit M L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *C* ≠ *DECISION-REASON*
⟹
(((L', C'), M'), (*L*, *C*), *M*) ∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *trail-pol* $\mathcal{A}$ ⟹
RETURN (*cons-trail-Propagated-tr* L' C' M')
≤ ⇓ {(*M0*, *M0*′′). (*M0*, *M0*′′) ∈ *trail-pol* $\mathcal{A}$ ∧ *M0*′′ = *Propagated L* C' # *M*}
(*RETURN* (*cons-trail-Propagated L C M*))⟩ **for** *C* C' :: *nat* **and** *L* **and** L' **and** *M* M' $\mathcal{A}$
using *cons-trail-Propagated-tr*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*, *of L C M* L' C' M']
by (*auto simp*: *cons-trail-Propagated-def*)

have ⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* S') (*mset C*)⟩
using *incl list-confl-*S' *literals-are-in-*$\mathcal{L}_{in}$*-mono* **by** *blast*
then have *C-Suc1-in*: ⟨*C* ! *Suc 0* ∈# $\mathcal{L}_{all}$ (*all-atms-st* S')⟩
using ⟨*1* < *length C*⟩
by (*cases C*; *cases* ⟨*tl C*⟩) (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
then have ⟨*nat-of-lit* (*C* ! *Suc 0*) < *length* W'⟩ ⟨*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl* S'))) < *length*

W′⟩ **and**
W′-eq: ⟨*W′* ! (*nat-of-lit* (*C* ! *Suc 0*)) = *W* (*C*! *Suc 0*)⟩
⟨*W′* ! (*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S′*)))) = *W* (− *lit-of* (*hd* (*get-trail-wl S′*)))⟩
using *uM-*$\mathcal{L}_{all}$ *W′W* **unfolding** *map-fun-rel-def* **by** (*auto simp*: *image-image S′ U′*)
have *le-C-ge*: ⟨*length C* ≤ *uint32-max div 2* + *1*⟩
using *clss-size-uint-max*[*OF bounded*, *of* ⟨*mset C*⟩] ⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S′*) (*mset C*)⟩ *list-confl-S′*
dist-S′ incl size-mset-mono[*OF incl*] *distinct-mset-mono*[*OF incl*]
simple-clss-size-upper-div2[*OF bounded - - tauto*]
by (*auto simp*: *uint32-max-def S′ U′ all-atms-def*[*symmetric*])
have *tr-SS′*: ⟨(*get-trail-wl-heur S*, *M*) ∈ *trail-pol* (*all-atms-st S′*)⟩
using ⟨(*S*, *S′*) ∈ *?R*⟩ **unfolding** *twl-st-heur-conflict-ana-def*
by (*auto simp*: *all-atms-def S′*)
have *hd-tr-S-M*: ⟨*lit-of-hd-trail-st-heur S* = *lit-of-hd-trail M*⟩
unfolding *lit-of-hd-trail-def lit-of-hd-trail-st-heur-def*
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *OF - tr-SS′*])
(*use tr-nempty* **in** ⟨*auto simp*: *lit-of-hd-trail-def T′*⟩)
have *All-atms-rew*: ⟨*set-mset* (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*)) =
set-mset (*all-atms N* (*NE* + *UE*))⟩ (**is** *?A*)
⟨*trail-pol* (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*)) =
trail-pol (*all-atms N* (*NE* + *UE*))⟩ (**is** *?B*)
⟨*isa-vmtf* (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*)) =
isa-vmtf (*all-atms N* (*NE* + *UE*))⟩ (**is** *?C*)
⟨*option-lookup-clause-rel* (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*)) =
option-lookup-clause-rel (*all-atms N* (*NE* + *UE*))⟩ (**is** *?D*)
⟨⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE*)))⟩ (**is** *?E*)
⟨*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))⟩
⟨*phase-saving* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
phase-saving ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?F*)
⟨*cach-refinement-empty* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
cach-refinement-empty ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?G*)
⟨*cach-refinement-nonull* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
cach-refinement-nonull ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?G2*)
⟨*vdom-m* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
vdom-m ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?H*)
⟨*isasat-input-bounded* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
isasat-input-bounded ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?I*)
⟨*isasat-input-nempty* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
isasat-input-nempty ((*all-atms N* (*NE* + *UE*)))⟩ (**is** *?J*)
if ⟨*x′* ∉# *dom-m N*⟩ **and** *C*: ⟨*C′* = *C*⟩ **for** *b x′ C′*
proof −
show *A*: *?A*
using ⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S′*) (*mset C*)⟩ *that*
by (*auto simp*: *all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*
U′ S′ in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literals-are-in-*$\mathcal{L}_{in}$*-def*)
have *A2*: ⟨*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x′* (*C*, *b*) *N*) (*NE* + *UE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))⟩
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
then show ⟨*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) (*NE* + *UE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))⟩
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
have *A3*: ⟨*set-mset* (*all-atms* (*fmupd x′* (*C*, *b*) *N*) (*NE* + *UE*)) =
set-mset (*all-atms N* (*NE* + *UE*))⟩
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)

show *?B* **and** *?C* **and** *?D* **and** *?E* **and** *?F* **and** *?G* **and** *?G2* **and** *?H* **and** *?I* **and** *?J*
unfolding *trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def*
isa-vmtf-def vmtf-def distinct-atoms-rel-def vmtf-$\mathcal{L}_{all}$-def atms-of-def
distinct-hash-atoms-rel-def
atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
cach-refinement-def
cach-refinement-list-def vdom-m-def
isasat-input-bounded-def
isasat-input-nempty-def cach-refinement-nonull-def
unfolding *trail-pol-def*[*symmetric*] *ann-lits-split-reasons-def*[*symmetric*]
isasat-input-bounded-def[*symmetric*]
vmtf-def[*symmetric*]
isa-vmtf-def[*symmetric*]
distinct-atoms-rel-def[*symmetric*]
vmtf-$\mathcal{L}_{all}$-def[*symmetric*] *atms-of-def*[*symmetric*]
distinct-hash-atoms-rel-def[*symmetric*]
atoms-hash-rel-def[*symmetric*]
option-lookup-clause-rel-def[*symmetric*]
lookup-clause-rel-def[*symmetric*]
phase-saving-def[*symmetric*] *cach-refinement-empty-def*[*symmetric*]
cach-refinement-def[*symmetric*] *cach-refinement-nonull-def*[*symmetric*]
cach-refinement-list-def[*symmetric*]
vdom-m-def[*symmetric*]
isasat-input-bounded-def[*symmetric*]
isasat-input-nempty-def[*symmetric*]
apply *auto*
done
qed

have *arena-le*: ‹*length arena* + *header-size C* + *length C* ≤ *6* + *r* + *uint32-max div 2*›
using *r r′ le-C-ge* **by** (*auto simp*: *uint32-max-def header-size-def S′ U*)
have *vm*: ‹*vm* ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) *M1* ⟹
vm ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) (*Propagated* (− *lit-of* (*hd M*)) *x2a* # *M1*)› **for** *x2a vm*
by (*cases vm*)
(*auto intro*!: *vmtf-consD simp*: *isa-vmtf-def*)
then show *?thesis*
supply [[*goals-limit=1*]]
using *empty-cach n-d-M1 C-L′ W′W outl vmtf undef* ‹*1* < *length C*› *lits*
uM-$\mathcal{L}_{all}$ vdom lcount vdom-m dist-vdom
apply (*subst propagate-bt-wl-D-alt-def*)
unfolding *U U′ H get-fresh-index-wl-def prod.case*
propagate-bt-wl-D-heur-alt-def rescore-clause-def hd-tr-S-M
apply (*rewrite* **in** ‹*let* - = -!*1 in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *update-lbd* - - - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *False in* -› *Let-def*)
apply (*refine-rcg cons-trail-Propagated-tr*[*THEN fref-to-Down-unRET-uncurry2*, *of* ‹*all-atms-st U′*›])
subgoal using *valid* **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using *valid size-mset-mono*[*OF avdom*] **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using ‹*nat-of-lit* (*C* ! *Suc 0*) < *length W′*› **by** *simp*
subgoal using ‹*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S′*))) < *length W′*›
by (*simp add*: *S′ lit-of-hd-trail-def*)

subgoal using *le-C-ge* **.**
subgoal by (*auto simp*: *append-and-length-fast-code-pre-def isasat-fast-def uint64-max-def uint32-max-def*)
subgoal
using *D′ C-1-neq-hd vmtf avdom M1′-M1 size-learned-clss-dom-m*[*of N*] *valid-arena-size-dom-m-le-arena*[*OF valid*]
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def S′ U′ lit-of-hd-trail-def all-atms-def*[*symmetric*] *isasat-fast-def uint64-max-def uint32-max-def*)
subgoal for *x uu x1 x2 vm uua- glue uub D′′ xa x′ x1a x2a*
by (*auto simp*: *update-lbd-pre-def arena-is-valid-clause-idx-def*)
subgoal using *length-watched-le*[*of S′ S* ‹−*lit-of-hd-trail M*›] *corr SS′* $uM\text{-}\mathcal{L}_{all}$ *W′-eq S-arena*
by (*auto simp*: *isasat-fast-def length-ll-def S′ U lit-of-hd-trail-def simp flip*: *all-atms-def*)
subgoal using *length-watched-le*[*of S′ S* ‹*C* ! *Suc 0*›] *corr SS′ W′-eq S-arena C-1-neq-hd C-Suc1-in*
by (*auto simp*: *length-ll-def S′ U lit-of-hd-trail-def isasat-fast-def simp flip*: *all-atms-def*)
subgoal
using *M1′-M1* **by** (*rule isa-length-trail-pre*)
subgoal using *D′ C-1-neq-hd vmtf avdom*
by (*auto simp*: *DECISION-REASON-def*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
subgoal
using *M1′-M1*
by (*rule cons-trail-Propagated-tr-pre*)
(*use undef* $uM\text{-}\mathcal{L}_{all}$ **in** ‹*auto simp*: *lit-of-hd-trail-def S′ U′ all-atms-def*[*symmetric*]›)
subgoal using *undef* **by** (*auto simp*: *S′*)
subgoal using $uM\text{-}\mathcal{L}_{all}$ **by** (*auto simp*: *S′ U′* $uminus\text{-}\mathcal{A}_{in}\text{-}iff$)
subgoal
using *D′ C-1-neq-hd vmtf avdom*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def*
intro!: *ASSERT-refine-left ASSERT-leI RES-refine exI*[*of - C*] *valid-arena-update-lbd*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
subgoal
using *D′ C-1-neq-hd vmtf avdom M1′-M1*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def S′ U′ lit-of-hd-trail-def all-atms-def*[*symmetric*])
subgoal by *auto*
subgoal
using *D′ C-1-neq-hd vmtf avdom M1′-M1*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def S′ U′ lit-of-hd-trail-def all-atms-def*[*symmetric*])
subgoal
using *D′ C-1-neq-hd vmtf avdom M1′-M1*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def S′ U′ lit-of-hd-trail-def all-atms-def*[*symmetric*])
subgoal
using *D′ C-1-neq-hd vmtf avdom M1′-M1*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def S′ U′ lit-of-hd-trail-def all-atms-def*[*symmetric*])
subgoal
supply *All-atms-rew*[*simp*]
unfolding *twl-st-heur-def*
using *D′ C-1-neq-hd vmtf avdom M1′-M1 bounded nempty r arena-le*
apply (*auto 0 0 simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def*
Let-def T′ U′ U rescore-clause-def S′ map-fun-rel-def

list-of-mset2-def vmtf-flush-def RES-RES2-RETURN-RES RES-RETURN-RES φ *uminus-*$\mathcal{A}_{in}$*-iff*
get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 lit-of-hd-trail-def
RES-RES-RETURN-RES lbd-empty-def get-LBD-def DECISION-REASON-def
all-atms-def[*symmetric*] *cons-trail-Propagated-def*
intro!: *ASSERT-refine-left ASSERT-leI RES-refine exI*[*of - C*] *valid-arena-update-lbd*
dest: *valid-arena-one-notin-vdomD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
apply (*auto simp*: *vdom-m-simps5*)
done — *?i* $\notin\#$ *dom-m ?N* $\Longrightarrow$ *vdom-m ?*$\mathcal{A}$ *?W* (*fmupd ?i ?C ?N*) = *insert ?i* (*vdom-m ?*$\mathcal{A}$ *?W ?N*) must apply after the other simp rules.
done
qed

have *propagate-unit-bt-wl-D-int*: ‹*propagate-unit-bt-wl-D-int*
(*lit-of-hd-trail-st-heur S*) *U*
≤ ⇓ *?S*
(*propagate-unit-bt-wl-D*
(*lit-of* (*hd* (*get-trail-wl S'*))) *U'*)›
if
SS': ‹(*S*, *S'*) ∈ *?R*› **and**
‹*backtrack-wl-D-inv S'*› **and**
‹*backtrack-wl-D-heur-inv S*› **and**
‹(*TnC*, *T'*) ∈ *?shorter S' S*› **and**
[*simp*]: ‹*nC* = (*n*, *C*)› **and**
[*simp*]: ‹*TnC* = (*T*, *nC*)› **and**
find-decomp: ‹(*U*, *U'*) ∈ *?find-decomp S T' n*› **and**
‹¬ *1* < *length C*› **and**
‹¬ *1* < *size* (*the* (*get-conflict-wl U'*))›
for *S S' TnC T' T nC n C U U'*
proof −
have
TT': ‹(*T*, *del-conflict-wl T'*) ∈ *twl-st-heur-bt*› **and**
n: ‹*n* = *get-maximum-level* (*get-trail-wl T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T'*))) (*mset C*))› **and**
T-C: ‹*get-conflict-wl T'* = *Some* (*mset C*)› **and**
T'S': ‹*equality-except-conflict-wl T' S'*› **and**
‹*C* ≠ []› **and**
hd-C: ‹*hd C* = − *lit-of* (*hd* (*get-trail-wl T'*))› **and**
incl: ‹*mset C* ⊆# *the* (*get-conflict-wl S'*)› **and**
dist-S': ‹*distinct-mset* (*the* (*get-conflict-wl S'*))› **and**
list-confl-S': ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*the* (*get-conflict-wl S'*))› **and**
‹*get-conflict-wl S'* ≠ *None*› **and**
‹*C* ≠ []› **and**
uL-M: ‹− *lit-of* (*hd* (*get-trail-wl S'*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st S'*)› **and**
tr-nempty: ‹*get-trail-wl T'* ≠ []›
using ‹(*TnC*, *T'*) ∈ *?shorter S' S*› ‹~*1* < *length C*›
by (*auto*)
obtain *K M2* **where**
UU': ‹(*U*, *U'*) ∈ *twl-st-heur-bt*› **and**
U'U': ‹*equality-except-trail-wl U' T'*› **and**
lev-K: ‹*get-level* (*get-trail-wl T'*) *K* = *Suc* (*get-maximum-level* (*get-trail-wl T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T'*)))
(*the* (*get-conflict-wl T'*))))› **and**
decomp: ‹(*Decided K* # *get-trail-wl U'*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T'*))›
and
r: ‹*length* (*get-clauses-wl-heur S*) = *r*›

using *find-decomp SS'*
by (*auto*)

obtain *M N NE UE Q W* **where**
T': ‹*T' = (M, N, Some (mset C), NE, UE, Q, W)*›
using *TT' T-C* ‹¬*1 < length C*›
apply (*cases T'*; *cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def*)
obtain *D'* **where**
S': ‹*S' = (M, N, D', NE, UE, Q, W)*›
using *T'S'*
apply (*cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T' del-conflict-wl-def*)

obtain *M1* **where**
U': ‹*U' = (M1, N, Some (mset C), NE, UE, Q, W)*›
using ‹(*TnC*, *T'*) ∈ *?shorter S' S*› *find-decomp*
apply (*cases U'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T'*)
obtain *vm' W' φ clvls cach lbd outl stats fema sema ccount vdom avdom lcount arena D' Q' opts M1'*
where
U: ‹*U = (M1', arena, D', Q', W', vm', φ, clvls, cach, lbd, outl, stats, fema, sema, ccount, vdom, avdom, lcount, opts, [])*› **and**
avdom: ‹*mset avdom ⊆# mset vdom*› **and**
r': ‹*length (get-clauses-wl-heur U) = r*›
using *UU' find-decomp r* **by** (*cases U*) (*auto simp*: *U' T' twl-st-heur-bt-def*)
have
M'M: ‹(*M1'*, *M1*) ∈ *trail-pol* (*all-atms-st U'*)› **and**
W'W: ‹(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st U'*))› **and**
vmtf: ‹*vm'* ∈ *isa-vmtf* (*all-atms-st U'*) *M1*› **and**
φ: ‹*phase-saving* (*all-atms-st U'*) *φ*› **and**
n-d-M1: ‹*no-dup M1*› **and**
empty-cach: ‹*cach-refinement-empty* (*all-atms-st U'*) *cach*› **and**
‹*length outl = Suc 0*› **and**
outl: ‹*out-learned M1 None outl*› **and**
lcount: ‹*lcount = size* (*learned-clss-l N*)› **and**
vdom: ‹*vdom-m* (*all-atms-st U'*) *W N* ⊆ *set vdom*› **and**
valid: ‹*valid-arena arena N* (*set vdom*)› **and**
D': ‹(*D'*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st U'*)› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st U'*)› **and**
nempty: ‹*isasat-input-nempty* (*all-atms-st U'*)› **and**
dist-vdom: ‹*distinct vdom*›
using *UU'* **by** (*auto simp*: *out-learned-def twl-st-heur-bt-def U U' all-atms-def*[*symmetric*])
have [*simp*]: ‹*C* ! *0* = − *lit-of* (*hd M*)› **and**
n-d: ‹*no-dup M*›
using *SS' hd-C* ‹*C* ≠ []› **by** (*auto simp*: *S' U' T' twl-st-heur-conflict-ana-def hd-conv-nth*)
have *undef*: ‹*undefined-lit M1* (*lit-of* (*hd M*))›
using *decomp n-d*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend simp*: *T' hd-append U' neq-Nil-conv split*: *if-splits*)
have *C*: ‹*C* = [− *lit-of* (*hd M*)]›
using ‹*C* ≠ []› ‹*C* ! *0* = − *lit-of* (*hd M*)› ‹¬*1 < length C*›
by (*cases C*) (*auto simp del*: ‹*C* ! *0* = − *lit-of* (*hd M*)›)
have *propagate-unit-bt-wl-D-alt-def*:
‹*propagate-unit-bt-wl-D* = (*λL* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *do* {

- ← *RETURN* ();
- ← *RETURN* ();
- ← *RETURN* ();
- ← *RETURN* ();
M ← *RETURN* (*cons-trail-Propagated* ($-L$) *0* M);
    D' ← *single-of-mset* (*the* D);
    RETURN (M, N, *None*, *NE*, *add-mset* {#D'#} *UE*, {#L#}, W)
  })›
  unfolding *propagate-unit-bt-wl-D-def Let-def cons-trail-Propagated-def* **by** *auto*
 have [*refine0*]:
  ‹*lbd-empty lbd* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* = *replicate* (*length lbd*) *False*})›
  by (*auto simp*: *lbd-empty-def*)
 have [*refine0*]: ‹(*isa-length-trail M1'*, ()) ∈ {(*j*, -). *j* = *length M1*}›
  by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*, *OF* - *M'M*]) *auto*

 have [*refine0*]: ‹*isa-vmtf-flush-int M1' vm'* ≤
    SPEC(λ*c*. (*c*, ()) ∈ {(*vm'*, -). *vm'* ∈ *isa-vmtf* (*all-atms-st U'*) *M1*})›
  for *vm i L*
 proof −
  obtain *vm0* **where**
   vm: ‹(*vm'*, *vm0*) ∈ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st U'*)› **and**
   vm0: ‹*vm0* ∈ *vmtf* (*all-atms-st U'*) *M1*›
   using *vmtf* **unfolding** *isa-vmtf-def* **by** (*cases vm'*) *auto*
  show *?thesis*
   apply (*rule order.trans*)
   apply (*rule isa-vmtf-flush-int*[*THEN fref-to-Down-curry*, *of* - - *M1 vm'*])
   apply ((*solves* ‹*use M'M in auto*›)+)[*2*]
   apply (*subst Down-id-eq*)
   apply (*rule order.trans*)
   apply (*rule vmtf-change-to-remove-order'*[*THEN fref-to-Down-curry*, *of* ‹*all-atms-st U'*› *M1 vm0 M1 vm'*])
   subgoal using *vm0 bounded nempty* **by** *auto*
   subgoal using *vm* **by** *auto*
   subgoal by (*auto simp*: *vmtf-flush-def conc-fun-RES RETURN-def intro*: *isa-vmtfI*)
   done
 qed
 have [*refine0*]: ‹*get-LBD lbd* ≤ *SPEC*(λ*c*. (*c*, ()) ∈ *UNIV*)›
  by (*auto simp*: *get-LBD-def*)

 have *tr-S*: ‹(*get-trail-wl-heur S*, *M*) ∈ *trail-pol* (*all-atms-st S'*)›
  using *SS'* **by** (*auto simp*: *S' twl-st-heur-conflict-ana-def all-atms-def*)

 have *hd-SM*: ‹*lit-of-last-trail-pol* (*get-trail-wl-heur S*) = *lit-of* (*hd M*)›
  unfolding *lit-of-hd-trail-def lit-of-hd-trail-st-heur-def*
  by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
   (*use M'M tr-S tr-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def T' S'*›)
 have [*of* - - ‹*all-atms-st U'*›, *refine0*]: ‹*undefined-lit M L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *C* ≠ *DECISION-REASON* ⟹
   (((*L'*, *C'*), *M'*), (*L*, *C*), *M*) ∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *trail-pol* $\mathcal{A}$ ⟹
   RETURN (*cons-trail-Propagated-tr L' C' M'*)
    ≤ ⇓ {(*M0*, *M0''*). (*M0*, *M0''*) ∈ *trail-pol* $\mathcal{A}$ ∧ *M0''* = *Propagated L C'* # *M*}
  (*RETURN* (*cons-trail-Propagated L C M*))› **for** *C C'* :: *nat* **and** *L* **and** *L'* **and** *M M'* $\mathcal{A}$
  using *cons-trail-Propagated-tr*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*, *of L C M L' C' M'*]
  by (*auto simp*: *cons-trail-Propagated-def*)
 have *uL-M*: ‹− *lit-of* (*hd* (*get-trail-wl S'*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st U'*)›
  using *uL-M* **by** (*simp add*: *S' U'*)

let *?NE* = ‹*add-mset* {#− *lit-of* (*hd M*)#} (*NE* + *UE*)›
have *All-atms-rew*: ‹*set-mset* (*all-atms* (*N*) (*?NE*)) =
set-mset (*all-atms N* (*NE* + *UE*))› (**is** *?A*)
‹*trail-pol* (*all-atms* (*N*) (*?NE*)) =
trail-pol (*all-atms N* (*NE* + *UE*))› (**is** *?B*)
‹*isa-vmtf* (*all-atms* (*N*) (*?NE*)) =
isa-vmtf (*all-atms N* (*NE* + *UE*))› (**is** *?C*)
‹*option-lookup-clause-rel* (*all-atms* (*N*) (*?NE*)) =
option-lookup-clause-rel (*all-atms N* (*NE* + *UE*))› (**is** *?D*)
‹⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms* (*N*) (*?NE*))) =
⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE*)))› (**is** *?E*)
‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*N*) (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))›
‹*phase-saving* ((*all-atms* (*N*) (*?NE*))) =
phase-saving ((*all-atms N* (*NE* + *UE*)))› (**is** *?F*)
‹*cach-refinement-empty* ((*all-atms* (*N*) (*?NE*))) =
cach-refinement-empty ((*all-atms N* (*NE* + *UE*)))› (**is** *?G*)
‹*vdom-m* ((*all-atms* (*N*) (*?NE*))) =
vdom-m ((*all-atms N* (*NE* + *UE*)))› (**is** *?H*)
‹*isasat-input-bounded* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-bounded ((*all-atms N* (*NE* + *UE*)))› (**is** *?I*)
‹*isasat-input-nempty* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-nempty ((*all-atms N* (*NE* + *UE*)))› (**is** *?J*)
for b x' C'
proof −
show *A*: *?A*
using *uL-M*
apply (*cases* ‹*hd M*›)
by (*auto simp*: *all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset* U' S' *in*-$\mathcal{L}_{all}$-*atm-of*-$\mathcal{A}_{in}$ *literals-are-in*-$\mathcal{L}_{in}$-*def atm-of-eq-atm-of all-lits-of-m-add-mset*)
have *A2*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms N* (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))›
using *A* **unfolding** $\mathcal{L}_{all}$-*def C* **by** (*auto simp*: *A*)
then show ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*N*) (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE*)))›
using *A* **unfolding** $\mathcal{L}_{all}$-*def C* **by** (*auto simp*: *A*)
have *A3*: ‹*set-mset* (*all-atms N* (*?NE*)) =
set-mset (*all-atms N* (*NE* + *UE*))›
using *A* **unfolding** $\mathcal{L}_{all}$-*def C* **by** (*auto simp*: *A*)

show *?B* **and** *?C* **and** *?D* **and** *?E* **and** *?F* **and** *?G* **and** *?H* **and** *?I* **and** *?J*
unfolding *trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def isa-vmtf-def vmtf-def distinct-atoms-rel-def vmtf*-$\mathcal{L}_{all}$-*def atms-of-def*
distinct-hash-atoms-rel-def
atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
cach-refinement-def
cach-refinement-list-def vdom-m-def
isasat-input-bounded-def
isasat-input-nempty-def cach-refinement-nonull-def
unfolding *trail-pol-def*[*symmetric*] *ann-lits-split-reasons-def*[*symmetric*]
isasat-input-bounded-def[*symmetric*]
vmtf-def[*symmetric*]
isa-vmtf-def[*symmetric*]
distinct-atoms-rel-def[*symmetric*]

vmtf-$\mathcal{L}_{all}$-def[*symmetric*] *atms-of-def*[*symmetric*]
distinct-hash-atoms-rel-def[*symmetric*]
atoms-hash-rel-def[*symmetric*]
option-lookup-clause-rel-def[*symmetric*]
lookup-clause-rel-def[*symmetric*]
phase-saving-def[*symmetric*] *cach-refinement-empty-def*[*symmetric*]
cach-refinement-def[*symmetric*]
cach-refinement-list-def[*symmetric*]
vdom-m-def[*symmetric*]
isasat-input-bounded-def[*symmetric*] *cach-refinement-nonull-def*[*symmetric*]
isasat-input-nempty-def[*symmetric*]
apply *auto*
done
qed

have *hd-tr-S-M*: ‹*lit-of-hd-trail-st-heur S* = *lit-of-hd-trail M*›
unfolding *lit-of-hd-trail-def lit-of-hd-trail-st-heur-def*
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *OF - tr-S*])
(*use tr-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def T'*›)

show *?thesis*
using *empty-cach n-d-M1 W'W outl vmtf C* φ *undef uL-M vdom lcount valid D' avdom*
unfolding *U U' propagate-unit-bt-wl-D-int-def prod.simps hd-SM*
propagate-unit-bt-wl-D-alt-def
apply (*rewrite at* ‹*let* - = *incr-uset* - *in* -› *Let-def*)
apply (*refine-rcg*)
subgoal using *M'M* **by** (*rule isa-length-trail-pre*)
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal
using *M'M* **by** (*rule cons-trail-Propagated-tr-pre*)
(*use undef uL-M* **in** ‹*auto simp*: *hd-SM all-atms-def*[*symmetric*] *hd-tr-S-M*
lit-of-hd-trail-def S'›)
subgoal
by (*auto simp*: *U U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$-iff RES-RES-RETURN-RES
DECISION-REASON-def hd-SM
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
by (*auto simp*: *U U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$-iff RES-RES-RETURN-RES
DECISION-REASON-def hd-SM
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
using *M'M*
by (*auto simp*: *U U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$-iff RES-RES-RETURN-RES
DECISION-REASON-def hd-SM
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*

simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
using *bounded nempty dist-vdom r′*
by (*auto simp*: *U U′ lit-of-hd-trail-st-heur-def RETURN-def single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def RES-RES2-RETURN-RES RES-RETURN-RES S′ uminus-*$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES DECISION-REASON-def hd-SM All-atms-rew all-atms-def*[*symmetric*]
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *isa-vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
done
qed

have *trail-nempty*: ‹*fst* (*get-trail-wl-heur S*) ≠ []›
if
‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-D-inv S′*›
for *S S′*
proof −
show *?thesis*
using *that* **unfolding** *backtrack-wl-D-inv-def backtrack-wl-D-heur-inv-def backtrack-wl-inv-def backtrack-l-inv-def* **apply** −
by *normalize-goal*+
(*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-def ann-lits-split-reasons-def*)
qed

show *?thesis*
supply [[*goals-limit=1*]]
apply (*intro frefI nres-relI*)
unfolding *backtrack-wl-D-nlit-heur-alt-def backtrack-wl-D-def*
apply (*refine-rcg shorter*)
subgoal by (*rule inv*)
subgoal by (*rule trail-nempty*)
subgoal for *x y xa S x1 x2 x1a x2a*
by (*auto simp*: *twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl*)
apply (*rule find-decomp-wl-nlit*; *solves assumption*)
subgoal by (*auto simp*: *twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl equality-except-trail-wl-get-clauses-wl*)
subgoal for *x y xa S x1 x2 x1a x2a Sa Sb*
by (*cases Sb*; *cases S*) (*auto simp*: *twl-st-heur-state-simp*)
apply (*rule fst-find-lit-of-max-level-wl*; *solves assumption*)
apply (*rule propagate-bt-wl-D-heur*; *assumption*)
apply (*rule propagate-unit-bt-wl-D-int*; *assumption*)
done
qed

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*›
by (*auto simp*: *uint-max-def uint64-max-def*)

lemma *propagate-bt-wl-D-heur-alt-def*:
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *φ0*, *y*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*, *res-info*, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*length vdom* ≤ *length N0*);

$ASSERT(length\ avdom \leq length\ N0)$;
$ASSERT(nat\text{-}of\text{-}lit\ (C!1) < length\ W0 \wedge nat\text{-}of\text{-}lit\ (-L) < length\ W0)$;
$ASSERT(length\ C > 1)$;
$let\ L' = C!1$;
$ASSERT(length\ C \leq uint32\text{-}max\ div\ 2 + 1)$;
$(vm, \varphi) \leftarrow isa\text{-}vmtf\text{-}rescore\ C\ M\ vm0\ \varphi 0$;
$glue \leftarrow get\text{-}LBD\ lbd$;
$let\ b = False$;
$let\ b' = (length\ C = 2)$;
$ASSERT(isasat\text{-}fast\ (M, N0, D, Q, W0, vm0, \varphi 0, y, cach, lbd, outl, stats, fema, sema,$
$res\text{-}info, vdom, avdom, lcount, opts) \longrightarrow append\text{-}and\text{-}length\text{-}fast\text{-}code\text{-}pre\ ((b, C), N0))$;
$ASSERT(isasat\text{-}fast\ (M, N0, D, Q, W0, vm0, \varphi 0, y, cach, lbd, outl, stats, fema, sema,$
$res\text{-}info, vdom, avdom, lcount, opts) \longrightarrow lcount < uint64\text{-}max)$;
$(N, i) \leftarrow fm\text{-}add\text{-}new\text{-}fast\ b\ C\ N0$;
$ASSERT(update\text{-}lbd\text{-}pre\ ((i, glue), N))$;
$let\ N = update\text{-}lbd\ i\ glue\ N$;
$ASSERT(isasat\text{-}fast\ (M, N0, D, Q, W0, vm0, \varphi 0, y, cach, lbd, outl, stats, fema, sema,$
$res\text{-}info, vdom, avdom, lcount, opts) \longrightarrow length\text{-}ll\ W0\ (nat\text{-}of\text{-}lit\ (-L)) < uint64\text{-}max)$;
$let\ W = W0[nat\text{-}of\text{-}lit\ (-\ L) := W0\ !\ nat\text{-}of\text{-}lit\ (-\ L)\ @\ [to\text{-}watcher\text{-}fast\ (i)\ L'\ b']]$;
$ASSERT(isasat\text{-}fast\ (M, N0, D, Q, W0, vm0, \varphi 0, y, cach, lbd, outl, stats, fema, sema,$
$res\text{-}info, vdom, avdom, lcount, opts) \longrightarrow length\text{-}ll\ W\ (nat\text{-}of\text{-}lit\ L') < uint64\text{-}max)$;
$let\ W = W[nat\text{-}of\text{-}lit\ L' := W!nat\text{-}of\text{-}lit\ L'\ @\ [to\text{-}watcher\text{-}fast\ (i)\ (-L)\ b']]$;
$lbd \leftarrow lbd\text{-}empty\ lbd$;
$ASSERT(isa\text{-}length\text{-}trail\text{-}pre\ M)$;
$let\ j = isa\text{-}length\text{-}trail\ M$;
$ASSERT(i \neq DECISION\text{-}REASON)$;
$ASSERT(cons\text{-}trail\text{-}Propagated\text{-}tr\text{-}pre\ ((-L, i), M))$;
$let\ M = cons\text{-}trail\text{-}Propagated\text{-}tr\ (-\ L)\ i\ M$;
$vm \leftarrow isa\text{-}vmtf\text{-}flush\text{-}int\ M\ vm$;
$ASSERT(atm\text{-}of\ L < length\ \varphi)$;
$RETURN\ (M, N, D, j, W, vm, save\text{-}phase\ (-L)\ \varphi, zero\text{-}uint32\text{-}nat,$
$cach, lbd, outl, add\text{-}lbd\ (uint64\text{-}of\text{-}nat\ glue)\ stats, ema\text{-}update\ glue\ fema, ema\text{-}update\ glue\ sema,$
$incr\text{-}conflict\text{-}count\text{-}since\text{-}last\text{-}restart\ res\text{-}info, vdom\ @\ [nat\text{-}of\text{-}uint64\text{-}conv\ i],$
$avdom\ @\ [nat\text{-}of\text{-}uint64\text{-}conv\ i],$
$lcount + 1, opts)$
})›
unfolding *propagate-bt-wl-D-heur-def uint64-of-nat-conv-def* **by** *auto*

lemma *propagate-bt-wl-D-fast-code-isasat-fastI2*: ‹$isasat\text{-}fast\ b \Longrightarrow$
$b = (a1', a2') \Longrightarrow$
$a2' = (a1'a, a2'a) \Longrightarrow$
$a < length\ a1'a \Longrightarrow a \leq uint64\text{-}max$›
by (*cases b*) (*auto simp*: *isasat-fast-def*)

lemma *propagate-bt-wl-D-fast-code-isasat-fastI3*: ‹$isasat\text{-}fast\ b \Longrightarrow$
$b = (a1', a2') \Longrightarrow$
$a2' = (a1'a, a2'a) \Longrightarrow$
$a \leq length\ a1'a \Longrightarrow a < uint64\text{-}max$›
by (*cases b*) (*auto simp*: *isasat-fast-def uint64-max-def uint32-max-def*)

lemma *lit-of-hd-trail-st-heur-alt-def*:
‹$lit\text{-}of\text{-}hd\text{-}trail\text{-}st\text{-}heur = (\lambda(M, N, D, Q, W, vm, \varphi).\ lit\text{-}of\text{-}last\text{-}trail\text{-}pol\ M)$›
by (*auto simp*: *lit-of-hd-trail-st-heur-def lit-of-hd-trail-def intro*!: *ext*)

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))
  }›
  unfolding isa-empty-conflict-and-extract-clause-heur-def WB-More-Refinement-List.swap-def IICF-List.swap-def[symm
  by auto

sepref-definition empty-conflict-and-extract-clause-heur-code
  is ‹uncurry2 (isa-empty-conflict-and-extract-clause-heur)›
  :: ‹[λ((M, D), outl). outl ≠ [] ∧ length outl ≤ uint-max]_a
      trail-pol-assn^k *_a lookup-clause-rel-assn^d *_a out-learned-assn^k →
      (bool-assn *a uint32-nat-assn *a array-assn option-bool-assn) *a clause-ll-assn *a uint32-nat-assn›
  supply [[goals-limit=1]] image-image[simp]
  unfolding isa-empty-conflict-and-extract-clause-heur-alt-def
    array-fold-custom-replicate length-uint32-nat-def zero-uint32-nat-def[symmetric]
    one-uint32-nat-def[symmetric]
  by sepref

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)›
  :: ‹[λ((M, D), outl). outl ≠ [] ∧ length outl ≤ uint-max]_a
      trail-pol-fast-assn^k *_a lookup-clause-rel-assn^d *_a out-learned-assn^k →
      (bool-assn *a uint32-nat-assn *a array-assn option-bool-assn) *a clause-ll-assn *a uint32-nat-assn›
  supply [[goals-limit=1]] image-image[simp]
  unfolding isa-empty-conflict-and-extract-clause-heur-alt-def
    array-fold-custom-replicate length-uint32-nat-def zero-uint32-nat-def[symmetric]
    one-uint32-nat-def[symmetric]
  by sepref
```

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*›
  supply *array-replicate-hnr*[*sepref-fr-rules*] *uint-max-def*[*simp*]
  unfolding *empty-cach-ref-set-def comp-def zero-uint32-nat-def*[*symmetric*]
    one-uint32-nat-def[*symmetric*]
  by *sepref*

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 −
  have *H*: ‹*?c*
    ∈[*comp-PRE Id*
     (λ(*cach*, *supp*).
         (∀ *L*∈*set supp*. *L* < *length cach*) ∧
         length supp ≤ *Suc* (*uint-max div 2*) ∧
         (∀ *L*<*length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set supp*))
     (λ*x y*. *True*)
     (λ*x*. *nofail* ((*RETURN* ∘ *empty-cach-ref*) *x*))]$_a$
      hrp-comp (*cach-refinement-l-assn*^{d})
                    Id → *hr-comp cach-refinement-l-assn Id*›
    (**is** ‹- ∈ [*?pre'*]$_a$ *?im'* → *?f'*›)
    using *hfref-compI-PRE*[*OF empty-cach-code.refine*[*unfolded PR-CONST-def convert-fref*]
        empty-cach-ref-set-empty-cach-ref[*unfolded convert-fref*]] **by** *simp*
  have *pre*: ‹*?pre' h x*› **if** ‹*?pre x*› **for** *x h*
    using *that* **by** (*auto simp*: *comp-PRE-def trail-pol-def*
        ann-lits-split-reasons-def empty-cach-ref-pre-def)
  have *im*: ‹*?im'* = *?im*›
    by *simp*
  have *f*: ‹*?f'* = *?f*›
    by *auto*
  show *?thesis*
    apply (*rule hfref-weaken-pre*[*OF*])
     defer
    using *H* **unfolding** *im f* **apply** *assumption*
    using *pre* **..**
qed

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*›
  by *sepref-to-hoare*
   (*sep-auto simp*: *uint32-nat-rel-def br-def uint64-of-uint32-def*)

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*›
  supply [[*goals-limit* = *1*]] *uminus-*$\mathcal{A}_{in}$*-iff*[*simp*] *image-image*[*simp*] *append-ll-def*[*simp*]
    rescore-clause-def[*simp*] *vmtf-flush-def*[*simp*] *le-uint32-max-div-2-le-uint32-max*[*simp*]
  unfolding *propagate-bt-wl-D-heur-def isasat-unbounded-assn-def cons-trail-Propagated-def*[*symmetric*]

unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*] *append-ll-def*[*symmetric*] *nat-of-uint32-conv-def*
cons-trail-Propagated-def[*symmetric*] *PR-CONST-def save-phase-def*
unfolding *length-uint32-nat-def*[*symmetric*] *two-uint32-nat-def*[*symmetric*]
by *sepref* — slow

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*›
supply [[*goals-limit* = *1*]] *append-ll-def*[*simp*] *isasat-fast-length-leD*[*dest*]
propagate-bt-wl-D-fast-code-isasat-fastI2[*intro*] *length-ll-def*[*simp*]
propagate-bt-wl-D-fast-code-isasat-fastI3[*intro*]
unfolding *propagate-bt-wl-D-heur-alt-def*
isasat-bounded-assn-def cons-trail-Propagated-def[*symmetric*]
to-watcher-fast-def[*symmetric*] *nat-of-uint64-conv-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*] *append-ll-def*[*symmetric*]
cons-trail-Propagated-def[*symmetric*] *PR-CONST-def save-phase-def two-uint32-nat-def*[*symmetric*]
apply (*rewrite at* ‹*let* - = - ! ⌑ *in* -› *one-uint32-nat-def*[*symmetric*])
apply (*rewrite at* ‹(- + ⌑, -)› *one-uint64-nat-def*[*symmetric*])
apply (*rewrite at* ‹*let* - = (⌑ = *two-uint32-nat*) *in* -› *length-uint32-nat-def*[*symmetric*])
by *sepref* — This call is now unreasonnably slow.

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*›
supply [[*goals-limit* = *1*]] *vmtf-flush-def*[*simp*] *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*simp*]
unfolding *propagate-unit-bt-wl-D-int-def cons-trail-Propagated-def*[*symmetric*] *isasat-unbounded-assn-def*
PR-CONST-def length-uint32-nat-def[*symmetric*]
by *sepref*

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*›
supply [[*goals-limit* = *1*]] *vmtf-flush-def*[*simp*] *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*simp*]
unfolding *propagate-unit-bt-wl-D-int-def cons-trail-Propagated-def*[*symmetric*] *isasat-bounded-assn-def*
PR-CONST-def length-uint32-nat-def[*symmetric*] *zero-uint64-nat-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ clause\text{-}ll\text{-}assn$⟩
supply [[*goals-limit=1*]] *empty-conflict-and-extract-clause-pre-def*[*simp*]
unfolding *extract-shorter-conflict-list-heur-st-def PR-CONST-def isasat-unbounded-assn-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
one-uint32-nat-def[*symmetric*] *zero-uint32-nat-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn *a\ uint32\text{-}nat\text{-}assn *a\ clause\text{-}ll\text{-}assn$⟩
supply [[*goals-limit=1*]] *empty-conflict-and-extract-clause-pre-def*[*simp*]
unfolding *extract-shorter-conflict-list-heur-st-def PR-CONST-def isasat-bounded-assn-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
one-uint32-nat-def[*symmetric*] *zero-uint32-nat-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$[\lambda S.\ fst\ (get\text{-}trail\text{-}wl\text{-}heur\ S) \neq []]_a\ isasat\text{-}unbounded\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$⟩
unfolding *lit-of-hd-trail-st-heur-alt-def isasat-unbounded-assn-def*
by *sepref*

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*⟩
:: ⟨$[\lambda S.\ fst\ (get\text{-}trail\text{-}wl\text{-}heur\ S) \neq []]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$⟩
unfolding *lit-of-hd-trail-st-heur-alt-def isasat-bounded-assn-def*
by *sepref*

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*⟩
:: ⟨$[isasat\text{-}fast]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
size-conflict-wl-def[*simp*] *isasat-fast-length-leD*[*intro*] *isasat-fast-def*[*simp*]
unfolding *backtrack-wl-D-nlit-heur-def PR-CONST-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*]
cons-trail-Propagated-def[*symmetric*]

size-conflict-wl-def[*symmetric*]
by *sepref*

sepref-definition *backtrack-wl-D-code*
is ⟨*backtrack-wl-D-nlit-heur*⟩
:: ⟨*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]
size-conflict-wl-def[*simp*] *isasat-fast-length-leD*[*intro*]
unfolding *backtrack-wl-D-nlit-heur-def PR-CONST-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*]
cons-trail-Propagated-def[*symmetric*]
size-conflict-wl-def[*symmetric*]
by *sepref*

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* (λ-. *True*) (λ*x s*. *RETURN* (*f x s*)) *s*
apply (*induction l arbitrary*: *s*) **apply** (*auto*) **done**

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-hash-int-empty n* = *RETURN* (*replicate n False*)›

lemma *atoms-hash-int-empty-atoms-hash-empty*:
‹(*atoms-hash-int-empty*, *RETURN o atoms-hash-empty*) ∈
[$\lambda n.$ ($\forall L \in \# \mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *n*)]$_f$ *nat-rel* → ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*use Max-less-iff* **in** ‹*auto simp*: *atoms-hash-rel-def atoms-hash-int-empty-def atoms-hash-empty-def*
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff Ball-def*
dest: *spec*[*of - Pos -*]›)

definition (**in** −) *distinct-atms-empty* **where**
‹*distinct-atms-empty -* = {}›

definition (**in** −) *distinct-atms-int-empty* **where**
‹*distinct-atms-int-empty n* = *RETURN* ([], *replicate n False*)›

lemma *distinct-atms-int-empty-distinct-atms-empty*:
‹(*distinct-atms-int-empty*, *RETURN o distinct-atms-empty*) ∈
[$\lambda n.$ ($\forall L \in \# \mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *n*)]$_f$ *nat-rel* → ⟨*distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*auto simp*: *distinct-atoms-rel-alt-def distinct-atms-empty-def distinct-atms-int-empty-def*)
by (*metis atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *atms-of-def imageE*)

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*›
by (*auto simp*: *isa-vmtf-init-def Image-iff intro*!: *bexI*[*of -* ‹(*vm*, *to-remove′*)›])

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*)›
by (*auto simp*: *isa-vmtf-init-def vmtf-init-def dest*: *vmtf-consD*)

lemma *vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf-init* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf-init* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]

unfolding *vmtf-init-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*

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*›
using *vmtf-init-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
apply (*subst* (*asm*) *isa-vmtf-init-def*)
by (*cases L*) (*auto intro*!: *isa-vmtf-initI*)

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* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena* *N'* *N* (*set vdom*) ∧ *set-mset* (*dom-m* *N*) ⊆ *set vdom*)) ∧
(*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* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m* *N*#} + *NE* + *UE*))
⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
(*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-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*, 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* (*dom-m* *N*) ⊆ *set vdom*)) ∧ (*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ∧
(*D'*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length* *M* ∧

$Q = uminus$ ‘# $lit\text{-}of$ ‘# $mset$ $(drop\ j\ (rev\ M))$ ∧
$vm \in isa\text{-}vmtf\text{-}init\ \mathcal{A}\ M$ ∧
$phase\text{-}saving\ \mathcal{A}\ \varphi$ ∧
$no\text{-}dup\ M$ ∧
$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach$ ∧
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A})$ ∧
$(W', empty\text{-}watched\ \mathcal{A}) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A})$ ∧
$isasat\text{-}input\text{-}bounded\ \mathcal{A}$ ∧
$distinct\ vdom$
}›

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* =
$\{((M', N', D', j, W', vm, \varphi, clvls, cach, lbd, vdom, failed), (M, N, D, NE, UE, Q, W)).$
$(unbdd \longrightarrow \neg failed)$ ∧
$((unbdd \vee \neg failed) \longrightarrow$
$((M', M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE))$ ∧
$set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom$ ∧
$valid\text{-}arena\ N'\ N\ (set\ vdom)))$ ∧
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE))$ ∧
$j \leq length\ M$ ∧
$Q = uminus$ ‘# $lit\text{-}of$ ‘# $mset\ (drop\ j\ (rev\ M))$ ∧
$vm \in isa\text{-}vmtf\text{-}init\ (all\text{-}atms\ N\ (NE + UE))\ M$ ∧
$phase\text{-}saving\ (all\text{-}atms\ N\ (NE + UE))\ \varphi$ ∧
$no\text{-}dup\ M$ ∧
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE))\ cach$ ∧
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE))\ W\ N \subseteq set\ vdom$ ∧
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ (all\text{-}atms\ N\ (NE + UE)))$ ∧
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}atms\ N\ (NE + UE)))$ ∧
$isasat\text{-}input\text{-}bounded\ (all\text{-}atms\ N\ (NE + UE))$ ∧
$distinct\ vdom$
}›

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* ($\lambda N\ n.$ *RES* (*vmtf-init N* []))) $\in$
$[\lambda(N,n).\ (\forall L{\in}\#\ N.\ L < n) \wedge (\textit{distinct-mset}\ N) \wedge \textit{size}\ N < \textit{uint32-max} \wedge \textit{set-mset}\ N = \textit{set-mset}\ \mathcal{A}]_f$
$(\langle \textit{uint32-nat-rel}\rangle \textit{list-rel-mset-rel}) \times_f \textit{nat-rel} \rightarrow$
$\langle((\langle Id\rangle \textit{list-rel} \times_r \textit{nat-rel} \times_r \langle \textit{nat-rel}\rangle\ \textit{option-rel} \times_r \langle \textit{nat-rel}\rangle\ \textit{option-rel} \times_r \langle \textit{nat-rel}\rangle\ \textit{option-rel})$
$\times_r \textit{distinct-atoms-rel}\ \mathcal{A}\rangle \textit{nres-rel}$›
(**is** ‹(*?init*, *?R*) $\in$ -›)
proof −
have *vmtf-ns-notin-empty*: ‹*vmtf-ns-notin* [] *0* (*replicate n* (*VMTF-Node 0 None None*))› **for** *n*
unfolding *vmtf-ns-notin-def*
by *auto*

have *K2*: ‹*distinct N* $\Longrightarrow$ *fst-As* $<$ *length N* $\Longrightarrow$ *N*!*fst-As* $\in$ *set* (*take fst-As N*) $\Longrightarrow$ *False*›
for *fst-As x N*
by (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-take less-not-refl min-less-iff-conj*
nth-eq-iff-index-eq nth-take)
have *W-ref*: ‹$\textit{WHILE}_T$ (λ(*i*, *A*, *cnext*). *i* $<$ *length-uint32-nat N'*)
(λ(*i*, *A*, *cnext*). *do* {
- $\leftarrow$ *ASSERT* (*i* $<$ *length-uint32-nat N'*);
let L = *nat-of-uint32* (*N'* ! *i*);
- $\leftarrow$ *ASSERT* (*L* $<$ *length A*);
- $\leftarrow$ *ASSERT* (*cnext* $\neq$ *None* $\longrightarrow$ *the cnext* $<$ *length A*);
- $\leftarrow$ *ASSERT* (*i* + *1* $\leq$ *uint-max*);
RETURN
(*i* + *one-uint32-nat*,
vmtf-cons A L cnext (*uint64-of-uint32-conv i*), *Some L*)
})
(*zero-uint32-nat*, *replicate n'* (*VMTF-Node zero-uint64-nat None None*),
None)
$\leq$ *SPEC*(λ(*i*, *A'*, *cnext*).
vmtf-ns (*rev* (*map* (*nat-of-uint32*) (*take i N'*))) *i A'*
$\wedge$ *cnext* = *map-option* (*nat-of-uint32*) (*option-last* (*take i N'*)) $\wedge$ *i* = *length N'* $\wedge$
length A' = *n* $\wedge$ *vmtf-ns-notin* (*rev* (*map* (*nat-of-uint32*) (*take i N'*))) *i A'*
)›
(**is** ‹- $\leq$ *SPEC ?P*›)
if *H*: ‹*case y of* (*N*, *n*) $\Rightarrow$($\forall L{\in}\#\ N.\ L < n$) $\wedge$ *distinct-mset N* $\wedge$ *size N* $<$ *uint32-max* $\wedge$
set-mset N = *set-mset* $\mathcal{A}$› **and**
ref: ‹(*x*, *y*) $\in$ $\langle$*uint32-nat-rel*$\rangle$*list-rel-mset-rel* $\times_f$ *nat-rel*› **and**
st[*simp*]: ‹*x* = (*N'*, *n'*)› ‹*y* = (*N*, *n*)›
for *N N' n n' A x y*
proof −
have [*simp*]: ‹*n* = *n'*› **and** *NN'*: ‹(*N'*, *N*) $\in$ $\langle$*uint32-nat-rel*$\rangle$*list-rel-mset-rel*›
using *ref* **unfolding** *st* **by** *auto*
have ‹*inj-on nat-of-uint32 S*› **for** *S*
by (*auto simp*: *inj-on-def*)
then have *dist*: ‹*distinct N'*›
using *NN' H* **by** (*auto simp*: *list-rel-def uint32-nat-rel-def br-def list-mset-rel-def*
list-all2-op-eq-map-right-iff' distinct-image-mset-inj list-rel-mset-rel-def)
have *L-N*: ‹$\forall$ *L*$\in$*set N'*. *nat-of-uint32 L* $<$ *n*›
using *H ref* **by** (*auto simp*: *list-rel-def uint32-nat-rel-def br-def list-mset-rel-def*
list-all2-op-eq-map-right-iff' list-rel-mset-rel-def)
let *?Q* = ‹λ(*i*, *A'*, *cnext*).
vmtf-ns (*rev* (*map* (*nat-of-uint32*) (*take i N'*))) *i A'* $\wedge$ *i* $\leq$ *length N'* $\wedge$

cnext = *map-option* (*nat-of-uint32*) (*option-last* (*take i N′*)) ∧
length A′ = *n* ∧ *vmtf-ns-notin* (*rev* (*map* (*nat-of-uint32*) (*take i N′*))) *i A′*›
show *?thesis*
apply (*refine-vcg WHILET-rule*[**where** *R* = ‹*measure* (λ(*i*, -). *length N′* + *1* − *i*)› **and** *I* = ‹*?Q*›])
subgoal by *auto*
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *S N′ x2 A′*
unfolding *assert-bind-spec-conv vmtf-ns-notin-def*
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2 nat-of-uint32-shiftr option-last-def hd-rev last-map intro*!: *vmtf-cons dest*: *K2*)
subgoal by *auto*
subgoal
using *L-N dist*
by (*auto simp*: *take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2 nat-of-uint32-shiftr option-last-def hd-rev last-map*)
subgoal
using *L-N dist*
by (*auto simp*: *last-take-nth-conv option-last-def*)
subgoal
using *H dist ref*
by (*auto simp*: *last-take-nth-conv option-last-def list-rel-mset-rel-imp-same-length*)
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth option-last-def hd-rev last-map intro*!: *vmtf-cons dest*: *K2*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth*)
subgoal by *auto*
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth hd-rev last-map option-last-def intro*!: *vmtf-notin-vmtf-cons dest*: *K2 split*: *if-splits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed
have [*simp*]: ‹*vmtf*-$\mathcal{L}_{all}$ *n′* [] ((*nat-of-uint32* ‘ *set N*, {}), {})›
if ‹(*N*, *n′*) ∈ ⟨*uint32-nat-rel*⟩*list-rel-mset-rel*› **for** *N N′ n′*
using *that* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def*
by (*auto simp*: $\mathcal{L}_{all}$-*def atms-of-def image-image image-Un list-rel-def uint32-nat-rel-def br-def list-mset-rel-def list-all2-op-eq-map-right-iff′ list-rel-mset-rel-def*)
have *in-N-in-N1*: ‹*L* ∈ *set N′* ⟹ *nat-of-uint32 L* ∈ *atms-of* ($\mathcal{L}_{all}$ *N*)›
if ‹(*N′*, *y*) ∈ ⟨*uint32-nat-rel*⟩*list-rel*› **and** ‹(*y*, *N*) ∈ *list-mset-rel*› **for** *L N N′ y*
using *that* **by** (*auto simp*: $\mathcal{L}_{all}$-*def atms-of-def image-image image-Un list-rel-def uint32-nat-rel-def br-def list-mset-rel-def list-all2-op-eq-map-right-iff′*)

have *length-ba*: ‹∀ *L*∈# *N*. *L* < *length ba* ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ *N*) ⟹

$L < length\ ba$›
if ‹$(N', y) \in \langle uint32\text{-}nat\text{-}rel\rangle list\text{-}rel\text{-}mset\text{-}rel$›
for $L\ ba\ N\ N'\ y$
using *that*
by (*auto simp*: $\mathcal{L}_{all}$*-def nat-shiftr-div2 nat-of-uint32-shiftr*
atms-of-def image-image image-Un split: *if-splits*)
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *initialise-VMTF-def uncurry-def conc-Id id-def vmtf-init-def*
distinct-atms-int-empty-def nres-monad1
apply (*refine-rcg*)
subgoal by (*auto dest*: *list-rel-mset-rel-imp-same-length*)
apply (*rule specify-left*)
apply (*rule W-ref*; *assumption?*)
subgoal for $N'\ N'n'\ n'\ Nn\ N\ n\ st$
apply (*case-tac st*)
apply *clarify*
apply (*subst RETURN-RES-refine-iff*)
apply (*auto dest*: *list-rel-mset-rel-imp-same-length*)
apply (*rule exI*[*of* - ‹{}›])
apply (*auto simp*: *distinct-atoms-rel-alt-def list-rel-mset-rel-def list-mset-rel-def*
br-def; *fail*)
apply (*rule exI*[*of* - ‹{}›])
unfolding *vmtf-def in-pair-collect-simp prod.case*
apply (*intro conjI impI*)
apply (*rule exI*[*of* - ‹*map nat-of-uint32* (*rev* (*fst* N'))›])
apply (*rule-tac exI*[*of* - ‹[]›])
apply (*intro conjI impI*)
subgoal
by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-last-def last-map*
hd-rev list-rel-mset-rel-def br-def list-mset-rel-def)

subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map last-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-rev last-map distinct-atms-empty-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last list-rel-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last dest: *length-ba*)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last list-rel-mset-rel-def dest: *in-N-in-N1*)
subgoal by (*auto simp*: *distinct-atoms-rel-alt-def list-rel-mset-rel-def list-mset-rel-def*
br-def)
done
done
qed

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* = (λ*L* ((*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* = (λ*L* (*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*›
by (*auto simp*: *DECISION-REASON-def*)

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*›
unfolding *twl-st-heur-parsing-no-WL-def propagate-unit-cls-heur-def propagate-unit-init-wl-def*
apply (*intro frefI nres-relI*)
apply (*clarsimp simp add*: *propagate-unit-init-wl.simps cons-trail-Propagated-def*[*symmetric*] *comp-def curry-def all-atms-def*[*symmetric*] *intro*!: *ASSERT-leI*)
apply (*rule ASSERT-leI*)
subgoal for *aa ab ac ad ae b af ag ah ba ai aj ak al am an bb ao bc ap aq ar bd as be*

at au av aw ax ay bg
by (*rule cons-trail-Propagated-tr-pre*[*of - -* $\mathcal{A}$])
(*auto simp*: *DECISION-REASON-def dest*: $\mathcal{L}_{all}$*-cong*)
apply (*rule RETURN-refine*)
apply (*subst in-pair-collect-simp*)
apply (*simp only*: *prod.simps*)
apply (*intro conjI*)
subgoal by *fast*
subgoal by (*auto simp*: *all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-*$\mathcal{A}_{in}$*-iff*)
subgoal by (*auto simp*: *all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-*$\mathcal{A}_{in}$*-iff*)
subgoal
unfolding *DECISION-REASON-def*
by (*auto intro*!: *cons-trail-Propagated-tr*[*of* $\mathcal{A}$, *THEN fref-to-Down-unRET-uncurry2*]
dest: $\mathcal{L}_{all}$*-cong*)
subgoal by *fast*
supply *cons-trail-Propagated-def*[*simp*]
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *isa-vmtf-init-consD*)
subgoal by *fast*
subgoal by *auto*
subgoal by *fast*
subgoal by (*auto simp*: *all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-*$\mathcal{A}_{in}$*-iff*)
subgoal by *auto*
done

definition *already-propagated-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls* = (λ*L* ((*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* = (λ*L* (*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*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-heur-def*
already-propagated-unit-init-wl-def all-lits-of-mm-add-mset all-lits-of-m-add-mset
literals-are-in-$\mathcal{L}_{in}$*-def*)

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*)) ∈
[λ(*L*, *D*). *D* = *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ →

⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*⟩
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def set-conflict-unit-heur-def set-conflict-unit-def*
option-lookup-clause-rel-def lookup-clause-rel-def in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
intro!: *mset-as-position.intros*)

definition *conflict-propagated-unit-cls*
:: ⟨*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *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* ⇒ *twl-st-wl-heur-init* ⇒ *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 ← *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*)) ∈
[λ(*L*, *S*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *get-conflict-init-wl S* = *None*]$_f$
nat-lit-lit-rel $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
nres-rel⟩
proof −
have *set-conflict-init-wl-alt-def*:
⟨*RETURN oo set-conflict-init-wl* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*). *do* {
D ← *RETURN* (*set-conflict-unit L D*);
RETURN ((*M*, *N*, *Some* {#*L*#}, *add-mset* {#*L*#} *NE*, *UE*, {#}), *OC*)
})⟩
by (*auto intro*!: *ext simp*: *set-conflict-init-wl-def*)
have [*refine0*]: ⟨*D* = *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ (*y*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ *L* = *L'*
⟹
set-conflict-unit-heur L' y ≤ ⇓ {(*D*, *D'*). (*D*, *D'*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧ *D'* = *Some* {#*L*#}}
(*RETURN* (*set-conflict-unit L D*))⟩
for *L D y L'*
apply (*rule order-trans*)
apply (*rule set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*,
unfolded comp-def, *of* $\mathcal{A}$ *L D L' y*])
subgoal
by *auto*
subgoal
by *auto*
subgoal
unfolding *conc-fun-RETURN*
by (*auto simp*: *set-conflict-unit-def*)
done

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *set-conflict-init-wl-alt-def conflict-propagated-unit-cls-heur-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)

subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def option-lookup-clause-rel-def*
lookup-clause-rel-def atms-of-def)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def conflict-propagated-unit-cls-def*
image-image set-conflict-unit-def
intro!: *set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*])
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
intro!: *isa-length-trail-pre*)
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
image-image set-conflict-unit-def all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$*-iff*
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*]
intro!: *set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*]
isa-length-trail-pre)
done
qed

definition *add-init-cls-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur unbdd* = (*λC* (*M*, *N*, *D*, *Q*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*). *do* {
let C = *C*;
ASSERT(*length C* ≤ *uint-max* + *2*);
ASSERT(*length C* ≥ *2*);
if unbdd ∨ (*length N* ≤ *uint64-max* − *length C* − *5* ∧ ¬*failed*)
then do {
ASSERT(*length vdom* ≤ *length N*);
(*N*, *i*) ← *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* ⇒ *twl-st-wl-heur-init* ⇒ *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* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b* = *add-init-cls-heur False*›

lemma *length-C-nempty-iff*: ‹*length C* ≥ *2* ⟷ *C* ≠ [] ∧ *tl C* ≠ []›
by (*cases C*; *cases* ‹*tl C*›) *auto*

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) ⇒ *2* ≤ *length C* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*› **and**

xy: ‹(x, y) ∈ *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* ≤ *uint-max* + *2*›
using *pre clss-size-uint-max*[*of* $\mathcal{A}$ ‹*mset x1g*›] *xy st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)

lemma *add-init-pre2*: ‹*2* ≤ *length x1g*›
using *pre xy st* **by** (*auto simp*:)

private lemma
x1g-x1: ‹*x1g* = *x1*› **and**

‹(x1h, x1b) ∈ trail-pol 𝒜› **and**
valid: ‹ valid-arena x1i x1c (set x1r')› **and**
‹(x1j, x1d) ∈ option-lookup-clause-rel 𝒜› **and** ‹x1k ≤ length x1b› **and**
‹x2e = {#− lit-of x. x ∈# mset (drop x1k (rev x1b))#}› **and**
‹x1m ∈ isa-vmtf-init 𝒜 x1b› **and**
‹phase-saving 𝒜 x1n› **and**
‹no-dup x1b› **and**
‹cach-refinement-empty 𝒜 x1p› **and**
vdom: ‹mset x1r' = dom-m x1c› **and**
var-incl:
‹set-mset (all-lits-of-mm ({#mset (fst x). x ∈# ran-m x1c#} + x1e + x1f))
⊆ set-mset ($\mathcal{L}_{all}$ 𝒜)› **and**
watched: ‹(x1l, empty-watched 𝒜) ∈ ⟨Id⟩map-fun-rel (D_0 𝒜)› **and**
bounded: ‹isasat-input-bounded 𝒜›
if ‹¬x2r' ∨ unbdd›
using *that xy* **unfolding** *st twl-st-heur-parsing-no-WL-def*
by *auto*

lemma *init-fm-add-new*:
‹¬x2r' ∨ unbdd ⟹ fm-add-new True x1g x1i
≤ ⇓ {((arena, i), (N', i')). valid-arena arena N' (insert i (set x1r')) ∧ i = i' ∧
i ∉# dom-m x1c ∧ i = length x1i + header-size x1g ∧
i ∉ set x1r'}
(SPEC
(λ(N', ia).
0 < ia ∧ ia ∉# dom-m x1c ∧ N' = fmupd ia (x1, True) x1c))›
(**is** ‹- ⟹ - ≤ ⇓ ?qq -›)
unfolding *x1g-x1*
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *valid vdom pre xy valid-arena-in-vdom-le-arena*[*OF valid*] *arena-lifting*(*2*)[*OF valid*]
valid **unfolding** *st*
by (*fastforce simp*: *x1g-x1 vdom-m-def*
intro!: *RETURN-RES-refine valid-arena-append-clause*)

lemma *add-init-cls-final-rel*:
fixes *xa* :: ‹arena-el list ×
nat› **and** *x'* :: ‹(nat, nat literal list × bool) fmap ×
nat› **and** *x1r* :: ‹(nat,
nat literal list ×
bool) fmap› **and** *x2r* :: ‹nat› **and** *x1s* :: ‹arena-el list› **and** *x2s* :: ‹nat›
assumes
‹(xa, x')
∈ {((arena, i), (N', i')). valid-arena arena N' (insert i (set x1r')) ∧ i = i' ∧
i ∉# dom-m x1c ∧ i = length x1i + header-size x1g ∧
i ∉ set x1r'}› **and**
‹x' ∈ {(N', ia).
0 < ia ∧ ia ∉# dom-m x1c ∧ 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)
∈ twl-st-heur-parsing-no-WL 𝒜 unbdd›
proof −

show *?thesis*
using *assms xy pre* **unfolding** *st*
apply (*auto simp*: *twl-st-heur-parsing-no-WL-def nat-of-uint32-conv-def*
intro!:)
apply (*auto simp*: *vdom-m-simps5 ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*
literals-are-in-$\mathcal{L}_{in}$*-def*)
done
qed
end

lemma *add-init-cls-heur-add-init-cls*:
‹(*uncurry* (*add-init-cls-heur unbdd*), *uncurry* (*add-to-clauses-init-wl*)) ∈
[λ(*C*, *S*). *length C* ≥ *2* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*]$_f$
Id $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
proof −
have ‹*42* + *Max-mset* (*add-mset 0* (*x1c*)) ∉# *x1c*› **and** ‹*42* + *Max-mset* (*add-mset* (*0* :: *nat*) (*x1c*)) ≠ *0*› **for** *x1c*
apply (*cases* ‹*x1c*›) **apply** (*auto simp*: *max-def*)
apply (*metis Max-ge add.commute add.right-neutral add-le-cancel-left finite-set-mset le-zero-eq set-mset-add-mset-insert union-single-eq-member zero-neq-numeral*)
by (*smt Max-ge Set.set-insert add.commute add.right-neutral add-mset-commute antisym diff-add-inverse diff-le-self finite-insert finite-set-mset insert-DiffM insert-commute set-mset-add-mset-insert union-single-eq-member zero-neq-numeral*)
then have [*iff*]: ‹(∀ *b*. *b* = (*0*::*nat*) ∨ *b* ∈# *x1c*) ⟷ *False*› ‹∃ *b*>*0*. *b* ∉# *x1c*›**for** *x1c*
by *blast*+
have *add-to-clauses-init-wl-alt-def*:
‹*add-to-clauses-init-wl* = (λ*i* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*). *do* {
let b = (*length i* = *2*);
(*N'*, *ia*) ← *SPEC* (λ(*N'*, *ia*). *ia* > *0* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*i*, *True*) *N*);
RETURN ((*M*, *N'*, *D*, *NE*, *UE*, *Q*), *OC*)
})›
by (*auto simp*: *add-to-clauses-init-wl-def get-fresh-index-def Let-def*
RES-RES2-RETURN-RES RES-RES-RETURN-RES2 RES-RETURN-RES uncurry-def image-iff
intro!: *ext*)
show *?thesis*
unfolding *add-init-cls-heur-def add-to-clauses-init-wl-alt-def uncurry-def Let-def*
to-watcher-def id-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg init-fm-add-new*)
subgoal
by (*rule add-init-pre1*)
subgoal
by (*rule add-init-pre2*)
apply (*rule lhs-step-If*)
apply (*refine-rcg*)
subgoal unfolding *twl-st-heur-parsing-no-WL-def*
by (*force dest*!: *valid-arena-vdom-le*(*2*) *simp*: *distinct-card*)
apply (*rule init-fm-add-new*)
apply *assumption*+
subgoal by *auto*
subgoal by (*rule add-init-cls-final-rel*)
unfolding *RES-RES2-RETURN-RES RETURN-def*
apply *simp*
subgoal unfolding *RETURN-def* **apply** (*rule RES-refine*)
by (*auto simp*: *twl-st-heur-parsing-no-WL-def RETURN-def intro*!: *RES-refine*)

done
qed

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*)) ∈
[λ(*L*, *S*). *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*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-conflict-heur-def*
already-propagated-unit-cls-conflict-def all-lits-of-mm-add-mset
all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$*-iff isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *vmtf-consD*
intro!: *ASSERT-leI isa-length-trail-pre*)

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*) ∈
[λD. *D* = *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *set-conflict-empty-def*
lookup-set-conflict-empty-def option-lookup-clause-rel-def
lookup-clause-rel-def)

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*) ∈
[λS. *get-conflict-init-wl S* = *None*]$_f$
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *set-empty-clause-as-conflict-heur-def add-empty-conflict-init-wl-def*
twl-st-heur-parsing-no-WL-def set-conflict-empty-def option-lookup-clause-rel-def

lookup-clause-rel-def isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*]
intro!: *isa-length-trail-pre ASSERT-leI*)

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*›
by (*intro frefI nres-relI*)
(*auto simp*: *add-clause-to-others-heur-def add-to-other-init.simps*
twl-st-heur-parsing-no-WL-def)

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 ‹$(S, T) \in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
shows ‹(*get-trail-wl-heur-init* S, *get-trail-init-wl* T) $\in$ *trail-pol* $\mathcal{A}$›
using *assms*
by (*cases* S; *auto simp*: *twl-st-heur-parsing-no-WL-def*; *fail*)+

definition *get-conflict-wl-is-None-init* :: ‹*nat twl-st-wl-init* $\Rightarrow$ *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 $\longleftrightarrow$ *get-conflict-init-wl* S = *None*›
by (*cases* S) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

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*) $\in$
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rename-tac x y*, *case-tac x*, *case-tac y*)
by (*auto simp*: *twl-st-heur-parsing-no-WL-def get-conflict-wl-is-None-heur-init-def option-lookup-clause-rel-def*
get-conflict-wl-is-None-init-def split: *option.splits*)

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*) $\in$
[λ(C, S). *literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*mset* C) $\wedge$ *distinct* C]$_f$
Id $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
supply [[*goals-limit=1*]]
unfolding *init-dt-step-wl-heur-def init-dt-step-wl-def uncurry-def*
option.case-eq-if get-conflict-wl-is-None-init-alt-def[*symmetric*]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict[*THEN fref-to-Down*,
unfolded comp-def]
propagate-unit-cls-heur-propagate-unit-cls[*THEN fref-to-Down-curry*, *unfolded comp-def*]
already-propagated-unit-cls-heur-already-propagated-unit-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
add-init-cls-heur-add-init-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
add-clause-to-others-heur-add-clause-to-others[*THEN fref-to-Down-curry*,
unfolded comp-def])
subgoal by (*auto simp*: *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*[*THEN fref-to-Down-unRET-Id*])
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def is-Nil-def split*: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by *auto*
subgoal by *simp*
subgoal by *simp*
subgoal by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-add-mset*

twl-st-heur-parsing-no-WL-def intro!: *polarity-pol-pre split*: *list.splits*)
subgoal for $C'S$ CT C T C' S
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id,
of ⟨*get-trail-init-wl T*⟩ ⟨*hd C*⟩])
(*auto simp*: *polarity-def twl-st-heur-parsing-no-WL-def*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
literals-are-in-$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def* *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
subgoal for $C'S$ CT C T C' S
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id,
of ⟨*get-trail-init-wl T*⟩ ⟨*hd C*⟩])
(*auto simp*: *polarity-def twl-st-heur-parsing-no-WL-def*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
literals-are-in-$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal by (*auto simp*: *list-mset-rel-def br-def*)
subgoal by (*simp add*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by *simp*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal for *x y x1 x2 C x2a*
by (*cases C*; *cases* ⟨*tl C*⟩)
(*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
done

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*)⟩
by (*auto simp*: *get-conflict-wl-is-None-heur-init-def intro*!: *ext*)

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*⟩
by (*cases S*) (*auto simp*: *polarity-st-heur-init-def*)

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*›
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

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 −
have ‹*set b* ⊆ {*0* ..< *Suc* (*Max* (*insert 0* (*set b*)))}›
by (*cases* ‹*set b* = {}›)
(*auto simp add*: *le-imp-less-Suc*)

from *card-mono*[*OF - this*] **show** *?thesis*
  using *distinct-card*[*OF assms*(*1*)] **by** *auto*
qed

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*
}›
by (*auto simp*: *isasat-atms-ext-rel-def distinct-length-le-Suc-Max*)

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*›
by (*auto simp*: *init-next-size-def uint32-max-uint32-def uint-max-def*)

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* ≤ *uint-max*);
  ASSERT(*length atms* ≤ *Suc n*);
  let n = *max i n*;
  (*if i* < *length-uint32-nat xs then do* {
    ASSERT(*xs*!*i* ≤ *uint64-max*);
    let atms = (*if xs*!*i AND one-uint64-nat* = *one-uint64-nat then atms else atms* @ [*i*]);
    RETURN (*xs*[*i* := (*sum-mod-uint64-max* (*xs* ! *i*) *2*) *OR one-uint64-nat*], *n*, *atms*)
  }
  else do {
    ASSERT(*i* + *1* ≤ *uint-max*);
    ASSERT(*length-uint32-nat xs* ≠ *0*);
    ASSERT(*i* < *init-next-size i*);
    RETURN ((*list-grow xs* (*init-next-size i*) *zero-uint64-nat*)[*i* := *one-uint64-nat*], *n*,
      atms @ [*i*])
  })
})›

lemma *init-valid-rep-upd-OR*:
‹*init-valid-rep* (*x1b*[*x1a* := *a OR one-uint64-nat*]) *x2* ⟷
  init-valid-rep (*x1b*[*x1a* := *one-uint64-nat*]) *x2* › (**is** ‹*?A* ⟷ *?B*›)
proof
  assume *?A*
  then have
    1: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *a OR one-uint64-nat*])› **and**
    2: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *a OR one-uint64-nat*] ! *L mod 2* = *1*› **and**

3: ‹∀ *L*<*length* (*x1b*[*x1a* := *a OR one-uint64-nat*]).
x1b[*x1a* := *a OR one-uint64-nat*] ! *L mod 2* = *1* ⟶
L ∈ *x2*›
unfolding *init-valid-rep-def* **by** *fast+*
have *1*: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *one-uint64-nat*])›
using *1* **by** *simp*
then have *2*: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1*›
using *2* **by** (*auto simp*: *nth-list-update′*)
then have *3*: ‹∀ *L*<*length* (*x1b*[*x1a* := *one-uint64-nat*]).
x1b[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1* ⟶
L ∈ *x2*›
using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
show *?B*
using *1 2 3*
unfolding *init-valid-rep-def* **by** *fast+*
next
assume *?B*
then have
1: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *one-uint64-nat*])› **and**
2: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1*› **and**
3: ‹∀ *L*<*length* (*x1b*[*x1a* := *one-uint64-nat*]).
x1b[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1* ⟶
L ∈ *x2*›
unfolding *init-valid-rep-def* **by** *fast+*
have *1*: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *a OR one-uint64-nat*])›
using *1* **by** *simp*
then have *2*: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *a OR one-uint64-nat*] ! *L mod 2* = *1*›
using *2* **by** (*auto simp*: *nth-list-update′ bitOR-1-if-mod-2-nat*)
then have *3*: ‹∀ *L*<*length* (*x1b*[*x1a* := *a OR one-uint64-nat*]).
x1b[*x1a* := *a OR one-uint64-nat*] ! *L mod 2* = *1* ⟶
L ∈ *x2*›
using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
show *?A*
using *1 2 3*
unfolding *init-valid-rep-def* **by** *fast+*
qed

lemma *init-valid-rep-insert*:
assumes *val*: ‹*init-valid-rep x1b x2*› **and** *le*: ‹*x1a* < *length x1b*›
shows ‹*init-valid-rep* (*x1b*[*x1a* := *one-uint64-nat*]) (*insert x1a x2*)›
proof −
have
1: ‹∀ *L*∈*x2*. *L* < *length x1b*› **and**
2: ‹∀ *L*∈*x2*. *x1b* ! *L mod 2* = *1*› **and**
3: ‹⋀*L*. *L*<*length x1b* ⟹ *x1b* ! *L mod 2* = *1* ⟶ *L* ∈ *x2*›
using *val* **unfolding** *init-valid-rep-def* **by** *fast+*
have *1*: ‹∀ *L*∈*insert x1a x2*. *L* < *length* (*x1b*[*x1a* := *one-uint64-nat*])›
using *1 le* **by** *simp*
then have *2*: ‹∀ *L*∈*insert x1a x2*. *x1b*[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1*›
using *2* **by** (*auto simp*: *nth-list-update′*)
then have *3*: ‹∀ *L*<*length* (*x1b*[*x1a* := *one-uint64-nat*]).
x1b[*x1a* := *one-uint64-nat*] ! *L mod 2* = *1* ⟶
L ∈ *insert x1a x2*›
using *3 le* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
show *?thesis*
using *1 2 3*

unfolding *init-valid-rep-def* **by** *fast+*
qed

lemma *init-valid-rep-extend*:
‹*init-valid-rep* (*x1b* @ *replicate n 0*) *x2* ⟷ *init-valid-rep* (*x1b*) *x2*›
(**is** ‹*?A* ⟷ *?B*› **is** ‹*init-valid-rep ?x1b - ⟷ -*›)
proof
assume *?A*
then have
1: ‹⋀*L*. *L*∈*x2* ⟹ *L* < *length ?x1b*› **and**
2: ‹⋀*L*. *L*∈*x2* ⟹ *?x1b* ! *L mod 2* = *1*› **and**
3: ‹⋀*L*. *L*<*length ?x1b* ⟹ *?x1b* ! *L mod 2* = *1* ⟶ *L* ∈ *x2*›
unfolding *init-valid-rep-def* **by** *fast+*
have *1*: ‹*L*∈*x2* ⟹ *L* < *length x1b*› **for** *L*
using *3*[*of L*] *2*[*of L*] *1*[*of L*]
by (*auto simp*: *nth-append split*: *if-splits*)
then have *2*: ‹∀ *L*∈*x2*. *x1b* ! *L mod 2* = *1*›
using *2* **by** (*auto simp*: *nth-list-update′*)
then have *3*: ‹∀ *L*<*length x1b*. *x1b* ! *L mod 2* = *1* ⟶ *L* ∈ *x2*›
using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
show *?B*
using *1 2 3*
unfolding *init-valid-rep-def* **by** *fast*
next
assume *?B*
then have
1: ‹⋀*L*. *L*∈*x2* ⟹ *L* < *length x1b*› **and**
2: ‹⋀*L*. *L*∈*x2* ⟹ *x1b* ! *L mod 2* = *1*› **and**
3: ‹⋀*L*. *L*<*length x1b* ⟶ *x1b* ! *L mod 2* = *1* ⟶ *L* ∈ *x2*›
unfolding *init-valid-rep-def* **by** *fast+*
have *10*: ‹∀ *L*∈*x2*. *L* < *length ?x1b*›
using *1* **by** *fastforce*
then have *20*: ‹*L*∈*x2* ⟹ *?x1b* ! *L mod 2* = *1*› **for** *L*
using *1*[*of L*] *2*[*of L*] *3*[*of L*] **by** (*auto simp*: *nth-list-update′ bitOR-1-if-mod-2-nat nth-append*)
then have *30*: ‹*L*<*length ?x1b* ⟹ *?x1b* ! *L mod 2* = *1* ⟶ *L* ∈ *x2*›**for** *L*
using *1*[*of L*] *2*[*of L*] *3*[*of L*]
by (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat nth-append*)
show *?A*
using *10 20 30*
unfolding *init-valid-rep-def* **by** *fast+*
qed

lemma *init-valid-rep-in-set-iff*:
‹*init-valid-rep x1b x2* ⟹ *x* ∈ *x2* ⟷ (*x* < *length x1b* ∧ (*x1b*!*x*) *mod 2* = *1*)›
unfolding *init-valid-rep-def*
by *auto*

lemma *add-to-atms-ext-op-set-insert*:
‹(*uncurry add-to-atms-ext*, *uncurry* (*RETURN oo Set.insert*))
∈ [λ(*n*, *l*). *n* ≤ *uint-max div 2*]$_f$ *nat-rel* ×$_f$ *isasat-atms-ext-rel* → ⟨*isasat-atms-ext-rel*⟩*nres-rel*›
proof −
have *H*: ‹*finite x2* ⟹ *Max* (*insert x1* (*insert 0 x2*)) = *Max* (*insert x1 x2*)›
‹*finite x2* ⟹ *Max* (*insert 0* (*insert x1 x2*)) = *Max* (*insert x1 x2*)›
for *x1* **and** *x2* :: ‹*nat set*›
by (*subst insert-commute*) *auto*
have [*simp*]: ‹(*a OR Suc 0*) *mod 2* = *Suc 0*› **for** *a*

by (*auto simp add*: *bitOR-1-if-mod-2-nat*)
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *isasat-atms-ext-rel-def add-to-atms-ext-def uncurry-def*
apply (*refine-vcg lhs-step-If*)
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *isasat-atms-ext-rel-def*[*symmetric*] *isasat-atms-ext-rel-alt-def* **by** *auto*
subgoal by *auto*
subgoal for *x y x1 x2 x1a x2a x1b x2b*
unfolding *comp-def*
apply (*rule RETURN-refine*)
apply (*subst in-pair-collect-simp*)
apply (*subst prod.case*)+
apply (*intro conjI impI allI*)
subgoal by (*simp add*: *init-valid-rep-upd-OR init-valid-rep-insert*
del: *one-uint64-nat-def*)
subgoal by (*auto simp*: *H Max-insert*[*symmetric*] *simp del*: *Max-insert*)
subgoal by *auto*
subgoal
using *sum-mod-uint64-max-le-uint64-max*
unfolding *bitOR-1-if-mod-2-nat one-uint64-nat-def*
by (*auto simp del*: *sum-mod-uint64-max-le-uint64-max simp*: *uint64-max-def*
sum-mod-uint64-max-def
elim!: *in-set-upd-cases*)
subgoal
unfolding *bitAND-1-mod-2 one-uint64-nat-def*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
unfolding *bitAND-1-mod-2 one-uint64-nat-def*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
unfolding *bitAND-1-mod-2 one-uint64-nat-def*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
by (*auto simp add*: *init-valid-rep-in-set-iff*)
done
subgoal by (*auto simp*: *uint-max-def*)
subgoal by (*auto simp*: *uint-max-def*)
subgoal by (*auto simp*: *uint-max-def init-next-size-def elim*: *neq-NilE*)
subgoal
unfolding *comp-def list-grow-def*
apply (*rule RETURN-refine*)
apply (*subst in-pair-collect-simp*)
apply (*subst prod.case*)+
apply (*intro conjI impI allI*)
subgoal
unfolding *init-next-size-def*
apply (*simp del*: *one-uint64-nat-def*)
apply (*subst init-valid-rep-insert*)
apply (*auto elim*: *neq-NilE*)
apply (*subst init-valid-rep-extend*)
apply (*auto elim*: *neq-NilE*)
done
subgoal by (*auto simp*: *H Max-insert*[*symmetric*] *simp del*: *Max-insert*)
subgoal by (*auto simp*: *init-next-size-def uint-max-def*)

subgoal
using *sum-mod-uint64-max-le-uint64-max*
unfolding *bitOR-1-if-mod-2-nat one-uint64-nat-def*
by (*auto simp del*: *sum-mod-uint64-max-le-uint64-max simp*: *uint64-max-def*
sum-mod-uint64-max-def
elim!: *in-set-upd-cases*)
subgoal by (*auto simp*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
done
done
qed

definition *extract-atms-cls* :: ‹$'a$ *clause-l* ⇒ $'a$ *set* ⇒ $'a$ *set*› **where**
‹*extract-atms-cls* C $\mathcal{A}_{in}$ = *fold* (λL $\mathcal{A}_{in}$. *insert* (*atm-of* L) $\mathcal{A}_{in}$) C $\mathcal{A}_{in}$›

definition *extract-atms-cls-i* :: ‹*nat clause-l* ⇒ *nat set* ⇒ *nat set nres*› **where**
‹*extract-atms-cls-i* C $\mathcal{A}_{in}$ = *nfoldli* C (λ-. *True*)
(λL $\mathcal{A}_{in}$. *do* {
ASSERT(*atm-of* $L \leq$ *uint-max div 2*);
RETURN(*insert* (*atm-of* L) $\mathcal{A}_{in}$)})
$\mathcal{A}_{in}$›

lemma *fild-insert-insert-swap*:
‹*fold* (λL. *insert* (*f* L)) C (*insert a* $\mathcal{A}_{in}$) = *insert a* (*fold* (λL. *insert* (*f* L)) C $\mathcal{A}_{in}$)›
by (*induction C arbitrary*: *a* $\mathcal{A}_{in}$) (*auto simp*: *extract-atms-cls-def*)

lemma *extract-atms-cls-alt-def*: ‹*extract-atms-cls* C $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ *atm-of* ‘ *set* C›
by (*induction C*) (*auto simp*: *extract-atms-cls-def fild-insert-insert-swap*)

lemma *extract-atms-cls-i-extract-atms-cls*:
‹(*uncurry extract-atms-cls-i*, *uncurry* (*RETURN oo extract-atms-cls*))
∈ [λ(C, $\mathcal{A}_{in}$). ∀ L∈*set* C. *nat-of-lit* $L \leq$ *uint-max*]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
proof −
have *H1*: ‹(*x1a*, *x1*) ∈ ⟨{(L, L'). $L = L' \wedge$ *nat-of-lit* $L \leq$ *uint-max*}⟩*list-rel*›
if
‹*case y of* (C, $\mathcal{A}_{in}$) ⇒ ∀ L∈*set* C. *nat-of-lit* $L \leq$ *uint-max*› **and**
‹(x, y) ∈ ⟨*nat-lit-lit-rel*⟩*list-rel* $\times_f$ *Id*› **and**
‹y = (*x1*, *x2*)› **and**
‹x = (*x1a*, *x2a*)›
for x :: ‹*nat literal list* × *nat set*› **and** y :: ‹*nat literal list* × *nat set*› **and**
x1 :: ‹*nat literal list*› **and** *x2* :: ‹*nat set*› **and** *x1a* :: ‹*nat literal list*› **and** *x2a* :: ‹*nat set*›
using *that* **by** (*auto simp*: *list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth*)

have *atm-le*: ‹*nat-of-lit xa* ≤ *uint-max* ⟹ *atm-of xa* ≤ *uint-max div 2*› **for** *xa*
by (*cases xa*) (*auto simp*: *uint-max-def*)

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *extract-atms-cls-i-def extract-atms-cls-def uncurry-def comp-def*
fold-eq-nfoldli
apply (*intro frefI nres-relI*)
apply (*refine-rcg H1*)
apply *assumption*+

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *atm-le*)
subgoal by *auto*
done
qed

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*))
∈ [λ(*N*, $\mathcal{A}_{in}$). ∀ *C*∈*set N*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint-max*]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*⟩
proof −
have *H1*: ⟨(*x1a*, *x1*) ∈ ⟨{(*C*, *C'*). *C* = *C'* ∧ (∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint-max*)}⟩*list-rel*⟩
if
⟨*case y of* (*N*, $\mathcal{A}_{in}$) ⇒ ∀ *C*∈*set N*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint-max*⟩ **and**
⟨(*x*, *y*) ∈ ⟨*Id*⟩*list-rel* $\times_f$ *Id*⟩ **and**
⟨*y* = (*x1*, *x2*)⟩ **and**
⟨*x* = (*x1a*, *x2a*)⟩
for *x* :: ⟨*nat literal list list* × *nat set*⟩ **and** *y* :: ⟨*nat literal list list* × *nat set*⟩ **and**
x1 :: ⟨*nat literal list list*⟩ **and** *x2* :: ⟨*nat set*⟩ **and** *x1a* :: ⟨*nat literal list list*⟩
and *x2a* :: ⟨*nat set*⟩
using *that* **by** (*auto simp*: *list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth*)

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *extract-atms-clss-i-def extract-atms-clss-def comp-def fold-eq-nfoldli uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg H1 extract-atms-cls-i-extract-atms-cls*[*THEN fref-to-Down-curry*,
unfolded comp-def])
apply *assumption+*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

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*⟩
by (*induction N arbitrary*: *a* $\mathcal{A}_{in}$) (*auto simp*: *extract-atms-cls-alt-def*)

lemma *extract-atms-clss-alt-def*:
⟨*extract-atms-clss N* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ ((⋃ *C*∈*set N*. *atm-of* ‘ *set C*))⟩
by (*induction N*)
(*auto simp*: *extract-atms-clss-def extract-atms-cls-alt-def*
fold-extract-atms-cls-union-swap)

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss* CS' {})› **for** CS'
by (*auto simp*: *extract-atms-clss-alt-def*)

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*›
by (*intro frefI nres-relI*)
(*simp add*: *op-extract-list-empty-def uint-max-def*
isasat-atms-ext-rel-def init-valid-rep-def extract-atms-clss-imp-empty-rel-def
del: *replicate-numeral*)

lemma *extract-atms-cls-Nil*[*simp*]:
‹*extract-atms-cls* [] $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$›
unfolding *extract-atms-cls-def fold.simps* **by** *simp*

lemma *extract-atms-clss-Cons*[*simp*]:
‹*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)›
by (*simp add*: *extract-atms-clss-def*)

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* {#} = {#}›
unfolding *all-lits-of-atms-m-def* **by** *auto*

definition (**in** −) *all-lits-of-atms-mm* :: ‹'*a multiset multiset* ⇒ '*a clause*› **where**
‹*all-lits-of-atms-mm N* = *poss* ($\bigcup$# *N*) + *negs* ($\bigcup$# *N*)›

lemma *all-lits-of-atms-m-all-lits-of-m*:
‹*all-lits-of-atms-m N* = *all-lits-of-m* (*poss N*)›
unfolding *all-lits-of-atms-m-def all-lits-of-m-def*
by (*induction N*) *auto*

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* {

$M \leftarrow SPEC(\lambda M.\ (M, []) \in\ trail\text{-}pol\ \mathcal{A})$;
$D \leftarrow SPEC(\lambda D.\ (D, None) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A})$;
$W \leftarrow SPEC\ (\lambda W.\ (W, empty\text{-}watched\ \mathcal{A}) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}))$;
$vm \leftarrow RES\ (isa\text{-}vmtf\text{-}init\ \mathcal{A}\ [])$;
$\varphi \leftarrow SPEC\ (phase\text{-}saving\ \mathcal{A})$;
$cach \leftarrow SPEC\ (cach\text{-}refinement\text{-}empty\ \mathcal{A})$;
let lbd = empty-lbd;
let vdom = [];
$RETURN\ (M, [], D, zero\text{-}uint32\text{-}nat, W, vm, \varphi, zero\text{-}uint32\text{-}nat, cach, lbd, vdom, False)\}$›

definition *init-state-wl-heur-fast* **where**
‹*init-state-wl-heur-fast = init-state-wl-heur*›

lemma *init-state-wl-heur-init-state-wl*:
‹$(\lambda\text{-}.\ (init\text{-}state\text{-}wl\text{-}heur\ \mathcal{A}), \lambda\text{-}.\ (RETURN\ init\text{-}state\text{-}wl)) \in$
$[\lambda\text{-}.\ isasat\text{-}input\text{-}bounded\ \mathcal{A}]_f\ \ unit\text{-}rel \to \langle twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl\ \mathcal{A}\ unbdd\rangle nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *init-state-wl-heur-def init-state-wl-def*
RES-RETURN-RES bind-RES-RETURN-eq RES-RES-RETURN-RES RETURN-def
twl-st-heur-parsing-no-WL-wl-def vdom-m-def empty-watched-def valid-arena-empty
intro!: *RES-refine*)

definition (**in** −)*to-init-state* :: ‹$nat\ twl\text{-}st\text{-}wl\text{-}init' \Rightarrow nat\ twl\text{-}st\text{-}wl\text{-}init$› **where**
‹$to\text{-}init\text{-}state\ S = (S, \{\#\})$›

definition (**in** −) *from-init-state* :: ‹$nat\ twl\text{-}st\text{-}wl\text{-}init\text{-}full \Rightarrow nat\ twl\text{-}st\text{-}wl$› **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**
‹$conflict\text{-}is\text{-}None\text{-}heur\text{-}wl = (\lambda(M, N, U, D, \text{-}).\ is\text{-}None\ D)$›

definition (**in** −) *finalise-init* **where**
‹*finalise-init = id*›

0.2.4 Parsing

lemma *init-dt-wl-heur-init-dt-wl*:
‹$(uncurry\ (init\text{-}dt\text{-}wl\text{-}heur\ unbdd), uncurry\ init\text{-}dt\text{-}wl) \in$
$[\lambda(CS, S).\ (\forall C \in set\ CS.\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C)) \wedge distinct\text{-}mset\text{-}set\ (mset\ `\ set\ CS)]_f$
$\langle Id\rangle list\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\ \mathcal{A}\ unbdd \to \langle twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\ \mathcal{A}\ unbdd\rangle\ nres\text{-}rel$›
proof −
have *H*: ‹$\bigwedge x\ y\ x1\ x2\ x1a\ x2a.$
$(\forall C{\in}set\ x1.\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C)) \wedge distinct\text{-}mset\text{-}set\ (mset\ `\ set\ x1) \Longrightarrow$
$(x1a, x1) \in \langle Id\rangle list\text{-}rel \Longrightarrow$
$(x1a, x1) \in \langle\{(C, C').\ C = C' \wedge literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C) \wedge$
$distinct\ C\}\rangle list\text{-}rel$›

apply (*auto simp*: *list-rel-def list-all2-conj*)
apply (*auto simp*: *list-all2-conv-all-nth distinct-mset-set-def*)
done

show *?thesis*
unfolding *init-dt-wl-heur-def init-dt-wl-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*case-tac y rule*: *prod.exhaust*)
apply (*case-tac x rule*: *prod.exhaust*)
apply (*simp only*: *prod.case prod-rel-iff*)
apply (*refine-vcg init-dt-step-wl-heur-init-dt-step-wl*[*THEN fref-to-Down-curry*] *H*)
apply *normalize-goal+*
subgoal by *fast*
subgoal by *fast*
subgoal by *simp*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
done
qed

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 −
obtain *M N D NE UE Q OC* **where**
T: ‹*T* = ((*M*,*N*, *D*, *NE*, *UE*, *Q*), *OC*)›
by (*cases T*) *auto*

obtain *M′ N′ D′ j W vm φ clvls cach lbd vdom* **where**
S: ‹*S* = (*M′*, *N′*, *D′*, *j*, *W*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *False*)›
using *failed* **by** (*cases S*) *auto*

have *valid*: ‹*valid-arena N′ N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
dom-m-vdom: ‹*set-mset* (*dom-m N*) ⊆ *set vdom*› **and**
W: ‹(*W*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)›
using *assms distinct-mset-dom*[*of N*] **apply** (*auto simp*: *twl-st-heur-parsing-no-WL-def S T*

simp flip: *distinct-mset-mset-distinct*)
by (*metis distinct-mset-set-mset-ident set-mset-mset subset-mset.eq-iff*)+
have *H*: ‹*RES* ({(*W*, *W'*).
$(W,\ W') \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge vdom\text{-}m\ \mathcal{A}\ W'\ N \subseteq set\text{-}mset\ (dom\text{-}m\ N)\}^{-1}$ “
{*W*. *Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)})
≤ *RES* ({(*W*, *W'*).
$(W,\ W') \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge vdom\text{-}m\ \mathcal{A}\ W'\ N \subseteq set\text{-}mset\ (dom\text{-}m\ N)\}^{-1}$ “
{*W*. *Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)})›
for *W'*
by (*rule order.refl*)
have *eq*: ‹*Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *None*, *NE*, *UE*, {#}, *xa*) ⟹
$vdom\text{-}m\ \mathcal{A}\ xa\ N = set\text{-}mset\ (dom\text{-}m\ N)$› **for** *xa*
by (*auto 5 5 simp*: *Watched-Literals-Watch-List-Initialisation.correct-watching-init.simps*
vdom-m-def)
show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *rewatch-heur-st-def T S*
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal by (*auto dest*: *valid-arena-vdom-subset simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist dom-m-vdom W lits*])
apply (*solves simp*)
apply (*solves simp*)
apply (*rule order-trans*[*OF ref-two-step'*])
apply (*rule rewatch-correctness*)
apply (*rule empty-watched-def*)
subgoal
using *assms*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*subst conc-fun-RES*)
apply (*rule H*) **apply** (*rule RETURN-RES-refine*)
apply (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def all-atms-def*[*symmetric*]
intro!: *exI*[*of - N*] *exI*[*of - D*] *exI*[*of - M*]
intro!:)
apply (*rule-tac x=W'* **in** *exI*)
apply (*auto simp*: *eq correct-watching-init-correct-watching dist*)
apply (*rule-tac x=W'* **in** *exI*)
apply (*auto simp*: *eq correct-watching-init-correct-watching dist*)
done
qed

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* =
$((\forall x \in set\ (get\text{-}vdom\text{-}heur\text{-}init\ S).\ x \leq uint64\text{-}max) \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\text{-}init\ S) \leq$

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**
⟨∀ *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 −
have *H*: ⟨*valid-arena x1g x1b* (*set x1p*)⟩ ⟨*set x1p* ⊆ *set x1p*⟩ ⟨*set-mset* (*dom-m x1b*) ⊆ *set x1p*⟩
⟨*distinct x1p*⟩ ⟨(*x1j*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩
if
xx': ⟨(*x*, *x'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*⟩ **and**
st: ⟨*x2c* = (*x1e*, *x2d*)⟩
⟨*x2b* = (*x1d*, *x2c*)⟩
⟨*x2a* = (*x1c*, *x2b*)⟩
⟨*x2* = (*x1b*, *x2a*)⟩
⟨*x1* = (*x1a*, *x2*)⟩
⟨*x'* = (*x1*, *x2e*)⟩
⟨*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*)⟩
⟨*x2f* = (*x1g*, *x2g*)⟩
⟨*x* = (*x1f*, *x2f*)⟩
for *x x' x1 x1a x2 x1b x2a x1c x2b x1d x2c x1e x2d x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p
proof −
show ⟨*valid-arena x1g x1b* (*set x1p*)⟩ ⟨*set x1p* ⊆ *set x1p*⟩ ⟨*set-mset* (*dom-m x1b*) ⊆ *set x1p*⟩
⟨*distinct x1p*⟩ ⟨(*x1j*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩
using *xx' distinct-mset-dom*[*of x1b*] **unfolding** *st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def empty-watched-def*
simp flip: *set-mset-mset distinct-mset-mset-distinct*)
qed

show *?thesis*
unfolding *init-dt-wl-heur-full-def init-dt-wl-full-def rewatch-heur-st-def*
apply (*refine-rcg rewatch-heur-rewatch*[*of - - - - - - $\mathcal{A}$*]
init-dt-wl-heur-init-dt-wl[*of True $\mathcal{A}$, THEN fref-to-Down-curry*])
subgoal using *assms* **by** *fast*
subgoal using *assms* **by** *fast*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*)
subgoal by (*auto dest*: *valid-arena-vdom-subset simp*: *twl-st-heur-parsing-no-WL-def*)
apply ((*rule H*; *assumption*)+)[*5*]
subgoal
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
literals-are-in-$\mathcal{L}_{in}$-mm-def all-lits-of-mm-union)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
empty-watched-def[*symmetric*] *map-fun-rel-def vdom-m-def*)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
empty-watched-def[*symmetric*])
done
qed

lemma *init-dt-wl-heur-full-init-dt-wl-spec-full*:
assumes
‹*init-dt-wl-pre CS T*› **and**
‹∀ *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*))›
apply (*rule order.trans*)
apply (*rule init-dt-wl-heur-full-init-dt-wl-full*[*OF assms*])
apply (*rule ref-two-step′*)
apply (*rule init-dt-wl-full-init-dt-wl-spec-full*[*OF assms*(*1*)])
done

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 −
have *K*: ‹*RETURN o mset-set* = (λ*v*. *do* {*v′* ← *SPEC*(λ*v′*. *v′* = *mset-set v*); *RETURN v′*})›
by *auto*
have *K′*: ‹*length x2a* < *uint32-max*› **if** ‹*distinct b*› ‹*init-valid-rep x1* (*set b*)›
‹*length x1* < *uint-max*› ‹*mset x2a* = *mset b*›**for** *x1 x2a b*

proof −
  have ‹*distinct x2a*›
    by (*simp add*: *same-mset-distinct-iff that*(*1*) *that*(*4*))
  have ‹*length x2a* = *length b*› ‹*set x2a* = *set b*›
    using ‹*mset x2a* = *mset b*› **apply** (*metis size-mset*)
     using ‹*mset x2a* = *mset b*› **by** (*rule mset-eq-setD*)
  then have ‹*set x2a* ⊆ {*0*..<*uint-max* − *1*}›
    using *that* **by** (*auto simp*: *init-valid-rep-def*)
  from *card-mono*[*OF* - *this*] **show** *?thesis*
    using ‹*distinct x2a*› **by** (*auto simp*: *uint-max-def distinct-card*)
qed
have *H-simple*: ‹*RETURN x2a*
   ≤ ⇓ (*list-mset-rel* ∩ {(*v*, *v′*). *length v* < *uint-max*})
      (*SPEC* (λ*v′*. *v′* = *mset-set y*))›
  if
   ‹(*x*, *y*) ∈ *isasat-atms-ext-rel*› **and**
   ‹*x2* = (*x1a*, *x2a*)› **and**
   ‹*x* = (*x1*, *x2*)›
  for *x* :: ‹*nat list* × *nat* × *nat list*› **and** *y* :: ‹*nat set*› **and** *x1* :: ‹*nat list*› **and**
   x2 :: ‹*nat* × *nat list*› **and** *x1a* :: ‹*nat*› **and** *x2a* :: ‹*nat list*›
  using *that mset-eq-length* **by** (*auto simp*: *isasat-atms-ext-rel-def list-mset-rel-def br-def*
      mset-set-set RETURN-def intro: *K′ intro*!: *RES-refine dest*: *mset-eq-length*)

show *?thesis*
  unfolding *extract-lits-sorted-def reorder-list-def K*
  apply (*intro frefI nres-relI*)
  apply (*refine-vcg H-simple*)
    apply *assumption*+
  by (*auto simp*: *lits-with-max-rel-def isasat-atms-ext-rel-def mset-set-set list-mset-rel-def*
     br-def dest!: *mset-eq-setD*)
qed

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*)
    ∈ *isa-vmtf-init* 𝒜 *au* ⟹ 𝒜 ≠ {#} ⟹ ∃ *y*. *an* = *Some y*›
  ‹((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*)
    ∈ *isa-vmtf-init* 𝒜 *au* ⟹ 𝒜 ≠ {#} ⟹ ∃ *y*. *am* = *Some y*›
 by (*auto simp*: *isa-vmtf-init-def vmtf-init-def*)

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*›
by (*auto simp*: *isa-vmtf-init-def vmtf-init-def Image-iff intro*!: *isa-vmtfI*)

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*))›
by (*auto 5 5 simp*: *finalise-init-def twl-st-heur-def twl-st-heur-parsing-no-WL-def*
twl-st-heur-parsing-no-WL-wl-def
finalise-init-code-def out-learned-def all-atms-def
twl-st-heur-post-parsing-wl-def
intro!: *ASSERT-leI intro*!: *isa-vmtf-init-isa-vmtf*
dest: *isa-vmtf-init-nemptyD*)

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*›
by (*intro frefI nres-relI*)
(*auto 5 5 simp*: *finalise-init-def twl-st-heur-def twl-st-heur-parsing-no-WL-def twl-st-heur-parsing-no-WL-wl-def*
finalise-init-code-def out-learned-def all-atms-def
twl-st-heur-post-parsing-wl-def
intro!: *ASSERT-leI intro*!: *isa-vmtf-init-isa-vmtf*
dest: *isa-vmtf-init-nemptyD*)

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-trail-D-fast* = *init-trail-D*›

definition *init-state-wl-D'* :: ‹*uint32 list* × *uint32* ⇒ (*trail-pol* × - × -) *nres*› **where**

‹*init-state-wl-D'* = $(\lambda(\mathcal{A}_{in}, n)$. *do* {
  ASSERT(*Suc* (*2* ∗ (*nat-of-uint32 n*)) ≤ *uint32-max*);
  let n = *Suc* (*nat-of-uint32 n*);
  let m = *2* ∗ *n*;
  M ← *init-trail-D* $\mathcal{A}_{in}$ *n m*;
  let N = [];
  let D = (*True*, *zero-uint32-nat*, *replicate n NOTIN*);
  let WS = *replicate m* [];
  vm ← *initialise-VMTF* $\mathcal{A}_{in}$ *n*;
  let φ = *replicate n False*;
  let cach = (*replicate n SEEN-UNKNOWN*, []);
  let lbd = *empty-lbd*;
  let vdom = [];
  RETURN (*M*, *N*, *D*, *zero-uint32-nat*, *WS*, *vm*, φ, *zero-uint32-nat*, *cach*, *lbd*, *vdom*, *False*)
})›

lemma *init-trail-D-ref*:
‹(*uncurry2 init-trail-D*, *uncurry2* (*RETURN ooo* (λ - - -. []))) ∈ [λ((*N*, *n*), *m*). *mset N* = $\mathcal{A}_{in}$ ∧
  distinct N ∧ (∀ *L*∈*set N*. *L* < *n*) ∧ *m* = *2* ∗ *n* ∧ *isasat-input-bounded* $\mathcal{A}_{in}]_f$
  ⟨*uint32-nat-rel*⟩*list-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* →
  ⟨*trail-pol* $\mathcal{A}_{in}$⟩ *nres-rel*›
proof −
  have *K*: ‹(∀ *L*∈*set N*. *nat-of-uint32 L* < *n*) ⟷
    (∀ *L* ∈# ($\mathcal{L}_{all}$ (*nat-of-uint32* ‘# *mset N*)). *atm-of L* < *n*)› **for** *N n*
    apply (*rule iffI*)
    subgoal by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
    subgoal by (*metis* (*full-types*) *image-eqI in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literal.sel*(*1*)
        set-image-mset set-mset-mset)
    done
  have *K'*: ‹(∀ *L*∈*set N*. *L* < *n*) ⟹
    (∀ *L* ∈# ($\mathcal{L}_{all}$ (*mset N*)). *nat-of-lit L* < *2* ∗ *n*)›
    (**is** ‹*?A* ⟹ *?B*›) **for** *N n*

  proof −
    assume *?A*
    then show *?B*
      apply (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
      apply (*case-tac L*)
      apply *auto*
      done
  qed
  show *?thesis*
    unfolding *init-trail-D-def*
    apply (*intro frefI nres-relI*)
    unfolding *uncurry-def Let-def comp-def trail-pol-def*
    apply *clarify*
    unfolding *RETURN-refine-iff*
    apply *clarify*
    apply (*intro conjI*)
    subgoal
      by (*auto simp*: *ann-lits-split-reasons-def*
        list-mset-rel-def Collect-eq-comp list-rel-def
        list-all2-op-eq-map-right-iff' uint32-nat-rel-def
        br-def in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
      dest: *multi-member-split*)
    subgoal

by *auto*
subgoal using K' **by** (*auto simp*: *polarity-def*)
subgoal
by (*auto simp*: *zero-uint32-def shiftr1-def*
nat-shiftr-div2 nat-of-uint32-shiftr in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
polarity-atm-def trail-pol-def K
phase-saving-def list-rel-mset-rel-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
list-rel-def uint32-nat-rel-def br-def list-all2-op-eq-map-right-iff'
ann-lits-split-reasons-def
list-mset-rel-def Collect-eq-comp)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp*: *control-stack.empty*)
subgoal by *auto*
done
qed

definition [*to-relAPP*]: *mset-rel A* $\equiv$ *p2rel* (*rel-mset* (*rel2p A*))
lemma *in-mset-rel-eq-f-iff*:
‹$(a, b) \in \langle\{(c, a).\ a = f\ c\}\rangle$*mset-rel* $\longleftrightarrow b = f$ ‘# a›
using *ex-mset*[*of a*]
by (*auto simp*: *mset-rel-def br-def rel2p-def*[*abs-def*] *p2rel-def rel-mset-def*
list-all2-op-eq-map-right-iff' cong: *ex-cong*)

lemma *in-mset-rel-eq-f-iff-set*:
‹$\langle\{(c, a).\ a = f\ c\}\rangle$*mset-rel* $= \{(b, a).\ a = f$ ‘# $b\}$›
using *in-mset-rel-eq-f-iff*[*of - - f*] **by** *blast*

lemma *init-state-wl-D0*:
‹(*init-state-wl-D'*, *init-state-wl-heur*) $\in$
$[\lambda N.\ N = \mathcal{A}_{in} \wedge$ *distinct-mset* $\mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$
lits-with-max-rel O $\langle$*uint32-nat-rel*$\rangle$*mset-rel* $\rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r$ *nat-rel* $\times_r \langle\langle Id\rangle$*list-rel*$\rangle$*list-rel* $\times_r$
$Id \times_r \langle$*bool-rel*$\rangle$*list-rel* $\times_r Id \times_r Id \times_r Id\rangle$*nres-rel*›
(**is** ‹*?C* $\in$ [*?Pre*]$_f$ *?arg* $\rightarrow$ $\langle$*?im*$\rangle$*nres-rel*›)
proof –
have *init-state-wl-heur-alt-def*: ‹*init-state-wl-heur* $\mathcal{A}_{in}$ = *do* {
$M \leftarrow$ *SPEC* ($\lambda M.\ (M, []) \in$ *trail-pol* $\mathcal{A}_{in}$);
$N \leftarrow$ *RETURN* [];
$D \leftarrow$ *SPEC* ($\lambda D.\ (D, None) \in$ *option-lookup-clause-rel* $\mathcal{A}_{in}$);
$W \leftarrow$ *SPEC* ($\lambda W.\ (W,$ *empty-watched* $\mathcal{A}_{in}) \in \langle Id\rangle$*map-fun-rel* $(D_0\ \mathcal{A}_{in})$);
vm $\leftarrow$ *RES* (*isa-vmtf-init* $\mathcal{A}_{in}$ []);
$\varphi \leftarrow$ *SPEC* (*phase-saving* $\mathcal{A}_{in}$);
cach $\leftarrow$ *SPEC* (*cach-refinement-empty* $\mathcal{A}_{in}$);
let lbd = *empty-lbd*;
let vdom = [];
RETURN (M, N, D, *0*, W, *vm*, φ, *zero-uint32-nat*, *cach*, *lbd*, *vdom*, *False*)}› **for** $\mathcal{A}_{in}$
unfolding *init-state-wl-heur-def Let-def* **by** *auto*

have *tr*: ‹*distinct-mset* $\mathcal{A}_{in} \wedge (\forall L \in\# \mathcal{A}_{in}.\ L < b) \Longrightarrow$

$(\mathcal{A}_{in}{}', \mathcal{A}_{in}) \in \langle uint32\text{-}nat\text{-}rel\rangle list\text{-}rel\text{-}mset\text{-}rel \Longrightarrow isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in} \Longrightarrow$
$b' = 2 * b \Longrightarrow$
$init\text{-}trail\text{-}D\ \mathcal{A}_{in}{}'\ b\ (2 * b) \le \Downarrow (trail\text{-}pol\ \mathcal{A}_{in})\ (RETURN\ [])$› **for** $b'\ b\ \mathcal{A}_{in}\ \mathcal{A}_{in}{}'\ x$
by (*rule init-trail-D-ref*[*unfolded fref-def nres-rel-def*, *simplified*, *rule-format*])
(*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def*)

have [*simp*]: ‹*comp-fun-idem* (*max* :: ′*a* :: {*zero*,*linorder*} ⇒ -)›
unfolding *comp-fun-idem-def comp-fun-commute-def comp-fun-idem-axioms-def*
by (*auto simp*: *max-def*[*abs-def*] *intro*!: *ext*)
have [*simp*]: ‹*fold max x a* = *Max* (*insert a* (*set x*))› **for** *x* **and** *a* :: ‹′*a* :: {*zero*,*linorder*}›
by (*auto simp*: *Max.eq-fold comp-fun-idem.fold-set-fold*)
have *in-N0*: ‹$L \in set\ \mathcal{A}_{in} \Longrightarrow nat\text{-}of\text{-}uint32\ L < Suc\ (nat\text{-}of\text{-}uint32\ (Max\ (insert\ 0\ (set\ \mathcal{A}_{in}))))$›
for $L\ \mathcal{A}_{in}$
using *Max-ge*[*of* ‹$insert\ 0\ (set\ \mathcal{A}_{in})$› *L*]
apply (*auto simp del*: *Max-ge simp*: *nat-shiftr-div2 nat-of-uint32-shiftr*)
using *div-le-mono le-imp-less-Suc nat-of-uint32-le-iff* **by** *blast*
define *P* **where** ‹$P\ x = \{(a, b).\ b = [] \wedge (a, b) \in trail\text{-}pol\ x\}$› **for** *x*
have *P*: ‹$(c, []) \in P\ x \longleftrightarrow (c, []) \in trail\text{-}pol\ x$› **for** *c x*
unfolding *P-def* **by** *auto*
have [*simp*]: ‹$\bigwedge a\ \mathcal{A}_{in}.\ (a, \mathcal{A}_{in}) \in \langle uint32\text{-}nat\text{-}rel\rangle mset\text{-}rel \longleftrightarrow \mathcal{A}_{in} = nat\text{-}of\text{-}uint32$ ‘# *a*›
by (*auto simp*: *uint32-nat-rel-def br-def in-mset-rel-eq-f-iff list-rel-mset-rel-def*)
have [*simp*]: ‹(*a*, *nat-of-uint32* ‘# *mset a*) ∈ ⟨*uint32-nat-rel*⟩*list-rel-mset-rel*› **for** *a*
unfolding *list-rel-mset-rel-def*
by (*rule relcompI* [*of* - ‹*map nat-of-uint32 a*›])
(*auto simp*: *list-rel-def uint32-nat-rel-def br-def list-all2-op-eq-map-right-iff*′
list-mset-rel-def)
have *init*: ‹*init-trail-D x1* (*Suc* (*nat-of-uint32 x2*))
(*2* ∗ *Suc* (*nat-of-uint32 x2*)) ≤
$SPEC\ (\lambda c.\ (c, []) \in trail\text{-}pol\ \mathcal{A}_{in})$›
if ‹$distinct\text{-}mset\ \mathcal{A}_{in}$› **and** *x*: ‹$(\mathcal{A}_{in}{}', \mathcal{A}_{in}) \in\ ?arg$› **and**
‹$\mathcal{A}_{in}{}' = (x1, x2)$› **and** ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}$›
for $\mathcal{A}_{in}\ \mathcal{A}_{in}{}'\ x1\ x2$
unfolding *x P*
by (*rule tr*[*unfolded conc-fun-RETURN*])
(*use that* **in** ‹*auto simp*: *lits-with-max-rel-def dest*: *in-N0*›)

have *H*:
‹(*replicate* (*2* ∗ *Suc* (*nat-of-uint32 b*)) [], $empty\text{-}watched\ \mathcal{A}_{in}$)
$\in \langle Id\rangle map\text{-}fun\text{-}rel\ ((\lambda L.\ (nat\text{-}of\text{-}lit\ L, L))$ ‘ $set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}_{in}))$›
if ‹$(x, \mathcal{A}_{in}) \in\ ?arg$› **and**
‹*x* = (*a*, *b*)›
for $\mathcal{A}_{in}\ x\ a\ b$
using *that* **unfolding** *map-fun-rel-def*
by (*auto simp*: *empty-watched-def* $\mathcal{L}_{all}$*-def*
lits-with-max-rel-def
intro!: *nth-replicate dest*!: *in-N0*
simp del: *replicate.simps*)
have *initialise-VMTF*: ‹(∀ *L*∈#*aa*. *L* < *b*) ∧ *distinct-mset aa* ∧ (*a*, *aa*) ∈
⟨*uint32-nat-rel*⟩*list-rel-mset-rel* ∧ *size aa* < *uint32-max* ⟹
initialise-VMTF a b ≤ *RES* (*isa-vmtf-init aa* [])›
for *aa b a*
using *initialise-VMTF*[*of aa*, *THEN fref-to-Down-curry*, *of aa b a b*]
by (*auto simp*: *isa-vmtf-init-def conc-fun-RES*)
have [*simp*]: ‹(*x*, *y*) ∈ ⟨*uint32-nat-rel*⟩*list-rel-mset-rel* ⟹ *L* ∈# *y* ⟹
L < *Suc* (*nat-of-uint32* (*Max* (*insert 0* (*set x*))))›
for *x y L*

by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def uint32-nat-rel-def list-all2-op-eq-map-right-iff′ list-mset-rel-def dest*: *in-N0*)

have *initialise-VMTF*: ‹*initialise-VMTF a* (*Suc* (*nat-of-uint32 b*)) ≤ ⇓ *Id* (*RES* (*isa-vmtf-init y* []))›
if ‹(*x*, *y*) ∈ *?arg*› **and** ‹*distinct-mset y*› **and** ‹*length a* < *uint-max*› **and** ‹*x* = (*a*, *b*)› **for** *x y a b*
using *that*
by (*auto simp*: *P-def lits-with-max-rel-def intro*!: *initialise-VMTF in-N0*)
have *K*[*simp*]: ‹(*x*, $\mathcal{A}_{in}$) ∈ ⟨*uint32-nat-rel*⟩*list-rel-mset-rel* ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹ *L* < *Suc* (*nat-of-uint32* (*Max* (*insert 0* (*set x*))))›
for *x L* $\mathcal{A}_{in}$
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def uint32-nat-rel-def list-all2-op-eq-map-right-iff′ list-mset-rel-def*)
have *cach*: ‹*RETURN* (*replicate* (*Suc* (*nat-of-uint32 b*)) *SEEN-UNKNOWN*, [])
≤ ⇓ *Id*
(*SPEC* (*cach-refinement-empty y*))›
if
‹*y* = $\mathcal{A}_{in}$ ∧ *distinct-mset* $\mathcal{A}_{in}$› **and**
‹(*x*, *y*) ∈ *?arg*› **and**
‹*x* = (*a*, *b*)›
for *M W vm vma φ x y a b*
proof −
show *?thesis*
unfolding *cach-refinement-empty-def RETURN-RES-refine-iff cach-refinement-alt-def Bex-def*
by (*rule exI*[*of* - ‹(*replicate* (*Suc* (*nat-of-uint32 b*)) *SEEN-UNKNOWN*, [])›]) (*use that* **in**
‹*auto simp*: *map-fun-rel-def empty-watched-def* $\mathcal{L}_{all}$*-def*
list-mset-rel-def lits-with-max-rel-def
simp del: *replicate-Suc*
dest!: *in-N0 intro*: *K*›)
qed
have *conflict*: ‹*RETURN* (*True*, *zero-uint32-nat*, *replicate* (*Suc* (*nat-of-uint32 b*)) *NOTIN*)
≤ *SPEC* (λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}_{in}$)›
if
‹*y* = $\mathcal{A}_{in}$ ∧ *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$› **and**
‹((*a*, *b*), $\mathcal{A}_{in}$) ∈ *lits-with-max-rel O* ⟨*uint32-nat-rel*⟩*mset-rel*› **and**
‹*x* = (*a*, *b*)›
for *a b x y*
proof −
have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹
L < *Suc* (*nat-of-uint32 b*)› **for** *L*
using *that in-N0* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
lits-with-max-rel-def)
then show *?thesis*
by (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def simp del: *replicate-Suc*
intro: *mset-as-position.intros*)
qed
have [*simp*]:
‹*NO-MATCH 0 a1* ⟹ *max 0* (*Max* (*insert a1* (*set a2*))) = *max a1* (*Max* (*insert 0* (*set a2*)))›
for *a1* :: *uint32* **and** *a2*
by (*metis* (*mono-tags*, *lifting*) *List.finite-set Max-insert all-not-in-conv finite-insert insertI1 insert-commute*)
have *le-uint32*: ‹∀ *L*∈#$\mathcal{L}_{all}$ (*nat-of-uint32* ‘# *mset a*). *nat-of-lit L* ≤ *uint-max* ⟹
Suc (2 ∗ *nat-of-uint32* (*Max* (*insert 0* (*set a*)))) ≤ *uint-max*› **for** *a*
apply (*induction a*)

apply (*auto simp*: *uint32-max-def*)
apply (*auto simp*: *max-def* $\mathcal{L}_{all}$*-add-mset*)
done
show *?thesis*
apply (*intro frefI nres-relI*)
subgoal for *x y*
unfolding *init-state-wl-heur-alt-def init-state-wl-D'-def*
apply (*rewrite* **in** ‹*let - = Suc -in -*› *Let-def*)
apply (*rewrite* **in** ‹*let - = 2 ∗ -in -*› *Let-def*)
apply (*cases x*; *simp only*: *prod.case*)
apply (*refine-rcg init*[*of y x*] *initialise-VMTF cach*)
subgoal for *a b* **by** (*auto simp*: *lits-with-max-rel-def intro*: *le-uint32*)
subgoal by (*auto intro*!: *K*[*of -* $\mathcal{A}_{in}$] *simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
lits-with-max-rel-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict*)
subgoal by (*rule RETURN-rule*) (*rule H*; *simp only*:)
apply *assumption*
subgoal by *fast*
subgoal by (*auto simp*: *lits-with-max-rel-def P-def*)
subgoal by *simp*
subgoal unfolding *phase-saving-def lits-with-max-rel-def* **by** (*auto intro*!: *K*)
subgoal by *fast*
subgoal by *fast*
apply *assumption*
apply (*rule refl*)
subgoal by (*auto simp*: *P-def init-rll-def option-lookup-clause-rel-def*
lookup-clause-rel-def lits-with-max-rel-def
simp del: *replicate.simps*
intro!: *mset-as-position.intros K*)
done
done
qed

lemma *init-state-wl-D'*:
‹(*init-state-wl-D'*, *init-state-wl-heur*) ∈
[$\lambda\mathcal{A}_{in}$. *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$
lits-with-max-rel O ⟨*uint32-nat-rel*⟩*mset-rel* →
⟨*Id* $\times_r$ *Id* $\times_r$
Id $\times_r$ *nat-rel* $\times_r$ ⟨⟨*Id*⟩*list-rel*⟩*list-rel* $\times_r$
Id $\times_r$ ⟨*bool-rel*⟩*list-rel* $\times_r$ *Id* $\times_r$ *Id* $\times_r$ *Id* $\times_r$ *Id*⟩*nres-rel*›
apply −
apply (*intro frefI nres-relI*)
by (*rule init-state-wl-D0*[*THEN fref-to-Down*, *THEN order-trans*]) *auto*

lemma *init-state-wl-heur-init-state-wl'*:
‹(*init-state-wl-heur*, *RETURN o* (λ-. *init-state-wl*))
∈ [λ*N*. *N* = $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *Id* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}_{in}$ *True*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *comp-def*
using *init-state-wl-heur-init-state-wl*[*THEN fref-to-Down*, *of* $\mathcal{A}_{in}$ ‹()› ‹()›]
by *auto*

lemma *all-blits-are-in-problem-init-blits-in*: ⟨*all-blits-are-in-problem-init* $S \Longrightarrow$ *blits-in-*$\mathcal{L}_{in}$ S⟩
  unfolding *blits-in-*$\mathcal{L}_{in}$*-def*
  by (*cases* S)
   (*auto simp*: *all-blits-are-in-problem-init.simps*
    $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-lits-def*)

lemma *correct-watching-init-blits-in-*$\mathcal{L}_{in}$:
  assumes ⟨*correct-watching-init* S⟩
  shows ⟨*blits-in-*$\mathcal{L}_{in}$ S⟩
proof −
  show *?thesis*
    using *assms*
    by (*cases* S)
      (*auto simp*: *all-blits-are-in-problem-init-blits-in*
      correct-watching-init.simps)
 qed

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 $\Longrightarrow$ *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))⟩
  unfolding *init-dt-wl*′*-def*
  by (*refine-vcg bind-refine-spec*[*OF - init-dt-wl-init-dt-wl-spec*])
   (*auto intro*!: *RETURN-RES-refine*)

lemma *init-dt-wl*′*-init-dt*:
  ⟨*init-dt-wl-pre* CS S $\Longrightarrow$ (S, S') ∈ *state-wl-l-init* $\Longrightarrow$ ∀ C∈*set* CS. *distinct* C $\Longrightarrow$
  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')⟩
  unfolding *init-dt-wl*′*-def*
  apply (*refine-vcg bind-refine*[*of - - - - -* ⟨*RETURN*⟩, *OF init-dt-wl-init-dt*, *simplified*])
  subgoal for S T
    by (*cases* S; *cases* T)
      auto
  done

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*⟩
unfolding *isasat-init-fast-slow-def trail-pol-slow-of-fast-alt-def convert-wlists-to-nat-conv-def*
by *auto*

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*⟩
unfolding *atoms-hash-int-empty-def array-fold-custom-replicate*
by *sepref*

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*⟩
unfolding *distinct-atms-int-empty-def array-fold-custom-replicate arl32.fold-custom-empty*
by *sepref*

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-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} → *vmtf-remove-conc-option-fst-As*⟩
supply *nat-of-uint32-int32-assn*[*sepref-fr-rules*] *uint64-max-def*[*simp*] *uint32-max-def*[*simp*]
unfolding *initialise-VMTF-def vmtf-cons-def Suc-eq-plus1 one-uint64-nat-def*[*symmetric*]
apply (*rewrite* **in** ⟨(-, -, *Some* ¤)⟩ *annotate-assn*[**where** *A*=⟨*uint32-nat-assn*⟩])
apply (*rewrite* **in** ⟨$WHILE_T$ - - (-, -, ¤)⟩ *annotate-assn*[**where** *A*=⟨*option-assn uint32-nat-assn*⟩])
apply (*rewrite* **in** ⟨*do* {*ASSERT* -; *let* - = ¤; -}⟩ *annotate-assn*[**where** *A*=⟨*uint32-nat-assn*⟩])
apply (*rewrite* **in** ⟨*let* - = ¤ *in* - ⟩ *array-fold-custom-replicate op-list-replicate-def*[*symmetric*])
apply (*rewrite* **in** ⟨*VMTF-Node zero-uint64-nat* ¤ -⟩ *annotate-assn*[**where** *A*=⟨*option-assn uint32-nat-assn*⟩])
apply (*rewrite* **in** ⟨*VMTF-Node zero-uint64-nat* - ¤⟩ *annotate-assn*[**where** *A*=⟨*option-assn uint32-nat-assn*⟩])
supply [[*goals-limit* = *1*]]
by *sepref*

declare *initialise-VMTF-code.refine*[*sepref-fr-rules*]

sepref-definition *propagate-unit-cls-code*
is ⟨*uncurry* (*propagate-unit-cls-heur*)⟩
:: ⟨*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*⟩
supply [[*goals-limit=1*]] *DECISION-REASON-def*[*simp*]
unfolding *propagate-unit-cls-heur-def isasat-init-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*] *zero-uint64-nat-def*[*symmetric*]
by *sepref*

sepref-definition *propagate-unit-cls-code-unb*
is ⟨*uncurry* (*propagate-unit-cls-heur*)⟩

:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
supply [[*goals-limit=1*]] *DECISION-REASON-def*[*simp*]
unfolding *propagate-unit-cls-heur-def isasat-init-unbounded-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

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-assn unat-lit-assn*)^k $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-heur-def isasat-init-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

sepref-definition *already-propagated-unit-cls-code-unb*
is ‹*uncurry already-propagated-unit-cls-heur*›
:: ‹(*list-assn unat-lit-assn*)^k $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-heur-def isasat-init-unbounded-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

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-of* $L <$ *length* xs]$_a$
unat-lit-assn^k $*_a$ *conflict-option-rel-assn*^{d} $\rightarrow$ *conflict-option-rel-assn*›
supply *one-uint32-nat*[*sepref-fr-rules*]
unfolding *set-conflict-unit-heur-def one-uint32-nat-def*[*symmetric*] *ISIN-def*[*symmetric*]
by *sepref*

declare *set-conflict-unit-code.refine*[*sepref-fr-rules*]

sepref-definition *conflict-propagated-unit-cls-code*
is ‹*uncurry* (*conflict-propagated-unit-cls-heur*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *conflict-propagated-unit-cls-heur-def isasat-init-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

sepref-definition *conflict-propagated-unit-cls-code-unb*
is ‹*uncurry conflict-propagated-unit-cls-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *conflict-propagated-unit-cls-heur-def isasat-init-unbounded-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

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*›
:: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
supply [[*goals-limit=1*]] *append-ll-def*[*simp*]
unfolding *add-init-cls-heur-def isasat-init-unbounded-assn-def add-init-cls-heur-unb-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*] *nat-of-uint32-conv-def if-True simp-thms*
unfolding *isasat-init-assn-def Array-List-Array.swap-ll-def*[*symmetric*]
nth-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*]
append-ll-def[*symmetric*]
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *op-list-copy-def*[*symmetric*])
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *op-array-of-list-def*[*symmetric*])
by *sepref*

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*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *uint64-max-def uint32-max-def*)

lemma *add-init-cls-code-bI2*:
assumes
‹*length at* ≤ *Suc* (*Suc uint-max*)›
shows ‹*5* ≤ *uint64-max* − *length at*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *uint64-max-def uint32-max-def*)

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*›
using *assms* **by** (*auto simp*: *uint64-max-def uint32-max-def*)

sepref-definition *add-init-cls-code-b*
is ‹*uncurry add-init-cls-heur-b*›
:: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]] *append-ll-def*[*simp*] *le-uint32-max-le-uint64-max*[*intro*] *add-init-clss-codebI*[*intro*]
uint64-max-uint64-nat-assn[*sepref-fr-rules*] *add-init-cls-code-bI*[*intro*] *add-init-cls-code-bI2*[*intro*]
unfolding *add-init-cls-heur-def isasat-init-unbounded-assn-def add-init-cls-heur-b-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*] *nat-of-uint32-conv-def*

five-uint64-nat-def[*symmetric*]
unfolding *isasat-init-assn-def Array-List-Array.swap-ll-def*[*symmetric*]
nth-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*] *uint64-max-uint64-def*[*symmetric*]
append-ll-def[*symmetric*] *fm-add-new-fast-def*[*symmetric*]
apply (*rewrite at* ‹- ≤ - − ⌑ − -› *length-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *op-list-copy-def*[*symmetric*])
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *op-array-of-list-def*[*symmetric*])
by *sepref*

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*›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-conflict-heur-def isasat-init-assn-def*
PR-CONST-def cons-trail-Propagated-def[*symmetric*]
by *sepref*

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*›
:: ‹$conflict\text{-}option\text{-}rel\text{-}assn^d \rightarrow_a conflict\text{-}option\text{-}rel\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *lookup-set-conflict-empty-def*
by *sepref*

declare *set-conflict-empty-code.refine*[*sepref-fr-rules*]

sepref-definition *set-empty-clause-as-conflict-code*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹$isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *set-empty-clause-as-conflict-heur-def isasat-init-assn-def*
by *sepref*

sepref-definition *set-empty-clause-as-conflict-code-unb*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹$isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *set-empty-clause-as-conflict-heur-def isasat-init-unbounded-assn-def*
by *sepref*

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*›
:: ‹$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *add-clause-to-others-heur-def isasat-init-assn-def*
by *sepref*

sepref-definition *add-clause-to-others-code-unb*
is ⟨*uncurry add-clause-to-others-heur*⟩
:: ⟨$(list\text{-}assn\ unat\text{-}lit\text{-}assn)^k *_a isasat\text{-}init\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *add-clause-to-others-heur-def isasat-init-unbounded-assn-def*
by *sepref*

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 ⟨$(return\ o\ list\text{-}length\text{-}1\text{-}code,\ RETURN\ o\ list\text{-}length\text{-}1) \in (list\text{-}assn\ R)^k \rightarrow_a bool\text{-}assn$⟩
proof −
obtain R' **where**
⟨$R' = the\text{-}pure\ R$⟩ **and**
$R\text{-}R'$: ⟨$R = pure\ R'$⟩
using *assms* **by** *fastforce*
show *?thesis*
unfolding $R\text{-}R'$ *list-assn-pure-conv*
by (*sepref-to-hoare*)
(*sep-auto simp*: *list-length-1-code-def list-rel-def list-all2-lengthD*[*symmetric*]
split: *list.splits*)
qed

sepref-definition *get-conflict-wl-is-None-init-code*
is ⟨*RETURN o get-conflict-wl-is-None-heur-init*⟩
:: ⟨$isasat\text{-}init\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
unfolding *get-conflict-wl-is-None-heur-init-alt-def isasat-init-assn-def length-ll-def*[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

sepref-definition *get-conflict-wl-is-None-init-code-unb*
is ⟨*RETURN o get-conflict-wl-is-None-heur-init*⟩
:: ⟨$isasat\text{-}init\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
unfolding *get-conflict-wl-is-None-heur-init-alt-def isasat-init-unbounded-assn-def*
length-ll-def[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

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*)⟩
:: ⟨$[\lambda(S, L).\ polarity\text{-}pol\text{-}pre\ (get\text{-}trail\text{-}wl\text{-}heur\text{-}init\ S)\ L]_a\ isasat\text{-}init\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$⟩
unfolding *polarity-st-heur-init-def isasat-init-assn-def*
supply [[*goals-limit = 1*]]
by *sepref*

sepref-definition *polarity-st-heur-init-code-unb*
is ⟨*uncurry* (*RETURN oo polarity-st-heur-init*)⟩
:: ⟨$[\lambda(S, L).\ polarity\text{-}pol\text{-}pre\ (get\text{-}trail\text{-}wl\text{-}heur\text{-}init\ S)\ L]_a$
$isasat\text{-}init\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow tri\text{-}bool\text{-}assn$⟩
unfolding *polarity-st-heur-init-def isasat-init-unbounded-assn-def*

supply [[*goals-limit* = *1*]]
by *sepref*

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→$_a$ *bool-assn*›
by *sepref-to-hoare* (*sep-auto split*: *list.splits*)

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›
supply [[*goals-limit*=*1*]]
supply *polarity-None-undefined-lit*[*simp*] *polarity-st-init-def*[*simp*]
option.splits[*split*] *get-conflict-wl-is-None-heur-init-alt-def*[*simp*]
tri-bool-eq-def[*simp*]
unfolding *init-dt-step-wl-heur-def lms-fold-custom-empty PR-CONST-def*
add-init-cls-heur-unb-def[*symmetric*] *init-dt-step-wl-heur-unb-def*
unfolding *watched-app-def*[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
unfolding *lms-fold-custom-empty swap-ll-def*[*symmetric*]
unfolding
cons-trail-Propagated-def[*symmetric*] *get-conflict-wl-is-None-init*
polarity-st-heur-init-alt-def[*symmetric*]
get-conflict-wl-is-None-heur-init-alt-def[*symmetric*]
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *UNSET-def*[*symmetric*]
tri-bool-eq-def[*symmetric*]
by *sepref*

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›
supply [[*goals-limit*=*1*]]
supply *polarity-None-undefined-lit*[*simp*] *polarity-st-init-def*[*simp*]
option.splits[*split*] *get-conflict-wl-is-None-heur-init-alt-def*[*simp*]
tri-bool-eq-def[*simp*]
unfolding *init-dt-step-wl-heur-def lms-fold-custom-empty PR-CONST-def*
add-init-cls-heur-b-def[*symmetric*] *init-dt-step-wl-heur-b-def*
unfolding *watched-app-def*[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
unfolding *lms-fold-custom-empty swap-ll-def*[*symmetric*]
unfolding
cons-trail-Propagated-def[*symmetric*] *get-conflict-wl-is-None-init*
polarity-st-heur-init-alt-def[*symmetric*]
get-conflict-wl-is-None-heur-init-alt-def[*symmetric*]

SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *UNSET-def*[*symmetric*]
tri-bool-eq-def[*symmetric*]
by *sepref*

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*)
$\in [\lambda L.\ L \leq \textit{uint32-max div 2}]_a$ $\textit{uint32-nat-assn}^k \rightarrow \textit{uint32-nat-assn}$›
by (*sepref-to-hoare*)
(*sep-auto simp*: *init-next-size-def br-def uint32-nat-rel-def nat-of-uint32-add*
nat-of-uint32-distrib-mult2 uint-max-def)

sepref-definition *nat-lit-lits-init-assn-assn-in*
is ‹*uncurry add-to-atms-ext*›
:: ‹$\textit{uint32-nat-assn}^k *_a \textit{isasat-atms-ext-rel-assn}^d \rightarrow_a \textit{isasat-atms-ext-rel-assn}$›
supply [[*goals-limit=1*]]
unfolding *add-to-atms-ext-def two-uint64-nat-def*[*symmetric*] *Suc-0-le-uint64-max*[*simp*]
heap-array-set-u-def[*symmetric*]
by *sepref*

lemma [*sepref-fr-rules*]:
‹(*uncurry nat-lit-lits-init-assn-assn-in*, *uncurry* (*RETURN* ∘∘ *op-set-insert*))
$\in [\lambda(a,\ b).\ a \leq \textit{uint-max div 2}]_a$
$\textit{uint32-nat-assn}^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow \textit{nat-lit-list-hm-assn}$›
by (*rule nat-lit-lits-init-assn-assn-in.refine*[*FCOMP add-to-atms-ext-op-set-insert*
[*unfolded convert-fref op-set-insert-def*[*symmetric*]]])

sepref-definition *extract-atms-cls-imp*
is ‹*uncurry extract-atms-cls-i*›
:: ‹$(\textit{list-assn unat-lit-assn})^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow_a \textit{nat-lit-list-hm-assn}$›
unfolding *extract-atms-cls-i-def*
by *sepref*

declare *extract-atms-cls-imp.refine*[*sepref-fr-rules*]

sepref-definition *extract-atms-clss-imp*
is ‹*uncurry extract-atms-clss-i*›
:: ‹$(\textit{list-assn}\ (\textit{list-assn unat-lit-assn}))^k *_a \textit{nat-lit-list-hm-assn}^d \rightarrow_a \textit{nat-lit-list-hm-assn}$›
unfolding *extract-atms-clss-i-def*
by *sepref*

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
‹(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* ∘∘ *extract-atms-clss*))
∈ [λ(*a*, *b*). ∀ *C*∈*set a*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint-max*]$_a$
(*list-assn* (*list-assn unat-lit-assn*))^k $*_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*›
using *extract-atms-clss-imp.refine*[*FCOMP extract-atms-clss-i-extract-atms-clss*[*unfolded convert-fref*]]
.

sepref-definition *extract-atms-clss-imp-empty-assn*
is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *isasat-atms-ext-rel-assn*›
unfolding *extract-atms-clss-imp-empty-rel-def*
array-fold-custom-replicate
supply [[*goals-limit=1*]]
apply (*rewrite at* ‹(-, -, ¤)› *arl.fold-custom-empty*)
apply (*rewrite* **in** ‹(-, -, ¤)› *annotate-assn*[**where** *A*=‹*arl-assn uint32-nat-assn*›])
apply (*rewrite* **in** ‹(¤, -, -)› *zero-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, ¤, -)› *zero-uint32-nat-def*[*symmetric*])
by *sepref*

lemma *extract-atms-clss-imp-empty-assn*[*sepref-fr-rules*]:
‹(*uncurry0 extract-atms-clss-imp-empty-assn*, *uncurry0* (*RETURN op-extract-list-empty*))
∈ *unit-assn*^{k} $\rightarrow_a$ *nat-lit-list-hm-assn*›
using *extract-atms-clss-imp-empty-assn.refine*[*FCOMP extract-atms-clss-imp-empty-rel*
[*unfolded convert-fref uncurry0-def*[*symmetric*]]] **.**

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*, []))›
by (*auto simp*: *extract-atms-clss-imp-empty-rel-def*)

Full Initialisation

sepref-definition *rewatch-heur-st-code*
is ‹(*rewatch-heur-st*)›
:: ‹*isasat-init-unbounded-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def*
isasat-init-unbounded-assn-def
by *sepref*
find-theorems *nfoldli WHILET*

sepref-definition *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-init-assn^d → *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def rewatch-heur-st-fast-pre-def*
isasat-init-assn-def rewatch-heur-st-fast-def
by *sepref*

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-assn* (*list-assn unat-lit-assn*))^k $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$
isasat-init-unbounded-assn›
supply [[*goals-limit=1*]]
unfolding *init-dt-wl-heur-def PR-CONST-def init-dt-step-wl-heur-unb-def*[*symmetric*] *if-True*
init-dt-wl-heur-unb-def
by *sepref*

sepref-definition *init-dt-wl-heur-code-b*
is ‹*uncurry* (*init-dt-wl-heur-b*)›
:: ‹(*list-assn* (*list-assn unat-lit-assn*))^k $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$
isasat-init-assn›
supply [[*goals-limit=1*]]
unfolding *init-dt-wl-heur-def PR-CONST-def init-dt-step-wl-heur-b-def*[*symmetric*] *if-True*
init-dt-wl-heur-b-def
by *sepref*

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-assn* (*list-assn unat-lit-assn*))^k $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$
isasat-init-unbounded-assn›
supply [[*goals-limit=1*]]
unfolding *init-dt-wl-heur-full-def PR-CONST-def init-dt-wl-heur-full-unb-def*
init-dt-wl-heur-unb-def[*symmetric*]
by *sepref*

declare *init-dt-wl-heur-full-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *extract-lits-sorted-code*
is ‹*extract-lits-sorted*›
:: ‹[λ(*xs*, *n*, *vars*). ($\forall x \in \#mset\ vars.\ x < length\ xs$)]$_a$
isasat-atms-ext-rel-assn^d $\rightarrow$
arl-assn uint32-nat-assn $*a$ *uint32-nat-assn*›
unfolding *extract-lits-sorted-def*
supply [[*goals-limit = 1*]]
supply *mset-eq-setD*[*dest*] *mset-eq-length*[*dest*]
by *sepref*

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-lit-list-hm-assn*^{d} $\rightarrow_a$ *lits-with-max-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −

have *H*: ‹*?c*
∈ [*comp-PRE isasat-atms-ext-rel* (λ-. *True*)
(λ- (*xs*, *n*, *vars*). ∀ *x*∈#*mset vars*. *x* < *length xs*) (λ-. *True*)]$_a$
hrp-comp (*isasat-atms-ext-rel-assn*^{d}) *isasat-atms-ext-rel* → *lits-with-max-assn*›
(**is** ‹- ∈ [*?pre'*]$_a$ *?im'* → *?f'*›)
using *hfref-compI-PRE-aux*[*OF extract-lits-sorted-code.refine*
extract-lits-sorted-mset-set[*unfolded convert-fref*]] **.**
have *pre*: ‹*?pre' x*› **if** ‹*?pre x*› **for** *x*
using *that* **by** (*auto simp*: *comp-PRE-def isasat-atms-ext-rel-def init-valid-rep-def*)
have *im*: ‹*?im'* = *?im*›
unfolding *prod-hrp-comp hrp-comp-dest hrp-comp-keep* **by** *simp*
show *?thesis*
apply (*rule hfref-weaken-pre*[*OF*])
defer
using *H* **unfolding** *im PR-CONST-def* **apply** *assumption*
using *pre* **..**
qed

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*›
unfolding *op-arl32-replicate-def op-list-replicate-def* **..**

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-nat-assn*^{k} $*_a$ R^k $\rightarrow_a$ *arl32-assn R*›
proof −
obtain *R'* **where**
R'[*symmetric*]: ‹*R'* = *the-pure R*› **and**
R-R': ‹*R* = *pure R'*›
using *assms* **by** *fastforce*
have [*simp*]: ‹*pure R' b bi* = ↑((*bi*, *b*) ∈ *R'*)› **for** *b bi*
by (*auto simp*: *pure-def*)
have [*simp*]: ‹*min a* (*max a 16*) = *a*› ‹*16* ≤ *uint32-max*› **for** *a* :: *nat*
by (*auto simp*: *uint32-max-def*)
show *?thesis*
using *assms* **unfolding** *op-arl32-replicate-def*
by *sepref-to-hoare*
(*sep-auto simp*: *arl32-replicate-def arl32-assn-def hr-comp-def R' R-R' list-rel-def*
is-array-list32-def minimum-capacity-def uint32-nat-rel-def br-def nat-of-uint32-le-uint32-max
intro!: *list-all2-replicate*)
qed

definition *INITIAL-OUTL-SIZE* :: ‹*nat*› **where**
[*simp*]: ‹*INITIAL-OUTL-SIZE* = *160*›

lemma [*sepref-fr-rules*]:
‹(*uncurry0* (*return 160*), *uncurry0* (*RETURN INITIAL-OUTL-SIZE*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
by (*sepref-to-hoare*) (*sep-auto simp*: *INITIAL-OUTL-SIZE-def uint32-nat-rel-def br-def*)

sepref-definition *finalise-init-code′*
is ‹*uncurry finalise-init-code*›
:: ‹[λ(-, *S*). *length* (*get-clauses-wl-heur-init S*) ≤ *uint64-max*]$_a$
opts-assn^d $*_a$ *isasat-init-assn*^{d} → *isasat-bounded-assn*›
supply *zero-uin64-hnr*[*sepref-fr-rules*] [[*goals-limit=1*]]
Pos-unat-lit-assn′[*sepref-fr-rules*] *uint-max-def*[*simp*] *op-arl-replicate-def*[*simp*]
unfolding *finalise-init-code-def isasat-init-assn-def isasat-bounded-assn-def*
arl32-fold-custom-replicate two-uint32-def[*symmetric*] *INITIAL-OUTL-SIZE-def*[*symmetric*]
one-uint32-nat-def[*symmetric*]
apply (*rewrite at* ‹(-, ⌑, -)› *arl64.fold-custom-empty*)
apply (*rewrite* **in** ‹*op-arl64-empty*› *annotate-assn*[**where** *A*=‹*vdom-fast-assn*›])
apply (*rewrite at* ‹(-, ⌑)› *arl64.fold-custom-empty*)
apply (*rewrite* **in** ‹*op-arl64-empty*› *annotate-assn*[**where** *A*=‹*arena-fast-assn*›])
by *sepref*

sepref-definition *finalise-init-code-unb*
is ‹*uncurry finalise-init-code*›
:: ‹*opts-assn*^{d} $*_a$ *isasat-init-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
supply *zero-uin64-hnr*[*sepref-fr-rules*] [[*goals-limit=1*]]
Pos-unat-lit-assn′[*sepref-fr-rules*] *uint-max-def*[*simp*] *op-arl-replicate-def*[*simp*]
unfolding *finalise-init-code-def isasat-init-unbounded-assn-def isasat-unbounded-assn-def*
arl32-fold-custom-replicate two-uint32-def[*symmetric*] *INITIAL-OUTL-SIZE-def*[*symmetric*]
one-uint32-nat-def[*symmetric*] *zero-uint64-nat-def*
apply (*rewrite at* ‹(-, ⌑, -)› *arl.fold-custom-empty*)
apply (*rewrite* **in** ‹*op-arl-empty*› *annotate-assn*[**where** *A*=‹*vdom-assn*›])
apply (*rewrite at* ‹(-, ⌑)› *arl.fold-custom-empty*)
apply (*rewrite* **in** ‹*op-arl-empty*› *annotate-assn*[**where** *A*=‹*arena-assn*›])
by *sepref*

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-assn^k $\rightarrow_a$ (*arlO-assn clause-ll-assn*)›
by *sepref-to-hoare* (*sep-auto simp*: *init-rll-def hr-comp-def clauses-ll-assn-def init-aa-def*)

lemma *init-aa′-alt-def*: ‹*RETURN o init-aa′* = (λ*n*. *RETURN op-arl-empty*)›
by (*auto simp*: *init-aa′-def op-arl-empty-def*)

sepref-definition *init-aa′-code*
is ‹*RETURN o init-aa′*›
:: ‹*nat-assn*^{k} $\rightarrow_a$ *arl-assn* (*clause-status-assn* ∗*a* *uint32-nat-assn* ∗*a* *uint32-nat-assn*)›
unfolding *init-aa′-alt-def*
by *sepref*

declare *init-aa′-code.refine*[*sepref-fr-rules*]

sepref-register *initialise-VMTF*

sepref-definition *init-trail-D-code*
is ‹*uncurry2 init-trail-D*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a trail\text{-}pol\text{-}assn$›
unfolding *init-trail-D-def PR-CONST-def*
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *arl.fold-custom-empty*)
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *annotate-assn*[**where** *A*=‹*arl-assn unat-lit-assn*›])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in -*› *IICF-Array-List.arl.fold-custom-empty*)
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in -*› *annotate-assn*[**where** *A*=‹*arl-assn uint32-nat-assn*›])

apply (*rewrite* **in** ‹*let - = -;- = ⌑ in -*› *annotate-assn*[**where** *A*=‹*array-assn* (*tri-bool-assn*)›])
apply (*rewrite* **in** ‹*let - = -;- = -;- = ⌑ in -*› *annotate-assn*[**where** *A*=‹*array-assn uint32-nat-assn*›])
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
supply [[*goals-limit = 1*]]
by *sepref*

declare *init-trail-D-code.refine*[*sepref-fr-rules*]

sepref-definition *init-trail-D-fast-code*
is ‹*uncurry2 init-trail-D-fast*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a trail\text{-}pol\text{-}fast\text{-}assn$›
unfolding *init-trail-D-def PR-CONST-def init-trail-D-fast-def*
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *arl32.fold-custom-empty*)
apply (*rewrite* **in** ‹*let - = ⌑ in -*› *annotate-assn*[**where** *A*=‹*arl32-assn unat-lit-assn*›])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in -*› *arl32.fold-custom-empty*)
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in -*› *annotate-assn*[**where** *A*=‹*arl32-assn uint32-nat-assn*›])

apply (*rewrite* **in** ‹*let - = -;- = ⌑ in -*› *annotate-assn*[**where** *A*=‹*array-assn* (*tri-bool-assn*)›])
apply (*rewrite* **in** ‹*let - = -;- = -;- = ⌑ in -*› *annotate-assn*[**where** *A*=‹*array-assn uint32-nat-assn*›])
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in -*› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = op-array-replicate - ⌑ in -*› *one-uint64-nat-def*[*symmetric*])
supply [[*goals-limit = 1*]]
by *sepref*

declare *init-trail-D-fast-code.refine*[*sepref-fr-rules*]

sepref-definition *init-state-wl-D′-code*
is ‹*init-state-wl-D′*›
:: ‹$(arl\text{-}assn\ uint32\text{-}assn * a\ uint32\text{-}assn)^d \rightarrow_a isasat\text{-}init\text{-}assn$›
unfolding *init-state-wl-D′-def PR-CONST-def init-trail-D-fast-def*[*symmetric*] *isasat-init-assn-def*
apply (*rewrite at* ‹*let - = (-, ⌑) in -*› *arl32.fold-custom-empty*)
apply (*rewrite at* ‹*let - = ⌑ in -*› *init-lrl-def*[*symmetric*])
unfolding *array-fold-custom-replicate init-lrl64-def*[*symmetric*]
apply (*rewrite at* ‹*let - = ⌑ in let - = (True, -, -) in -*› *arl64.fold-custom-empty*)
apply (*rewrite at* ‹*let - = ⌑ in -*› *annotate-assn*[**where** *A*=‹*arena-fast-assn*›])
apply (*rewrite at* ‹*let -= -; -= ⌑ in -*› *annotate-assn*[**where** *A*=‹*watchlist-fast-assn*›])
apply (*rewrite at* ‹*let -= ⌑ in RETURN -*› *arl64.fold-custom-empty*)
supply [[*goals-limit = 1*]]
by *sepref*

sepref-definition *init-state-wl-D′-code-unb*

is ‹*init-state-wl-D'*›
:: ‹(*arl-assn uint32-assn* $*a$ *uint32-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*›
unfolding *init-state-wl-D'-def PR-CONST-def*
apply (*rewrite at* ‹*let* - = (-, ¤) *in* -› *arl32.fold-custom-empty*)
apply (*rewrite at* ‹*let* - = ¤ *in* -› *init-lrl-def*[*symmetric*])
unfolding *array-fold-custom-replicate*
apply (*rewrite at* ‹*let* - = ¤ *in let* - = (*True*, -, -) *in* -› *IICF-Array-List.arl.fold-custom-empty*)
apply (*rewrite at* ‹*let* - = ¤ *in* -› *annotate-assn*[**where** *A*=‹*arena-assn*›])
apply (*rewrite at* ‹*let* -= -; -= ¤ *in* -› *annotate-assn*[**where** *A*=‹*watchlist-assn*›])
apply (*rewrite at* ‹*let* -= ¤ *in RETURN* -› *IICF-Array-List.arl.fold-custom-empty*)
supply [[*goals-limit* = *1*]]
by *sepref*

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-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
unfolding *to-init-state-code-def*
by (*rule id-ref*)

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* ∈# *D*› **and** *M-nempty*: ‹*M* ≠ []›
shows ‹*get-maximum-level-remove M D L* = *count-decided M* ⟷
(*count-decided M* = *0* ∨ *card-max-lvl M D* > *1*)›
(**is** ‹*?max* ⟷ *?count*›)
proof
assume *H*: *?max*
let *?D* = ‹*remove1-mset L D*›
have [*simp*]: ‹*get-level M L* = *count-decided M*›
using *M-nempty L* **by** (*cases M*) *auto*
define *MD* **where** ‹*MD* ≡ {#*L* ∈# *D*. *get-level M L* = *count-decided M*#}›
show *?count*
proof (*cases* ‹*?D* = {#}›)
case *True*
then show *?thesis*
using *LD H* **by** (*auto dest*!: *multi-member-split simp*: *get-maximum-level-remove-def*)
next
case *False*
then obtain *L'* **where**
‹*get-level M L'* = *get-maximum-level-remove M D L*› **and** *L'-D*: ‹*L'* ∈# *?D*›

using *get-maximum-level-exists-lit-of-max-level*[*of* ‹*remove1-mset L D*›]
unfolding *get-maximum-level-remove-def* **by** *blast*
then have ‹$L' \in\#$ {#$L \in\#$ *D*. *get-level M L* = *count-decided M*#}›
using *H* **by** (*auto dest*: *in-diffD simp*: *get-maximum-level-remove-def*)
moreover have ‹$L \in\#$ {#$L \in\#$ *D*. *get-level M L* = *count-decided M*#}›
using *LD* **by** *auto*
ultimately have ‹{#*L*, *L'*#} ⊆# *MD*›
using *L'-D LD* **by** (*cases* ‹*L* = *L'*›)
(*auto dest*!: *multi-member-split simp*: *MD-def add-mset-eq-add-mset*)
from *size-mset-mono*[*OF this*] **show** *?thesis*
unfolding *card-max-lvl-def H MD-def*[*symmetric*]
by *auto*
qed
next
let *?D* = ‹*remove1-mset L D*›
have [*simp*]: ‹*get-level M L* = *count-decided M*›
using *M-nempty L* **by** (*cases M*) *auto*
define *MD* **where** ‹*MD* ≡ {#$L \in\#$ *D*. *get-level M L* = *count-decided M*#}›
have *L-MD*: ‹$L \in\#$ *MD*›
using *LD* **unfolding** *MD-def* **by** *auto*
assume *?count*
then consider
(*lev-0*) ‹*count-decided M* = *0*› |
(*count*) ‹*card-max-lvl M D* > *1*›
by (*cases* ‹*D* ≠ {#*L*#}›) *auto*
then show *?max*
proof *cases*
case *lev-0*
then show *?thesis*
using *count-decided-ge-get-maximum-level*[*of M ?D*]
by (*auto simp*: *get-maximum-level-remove-def*)
next
case *count*
then obtain *L'* **where**
‹$L' \in\#$ *MD*› **and**
LL': ‹{#*L*, *L'*#} ⊆# *MD*›
using *L-MD*
unfolding *get-maximum-level-remove-def card-max-lvl-def MD-def*[*symmetric*]
by (*force simp*: *nonempty-has-size*[*symmetric*]
dest!: *multi-member-split multi-nonempty-split*)
then have ‹*get-level M L'* = *count-decided M*›
unfolding *MD-def* **by** *auto*
moreover have ‹$L' \in\#$ *remove1-mset L D*›
proof −
have ‹{#*L*, *L'*#} ⊆# *D*›
using *LL'* **unfolding** *MD-def*
by (*meson multiset-filter-subset subset-mset.dual-order.trans*)
then show *?thesis*
by (*metis* (*no-types*) *LD insert-DiffM mset-subset-eq-add-mset-cancel single-subset-iff*)
qed
ultimately have ‹*get-maximum-level M* (*remove1-mset L D*) ≥ *count-decided M*›
using *get-maximum-level-ge-get-level*[*of L' ?D M*]
by *simp*
then show *?thesis*
using *count-decided-ge-get-maximum-level*[*of M ?D*]
by (*auto simp*: *get-maximum-level-remove-def*)

qed
qed

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* $\rightarrow$ ⟨*bool-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *get-maximum-level-remove-count-max-lvls*[*of* ‹*fst x*› ‹*get-trail-wl* (*snd y*)›
‹*the* (*get-conflict-wl* (*snd y*))›]
by (*cases x*)
(*auto simp*: *count-decided-st-def counts-maximum-level-def twl-st-heur-conflict-ana-def*
maximum-level-removed-eq-count-dec-heur-def maximum-level-removed-eq-count-dec-def
maximum-level-removed-eq-count-dec-pre-def)
done

lemma *get-trail-wl-heur-def*: ‹*get-trail-wl-heur* = (λ(*M*, *S*). *M*)›
by (*intro ext*, *rename-tac S*, *case-tac S*) *auto*

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* $\rightarrow$ ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
by (*rename-tac x y*; *case-tac x*; *case-tac y*; *case-tac* ‹*hd* (*fst y*)›; *case-tac* ‹*fst y*›;
case-tac ‹*fst* (*fst x*)› *rule*: *rev-cases*)
(*auto simp*: *twl-st-heur-conflict-ana-def lit-and-ann-of-propagated-st-heur-def*
lit-and-ann-of-propagated-st-def trail-pol-def ann-lits-split-reasons-def)

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*›

using *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*[*THEN fref-to-Down-unRET*, *of y x*]
by *auto*

definition *tl-state-wl-heur-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*tl-state-wl-heur-pre* =
$(\lambda(M, N, D, WS, Q, ((A, m, \textit{fst-As}, \textit{lst-As}, \textit{next-search}), \textit{to-remove}), \varphi, \text{-}).\ \textit{fst}\ M \neq [] \wedge$
tl-trailt-tr-pre $M \wedge$
vmtf-unset-pre (*atm-of* (*last* (*fst* M))) ((A, m, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\wedge$
atm-of (*last* (*fst* M)) $<$ *length* $\varphi \wedge$
atm-of (*last* (*fst* M)) $<$ *length* $A \wedge$
(*next-search* $\neq$ *None* $\longrightarrow$ *the next-search* $<$ *length* A))›

definition *tl-state-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*tl-state-wl-heur* = $(\lambda(M, N, D, WS, Q, \textit{vmtf}, \varphi, \textit{clvls})$.
(*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* = $(\lambda(M, N, D, WS, Q, \textit{vmtf}, \varphi, \textit{clvls})$.
(*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*)))›
by (*auto simp*: *tl-state-wl-heur-def Let-def*)

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* $a \neq 0$ *then card-max-lvl* a y + *1*
else card-max-lvl a y)›
proof −
have [*simp*]: ‹*count-decided* a = *0* ⟹ *get-level* a L = *0*› **for** L
by (*simp add*: *count-decided-0-iff*)
have [*simp*]: ‹*lit-of* L ∉# A ⟹
− *lit-of* L ∉# A ⟹
{#*La* ∈# A. *La* ≠ *lit-of* L ∧ *La* ≠ − *lit-of* L ⟶ *get-level* a *La* = *b*#} =
{#*La* ∈# A. *get-level* a *La* = *b*#}› **for** A b
apply (*rule filter-mset-cong*)
apply (*rule refl*)
by *auto*
show *?thesis*
using *assms* **by** (*auto simp*: *card-max-lvl-def get-level-cons-if tautology-add-mset*
atm-of-eq-atm-of
dest!: *multi-member-split*)
qed

lemma *card-max-lvl-tl*:
assumes ‹$a \neq []$› ‹*distinct-mset* y›‹¬*tautology* y› ‹¬*is-decided* (*hd* a)› ‹*no-dup* a›
‹*count-decided* $a \neq 0$›
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)›
using *assms* **by** (*cases* a) (*auto simp*: *card-max-lvl-Cons*)

definition *tl-state-wl-pre* **where**
‹*tl-state-wl-pre* S ⟷ *get-trail-wl* $S \neq [] \wedge$
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st* S) (*get-trail-wl* S) $\wedge$

$(lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ S))) \notin\#\ the\ (get\text{-}conflict\text{-}wl\ S) \wedge$
$-(lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ S))) \notin\#\ the\ (get\text{-}conflict\text{-}wl\ S) \wedge$
$\neg tautology\ (the\ (get\text{-}conflict\text{-}wl\ S)) \wedge$
$distinct\text{-}mset\ (the\ (get\text{-}conflict\text{-}wl\ S)) \wedge$
$\neg is\text{-}decided\ (hd\ (get\text{-}trail\text{-}wl\ S)) \wedge$
$count\text{-}decided\ (get\text{-}trail\text{-}wl\ S) > 0$›

lemma *tl-state-out-learned*:
‹$lit\text{-}of\ (hd\ a) \notin\#\ the\ at \Longrightarrow$
$-\ lit\text{-}of\ (hd\ a) \notin\#\ the\ at \Longrightarrow$
$\neg\ is\text{-}decided\ (hd\ a) \Longrightarrow$
$out\text{-}learned\ (tl\ a)\ at\ an \longleftrightarrow out\text{-}learned\ a\ at\ an$›
by (*cases a*) (*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of*
intro!: *filter-mset-cong*)

lemma *tl-state-wl-heur-tl-state-wl*:
‹(*RETURN o tl-state-wl-heur*, *RETURN o tl-state-wl*) ∈
$[tl\text{-}state\text{-}wl\text{-}pre]_f$ *twl-st-heur-conflict-ana* → ⟨*twl-st-heur-conflict-ana*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*auto simp*: *twl-st-heur-conflict-ana-def tl-state-wl-heur-def tl-state-wl-def vmtf-unset-vmtf-tl*
$in\text{-}\mathcal{L}_{all}$*-atm-of-in-atms-of-iff phase-saving-def counts-maximum-level-def*
card-max-lvl-tl tl-state-wl-pre-def tl-state-out-learned neq-Nil-conv
literals-are-in-$\mathcal{L}_{in}$*-trail-Cons all-atms-def*[*symmetric*] *card-max-lvl-Cons*
dest: *no-dup-tlD*
intro!: *tl-trail-tr*[*THEN fref-to-Down-unRET*] *isa-vmtf-tl-isa-vmtf*
simp: *lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*]
intro: *tl-state-out-learned*[*THEN iffD2*, *of* ‹*Cons - -*›, *simplified*])
apply (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
apply (*auto simp*: *lit-of-hd-trail-def*)[*3*]
apply (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
apply (*auto simp*: *lit-of-hd-trail-def*)[*3*]
done

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-used arena C*) *C*›
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
apply *clarify*
apply (*rule-tac x=N* **in** *exI*)
apply (*rule-tac x=vdom* **in** *exI*)
by (*auto simp*: *arena-act-pre-def*
simp: *valid-arena-mark-used*)

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*);

$((b, (n, xs)), clvls, lbd, outl) \leftarrow$
  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* $(n, xs) \wedge clvls \geq 1$);
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*) $\in\#$ *y* $\Longrightarrow$ *is-proped* (*hd M*) $\Longrightarrow$
  card-max-lvl M (*remove1-mset* ($-$*lit-of* (*hd M*)) *y*) = *card-max-lvl M y* $-$ *1*›
by (*cases M*) (*auto dest*!: *multi-member-split simp*: *card-max-lvl-add-mset*)

lemma *update-confl-tl-wl-heur-state-helper*:
‹(L, C) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) $\Longrightarrow$ *get-trail-wl S* $\neq$ [] $\Longrightarrow$
is-proped (*hd* (*get-trail-wl S*)) $\Longrightarrow$ *L* = *lit-of* (*hd* (*get-trail-wl S*))›
by (*cases S*; *cases* ‹*hd* (*get-trail-wl S*)›) *auto*

lemma (**in** $-$) *not-ge-Suc0*: ‹$\neg$*Suc 0* $\leq n \longleftrightarrow n = 0$›
by *auto*

definition *update-confl-tl-wl-pre* **where**
‹*update-confl-tl-wl-pre* = (λ((*C*, *L*), *S*).
  C $\in\#$ *dom-m* (*get-clauses-wl S*) $\wedge$
  get-conflict-wl S $\neq$ *None* $\wedge$ *get-trail-wl S* $\neq$ [] $\wedge$
  $-$ *L* $\in\#$ *the* (*get-conflict-wl S*) $\wedge$
  (L, C) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) $\wedge$
  L $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$
  is-proped (*hd* (*get-trail-wl S*)) $\wedge$
  C > *0* $\wedge$
  card-max-lvl (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) $\geq$ *1* $\wedge$
  distinct-mset (*the* (*get-conflict-wl S*)) $\wedge$
  $-$ *L* $\notin$ *set* (*get-clauses-wl S* $\propto$ *C*) $\wedge$
  (*length* (*get-clauses-wl S* $\propto$ *C*) > *2* $\longrightarrow$
    L $\notin$ *set* (*tl* (*get-clauses-wl S* $\propto$ *C*)) $\wedge$
    get-clauses-wl S $\propto$ *C* ! *0* = *L*) $\wedge$
  L $\in$ *set* (*watched-l* (*get-clauses-wl S* $\propto$ *C*)) $\wedge$
  distinct (*get-clauses-wl S* $\propto$ *C*) $\wedge$
  $\neg$*tautology* (*the* (*get-conflict-wl S*)) $\wedge$
  $\neg$*tautology* (*mset* (*get-clauses-wl S* $\propto$ *C*)) $\wedge$
  $\neg$*tautology* (*remove1-mset L* (*remove1-mset* ($-$ *L*)
    ((*the* (*get-conflict-wl S*) $\cup\#$ *mset* (*get-clauses-wl S* $\propto$ *C*))))) $\wedge$
  count-decided (*get-trail-wl S*) > *0* $\wedge$
  literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*)) $\wedge$
  literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* $\wedge$
  literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)
)›

lemma (**in** $-$)*out-learned-add-mset-highest-level*:

⟨L = *lit-of* (*hd* M) $\Longrightarrow$ *out-learned* M (*Some* (*add-mset* (− L) A)) *outl* $\longleftrightarrow$
out-learned M (*Some* A) *outl*⟩
by (*cases* M)
(*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of count-decided-ge-get-level*
uminus-lit-swap cong: *if-cong*
intro!: *filter-mset-cong2*)

lemma (**in** −)*out-learned-tl-Some-notin*:
⟨*is-proped* (*hd* M) $\Longrightarrow$ *lit-of* (*hd* M) ∉# C $\Longrightarrow$ −*lit-of* (*hd* M) ∉# C $\Longrightarrow$
out-learned M (*Some* C) *outl* $\longleftrightarrow$ *out-learned* (*tl* M) (*Some* C) *outl*⟩
by (*cases* M) (*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of*
intro!: *filter-mset-cong2*)

abbreviation *twl-st-heur-conflict-ana'* :: ⟨*nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*⟩ **where**
⟨*twl-st-heur-conflict-ana'* r ≡ {(S, T). (S, T) ∈ *twl-st-heur-conflict-ana* ∧
length (*get-clauses-wl-heur* S) = r}⟩

lemma *literals-are-in-$\mathcal{L}_{in}$-mm-all-atms-self*[*simp*]:
⟨*literals-are-in-$\mathcal{L}_{in}$-mm* (*all-atms ca NUE*) {#*mset* (*fst* x). x ∈# *ran-m ca*#}⟩
by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-mm-def in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*
all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff)

lemma *update-confl-tl-wl-heur-update-confl-tl-wl*:
⟨(*uncurry2* (*update-confl-tl-wl-heur*), *uncurry2* (*RETURN ooo update-confl-tl-wl*)) ∈
[*update-confl-tl-wl-pre*]$_f$
nat-rel $\times_f$ *Id* $\times_f$ *twl-st-heur-conflict-ana'* r → ⟨*bool-rel* $\times_f$ *twl-st-heur-conflict-ana'* r⟩*nres-rel*⟩
proof −

have H: ⟨*isa-resolve-merge-conflict-gt2 ba c a* (*aa*, *ab*, *bb*) *i k ag*
≤ *SPEC*
(λx. (*case* x *of*
(x, *xa*) ⇒
(*case* x *of*
(*bb*, n, *xs*) ⇒
λ(*clvls*, *lbd*, *outl*). *do* {
- ← *ASSERT* (*curry lookup-conflict-remove1-pre b* (n, *xs*) ∧
1 ≤ *clvls*);
let (n, *xs*) = *lookup-conflict-remove1 b* (n, *xs*);
ASSERT (*arena-act-pre c a*);
let c = *mark-used c a*;
ASSERT (*arena-act-pre c a*);
let c = *arena-incr-act c a*;
ASSERT(*vmtf-unset-pre* (*atm-of b*) *ivmtf*);
ASSERT(*tl-trailt-tr-pre ba*);
RETURN
(*False*, *tl-trailt-tr ba*, c, (*bb*, n, *xs*), e, f, *isa-vmtf-unset* (*atm-of b*) *ivmtf*,
h, *fast-minus clvls one-uint32-nat*, j,
lbd, *outl*, (*ah*, *ai*, *aj*, *be*), *ak*, *al*, *am*, *an*, *bf*)
})
xa)
≤ ⇓ (*bool-rel* $\times_f$ *twl-st-heur-conflict-ana'* r)
(*RETURN*
(*let* D = *resolve-cls-wl'* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *ao bg*
in (*False*, *tl baa*, *ca*, *Some* D, *ea*, *fa*, *ga*, *ha*))))⟩
if

inv: ‹*update-confl-tl-wl-pre* ((*ao*, *bg*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
rel: ‹(((*a*, *b*), *ba*, *c*, (*aa*, *ab*, *bb*), *e*, *f*, *ivmtf*, *h*, *i*, *j*,
k, *ag*, (*ah*, *ai*, *aj*, *be*), *ak*, *al*, *am*, *an*, *bf*),
(*ao*, *bg*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)
∈ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana′ r*› **and**
CLS[*simp*]: ‹*CLS* = ((*ao*, *bg*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
‹*CLS′* = ((*a*, *b*), *ba*, *c*, (*aa*, *ab*, *bb*), *e*, *f*, *ivmtf*, *h*, *i*, *j*, *k*,
ag, (*ah*, *ai*, *aj*, *be*), *ak*, *al*, *am*, *an*, *bf*)› **and**
le2: ‹*arena-length c a* ≠ *2*›
for *a b ba c aa ab bb e f ac ad ae af bc bd h i j k ag ah ai aj be ak al am an*
bf ao bg baa ca da ea fa ga ha CLS CLS′ ivmtf
proof −
let *?𝒜* = ‹*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)›
have *le2*: ‹*length* (*ca* ∝ *ao*) > *2*›
using *arena-lifting*(*19*)[*of c ca* ‹*set an*› *ao*]
using *rel inv le2* **unfolding** *CLS update-confl-tl-wl-pre-def prod.case*
get-clauses-wl.simps
by (*auto simp*: *twl-st-heur-conflict-ana-def arena-lifting*
simp del: *arena-lifting*(*19*))

have
ao: ‹*ao* ∈# *dom-m* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
conf: ‹*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ≠ *None*› **and**
nempty: ‹*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ≠ []› **and**
uL-D: ‹− *bg* ∈# *the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
L-M: ‹(*bg*, *ao*) = *lit-and-ann-of-propagated*
(*hd* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
bg-D0: ‹*bg* ∈# $\mathcal{L}_{all}$ *?𝒜*› **and**
proped: ‹*is-proped* (*hd* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
‹*0* < *ao*› **and**
card-max-lvl: ‹*1* ≤ *card-max-lvl* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))
(*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
dist-D: ‹*distinct-mset* (*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
uL-NC: ‹− *bg* ∉ *set* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ∝ *ao*)› **and**
L-NC: ‹*bg* ∉ *set* (*tl* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ∝ *ao*))› **and**
bg-hd: ‹*ca* ∝ *ao* ! *0* = *bg*› **and**
dist-NC: ‹*distinct* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ∝ *ao*)› **and**
tauto-D: ‹¬ *tautology* (*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
tauto-NC: ‹¬ *tautology* (*mset* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ∝ *ao*))› **and**
tauto-NC-D: ‹¬ *tautology*
(*remove1-mset bg* (*remove1-mset* (− *bg*)
(*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)) ∪#
mset (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) ∝ *ao*))))› **and**
count-dec-ge: ‹*0* < *count-decided* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
lits-confl: ‹*literals-are-in*-$\mathcal{L}_{in}$ *?𝒜* (*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
lits: ‹*literals-are*-$\mathcal{L}_{in}$ *?𝒜* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
lits-trail: ‹*literals-are-in*-$\mathcal{L}_{in}$*-trail ?𝒜* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))›
using *inv le2* **unfolding** *CLS update-confl-tl-wl-pre-def prod.case*
get-clauses-wl.simps
by *blast*+
have
n-d: ‹*no-dup baa*› **and**
outl: ‹*out-learned baa da ag*› **and**
i: ‹*i* ∈ *counts-maximum-level baa da*›
using *rel* **unfolding** *twl-st-heur-conflict-ana-def*
by *auto*

have
  [*simp*]: ⟨$a = ao$⟩ **and**
  [*simp*]: ⟨$b = bg$⟩ **and**
  n-d: ⟨*no-dup baa*⟩ **and**
  arena: ⟨*valid-arena c ca* (*set an*)⟩ **and**
  ocr: ⟨((*aa*, *ab*, *bb*), *da*) $\in$ *option-lookup-clause-rel ?$\mathcal{A}$*⟩ **and**
  trail: ⟨(*ba*, *baa*) $\in$ *trail-pol ?$\mathcal{A}$*⟩ **and**
  bounded: ⟨*isasat-input-bounded ?$\mathcal{A}$*⟩
  using *rel* **by** (*auto simp*: *CLS twl-st-heur-conflict-ana-def all-atms-def*[*symmetric*])
have *lookup-remove1-uminus*:
  ⟨*lookup-conflict-remove1* ($-bg$) A = *lookup-conflict-remove1 bg A*⟩ **for** A
  by (*auto simp*: *lookup-conflict-remove1-def*)
have [*simp*]: ⟨*lit-of* (*hd baa*) = *bg*⟩ **and** *hd-M-L-C*: ⟨*hd baa* = *Propagated bg ao*⟩
  using *L-M nempty proped* **by** (*cases baa*; *cases* ⟨*hd baa*⟩; *auto*; *fail*)+
have *bg-D*[*simp*]: ⟨*bg* $\notin\#$ *the da*⟩
  using *uL-D tauto-D* **by** (*auto simp*: *tautology-add-mset add-mset-eq-add-mset*
  dest!: *multi-member-split*)
have *bg-$\mathcal{A}$*: ⟨*bg* $\in\#$ $\mathcal{L}_{all}$ *?$\mathcal{A}$*⟩
  using ⟨*lit-of* (*hd baa*) = *bg*⟩ *lits-trail uL-D nempty*
  by (*cases baa*)
    (*auto simp del*: ⟨*lit-of* (*hd baa*) = *bg*⟩ *simp*: *literals-are-in-$\mathcal{L}_{in}$-trail-Cons*)

have [*simp*]: ⟨*bg* $\notin$ *set* (*tl* (*ca* $\propto$ *ao*))⟩
  using *L-NC*
  by (*auto simp*: *resolve-cls-wl′-def split*: *if-splits*)
have *mset-ge0-iff*: $0 <$ *size* $M \longleftrightarrow M \neq \{\#\}$ **for** M
  by (*cases* M) *auto*
have *no-dup*: ⟨$L \in$ *set* (*tl* (*ca* $\propto$ *ao*)) $\Longrightarrow -L \in\#$ *the da* $\Longrightarrow$ *False*⟩ **for** L
  using *tauto-NC-D tauto-NC* ⟨*bg* $\notin$ *set* (*tl* (*ca* $\propto$ *ao*))⟩ *bg-hd le2*
  by (*cases* ⟨$- L \in\#$ *mset* (*tl* (*ca* $\propto$ *ao*))⟩; *cases* ⟨*bg* = L⟩; *cases* ⟨*ca* $\propto$ *ao*⟩)
    (*auto dest*!: *multi-member-split*
    simp: *sup-union-left2 add-mset-remove-trivial-If*
      tautology-add-mset add-mset-eq-add-mset
      add-mset-remove-trivial-eq remove1-mset-union-distrib
dest: *in-set-tlD tautology-minus*[*of L*]
    split: *if-splits*)

have *size-union-ge1*: ⟨*Suc 0* $\leq$ *size* $A \Longrightarrow$ *Suc 0* $\leq$ *size* ($A \cup\# B$)⟩ **for** A B
  apply (*cases* A)
  apply (*simp*; *fail*)
  apply (*case-tac* ⟨$x \in\# B$⟩)
  by (*auto dest*!: *multi-member-split simp*: *add-mset-union*)
have *merge-conflict-m-pre*: ⟨*merge-conflict-m-pre ?$\mathcal{A}$* ((((((*baa*, *ca*), *ao*), *da*), *i*), *k*), *ag*)⟩
  using *ao conf dist-D dist-NC tauto-NC n-d outl i no-dup lits lits-confl bounded*
  unfolding *merge-conflict-m-pre-def counts-maximum-level-def literals-are-$\mathcal{L}_{in}$-def*
  is-$\mathcal{L}_{all}$-def literals-are-in-$\mathcal{L}_{in}$-mm-def
  by (*auto simp*: *all-lits-of-mm-union all-lits-def*)

have *arena-in-L*: ⟨*arena-lit c C* $\in\#$ $\mathcal{L}_{all}$ *?$\mathcal{A}$*⟩
  if ⟨*Suc ao* $\leq C$⟩ ⟨$C <$ *ao* + *arena-length c ao*⟩ **for** C
proof −
  define D **where** ⟨$D = C -$ *ao*⟩
  with *that* **have** [*simp*]: ⟨C = *ao* + D⟩ **and** *D-le*: ⟨$D <$ *arena-length c ao*⟩
    by *auto*

  have *is-in*: ⟨*ca* $\propto$ *ao* ! $D \in\#$ *mset* (*ca* $\propto$ *ao*)⟩

using *arena that D-le ao*
by (*auto intro*!: *nth-mem simp*: *arena-lifting*(*4*))
have ‹*set-mset* (*all-lits-of-m* (*mset* (*ca* $\propto$ *ao*))) $\subseteq$ *set-mset* ($\mathcal{L}_{all}$ *?*$\mathcal{A}$)›
using *lits ao* **by** (*auto simp*: *literals-are-*$\mathcal{L}_{in}$*-def ran-m-def all-lits-of-mm-add-mset*
is-$\mathcal{L}_{all}$*-def all-lits-def*
dest!: *multi-member-split*)
then have ‹*ca* $\propto$ *ao* ! *D* $\in$# $\mathcal{L}_{all}$ *?*$\mathcal{A}$›
using *multi-member-split*[*OF is-in*]
by (*auto simp*: *all-lits-of-m-add-mset*)

then show *?thesis*
using *arena ao D-le* **by** (*auto simp*: *arena-lifting*)
qed

have [*simp*]: ‹$-$ *bg* $\notin$# *remove1-mset* ($-$ *bg*) (*the da*)›
using *dist-D uL-D multi-member-split*[*of* ‹$-$*bg*› ‹*the da*›]
by *auto*
moreover have [*simp*]: ‹$-$ *bg* $\notin$ *set* (*tl* (*ca* $\propto$ *ao*))›
using *uL-D proped L-M nempty uL-NC*
by (*cases* ‹*ca* $\propto$ *ao*›) (*auto simp*: *resolve-cls-wl*$'$*-def split*: *if-splits*)
ultimately have [*simp*]: ‹$-$ *bg* $\notin$# *remove1-mset* ($-$ *bg*) (*the da* $\cup$# *mset* (*tl* (*ca* $\propto$ *ao*)))›
by (*metis* ‹*a* = *ao*› *diff-single-trivial in-multiset-in-set multi-drop-mem-not-eq*
remove1-mset-union-distrib)
have [*simp*]: ‹*bg* $\notin$# *the da* $\cup$# *mset* (*ca* $\propto$ *ao*) $-$ {#*bg*, $-$ *bg*#}›
‹$-$*bg* $\notin$# *the da* $\cup$# *mset* (*ca* $\propto$ *ao*) $-$ {#*bg*, $-$ *bg*#}›
unfolding *resolve-cls-wl*$'$*-def lookup-clause-rel-def*
lookup-conflict-remove1-def
using *dist-NC bg-hd le2*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bg*› ‹*mset* (*ca* $\propto$ *ao*)›] *tauto-NC-D*
tauto-NC tauto-D uL-D dist-D
apply (*cases* ‹*bg* $\in$# *the da*›)
apply (*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset
dest!: *in-set-takeD multi-member-split*)
apply (*metis distinct-mset-add-mset distinct-mset-union-mset*
sup-union-right1 union-single-eq-member)
by (*smt* ‹*distinct-mset* (*mset* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *ao*))›
add-mset-commute add-mset-diff-bothsides add-mset-remove-trivial-eq dist-D
distinct-mset-add-mset distinct-mset-union-mset get-clauses-wl.simps
get-conflict-wl.simps minus-notin-trivial2 remove-1-mset-id-iff-notin)

have *eq*: ‹(*the da* $\cup$# *mset* (*ca* $\propto$ *ao*) $-$ {#$-$ *bg*, *bg*#}) =
remove1-mset ($-$*bg*) (*the da* $\cup$# *mset* (*tl* (*ca* $\propto$ *ao*)))›
unfolding *resolve-cls-wl*$'$*-def lookup-clause-rel-def*
lookup-conflict-remove1-def
using *dist-NC bg-hd le2*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bg*› ‹*mset* (*ca* $\propto$ *ao*)›] *tauto-NC-D*
tauto-NC tauto-D uL-D dist-D
by (*cases* ‹*ca* $\propto$ *ao*›)
(*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset
dest!: *in-set-takeD multi-member-split*)

have ‹*vmtf-unset-pre* (*atm-of b*) *ivmtf*›
if ‹*ivmtf* ∈ *isa-vmtf ?𝒜 baa*›
apply (*rule vmtf-unset-pre*′[*OF that*])
using *that bg-𝒜*
by (*auto simp*: *vmtf-unset-pre-def in*-$\mathcal{L}_{all}$-*atm-of*-$\mathcal{A}_{in}$)
moreover have ‹*isa-vmtf-unset* (*atm-of bg*) *ivmtf* ∈ *isa-vmtf ?𝒜* (*tl baa*)›
if ‹*ivmtf* ∈ *isa-vmtf ?𝒜 baa*›
by (*rule isa-vmtf-tl-isa-vmtf*[*OF that*])
(*use inv rel that* **in** ‹*auto simp*: *atms-of-def update-confl-tl-wl-pre-def*
twl-st-heur-conflict-ana-def
intro!: *isa-vmtf-unset-isa-vmtf*›)
moreover have
‹*out-learned* (*tl baa*) (*Some* (*remove1-mset* (− *bg*) ((*the da*) ∪# *mset* (*tl* (*ca* ∝ *ao*))))) *b*›
if *H*: ‹*out-learned baa* (*Some* ((*the da*) ∪# *mset* (*tl* (*ca* ∝ *ao*)))) *b*› **for** *b*
proof −
have ‹(− *bg*) ∉# {#*bg* ∈# (*the da*). *get-level baa bg* < *count-decided baa*#}›
using *L-M nempty proped*
by (*cases baa*; *cases* ‹*hd baa*›) *auto*
then have *out*:
‹*out-learned baa* (*Some* (*resolve-cls-wl*′ (*baa*, *ca*, *Some* (*the da*), *ea*, *fa*, *ga*, *ha*) *ao bg*)) *b*›
using *uL-D H bg-hd*
by (*cases* ‹*ca* ∝ *ao*›)
(*auto simp*: *resolve-cls-wl*′-*def out-learned-def ac-simps*)
have ‹*out-learned* (*tl baa*)
(*Some* (*resolve-cls-wl*′ (*baa*, *ca*, *Some* (*the da*), *ea*, *fa*, *ga*, *ha*) *ao bg*)) *b*›
apply (*rule out-learned-tl-Some-notin*[*THEN iffD1*])
using *uL-D out proped L-M nempty proped nempty*
by (*cases baa*; *cases* ‹*hd baa*›; *auto simp*: *resolve-cls-wl*′-*def split*: *if-splits*; *fail*)+
then show *?thesis*
using *rel*
by (*auto simp*: *twl-st-heur-conflict-ana-def merge-conflict-m-def update-confl-tl-wl-pre-def*
resolve-cls-wl′-*def ac-simps eq*)
qed
moreover have ‹*card-max-lvl baa* (*mset* (*tl* (*ca* ∝ *ao*)) ∪# (*the da*)) − *Suc 0*
∈ *counts-maximum-level* (*tl baa*)
(*Some* (*resolve-cls-wl*′ (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *ao bg*))›
proof −
have ‹*distinct-mset* (*remove1-mset* (− *bg*) (*the da* ∪# *mset* (*tl* (*ca* ∝ *ao*))))›
using *dist-NC dist-D* **by** (*auto intro*!: *distinct-mset-minus*)
moreover have ‹¬*tautology* (*remove1-mset* (− *bg*) (*the da* ∪# *mset* (*tl* (*ca* ∝ *ao*))))›
using *tauto-NC-D* **by** (*simp add*: *eq*[*symmetric*] *ac-simps*)
moreover have ‹*card-max-lvl baa* (*mset* (*tl* (*ca* ∝ *ao*)) ∪# *the da*) − *1* =
card-max-lvl baa (*remove1-mset* (− *bg*) (*the da* ∪# *mset* (*tl* (*ca* ∝ *ao*))))›
unfolding ‹*lit-of* (*hd baa*) = *bg*› [*symmetric*]
apply (*subst card-max-lvl-remove1-mset-hd*)
using *uL-D*
by (*auto simp*: *hd-M-L-C ac-simps*)
ultimately show *?thesis*
unfolding *counts-maximum-level-def*
using *uL-D L-M proped nempty* ‹*ao* > *0*› *n-d count-dec-ge*
unfolding *eq*
by (*auto simp del*: *simp*:*ac-simps eq*
card-max-lvl-tl resolve-cls-wl′-*def card-max-lvl-remove1-mset-hd*)
qed
moreover have ‹*da* = *Some y* ⟹ ((*a*, *aaa*, *b*), *Some* (*y* ∪# *mset* (*tl* (*ca* ∝ *ao*))))
∈ *option-lookup-clause-rel ?𝒜* ⟹

((*a*, *lookup-conflict-remove1* (−*bg*) (*aaa*, *b*)),
Some (*remove1-mset* (− *bg*) (*y* ∪# *mset* (*tl* (*ca* ∝ *ao*)))))
∈ *option-lookup-clause-rel ?𝒜*›
for *a aaa b ba y*
using *uL-D bg-D0 bg-D*
using *lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*, *of ?𝒜* ‹−*bg*›
‹*y* ∪# *mset* (*tl* (*ca* ∝ *ao*))› ‹−*bg*› ‹(*aaa*, *b*)›]
by (*auto simp*: *option-lookup-clause-rel-def*
size-remove1-mset-If image-image uminus-$\mathcal{A}_{in}$*-iff*)
moreover have ‹*1* ≤ *card-max-lvl baa* (*the da* ∪# *mset* (*tl* (*ca* ∝ *ao*)))›
using *card-max-lvl* **by** (*auto simp*: *card-max-lvl-def size-union-ge1*)
moreover have ‹((*a*, *aaa*, *b*), *Some* (*the da* ∪# *mset* (*tl* (*ca* ∝ *ao*))))
∈ *option-lookup-clause-rel ?𝒜* ⟹
lookup-conflict-remove1-pre (*bg*, *aaa*, *b*)›
for *a aaa b ba y*
using *uL-D bg-D0 bg-D uL-D*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def atms-of-def*
lookup-conflict-remove1-pre-def mset-ge0-iff
size-remove1-mset-If image-image uminus-$\mathcal{A}_{in}$*-iff simp del*: *bg-D*)
moreover have ‹*tl-trailt-tr-pre ba*›
by (*rule tl-trailt-tr-pre*[*OF - trail*])
(*use nempty* **in** *auto*)
moreover have ‹*arena-act-pre c a*›
using *arena ao* **by** (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*)
moreover have ‹*valid-arena* (*mark-used c a*) *ca* (*set an*)›
using *arena ao* **by** (*auto intro*: *valid-arena-mark-used*)
ultimately show *?thesis*
using *rel inv*
apply −
apply (*rule order-trans*)
apply (*rule isa-resolve-merge-conflict-gt2*[*of ?𝒜* ‹*set an*›,
THEN fref-to-Down-curry6, *OF merge-conflict-m-pre*])
subgoal using *arena ocr trail* **by** (*auto simp*: *all-atms-def*[*symmetric*])
subgoal unfolding *merge-conflict-m-def conc-fun-SPEC*
by (*auto simp*: *twl-st-heur-conflict-ana-def merge-conflict-m-def update-confl-tl-wl-pre-def*
resolve-cls-wl′-def ac-simps no-dup-tlD lookup-remove1-uminus arena-in-L
all-atms-def[*symmetric*] *eq Let-def*
intro!: *ASSERT-refine-left valid-arena-arena-incr-act*
tl-trail-tr[*THEN fref-to-Down-unRET*] *arena-act-pre-mark-used*)
done
qed

have *isasat-lookup-merge-eq2*:
‹*isasat-lookup-merge-eq2 b* (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*) *c a* (*af*, *ag*, *bb*) *i k l*
≤ *SPEC*
(λ*x*. (*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*bb*, *n*, *xs*) ⇒
λ(*clvls*, *lbd*, *outl*). *do* {
ASSERT
(*curry lookup-conflict-remove1-pre b* (*n*, *xs*) ∧
1 ≤ *clvls*);
let (*n*, *xs*) = *lookup-conflict-remove1 b* (*n*, *xs*);
ASSERT (*arena-act-pre c a*);
let c = *mark-used c a*;

ASSERT (*arena-act-pre c a*);
let c = *arena-incr-act c a*;
ASSERT
(*vmtf-unset-pre* (*atm-of b*)
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*));
ASSERT (*tl-trailt-tr-pre* (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*));
RETURN
(*False*, *tl-trailt-tr* (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*,
(*bb*, *n*, *xs*), *e*, *f*,
isa-vmtf-unset (*atm-of b*)
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*),
h, *fast-minus clvls one-uint32-nat*, (*am*, *be*), *lbd*,
outl, (*an*, *ao*, *ap*, *aq*, *ar*, *bf*), (*as*, *at*, *au*, *av*, *bg*),
(*aw*, *ax*, *ay*, *az*, *bh*), (*bi*, *bj*), *ra*, *s*, *t*, *u*, *v*)
})
xa)
$\leq \Downarrow$ (*bool-rel* $\times_f$ *twl-st-heur-conflict-ana′ r*)
(*RETURN*
(*let D* = *resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*
in (*False*, *tl baa*, *ca*, *Some D*, *ea*, *fa*, *ga*, *ha*))))›
if
inv: ‹*update-confl-tl-wl-pre* ((*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
rel: ‹(((*a*, *b*), ((*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*, (*af*, *ag*, *bb*), *e*, *f*,
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*), *h*, *i*, (*am*, *be*), *k*, *l*,
(*an*, *ao*, *ap*, *aq*, *ar*, *bf*), (*as*, *at*, *au*, *av*, *bg*), (*aw*, *ax*, *ay*, *az*, *bh*),
(*bi*, *bj*), *ra*, *s*, *t*, *u*, *v*)),
(*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)
$\in$ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana′ r*› **and**
‹*CLS* = ((*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
‹*CLS′* =
((*a*, *b*), (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*, (*af*, *ag*, *bb*), *e*, *f*,
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*), *h*, *i*, (*am*, *be*), *k*, *l*,
(*an*, *ao*, *ap*, *aq*, *ar*, *bf*), (*as*, *at*, *au*, *av*, *bg*), (*aw*, *ax*, *ay*, *az*, *bh*),
(*bi*, *bj*), *ra*, *s*, *t*, *u*, *v*)› **and**
‹*1* $\leq$ *i*› **and**
‹*2* $\neq$ *2* $\longrightarrow$
curry2 (*curry2* (*curry2 isa-set-lookup-conflict-aa-pre*))
(*aa*, *ab*, *ac*, *ad*, *ae*, *ba*) *c a* (*af*, *ag*, *bb*) *i k l*› **and**
le2: ‹*arena-length c a* = *2*›
for *a b aa ab ac ad ae ba c af ag bb e f ah ai aj ak bc al bd h i am be k l an ao ap aq ar bf as at au av bg aw ax ay az bh bi bj ra s t u v bk bl baa w ca da ea fa ga ha CLS CLS′*
proof −
let *?*$\mathcal{A}$ = ‹*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)›
have
bk: ‹*bk* $\in\#$ *dom-m* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
confl: ‹*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\neq$ *None*› **and**
tr-nempty: ‹*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\neq$ []› **and**
ubl: ‹− *bl* $\in\#$ *the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
L-M: ‹(*bl*, *bk*) =
lit-and-ann-of-propagated
(*hd* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
bl-all: ‹*bl* $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
proped: ‹*is-proped* (*hd* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
‹*0* < *bk*› **and**
card-ge1: ‹*1* $\leq$ *card-max-lvl* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))

(*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
dist: ‹*distinct-mset* (*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
‹$-$ *bl* $\notin$ *set* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*)› **and**
‹*2* $<$ *length* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*) $\longrightarrow$
bl $\notin$ *set* (*tl* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*))› **and**
bl: ‹*bl* $\in$ *set* (*watched-l*
(*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*))› **and**
dist-NC: ‹*distinct* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*)› **and**
tauto-D: ‹$\neg$ *tautology* (*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
tauto-NC: ‹$\neg$ *tautology* (*mset* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*))› **and**
new-tauto: ‹$\neg$ *tautology*
(*remove1-mset bl*
(*remove1-mset* ($-$ *bl*)
(*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)) $\cup\#$
mset (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) $\propto$ *bk*))))› **and**
count-dec: ‹*0* $<$ *count-decided* (*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))
(*the* (*get-conflict-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)))› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))
(*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)› **and**
lits-tr: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))
(*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))›
using *inv* **unfolding** *update-confl-tl-wl-pre-def prod.simps*
by *blast+*
have
[*simp*]: ‹*a* = *bk*› ‹*b* = *bl*› **and**
tr: ‹((*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *baa*) $\in$ *trail-pol* (*all-atms ca* (*ea* + *fa*))› **and**
valid: ‹*valid-arena c ca* (*set ra*)› **and**
o: ‹((*af*, *ag*, *bb*), *da*) $\in$ *option-lookup-clause-rel* (*all-atms ca* (*ea* + *fa*))› **and**
‹(*f*, *ha*) $\in$ $\langle$*Id*$\rangle$*map-fun-rel* (D_0 (*all-atms ca* (*ea* + *fa*)))› **and**
vmtf: ‹((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*) $\in$ *isa-vmtf* (*all-atms ca* (*ea* + *fa*)) *baa*› **and**
‹*phase-saving* (*all-atms ca* (*ea* + *fa*)) *h*› **and**
‹*no-dup baa*› **and**
i: ‹*i* = *card-max-lvl baa* (*the da*)› **and**
‹*cach-refinement-empty* (*all-atms ca* (*ea* + *fa*)) (*am*, *be*)› **and**
out: ‹*out-learned baa da l*› **and**
‹*t* = *size* (*learned-clss-l ca*)› **and**
‹*vdom-m* (*all-atms ca* (*ea* + *fa*)) *ha ca* $\subseteq$ *set ra*› **and**
‹*set s* $\subseteq$ *set ra*› **and**
‹*distinct ra*› **and**
bounded: ‹*isasat-input-bounded* (*all-atms ca* (*ea* + *fa*))› **and**
‹*all-atms ca* (*ea* + *fa*) $\neq$ {#}› **and**
r: ‹*r* = *length c*›
using *rel confl* **unfolding** *twl-st-heur-conflict-ana-def counts-maximum-level-def*
by *auto*
have *n-d*: ‹*no-dup baa*›
using *tr* **unfolding** *trail-pol-alt-def*
by *auto*

have [*simp*]: ‹*lit-of* (*hd baa*) = *bl*› **and** *hd-M-L-C*: ‹*hd baa* = *Propagated bl bk*›
using *L-M tr-nempty proped* **by** (*cases baa*; *cases* ‹*hd baa*›; *auto*; *fail*)+
have *H*: ‹*False*›
if
‹*K* $\in$ *set* (*ca* $\propto$ *bk*)› **and**
‹*K* $\neq$ *bl*› **and**
‹$-$ *K* $\in\#$ *the da*›

for *K*
proof –
have ‹$K \neq -bl$›
using *new-tauto tauto-D tauto-NC that multi-member-split*[*of K* ‹*mset* (*ca* ∝ *bk*)›] *dist*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹$-K$› ‹*mset* (*ca* ∝ *bk*)›] *bl*
by (*cases* ‹*K* ∈# *the da*›)
(*auto dest*!: *multi-member-split*
simp: *sup-union-left2 sup-union-left1 add-mset-eq-add-mset*
tautology-add-mset diff-add-mset-swap
dest: *in-set-takeD*
simp del: *distinct-mset-mset-distinct*)
then show *?thesis*
using *new-tauto tauto-D tauto-NC that multi-member-split*[*of K* ‹*mset* (*ca* ∝ *bk*)›] *dist*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹$-K$› ‹*mset* (*ca* ∝ *bk*)›]
apply (*cases* ‹*K* ∈# *the da*›)
apply (*auto dest*!: *multi-member-split*
simp: *sup-union-left2 sup-union-left1 add-mset-eq-add-mset*
tautology-add-mset diff-add-mset-swap
simp del: *distinct-mset-mset-distinct*)
apply (*subst* (*asm*) *diff-add-mset-swap*)
apply (*auto dest*!: *multi-member-split*
simp: *sup-union-left2 sup-union-left1 add-mset-eq-add-mset*
tautology-add-mset diff-add-mset-swap
simp del: *distinct-mset-mset-distinct*)
apply (*subst* (*asm*) *diff-add-mset-swap*)
apply *auto*
done
qed
have *merge*: ‹*merge-conflict-m-eq2-pre ?*$\mathcal{A}$
(((((((*bl*, *baa*), *ca*), *bk*), *da*), *i*), *k*), *l*)›
using *bk confl dist dist-NC tauto-D tauto-NC lits-D lits lits-tr*
i out n-d bounded le2 valid bl H
unfolding *merge-conflict-m-eq2-pre-def*
by (*auto simp flip*: *all-atms-def simp*: *arena-lifting dest*: *in-set-takeD*)
have *rel*′: ‹((((((((*b*, *aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*), *a*), *af*, *ag*, *bb*), *i*), *k*), *l*),
((((((*bl*, *baa*), *ca*), *bk*), *da*), *i*), *k*), *l*)
∈ *nat-lit-lit-rel* $\times_f$
trail-pol (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)) $\times_f$
{(*arena*, *N*). *valid-arena arena N* (*set ra*)} $\times_f$
nat-rel $\times_f$
option-lookup-clause-rel (*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)) $\times_f$
nat-rel $\times_f$
Id $\times_f$
Id›
using *that* **unfolding** *twl-st-heur-conflict-ana-def* **by** (*auto simp*: *all-atms-def*[*symmetric*])
have *1*: ‹*lookup-conflict-remove1-pre* (*bl*, *aa*, *b*)›
if ‹((*a*, *aa*, *b*), *Some* (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*))
∈ *option-lookup-clause-rel* (*all-atms ca* (*ea* + *fa*))›
for *a aa b ba*
using *o that ubl confl bl-all* **unfolding** *lookup-conflict-remove1-pre-def*
by (*auto simp*: *option-lookup-clause-rel-def atms-of-def*
lookup-clause-rel-def nonempty-has-size[*symmetric*]
dest: *multi-member-split*
simp flip: *all-atms-def*)

have *2*: ‹*tl-trailt-tr-pre* (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*)›
by (*rule tl-trailt-tr-pre*[*OF - tr*])
(*use tr-nempty* **in** *auto*)
have *3*: ‹*vmtf-unset-pre* (*atm-of bl*) ((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*)›
by (*rule vmtf-unset-pre*[*OF vmtf*])
(*use bl-all* **in** ‹*auto simp flip*: *all-atms-def*›)
have *4*: ‹*Suc 0* ≤ *card-max-lvl baa* (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*)›
using *card-ge1 ubl*
by (*auto simp*: *card-max-lvl-def Suc-le-eq nonempty-has-size*[*symmetric*]
dest!: *multi-member-split*)
have *5*: ‹*isa-vmtf-unset* (*atm-of bl*) ((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*)
∈ *isa-vmtf* (*all-atms ca* (*ea* + *fa*)) (*tl baa*)›
by (*rule isa-vmtf-tl-isa-vmtf*[*OF vmtf*])
(*use tr-nempty bl-all* **in** ‹*auto simp flip*: *all-atms-def*›)
have *res-eq*: ‹*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl* =
remove1-mset (−*bl*) (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*)›
using *dist dist-NC bl*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bl*› ‹*mset* (*ca* ∝ *bk*)›] *ubl new-tauto*
tauto-NC tauto-D
unfolding *resolve-cls-wl′-def*
by (*cases* ‹*bl* ∈# *the da*›)
(*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset ac-simps
dest!: *in-set-takeD multi-member-split*)
then have *res-eq′*: ‹*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da* =
add-mset (−*bl*) (*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*)›
using *ubl* **by** *auto*
have *eq6*: ‹(*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da* −
{#*Pos* (*atm-of bl*), *Neg* (*atm-of bl*)#}) =
(*remove1-mset* (− *bl*) (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*))›
using *dist dist-NC bl*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bl*› ‹*mset* (*ca* ∝ *bk*)›] *ubl new-tauto*
tauto-NC tauto-D
by (*cases bl*; *cases* ‹*bl* ∈# *the da*›)
(*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset ac-simps
dest!: *in-set-takeD multi-member-split*)
have *7*: ‹((*a*, *aaa*, *b*), *Some* (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*))
∈ *option-lookup-clause-rel* (*all-atms ca* (*ea* + *fa*)) ⟹
((*a*, *lookup-conflict-remove1 bl* (*aaa*, *b*)),
Some (*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*))
∈ *option-lookup-clause-rel* (*all-atms ca* (*ea* + *fa*))›
for *a aa b ba aaa*
using *ubl bl-all eq6*
mset-as-position-remove[*of b* ‹*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*› ‹*atm-of bl*›]
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
lookup-conflict-remove1-def res-eq
size-remove1-mset-If[*of -* ‹−*bl*›] *atms-of-def*
simp flip: *all-atms-def*)
have [*iff*]: ‹*bl* ∉# *resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*›
unfolding *resolve-cls-wl′-def lookup-clause-rel-def*
lookup-conflict-remove1-def

using *dist dist-NC bl*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bl*› ‹*mset* (*ca* ∝ *bk*)›] *ubl new-tauto*
tauto-NC tauto-D
apply (*cases* ‹*bl* ∈# *the da*›)
apply (*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset
dest!: *in-set-takeD multi-member-split*)
by (*metis distinct-mset-add-mset distinct-mset-union-mset*
sup-union-right1 union-single-eq-member)
have [*iff*]: ‹−*bl* ∉# *resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*›
unfolding *resolve-cls-wl′-def lookup-clause-rel-def*
lookup-conflict-remove1-def
using *dist dist-NC bl*
dist-NC[*unfolded distinct-mset-mset-distinct*[*symmetric*]]
multi-member-split[*of* ‹*bl*› ‹*mset* (*ca* ∝ *bk*)›] *ubl new-tauto*
tauto-NC tauto-D
apply (*cases* ‹*bl* ∈# *the da*›)
apply (*auto simp del*: *distinct-mset-mset-distinct*
simp: *tautology-add-mset sup-union-left2 sup-union-left1*
add-mset-eq-add-mset
dest!: *in-set-takeD multi-member-split*)
by (*metis distinct-mset-add-mset distinct-mset-union-mset*
sup-union-right1 union-single-eq-member)

have *6*: ‹*out-learned* (*tl baa*)
(*Some* (*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*)) *bab*›
if ‹*out-learned baa* (*Some* (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*)) *bab*›
for *bab*
apply (*rule out-learned-tl-Some-notin*[*THEN iffD1*])
apply (*use that proped* **in** ‹*auto simp*: ›)
apply (*auto simp*: *res-eq intro*!: *out-learned-add-mset-highest-level*)
by (*metis* ‹*lit-of* (*hd baa*) = *bl*› *diff-single-trivial insert-DiffM*
out-learned-add-mset-highest-level)
have *new-dist*: ‹*distinct-mset* (*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*)›
using *dist-NC dist*
by (*auto simp*: *resolve-cls-wl′-def*)
have *eq8*: ‹*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl* =
the da ∪# *mset* (*ca* ∝ *bk*) − {#*bl*, − *bl*#}›
by (*simp add*: *resolve-cls-wl′-def*)
have *highest-lev*: ‹*get-level baa bl* = *count-decided baa*›
using *tr-nempty hd-M-L-C*
by (*cases baa*) (*auto*)
have *act*: ‹*arena-act-pre c bk*›
using *bk valid* **by** (*auto simp*: *arena-act-pre-def*
arena-is-valid-clause-idx-def)
have *valid-used*: ‹*valid-arena* (*mark-used c bk*) *ca* (*set ra*)›
using *valid bk* **by** (*auto intro*: *valid-arena-mark-used*)
have *8*: ‹*card-max-lvl baa* (*remove1-mset bl* (*mset* (*ca* ∝ *bk*)) ∪# *the da*) − *Suc 0*
∈ *counts-maximum-level* (*tl baa*)
(*Some* (*resolve-cls-wl′* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*) *bk bl*))›
for *a aa b ba aaa*
using *count-dec tr-nempty new-tauto n-d proped new-dist highest-lev*
unfolding *res-eq′*
by (*auto simp*: *counts-maximum-level-def res-eq′*

card-max-lvl-tl eq8[*symmetric*] *card-max-lvl-add-mset*)
show *?thesis*
apply (*rule isasat-lookup-merge-eq2*[*THEN fref-to-Down-curry7*, *THEN order-trans*])
apply (*rule merge*)
apply (*rule rel′*)
using *1 2 3 4 5 6 7 8 tr valid rel tr-nempty n-d no-dup-tlD*[*OF n-d*] *act valid-used bk*
unfolding *merge-conflict-m-g-eq2-def merge-conflict-m-eq2-def Let-def*
by (*auto intro*!: *RES-refine ASSERT-refine-left*
tl-trail-tr[*THEN fref-to-Down-unRET*] *valid-arena-arena-incr-act*
arena-act-pre-mark-used
simp: *conc-fun-RES r twl-st-heur-conflict-ana-def*
simp flip: *all-atms-def*)
qed

have *isa-set-lookup-conflict-aa-pre*:
⟨*curry6 isa-set-lookup-conflict-aa-pre*
(*aa*, *ab*, *ac*, *ad*, *ae*, *ba*) *c a* (*af*, *ag*, *bb*) *i k l*⟩ (**is** *?A*) **and**
valid: ⟨*arena-is-valid-clause-idx c a*⟩ (**is** *?B*)
if
inv: ⟨*update-confl-tl-wl-pre* ((*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)⟩ **and**
⟨$1 \le i$⟩ **and**
rel: ⟨(((*a*, *b*), (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*, (*af*, *ag*, *bb*), *e*, *f*,
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*), *h*, *i*, (*am*, *be*), *k*, *l*,
(*an*, *ao*, *ap*, *aq*, *ar*, *bf*), (*as*, *at*, *au*, *av*, *bg*), (*aw*, *ax*, *ay*, *az*, *bh*),
(*bi*, *bj*), *ra*, *s*, *t*, *u*, *v*),
(*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)
∈ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana′ r*⟩ **and**
CLS: ⟨*CLS* = ((*bk*, *bl*), *baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)⟩ **and**
⟨*CLS′* =
((*a*, *b*), (*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *c*, (*af*, *ag*, *bb*), *e*, *f*,
((*ah*, *ai*, *aj*, *ak*, *bc*), *al*, *bd*), *h*, *i*, (*am*, *be*), *k*, *l*,
(*an*, *ao*, *ap*, *aq*, *ar*, *bf*), (*as*, *at*, *au*, *av*, *bg*), (*aw*, *ax*, *ay*, *az*, *bh*),
(*bi*, *bj*), *ra*, *s*, *t*, *u*, *v*)⟩
for *a b aa ab ac ad ae ba c af ag bb e f ah ai aj ak bc al bd h i am be k l an ao ap aq ar bf as at au av bg aw ax ay az bh bi bj ra s t u v bk bl baa ca da ea fa ga ha CLS CLS′*
proof −
let *?𝒜* = ⟨*all-atms-st* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*)⟩

have
ao: ⟨*bk* ∈# *dom-m* (*get-clauses-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))⟩ **and**
lits-trail: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail ?𝒜*
(*get-trail-wl* (*baa*, *ca*, *da*, *ea*, *fa*, *ga*, *ha*))⟩
using *inv* **unfolding** *CLS update-confl-tl-wl-pre-def prod.case* **apply** −
by *blast*+

have
arena: ⟨*valid-arena c ca* (*set ra*)⟩ **and**
ocr: ⟨((*af*, *ag*, *bb*), *da*) ∈ *option-lookup-clause-rel ?𝒜*⟩ **and**
trail: ⟨((*aa*, *ab*, *ac*, *ad*, *ae*, *ba*), *baa*) ∈ *trail-pol ?𝒜*⟩ **and**
[*simp*]: ⟨*bk* = *a*⟩
using *rel* **by** (*auto simp*: *CLS twl-st-heur-conflict-ana-def all-atms-def*[*symmetric*])
show *?A*
using *arena lits-trail ao*
unfolding *isa-set-lookup-conflict-aa-pre-def*

by (*auto simp*: *arena-lifting lookup-conflict-remove1-def*)
show *?B*
using *arena ao*
unfolding *arena-is-valid-clause-idx-def*
by *auto*
qed
show *?thesis*
supply [[*goals-limit* = *2*]]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
subgoal for CLS' *CLS*
unfolding *uncurry-def update-confl-tl-wl-heur-def comp-def update-confl-tl-wl-def*
apply (*cases* CLS'; *cases CLS*)
apply *clarify*
apply (*refine-rcg lhs-step-If specify-left*; *remove-dummy-vars*)
subgoal
by (*auto simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre-def RES-RETURN-RES RETURN-def counts-maximum-level-def*)
subgoal
by (*rule isa-set-lookup-conflict-aa-pre*)
subgoal by (*rule valid*)
subgoal by (*rule isasat-lookup-merge-eq2*)
subgoal by (*rule H*)
done
done
qed

lemma *phase-saving-le*: ‹$phase\text{-}saving\ \mathcal{A}\ \varphi \Longrightarrow A \in\#\ \mathcal{A} \Longrightarrow A < length\ \varphi$›
‹$phase\text{-}saving\ \mathcal{A}\ \varphi \Longrightarrow B \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow atm\text{-}of\ B < length\ \varphi$›
by (*auto simp*: *phase-saving-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *isa-vmtf-le*:
‹$((a, b), M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow A \in\#\ \mathcal{A} \Longrightarrow A < length\ a$›
‹$((a, b), M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow B \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow atm\text{-}of\ B < length\ a$›
by (*auto simp*: *isa-vmtf-def vmtf-def vmtf-*$\mathcal{L}_{all}$*-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *isa-vmtf-next-search-le*:
‹$((a, b, c, c', Some\ d), M) \in isa\text{-}vmtf\ \mathcal{A}\ M' \Longrightarrow d < length\ a$›
by (*auto simp*: *isa-vmtf-def vmtf-def vmtf-*$\mathcal{L}_{all}$*-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *trail-pol-nempty*: ‹$\neg(([], aa, ab, ac, ad, b), L\ \#\ ys) \in trail\text{-}pol\ \mathcal{A}$›
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

definition *is-decided-hd-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹$is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur = (\lambda S.\ is\text{-}None\ (snd\ (last\text{-}trail\text{-}pol\ (get\text{-}trail\text{-}wl\text{-}heur\ S))))$›

lemma *is-decided-hd-trail-wl-heur-hd-get-trail*:
‹$(RETURN\ o\ is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur,\ RETURN\ o\ (\lambda M.\ is\text{-}decided\ (hd\ (get\text{-}trail\text{-}wl\ M))))$
$\in [\lambda M.\ get\text{-}trail\text{-}wl\ M \neq []]_f\ twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana'\ r \rightarrow \langle bool\text{-}rel\rangle\ nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *is-decided-hd-trail-wl-heur-def twl-st-heur-conflict-ana-def neq-Nil-conv trail-pol-def ann-lits-split-reasons-def is-decided-no-proped-iff last-trail-pol-def*
split: *option.splits*)

definition *is-decided-hd-trail-wl-heur-pre* **where**
‹*is-decided-hd-trail-wl-heur-pre* =
(λS. *fst* (*get-trail-wl-heur S*) $\neq$ [] $\wedge$ *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'). $\exists S\ S_0$. (S', S) $\in$ *twl-st-heur-conflict-ana* $\wedge$ ($S_0{}'$, S_0) $\in$ *twl-st-heur-conflict-ana* $\wedge$
skip-and-resolve-loop-wl-D-inv S_0 *brk S* $\wedge$
length (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* $S_0{}'$) $\wedge$
is-decided-hd-trail-wl-heur-pre S')›

definition *update-confl-tl-wl-heur-pre*
:: ‹(*nat* $\times$ *nat literal*) $\times$ *twl-st-wl-heur* $\Rightarrow$ *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* $\wedge$
(*fst M*) $\neq$ [] $\wedge$
atm-of ((*last* (*fst M*))) < *length* φ $\wedge$
atm-of ((*last* (*fst M*))) < *length A* $\wedge$ (*next-search* $\neq$ *None* $\longrightarrow$ *the next-search* < *length A*) $\wedge$
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* $\wedge$ *M* $\neq$ [])›

definition *atm-is-in-conflict-st-heur-pre*
:: ‹*nat literal* $\times$ *twl-st-wl-heur* $\Rightarrow$ *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* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*skip-and-resolve-loop-wl-D-heur* S_0 =
do {
(-, *S*) $\leftarrow$
$WHILE_T{}^{\textit{skip-and-resolve-loop-wl-D-heur-inv}\ S_0}$
(λ(*brk*, *S*). $\neg$*brk* $\wedge$ $\neg$*is-decided-hd-trail-wl-heur S*)
(λ(*brk*, *S*).
do {
ASSERT($\neg$*brk* $\wedge$ $\neg$*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 −
obtain *x xa xb xc* **where**
lits: ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st x2*) *x2*⟩ **and**
x2-x: ⟨(*x2*, *x*) ∈ *state-wl-l None*⟩ **and**
y-xa: ⟨(*y*, *xa*) ∈ *state-wl-l None*⟩ **and**
⟨*correct-watching x2*⟩ **and**
x-xb: ⟨(*x*, *xb*) ∈ *twl-st-l None*⟩ **and**
xa-xc: ⟨(*xa*, *xc*) ∈ *twl-st-l None*⟩ **and**
⟨*cdcl-twl-o*** *xc xb*⟩ **and**
list-invs: ⟨*twl-list-invs x*⟩ **and**
struct: ⟨*twl-struct-invs xb*⟩ **and**
⟨*clauses-to-update-l x* = {#}⟩ **and**
⟨¬ *is-decided* (*hd* (*get-trail-l x*)) ⟶ *0* < *mark-of* (*hd* (*get-trail-l x*))⟩ **and**
⟨*twl-stgy-invs xb*⟩ **and**
⟨*clauses-to-update xb* = {#}⟩ **and**
⟨*literals-to-update xb* = {#}⟩ **and**
confl: ⟨*get-conflict xb* ≠ *None*⟩ **and**
count-dec: ⟨*count-decided* (*get-trail xb*) ≠ *0*⟩ **and**
trail-nempty: ⟨*get-trail xb* ≠ []⟩ **and**
⟨*get-conflict xb* ≠ *Some* {#}⟩ **and**
⟨*x1* ⟶
(*no-step* $cdcl_W$*-restart-mset.skip* ($state_W$*-of xb*)) ∧
(*no-step* $cdcl_W$*-restart-mset.resolve* ($state_W$*-of xb*))⟩
using *sor-inv*
unfolding *skip-and-resolve-loop-wl-D-inv-def prod.simps xa skip-and-resolve-loop-wl-D-heur-inv-def skip-and-resolve-loop-wl-inv-def x′ skip-and-resolve-loop-inv-l-def*

skip-and-resolve-loop-inv-def **by** *blast*
show ‹*literals-are-$\mathcal{L}_{in}$* (*all-atms-st x2*) *x2*›
using *lits* **.**
show ‹*get-conflict-wl x2* ≠ *None*›
using *x2-x y-xa confl x-xb*
by *auto*
show ‹*literals-are-in-$\mathcal{L}_{in}$-trail* (*all-atms-st x2*) (*get-trail-wl x2*)›
using ‹*twl-struct-invs xb*› *literals-are-$\mathcal{L}_{in}$-literals-are-$\mathcal{L}_{in}$-trail lits x2-x x-xb*
by *blast*
show *trail-nempty-x2*: ‹*get-trail-wl x2* ≠ []›
using *x2-x y-xa confl x-xb trail-nempty*
by *auto*

have *cdcl-confl*: ‹*cdcl$_W$-restart-mset.cdcl$_W$-conflicting* (*state$_W$-of xb*)› **and**
‹*cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv* (*state$_W$-of xb*)› **and**
dist: ‹*cdcl$_W$-restart-mset.distinct-cdcl$_W$-state* (*state$_W$-of xb*)›
using *struct*
unfolding *twl-struct-invs-def cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv-def*
by *blast+*
then have *confl'*: ‹∀ *T*. *conflicting* (*state$_W$-of xb*) = *Some T* ⟶
trail (*state$_W$-of xb*) ⊨*as CNot T*› **and**
‹*no-dup* (*trail* (*state$_W$-of xb*))› **and**
confl-annot: ‹⋀*L mark a b*.
a @ *Propagated L mark* # *b* = *trail* (*state$_W$-of xb*) ⟶
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*›
unfolding *cdcl$_W$-restart-mset.cdcl$_W$-conflicting-def*
cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv-def
by *blast+*
then have *conflicting*: ‹*get-trail-wl x2* ⊨*as CNot* (*the* (*get-conflict-wl x2*))› **and**
‹*no-dup* (*get-trail-wl x2*)›
using *x2-x y-xa confl x-xb trail-nempty*
by (*auto simp*: *twl-st*)

show ‹¬*tautology* (*the* (*get-conflict-wl x2*))›
using ‹*get-conflict-wl x2* ≠ *None*› *conflict-not-tautology*
struct x2-x x-xb **by** *blast*
show *dist-mset*: ‹*distinct-mset* (*the* (*get-conflict-wl x2*))› **and**
‹*count-decided* (*get-trail-wl x2*) ≠ *0*›
using *dist x2-x x-xb* ‹*get-conflict-wl x2* ≠ *None*› *count-dec*
by (*auto simp*: *cdcl$_W$-restart-mset.distinct-cdcl$_W$-state-def*
twl-st)
show ‹*no-dup* (*get-trail-wl x2*)›
using ‹*no-dup* (*get-trail-wl x2*)› **.**

show *lits-confl*: ‹*literals-are-in-$\mathcal{L}_{in}$* (*all-atms-st x2*) (*the* (*get-conflict-wl x2*))›
by (*rule literals-are-$\mathcal{L}_{in}$-literals-are-in-$\mathcal{L}_{in}$-conflict*[*OF x2-x struct x-xb lits*])
(*use x2-x x-xb confl* **in** *auto*)
qed

private lemma *sor-heur-inv-heur1*:
‹*fst* (*get-trail-wl-heur x2b*) ≠ []›
using *xa-x' trail-nempty* **unfolding** *xa x'*
by (*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-nempty last-trail-pol-pre-def*
dest!: *neq-Nil-conv*[*THEN iffD1*])

private lemma *sor-heur-inv-heur2*:

⟨*last-trail-pol-pre* (*get-trail-wl-heur x2b*)⟩
using *xa-x′ trail-nempty*[*THEN neq-Nil-conv*[*THEN iffD1*]] *sor-heur-inv-heur1* **unfolding** *xa x′*
by (*cases x2b*; *cases x2*; *cases* ⟨*get-trail-wl-heur x2b*⟩)
(*auto simp add*: *twl-st-heur-conflict-ana-def trail-pol-def last-trail-pol-pre-def*
ann-lits-split-reasons-def)

lemma *sor-heur-inv*:
⟨*skip-and-resolve-loop-wl-D-heur-inv x xa*⟩
using *sor-inv xa-x′ xy sor-heur-inv-heur1 sor-heur-inv-heur2* **unfolding** *xa x′*
unfolding *skip-and-resolve-loop-wl-D-heur-inv-def prod.simps* **apply** −
apply (*rule exI*[*of* - ⟨*x2*⟩])
apply (*rule exI*[*of* - *y*])
by (*auto simp*: *is-decided-hd-trail-wl-heur-pre-def*)

lemma *conflict-ana-same-cond*:
⟨(¬ *x1b* ∧ ¬ *is-decided-hd-trail-wl-heur x2b*) =
(¬ *x1* ∧ ¬ *is-decided* (*hd* (*get-trail-wl x2*)))⟩
apply (*subst is-decided-hd-trail-wl-heur-hd-get-trail*[*THEN fref-to-Down-unRET-Id*, *OF trail-nempty*,
of - ⟨*length* (*get-clauses-wl-heur x*)⟩])
using *xa-x′* **unfolding** *xa x′*
by *auto*

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*))⟩
using *xy xa-x′ x′*
twl-st-heur-conflict-ana-lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st[*of x2b x2*]
xa xc assert
by (*auto simp*: *is-decided-no-proped-iff xc*
lit-and-ann-of-propagated-st-def hd-xa)

private 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 −
obtain *x xa xb xc* **where**
lits: ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st x2*) *x2*⟩ **and**
x2-x: ⟨(*x2*, *x*) ∈ *state-wl-l None*⟩ **and**
y-xa: ⟨(*y*, *xa*) ∈ *state-wl-l None*⟩ **and**
⟨*correct-watching x2*⟩ **and**
x-xb: ⟨(*x*, *xb*) ∈ *twl-st-l None*⟩ **and**
xa-xc: ⟨(*xa*, *xc*) ∈ *twl-st-l None*⟩ **and**
⟨*cdcl-twl-o*** *xc xb*⟩ **and**
list-invs: ⟨*twl-list-invs x*⟩ **and**
struct: ⟨*twl-struct-invs xb*⟩ **and**
⟨*clauses-to-update-l x* = {#}⟩ **and**
⟨¬ *is-decided* (*hd* (*get-trail-l x*)) ⟶ *0* < *mark-of* (*hd* (*get-trail-l x*))⟩ **and**
⟨*twl-stgy-invs xb*⟩ **and**
⟨*clauses-to-update xb* = {#}⟩ **and**
⟨*literals-to-update xb* = {#}⟩ **and**
confl: ⟨*get-conflict xb* ≠ *None*⟩ **and**
count-dec: ⟨*count-decided* (*get-trail xb*) ≠ *0*⟩ **and**
trail-nempty: ⟨*get-trail xb* ≠ []⟩ **and**
⟨*get-conflict xb* ≠ *Some* {#}⟩ **and**
⟨*x1* ⟶
(*no-step cdcl*$_W$*-restart-mset.skip* (*state*$_W$*-of xb*)) ∧
(*no-step cdcl*$_W$*-restart-mset.resolve* (*state*$_W$*-of xb*))⟩
using *sor-inv*
unfolding *skip-and-resolve-loop-wl-D-inv-def prod.simps xa skip-and-resolve-loop-wl-D-heur-inv-def skip-and-resolve-loop-wl-inv-def x′ skip-and-resolve-loop-inv-l-def skip-and-resolve-loop-inv-def* **by** *blast*
show *st2*[*simp*]: ⟨*x1a* = *x1c*⟩⟨*x2a* = *x2c*⟩
using *trail-nempty xy xa-x′ x′ xc x-xb x2-x*
twl-st-heur-conflict-ana-lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st[*of x2b x2*]
assert xa
by (*auto simp*: *is-decided-no-proped-iff xc*
lit-and-ann-of-propagated-st-def hd-xa)
have ⟨*get-conflict-wl x2* ≠ *None*⟩
using *x2-x y-xa confl x-xb*
by *auto*
have ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st x2*) (*get-trail-wl x2*)⟩
using ⟨*twl-struct-invs xb*⟩ *literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail lits x2-x x-xb*
by *blast*
moreover have *trail-nempty-x2*: ⟨*get-trail-wl x2* ≠ []⟩
using *x2-x y-xa confl x-xb trail-nempty*
by *auto*
ultimately show ⟨*x1c* ∈# $\mathcal{L}_{all}$ (*all-atms-st x2*)⟩
using *hd-xa assert*
apply (*cases* ⟨*get-trail-wl x2*⟩; *cases* ⟨*hd* (*get-trail-wl x2*)⟩)
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons*)

have *cdcl-confl*: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of xb*)⟩ **and**
⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of xb*)⟩ **and**
dist: ⟨*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of xb*)⟩
using *struct*
unfolding *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *blast+*
then have *confl′*: ⟨∀ *T*. *conflicting* (*state*$_W$*-of xb*) = *Some T* ⟶

trail (*state*$_W$-*of xb*) $\models$*as CNot T*› **and**
‹*no-dup* (*trail* (*state*$_W$-*of xb*))› **and**
confl-annot: ‹$\bigwedge$*L mark a b*.
a @ *Propagated L mark* # *b* = *trail* (*state*$_W$-*of xb*) $\longrightarrow$
b $\models$*as CNot* (*remove1-mset L mark*) $\wedge$ *L* $\in$# *mark*›
unfolding *cdcl*$_W$-*restart-mset.cdcl*$_W$-*conflicting-def*
cdcl$_W$-*restart-mset.cdcl*$_W$-*M-level-inv-def*
by *blast*+
then have *conflicting*: ‹*get-trail-wl x2* $\models$*as CNot* (*the* (*get-conflict-wl x2*))› **and**
‹*no-dup* (*get-trail-wl x2*)›
using *x2-x y-xa confl x-xb trail-nempty*
by (*auto simp*: *twl-st*)
then show ‹*x1c* $\notin$# *the* (*get-conflict-wl x2*)›
using *hd-xa assert*
by (*cases* ‹*get-trail-wl x2*›; *cases* ‹*hd* (*get-trail-wl x2*)›)
(*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons dest*!: *multi-member-split*
dest: *in-lits-of-l-defined-litD*)
have ‹$\neg$*tautology* (*the* (*get-conflict-wl x2*))›
using ‹*get-conflict-wl x2* $\neq$ *None*› *conflict-not-tautology*
struct x2-x x-xb **by** *blast*
have *dist-mset*: ‹*distinct-mset* (*the* (*get-conflict-wl x2*))› **and**
‹*count-decided* (*get-trail-wl x2*) $\neq$ *0*›
using *dist x2-x x-xb* ‹*get-conflict-wl x2* $\neq$ *None*› *count-dec*
by (*auto simp*: *cdcl*$_W$-*restart-mset.distinct-cdcl*$_W$-*state-def*
twl-st)
have ‹*no-dup* (*get-trail-wl x2*)›
using ‹*no-dup* (*get-trail-wl x2*)› **.**
show *mark-ge0*: ‹*0* < *mark-of* (*hd* (*get-trail-wl x2*))›
using ‹$\neg$ *is-decided* (*hd* (*get-trail-l x*)) $\longrightarrow$ *0* < *mark-of* (*hd* (*get-trail-l x*))›
x2-x assert **by** *auto*
then show ‹*x2c* $\in$# *dom-m* (*get-clauses-wl x2*)› **and**
hd-x1c: ‹*length* (*get-clauses-wl x2* $\propto$ *x2c*) > *2* $\longrightarrow$ *hd* (*get-clauses-wl x2* $\propto$ *x2c*) = *x1c*› **and**
x2c-nempty: ‹*get-clauses-wl x2* $\propto$ *x2c* $\neq$ []›
using *list-invs x2-x assert* ‹*get-trail-wl x2* $\neq$ []› *hd-xa trail-nempty-x2*
by (*cases* ‹*get-trail-wl x2*›; *cases* ‹*hd* (*get-trail-wl x2*)›;
cases ‹*get-clauses-wl x2* $\propto$ *x2c*›;
auto simp: *twl-list-invs-def all-conj-distrib hd-conv-nth*; *fail*)+
have *lits-confl*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st x2*) (*the* (*get-conflict-wl x2*))›
by (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-in-*$\mathcal{L}_{in}$*-conflict*[*OF x2-x struct x-xb lits*])
(*use x2-x x-xb confl* **in** *auto*)
show *hd-x2*: ‹*hd* (*get-trail-wl x2*) = *Propagated x1c x2c*›
using *trail-nempty assert hd-xa*
by (*cases* ‹*get-trail-wl x2*›; *cases* ‹*hd* (*get-trail-wl x2*)›) *auto*
then have ‹*0* < *x2c*›
using *mark-ge0* **by** *auto*
then show
x1c-watched: ‹*x1c* $\in$ *set* (*watched-l* (*get-clauses-wl x2* $\propto$ *x2c*))›
using *list-invs x2-x assert* ‹*get-trail-wl x2* $\neq$ []› *hd-xa trail-nempty-x2*
by (*cases* ‹*get-trail-wl x2*›; *cases* ‹*hd* (*get-trail-wl x2*)›;
auto simp: *twl-list-invs-def all-conj-distrib hd-conv-nth*)
have ‹$\neg$*is-decided* (*hd* (*get-trail xb*))›
using *trail-nempty trail-nempty-x2 assert hd-xa x-xb x2b-x2 x2-x x-xb*
by (*auto simp*: *twl-st-l-mark-of-is-decided state-wl-l-mark-of-is-decided*)
then have *Neg*:
‹*tl* (*get-trail-wl x2*) $\models$*as CNot* (*remove1-mset x1c* (*mset* (*get-clauses-wl x2* $\propto$ *x2c*))) $\wedge$
x1c $\in$# *mset* (*get-clauses-wl x2* $\propto$ *x2c*)›

using *confl-annot*[*of Nil*] *x2-x x-xb hd-get-trail-twl-st-of-get-trail-l*[*OF x-xb*] *trail-nempty hd-xa hd-x2 trail-nempty trail-nempty-x2 assert* ‹*0* < *x2c*›
twl-st-l-mark-of-hd[*OF x-xb*] *twl-st-l-lits-of-tl*[*OF x-xb*]
by (*cases* ‹*get-trail xb*›; *cases* ‹*hd* (*get-trail xb*)›; *cases* ‹*get-trail-wl x2*›)
(*auto simp*: *twl-st true-annots-true-cls simp del*: *hd-get-trail-twl-st-of-get-trail-l*)

show ‹− *x1c* ∉ *set* (*get-clauses-wl x2* ∝ *x2c*)›
using *Neg hd-x1c x2c-nempty* ‹*no-dup* (*get-trail-wl x2*)› *hd-x2*
apply (*cases* ‹*get-clauses-wl x2* ∝ *x2c*›; *cases* ‹*get-trail-wl x2*›)
by (*auto 5 5 simp*: *true-annots-true-cls-def-iff-negation-in-model*
dest: *in-lits-of-l-defined-litD*)
show ‹*length* (*get-clauses-wl x2* ∝ *x2c*) > *2* ⟶ *x1c* ∉ *set* (*tl* (*get-clauses-wl x2* ∝ *x2c*))›
using *Neg hd-x1c x2c-nempty* ‹*no-dup* (*get-trail-wl x2*)› *hd-x2*
by (*cases* ‹*get-clauses-wl x2* ∝ *x2c*›; *cases* ‹*get-trail-wl x2*›)
(*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*
dest: *in-lits-of-l-defined-litD*)

show *dist-NC*: ‹*distinct* (*get-clauses-wl x2* ∝ *x2c*)›
using *dist x-xb x2-x* ‹*x2c* ∈# *dom-m* (*get-clauses-wl x2*)›
unfolding $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-alt-def*
by (*auto simp*: *twl-st ran-m-def dest*!: *multi-member-split*)
have [*iff*]: ‹*x1c* ∉ *lits-of-l* (*tl* (*get-trail-wl x2*))›
‹−*x1c* ∉ *lits-of-l* (*tl* (*get-trail-wl x2*))›
using ‹*no-dup* (*get-trail-wl x2*)› *trail-nempty-x2 hd-x2*
by (*cases* ‹*get-trail-wl x2*›; *auto dest*: *in-lits-of-l-defined-litD*; *fail*)+
have ‹¬*tautology* (*remove1-mset x1c* (*mset* (*get-clauses-wl x2* ∝ *x2c*)))›
apply (*rule consistent-CNot-not-tautology*[*of* ‹*lits-of-l* (*tl* (*get-trail-wl x2*))›])
using *Neg* ‹*no-dup* (*get-trail-wl x2*)›
by (*auto simp*: *true-annots-true-cls intro*!: *distinct-consistent-interp*
dest: *no-dup-tlD*)
then show ‹¬*tautology* (*mset* (*get-clauses-wl x2* ∝ *x2c*))›
using *Neg multi-member-split*[*of x1c* ‹*mset* (*get-clauses-wl x2* ∝ *x2c*)›] *hd-x1c*
‹− *x1c* ∉ *set* (*get-clauses-wl x2* ∝ *x2c*)› *dist-NC*
‹*no-dup* (*get-trail-wl x2*)› *x1c-watched*
by (*cases* ‹*get-clauses-wl x2* ∝ *x2c*›;
cases ‹*tl* (*get-clauses-wl x2* ∝ *x2c*)›)
(*auto simp*: *tautology-add-mset add-mset-eq-add-mset*
uminus-lit-swap
true-annots-true-cls dest!: *in-set-takeD*)
have ‹*distinct-mset* (*the* (*get-conflict-wl x2*) ∪# *mset* (*tl* (*get-clauses-wl x2* ∝ *x2c*)))›
using *dist dist-mset dist-NC*
by (*auto*)
then have *H*: ‹*get-trail-wl x2* ⊨*as*
CNot (*remove1-mset x1c* (*remove1-mset* (− *x1c*)
(*the* (*get-conflict-wl x2*) ∪# *mset* ((*get-clauses-wl x2* ∝ *x2c*)))))›
using *Neg hd-x1c trail-nempty-x2 hd-x2 conflicting* ‹− *x1c* ∉ *lits-of-l* (*tl* (*get-trail-wl x2*))›
apply (*cases* ‹*get-clauses-wl x2* ∝ *x2c*›;
cases ‹*get-trail-wl x2*›)
apply (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model distinct-mset-remove1-All uminus-lit-swap*)
using ‹*x1c* ∉# *the* (*get-conflict-wl x2*)› *remove-1-mset-id-iff-notin* **apply** *fastforce*
using ‹*x1c* ∉# *the* (*get-conflict-wl x2*)› *remove-1-mset-id-iff-notin* **apply** *fastforce*
apply (*smt dist-NC distinct-mem-diff-mset distinct-mset-mset-distinct*
distinct-mset-set-mset-remove1-mset distinct-mset-union-mset in-diffD
in-remove1-mset-neq member-add-mset mset.simps(*2*) *remove1-mset-union-distrib*
remove-1-mset-id-iff-notin set-mset-mset)

done
show ‹¬*tautology* (*remove1-mset x1c*
(*remove1-mset* (− *x1c*) (*the* (*get-conflict-wl x2*) ∪# *mset* (*get-clauses-wl x2* ∝ *x2c*))))›
apply (*rule consistent-CNot-not-tautology*[*OF* - *H*[*unfolded true-annots-true-cls*]])
using ‹*no-dup* (*get-trail-wl x2*)›
by (*auto simp*: *true-annots-true-cls intro*!: *distinct-consistent-interp*
dest: *no-dup-tlD*)
show
x1c-NC-0: ‹*2* < *length* (*get-clauses-wl x2* ∝ *x2c*) ⟶ *get-clauses-wl x2* ∝ *x2c* ! *0* = *x1c*›
by (*metis hd-conv-nth hd-x1c x2c-nempty*)
qed

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*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *atm-is-in-conflict-st-heur-is-in-conflict-st-ana*[*THEN fref-to-Down*, *of y x*]
by (*case-tac x*, *case-tac y*)
auto
done

lemma *atm-is-in-conflict-st-heur-iff*: ‹(¬ *atm-is-in-conflict-st-heur* (− *x1c*) *x2b*) =
(− *x1a* ∉# *the* (*get-conflict-wl x2*))›
proof −
show *?thesis*
unfolding *is-in-conflict-st-def*[*symmetric*] *is-in-conflict-def*[*symmetric*]
apply (*subst Not-eq-iff*)
apply (*rule atm-is-in-conflict-st-heur-ana-is-in-conflict-st*[*THEN fref-to-Down-unRET-uncurry-Id*])
subgoal
using *confl-x2 x1c x1c-notin* **by** (*auto simp*: *uminus-*$\mathcal{A}_{in}$*-iff*)
subgoal
using *x2b-x2* **by** (*auto simp*: *lit-and-ann-of-propagated-st-heur-def*)
done
qed

lemma *ca-lit-and-ann-of-propagated-st-heur-pre*:
‹*lit-and-ann-of-propagated-st-heur-pre x2b*›
using *x1c xa-x′ confl-x2 trail-nempty*[*THEN neq-Nil-conv*[*THEN iffD1*]]
unfolding *xa x′ lit-and-ann-of-propagated-st-heur-pre-def*
by (*cases x2b*; *cases x2*)
(*auto simp add*: *twl-st-heur-conflict-ana-def*
trail-pol-def all-atms-def[*symmetric*] *ann-lits-split-reasons-def*)

lemma *atm-is-in-conflict-st-heur-pre*: ‹*atm-is-in-conflict-st-heur-pre* (− *x1c*, *x2b*)›
using *x1c xa-x′ confl-x2*
unfolding *atm-is-in-conflict-st-heur-pre-def xa x′*
by (*cases x2b*)
(*auto simp*: *twl-st-heur-conflict-ana-def option-lookup-clause-rel-def lookup-clause-rel-def*
atms-of-def all-atms-def)

context
  assumes *x1a-notin*: ‹− *x1a* ∉# *the* (*get-conflict-wl x2*)›
begin

lemma *tl-state-wl-heur-pre*: ‹*tl-state-wl-heur-pre x2b*›
  using *trail-nempty x2b-x2 xc x1c assert*
  unfolding *tl-state-wl-heur-pre-def*
  by (*cases x2b*; *cases* ‹*get-trail-wl x2*›)
    (*auto simp*: *twl-st-heur-conflict-ana-def lit-and-ann-of-propagated-st-heur-def*
      is-decided-no-proped-iff trail-pol-nempty
      all-atms-def[*symmetric*]
    intro: *isa-vmtf-le phase-saving-le isa-vmtf-next-search-le*
    intro!: *vmtf-unset-pre tl-trailt-tr-pre*
    dest!: *neq-Nil-conv*[*THEN iffD1*]
    dest: *multi-member-split*)

private lemma *tl-state-wl-pre*: ‹*tl-state-wl-pre x2*›
  using *trail-nempty x2b-x2 xc x1c assert hd-x2 x1c-notin x1a-notin not-tauto*
    dist-confl count-dec-not0 lits-trail
  unfolding *tl-state-wl-pre-def*
  by (*cases* ‹*get-trail-wl x2*›)
    (*auto simp*:
      phase-saving-def atms-of-def vmtf-def is-decided-no-proped-iff
      neq-Nil-conv image-image st)

private lemma *length-tl*: ‹*length* (*get-clauses-wl-heur* (*tl-state-wl-heur x2b*)) =
    length (*get-clauses-wl-heur x2b*)›
  by (*cases x2b*) (*auto simp*: *tl-state-wl-heur-def*)

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*))›
  using *x2b-x2 tl-state-wl-pre*
  by (*auto intro*!: *tl-state-wl-heur-tl-state-wl*[*THEN fref-to-Down-unRET*] *simp*: *length-tl*)

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*)›
  using *trail-nempty x2b-x2 xc x1c assert hd-x2 x1c-notin x1a-notin not-tauto*
    dist-confl count-dec-not0 confl-x2
  unfolding *maximum-level-removed-eq-count-dec-pre-def prod.simps*
  apply −
  apply (*intro conjI*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  done

lemma *skip-rel*:
  ‹((− *x1c*, *x2b*), − *x1a*, *x2*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-conflict-ana*›

using *x2b-x2*
by *auto*

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*)⟩
using *trail-nempty x2b-x2 xc x1c assert hd-x2 x1c-notin x1a-notin not-tauto*
dist-confl count-dec-not0 confl-x2 no-dup-x2 x1c not-dec-ge0 lits-trail
unfolding *update-confl-tl-wl-heur-pre-def lit-and-ann-of-propagated-st-heur-def*
by (*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-nempty atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *all-atms-def*[*symmetric*]
simp del: *isasat-input-bounded-def*
dest!: *neq-Nil-conv*[*THEN iffD1*]
intro: *phase-saving-le isa-vmtf-le isa-vmtf-next-search-le*)

private lemma *counts-maximum-level*:
⟨*get-count-max-lvls-heur x2b* ∈ *counts-maximum-level* (*get-trail-wl x2*) (*get-conflict-wl x2*)⟩
using *x2b-x2* **unfolding** *twl-st-heur-conflict-ana-def*
by *auto*

private lemma *card-max-lvl-ge0*:
⟨*Suc 0* ≤ *card-max-lvl* (*get-trail-wl x2*) (*the* (*get-conflict-wl x2*))⟩
using *counts-maximum-level confl-x2 max-lvl count-dec-not0*
get-maximum-level-exists-lit[*of*
⟨*get-maximum-level* (*get-trail-wl x2*) (*remove1-mset* (− *x1c*) (*the* (*get-conflict-wl x2*)))⟩
⟨(*get-trail-wl x2*)⟩ ⟨*remove1-mset* (− *x1c*) (*the* (*get-conflict-wl x2*))⟩]
unfolding *counts-maximum-level-def maximum-level-removed-eq-count-dec-def card-max-lvl-def*
get-maximum-level-remove-def
by (*auto dest*!: *in-diffD dest*: *multi-member-split*)

lemma *update-confl-tl-wl-pre*:
⟨*update-confl-tl-wl-pre* ((*x2a*, *x1a*), *x2*)⟩
using *trail-nempty x2b-x2 xc x1c assert hd-x2 x1c-notin x1a-notin not-tauto*
dist-confl count-dec-not0 confl-x2 no-dup-x2 x1c not-dec-ge0 lits-trail
x2c-dom lits lits-confl card-max-lvl-ge0 x1c-notin-tl ux1c-notin-tl not-tauto-x2c
dist-x2c not-tauto-resolved x1c-NC-0 x1c-watched
unfolding *update-confl-tl-wl-pre-def prod.simps*
by (*simp add*: *all-atms-def*[*symmetric*])

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*))⟩
using *x2b-x2* **by** *auto*

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 −
  have *H*[*refine0*]: ‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹
      ((*False*, *x*), *False*, *y*)
      ∈ *bool-rel* $\times_f$
        twl-st-heur-conflict-ana′ (*length* (*get-clauses-wl-heur* *x*))› **for** *x y*
    by *auto*

  show *?thesis*
    supply [[*goals-limit=1*]]
    unfolding *skip-and-resolve-loop-wl-D-heur-def skip-and-resolve-loop-wl-D-def*
      Let-def
      maximum-level-removed-eq-count-dec-def[*symmetric*]
      get-maximum-level-remove-def[*symmetric*]
    apply (*intro frefI nres-relI*)
    apply (*refine-vcg*
        update-confl-tl-wl-heur-update-confl-tl-wl[*THEN fref-to-Down-curry2*, *unfolded comp-def*]
        maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec
          [*THEN fref-to-Down-curry-no-nres-Id*]
       tl-state-wl-heur-tl-state-wl[*THEN fref-to-Down-no-nres*])
    subgoal by *auto*
    subgoal for *x y xa x′*
      apply (*cases x′*; *cases xa*)
      by (*rule sor-heur-inv*)
    subgoal
      by (*rule conflict-ana-same-cond*)
    subgoal by (*auto simp*:)
    subgoal by (*auto simp*:)
    subgoal by (*rule ca-lit-and-ann-of-propagated-st-heur-pre*)
    subgoal
      by (*rule atm-is-in-conflict-st-heur-pre*)
    subgoal for *x y xa x′ x1 x2 x1a x2a x1b x2b x1c x2c*
      by (*rule atm-is-in-conflict-st-heur-iff*)
    subgoal
      by (*rule tl-state-wl-heur-pre*)
    subgoal by (*rule tl-state-wl-heur-rel*)
    subgoal
      by (*rule maximum-level-removed-eq-count-dec-pre*)
    subgoal
      by (*rule skip-rel*)
    subgoal
      by (*rule update-confl-tl-wl-heur-pre*)
    subgoal
      by (*rule update-confl-tl-wl-pre*)
    subgoal
      by (*rule update-confl-tl-rel*)
    subgoal
      by *auto*
    subgoal
      by *auto*
    done
qed

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*)))›
by (*auto intro*!: *ext simp*: *is-decided-hd-trail-wl-heur-def*)

lemma *atm-of-in-atms-of*: ‹*atm-of x* $\in$ *atms-of C* $\longleftrightarrow$ *x* $\in\#$ *C* $\vee$ $-x$ $\in\#$ *C*›
using *atm-of-notin-atms-of-iff* **by** *blast*

definition *atm-is-in-conflict* **where**
‹*atm-is-in-conflict L D* $\longleftrightarrow$ *atm-of L* $\in$ *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*) $\longleftrightarrow$ *is-in-lookup-conflict* (*n*, *xs*) *L*›

lemma *is-in-option-lookup-conflict-atm-is-in-conflict-iff*:
assumes
‹*ba* $\neq$ *None*› **and** *aa*: ‹*aa* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *uaa*: ‹$-$ *aa* $\notin\#$ *the ba*› **and**
‹((*b*, *c*, *d*), *ba*) $\in$ *option-lookup-clause-rel* $\mathcal{A}$›
shows ‹*is-in-option-lookup-conflict aa* (*b*, *c*, *d*) =
atm-is-in-conflict aa ba›
proof $-$
obtain *yb* **where** *ba*[*simp*]: ‹*ba* = *Some yb*›
using *assms* **by** *auto*

have *map*: ‹*mset-as-position d yb*› **and** *le*: ‹$\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length d*› **and** [*simp*]: ‹$\neg$*b*›
using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)
have *aa-d*: ‹*atm-of aa* < *length d*› **and** *uaa-d*: ‹*atm-of* ($-$*aa*) < *length d*›
using *le aa* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
from *mset-as-position-in-iff-nth*[*OF map aa-d*]
have *1*: ‹(*aa* $\in\#$ *yb*) = (*d* ! *atm-of aa* = *Some* (*is-pos aa*))›
.

from *mset-as-position-in-iff-nth*[*OF map uaa-d*] **have** *2*: ‹(*d* ! *atm-of aa* $\neq$ *Some* (*is-pos* ($-$*aa*)))›
using *uaa* **by** *simp*

then show *?thesis*
using *uaa 1 2*
by (*auto simp*: *is-in-lookup-conflict-def is-in-option-lookup-conflict-def atm-is-in-conflict-def*
atm-of-in-atms-of is-neg-neg-not-is-neg
split: *option.splits*)
qed

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*))
$\in$ [λ(*L*, *D*). *D* $\neq$ *None* $\wedge$ *L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ $-L$ $\notin\#$ *the D*]$_f$
Id $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ $\rightarrow$ ⟨*bool-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*case-tac x*, *case-tac y*)
by (*simp add*: *is-in-option-lookup-conflict-atm-is-in-conflict-iff*[*of - -* $\mathcal{A}$])

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*)›

by (*auto intro*!: *ext simp*: *is-in-option-lookup-conflict-def*)

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*⟩
using *assms* **unfolding** *skip-and-resolve-loop-wl-D-heur-inv-def prod.simps*
by *blast*

lemma *isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv*:
⟨*isasat-fast x* $\Longrightarrow$
skip-and-resolve-loop-wl-D-heur-inv x
(*False*, *a2'*) $\Longrightarrow$ *isasat-fast a2'*⟩
unfolding *skip-and-resolve-loop-wl-D-heur-inv-def isasat-fast-def*
by *auto*

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* (λC. *the* (*snd C*)), *RETURN o mark-of*) $\in$
[λC. *is-proped C*]$_a$ *pair-nat-ann-lit-assn*^{k} $\rightarrow$ *nat-assn*⟩
apply *sepref-to-hoare*
apply (*case-tac x*; *case-tac xi*; *case-tac* ⟨*snd xi*⟩)
by (*sep-auto simp*: *nat-ann-lit-rel-def*)+

lemma *mark-of-fast-refine*[*sepref-fr-rules*]:
⟨(*return o* (λC. *the* (*snd C*)), *RETURN o mark-of*) $\in$
[λC. *is-proped C*]$_a$ *pair-nat-ann-lit-fast-assn*^{k} $\rightarrow$ *uint64-nat-assn*⟩
proof −
have *1*: ⟨*option-assn* ($\lambda a\ c. \uparrow ((c, a) \in$ *uint64-nat-rel*)) = *pure* (⟨*uint64-nat-rel*⟩*option-rel*)⟩
unfolding *option-assn-pure-conv*[*symmetric*]
by (*auto simp*: *pure-def*)
show *?thesis*
apply *sepref-to-hoare*
unfolding *1*
apply (*case-tac x*; *case-tac xi*; *case-tac* ⟨*snd xi*⟩)
apply (*sep-auto simp*: *br-def*)
apply (*sep-auto simp*: *nat-ann-lit-rel-def uint64-nat-rel-def br-def*
ann-lit-of-pair-if cong:)+
apply (*sep-auto simp*: *hr-comp-def*)
apply (*sep-auto simp*: *hr-comp-def uint64-nat-rel-def br-def*)
apply (*auto simp*: *nat-ann-lit-rel-def elim*: *option-relE*)[]
apply (*auto simp*: *ent-refl-true*)
done
qed

lemma *get-count-max-lvls-heur-hnr*[*sepref-fr-rules*]:
⟨(*return o get-count-max-lvls-code*, *RETURN o get-count-max-lvls-heur*) $\in$
isasat-unbounded-assn^k $\rightarrow_a$ *uint32-nat-assn*⟩
apply *sepref-to-hoare*
subgoal for *x x'*

by (*cases* x; *cases* x')
(*sep-auto simp*: *isasat-unbounded-assn-def get-count-max-lvls-code-def*
elim!: *mod-starE*)
done

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 $\rightarrow_a$ *uint32-nat-assn*›
apply *sepref-to-hoare*
subgoal for x x'
by (*cases* x; *cases* x')
(*sep-auto simp*: *isasat-bounded-assn-def get-count-max-lvls-code-def*
elim!: *mod-starE*)
done

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} $\rightarrow_a$ *bool-assn*›
unfolding *maximum-level-removed-eq-count-dec-heur-def*
by *sepref*

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} $\rightarrow_a$ *bool-assn*›
unfolding *maximum-level-removed-eq-count-dec-heur-def*
by *sepref*

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 $\rightarrow$ *bool-assn*›
unfolding *is-decided-hd-trail-wl-heur-alt-def isasat-unbounded-assn-def is-decided-hd-trail-wl-heur-pre-def*
by *sepref*

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} $\rightarrow$ *bool-assn*›
unfolding *is-decided-hd-trail-wl-heur-alt-def isasat-bounded-assn-def is-decided-hd-trail-wl-heur-pre-def*
by *sepref*

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 $\rightarrow$ (*unat-lit-assn* $*a$ *nat-assn*)›
supply [[*goals-limit=1*]]
supply *get-trail-wl-heur-def*[*simp*]
unfolding *lit-and-ann-of-propagated-st-heur-def isasat-unbounded-assn-def lit-and-ann-of-propagated-st-heur-pre-def*
by *sepref*

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*)⟩
supply [[*goals-limit=1*]]
supply *get-trail-wl-heur-def*[*simp*]
unfolding *lit-and-ann-of-propagated-st-heur-def isasat-bounded-assn-def lit-and-ann-of-propagated-st-heur-pre-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]] *if-splits*[*split*] *lit-of-last-trail-pol-def*[*simp*]
unfolding *tl-state-wl-heur-alt-def*[*abs-def*] *isasat-unbounded-assn-def get-trail-wl-heur-def*[*simp*]
vmtf-unset-def bind-ref-tag-def[*simp*] *short-circuit-conv*
unfolding *tl-state-wl-heur-pre-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]] *if-splits*[*split*] *lit-of-last-trail-pol-def*[*simp*]
unfolding *tl-state-wl-heur-alt-def*[*abs-def*] *isasat-bounded-assn-def get-trail-wl-heur-def*[*simp*]
vmtf-unset-def bind-ref-tag-def[*simp*] *short-circuit-conv lit-of-last-trail-pol-def*
unfolding *tl-state-wl-heur-pre-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *update-confl-tl-wl-heur-def isasat-unbounded-assn-def*
update-confl-tl-wl-heur-pre-def PR-CONST-def
two-uint64-nat-def[*symmetric*]
by *sepref*

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*)⟩
supply *four-uint32-hnr*[*sepref-fr-rules*] *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-mark-used-def four-uint32-def*[*symmetric*]

by *sepref*

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*›
using *isa-mark-used-fast-code2.refine*[*FCOMP isa-mark-used-mark-used*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

thm *isa-mark-used-code*
sepref-definition *update-confl-tl-wl-fast-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹[λ((*i*, *L*), *S*). *update-confl-tl-wl-heur-pre* ((*i*, *L*), *S*) ∧ *isasat-fast S*]$_a$
uint64-nat-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *bool-assn* ∗*a isasat-bounded-assn*›
supply [[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*]
unfolding *update-confl-tl-wl-heur-def isasat-bounded-assn-def*
update-confl-tl-wl-heur-pre-def PR-CONST-def
two-uint64-nat-def[*symmetric*]
by *sepref*

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*)›
:: ‹[λ(*L*, (*c*, *n*, *xs*)). *atm-of L* < *length xs*]$_a$
unat-lit-assn^k $*_a$ *conflict-option-rel-assn*^{k} → *bool-assn*›
unfolding *is-in-option-lookup-conflict-alt-def is-in-lookup-conflict-def PROTECT-def*
by *sepref*

sepref-definition *atm-is-in-conflict-st-heur-fast-code*
is ‹*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*)›
:: ‹[*atm-is-in-conflict-st-heur-pre*]$_a$ *unat-lit-assn*^{k} $*_a$ *isasat-unbounded-assn*^{k} → *bool-assn*›
unfolding *atm-is-in-conflict-st-heur-def atm-is-in-conflict-st-heur-pre-def isasat-unbounded-assn-def*
atm-in-conflict-lookup-def
by *sepref*

sepref-definition *atm-is-in-conflict-st-heur-code*
is ‹*uncurry* (*RETURN oo atm-is-in-conflict-st-heur*)›
:: ‹[*atm-is-in-conflict-st-heur-pre*]$_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} → *bool-assn*›
unfolding *atm-is-in-conflict-st-heur-def atm-is-in-conflict-st-heur-pre-def isasat-bounded-assn-def*
atm-in-conflict-lookup-def
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
supply [[*goals-limit=1*]]
skip-and-resolve-loop-wl-DI[*intro*]
unfolding *skip-and-resolve-loop-wl-D-heur-def*
apply (*rewrite at* ‹¬- ∧ ¬ -› *short-circuit-conv*)

by *sepref*

sepref-definition *skip-and-resolve-loop-wl-D-fast*
is ‹*skip-and-resolve-loop-wl-D-heur*›
:: ‹$[\lambda S.\ \textit{isasat-fast}\ S]_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
skip-and-resolve-loop-wl-DI[*intro*]
isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv[*intro*]
unfolding *skip-and-resolve-loop-wl-D-heur-def*
apply (*rewrite at* ‹¬- ∧ ¬ -› *short-circuit-conv*)
by *sepref*

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*))›
unfolding *length-ll-fs-heur-def*
apply (*intro ext*)
apply (*case-tac S*)
by *auto*

lemma (**in** −) *get-watched-wl-heur-def*: ‹*get-watched-wl-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -). *W*)›
by (*intro ext*, *rename-tac x*, *case-tac x*) *auto*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur b*) ≤ *uint64-max* ⟹
unit-propagation-inner-loop-wl-loop-D-heur-inv b a (*a1'*, *a1'a*, *a2'a*) ⟹
length (*get-clauses-wl-heur a2'a*) ≤ *uint64-max*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv-def*
by *auto*

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*) ≤ *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);
$\textit{WHILE}_T{}^{\textit{unit-propagation-inner-loop-wl-loop-D-heur-inv}\ S_0\ L}$
(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
})
b
}⟩
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def Let-def zero-uint64-nat-def* **..**

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* ≤ *uint32-max*);
L ← *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*), $-L$)
})
⟩
unfolding *select-and-remove-from-literals-to-update-wl-heur-def*
apply (*intro ext*)
apply (*rename-tac S*; *case-tac S*)
by (*auto intro*!: *ext simp*: *rev-trail-nth-def Let-def*)

definition *literals-to-update-wl-literals-to-update-wl-empty* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*literals-to-update-wl-literals-to-update-wl-empty S* ⟷
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')⟩
unfolding *literals-to-update-wl-literals-to-update-wl-empty-def isa-length-trail-def*
by (*auto intro*!: *ext split*: *prod.splits*)

lemma *unit-propagation-outer-loop-wl-D-invI*:
⟨*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S ⟹
isa-length-trail-pre (*get-trail-wl-heur S*)⟩
unfolding *unit-propagation-outer-loop-wl-D-heur-inv-def*
by *blast*

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
⟨*length* (*get-clauses-wl-heur x*) ≤ *uint64-max* ⟹
unit-propagation-outer-loop-wl-D-heur-inv x s' ⟹
length (*get-clauses-wl-heur* $a1'$) =
length (*get-clauses-wl-heur* s') ⟹
length (*get-clauses-wl-heur* s') ≤ *uint64-max*⟩
by (*auto simp*: *unit-propagation-outer-loop-wl-D-heur-inv-def*)

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} → *nat-assn*⟩

unfolding *length-ll-fs-heur-alt-def get-watched-wl-heur-def isasat-unbounded-assn-def*
length-ll-def[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

declare *length-ll-fs-heur-code.refine*[*sepref-fr-rules*]

definition *length-aa64-u32* :: ‹('*a*::*heap array-list64*) *array* ⇒ *uint32* ⇒ *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'*
<λr. *arrayO-assn* (*arl64-assn R*) *xs a* $*$ $\uparrow$ (*nat-of-uint64 r* = *length-ll xs b*)>$_t$›
unfolding *length-aa64-u32-def nth-u-code-def Array.nth'-def*
apply (*sep-auto simp flip*: *nat-of-uint32-code simp*: *br-def uint32-nat-rel-def length-ll-def*)
apply (*subst arrayO-except-assn-array0-index*[*symmetric*, *of b*])
apply (*simp add*: *nat-of-uint32-code br-def uint32-nat-rel-def*)
apply (*sep-auto simp*: *arrayO-except-assn-def*)
done

lemma *length-aa64-u32-hnr*[*sepref-fr-rules*]: ‹(*uncurry length-aa64-u32*, *uncurry* (*RETURN* ∘∘ *length-ll*)) ∈
[λ(*xs*, *i*). $i <$ *length xs*]$_a$ (*arrayO-assn* (*arl64-assn R*))^k $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint32-nat-rel-def br-def uint64-nat-rel-def*)

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*›
unfolding *length-ll-fs-heur-alt-def get-watched-wl-heur-def isasat-bounded-assn-def*
length-ll-def[*symmetric*]
supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
by *sepref*

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*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def PR-CONST-def*
unfolding *watched-by-nth-watched-app watched-app-def*[*symmetric*]
length-ll-fs-heur-def[*symmetric*]
unfolding *nth-ll-def*[*symmetric*]
unfolding *swap-ll-def*[*symmetric*]
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
is-None-def[*symmetric*] *get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
length-ll-fs-def[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

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*⟩
:: ⟨$[\lambda(L, S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \to uint64\text{-}nat\text{-}assn *a\ uint64\text{-}nat\text{-}assn *a\ isasat\text{-}bounded\text{-}assn$⟩
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def PR-CONST-def*
unfolding *watched-by-nth-watched-app watched-app-def*[*symmetric*]
length-ll-fs-heur-def[*symmetric*]
unfolding *nth-ll-def*[*symmetric*]
unfolding *swap-ll-def*[*symmetric*]
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
is-None-def[*symmetric*] *get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
length-ll-fs-def[*symmetric*] *zero-uint64-nat-def*[*symmetric*]
supply [[*goals-limit=1*]] *unit-propagation-inner-loop-wl-loop-D-heur-fast*[*intro*]
by *sepref*

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*⟩
:: ⟨$nat\text{-}assn^k *_a nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
$isasat\text{-}unbounded\text{-}assn^d \to_a isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
unfolding *cut-watch-list-heur2-def isasat-unbounded-assn-def length-ll-def*[*symmetric*]
update-ll-def[*symmetric*] *nth-rll-def*[*symmetric*] *shorten-take-ll-def*[*symmetric*]
by *sepref*

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*)) ∈
$[\lambda((L, j), W).\ j \leq length\ (W\ !\ L) \land L < length\ W]_a$
$uint32\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a (arrayO\text{-}assn\ (arl64\text{-}assn\ R))^d \to arrayO\text{-}assn\ (arl64\text{-}assn\ R)$⟩
unfolding *shorten-take-aa64-u32-def shorten-take-ll-def nth-u-code-def Array.nth′-def*
comp-def nat-of-uint32-code[*symmetric*] *Array-upd-u-def*
by *sepref-to-hoare* (*sep-auto simp*: *uint32-nat-rel-def br-def uint64-nat-rel-def*)

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*⟩
:: ⟨$[\lambda(((j, w), L), S).\ length\ (watched\text{-}by\text{-}int\ S\ L) \leq uint64\text{-}max-4]_a$
$uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
$isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]] *length-ll-def*[*simp*] *uint64-max-def*[*simp*] *length-rll-def*[*simp*]

unfolding *cut-watch-list-heur2-def isasat-bounded-assn-def length-ll-def*[*symmetric*]
update-ll-def[*symmetric*] *nth-rll-def*[*symmetric*] *shorten-take-ll-def*[*symmetric*]
one-uint64-nat-def[*symmetric*] *length-rll-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *PR-CONST-def unit-propagation-inner-loop-wl-D-heur-def*
by *sepref*

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*⟩
:: ⟨$[\lambda(L, S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]] *nat-of-uint64-conv-hnr*[*sepref-fr-rules*]
unfolding *PR-CONST-def unit-propagation-inner-loop-wl-D-heur-def*
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ unat\text{-}lit\text{-}assn$⟩
supply [[*goals-limit=1*]] *uint32-nat-assn-plus*[*sepref-fr-rules*]
unfolding *select-and-remove-from-literals-to-update-wl-heur-alt-def isasat-unbounded-assn-def*
one-uint32-nat-def[*symmetric*]
supply [[*goals-limit = 1*]]
by *sepref*

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*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn *a\ unat\text{-}lit\text{-}assn$⟩
supply [[*goals-limit=1*]] *uint32-nat-assn-plus*[*sepref-fr-rules*]
unfolding *select-and-remove-from-literals-to-update-wl-heur-alt-def isasat-bounded-assn-def*
one-uint32-nat-def[*symmetric*]
supply [[*goals-limit = 1*]]
by *sepref*

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*⟩
:: ⟨$[\lambda S.\ isa\text{-}length\text{-}trail\text{-}pre\ (get\text{-}trail\text{-}wl\text{-}heur\ S)]_a\ isasat\text{-}unbounded\text{-}assn^k \rightarrow bool\text{-}assn$⟩
unfolding *literals-to-update-wl-literals-to-update-wl-empty-alt-def*
isasat-unbounded-assn-def
by *sepref*

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-length-trail-pre* (*get-trail-wl-heur S*)$]_a$ *isasat-bounded-assn*^{k} $\rightarrow$ *bool-assn*›
  unfolding *literals-to-update-wl-literals-to-update-wl-empty-alt-def*
    isasat-bounded-assn-def
  by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*›
  supply [[*goals-limit=1*]]
    unit-propagation-outer-loop-wl-D-invI[*intro*]
  unfolding *unit-propagation-outer-loop-wl-D-heur-def*
    literals-to-update-wl-literals-to-update-wl-empty-def[*symmetric*]
  by *sepref*

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-clauses-wl-heur S*) $\leq$ *uint64-max*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]] *unit-propagation-outer-loop-wl-D-heur-fast*[*intro*]
    unit-propagation-outer-loop-wl-D-invI[*intro*]
  unfolding *unit-propagation-outer-loop-wl-D-heur-def*
    literals-to-update-wl-literals-to-update-wl-empty-def[*symmetric*]
  by *sepref*

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-None s* $\Longrightarrow$ *s* $\neq$ *None*›
  by (*auto split*: *option.splits*)

definition *vmtf-find-next-undef-upd*
  :: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*)*ann-lits* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$
      (((*nat*,*nat*)*ann-lits* $\times$ *vmtf-remove-int*) $\times$ *nat option*)*nres*›
where
  ‹*vmtf-find-next-undef-upd* $\mathcal{A}$ = (λ*M vm. do*{
      L $\leftarrow$ *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* $\mathcal{A}$) $\to_f$
    ⟨*trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
unfolding *isa-vmtf-find-next-undef-upd-def vmtf-find-next-undef-upd-def uncurry-def*
defined-atm-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*refine-rcg isa-vmtf-find-next-undef-vmtf-find-next-undef*[*THEN fref-to-Down-curry*])
subgoal by *auto*
subgoal by (*auto simp*: *update-next-search-def split*: *prod.splits*)
done

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*)›
by (*auto simp*: *lit-of-found-atm-D-pre-def phase-saving-def*
atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff dest*: *bspec*[*of* - - ‹*Pos* (*the L*)›])

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* ∧

$(T, U) \in$ *twl-st-l None* $\wedge$
twl-struct-invs U $\wedge$
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* S) S $\wedge$
get-conflict-wl S = *None*
)›

lemma *vmtf-find-next-undef-upd*:
‹(*uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$), *uncurry* (*find-undefined-atm* $\mathcal{A}$)) $\in$
[$\lambda(M, vm)$. $vm \in$ *vmtf* $\mathcal{A}$ M]$_f$ *Id* $\times_f$ *Id* $\rightarrow$ ⟨*Id* $\times_f$ *Id* $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel*›
unfolding *vmtf-find-next-undef-upd-def find-undefined-atm-def*
update-next-search-def uncurry-def
apply (*intro frefI nres-relI*)
apply (*clarify*)
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule vmtf-find-next-undef-ref*[*simplified*])
by (*auto intro*!: *RETURN-SPEC-refine simp*: *image-image defined-atm-def*[*symmetric*])

lemma *find-unassigned-lit-wl-D′-find-unassigned-lit-wl-D*:
‹(*find-unassigned-lit-wl-D-heur*, *find-unassigned-lit-wl-D*) $\in$
[*find-unassigned-lit-wl-D-heur-pre*]$_f$
twl-st-heur‴ r $\rightarrow$ ⟨{((T, L), (T', L')). (T, T') $\in$ *twl-st-heur‴* r $\wedge$ $L = L'$ $\wedge$
($L \neq$ *None* $\longrightarrow$ *undefined-lit* (*get-trail-wl* T') (*the* L) $\wedge$ *the* $L \in\#$ $\mathcal{L}_{all}$ (*all-atms-st* T')) $\wedge$
get-conflict-wl T' = *None*}⟩*nres-rel*›
proof −
have [*simp*]: ‹*undefined-lit* M (*Pos* (*atm-of* y)) = *undefined-lit* M y› **for** M y
by (*auto simp*: *defined-lit-map*)
have [*simp*]: ‹*defined-atm* M (*atm-of* y) = *defined-lit* M y› **for** M y
by (*auto simp*: *defined-lit-map defined-atm-def*)

have *ID-R*: ‹*Id* $\times_r$ ⟨*Id*⟩*option-rel* = *Id*›
by *auto*
have *atms*: ‹*atms-of* ($\mathcal{L}_{all}$ (*all-atms-st* (M, N, D, *NE*, *UE*, *WS*, Q))) =
atms-of-mm (*mset* ‘# *init-clss-lf* N) $\cup$
atms-of-mm NE $\wedge$ D = *None*› (**is** *?A*) **and**
atms-2: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* N (*NE* + *UE*))) = *set-mset* ($\mathcal{L}_{all}$ (*all-atms* N *NE*))› (**is** *?B*) **and**
atms-3: ‹$y \in$ *atms-of-ms* ((λx. *mset* (*fst* x)) ‘ *set-mset* (*ran-m* N)) $\Longrightarrow$
$y \notin$ *atms-of-mm NE* $\Longrightarrow$
$y \in$ *atms-of-ms* ((λx. *mset* (*fst* x)) ‘ {a. $a \in\#$ *ran-m* N $\wedge$ *snd* a})› (**is** ‹*?C1* $\Longrightarrow$ *?C2* $\Longrightarrow$ *?C*›)
if *inv*: ‹*find-unassigned-lit-wl-D-heur-pre* (M, N, D, *NE*, *UE*, *WS*, Q)›
for M N D *NE UE WS* Q y
proof −
obtain T U **where**
S-T: ‹((M, N, D, *NE*, *UE*, *WS*, Q), T) $\in$ *state-wl-l None*› **and**
T-U: ‹(T, U) $\in$ *twl-st-l None*› **and**
inv: ‹*twl-struct-invs* U› **and**
$\mathcal{A}_{in}$: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (M, N, D, *NE*, *UE*, *WS*, Q)) (M, N, D, *NE*, *UE*, *WS*, Q)› **and**
confl: ‹*get-conflict-wl* (M, N, D, *NE*, *UE*, *WS*, Q) = *None*›
using *inv* **unfolding** *find-unassigned-lit-wl-D-heur-pre-def*
apply − **apply** *normalize-goal+*
by *blast*

have ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of* U)› **and**
unit: ‹*entailed-clss-inv* U›
using *inv* **unfolding** *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast+*

then show *?A*
using $\mathcal{A}_{in}$ *confl S-T T-U* **unfolding** *is-*$\mathcal{L}_{all}$*-alt-def state-wl-l-def twl-st-l-def literals-are-*$\mathcal{L}_{in}$*-def all-atms-def all-lits-def*
apply −
apply (*subst* (*asm*) *all-clss-l-ran-m*[*symmetric*], *subst* (*asm*) *image-mset-union*)+
apply (*subst all-clss-l-ran-m*[*symmetric*], *subst image-mset-union*)
by (*auto simp*: *cdcl*$_W$*-restart-mset.no-strange-atm-def entailed-clss-inv.simps mset-take-mset-drop-mset mset-take-mset-drop-mset′ atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *clauses-def simp del*: *entailed-clss-inv.simps*)
then show *?B* **and** ‹*?C1* $\Longrightarrow$ *?C2* $\Longrightarrow$ *?C*›
apply −
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *all-atms-def all-lits-def*
apply (*subst* (*asm*) *all-clss-l-ran-m*[*symmetric*], *subst* (*asm*) *image-mset-union*)+
apply (*subst all-clss-l-ran-m*[*symmetric*], *subst image-mset-union*)+
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff all-lits-of-mm-union atms-of-def* $\mathcal{L}_{all}$*-union image-Un atm-of-eq-atm-of atm-of-all-lits-of-mm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
qed

have [*dest*]: ‹$(S,\ T) \in$ *twl-st-heur* $\Longrightarrow \varphi =$ *get-phase-saver-heur S* $\Longrightarrow$ *phase-saving* (*all-atms-st T*) φ› **for** $S\ T\ \varphi$
by (*auto simp*: *twl-st-heur-def all-atms-def*)
define *unassigned-atm* **where**
‹*unassigned-atm S L* $\equiv \exists$ *M N D NE UE WS Q*.
$S = (M,\ N,\ D,\ NE,\ UE,\ WS,\ Q) \wedge$
$(L \neq None \longrightarrow$
undefined-lit M (*the L*) $\wedge$ *the L* $\in\#$ $\mathcal{L}_{all}$ (*all-atms N NE*) $\wedge$
atm-of (*the L*) $\in$ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *init-clss-lf N* + *NE*)) $\wedge$
$(L = None \longrightarrow (\nexists L'.$ *undefined-lit M* $L' \wedge$
atm-of $L' \in$ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *init-clss-lf N* + *NE*)))›
for *L* :: ‹*nat literal option*› **and** *S* :: ‹*nat twl-st-wl*›
have *find-unassigned-lit-wl-D-alt-def*:
‹*find-unassigned-lit-wl-D S* = *do* {
$L \leftarrow$ *SPEC*(*unassigned-atm S*);
$L \leftarrow$ *RES* {*L*, *map-option uminus L*};
SPEC(λ((*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*), *L′*).
$S = (M,\ N,\ D,\ NE,\ UE,\ WS,\ Q) \wedge L = L'$)
}› **for** *S*
unfolding *find-unassigned-lit-wl-D-def RES-RES-RETURN-RES unassigned-atm-def RES-RES-RETURN-RES*
by (*cases S*) (*auto simp*: *mset-take-mset-drop-mset′ uminus-*$\mathcal{A}_{in}$*-iff*)

have *isa-vmtf-find-next-undef-upd*:
‹*isa-vmtf-find-next-undef-upd* (*a*, *aa*, *ab*, *ac*, *ad*, *b*)
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*)
$\leq \Downarrow$ {(((*M*, *vm*), *A*), *L*). *A* = *map-option atm-of L* $\wedge$
unassigned-atm (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*) *L* $\wedge$
vm $\in$ *isa-vmtf* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)) *bt* $\wedge$
$(L \neq None \longrightarrow$ *the A* $\in\#$ *all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)) $\wedge$
(*M*, *bt*) $\in$ *trail-pol* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*))}
(*SPEC* (*unassigned-atm* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)))›
(**is** ‹- $\leq \Downarrow$ *?find* -›)
if
pre: ‹*find-unassigned-lit-wl-D-heur-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)› **and**
T: ‹(((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,

(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*),
bt, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)
$\in$ *twl-st-heur*› **and**
‹*r* =
length
(*get-clauses-wl-heur*
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*))›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
bw bx by bz
proof −
let *?*$\mathcal{A}$ = ‹*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)›
have *pol*:
‹((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *bt*) $\in$ *trail-pol* (*all-atms bu* (*bw* + *bx*))›
using *that* **by** (*auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*])
obtain *vm0* **where**
vm0: ‹((*an*, *bc*), *vm0*) $\in$ *distinct-atoms-rel* (*all-atms bu* (*bw* + *bx*))› **and**
vm: ‹((*aj*, *ak*, *al*, *am*, *bb*), *vm0*) $\in$ *vmtf* (*all-atms bu* (*bw* + *bx*)) *bt*›
using *T* **by** (*auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*] *isa-vmtf-def*)
have [*intro*]: ‹*Multiset.Ball* ($\mathcal{L}_{all}$ (*all-atms bu* (*bw* + *bx*))) (*defined-lit bt*) $\Longrightarrow$
atm-of L'
$\in$ *atms-of-ms* ((λ*x*. *mset* (*fst x*)) ‘ {*a*. *a* $\in$# *ran-m bu* $\wedge$ *snd a*}) $\Longrightarrow$
undefined-lit bt L' $\Longrightarrow$ *False*›
‹*Multiset.Ball* ($\mathcal{L}_{all}$ (*all-atms bu* (*bw* + *bx*))) (*defined-lit bt*) $\Longrightarrow$
atm-of L'
$\in$ *atms-of-mm bw* $\Longrightarrow$
undefined-lit bt L' $\Longrightarrow$ *False*›
‹*Multiset.Ball* ($\mathcal{L}_{all}$ (*all-atms bu* (*bw* + *bx*))) (*defined-lit bt*) $\Longrightarrow$
atm-of L'
$\in$ *atms-of-mm bx* $\Longrightarrow$
undefined-lit bt L'$\Longrightarrow$ *False*› **for** *L'*
by (*auto simp*: *all-atms-def atms-of-ms-def atm-of-eq-atm-of all-lits-def*
all-lits-of-mm-union ran-m-def all-lits-of-mm-add-mset $\mathcal{L}_{all}$*-union*
eq-commute[*of* - ‹*the* (*fmlookup* - -)›] $\mathcal{L}_{all}$*-atm-of-all-lits-of-m*
atms-of-def
dest!: *multi-member-split*
)

show *?thesis*
apply (*rule order.trans*)
apply (*rule isa-vmtf-find-next-undef-vmtf-find-next-undef*[*of ?*$\mathcal{A}$, *THEN fref-to-Down-curry*,
of - - *bt* ‹((*aj*, *ak*, *al*, *am*, *bb*), *vm0*)›])
subgoal by *fast*
subgoal
using *pol vm0* **by** (*auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*])
apply (*rule order.trans*)
apply (*rule ref-two-step'*)
apply (*rule vmtf-find-next-undef-upd*[*THEN fref-to-Down-curry*, *of ?*$\mathcal{A}$ *bt* ‹((*aj*, *ak*, *al*, *am*, *bb*),
vm0)›])
subgoal using *vm* **by** (*auto simp*: *all-atms-def*)
subgoal by *auto*
subgoal using *vm vm0 pre*

apply (*auto 5 0 simp*: *find-undefined-atm-def unassigned-atm-def conc-fun-RES all-atms-def*[*symmetric*]
mset-take-mset-drop-mset′ atms-2 defined-atm-def
intro!: *RES-refine intro*: *isa-vmtfI*)
apply (*auto intro*: *isa-vmtfI simp*: *defined-atm-def atms-2*)
apply (*rule-tac x* = ‹*Some* (*Pos y*)› **in** *exI*)
apply (*auto intro*: *isa-vmtfI simp*: *defined-atm-def atms-2* *in*-$\mathcal{L}_{all}$-*atm-of*-$\mathcal{A}_{in}$
in-set-all-atms-iff atms-3)
done
done
qed

have *lit-of-found-atm*: ‹*lit-of-found-atm ao x2a*
$\leq \Downarrow \{(L, L'). L = L' \wedge$ *map-option atm-of* $L = x2a\}$
(*RES* {*L*, *map-option uminus L*})›
if
‹*find-unassigned-lit-wl-D-heur-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)› **and**
‹(((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*),
bt, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)
∈ *twl-st-heur*› **and**
‹*r* =
length
(*get-clauses-wl-heur*
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*))› **and**
‹(*x*, *L*) ∈ *?find bt bu bv bw bx by bz*› **and**
‹*x1* = (*x1a*, *x2*)› **and**
‹*x* = (*x1*, *x2a*)›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
bw bx by bz x L x1 x1a x2 x2a
proof −
show *?thesis*
using *that* **unfolding** *lit-of-found-atm-def*
by (*auto simp*: *atm-of-eq-atm-of twl-st-heur-def intro*!: *RES-refine*)
qed
show *?thesis*
unfolding *find-unassigned-lit-wl-D-heur-def find-unassigned-lit-wl-D-alt-def find-undefined-atm-def*
ID-R
apply (*intro frefI nres-relI*)
apply *clarify*
apply *refine-vcg*
apply (*rule isa-vmtf-find-next-undef-upd*; *assumption*)
subgoal
by (*rule lit-of-found-atm-D-pre*)
(*auto simp add*: *twl-st-heur-def in*-$\mathcal{L}_{all}$-*atm-of-in-atms-of-iff Ball-def image-image*
mset-take-mset-drop-mset′ atms all-atms-def[*symmetric*] *unassigned-atm-def*
simp del: *twl-st-of-wl.simps dest*!: *atms intro*!: *RETURN-RES-refine*)
apply (*rule lit-of-found-atm*; *assumption*)
subgoal
by (*auto simp add*: *twl-st-heur-def in*-$\mathcal{L}_{all}$-*atm-of-in-atms-of-iff Ball-def image-image*
mset-take-mset-drop-mset′ atms all-atms-def[*symmetric*] *unassigned-atm-def*

atm-of-eq-atm-of
simp del: *twl-st-of-wl.simps dest*!: *atms intro*!: *RETURN-RES-refine*)
done
qed

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*›
apply (*intro frefI nres-relI*)
unfolding *lit-of-found-atm-D-def lit-of-found-atm-def*
by (*auto split*: *option.splits*)

definition *decide-lit-wl-heur* :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*decide-lit-wl-heur* = (λ*L'* (*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*) ∈ *twl-st-heur'''* *r* $\to_f$ ⟨*bool-rel* $\times_f$ *twl-st-heur'''* *r*⟩ *nres-rel*›
proof −
have [*simp*]:
‹*rev* (*cons-trail-Decided L M*) = *rev M* @ [*Decided L*]›
‹*no-dup* (*cons-trail-Decided L M*) = *no-dup* (*Decided L* # *M*)›
‹*isa-vmtf* 𝒜 (*cons-trail-Decided L M*) = *isa-vmtf* 𝒜 (*Decided L* # *M*)›

for $M\ L\ \mathcal{A}$
by (*auto simp*: *cons-trail-Decided-def*)

have *final*: ‹*decide-lit-wl-heur xb x1a*
$\leq$ *SPEC*
(λT. *RETURN* (*False*, *T*)
$\leq$ *SPEC*
(λc. (*c*, *False*, *decide-lit-wl x'a x1*)
$\in$ *bool-rel* $\times_f$ *twl-st-heur''' r*))›
if
‹(*x*, *y*) $\in$ *twl-st-heur''' r*› **and**
‹*decide-wl-or-skip-pre y* $\wedge$ *literals-are-*$\mathcal{L}_{in}$ (*all-atms-st y*) *y*› **and**
‹(*xa*, *x'*)
$\in$ {((*T*, *L*), *T'*, *L'*).
(*T*, *T'*) $\in$ *twl-st-heur''' r* $\wedge$
L = *L'* $\wedge$
(*L* $\neq$ *None* $\longrightarrow$
undefined-lit (*get-trail-wl T'*) (*the L*) $\wedge$
the L $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st T'*)) $\wedge$
get-conflict-wl T' = *None*}› **and**
st:
‹*x'* = (*x1*, *x2*)›
‹*xa* = (*x1a*, *x2a*)›
‹*x2a* = *Some xb*›
‹*x2* = *Some x'a*› **and**
‹(*xb*, *x'a*) $\in$ *nat-lit-lit-rel*›
for *x y xa x' x1 x2 x1a x2a xb x'a*
proof −
show *?thesis*
unfolding *decide-lit-wl-heur-def*
decide-lit-wl-def
apply (*cases x1a*)
apply *refine-vcg*
subgoal
by (*rule isa-length-trail-pre*[*of* - ‹*get-trail-wl x1*› ‹*all-atms-st x1*›])
(*use that*(*3*) **in** ‹*auto simp*: *twl-st-heur-def st all-atms-def*[*symmetric*]›)
subgoal
by (*rule cons-trail-Decided-tr-pre*[*of* - ‹*get-trail-wl x1*› ‹*all-atms-st x1*›])
(*use that*(*3*) **in** ‹*auto simp*: *twl-st-heur-def st all-atms-def*[*symmetric*]›)
subgoal
using *that*(*2*,*3*) **unfolding** *cons-trail-Decided-def*[*symmetric*] *st*
by (*auto simp add*: *twl-st-heur-def all-atms-def*[*symmetric*]
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*] *out-learned-def*
intro!: *cons-trail-Decided-tr*[*THEN fref-to-Down-unRET-uncurry*]
isa-vmtf-consD2)
done
qed

show *?thesis*
supply [[*goals-limit*=*1*]]
unfolding *decide-wl-or-skip-D-heur-def decide-wl-or-skip-D-def decide-wl-or-skip-D-pre-def*
decide-l-or-skip-pre-def twl-st-of-wl.simps[*symmetric*]
apply (*intro nres-relI frefI same-in-Id-option-rel*)
apply (*refine-vcg find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D*[*of r*, *THEN fref-to-Down*])
subgoal
unfolding *decide-wl-or-skip-pre-def find-unassigned-lit-wl-D-heur-pre-def*

decide-wl-or-skip-pre-def decide-l-or-skip-pre-def
apply *normalize-goal*+
apply (*rule-tac x* = *xa* **in** *exI*)
apply (*rule-tac x* = *xb* **in** *exI*)
apply *auto*
done
apply (*rule same-in-Id-option-rel*)
subgoal by (*auto simp del*: *simp*: *twl-st-heur-def*)
subgoal by (*auto simp del*: *simp*: *twl-st-heur-def*)
apply (*rule final*; *assumption*)
done
qed

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*›
supply [[*goals-limit=1*]]
supply *not-is-None-not-None*[*simp*]
unfolding *isa-vmtf-find-next-undef-def PR-CONST-def*
apply (*rewrite at* ‹*WHILE*$_T$$^-$ (λ - . ¤) - -› *short-circuit-conv*)
by *sepref*

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*›
supply [[*goals-limit=1*]]
supply *not-is-None-not-None*[*simp*]
unfolding *isa-vmtf-find-next-undef-def PR-CONST-def*
apply (*rewrite at* ‹*WHILE*$_T$$^-$ (λ - . ¤) - -› *short-circuit-conv*)
by *sepref*

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›
supply [[*goals-limit=1*]]
supply *not-is-None-not-None*[*simp*]
unfolding *isa-vmtf-find-next-undef-upd-def PR-CONST-def*
by *sepref*

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⟩
supply [[*goals-limit=1*]]
supply *not-is-None-not-None*[*simp*]
unfolding *isa-vmtf-find-next-undef-upd-def PR-CONST-def*
by *sepref*

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*]$_a$
($array\text{-}assn\ bool\text{-}assn)^k *_a (option\text{-}assn\ uint32\text{-}nat\text{-}assn)^d \rightarrow$
option-assn unat-lit-assn⟩
supply [[*goals-limit=1*]]
supply *not-is-None-not-None*[*simp*] *Pos-unat-lit-assn′*[*sepref-fr-rules*]
Neg-unat-lit-assn′[*sepref-fr-rules*]
unfolding *lit-of-found-atm-D-def PR-CONST-def lit-of-found-atm-D-pre-def*
by *sepref*

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*)
∈ [*lit-of-found-atm-D-pre*]$_a$
$phase\text{-}saver\text{-}conc^k *_a (option\text{-}assn\ uint32\text{-}nat\text{-}assn)^d \rightarrow$
option-assn unat-lit-assn⟩
using *lit-of-found-atm-D-code.refine*[*FCOMP lit-of-found-atm-D-lit-of-found-atm*[*unfolded convert-fref*]]
by *simp*

sepref-register *find-undefined-atm*
sepref-definition *find-unassigned-lit-wl-D-code*
is ⟨*find-unassigned-lit-wl-D-heur*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a$ (*isasat-unbounded-assn* ∗*a* *option-assn unat-lit-assn*)⟩
supply [[*goals-limit=1*]]
unfolding *find-unassigned-lit-wl-D-heur-def isasat-unbounded-assn-def PR-CONST-def*
by *sepref*

sepref-definition *find-unassigned-lit-wl-D-fast-code*
is ⟨*find-unassigned-lit-wl-D-heur*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^d \rightarrow_a$ (*isasat-bounded-assn* ∗*a* *option-assn unat-lit-assn*)⟩
supply [[*goals-limit=1*]]
unfolding *find-unassigned-lit-wl-D-heur-def isasat-bounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *decide-lit-wl-heur-def isasat-unbounded-assn-def PR-CONST-def*
cons-trail-Decided-def[*symmetric*]

by *sepref*

sepref-definition *decide-lit-wl-fast-code*
is ⟨*uncurry decide-lit-wl-heur*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *decide-lit-wl-heur-def isasat-bounded-assn-def PR-CONST-def*
cons-trail-Decided-def[*symmetric*]
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a bool\text{-}assn *a\ isasat\text{-}unbounded\text{-}assn$⟩
unfolding *decide-wl-or-skip-D-heur-def PR-CONST-def*
supply [[*goals-limit = 1*]]
find-unassigned-lit-wl-D-def[*simp*] *image-image*[*simp*]
by *sepref*

sepref-definition *decide-wl-or-skip-D-fast-code*
is ⟨*decide-wl-or-skip-D-heur*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^d \rightarrow_a bool\text{-}assn *a\ isasat\text{-}bounded\text{-}assn$⟩
unfolding *decide-wl-or-skip-D-heur-def PR-CONST-def*
supply [[*goals-limit = 1*]]
find-unassigned-lit-wl-D-def[*simp*] *image-image*[*simp*]
by *sepref*

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
by *standard*
(*auto simp*: *shows-prec-uint64-def shows-list-uint64-def*
shows-prec-append shows-list-append)
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
by *standard*
(*auto simp*: *shows-prec-uint32-def shows-list-uint32-def*
shows-prec-append shows-list-append)
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*›
  by (*intro frefI nres-relI*)
   (*auto simp*: *twl-st-heur-def isasat-current-status-def*)

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*));

$U \leftarrow$ *backtrack-wl-D-nlit-heur* T;
$U \leftarrow$ *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. $f \in$ *twl-st-heur''* $\mathcal{D}$ $r \rightarrow_f$ ⟨*twl-st-heur''* $\mathcal{D}$ r⟩ *nres-rel*)›
shows ‹$f \in$ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- $\in$ *?A B*›)
proof −
obtain *f1 f2* **where** f: ‹f = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
unfolding f
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
using *assms*[*of* ‹*dom-m* (*get-clauses-wl y*)› ‹*length* (*get-clauses-wl-heur x*)›,
unfolded ref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
rule-format] **unfolding** f
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*drule spec*[*of* - *x*])
apply (*drule spec*[*of* - *y*])
apply *simp*
apply (*rule weaken-⇓'*[*of* - ‹*twl-st-heur''* (*dom-m* (*get-clauses-wl y*))
(*length* (*get-clauses-wl-heur x*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *twl-st-heur'''D-twl-st-heurD*:
assumes H: ‹($\bigwedge r$. $f \in$ *twl-st-heur'''* $r \rightarrow_f$ ⟨*twl-st-heur'''* r⟩ *nres-rel*)›
shows ‹$f \in$ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- $\in$ *?A B*›)
proof −
obtain *f1 f2* **where** f: ‹f = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
unfolding f
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
using *assms*[*of* ‹*length* (*get-clauses-wl-heur x*)›,
unfolded ref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
rule-format] **unfolding** f
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*drule spec*[*of* - *x*])
apply (*drule spec*[*of* - *y*])
apply *simp*
apply (*rule weaken-⇓'*[*of* - ‹*twl-st-heur'''* (*length* (*get-clauses-wl-heur x*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done

  done
qed

lemma *twl-st-heur'''D-twl-st-heurD-prod*:
  assumes *H*: ‹($\bigwedge$*r*. *f* ∈ *twl-st-heur''' r* $\rightarrow_f$ ⟨*A* $\times_r$ *twl-st-heur''' r*⟩ *nres-rel*)›
  shows ‹*f* ∈ *twl-st-heur* $\rightarrow_f$ ⟨*A* $\times_r$ *twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
proof −
  obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
    by (*cases f*) *auto*
  show *?thesis*
    unfolding *f*
    apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
    apply (*intro conjI impI allI*)
    subgoal for *x y*
      using *assms*[*of* ‹*length* (*get-clauses-wl-heur x*)›,
        unfolded ref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
        rule-format] **unfolding** *f*
      apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
      apply (*drule spec*[*of - x*])
      apply (*drule spec*[*of - y*])
      apply *simp*
      apply (*rule weaken-⇓'*[*of -* ‹*A* $\times_r$ *twl-st-heur'''* (*length* (*get-clauses-wl-heur x*))›])
      apply (*fastforce simp*: *twl-st-heur'-def*)+
      done
    done
qed

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*} $\rightarrow_f$
      ⟨*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 −
  have *H*: ‹(*x*, *y*) ∈ {(*S*, *T*).
        (*S*, *T*) ∈ *twl-st-heur* ∧
        length (*get-clauses-wl-heur S*) =
        length (*get-clauses-wl-heur x*)} ⟹
      (*x*, *y*)
      ∈ {(*S*, *T*).
        (*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧
        length (*get-clauses-wl-heur S*) =
        length (*get-clauses-wl-heur x*)}› **for** *x y*
    by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
  show *?thesis*
    unfolding *cdcl-twl-o-prog-wl-D-heur-def cdcl-twl-o-prog-wl-D-def*
      get-conflict-wl-is-None
    apply (*intro frefI nres-relI*)
    apply (*refine-vcg*
      decide-wl-or-skip-D-heur-decide-wl-or-skip-D[**where** *r*=*r*, *THEN fref-to-Down*, *THEN order-trans*]
        skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
        backtrack-wl-D-nlit-backtrack-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
        isasat-current-status-id[*THEN fref-to-Down*, *THEN order-trans*])
    subgoal
      by (*auto simp*: *twl-st-heur-state-simp*
          get-conflict-wl-is-None-heur-get-conflict-wl-is-None[*THEN fref-to-Down-unRET-Id*])

apply (*assumption*)
subgoal by (*rule conc-fun-R-mono*) *auto*
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-count-decided-st-alt-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
apply *assumption*
subgoal by (*auto simp*: *conc-fun-RES RETURN-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
done
qed

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*} $\rightarrow_f$
⟨*bool-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*}⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rule cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*[*THEN fref-to-Down*, *THEN order-trans*])
apply (*auto intro*!: *conc-fun-R-mono*)
done

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*›
using *twl-st-heur''D-twl-st-heurD*[*OF*
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D']
.

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 −
have *H*: ‹(*x*, *y*) ∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) =
length (*get-clauses-wl-heur x*)} ⟹
(*x*, *y*)
∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧

length (*get-clauses-wl-heur* S) =
length (*get-clauses-wl-heur* x)}› **for** x y
by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
show *?thesis*
unfolding *cdcl-twl-stgy-prog-wl-D-heur-def cdcl-twl-stgy-prog-wl-D-def*
apply (*intro frefI nres-relI*)
subgoal for x y
apply (*refine-vcg*
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN twl-st-heur''D-twl-st-heurD*, *THEN fref-to-Down*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[*THEN fref-to-Down*])
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur'-def*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
done
done
qed

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*›
by *sepref-to-hoare* (*sep-auto simp*: *isasat-information-banner-code-def isasat-information-banner-def*)

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*›
by *sepref-to-hoare* (*sep-auto simp*: *isasat-current-information-def print-current-information-def zero-some-stats-def*)

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*›
by *sepref-to-hoare* (*sep-auto simp*: *isasat-current-information-def print-current-information-def zero-some-stats-def*)

sepref-definition *isasat-current-status-code*
is ‹*isasat-current-status*›
:: ‹*isasat-unbounded-assn*$^d \rightarrow_a$ *isasat-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-unbounded-assn-def isasat-current-status-def*
by *sepref*

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*›
supply [[*goals-limit=1*]]
unfolding *isasat-bounded-assn-def isasat-current-status-def*
by *sepref*

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*›

unfolding *cdcl-twl-o-prog-wl-D-heur-def PR-CONST-def*
unfolding *get-conflict-wl-is-None get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
unfolding *cdcl-twl-o-prog-wl-D-heur-def PR-CONST-def*
unfolding *get-conflict-wl-is-None get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
by *sepref*

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} $\rightarrow_a$ *isasat-unbounded-assn*⟩
unfolding *cdcl-twl-stgy-prog-wl-D-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*, *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-init-atms N NE*) ∧
    valid-arena N′ N (*set vdom*) ∧
    (*D′*, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N NE*) ∧
    (*D* = *None* ⟶ *j* ≤ *length M*) ∧
    Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
    (*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms N NE*)) ∧
    vm ∈ *isa-vmtf* (*all-init-atms N NE*) *M* ∧
    phase-saving (*all-init-atms N NE*) *φ* ∧
    no-dup M ∧
    clvls ∈ *counts-maximum-level M D* ∧
    cach-refinement-empty (*all-init-atms N NE*) *cach* ∧
    out-learned M D outl ∧
    lcount = *size* (*learned-clss-lf N*) ∧
    vdom-m (*all-init-atms N NE*)   *W N* ⊆ *set vdom* ∧
    mset avdom ⊆# *mset vdom* ∧
    isasat-input-bounded (*all-init-atms N NE*) ∧
    isasat-input-nempty (*all-init-atms N NE*) ∧
    distinct vdom ∧ *old-arena* = []
  }›

abbreviation *twl-st-heur′′′′* **where**
  ‹*twl-st-heur′′′′ r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*}›

abbreviation *twl-st-heur-restart′′′* **where**
  ‹*twl-st-heur-restart′′′ r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) = *r*}›

abbreviation *twl-st-heur-restart′′′′* **where**
  ‹*twl-st-heur-restart′′′′ r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*}›

definition *twl-st-heur-restart-ana* :: ‹*nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-restart-ana r* = {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) = *r*}›

lemma *twl-st-heur-restart-anaD*: ‹*x* ∈ *twl-st-heur-restart-ana r* ⟹ *x* ∈ *twl-st-heur-restart*›
  by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma *twl-st-heur-restartD*: ‹*x* ∈ *twl-st-heur-restart* ⟹ *x* ∈ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur* (*fst x*)))›
  by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

definition *clause-score-ordering* **where**
⟨*clause-score-ordering* = (λ(*lbd*, *act*) (*lbd'*, *act'*). *lbd* < *lbd'* ∨ (*lbd* = *lbd'* ∧ *act* ≤ *act'*))⟩

lemma *unbounded-id*: ⟨*unbounded* (*id* :: *nat* ⇒ *nat*)⟩
by (*auto simp*: *bounded-def*) *presburger*

global-interpretation *twl-restart-ops id*
by *unfold-locales*

global-interpretation *twl-restart id*
by *standard* (*rule unbounded-id*)

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*)⟩
by *auto*

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*)⟩
by *auto*

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*)⟩
by *auto*

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*)⟩
by *auto*

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
}›
unfolding *find-local-restart-target-level-def* **by** (*auto simp*: *RES-RETURN-RES2 uncurry-def*)

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*›
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-alt-def*
uncurry-def Let-def
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b ae af ag ah ba bb ai aj ak al am bc bd*
apply (*refine-rcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*brk*, *i*). (*If brk 0 1*) + *length b* − *i*)›]
assert.*ASSERT-leI*)
subgoal by *auto*
subgoal
unfolding *find-local-restart-target-level-int-inv-def*
by (*auto simp*: *trail-pol-alt-def control-stack-length-count-dec*)
subgoal by *auto*
subgoal by (*auto simp*: *trail-pol-alt-def intro*: *control-stack-le-length-M*)
subgoal for *s x1 x2*
by (*subgoal-tac* ‹*a* ! (*b* ! *x2*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›)
(*auto simp*: *trail-pol-alt-def rev-map lits-of-def rev-nth*
vmtf-def atms-of-def isa-vmtf-def
intro!: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*)
subgoal by (*auto simp*: *find-local-restart-target-level-int-inv-def*)
subgoal by (*auto simp*: *trail-pol-alt-def control-stack-length-count-dec*
find-local-restart-target-level-int-inv-def)
subgoal by *auto*
done
done

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})›
apply (*intro ext*)
apply (*case-tac x*)
by (*auto simp*: *find-local-restart-target-level-st-def*)

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 ‹$(S,\ T) \in$ *twl-st-heur-restart*›
shows ‹(*get-trail-wl-heur* S, *get-trail-wl* T) $\in$ *trail-pol* (*all-init-atms-st* T)›
using *assms* **by** (*cases* S; *cases* T; *auto simp*: *twl-st-heur-restart-def*)

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›
unfolding *literals-are-in-$\mathcal{L}_{in}$-trail-def trail-pol-def*
by *auto*

lemma *refine-generalise1*: $A \le B \Longrightarrow do\ \{x \leftarrow B;\ C\ x\} \le D \Longrightarrow do\ \{x \leftarrow A;\ C\ x\} \le (D{::}\ 'a\ nres)$
using *Refine-Basic.bind-mono*(*1*) *dual-order.trans* **by** *blast*

lemma *refine-generalise2*: $A \le B \Longrightarrow do\ \{x \leftarrow do\ \{x \leftarrow B;\ A'\ x\};\ C\ x\} \le D \Longrightarrow$
$do\ \{x \leftarrow do\ \{x \leftarrow A;\ A'\ x\};\ C\ x\} \le (D{::}\ 'a\ nres)$
by (*simp add*: *refine-generalise1*)

lemma *cdcl-twl-local-restart-wl-D-spec-int*:
‹*cdcl-twl-local-restart-wl-D-spec* $(M,\ N,\ D,\ NE,\ UE,\ Q,\ W) \ge ($ *do* {
ASSERT(*restart-abs-wl-D-pre* $(M,\ N,\ D,\ NE,\ UE,\ Q,\ W)$ *False*);
$i \leftarrow$ *SPEC*(λ-. *True*);
if i
then RETURN $(M,\ N,\ D,\ NE,\ UE,\ Q,\ W)$
else do {
$(M,\ Q') \leftarrow$ *SPEC*($\lambda(M',\ Q')$. ($\exists K\ M2$. (*Decided* K # M', $M2$) $\in$ *set* (*get-all-ann-decomposition* M) $\wedge$
$Q' = \{\#\}) \vee (M' = M \wedge Q' = Q)$);
RETURN $(M,\ N,\ D,\ NE,\ UE,\ Q',\ W)$
}
})›
proof −
have *If-Res*: ‹(*if* i *then* (*RETURN* f) *else* (*RES* g)) = (*RES* (*if* i *then* $\{f\}$ *else* g))› **for** $i\ f\ g$
by *auto*
show *?thesis*
unfolding *cdcl-twl-local-restart-wl-D-spec-def prod.case RES-RETURN-RES2 If-Res*
by *refine-vcg*
(*auto simp*: *If-Res RES-RETURN-RES2 RES-RES-RETURN-RES uncurry-def*
image-iff split:*if-splits*)
qed

lemma *trail-pol-no-dup*: ‹$(M,\ M') \in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *no-dup* M'›
by (*auto simp*: *trail-pol-def*)

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*) $\in$
twl-st-heur''' $r \rightarrow_f \langle$*twl-st-heur'''* $r\rangle$*nres-rel*›
proof −
have K: ‹((*case* S *of*
$(M,\ N,\ D,\ Q,\ W,\ vm,\ \varphi,\ clvls,\ cach,\ lbd,\ outl,\ stats,\ fema,\ sema,$
$ccount,\ vdom,\ lcount) \Rightarrow$
ASSERT (*isa-length-trail-pre* M) $\gg=$
(λ-. *RES* {$(M,\ N,\ D,$ *isa-length-trail* $M,\ W,\ vm,\ \varphi,\ clvls,\ cach,$
$lbd,\ outl,\ stats,\ fema,\ sema,$
restart-info-restart-done $ccount,\ vdom,\ lcount)$}))) =
((*ASSERT* (*case* S *of* $(M,\ N,\ D,\ Q,\ W,\ vm,\ \varphi,\ clvls,\ cach,\ lbd,\ outl,\ stats,\ fema,\ sema,$

ccount, *vdom*, *lcount*) ⇒ *isa-length-trail-pre M*)) ≫=
(λ -. (*case S of*
(*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*,
ccount, *vdom*, *lcount*) ⇒ *RES* {(*M*, *N*, *D*, *isa-length-trail M*, *W*, *vm*, φ, *clvls*, *cach*,
lbd, *outl*, *stats*, *fema*, *sema*,
restart-info-restart-done ccount, *vdom*, *lcount*)}))› **for** *S*
by (*cases S*) *auto*

have *K2*: ‹(*case S of*
(*a*, *b*) ⇒ *RES* (Φ *a b*)) =
(*RES* (*case S of* (*a*, *b*) ⇒ Φ *a b*))› **for** *S*
by (*cases S*) *auto*

have [*dest*]: ‹*av* = *None*› ‹*out-learned a av am* ⟹ *out-learned x1 av am*›
if ‹*restart-abs-wl-D-pre* (*a*, *au*, *av*, *aw*, *ax*, *ay*, *bd*) *False*›
for *a au av aw ax ay bd x1 am*
using *that*
unfolding *restart-abs-wl-D-pre-def restart-abs-wl-pre-def restart-abs-l-pre-def*
restart-prog-pre-def
by (*auto simp*: *twl-st-l-def state-wl-l-def out-learned-def*)
have [*refine0*]:
‹*find-local-restart-target-level-int* (*get-trail-wl-heur S*) (*get-vmtf-heur S*) ≤
⇓ {(*i*, *b*). *b* = (*i* = *count-decided* (*get-trail-wl T*)) ∧
i ≤ *count-decided* (*get-trail-wl T*)} (*SPEC* (λ-. *True*))›
if ‹(*S*, *T*) ∈ *twl-st-heur*› **for** *S T*
apply (*rule find-local-restart-target-level-int-find-local-restart-target-level*[*THEN fref-to-Down-curry*,
THEN order-trans, *of* ‹*all-atms-st T*› ‹*get-trail-wl T*› ‹*get-vmtf-heur S*›])
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal by (*auto simp*: *find-local-restart-target-level-def conc-fun-RES*)
done
have *P*: ‹*P*›
if
ST: ‹(((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*),
bt, *bu*, *bv*, *bw*, *bx*, *by*, *bz*)
∈ *twl-st-heur*› **and**
‹*restart-abs-wl-D-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*) *False*› **and**
‹*restart-abs-wl-heur-pre*
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*), *bo*, *bp*, *bq*, *br*, *bs*)
False› **and**
lvl: ‹(*lvl*, *i*)
∈ {(*i*, *b*).
b = (*i* = *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*))) ∧
i ≤ *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*))}› **and**
‹*i* ∈ {-. *True*}› **and**
‹*lvl* ≠
count-decided-st-heur
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, *ap*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), (*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),

$(bm, bn), bo, bp, bq, br, bs)$› **and**
i: ‹¬ *i*› **and**
H: ‹$(\bigwedge vm0. ((an, bc), vm0) \in$ *distinct-atoms-rel* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)) \Longrightarrow$
$((aj, ak, al, am, bb), vm0) \in$ *vmtf* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz))$ *bt* $\Longrightarrow$
isa-find-decomp-wl-imp (a, aa, ab, ac, ad, b) *lvl*
$((aj, ak, al, am, bb), an, bc)$
$\leq \Downarrow \{(a, b). (a,b) \in$ *trail-pol* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)) \times_f$
$(Id \times_f$ *distinct-atoms-rel* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)))\}$
(*find-decomp-w-ns* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz))$ *bt lvl vm0*) $\Longrightarrow P$)›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx by bz lvl i x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i P*
proof −
have
tr: ‹$((a, aa, ab, ac, ad, b), bt) \in$ *trail-pol* (*all-atms bu* $(bw + bx)$)› **and**
‹*valid-arena ae bu* (*set bo*)› **and**
‹$((af, ag, ba), bv)$
$\in$ *option-lookup-clause-rel* (*all-atms bu* $(bw + bx)$)› **and**
‹$by = \{\#-$ *lit-of x*. $x \in\#$ *mset* (*drop ah* (*rev bt*))$\#\}$› **and**
‹$(ai, bz) \in \langle Id\rangle$*map-fun-rel* ($D_0$ (*all-atms bu* $(bw + bx)$))› **and**
vm: ‹$((aj, ak, al, am, bb), an, bc) \in$ *isa-vmtf* (*all-atms bu* $(bw + bx)$) *bt*› **and**
‹*phase-saving* (*all-atms bu* $(bw + bx)$) *ao*› **and**
‹*no-dup bt*› **and**
‹$ap \in$ *counts-maximum-level bt bv*› **and**
‹*cach-refinement-empty* (*all-atms bu* $(bw + bx)$) (aq, bd)› **and**
‹*out-learned bt bv as*› **and**
‹$bq =$ *size* (*learned-clss-l bu*)› **and**
‹*vdom-m* (*all-atms bu* $(bw + bx)$) *bz bu* $\subseteq$ *set bo*› **and**
‹*set bp* $\subseteq$ *set bo*› **and**
‹$\forall L \in\# \mathcal{L}_{all}$ (*all-atms bu* $(bw + bx)$). *nat-of-lit* $L \leq$ *uint-max*› **and**
‹*all-atms bu* $(bw + bx) \neq \{\#\}$› **and**
bounded: ‹*isasat-input-bounded* (*all-atms bu* $(bw + bx)$)›
using *ST* **unfolding** *twl-st-heur-def all-atms-def*[*symmetric*]
by (*auto*)

obtain *vm0* **where**
vm: ‹$((aj, ak, al, am, bb), vm0) \in$ *vmtf* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)$) *bt*› **and**
vm0: ‹$((an, bc), vm0) \in$ *distinct-atoms-rel* (*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)$)›
using *vm*
by (*auto simp*: *isa-vmtf-def*)
have *n-d*: ‹*no-dup bt*›
using *tr* **by** (*auto simp*: *trail-pol-def*)
show *?thesis*
apply (*rule H*)
apply (*rule vm0*)
apply (*rule vm*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*, *THEN order-trans*, *of bt lvl* ‹$((aj, ak, al, am, bb), vm0)$› - - - ‹*all-atms-st* $(bt, bu, bv, bw, bx, by, bz)$›])
subgoal using *lvl i* **by** *auto*
subgoal using *vm0 tr* **by** *auto*
apply (*subst* (*3*) *Down-id-eq*[*symmetric*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule find-decomp-wl-imp-find-decomp-wl′*[*THEN fref-to-Down-curry2*, *of - bt lvl* ‹$((aj, ak, al, am, bb), vm0)$›])

subgoal
using *that*(*1*−*8*) *vm vm0 bounded n-d tr*
by (*auto simp*: *find-decomp-w-ns-pre-def dest*: *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*)
subgoal by *auto*
using *ST*
by (*auto simp*: *find-decomp-w-ns-def conc-fun-RES twl-st-heur-def*)
qed

show *?thesis*
supply [[*goals-limit*=*1*]]
unfolding *cdcl-twl-local-restart-wl-D-heur-def*
unfolding
find-decomp-wl-st-int-def find-local-restart-target-level-def incr-lrestart-stat-def
empty-Q-def find-local-restart-target-level-st-def nres-monad-laws
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule ref-two-step*)
prefer *2*
apply (*rule cdcl-twl-local-restart-wl-D-spec-int*)
unfolding *bind-to-let-conv Let-def RES-RETURN-RES2 nres-monad-laws*
apply (*refine-vcg*)
subgoal unfolding *restart-abs-wl-heur-pre-def* **by** *blast*
apply *assumption*
subgoal by (*auto simp*: *twl-st-heur-def count-decided-st-heur-def trail-pol-def*)
subgoal by *auto*
apply (*rule P*)
apply *assumption+*
apply (*rule refine-generalise1*)
apply *assumption*
subgoal for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
bw bx by bz lvl i vm0
unfolding *RETURN-def RES-RES2-RETURN-RES RES-RES13-RETURN-RES find-decomp-w-ns-def conc-fun-RES*
RES-RES13-RETURN-RES K K2
apply (*auto simp*: *intro-spec-iff intro*!: *ASSERT-leI isa-length-trail-pre*)
apply (*auto simp*: *isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *isa-vmtfI trail-pol-no-dup*)
apply (*clarsimp simp*: *twl-st-heur-def*)
apply (*rule-tac x*=*aja* **in** *exI*)
apply (*auto simp*: *isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *isa-vmtfI trail-pol-no-dup*)
done
done
qed

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*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*,
  fast-ema, *slow-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* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, (*props*, *decs*, *confl*, *restarts*, -), *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*).
lcount < *3000* + *1000* ∗ *nat-of-uint64 restarts*)›

definition (**in** −) *lower-restart-bound-not-reached* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*lower-restart-bound-not-reached* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*, *opts*, *old*).
(¬*opts-reduce opts* ∨ (*opts-restart opts* ∧ (*lcount* < *2000* + *1000* ∗ *nat-of-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* ⟷
(∃ *i vdom*. *C* = *vdom* ! *i* ∧ *arena-is-valid-clause-vdom arena* (*vdom*!*i*) ∧
(*arena-status arena* (*vdom*!*i*) ≠ *DELETED* ⟶
(*get-clause-LBD-pre arena* (*vdom*!*i*) ∧ *arena-act-pre arena* (*vdom*!*i*)))
∧ *i* < *length vdom*)›

definition (**in** −)*valid-sort-clause-score-pre* **where**
‹*valid-sort-clause-score-pre arena vdom* ⟷
(∀ *C* ∈ *set vdom*. *arena-is-valid-clause-vdom arena C* ∧
(*arena-status arena C* ≠ *DELETED* ⟶
(*get-clause-LBD-pre arena C* ∧ *arena-act-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*(λ*avdom*. *mset avdom* ⊆# *mset avdom0*)›
  unfolding *remove-deleted-clauses-from-avdom-def Let-def*
  apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*, *j*, *avdom*). *length avdom* − *j*)›])
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *s a b aa ba x1 x2 x1a x2a*
    by (*cases* ‹*Suc a* ≤ *aa*›)
     (*auto simp*: *drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last*
       mset-append[*symmetric*] *Cons-nth-drop-Suc simp del*: *mset-append*
     simp flip: *take-Suc-conv-app-nth*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *s a b aa ba x1 x2 x1a x2a*
    by (*cases* ‹*Suc aa* ≤ *length x2a*›)
      (*auto simp*: *drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last*
        Cons-nth-drop-Suc[*symmetric*] *intro*: *subset-mset.dual-order.trans*
     simp flip: *take-Suc-conv-app-nth*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal by *auto*
  done

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*) ← $WHILE_T$ $^{\lambda(i,\ j,\ -).\ i \le j \wedge j \le n}$
    (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*)›
  unfolding *isa-remove-deleted-clauses-from-avdom-def remove-deleted-clauses-from-avdom-def Let-def*
  apply (*refine-vcg WHILEIT-refine*[**where** *R*= ‹*Id* $\times_r$ *Id* $\times_r$ ⟨*Id*⟩*list-rel*›])
  subgoal by (*auto dest*!: *valid-arena-vdom-le*(*2*) *size-mset-mono simp*: *distinct-card*)

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c* **unfolding** *arena-is-valid-clause-vdom-def*
by (*force intro*!: *exI*[*of - N*] *exI*[*of - vdom*] *dest*!: *mset-eq-setD dest*: *mset-le-add-mset simp*: *Cons-nth-drop-Suc*[*symmetric*])
subgoal by *auto*
subgoal by *auto*
subgoal
by (*force simp*: *arena-lifting arena-dom-status-iff*(*1*) *Cons-nth-drop-Suc*[*symmetric*]
dest!: *mset-eq-setD dest*: *mset-le-add-mset*)
subgoal by *auto*
subgoal
by (*force simp*: *arena-lifting arena-dom-status-iff*(*1*) *Cons-nth-drop-Suc*[*symmetric*]
dest!: *mset-eq-setD dest*: *mset-le-add-mset*)
subgoal by *auto*
subgoal by *auto*
done

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′*)›
unfolding *quicksort-clauses-by-score-def*
by (*rule full-quicksort-ref-full-quicksort*[*THEN fref-to-Down*, *THEN order-trans*])
(*auto simp add*: *reorder-list-def clause-score-ordering-def*
intro!: *full-quicksort-correct*[*THEN order-trans*])

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 −
show *?thesis*
unfolding *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
apply (*rule ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left*)
apply (*rule-tac N1* = ‹*get-clauses-wl y*› **and** *vdom1* = ‹*get-vdom x*› **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF - - - remove-deleted-clauses-from-avdom*])

subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def*
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*2*−) *arena-dom-status-iff*(*1*)[*symmetric*] *arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def intro*!: *exI*[*of* - ‹*get-clauses-wl y*›] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal
apply (*rewrite at* ‹- ≤ ¤› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*)
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*)
done
qed

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′*)›
unfolding *valid-sort-clause-score-pre-def all-set-conv-nth valid-sort-clause-score-pre-at-def*
by (*metis in-mset-conv-nth*)+

lemma *sort-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a b* ⟹
mset b = *mset a2′* ⟹ *valid-sort-clause-score-pre a a2′*›
using *mset-eq-setD* **unfolding** *valid-sort-clause-score-pre-def* **by** *blast*

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*)›
by (*auto simp*: *valid-sort-clause-score-pre-def*)

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*)›
by *auto*

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*)›
unfolding *mark-to-delete-clauses-wl-post-def mark-to-delete-clauses-l-post-def*
mark-to-delete-clauses-l-pre-def mark-to-delete-clauses-wl-D-pre-def
apply (*rule iffI*; *normalize-goal*+)
subgoal for *T0 T U0*
apply (*rule exI*[*of* - *T0*])
apply (*rule exI*[*of* - *T*])
apply (*intro conjI*)
apply *auto*[*2*]
apply (*rule exI*[*of* - *U0*])
apply *auto*
using *rtranclp-remove-one-annot-true-clause-cdcl-twl-restart-l2*[*of T0 T U0*]
rtranclp-cdcl-twl-restart-l-list-invs[*of T0*]
apply (*auto dest*:)
using *rtranclp-cdcl-twl-restart-l-list-invs* **by** *blast*
subgoal for *T0 T U0 U*
apply (*rule exI*[*of* - *T0*])
apply (*rule exI*[*of* - *T*])
apply (*intro conjI*)
apply *auto*[*2*]
apply (*rule exI*[*of* - *U0*])
apply *auto*
done
done

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 −
note *cong* = *trail-pol-cong*
option-lookup-clause-rel-cong D_0-*cong isa-vmtf-cong phase-saving-cong*

cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong

show *?A*
supply [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def*
mark-to-delete-clauses-l-pre-def mark-to-delete-clauses-wl-D-pre-def
apply (*rule iffI*)
apply *normalize-goal*+
subgoal for *T U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*rule exI*[*of - T*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - U*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - V*])
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
apply *normalize-goal*+
subgoal for *T U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*rule exI*[*of - T*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - U*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - V*])
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
show ‹*mark-to-delete-clauses-wl-D-pre T* ⟹ *?B*›
supply [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def*
mark-to-delete-clauses-l-pre-def mark-to-delete-clauses-wl-D-pre-def
apply *normalize-goal*+
apply (*rule iffI*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −

apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
show ⟨*mark-to-delete-clauses-wl-post T0 T* $\Longrightarrow$ *?C*⟩
supply [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-post-alt-def*
apply *normalize-goal+*
apply (*rule iffI*)
subgoal for *U0 U V0 V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩]
vdom-m-cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
apply −
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
subgoal for *U0 U V0 V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩]
vdom-m-cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
apply −
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done

qed

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*⟩
by (*cases S*; *auto simp*: *mark-garbage-heur-def*; *fail*)+

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*⟩
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def mset-butlast-remove1-mset*
simp del: *all-init-atms-def*[*symmetric*]
intro: *valid-arena-extra-information-mark-to-delete'*

dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

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*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-garbage-heur-def mark-garbage-wl-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If init-clss-l-fmdrop-irrelev
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro: *valid-arena-extra-information-mark-to-delete′*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

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*›
by (*cases T*; *auto simp*: *mark-unused-st-heur-def*; *fail*)+

lemma *mark-unused-st-heur*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-unused-st-heur-def*
all-init-atms-def[*symmetric*])
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused valid-arena-arena-decr-act*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *mark-unused-st-heur-ana*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›

shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart-ana r*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-unused-st-heur-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused valid-arena-arena-decr-act*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *twl-st-heur-restart-valid-arena*[*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def*)

lemma *twl-st-heur-restart-get-avdom-nth-get-vdom*[*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› ‹*i* < *length* (*get-avdom S*)›
shows ‹*get-avdom S* ! *i* ∈ *set* (*get-vdom S*)›
using *assms* **by** (*fastforce simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *set-mset-mono*)

lemma [*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈ *set* (*get-avdom S*)›
shows ‹*clause-not-marked-to-delete-heur S C* ⟷
(*C* ∈# *dom-m* (*get-clauses-wl T*))› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›**and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
proof −
show ‹*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))›
using *assms*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-dom-status-iff(*1*)
split: *prod.splits*)
assume *C*: ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
show ‹*arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹*arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting

split: *prod.splits*)
show ‹*arena-length* (*get-clauses-wl-heur* *S*) *C* = *length* (*get-clauses-wl* *T* ∝ *C*)›
using *assms* *C*
by (*cases* *S*; *cases* *T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
qed

definition *number-clss-to-keep* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*number-clss-to-keep* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *fast-ema*, *slow-ema*, *ccount*,
vdom, *avdom*, *lcount*).
nat-of-uint64 (*1000* + *150* ∗ *restarts*))›

definition *access-vdom-at* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat*› **where**
‹*access-vdom-at* *S* *i* = *get-avdom* *S* ! *i*›

lemma *access-vdom-at-alt-def*:
‹*access-vdom-at* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
ccount, *vdom*, *avdom*, *lcount*) *i*. *avdom* ! *i*)›
by (*intro ext*) (*auto simp*: *access-vdom-at-def*)

definition *access-vdom-at-pre* **where**
‹*access-vdom-at-pre* *S* *i* ⟷ *i* < *length* (*get-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*›
apply (*cases* ‹*i* < *length xs*›)
apply (*auto dest*!: *in-set-butlastD*)
by (*metis List.last-in-set in-set-upd-cases list.size*(*3*) *not-less-zero*)

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*›
by (*auto simp*: *twl-st-heur-restart-def delete-index-vdom-heur-def*
dest: *in-set-delete-index-and-swapD*)

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*›
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def delete-index-vdom-heur-def*

dest: *in-set-delete-index-and-swapD*)

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*›
by (*cases S*) (*auto simp*: *delete-index-vdom-heur-def*)

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 −
have *1*: ‹ ⇓ (*twl-st-heur-restart-ana r*) (*SPEC* ((=) *T*)) = *do* {
(*i*, *T*) ← *SPEC* (λ(*i*::*nat*, *T'*). (*T'*, *T*) ∈ *twl-st-heur-restart-ana r*);
RETURN T
}›
by (*auto simp*: *RES-RETURN-RES2 uncurry-def conc-fun-RES*)
show *?thesis*
unfolding *mark-clauses-as-unused-wl-D-heur-def 1*
apply (*rule Refine-Basic.bind-mono*)
subgoal
apply (*refine-vcg*
WHILET-rule[**where** *R* = ‹*measure* (λ(*i*, *T*). *length* (*get-avdom T*) − *i*)› **and**
I = ‹λ(-, *S'*). (*S'*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *length* (*get-avdom S'*) ≤ *length*(*get-avdom S*)›])
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *access-vdom-at-pre-def* **by** *auto*
subgoal for *st a S'*
unfolding *clause-not-marked-to-delete-heur-pre-def*
arena-is-valid-clause-vdom-def
by (*fastforce simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *set-mset-mono*
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *exI*[*of* - ‹*set* (*get-vdom S'*)›])

subgoal
by (*auto intro*: *delete-index-vdom-heur-twl-st-heur-restart-ana*)
subgoal by *auto*
subgoal by *auto*
subgoal
unfolding *arena-is-valid-clause-idx-def*
arena-is-valid-clause-vdom-def arena-act-pre-def
by (*fastforce simp*: *twl-st-heur-restart-def twl-st-heur-restart*
dest!: *twl-st-heur-restart-anaD*)
subgoal by (*auto intro*!: *mark-unused-st-heur-ana simp*: *twl-st-heur-restart*
dest: *twl-st-heur-restart-anaD*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
subgoal
by *auto*
done
qed

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* = (λ*S0*. *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*) ← $WHILE_T^{\lambda\text{-}.\ True}$
(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* $T \geq 1$);
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*›
by (*auto simp*: *twl-st-heur-restart-def dest*: *no-dup-no-propa-and-dec*
no-dup-same-annotD)

lemma $\mathcal{L}_{all}$*-mono*:
‹*set-mset* $\mathcal{A}$ ⊆ *set-mset* $\mathcal{B}$ ⟹ L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ L ∈# $\mathcal{L}_{all}$ $\mathcal{B}$›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma $\mathcal{L}_{all}$*-init-all*:
‹L ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*) ⟹ L ∈# $\mathcal{L}_{all}$ (*all-atms-st x1a*)›
apply (*rule* $\mathcal{L}_{all}$*-mono*)
defer
apply *assumption*
apply (*cases x1a*)
apply (*auto simp*: *all-init-atms-def all-lits-def all-init-lits-def*
all-lits-of-mm-union
simp del: *all-init-atms-def*[*symmetric*])
by (*metis all-clss-l-ran-m all-lits-of-mm-union imageI image-mset-union union-iff*)

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 ← *mark-clauses-as-unused-wl-D-heur* *i* *T*;
incr-restart-stat *T*
})›
unfolding *mark-to-delete-clauses-wl-D-heur-def*
by (*auto* *intro*!: *ext* *simp*: *get-clauses-wl-heur.simps*)

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*) ∈
twl-st-heur-restart-ana *r* $\rightarrow_f$ ⟨*twl-st-heur-restart-ana* *r*⟩*nres-rel*›
proof −
have *mark-to-delete-clauses-wl-D-alt-def*:
‹*mark-to-delete-clauses-wl-D* = (λ*S0*. *do* {
ASSERT(*mark-to-delete-clauses-wl-D-pre* *S0*);
S ← *reorder-vdom-wl* *S0*;
xs ← *collect-valid-indices-wl* *S*;
l ← *SPEC*(λ-::*nat*. *True*);
(-, *S*, -) ← $WHILE_T{}^{mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}D\text{-}inv\ S\ xs}$
(λ(*i*, *T*, *xs*). *i* < *length* *xs*)
(λ(*i*, *T*, *xs*). *do* {
if(*xs*!*i* ∉# *dom-m* (*get-clauses-wl* *T*)) *then* *RETURN* (*i*, *T*, *delete-index-and-swap* *xs* *i*)
else *do* {
ASSERT(*0* < *length* (*get-clauses-wl* *T*∝(*xs*!*i*)));
ASSERT (*get-clauses-wl* *T* ∝ (*xs* ! *i*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st* *T*));
can-del ← *SPEC*(λ*b*. *b* ⟶
(*Propagated* (*get-clauses-wl* *T*∝(*xs*!*i*)!*0*) (*xs*!*i*) ∉ *set* (*get-trail-wl* *T*)) ∧
¬*irred* (*get-clauses-wl* *T*) (*xs*!*i*) ∧ *length* (*get-clauses-wl* *T* ∝ (*xs*!*i*)) ≠ *2*);
ASSERT(*i* < *length* *xs*);
if *can-del*
then

RETURN (*i*, *mark-garbage-wl* (*xs*!*i*) *T*, *delete-index-and-swap xs i*)
else
RETURN (*i*+*1*, *T*, *xs*)
}
})
(*l*, *S*, *xs*);
RETURN S
})›
unfolding *mark-to-delete-clauses-wl-D-def reorder-vdom-wl-def*
by (*auto intro*!: *ext*)
have *mono*: ‹$g = g' \Longrightarrow$ *do* {*f*; *g*} = *do* {*f*; *g'*}›
‹($\bigwedge x$. *h x* = *h' x*) $\Longrightarrow$ *do* {*x* ← *f*; *h x*} = *do* {*x* ← *f*; *h' x*}› **for** *f f'* :: ‹- *nres*› **and** *g g'* **and** *h h'*
by *auto*

have [*refine0*]: ‹*RETURN* (*get-avdom x*) ≤ ⇓ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-avdom x*} (*collect-valid-indices-wl y*)›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*›
for *x y*
proof −
show *?thesis* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed
have *init-rel*[*refine0*]: ‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r* $\Longrightarrow$
(*l*, *la*) ∈ *nat-rel* $\Longrightarrow$
((*l*, *x*), *la*, *y*) ∈ *nat-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *get-avdom S* = *get-avdom x*}›
for *x y l la*
by *auto*

have *get-the-propagation-reason*:
‹*get-the-propagation-reason-pol* (*get-trail-wl-heur x2b*)
(*arena-lit* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b* + *0*))
≤ ⇓ {(*D*, *b*). *b* ⟷ ((*D* ≠ *Some* (*get-avdom x2b* ! *x1b*)) ∧
arena-lbd (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) > *MINIMUM-DELETION-LBD* ∧
arena-status (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) = *LEARNED*) ∧
arena-length (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) ≠ *two-uint32-nat* ∧
¬*marked-as-used* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*)}
(*SPEC*
(λ*b*. *b* ⟶
Propagated (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*) (*x2a* ! *x1*)
∉ *set* (*get-trail-wl x1a*) ∧
¬ *irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*) ∧
length (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*)) ≠ *2*))›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧
V = *y* ∧
(*mark-to-delete-clauses-wl-D-pre y* ⟶
mark-to-delete-clauses-wl-D-pre V) ∧
(*mark-to-delete-clauses-wl-D-heur-pre x* ⟶

mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*ys*, *xs*) ∈ {(*xs*, *xs*′). *xs* = *xs*′ ∧ *xs* = *get-avdom S*}› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {-. *True*}› **and**
xa-x′: ‹(*xa*, *x*′)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)› **and**
‹*case x*′ *of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*x1b* < *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)› **and**
‹¬ ¬ *clause-not-marked-to-delete-heur x2b* (*get-avdom x2b* ! *x1b*)› **and**
‹¬ *x2a* ! *x1* ∉# *dom-m* (*get-clauses-wl x1a*)› **and**
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))› **and**
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-avdom x2b* ! *x1b*), *0*)› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x*′ = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
L: ‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
for *x y S Sa xs*′ *xs l la xa x*′ *x1 x2 x1a x2a x1*′ *x2*′ *x3 x1b ys x2b*
proof −
have *L*: ‹*arena-lit* (*get-clauses-wl-heur x2b*) (*x2a* ! *x1*) ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
using *L that* **by** (*auto simp*: *twl-st-heur-restart st arena-lifting dest*: $\mathcal{L}_{all}$*-init-all twl-st-heur-restart-anaD*)

show *?thesis*
apply (*rule order.trans*)
apply (*rule get-the-propagation-reason-pol*[*THEN fref-to-Down-curry*,
of ‹*all-init-atms-st x1a*› ‹*get-trail-wl x1a*›
‹*arena-lit* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b* + *0*)›])
subgoal
using *xa-x*′ *L* **by** (*auto simp add*: *twl-st-heur-restart-def st*)
subgoal
using *xa-x*′ **by** (*auto simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def st*)
using *that* **unfolding** *get-the-propagation-reason-def* **apply** −
by (*auto simp*: *twl-st-heur-restart lits-of-def get-the-propagation-reason-def*
conc-fun-RES
dest!: *twl-st-heur-restart-anaD dest*: *twl-st-heur-restart-same-annotD imageI*[*of - - lit-of*])
qed
have ‹((*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
slow-ema, *ccount*, *vdom*, *avdom*, *lcount*),
S′)
∈ *twl-st-heur-restart* ⟹
((*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
slow-ema, *ccount*, *vdom*, *avdom*′, *lcount*),
S′)
∈ *twl-st-heur-restart*›
if ‹*mset avdom*′ ⊆# *mset avdom*›
for *M*′ *N*′ *D*′ *j W*′ *vm φ clvls cach lbd outl stats fast-ema slow-ema*
ccount vdom lcount S′ *avdom*′ *avdom*
using *that* **unfolding** *twl-st-heur-restart-def*
by *auto*
then have *mark-to-delete-clauses-wl-D-heur-pre-vdom*′:
‹*mark-to-delete-clauses-wl-D-heur-pre* (*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*′, *lcount*) ⟹
mark-to-delete-clauses-wl-D-heur-pre (*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*,

fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*)⟩
if ⟨*mset avdom* ⊆# *mset avdom'*⟩
for *M' N' D' j W' vm* φ *clvls cach lbd outl stats fast-ema slow-ema avdom avdom'*
ccount vdom lcount
using *that*
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def*
by *metis*
have [*refine0*]:
⟨*sort-vdom-heur S* ≤ ⇓ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧ *V* = *T* ∧
(*mark-to-delete-clauses-wl-D-pre T* ⟶ *mark-to-delete-clauses-wl-D-pre V*) ∧
(*mark-to-delete-clauses-wl-D-heur-pre S* ⟶ *mark-to-delete-clauses-wl-D-heur-pre U*)}
(*reorder-vdom-wl T*)⟩
if ⟨(*S*, *T*) ∈ *twl-st-heur-restart-ana r*⟩ **for** *S T*
using *that* **unfolding** *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*refine-rcg ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left*)
apply (*rule-tac N1* = ⟨*get-clauses-wl T*⟩ **and** *vdom1* = ⟨(*get-vdom S*)⟩ **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF - - - remove-deleted-clauses-from-avdom*])
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of -* ⟨*get-clauses-wl T*⟩] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def valid-arena-vdom-subset twl-st-heur-restart-def*
dest!: *size-mset-mono valid-arena-vdom-subset*)
subgoal
apply (*rewrite at* ⟨- ≤ ¤⟩ *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*)
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
intro: *mark-to-delete-clauses-wl-D-heur-pre-vdom'*
dest: *mset-eq-setD*)
done
have *already-deleted*:
⟨((*x1b*, *delete-index-vdom-heur x1b x2b*), *x1*, *x1a*,
delete-index-and-swap x2a x1)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩
if
⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r*⟩ **and**

‹mark-to-delete-clauses-wl-D-pre y› **and**
‹mark-to-delete-clauses-wl-D-heur-pre x› **and**
‹(S, Sa)
∈ {(U, V).
(U, V) ∈ twl-st-heur-restart-ana r ∧
V = y ∧
(mark-to-delete-clauses-wl-D-pre y ⟶
mark-to-delete-clauses-wl-D-pre V) ∧
(mark-to-delete-clauses-wl-D-heur-pre x ⟶
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(ys, xs) ∈ {(xs, xs'). xs = xs' ∧ xs = get-avdom S}› **and**
‹(l, la) ∈ nat-rel› **and**
‹la ∈ {-. True}› **and**
xx: ‹(xa, x')
∈ nat-rel $\times_f$ {(Sa, T, xs). (Sa, T) ∈ twl-st-heur-restart-ana r ∧ xs = get-avdom Sa}› **and**
‹case xa of (i, S) ⇒ i < length (get-avdom S)› **and**
‹case x' of (i, T, xs) ⇒ i < length xs› **and**
‹mark-to-delete-clauses-wl-D-inv Sa xs x'› **and**
st:
‹x2 = (x1a, x2a)›
‹x' = (x1, x2)›
‹xa = (x1b, x2b)› **and**
le: ‹x1b < length (get-avdom x2b)› **and**
‹access-vdom-at-pre x2b x1b› **and**
‹clause-not-marked-to-delete-heur-pre (x2b, get-avdom x2b ! x1b)› **and**
‹¬ clause-not-marked-to-delete-heur x2b (get-avdom x2b ! x1b)› **and**
‹x2a ! x1 ∉# dom-m (get-clauses-wl x1a)›
for x y S xs l la xa x' xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa
proof −
show ?thesis
using xx le **unfolding** st
by (auto simp: twl-st-heur-restart-ana-def delete-index-vdom-heur-def
twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
learned-clss-l-l-fmdrop size-remove1-mset-If
intro: valid-arena-extra-information-mark-to-delete'
dest!: in-set-butlastD in-vdom-m-fmdropD
elim!: in-set-upd-cases)
qed
have get-learned-count-ge: ‹1 ≤ get-learned-count x2b›
if
xy: ‹(x, y) ∈ twl-st-heur-restart-ana r› **and**
‹(xa, x')
∈ nat-rel $\times_f$
{(Sa, T, xs).
(Sa, T) ∈ twl-st-heur-restart-ana r ∧ xs = get-avdom Sa}› **and**
‹mark-to-delete-clauses-wl-D-inv Sa xs x'› **and**
‹x2 = (x1a, x2a)› **and**
‹x' = (x1, x2)› **and**
‹xa = (x1b, x2b)› **and**
dom: ‹¬ x2a ! x1 ∉# dom-m (get-clauses-wl x1a)› **and**
‹can-del
∈ {b. b ⟶
Propagated (get-clauses-wl x1a ∝ (x2a ! x1) ! 0) (x2a ! x1)
∉ set (get-trail-wl x1a) ∧
¬ irred (get-clauses-wl x1a) (x2a ! x1) ∧
length (get-clauses-wl x1a ∝ (x2a ! x1)) ≠ 2}› **and**

⟨*can-del*⟩ **for** *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b D can-del*
proof −
have ⟨¬*irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)⟩ **and** ⟨(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*⟩
using *that* **by** (*auto simp*:)
then show *?thesis*
using *dom* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*)
qed
have *init*:
⟨(*u*, *xs*) ∈ {(*xs*, *xs′*). *xs* = *xs′* ∧ *xs* = *get-avdom S*} ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
(*S*, *Sa*) ∈ *twl-st-heur-restart-ana r* ⟹
mark-to-delete-clauses-wl-D-inv Sa xs (*la*, *Sa*, *xs*) ⟹
((*l*, *S*), *la*, *Sa*, *xs*) ∈ *nat-rel* $\times_f$
{(*Sa*, (*T*, *xs*)). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩
for *x y S Sa xs l la u*
by *auto*
have [*refine0*]: ⟨*mark-clauses-as-unused-wl-D-heur i T*
≤ *SPEC*
(λ*x*. *incr-restart-stat x* ≤ *SPEC* (λ*c*. (*c*, *S*) ∈ *twl-st-heur-restart-ana r*))⟩
if ⟨(*T*, *S*) ∈ *twl-st-heur-restart-ana r*⟩ **for** *S T i*
by (*rule order-trans*, *rule mark-clauses-as-unused-wl-D-heur*[*OF that*, *of i*])
(*auto simp*: *conc-fun-RES incr-restart-stat-def*
twl-st-heur-restart-ana-def twl-st-heur-restart-def)
show *?thesis*
supply *sort-vdom-heur-def*[*simp*] *twl-st-heur-restart-anaD*[*dest*]
unfolding *mark-to-delete-clauses-wl-D-heur-alt-def mark-to-delete-clauses-wl-D-alt-def*
access-lit-in-clauses-heur-def Let-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg sort-vdom-heur-reorder-vdom-wl*[*THEN fref-to-Down*])
subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule init*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-vdom-at-pre-def*)
subgoal for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d*
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def*
prod.simps
by (*rule exI*[*of* - ⟨*get-clauses-wl x2a*⟩], *rule exI*[*of* - ⟨*set* (*get-vdom x2d*)⟩])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal
by (*auto simp*: *twl-st-heur-restart*)
subgoal
by (*rule already-deleted*)
subgoal for *x y* - - - - - *xs l la xa x′ x1 x2 x1a x2a*
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def
by (*rule bex-leI*[*of* - ⟨*get-avdom x2a* ! *x1a*⟩], *simp*, *rule exI*[*of* - ⟨*get-clauses-wl x1*⟩])
(*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset*

size-mset-mono)
  subgoal premises *p* **using** *p*(*7*−) **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
  apply (*rule get-the-propagation-reason*; *assumption*)
  subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
    unfolding *prod.simps*
      get-clause-LBD-pre-def arena-is-valid-clause-idx-def
    by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
      (*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
  subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
    unfolding *prod.simps*
      arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
    by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
      (*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*
  twl-st-heur-restart-get-avdom-nth-get-vdom)
  subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
    unfolding *prod.simps*
      arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
    by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
      (*auto simp*: *twl-st-heur-restart arena-dom-status-iff*
        dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
  subgoal unfolding *marked-as-used-pre-def* **by** *fast*
  subgoal
    by (*auto simp*: *twl-st-heur-restart*)
  subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
    unfolding *prod.simps mark-garbage-pre-def*
      arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
    by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
      (*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
  subgoal for *x y S Sa uu- xs l la xa x′ x1 x2 x1a x2a x1b x2b D can-del*
      by (*rule get-learned-count-ge*)
  subgoal
    by (*auto intro*!: *mark-garbage-heur-wl-ana*)
  subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
    unfolding *prod.simps mark-garbage-pre-def arena-act-pre-def*
      arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
    by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
      (*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
  subgoal
    by (*auto intro*!: *mark-unused-st-heur-ana*)
 subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
  subgoal
    by (*auto simp*:)
  done
qed

definition *cdcl-twl-full-restart-wl-prog-heur* **where**
‹*cdcl-twl-full-restart-wl-prog-heur S* = *do* {
 - ← *ASSERT* (*mark-to-delete-clauses-wl-D-heur-pre S*);
 T ← *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*) ∈

twl-st-heur''' r $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*›
unfolding *cdcl-twl-full-restart-wl-prog-heur-def cdcl-twl-full-restart-wl-prog-D-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D[*THEN fref-to-Down*])
subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-alt-def*
by *fast*
apply (*rule twl-st-heur-restartD*)
subgoal
by (*subst mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur*[*symmetric*]) *auto*
subgoal
by (*auto simp*: *mark-to-delete-clauses-wl-post-twl-st-heur twl-st-heur-restart-anaD*)
(*auto simp*: *twl-st-heur-restart-ana-def*)
done

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*) ∈
twl-st-heur''' r $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*›
unfolding *cdcl-twl-restart-wl-D-prog-def cdcl-twl-restart-wl-heur-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec[*THEN fref-to-Down*]
cdcl-twl-full-restart-wl-prog-heur-cdcl-twl-full-restart-wl-prog-D[*THEN fref-to-Down*])
subgoal by *auto*
subgoal by *auto*
done

definition *isasat-replace-annot-in-trail*
:: ‹*nat literal* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *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* ∈# *all-init-atms-st* (*ys* @ *Propagated x2 C* # *zs*, *x2n'*)› **and**
‹(((*x1b*, *x1c*, *x1d*, *x1e*, *x2d*), *x2n*),
(*ys* @ *Propagated x2 C* # *zs*, *x2n'*))
∈ *twl-st-heur-restart-ana r*›
shows
‹(((*x1b*, *x1c*, *x1d*, *x1e*[*atm-of x2* := *0*], *x2d*), *x2n*),
(*ys* @ *Propagated x2 0* # *zs*, *x2n'*))
∈ *twl-st-heur-restart-ana r*›

proof −
  let *?S* = ‹(*ys* @ *Propagated x2 C* # *zs*, *x2n'*)›
  let *?𝒜* = ‹*all-init-atms-st ?S*›
  have *pol*: ‹((*x1b*, *x1c*, *x1d*, *x1e*, *x2d*), *ys* @ *Propagated x2 C* # *zs*)
      ∈ *trail-pol* (*all-init-atms-st ?S*)›
    using *assms*(*3*) **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
    by *auto*
  obtain *x y* **where**
    x2d: ‹*x2d* = (*count-decided* (*ys* @ *Propagated x2 C* # *zs*), *y*)› **and**
    reasons: ‹((*map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*)), *x1e*),
      ys @ *Propagated x2 C* # *zs*)
     ∈ *ann-lits-split-reasons ?𝒜*› **and**
    pol: ‹∀ L∈#$\mathcal{L}_{all}$ *?𝒜*.   *nat-of-lit L* < *length x1c* ∧
       x1c ! *nat-of-lit L* = *polarity* (*ys* @ *Propagated x2 C* # *zs*) *L*› **and**
    n-d: ‹*no-dup* (*ys* @ *Propagated x2 C* # *zs*)› **and**
    lvls: ‹∀ L∈#$\mathcal{L}_{all}$ *?𝒜*. *atm-of L* < *length x1d* ∧
       x1d ! *atm-of L* = *get-level* (*ys* @ *Propagated x2 C* # *zs*) *L*› **and**
    ‹*undefined-lit ys* (*lit-of* (*Propagated x2 C*))› **and**
    ‹*undefined-lit zs* (*lit-of* (*Propagated x2 C*))› **and**
    inA:‹∀ L∈*set* (*ys* @ *Propagated x2 C* # *zs*). *lit-of L* ∈# $\mathcal{L}_{all}$ *?𝒜*› **and**
    cs: ‹*control-stack y* (*ys* @ *Propagated x2 C* # *zs*)› **and**
    ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail ?𝒜* (*ys* @ *Propagated x2 C* # *zs*)› **and**
    ‹*length* (*ys* @ *Propagated x2 C* # *zs*) < *uint-max*› **and**
    ‹*length* (*ys* @ *Propagated x2 C* # *zs*) ≤ *uint-max div 2* + *1*› **and**
    ‹*count-decided* (*ys* @ *Propagated x2 C* # *zs*) < *uint-max*› **and**
    ‹*length* (*map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*))) =
     length (*ys* @ *Propagated x2 C* # *zs*)› **and**
    bounded: ‹*isasat-input-bounded ?𝒜*› **and**
    x1b: ‹*x1b* = *map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*))›
    using *pol* **unfolding** *trail-pol-alt-def*
    by *blast*
  have *lev-eq*: ‹*get-level* (*ys* @ *Propagated x2 C* # *zs*) =
    get-level (*ys* @ *Propagated x2 0* # *zs*)›
    ‹*count-decided* (*ys* @ *Propagated x2 C* # *zs*) =
      count-decided (*ys* @ *Propagated x2 0* # *zs*)›
    by (*auto simp*: *get-level-cons-if get-level-append-if*)
  have [*simp*]: ‹*atm-of x2* < *length x1e*›
    using *reasons x2 in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
    by (*auto simp*: *ann-lits-split-reasons-def*
      dest: *multi-member-split*)
  have ‹((*x1b*, *x1e*[*atm-of x2* := *0*]), *ys* @ *Propagated x2 0* # *zs*)
     ∈ *ann-lits-split-reasons ?𝒜*›
    using *reasons n-d undefined-notin*
    by (*auto simp*: *ann-lits-split-reasons-def x1b*
      DECISION-REASON-def atm-of-eq-atm-of)
  moreover have *n-d'*: ‹*no-dup* (*ys* @ *Propagated x2 0* # *zs*)›
    using *n-d* **by** *auto*
  moreover have ‹∀ L∈#$\mathcal{L}_{all}$ *?𝒜*.
     nat-of-lit L < *length x1c* ∧
       x1c ! *nat-of-lit L* = *polarity* (*ys* @ *Propagated x2 0* # *zs*) *L*›
    using *pol* **by** (*auto simp*: *polarity-def*)
  moreover have ‹∀ L∈#$\mathcal{L}_{all}$ *?𝒜*.
    atm-of L < *length x1d* ∧
        x1d ! *atm-of L* = *get-level* (*ys* @ *Propagated x2 0* # *zs*) *L*›
    using *lvls* **by** (*auto simp*: *get-level-append-if get-level-cons-if*)
  moreover have ‹*control-stack y* (*ys* @ *Propagated x2 0* # *zs*)›

using *cs* **apply** −
apply (*subst control-stack-alt-def*[*symmetric*, *OF n-d'*])
apply (*subst* (*asm*) *control-stack-alt-def*[*symmetric*, *OF n-d*])
unfolding *control-stack'-def lev-eq*
apply *normalize-goal*
apply (*intro ballI conjI*)
apply (*solves auto*)
unfolding *set-append list.set*(*2*) *Un-iff insert-iff*
apply (*rule disjE*, *assumption*)
subgoal for *L*
by (*drule-tac x* = *L* **in** *bspec*)
(*auto simp*: *nth-append nth-Cons split*: *nat.splits*)
apply (*rule disjE*[*of* ‹- = -›], *assumption*)
subgoal for *L*
by (*auto simp*: *nth-append nth-Cons split*: *nat.splits*)
subgoal for *L*
by (*drule-tac x* = *L* **in** *bspec*)
(*auto simp*: *nth-append nth-Cons split*: *nat.splits*)

done
ultimately have
‹((*x1b*, *x1c*, *x1d*, *x1e*[*atm-of x2* := *0*], *x2d*), *ys* @ *Propagated x2 0* # *zs*)
∈ *trail-pol ?*$\mathcal{A}$›
using *n-d x2 inA bounded*
unfolding *trail-pol-def x2d*
by *simp*
moreover { **fix** *aaa ca*
have ‹*vmtf-*$\mathcal{L}_{all}$ (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 C* # *zs*) =
vmtf-$\mathcal{L}_{all}$ (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *vmtf-*$\mathcal{L}_{all}$*-def*)
then have ‹*isa-vmtf* (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 C* # *zs*) =
isa-vmtf (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *isa-vmtf-def vmtf-def*
image-iff)
}
moreover { **fix** *D*
have ‹*get-level* (*ys* @ *Propagated x2 C* # *zs*) = *get-level* (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *get-level-append-if get-level-cons-if*)
then have ‹*counts-maximum-level* (*ys* @ *Propagated x2 C* # *zs*) *D* =
counts-maximum-level (*ys* @ *Propagated x2 0* # *zs*) *D*› **and**
‹*out-learned* (*ys* @ *Propagated x2 C* # *zs*) = *out-learned* (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *counts-maximum-level-def card-max-lvl-def*
out-learned-def intro!: *ext*)
}
ultimately show *?thesis*
using *assms*(*3*) **unfolding** *twl-st-heur-restart-ana-def*
by (*cases x2n*; *cases x2n'*)
(*auto simp*: *twl-st-heur-restart-def*
simp flip: *mset-map drop-map*)
qed

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) ∈
[λ((*L*, *C*), *S*).
Propagated L C ∈ *set* (*get-trail-wl S*) ∧ *atm-of L* ∈# *all-init-atms-st S*]$_f$
Id $\times_f$ *Id*$\times_f$ *twl-st-heur-restart-ana r* → ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*⟩
unfolding *isasat-replace-annot-in-trail-def replace-annot-l-def*
uncurry-def
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal
using *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def*
twl-st-heur-restart-def twl-st-heur-restart-ana-def)
subgoal for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f*
x2f x1g x2g x1h x1i
x2h x2i x1j x1k x2j x1l x2k x1m x2l x1n x2m x2n
apply (*clarify dest*!: *split-list*[*of* ⟨*Propagated - -*⟩])
apply (*rule RETURN-SPEC-refine*)
apply (*rule-tac x* = ⟨(*ys* @ *Propagated x1a 0* # *zs*, *x1c*, *x1d*,
x1e, *x1f*, *x1g*, *x2g*)⟩ **in** *exI*)
apply (*intro conjI*)
prefer *2*
apply (*rule-tac x* = ⟨*ys* @ *Propagated x1a 0* # *zs*⟩ **in** *exI*)
apply (*intro conjI*)
apply *blast*
by (*auto intro*!: *trail-pol-replace-annot-in-trail-spec*
trail-pol-replace-annot-in-trail-spec2
simp: *atm-of-eq-atm-of all-init-atms-def*
simp del: *all-init-atms-def*[*symmetric*])
done

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(¬*st* ⟶ *lcount* ≥ *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* = (λ*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* ≠ *None* ∧ *i* + *1* ≤ *uint32-max*);
if the C = *0 then RETURN* (*i*+*1*, *S*)
else do {
ASSERT(*C* ≠ *None*);
S ← *isasat-replace-annot-in-trail L* (*the C*) *S*;
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *the C*) ∧ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) (*the C*));
S ← *mark-garbage-heur2* (*the C*) *S*;

— $S \leftarrow$ *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* ⟷
(∃ *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*).
(∃ *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* = (λ*S*. *do* {
ASSERT((*isa-length-trail-pre o get-trail-wl-heur*) *S*);
k ← (*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*) ← $WHILE_T{}^{remove\text{-}one\text{-}annot\text{-}true\text{-}clause\text{-}imp\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S}$
(λ(*i*, *S*). *i* < *k*)
(λ(*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*) ∈ *twl-st-heur-restart*›
shows ‹*get-pos-of-level-in-trail* (*get-trail-wl T*) *0*
≤ *SPEC*
(λ*k*. ∃ *M1*. (∃ *M2 K*.
(*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) ∧
count-decided M1 = *0* ∧ *k* = *length M1*)›
unfolding *get-pos-of-level-in-trail-def*
proof (*rule SPEC-rule*)
fix *x*
assume *H*: ‹*x* < *length* (*get-trail-wl T*) ∧
is-decided (*rev* (*get-trail-wl T*) ! *x*) ∧
get-level (*get-trail-wl T*) (*lit-of* (*rev* (*get-trail-wl T*) ! *x*)) = *0* + *1*›
let *?M1* = ‹*rev* (*take x* (*rev* (*get-trail-wl T*)))›
let *?K* = ‹*Decided* ((*lit-of*(*rev* (*get-trail-wl T*) ! *x*)))›
let *?M2* = ‹*rev* (*drop* (*Suc x*) (*rev* (*get-trail-wl T*)))›
have *T*: ‹(*get-trail-wl T*) = *?M2* @ *?K* # *?M1*› **and**
K: ‹*Decided* (*lit-of ?K*) = *?K*›
apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* (*get-trail-wl T*) − *Suc x*›])
apply (*subst Cons-nth-drop-Suc*[*symmetric*])
using *H*
apply (*auto simp*: *rev-take rev-drop rev-nth*)
apply (*cases* ‹*rev* (*get-trail-wl T*) ! *x*›)

apply (*auto simp*: *rev-take rev-drop rev-nth*)
done
have *n-d*: ‹*no-dup* (*get-trail-wl T*)›
using *assms*(*1*)
by (*auto simp*: *twl-st-heur-restart-def*)
obtain *M2* **where**
‹(*?K* # *?M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))›
using *get-all-ann-decomposition-ex*[*of* ‹*lit-of ?K*› *?M1 ?M2*]
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *T*[*symmetric*])
by *blast*
moreover have ‹*count-decided ?M1* = *0*›
using *n-d H*
by (*subst* (*asm*)(*1*) *T*, *subst* (*asm*)(*11*)*T*, *subst T*) *auto*
moreover have ‹*x* = *length ?M1*›
using *n-d H* **by** *auto*
ultimately show ‹∃ *M1*. (∃ *M2 K*. (*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) ∧
count-decided M1 = *0* ∧ *x* = *length M1* ›
by *blast*
qed

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *isa-length-trail* (*get-trail-wl-heur S*) = *length* (*get-trail-wl T*)›
unfolding *isa-length-trail-def twl-st-heur-restart-ana-def twl-st-heur-restart-def trail-pol-def*
by (*cases S*) (*auto dest*: *ann-lits-split-reasons-map-lit-of*)

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
unfolding *count-decided-st-def twl-st-heur-restart-ana-def trail-pol-def twl-st-heur-restart-def*
by (*cases S*) (*auto simp*: *count-decided-st-heur-def*)

lemma *twl-st-heur-restart-trailD*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹
(*get-trail-wl-heur S*, *get-trail-wl T*)
∈ *trail-pol* (*all-init-atms* (*get-clauses-wl T*) (*get-unit-init-clss-wl T*))›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup M* ⟹ *k* < *length M* ⟹ *M* ! *k* = *Propagated L C* ⟹ *Decided L* ∉ *set M*›
apply (*auto dest*!: *split-list simp*: *nth-append nth-Cons defined-lit-def in-set-conv-nth*
split: *if-splits nat.splits*)
by (*metis no-dup-no-propa-and-dec nth-mem*)

lemma *remove-all-annot-true-clause-imp-wl-inv-length-cong*:
‹*remove-all-annot-true-clause-imp-wl-inv S xs T* ⟹
length xs = *length ys* ⟹ *remove-all-annot-true-clause-imp-wl-inv S ys T*›
by (*auto simp*: *remove-all-annot-true-clause-imp-wl-inv-def*
remove-all-annot-true-clause-imp-inv-def)

lemma *get-literal-and-reason*:
assumes
‹((*k*, *S*), *k'*, *T*) ∈ *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'). $L = L' \wedge C' =$ *the* $C \wedge C \neq$ *None*}
(*SPEC* (λp. *rev* (*get-trail-wl* T) ! k' = *Propagated* (*fst* p) (*snd* p)))⟩
proof −
have *n-d*: ⟨*no-dup* (*get-trail-wl* T)⟩ **and**
res: ⟨((k, S), k', T) ∈ *nat-rel* $\times_f$ *twl-st-heur-restart*⟩
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
from *no-dup-nth-proped-dec-notin*[*OF this*(*1*), *of* ⟨*length* (*get-trail-wl* T) − *Suc* k'⟩]
have *dec-notin*: ⟨*Decided* (*lit-of* (*rev* (*fst* T) ! k')) ∉ *set* (*fst* T)⟩
using *proped assms*(*2*) **by** (*cases* T; *cases* ⟨*rev* (*get-trail-wl* T) ! k'⟩)
(*auto simp*: *twl-st-heur-restart-def*
remove-one-annot-true-clause-one-imp-wl-D-pre-def state-wl-l-def
remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
remove-one-annot-true-clause-one-imp-pre-def rev-nth
dest: *no-dup-nth-proped-dec-notin*)
have k': ⟨k' < *length* (*get-trail-wl* T)⟩ **and** [*simp*]: ⟨*fst* T = *get-trail-wl* T⟩
using *proped assms*(*2*)
by (*cases* T; *auto simp*: *twl-st-heur-restart-def*
remove-one-annot-true-clause-one-imp-wl-D-pre-def state-wl-l-def
remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
remove-one-annot-true-clause-one-imp-pre-def; *fail*)+
define k'' **where** ⟨$k'' \equiv$ *length* (*get-trail-wl* T) − *Suc* k'⟩
have k'': ⟨k'' < *length* (*get-trail-wl* T)⟩
using k' **by** (*auto simp*: k''-*def*)
have ⟨*rev* (*get-trail-wl* T) ! k' = *get-trail-wl* T ! k''⟩
using k' k'' **by** (*auto simp*: k''-*def nth-rev*)
then have ⟨*rev-trail-nth* (*fst* T) k' $\in\#$ $\mathcal{L}_{all}$ (*all-init-atms* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T))⟩
using k'' *assms nth-mem*[*OF* k']
by (*auto simp*: *twl-st-heur-restart-ana-def rev-trail-nth-def*
trail-pol-alt-def twl-st-heur-restart-def)
then have *1*: ⟨(*SPEC* (λp. *rev* (*get-trail-wl* T) ! k' = *Propagated* (*fst* p) (*snd* p))) =
do {
$L \leftarrow$ *RETURN* (*rev-trail-nth* (*fst* T) k');
ASSERT($L \in\#$ $\mathcal{L}_{all}$ (*all-init-atms* (*get-clauses-wl* T) (*get-unit-init-clss-wl* T)));
$C \leftarrow$ (*get-the-propagation-reason* (*fst* T) L);
ASSERT($C \neq$ *None*);
RETURN (L, *the* C)
}⟩
using *proped dec-notin* k' *nth-mem*[*OF* k''] *no-dup-same-annotD*[*OF n-d*]
apply (*subst order-class.eq-iff*)
apply (*rule conjI*)
subgoal
unfolding *get-the-propagation-reason-def*
by (*cases* ⟨*rev* (*get-trail-wl* T) ! k'⟩)
(*auto simp*: *RES-RES-RETURN-RES rev-trail-nth-def*
get-the-propagation-reason-def lits-of-def rev-nth
RES-RETURN-RES
dest: *split-list*
simp flip: k''-*def*
intro!: *le-SPEC-bindI*[*of* - ⟨*Some* (*mark-of* (*get-trail-wl* T ! k''))⟩])
subgoal

apply (*cases* ‹*rev* (*get-trail-wl* *T*) ! *k'*›)
apply (*auto simp*: *RES-RES-RETURN-RES rev-trail-nth-def*
get-the-propagation-reason-def lits-of-def rev-nth
RES-RETURN-RES
simp flip: *k''-def*
dest: *split-list*
intro!: *exI*[*of* - ‹*Some* (*mark-of* (*rev* (*fst* *T*) ! *k'*))›])
apply (*subst RES-ASSERT-moveout*)
apply (*auto simp*: *RES-RETURN-RES*
dest: *split-list*)
done
done

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*subst 1*)
apply (*refine-rcg*
isa-trail-nth-rev-trail-nth[*THEN fref-to-Down-curry*, *unfolded comp-def*,
of - - - - ‹(*all-init-atms* (*get-clauses-wl* *T*) (*get-unit-init-clss-wl* *T*))›]
get-the-propagation-reason-pol[*THEN fref-to-Down-curry*, *unfolded comp-def*,
of ‹(*all-init-atms* (*get-clauses-wl* *T*) (*get-unit-init-clss-wl* *T*))›])
subgoal using *k'* **by** *auto*
subgoal using *assms* **by** (*cases* *S*; *auto dest*: *twl-st-heur-restart-trailD*)
subgoal by *auto*
subgoal for *K K'*
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
subgoal
by *auto*
done
qed

lemma *red-in-dom-number-of-learned-ge1*: ‹$C' \in\#$ *dom-m baa* $\Longrightarrow$ ¬ *irred baa* C' $\Longrightarrow$ *Suc 0* ≤ *size* (*learned-clss-l baa*)›
by (*auto simp*: *ran-m-def dest*!: *multi-member-split*)

lemma *mark-garbage-heur2-remove-and-add-cls-l*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ (*C*, *C'*) ∈ *Id* ⟹
C ∈# *dom-m* (*get-clauses-wl* *T*) ⟹
mark-garbage-heur2 C S
≤ ⇓ (*twl-st-heur-restart-ana r*) (*remove-and-add-cls-l* *C'* *T*)›
unfolding *mark-garbage-heur2-def remove-and-add-cls-l-def*
apply (*cases* *S*; *cases* *T*)
by (*auto simp*: *twl-st-heur-restart-def arena-lifting*
valid-arena-extra-information-mark-to-delete'
all-init-atms-fmdrop-add-mset-unit learned-clss-l-l-fmdrop
learned-clss-l-l-fmdrop-irrelev twl-st-heur-restart-ana-def
size-Diff-singleton red-in-dom-number-of-learned-ge1 intro!: *ASSERT-leI*
dest: *in-vdom-m-fmdropD*)

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) ∈
nat-rel $\times_f$ *twl-st-heur-restart-ana r* $\rightarrow_f$ ⟨*nat-rel* $\times_f$ *twl-st-heur-restart-ana r*⟩*nres-rel*›
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def*
remove-one-annot-true-clause-one-imp-wl-D-def case-prod-beta uncurry-def

apply (*intro frefI nres-relI*)
subgoal for *x y*
apply (*refine-rcg get-literal-and-reason*[**where** *r*=*r*]
isasat-replace-annot-in-trail-replace-annot-in-trail-spec
[**where** *r*=*r*, *THEN fref-to-Down-curry2*]
mark-garbage-heur2-remove-and-add-cls-l[**where** *r*=*r*])
subgoal by *auto*
subgoal unfolding *remove-one-annot-true-clause-one-imp-wl-D-pre-def*
by *auto*
subgoal unfolding *remove-one-annot-true-clause-one-imp-wl-D-pre-def*
remove-one-annot-true-clause-one-imp-wl-pre-def
remove-one-annot-true-clause-one-imp-pre-def
by (*cases x*; *cases y*)
(*auto simp*: *uint32-max-def twl-st-heur-restart-def twl-st-heur-restart-ana-def*
state-wl-l-def trail-pol-alt-def)
subgoal by *auto*
subgoal by (*cases x*, *cases y*) *auto*
subgoal by *auto*
subgoal
by (*cases x*, *cases y*)
(*fastforce simp*: *twl-st-heur-restart-def*
trail-pol-alt-def)+
subgoal
by (*cases x*, *cases y*)
(*fastforce simp*: *twl-st-heur-restart-def*
trail-pol-alt-def)+
subgoal
by (*cases x*, *cases y*)
(*fastforce simp*: *twl-st-heur-restart-def*
trail-pol-alt-def)+
subgoal for *p pa S Sa*
unfolding *mark-garbage-pre-def*
arena-is-valid-clause-idx-def
prod.case
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac S*, *case-tac Sa*)
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
subgoal for *p pa S Sa*
unfolding *arena-is-valid-clause-vdom-def*
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac S*, *case-tac Sa*)
apply (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
done
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*cases x*, *cases y*) *fastforce*
done
done

lemma *RES-RETURN-RES5*:
  ‹*SPEC* Φ ≫= (λ(*T1*, *T2*, *T3*, *T4*, *T5*). *RETURN* (*f T1 T2 T3 T4 T5*)) =
    RES ((λ(*a*, *b*, *c*, *d*, *e*). *f a b c d e*) ‘ {*T*. Φ *T*})›
  using *RES-RETURN-RES*[*of* ‹*Collect* Φ› ‹λ(*a*, *b*, *c*, *d*, *e*). *f a b c d e*›]
  apply (*subst* (*asm*)(*2*) *split-prod-bound*)
  apply (*subst* (*asm*)(*3*) *split-prod-bound*)
  apply (*subst* (*asm*)(*4*) *split-prod-bound*)
  apply (*subst* (*asm*)(*5*) *split-prod-bound*)
  by *simp*

lemma *RES-RETURN-RES6*:
  ‹*SPEC* Φ ≫= (λ(*T1*, *T2*, *T3*, *T4*, *T5*, *T6*). *RETURN* (*f T1 T2 T3 T4 T5 T6*)) =
    RES ((λ(*a*, *b*, *c*, *d*, *e*, *f'*). *f a b c d e f'*) ‘ {*T*. Φ *T*})›
  using *RES-RETURN-RES*[*of* ‹*Collect* Φ› ‹λ(*a*, *b*, *c*, *d*, *e*, *f'*). *f a b c d e f'*›]
  apply (*subst* (*asm*)(*2*) *split-prod-bound*)
  apply (*subst* (*asm*)(*3*) *split-prod-bound*)
  apply (*subst* (*asm*)(*4*) *split-prod-bound*)
  apply (*subst* (*asm*)(*5*) *split-prod-bound*)
  apply (*subst* (*asm*)(*6*) *split-prod-bound*)
  by *simp*

lemma *RES-RETURN-RES7*:
  ‹*SPEC* Φ ≫= (λ(*T1*, *T2*, *T3*, *T4*, *T5*, *T6*, *T7*). *RETURN* (*f T1 T2 T3 T4 T5 T6 T7*)) =
    RES ((λ(*a*, *b*, *c*, *d*, *e*, *f'*, *g*). *f a b c d e f' g*) ‘ {*T*. Φ *T*})›
  using *RES-RETURN-RES*[*of* ‹*Collect* Φ› ‹λ(*a*, *b*, *c*, *d*, *e*, *f'*, *g*). *f a b c d e f' g*›]
  apply (*subst* (*asm*)(*2*) *split-prod-bound*)
  apply (*subst* (*asm*)(*3*) *split-prod-bound*)
  apply (*subst* (*asm*)(*4*) *split-prod-bound*)
  apply (*subst* (*asm*)(*5*) *split-prod-bound*)
  apply (*subst* (*asm*)(*6*) *split-prod-bound*)
  apply (*subst* (*asm*)(*7*) *split-prod-bound*)
  by *simp*

definition *find-decomp-wl0* **where**
  ‹*find-decomp-wl0* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) (*M'*, *N'*, *D'*, *NE'*, *UE'*, *Q'*, *W'*).
  (∃ *K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
     count-decided M' = *0*) ∧
   (*N'*, *D'*, *NE'*, *UE'*, *Q'*, *W'*) = (*N*, *D*, *NE*, *UE*, *Q*, *W*))›

definition *empty-Q-wl* :: ‹-› **where**
‹*empty-Q-wl* = (λ(*M'*, *N*, *D*, *NE*, *UE*, -, *W*). (*M'*, *N*, *D*, *NE*, *UE*, {#}, *W*))›

lemma *cdcl-twl-local-restart-wl-spec0-alt-def*:
  ‹*cdcl-twl-local-restart-wl-spec0* = (λ*S*.
    if count-decided (*get-trail-wl S*) > *0*
    then do {
      T ← *SPEC*(*find-decomp-wl0 S*);
      RETURN (*empty-Q-wl T*)
    } *else RETURN S*)›
  by (*intro ext*; *case-tac S*)
    (*fastforce simp*: *cdcl-twl-local-restart-wl-spec0-def*
 RES-RETURN-RES2 image-iff RES-RETURN-RES
 find-decomp-wl0-def empty-Q-wl-def
      dest: *get-all-ann-decomposition-exists-prepend*)

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*
    }
        $\le \Downarrow$ (*twl-st-heur-restart-ana r*) (*cdcl-twl-local-restart-wl-spec0 y*)›
proof −
  define *upd* :: ‹- ⇒ - ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
    ‹*upd M′ vm* = (λ (-, *N*, *D*, *Q*, *W*, -, φ, *clvls*, *cach*, *lbd*, *stats*).
      (*M′*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*))›
    for *M′* :: *trail-pol* **and** *vm*
  have *find-decomp-wl-st-int-alt-def*:
    ‹*find-decomp-wl-st-int* = (λ*highest S*. *do*{
      (*M′*, *vm*) ← *isa-find-decomp-wl-imp* (*get-trail-wl-heur S*) *highest* (*get-vmtf-heur S*);
      RETURN (*upd M′ vm S*)
    })›
    unfolding *upd-def find-decomp-wl-st-int-def*
    by (*auto intro*!: *ext*)

  have [*refine0*]: ‹*do* {
   (*M′*, *vm*) ←
     isa-find-decomp-wl-imp (*get-trail-wl-heur S*) *0* (*get-vmtf-heur S*);
   RETURN (*upd M′ vm S*)
 } $\le \Downarrow$ {((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
   ccount,
       vdom, *avdom*, *lcount*, *opts*),
      T).
        ((*M′*, *N′*, *D′*, *isa-length-trail M′*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
          slow-ema, *restart-info-restart-done ccount*, *vdom*, *avdom*, *lcount*, *opts*),
   (*empty-Q-wl T*)) ∈ *twl-st-heur-restart-ana r* ∧
   isa-length-trail-pre M′} (*SPEC* (*find-decomp-wl0 y*))›
     (**is** ‹- ≤ ⇓ *?A* -›)
    if
      ‹*0* < *count-decided-st-heur S*› **and**
      ‹*0* < *count-decided* (*get-trail-wl y*)›
  proof −
    have *A*:
      ‹*?A* = {((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*,
   ccount,
       vdom, *avdom*, *lcount*, *opts*),
      T).
        ((*M′*, *N′*, *D′*, *length* (*get-trail-wl T*), *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
          slow-ema, *restart-info-restart-done ccount*, *vdom*, *avdom*, *lcount*, *opts*),
   (*empty-Q-wl T*)) ∈ *twl-st-heur-restart-ana r* ∧
   isa-length-trail-pre M′}›
     supply[[*goals-limit=1*]]
      apply (*rule* ; *rule*)
      subgoal for *x*
        apply *clarify*
 apply (*frule twl-st-heur-restart-isa-length-trail-get-trail-wl*)
        by (*auto simp*: *trail-pol-def empty-Q-wl-def*

isa-length-trail-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal for x
apply *clarify*
apply (*frule twl-st-heur-restart-isa-length-trail-get-trail-wl*)
by (*auto simp*: *trail-pol-def empty-Q-wl-def*
isa-length-trail-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done

let *?𝒜* = ⟨*all-init-atms-st y*⟩
have ⟨*get-vmtf-heur S* ∈ *isa-vmtf ?𝒜* (*get-trail-wl y*)⟩**and**
n-d: ⟨*no-dup* (*get-trail-wl y*)⟩
using *Sy*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
then obtain *vm'* **where**
vm': ⟨(*get-vmtf-heur S*, *vm'*) ∈ *Id* $\times_f$ *distinct-atoms-rel ?𝒜*⟩ **and**
vm: ⟨*vm'* ∈ *vmtf* (*all-init-atms-st y*) (*get-trail-wl y*)⟩
unfolding *isa-vmtf-def*
by *force*

have *find-decomp-w-ns-pre*:
⟨*find-decomp-w-ns-pre* (*all-init-atms-st y*) ((*get-trail-wl y*, *0*), *vm'*)⟩
using *that assms vm' vm* **unfolding** *find-decomp-w-ns-pre-def*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
dest: *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*)
have *1*: ⟨*isa-find-decomp-wl-imp* (*get-trail-wl-heur S*) *0* (*get-vmtf-heur S*) ≤
⇓ ({(*M*, *M'*). (*M*, *M'*) ∈ *trail-pol ?𝒜* ∧ *count-decided M'* = *0*} $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel ?𝒜*))
(*find-decomp-w-ns ?𝒜* (*get-trail-wl y*) *0 vm'*)⟩
apply (*rule order-trans*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*,
of ⟨*get-trail-wl y*⟩ *0 vm'* - - - *?𝒜*])
subgoal using *that* **by** *auto*
subgoal
using *Sy vm'*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
apply (*rule order-trans*, *rule ref-two-step'*)
apply (*rule find-decomp-wl-imp-find-decomp-wl'*[*THEN fref-to-Down-curry2*,
of ?𝒜 ⟨*get-trail-wl y*⟩ *0 vm'*])
subgoal by (*rule find-decomp-w-ns-pre*)
subgoal by *auto*
subgoal
using *n-d*
by (*fastforce simp*: *find-decomp-w-ns-def conc-fun-RES Image-iff*)
done
show *?thesis*
supply [[*goals-limit=1*]] **unfolding** *A*
apply (*rule bind-refine-res*[*OF - 1*[*unfolded find-decomp-w-ns-def conc-fun-RES*]])
apply (*case-tac y*, *cases S*)
apply *clarify*
apply (*rule RETURN-SPEC-refine*)
using *assms*
by (*auto simp*: *upd-def find-decomp-wl0-def*
intro!: *RETURN-SPEC-refine simp*: *twl-st-heur-restart-def out-learned-def*
empty-Q-wl-def twl-st-heur-restart-ana-def
intro: *isa-vmtfI isa-length-trail-pre dest*: *no-dup-appendD*)

qed

show *?thesis*
unfolding *find-decomp-wl-st-int-alt-def*
cdcl-twl-local-restart-wl-spec0-alt-def
apply *refine-vcg*
subgoal
using *Sy* **by** (*auto simp*: *twl-st-heur-restart-count-decided-st-alt-def*)
subgoal
unfolding *empty-Q-def empty-Q-wl-def*
by *refine-vcg*
(*simp-all add*: *twl-st-heur-restart-isa-length-trail-get-trail-wl*)
subgoal
using *Sy* **.**
done
qed

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*›
apply (*induction M rule*: *ann-lit-list-induct*)
subgoal by *auto*
subgoal for *L M*
by *auto*
subgoal for *L C M*
by (*cases* ‹*get-all-ann-decomposition M*›) *fastforce+*
done

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
assume *?A*
then obtain *K M1 M2* **where**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
[*simp*]: ‹*count-decided M1* = *0*› **and**
k-M1: ‹*length M1* = *k*›
by *auto*
then have ‹*k* < *length M*›
by *auto*
moreover have ‹*rev M* ! *k* = *Decided K*›
using *decomp*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *nth-append*
simp flip: *k-M1*)
moreover have ‹*get-level M* (*lit-of* (*rev M* ! *k*)) = *1*›
using *assms decomp*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *get-level-append-if nth-append*
simp flip: *k-M1*)

ultimately show *?B*
  using *decomp* **by** *auto*
next
  assume *?B*
  define *K* **where** ‹*K* = *lit-of* (*rev M* ! *k*)›
  obtain *M1 M2* **where**
    M: ‹*M* = *M2* @ *Decided K* # *M1*› **and**
    k-M1: ‹*length M1* = *k*›
    apply (*subst* (*asm*) *append-take-drop-id*[*of* ‹*length M* − *Suc k*›, *symmetric*])
    apply (*subst* (*asm*) *Cons-nth-drop-Suc*[*symmetric*])
    unfolding *K-def*
    subgoal using ‹*?B*› **by** *auto*
    subgoal using ‹*?B*› *K-def* **by** (*cases* ‹*rev M* ! *k*›) (*auto simp*: *rev-nth*)
    done
  moreover have ‹*count-decided M1* = *0*›
    using *assms* ‹*?B*› **unfolding** *M*
    by (*auto simp*: *nth-append k-M1*)
  ultimately show *?A*
    using *get-all-ann-decomposition-ex*[*of K M1 M2*]
    unfolding *M*
    by *auto*
qed

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*›
  unfolding *remove-one-annot-true-clause-imp-wl-D-def*
    remove-one-annot-true-clause-imp-wl-D-heur-def
  apply (*intro frefI nres-relI*)
  apply (*refine-vcg*
    WHILEIT-refine[**where** *R* = ‹*nat-rel* $\times_r$ *twl-st-heur-restart-ana r*›]
   remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D[*THEN*
      fref-to-Down-curry])
  subgoal by (*auto simp*: *trail-pol-alt-def isa-length-trail-pre-def*
    twl-st-heur-restart-def twl-st-heur-restart-ana-def)
  subgoal by (*auto simp*: *twl-st-heur-restart-isa-length-trail-get-trail-wl*
    twl-st-heur-restart-count-decided-st-alt-def)
  subgoal for *x y*
    apply (*rule order-trans*)
    apply (*rule get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*[*THEN fref-to-Down-curry*,
        of ‹*get-trail-wl y*› *0* - - ‹*all-init-atms-st y*›])
    subgoal by (*auto simp*: *get-pos-of-level-in-trail-pre-def*
      twl-st-heur-restart-count-decided-st-alt-def)
    subgoal by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
    subgoal
      apply (*subst get-pos-of-level-in-trail-decomp-iff*)
      apply (*solves* ‹*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*›)
      apply (*auto simp*: *get-pos-of-level-in-trail-def*
        twl-st-heur-restart-count-decided-st-alt-def)
      done
    done
    subgoal by *auto*
    subgoal for *x y k k′ T T′*
      apply (*subst* (*asm*)(*12*) *surjective-pairing*)
      apply (*subst* (*asm*)(*10*) *surjective-pairing*)
      unfolding *remove-one-annot-true-clause-imp-wl-D-heur-inv-def*

prod-rel-iff
apply (*subst* (*10*) *surjective-pairing*, *subst prod.case*)
by (*fastforce intro*: *twl-st-heur-restart-anaD simp*: *twl-st-heur-restart-anaD*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

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 −
have *mark-to-delete-clauses-wl-D-alt-def*:
‹*mark-to-delete-clauses-wl2-D* = (λ*S*. *do* {
ASSERT(*mark-to-delete-clauses-wl-D-pre S*);
S ← *reorder-vdom-wl S*;
xs ← *collect-valid-indices-wl S*;
l ← *SPEC*(λ-::*nat*. *True*);
(-, *S*, -) ← $WHILE_T{}^{mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl2\text{-}D\text{-}inv\ S\ xs}$
(λ(*i*, *T*, *xs*). *i* < *length xs*)
(λ(*i*, *T*, *xs*). *do* {
if(*xs*!*i* ∉# *dom-m* (*get-clauses-wl T*)) *then RETURN* (*i*, *T*, *delete-index-and-swap xs i*)
else do {
ASSERT(*0* < *length* (*get-clauses-wl T*∝(*xs*!*i*)));
ASSERT (*get-clauses-wl T* ∝ (*xs* ! *i*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st T*));
can-del ← *SPEC*(λ*b*. *b* ⟶
(*Propagated* (*get-clauses-wl T*∝(*xs*!*i*)!*0*) (*xs*!*i*) ∉ *set* (*get-trail-wl T*)) ∧
¬*irred* (*get-clauses-wl T*) (*xs*!*i*) ∧ *length* (*get-clauses-wl T* ∝ (*xs*!*i*)) ≠ *2*);
ASSERT(*i* < *length xs*);
if can-del
then
RETURN (*i*, *mark-garbage-wl* (*xs*!*i*) *T*, *delete-index-and-swap xs i*)
else
RETURN (*i*+*1*, *T*, *xs*)
}
})
(*l*, *S*, *xs*);
RETURN S
})›
unfolding *mark-to-delete-clauses-wl2-D-def reorder-vdom-wl-def*
by (*auto intro*!: *ext*)

have [*refine0*]: ‹*RETURN* (*get-avdom x*) ≤ ⇓ {(*xs*, *xs*′). *xs* = *xs*′ ∧ *xs* = *get-avdom x*} (*collect-valid-indices-wl y*)›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*›
for *x y*
proof −
show *?thesis* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed
have *init-rel*[*refine0*]: ‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r* ⟹
(*l*, *la*) ∈ *nat-rel* ⟹

$((l, x), la, y) \in$ *nat-rel* $\times_f$ $\{(S, T).\ (S, T) \in$ *twl-st-heur-restart-ana r* $\wedge$ *get-avdom S* $=$ *get-avdom x*$\}$›
for *x y l la*
by *auto*

have *get-the-propagation-reason*:
‹*get-the-propagation-reason-pol* (*get-trail-wl-heur x2b*)
(*arena-lit* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b* + *0*))
$\leq \Downarrow \{(D, b).\ b \longleftrightarrow ((D \neq$ *Some* (*get-avdom x2b* ! *x1b*)) $\wedge$
arena-lbd (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) $>$ *MINIMUM-DELETION-LBD* $\wedge$
arena-status (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) $=$ *LEARNED*) $\wedge$
arena-length (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) $\neq$ *two-uint32-nat* $\wedge$
$\neg$*marked-as-used* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*)$\}$
(*SPEC*
($\lambda b.\ b \longrightarrow$
Propagated (*get-clauses-wl x1a* $\propto$ (*x2a* ! *x1*) ! *0*) (*x2a* ! *x1*)
$\notin$ *set* (*get-trail-wl x1a*) $\wedge$
$\neg$ *irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*) $\wedge$
length (*get-clauses-wl x1a* $\propto$ (*x2a* ! *x1*)) $\neq$ *2*))›
if
‹$(x, y) \in$ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(S, Sa)
$\in \{(U, V).$
$(U, V) \in$ *twl-st-heur-restart-ana r* $\wedge$
$V = y \wedge$
(*mark-to-delete-clauses-wl-D-pre y* $\longrightarrow$
mark-to-delete-clauses-wl-D-pre V) $\wedge$
(*mark-to-delete-clauses-wl-D-heur-pre x* $\longrightarrow$
mark-to-delete-clauses-wl-D-heur-pre U)$\}$› **and**
‹$(ys, xs) \in \{(xs, xs').\ xs = xs' \wedge xs =$ *get-avdom S*$\}$› **and**
‹$(l, la) \in$ *nat-rel*› **and**
‹$la \in \{$-. *True*$\}$› **and**
xa-x': ‹(xa, x')
$\in$ *nat-rel* $\times_f$ $\{(Sa, T, xs).\ (Sa, T) \in$ *twl-st-heur-restart-ana r* $\wedge$ $xs =$ *get-avdom Sa*$\}$› **and**
‹*case xa of* $(i, S) \Rightarrow i <$ *length* (*get-avdom S*)› **and**
‹*case x' of* $(i, T, xs) \Rightarrow i <$ *length xs*› **and**
‹$x1b <$ *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)› **and**
‹$\neg\ \neg$ *clause-not-marked-to-delete-heur x2b* (*get-avdom x2b* ! *x1b*)› **and**
‹$\neg$ *x2a* ! *x1* $\notin\#$ *dom-m* (*get-clauses-wl x1a*)› **and**
‹$0 <$ *length* (*get-clauses-wl x1a* $\propto$ (*x2a* ! *x1*))› **and**
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-avdom x2b* ! *x1b*), *0*)› **and**
st:
‹$x2 = (x1a, x2a)$›
‹$x' = (x1, x2)$›
‹$xa = (x1b, x2b)$› **and**
L: ‹*get-clauses-wl x1a* $\propto$ (*x2a* ! *x1*) ! *0* $\in\#$ $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
for *x y S Sa xs' xs l la xa x' x1 x2 x1a x2a x1' x2' x3 x1b ys x2b*
proof −
have *L*: ‹*arena-lit* (*get-clauses-wl-heur x2b*) (*x2a* ! *x1*) $\in\#$ $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
using *L that* **by** (*auto simp*: *twl-st-heur-restart st arena-lifting dest*: $\mathcal{L}_{all}$*-init-all twl-st-heur-restart-anaD*)

show *?thesis*

apply (*rule order.trans*)
apply (*rule get-the-propagation-reason-pol*[*THEN fref-to-Down-curry*,
of ‹*all-init-atms-st x1a*› ‹*get-trail-wl x1a*›
‹*arena-lit* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b* + *0*)›])
subgoal
using *xa-x′ L* **by** (*auto simp add*: *twl-st-heur-restart-def st*)
subgoal
using *xa-x′* **by** (*auto simp add*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def st*)
using *that* **unfolding** *get-the-propagation-reason-def* **apply** −
by (*auto simp*: *twl-st-heur-restart lits-of-def get-the-propagation-reason-def*
conc-fun-RES
dest: *twl-st-heur-restart-same-annotD imageI*[*of - - lit-of*]
dest!: *twl-st-heur-restart-anaD*)
qed
have ‹((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
slow-ema, *ccount*, *vdom*, *avdom′*, *lcount*),
S′)
∈ *twl-st-heur-restart* ⟹
((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
slow-ema, *ccount*, *vdom*, *avdom*, *lcount*),
S′)
∈ *twl-st-heur-restart*›
if ‹*mset avdom* ⊆# *mset avdom′*›
for *M′ N′ D′ j W′ vm φ clvls cach lbd outl stats fast-ema slow-ema*
ccount vdom lcount S′ avdom′ avdom
using *that* **unfolding** *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
by *auto*
then have *mark-to-delete-clauses-wl-D-heur-pre-vdom′*:
‹*mark-to-delete-clauses-wl-D-heur-pre* (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom′*, *lcount*) ⟹
mark-to-delete-clauses-wl-D-heur-pre (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
fast-ema, *slow-ema*, *ccount*, *vdom*, *avdom*, *lcount*)›
if ‹*mset avdom* ⊆# *mset avdom′*›
for *M′ N′ D′ j W′ vm φ clvls cach lbd outl stats fast-ema slow-ema avdom avdom′*
ccount vdom lcount
using *that twl-st-heur-restart-anaD*
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def*
by *metis*
have [*refine0*]:
‹*sort-vdom-heur S* ≤ ⇓ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧ *V* = *T* ∧
(*mark-to-delete-clauses-wl-D-pre T* ⟶ *mark-to-delete-clauses-wl-D-pre V*) ∧
(*mark-to-delete-clauses-wl-D-heur-pre S* ⟶ *mark-to-delete-clauses-wl-D-heur-pre U*)}
(*reorder-vdom-wl T*)›
if ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **for** *S T*
using *that* **unfolding** *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*refine-rcg ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset*
size-mset-mono)
apply (*rule specify-left*)
apply (*rule-tac N1* = ‹*get-clauses-wl T*› **and** *vdom1* = ‹(*get-vdom S*)› **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF - - - remove-deleted-clauses-from-avdom*])
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*

x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-ana-def
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *exI*[*of* - ‹*set* (*get-vdom S*)›] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
dest!: *size-mset-mono valid-arena-vdom-subset*)
subgoal
apply (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*)
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
intro: *mark-to-delete-clauses-wl-D-heur-pre-vdom′*
dest: *mset-eq-setD*)
done
have *already-deleted*:
‹((*x1b*, *delete-index-vdom-heur x1b x2b*), *x1*, *x1a*,
delete-index-and-swap x2a x1)
$\in$ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) $\in$ *twl-st-heur-restart-ana r* $\wedge$ *xs* = *get-avdom Sa*}›
if
‹(*x*, *y*) $\in$ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
$\in$ {(*U*, *V*).
(*U*, *V*) $\in$ *twl-st-heur-restart-ana r* $\wedge$
V = *y* $\wedge$
(*mark-to-delete-clauses-wl-D-pre y* $\longrightarrow$
mark-to-delete-clauses-wl-D-pre V) $\wedge$
(*mark-to-delete-clauses-wl-D-heur-pre x* $\longrightarrow$
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*ys*, *xs*) $\in$ {(*xs*, *xs′*). *xs* = *xs′* $\wedge$ *xs* = *get-avdom S*}› **and**
‹(*l*, *la*) $\in$ *nat-rel*› **and**
‹*la* $\in$ {-. *True*}› **and**
xx: ‹(*xa*, *x′*)
$\in$ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) $\in$ *twl-st-heur-restart-ana r* $\wedge$ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) $\Rightarrow$ *i* < *length* (*get-avdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) $\Rightarrow$ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl2-D-inv Sa xs x′*› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
x1b: ‹*x1b* < *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)› **and**
‹¬ *clause-not-marked-to-delete-heur x2b* (*get-avdom x2b* ! *x1b*)› **and**

⟨*x2a* ! *x1* ∉# *dom-m* (*get-clauses-wl x1a*)⟩
for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa*
proof −
show *?thesis*
using *xx x1b* **unfolding** *st*
by (*auto simp*: *twl-st-heur-restart-ana-def delete-index-vdom-heur-def*
twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
learned-clss-l-l-fmdrop size-remove1-mset-If
intro: *valid-arena-extra-information-mark-to-delete′*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
qed

have *get-learned-count-ge*: ⟨*1* ≤ *get-learned-count x2b*⟩
if
xy: ⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
⟨(*xa*, *x′*)
∈ *nat-rel* $\times_f$
{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩ **and**
⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*x′* = (*x1*, *x2*)⟩ **and**
⟨*xa* = (*x1b*, *x2b*)⟩ **and**
dom: ⟨¬ *x2a* ! *x1* ∉# *dom-m* (*get-clauses-wl x1a*)⟩ **and**
⟨*can-del*
∈ {*b*. *b* ⟶
Propagated (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*) (*x2a* ! *x1*)
∉ *set* (*get-trail-wl x1a*) ∧
¬ *irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*) ∧
length (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*)) ≠ *2*}⟩ **and**
⟨*can-del*⟩ **for** *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b D can-del*
proof −
have ⟨¬*irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)⟩ **and** ⟨(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*⟩
using *that* **by** (*auto simp*:)
then show *?thesis*
using *dom* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*)
qed
have *init*:
⟨(*u*, *xs*) ∈ {(*xs*, *xs′*). *xs* = *xs′* ∧ *xs* = *get-avdom S*} ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
(*S*, *Sa*) ∈ *twl-st-heur-restart-ana r* ⟹
mark-to-delete-clauses-wl2-D-inv Sa xs (*la*, *Sa*, *xs*) ⟹
((*l*, *S*), *la*, *Sa*, *xs*) ∈ *nat-rel* $\times_f$
{(*Sa*, (*T*, *xs*)). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩
for *x y S Sa xs l la u*
by *auto*
have [*refine0*]: ⟨*mark-clauses-as-unused-wl-D-heur i T*
≤ *SPEC*
(λ*x*. *incr-restart-stat x* ≤ *SPEC* (λ*c*. (*c*, *S*) ∈ *twl-st-heur-restart-ana r*))⟩
if ⟨(*T*, *S*) ∈ *twl-st-heur-restart-ana r*⟩ **for** *S T i*
by (*rule order-trans*, *rule mark-clauses-as-unused-wl-D-heur*[*OF that*, *of i*])
(*use that* **in** ⟨*auto simp*: *conc-fun-RES incr-restart-stat-def*
twl-st-heur-restart-ana-def twl-st-heur-restart-def⟩)
show *?thesis*
supply *sort-vdom-heur-def*[*simp*] *twl-st-heur-restart-anaD*[*dest*]

unfolding *mark-to-delete-clauses-wl-D-heur-alt-def mark-to-delete-clauses-wl-D-alt-def access-lit-in-clauses-heur-def Let-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg sort-vdom-heur-reorder-vdom-wl*[*THEN fref-to-Down*])
subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def* **by** (*fast dest*: *twl-st-heur-restart-anaD*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule init*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-vdom-at-pre-def*)
subgoal for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d*
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def prod.simps*
by (*rule exI*[*of - ‹get-clauses-wl x2a›*], *rule exI*[*of - ‹set* (*get-vdom x2d*)*›*])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-get-avdom-nth-get-vdom twl-st-heur-restart-anaD*)
subgoal
by (*auto simp*: *twl-st-heur-restart dest*!: *twl-st-heur-restart-anaD*)
subgoal
by (*rule already-deleted*)
subgoal for *x y - - - - - xs l la xa x′ x1 x2 x1a x2a*
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of - ‹get-avdom x2a* ! *x1a›*], *simp*, *rule exI*[*of - ‹get-clauses-wl x1›*])
(*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal premises *p* **using** *p*(*7*−) **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule get-the-propagation-reason*; *assumption*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps get-clause-LBD-pre-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)*›*])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)*›*])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of - ‹get-clauses-wl x1a›*], *rule exI*[*of - ‹set* (*get-vdom x2b*)*›*])
(*auto simp*: *twl-st-heur-restart arena-dom-status-iff dest*: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal unfolding *marked-as-used-pre-def* **by** *fast*
subgoal
by (*auto simp*: *twl-st-heur-restart*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps mark-garbage-pre-def*

arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal by (*rule get-learned-count-ge*)
subgoal
by (*auto intro*!: *mark-garbage-heur-wl-ana*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps mark-garbage-pre-def arena-act-pre-def*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal
by (*auto intro*!: *mark-unused-st-heur-ana*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal
by (*auto simp*:)
done
qed

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-*$\mathcal{L}_{all}$ = (λ*f* $\mathcal{A}_0$ *I x*. *do* {
$\mathcal{A}$ ← *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}_0$ ∧ *distinct-mset* $\mathcal{A}$);
(-, *x*) ← $WHILE_T{}^{\lambda(-,\ x).\ I\ x}$
(λ($\mathcal{B}$, -). $\mathcal{B}$ ≠ {#})
(λ($\mathcal{B}$, *x*). *do* {
ASSERT($\mathcal{B}$ ≠ {#});
A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
x ← *f A x*;
RETURN (*remove1-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*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
nempty: ‹$\mathcal{A}$ ≠ {#}› ‹*isasat-input-bounded* $\mathcal{A}$›

```
  shows ‹iterate-over-VMTF f I (ns, Some fst-As) x ≤ ⇓ Id (iterate-over-ℒall f 𝒜 I x)›
proof –
  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    ‹fst-As = hd (ys' @ xs')› and
    ‹lst-As = last (ys' @ xs')› and
    vmtf-ℒ: ‹vmtf-ℒall 𝒜 M ((set xs', set ys'), to-remove)› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    le: ‹∀ L∈atms-of (ℒall 𝒜). L < length ns›
    using vmtf unfolding vmtf-def
    by blast
  define zs where ‹zs = ys' @ xs'›
  define is-lasts where
    ‹is-lasts ℬ n m ⟷ set-mset ℬ = set (drop m zs) ∧ set-mset ℬ ⊆ set-mset 𝒜 ∧
        distinct-mset ℬ ∧
        card (set-mset ℬ) ≤ length zs ∧
        card (set-mset ℬ) + m = length zs ∧
        (n = option-hd (drop m zs)) ∧
        m ≤ length zs› for ℬ and n :: ‹nat option› and m
  have card-𝒜: ‹card (set-mset 𝒜) = length zs›
    ‹set-mset 𝒜 = set zs› and
    nempty': ‹zs ≠ []› and
    dist-zs: ‹distinct zs›
    using vmtf-ℒ vmtf-ns-distinct[OF vmtf-ns] nempty
    unfolding vmtf-ℒall-def eq-commute[of - ‹atms-of -›] zs-def
    by (auto simp: atms-of-ℒall-𝒜in card-Un-disjoint distinct-card)
  have hd-zs-le: ‹hd zs < length ns›
    using vmtf-ns-le-length[OF vmtf-ns, of ‹hd zs›] nempty'
    unfolding zs-def[symmetric]
    by auto
  have [refine0]: ‹
      (the x1a, A) ∈ nat-rel ⟹
      x = x2b ⟹
      f (the x1a) x2b ≤ ⇓ Id (f A x)› for x1a A x x2b
    by auto
  define iterate-over-VMTF2 where
    ‹iterate-over-VMTF2 ≡ (λf (I :: 'a ⇒ bool) (vm :: (nat, nat) vmtf-node list, n) x. do {
      let - = remdups-mset 𝒜;
      (-, -, x) ← WHILE_T^λ(n,m, x). I x
        (λ(n, -, -). n ≠ None)
        (λ(n, m, 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)), Suc m, x)
        })
        (n, 0, x);
      RETURN x
    })›
  have iterate-over-VMTF2-alt-def:
    ‹iterate-over-VMTF2 ≡ (λf (I :: 'a ⇒ bool) (vm :: (nat, nat) vmtf-node list, n) x. do {
      (-, -, x) ← WHILE_T^λ(n,m, x). I x
        (λ(n, -, -). n ≠ None)
```

```
      (λ(n, m, 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)), Suc m, x)
      })
      (n, 0, x);
    RETURN x
  })›
  unfolding iterate-over-VMTF2-def by force
have nempty-iff: ‹(x1 ≠ None) = (x1b ≠ {#})›
if
  ‹(remdups-mset 𝒜, 𝒜') ∈ Id› and
  H: ‹(x, x') ∈ {((n, m, x), 𝒜', y). is-lasts 𝒜' n m ∧ x = y}› and
  ‹case x of (n, m, xa) ⇒ I xa› and
  ‹case x' of (uu-, x) ⇒ I x› and
  st[simp]:
    ‹x2 = (x1a, x2a)›
    ‹x = (x1, x2)›
    ‹x' = (x1b, xb)›
  for 𝒜' x x' x1 x2 x1a x2a x1b xb
proof
  show ‹x1b ≠ {#}› if ‹x1 ≠ None›
    using that H
    by (auto simp: is-lasts-def)
  show ‹x1 ≠ None› if ‹x1b ≠ {#}›
    using that H
    by (auto simp: is-lasts-def)
qed
have IH: ‹((get-next (ns ! the x1a), Suc x1b, xa), remove1-mset A x1, xb)
      ∈ {((n, m, x), 𝒜', y). is-lasts 𝒜' n m ∧ x = y}›
   if
    ‹(remdups-mset 𝒜, 𝒜') ∈ Id› and
    H: ‹(x, x') ∈ {((n, m, x), 𝒜', y). is-lasts 𝒜' n m ∧ x = y}› and
    ‹case x of (n, uu-, uua-) ⇒ n ≠ None› and
    nempty: ‹case x' of (ℬ, uu-) ⇒ ℬ ≠ {#}› and
    ‹case x of (n, m, xa) ⇒ I xa› and
    ‹case x' of (uu-, x) ⇒ I x› and
    st:
      ‹x' = (x1, x2)›
      ‹x2a = (x1b, x2b)›
      ‹x = (x1a, x2a)›
      ‹(xa, xb) ∈ Id› and
    ‹x1 ≠ {#}› and
    ‹x1a ≠ None› and
    A: ‹(the x1a, A) ∈ nat-rel› and
    ‹the x1a < length ns›
    for 𝒜' x x' x1 x2 x1a x2a x1b x2b A xa xb
proof −
  have [simp]: ‹distinct-mset x1› ‹x1b < length zs›
    using H A nempty
    apply (auto simp: st is-lasts-def simp flip: Cons-nth-drop-Suc)
    apply (cases ‹x1b = length zs›)
    apply auto
```

done
then have [*simp*]: ‹*zs* ! *x1b* ∉ *set* (*drop* (*Suc x1b*) *zs*)›
by (*auto simp*: *in-set-drop-conv-nth nth-eq-iff-index-eq dist-zs*)
have [*simp*]: ‹*length zs* − *Suc x1b* + *x1b* = *length zs* ⟷ *False*›
using ‹*x1b* < *length zs*› **by** *presburger*
have ‹*vmtf-ns* (*take x1b zs* @ *zs* ! *x1b* # *drop* (*Suc x1b*) *zs*) *m ns*›
using *vmtf-ns*
by (*auto simp*: *Cons-nth-drop-Suc simp flip*: *zs-def*)
from *vmtf-ns-last-mid-get-next-option-hd*[*OF this*]
show *?thesis*
using *H A st*
by (*auto simp*: *st is-lasts-def dist-zs distinct-card distinct-mset-set-mset-remove1-mset simp flip*: *Cons-nth-drop-Suc*)
qed
have *WTF*[*simp*]: ‹*length zs* − *Suc 0* = *length zs* ⟷ *zs* = []›
by (*cases zs*) *auto*
have *zs2*: ‹*set* (*xs'* @ *ys'*) = *set zs*›
by (*auto simp*: *zs-def*)
have *is-lasts-le*: ‹*is-lasts x1* (*Some A*) *x1b* ⟹ *A* < *length ns*› **for** *x2 xb x1b x1 A*
using *vmtf-*$\mathcal{L}$ *le nth-mem*[*of* ‹*x1b*› *zs*] **unfolding** *is-lasts-def prod.case vmtf-*$\mathcal{L}_{all}$*-def set-append*[*symmetric*]*zs-def*[*symmetric*] *zs2*
by (*auto simp*: *eq-commute*[*of* ‹*set zs*› ‹*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›] *hd-drop-conv-nth simp del*: *nth-mem*)
have *le-uint-max*: ‹*the x1a* ≤ *uint-max div 2*›
if
‹(*remdups-mset* $\mathcal{A}$, $\mathcal{A}'$) ∈ *Id*› **and**
‹(*x*, *x'*) ∈ {((*n*, *m*, *x*), $\mathcal{A}'$, *y*). *is-lasts* $\mathcal{A}'$ *n m* ∧ *x* = *y*}› **and**
‹*case x of* (*n*, *uu-*, *uua-*) ⇒ *n* ≠ *None*› **and**
‹*case x' of* ($\mathcal{B}$, *uu-*) ⇒ $\mathcal{B}$ ≠ {#}› **and**
‹*case x of* (*n*, *m*, *xa*) ⇒ *I xa*› **and**
‹*case x' of* (*uu-*, *x*) ⇒ *I x*› **and**
‹*x'* = (*x1*, *x2*)› **and**
‹*x2a* = (*x1b*, *xb*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹*x1* ≠ {#}› **and**
‹*x1a* ≠ *None*› **and**
‹(*the x1a*, *A*) ∈ *nat-rel*› **and**
‹*the x1a* < *length ns*›
for $\mathcal{A}'$ *x x' x1 x2 x1a x2a x1b xb A*
proof −
have ‹*the x1a* ∈# $\mathcal{A}$›
using *that* **by** (*auto simp*: *is-lasts-def*)
then show *?thesis*
using *nempty* **by** (*auto dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
qed
have ‹*iterate-over-VMTF2 f I* (*ns*, *Some fst-As*) *x* ≤ ⇓ *Id* (*iterate-over-*$\mathcal{L}_{all}$ *f* $\mathcal{A}$ *I x*)›
unfolding *iterate-over-VMTF2-def iterate-over-*$\mathcal{L}_{all}$*-def prod.case*
apply (*refine-vcg WHILEIT-refine*[**where** *R* = ‹{((*n* :: *nat option*, *m*::*nat*, *x*::'*a*), ($\mathcal{A}'$ :: *nat multiset*, *y*)).
is-lasts $\mathcal{A}'$ *n m* ∧ *x* = *y*}›])
subgoal by *simp*
subgoal by *simp*
subgoal
using *card-*$\mathcal{A}$ *fst-As nempty nempty' hd-conv-nth*[*OF nempty'*] *hd-zs-le* **unfolding** *zs-def*[*symmetric*] *is-lasts-def*
by (*simp-all add*: *eq-commute*[*of* ‹*remdups-mset* -›])

subgoal by *auto*
subgoal for $\mathcal{A}'$ *x x' x1 x2 x1a x2a x1b xb*
by (*rule nempty-iff*)
subgoal by *auto*
subgoal for $\mathcal{A}'$ *x x' x1 x2 x1a x2a x1b xb*
by (*simp add*: *is-lasts-def in-set-dropI*)
subgoal for $\mathcal{A}'$ *x x' x1 x2 x1a x2a x1b xb*
by (*auto simp*: *is-lasts-le*)
subgoal by (*rule le-uint-max*)
subgoal by *auto*
subgoal for $\mathcal{A}'$ *x x' x1 x2 x1a x2a x1b x2b A xa xb*
by (*rule IH*)
subgoal by *auto*
done
moreover have ‹*iterate-over-VMTF f I* (*ns*, *Some fst-As*) *x* ≤ ⇓ *Id* (*iterate-over-VMTF2 f I* (*ns*, *Some fst-As*) *x*)›
unfolding *iterate-over-VMTF2-alt-def iterate-over-VMTF-def prod.case*
by (*refine-vcg WHILEIT-refine*[**where** *R* = ‹{((*n* :: *nat option*, *x*::'*a*), (*n'* :: *nat option*, *m'*::*nat*, *x'*::'*a*)).
n = *n'* ∧ *x* = *x'*}›]) *auto*
ultimately show *?thesis*
by *simp*
qed

definition *arena-is-packed* :: ‹*arena* ⇒ *nat clauses-l* ⇒ *bool*› **where**
‹*arena-is-packed arena N* ⟷ *length arena* = (∑ *C* ∈# *dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*))›

lemma *arena-is-packed-empty*[*simp*]: ‹*arena-is-packed* [] *fmempty*›
by (*auto simp*: *arena-is-packed-def*)

lemma *sum-mset-cong*:
‹(⋀*A*. *A* ∈# *M* ⟹ *f A* = *g A*) ⟹ (∑ *A* ∈# *M*. *f A*) = (∑ *A* ∈# *M*. *g A*)›
by (*induction M*) *auto*
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* ∉# *dom-m N*›
shows ‹*arena-is-packed* (*arena* @ *C*) (*fmupd a C' N*)›
proof −
show *?thesis*
using *assms*(*1*) **by** (*auto simp*: *arena-is-packed-def*
intro!: *sum-mset-cong*)
qed

lemma *arena-is-packed-append-valid*:
assumes
in-dom: ‹*fst C* ∈# *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* ∝ (*fst C*))›
shows ‹*arena-is-packed*
(*x1d* @

$Misc.slice\ (fst\ C - n)$
$(fst\ C + arena\text{-}length\ x1c\ (fst\ C))\ x1c)$
$(fmupd\ (length\ x1d + n)\ (the\ (fmlookup\ x1a\ (fst\ C)))\ x2a)$›

proof −
have [*simp*]: ‹$length\ x1d + n \notin\#\ dom\text{-}m\ x2a$›
using *valid* **by** (*auto dest*: *arena-lifting*(*2*) *valid-arena-in-vdom-le-arena*
simp: *arena-is-valid-clause-vdom-def header-size-def*)
have [*simp*]: ‹$arena\text{-}length\ x1c\ (fst\ C) = length\ (x1a \propto (fst\ C))$› ‹$fst\ C \geq n$›
‹$fst\ C - n < length\ x1c$› ‹$fst\ C < length\ x1c$›
using *valid0 valid in-dom* **by** (*auto simp*: *arena-lifting n less-imp-diff-less*)
have [*simp*]: ‹*length*
(*Misc.slice* (*fst C* − *n*)
(*fst C* + *length* (*x1a* ∝ (*fst C*))) *x1c*) =
length (*x1a* ∝ *fst C*) + *header-size* (*x1a* ∝ *fst C*)›
using *valid in-dom arena-lifting*(*10*)[*OF valid0*]
by (*fastforce simp*: *slice-len-min-If min-def arena-lifting*(*4*) *simp flip*: *n*)
show *?thesis*
by (*rule arena-is-packed-append*[*OF packed*]) *auto*
qed

definition *move-is-packed* :: ‹*arena* ⇒ - ⇒ *arena* ⇒ - ⇒ *bool*› **where**
‹$move\text{-}is\text{-}packed\ arena_o\ N_o\ arena\ N \longleftrightarrow$
$((\sum C\in\#dom\text{-}m\ N_o.\ length\ (N_o \propto C) + header\text{-}size\ (N_o \propto C)) +$
$(\sum C\in\#dom\text{-}m\ N.\ length\ (N \propto C) + header\text{-}size\ (N \propto C)) \leq length\ arena_o)$›

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*) ← $WHILE_T$
(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*›
by (*auto simp*: *move-is-packed-def valid-arena-ge-length-clauses*)

lemma *move-is-packed-append*:
assumes
dom: ‹*C* ∈# *dom-m x1a*› **and**
E: ‹*length E* = *length* (*x1a* ∝ *C*) + *header-size* (*x1a* ∝ *C*)› ‹(*fst* E') = (*x1a* ∝ *C*)›
‹*n* = *header-size* (*x1a* ∝ *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 −
have [*simp*]: ‹(∑ *x*∈#*remove1-mset C*
(*dom-m*
x1a). *length*
(*fst* (*the* (*if x* = *C then None*
else fmlookup x1a x))) +
header-size
(*fst* (*the* (*if x* = *C then None*
else fmlookup x1a x)))) =
(∑ *x*∈#*remove1-mset C*
(*dom-m*
x1a). *length*
(*x1a* ∝ *x*) +
header-size
(*x1a* ∝ *x*))›
by (*rule sum-mset-cong*)
(*use distinct-mset-dom*[*of x1a*] **in** ‹*auto dest*!: *simp*: *distinct-mset-remove1-All*›)
have [*simp*]: ‹(*length x1d* + *header-size* (*x1a* ∝ *C*)) ∉# (*dom-m x2a*)›
using *valid arena-lifting*(*2*) **by** *blast*
have [*simp*]: ‹(∑ *x*∈#(*dom-m x2a*). *length*
(*fst* (*the* (*if length x1d* + *header-size* (*x1a* ∝ *C*) = *x*
then Some E'
else fmlookup x2a x))) +
header-size
(*fst* (*the* (*if length x1d* + *header-size* (*x1a* ∝ *C*) = *x*
then Some E'

else fmlookup x2a x)))) =
($\sum$ *x*∈#*dom-m x2a*. *length*
(*x2a* ∝ *x*) +
header-size
(*x2a* ∝ *x*))⟩
by (*rule sum-mset-cong*)
(*use distinct-mset-dom*[*of x2a*] **in** ⟨*auto dest*!: *simp*: *distinct-mset-remove1-All*⟩)
show *?thesis*
using *packed dom E*
by (*auto simp*: *move-is-packed-def split*: *if-splits dest*!: *multi-member-split*)
qed

definition *arena-header-size* :: ⟨*arena* ⇒ *nat* ⇒ *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* ⟹ *C* ∈# *dom-m N* ⟹ *arena-header-size arena C* = *header-size* (*N* ∝ *C*)⟩

by (*auto simp*: *arena-header-size-def header-size-def arena-lifting*)
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* ⊆ *vdom0*⟩ **and**
L: ⟨*atm-of A* ∈# $\mathcal{A}$⟩ **and**
L′-L: ⟨(*A′*, *A*) ∈ *nat-lit-lit-rel*⟩ **and**
W: ⟨(*W′* , *W*) ∈ ⟨*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* ⊆# *mset vdom*⟩ **and**
r: ⟨*length arena* = *r*⟩ **and**
le: ⟨∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W L*) ≤ *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*)
≤ ⇓ (*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena W*)
(*cdcl-GC-clauses-prog-copy-wl-entry N* (*W A*) *A N′*)⟩
(**is** ⟨- ≤ ⇓ (*?R*) -⟩)
proof −
have *A*: ⟨*A′* = *A*⟩ **and** *K*[*simp*]: ⟨*W′* ! *nat-of-lit A* = *W A*⟩
using *L′-L L W* **apply** *auto*
by (*cases A*) (*auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *A-le*: ⟨*nat-of-lit A* < *length W′*⟩
using *W L* **by** (*cases A*; *auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *length-slice*: ⟨*C* ∈# *dom-m x1a* ⟹ *valid-arena x1c x1a vdom′* ⟹
length
(*Misc.slice* (*C* − *header-size* (*x1a* ∝ *C*))
(*C* + *arena-length x1c C*) *x1c*) =
arena-length x1c C + *header-size* (*x1a* ∝ *C*)⟩ **for** *x1c x1a C vdom′*
using *arena-lifting*(*1*−*4*,*10*)[*of x1c x1a vdom′ C*]
by (*auto simp*: *header-size-def slice-len-min-If min-def split*: *if-splits*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def cdcl-GC-clauses-prog-copy-wl-entry-def prod.case A*
arena-header-size-def[*symmetric*]
apply (*refine-vcg ASSERT-leI WHILET-refine*[**where** *R* = ⟨*nat-rel* $\times_r$ *?R*⟩])
subgoal using *A-le* **by** (*auto simp*: *isasat-GC-entry-def*)

```
    subgoal using le L K by (cases A) (auto dest!: multi-member-split simp: ℒall-add-mset)
    subgoal using assms by (auto simp: isasat-GC-entry-def)
    subgoal using W L by auto
    subgoal by auto
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d
      using vdom L
      unfolding arena-is-valid-clause-vdom-def K isasat-GC-entry-def
      by (cases A)
        (force dest!: multi-member-split simp: vdom-m-def ℒall-add-mset)+
    subgoal
      using vdom L
      unfolding arena-is-valid-clause-vdom-def K isasat-GC-entry-def
      by (subst arena-dom-status-iff)
        (cases A ; auto dest!: multi-member-split simp: arena-lifting arena-dom-status-iff
            vdom-m-def ℒall-add-mset; fail)+
   subgoal
     unfolding arena-is-valid-clause-idx-def isasat-GC-entry-def
     by auto
   subgoal unfolding isasat-GC-entry-def move-is-packed-def arena-is-packed-def
       by (auto simp: valid-arena-header-size arena-lifting dest!: multi-member-split)
   subgoal using r by (auto simp: isasat-GC-entry-def)
    subgoal by (auto dest: valid-arena-header-size simp: arena-lifting dest!: valid-arena-vdom-subset
multi-member-split simp: arena-header-size-def isasat-GC-entry-def
    split: if-splits)
    subgoal by (auto simp: isasat-GC-entry-def dest!: size-mset-mono)
   subgoal
     by (force simp: isasat-GC-entry-def dest: arena-lifting(2))
   subgoal by (auto simp: arena-header-size-def)
   subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d D
     by (rule order-trans[OF fm-mv-clause-to-new-arena])
       (auto intro: valid-arena-extra-information-mark-to-delete'
         simp: arena-lifting remove-1-mset-id-iff-notin
            mark-garbage-pre-def isasat-GC-entry-def min-def
            valid-arena-header-size
       dest: in-vdom-m-fmdropD arena-lifting(2)
       intro!: arena-is-packed-append-valid subset-mset-trans-add-mset
        move-is-packed-append length-slice)
   subgoal
     by auto
   subgoal
     by auto
   done
 qed

definition isasat-GC-clauses-prog-single-wl
  :: ‹arena ⇒  (arena × - × -) ⇒ (nat watcher) list list ⇒ nat ⇒
       (arena × (arena × - × -) × (nat watcher) list list) nres›
where
‹isasat-GC-clauses-prog-single-wl = (λN0 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)) ← 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') ← 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'*) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*› **and**
X: ‹*X* = (*arena*, (*arena'*, *vdom*, *avdom*), *W*)› ‹*X'* = (*N*, *N'*, *W'*)› **and**
L: ‹*A* ∈# $\mathcal{A}$› **and**
st: ‹(*A*, *A'*) ∈ *Id*› **and** *st'*: ‹*narena* = (*arena'*, *vdom*, *avdom*)› **and**
ae: ‹*length arena0* = *length arena*› **and**
le-all: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length arena*›
shows ‹*isasat-GC-clauses-prog-single-wl arena narena W A*
≤ ⇓ (*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*)
(*cdcl-GC-clauses-prog-single-wl N W' A' N'*)›
(**is** ‹- ≤ ⇓ *?R* -›)
proof −
have *H*:
‹*valid-arena arena N vdom0*›
‹*valid-arena arena' N'* (*set vdom*)› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W' N* ⊆ *vdom0*› **and**
L: ‹*A* ∈# $\mathcal{A}$› **and**
eq: ‹*A'* = *A*› **and**
WW': ‹(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
vdom-dom: ‹*dom-m N'* = *mset vdom*› **and**
dist: ‹*distinct vdom*› **and**
packed: ‹*arena-is-packed arena' N'*› **and**
avdom: ‹*mset avdom* ⊆# *mset vdom*› **and**
packed2: ‹*move-is-packed arena N arena' N'*› **and**
incl: ‹*vdom-m* $\mathcal{A}$ *W' N* ⊆ *vdom0*›
using *assms X st* **by** (*auto simp*: *isasat-GC-refl-def*)

have *vdom2*: ‹*vdom-m* $\mathcal{A}$ *W' x1* ⊆ *vdom0* ⟹ *vdom-m* $\mathcal{A}$ (*W'*(*L* := [])) *x1* ⊆ *vdom0*› **for** *x1 L*
by (*force simp*: *vdom-m-def dest*!: *multi-member-split*)
have *vdom-m-upd*: ‹*x* ∈ *vdom-m* $\mathcal{A}$ (*W*(*Pos A* := [], *Neg A* := [])) *N* ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ *W N*› **for** *x W A N*
by (*auto simp*: *image-iff vdom-m-def dest*: *multi-member-split*)
have *vdom-m3*: ‹*x* ∈ *vdom-m* $\mathcal{A}$ *W a* ⟹ *dom-m a* ⊆# *dom-m b* ⟹ *dom-m b* ⊆# *dom-m c* ⟹*x* ∈ *vdom-m* $\mathcal{A}$ *W c*› **for** *a b c W x*
unfolding *vdom-m-def* **by** *auto*
have *W*: ‹(*W*[*2* ∗ *A* := [], *Suc* (*2* ∗ *A*) := []], *W'*(*Pos A* := [], *Neg A* := []))
∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **for** *A*
using *WW'* **unfolding** *map-fun-rel-def*
apply *clarify*
apply (*intro conjI*)
apply *auto*[]
apply (*drule multi-member-split*)
apply (*case-tac L*)
apply (*auto dest*!: *multi-member-split*)
done
have *le*: ‹*nat-of-lit* (*Pos A*) < *length W*› ‹*nat-of-lit* (*Neg A*) < *length W*›
using *WW' L* **by** (*auto dest*!: *multi-member-split simp*: *map-fun-rel-def* $\mathcal{L}_{all}$-*add-mset*)
have [*refine0*]: ‹*RETURN* (*Pos A*) ≤ ⇓ *Id* (*RES* {*Pos A*, *Neg A*})› **by** *auto*
have *vdom-upD*:‹ *x* ∈ *vdom-m* $\mathcal{A}$ (*W'*(*Pos A* := [], *Neg A* := [])) *xd* ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ (λ*a*. *if a* = *Pos A then* [] *else W' a*) *xd*›

for *W' a A x xd*
by (*auto simp*: *vdom-m-def*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-single-wl-def*
cdcl-GC-clauses-prog-single-wl-def eq st' isasat-GC-refl-def
apply (*refine-vcg*
isasat-GC-clauses-prog-copy-wl-entry[**where** r= ‹*length arena*› **and** $\mathcal{A} = \mathcal{A}$])
subgoal using *le* **by** *auto*
subgoal using *le* **by** *auto*
apply (*rule H*(*1*); *fail*)
apply (*rule H*(*2*); *fail*)
subgoal using *incl* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW'* **by** *auto*
subgoal using *vdom-dom* **by** *blast*
subgoal using *dist* **by** *blast*
subgoal using *packed* **by** *blast*
subgoal using *avdom* **by** *blast*
subgoal by *blast*
subgoal using *le-all* **by** *auto*
subgoal using *packed2* **by** *auto*
subgoal using *ae* **by** (*auto simp*: *isasat-GC-entry-def*)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*rule vdom2*; *auto*)
supply *isasat-GC-entry-def*[*simp*]
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *L* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *W ae le-all vdom* **by** (*auto simp*: *dest*!: *vdom-upD*)
done
qed

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*($\lambda\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}$*0*);
  (-, (*N*, *N'*, *WS*)) ← $WHILE_T$ $^{\textit{cdcl-GC-clauses-prog-wl-inv}\ \mathcal{A}\ N0}$
    (λ($\mathcal{B}$, -). $\mathcal{B}$ ≠ {#})
    (λ($\mathcal{B}$, (*N*, *N'*, *WS*)). *do* {
      ASSERT($\mathcal{B}$ ≠ {#});
      A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
      (*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'* ⟹ (*x*,*x'*)∈*R*
  assumes *IREF*: ⋀*x x'*. ⟦ (*x*,*x'*)∈*R*; *I' x'* ⟧ ⟹ *I x*
  assumes *COND-REF*: ⋀*x x'*. ⟦ (*x*,*x'*)∈*R*; *I x*; *I' x'* ⟧ ⟹ *b x* = *b' x'*
  assumes *STEP-REF*:
    ⋀*x x'*. ⟦ (*x*,*x'*)∈*R*; *b x*; *b' x'*; *I x*; *I' x'* ⟧ ⟹ *f x* ≤ ⇓*R* (*f' x'*)
  shows *WHILEIT I b f x* ≤⇓{(*x*, *x'*). (*x*, *x'*) ∈ *R* ∧ *I x* ∧ *I' x'* ∧ ¬*b' x'*} (*WHILEIT I' b' f' x'*)
  (**is** ⟨- ≤ ⇓*?R'* -⟩)
    apply (*subst* (*2*) *WHILEIT-add-post-condition*)
    apply (*refine-vcg WHILEIT-refine-genR*[**where** *R'*=*R* **and** *R* = *?R'*])
    subgoal by (*auto intro*: *assms*)[]
    subgoal by (*auto intro*: *assms*)[]
    subgoal using *COND-REF* **by** (*auto*)
    subgoal by (*auto intro*: *assms*)[]
    subgoal by (*auto intro*: *assms*)[]
    done

lemma *cdcl-GC-clauses-prog-wl-inv-cong-empty*:
  ⟨*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹
  cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N* ({#}, *x*) ⟹ *cdcl-GC-clauses-prog-wl-inv* $\mathcal{B}$ *N* ({#}, *x*)⟩
  by (*auto simp*: *cdcl-GC-clauses-prog-wl-inv-def*)

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 ⊆ *vdom0*⟩ **and**
    vmtf: ⟨((*ns*, *m*, *n*, *lst-As1*, *next-search1*), *to-remove1*) ∈ *vmtf* $\mathcal{A}$ *M*⟩ **and**
    nempty: ⟨$\mathcal{A}$ ≠ {#}⟩ **and**
    W-W': ⟨(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩ **and**
    bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩ **and** *old*: ⟨*old-arena* = []⟩ **and**
    le-all: ⟨∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length* $arena_o$⟩
 shows
    ⟨*isasat-GC-clauses-prog-wl2* (*ns*, *Some n*) ($arena_o$, (*old-arena*, [], []), *W*)
      ≤ ⇓ ({((($arena_o$', (*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* ∧
      cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ N_o ({#}, N_o', *N*, *W'*) ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom*
∧
      arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length* $arena_o$' = *length* $arena_o$})
      (*cdcl-GC-clauses-prog-wl2* N_o $\mathcal{A}$ *W'*)⟩

proof −
  define f **where**
    ‹$f\ A \equiv (\lambda(arena_o,\ arena,\ W).\ isasat\text{-}GC\text{-}clauses\text{-}prog\text{-}single\text{-}wl\ arena_o\ arena\ W\ A)$› **for** $A :: nat$
  let $?R$ = ‹$\{((\mathcal{A}',\ arena_o',\ (arena,\ vdom),\ W),\ (\mathcal{A}'',\ N_o',\ N,\ W')).\ \mathcal{A}' = \mathcal{A}'' \wedge$
      $((arena_o',\ (arena,\ vdom),\ W),\ (N_o',\ N,\ W')) \in isasat\text{-}GC\text{-}refl\ \mathcal{A}\ vdom0\ arena_o \wedge$
      $length\ arena_o' = length\ arena_o\}$›
  have H: ‹$(X,\ X') \in\ ?R \Longrightarrow X = (x1,\ x2) \Longrightarrow x2 = (x3,\ x4) \Longrightarrow x4 = (x5,\ x6) \Longrightarrow$
      $X' = (x1',\ x2') \Longrightarrow x2' = (x3',\ x4') \Longrightarrow x4' = (x5',\ x6') \Longrightarrow$
      $((x3,\ (fst\ x5,\ fst\ (snd\ x5),\ snd\ (snd\ x5)),\ x6),\ (x3',\ x5',\ x6')) \in isasat\text{-}GC\text{-}refl\ \mathcal{A}\ vdom0\ arena_o$›
    for $X\ X'\ A\ B\ x1\ x1'\ x2\ x2'\ x3\ x3'\ x4\ x4'\ x5\ x5'\ x6\ x6'\ x0\ x0'\ x\ x'$
    supply [[*show-types*]]
    by *auto*
  have *isasat-GC-clauses-prog-wl-alt-def*:
    ‹$isasat\text{-}GC\text{-}clauses\text{-}prog\text{-}wl2\ n\ x0 = iterate\text{-}over\text{-}VMTF\ f\ (\lambda x.\ length\ (fst\ x) = length\ (fst\ x0))\ n\ x0$›
    for $n\ x0$
    unfolding *f-def isasat-GC-clauses-prog-wl2-def iterate-over-VMTF-def* **by** (*cases n*) (*auto intro*!:
ext)
  show *?thesis*
    unfolding *isasat-GC-clauses-prog-wl-alt-def prod.case f-def*[*symmetric*] *old*
    apply (*rule order-trans*[*OF iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$[*OF vmtf nempty bounded*]])
    unfolding *Down-id-eq iterate-over-*$\mathcal{L}_{all}$*-def cdcl-GC-clauses-prog-wl2-def f-def*
    apply (*refine-vcg WHILEIT-refine-with-invariant-and-break*[**where** $R = ?R$]
        isasat-GC-clauses-prog-single-wl)
    subgoal by *fast*
    subgoal using *assms* **by** (*auto simp*: *valid-arena-empty isasat-GC-refl-def*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    apply (*rule H*; *assumption*; *fail*)
    apply (*rule refl*)+
    subgoal by (*auto simp add*: *cdcl-GC-clauses-prog-wl-inv-def*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal using *le-all* **by** (*auto simp*: *isasat-GC-refl-def split*: *prod.splits*)
    subgoal by (*auto simp*: *isasat-GC-refl-def*)
    subgoal by (*auto simp*: *isasat-GC-refl-def*
      dest: *cdcl-GC-clauses-prog-wl-inv-cong-empty*)
    done
qed

lemma *cdcl-GC-clauses-prog-wl-alt-def*:
  ‹$cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl = (\lambda(M,\ N0,\ D,\ NE,\ UE,\ Q,\ WS).\ do\ \{$
    $ASSERT(cdcl\text{-}GC\text{-}clauses\text{-}pre\text{-}wl\ (M,\ N0,\ D,\ NE,\ UE,\ Q,\ WS));$
    $(N,\ N',\ WS) \leftarrow cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl2\ N0\ (all\text{-}init\text{-}atms\ N0\ NE)\ WS;$
    $RETURN\ (M,\ N',\ D,\ NE,\ UE,\ Q,\ WS)$
    $\})$›
 proof −
   have [*refine0*]: ‹$(x1c,\ x1) \in Id \Longrightarrow RES\ (set\text{-}mset\ x1c)$
      $\leq \Downarrow Id\ \ (RES\ (set\text{-}mset\ x1))$› **for** $x1\ x1c$
     by *auto*
   have [*refine0*]: ‹$(xa,\ x') \in Id \Longrightarrow$
       $x2a = (x1b,\ x2b) \Longrightarrow$
       $x2 = (x1a,\ x2a) \Longrightarrow$
       $x' = (x1,\ x2) \Longrightarrow$

$x2d = (x1e, x2e) \Longrightarrow$
$x2c = (x1d, x2d) \Longrightarrow$
$xa = (x1c, x2c) \Longrightarrow$
$(A, Aa) \in Id \Longrightarrow$
cdcl-GC-clauses-prog-single-wl x1d x2e A x1e
$\leq \Downarrow Id$
(*cdcl-GC-clauses-prog-single-wl x1a x2b Aa x1b*)›
for $\mathcal{A}$ *x xa x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e A aaa Aa*
by *auto*
show *?thesis*
unfolding *cdcl-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl2-def*
while.imonad3

apply (*intro ext*)
apply (*clarsimp simp add*: *while.imonad3*)
apply (*subst order-class.eq-iff*[*of* ‹(- :: - *nres*)›])
apply (*intro conjI*)
subgoal
by (*rewrite at* ‹- ≤ ¤› *Down-id-eq*[*symmetric*]) (*refine-rcg WHILEIT-refine*[**where** $R = Id$], *auto*)
subgoal
by (*rewrite at* ‹- ≤ ¤› *Down-id-eq*[*symmetric*]) (*refine-rcg WHILEIT-refine*[**where** $R = Id$], *auto*)
done
qed

definition *isasat-GC-clauses-prog-wl* :: ‹*twl-st-wl-heur* ⇒ *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*) ← *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*) ∈ *twl-st-heur-restart*› **and**
prop-inv: ‹*correct-watching″ x1*›
shows ‹∀ *x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
proof
fix *x2*
assume *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
have ‹*correct-watching″ x1*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-D-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2* **unfolding** *all-init-atms-def all-init-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching″.simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x′a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›

using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms* (*get-clauses-wl x1*) (*get-unit-init-clss-wl x1*))›
using *x2 xb-x'a* **unfolding** *all-init-atms-def all-init-lits-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)

have ‹*vdom-m* (*all-init-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def all-atms-def*[*symmetric*])
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur-restart-def simp flip*: *all-init-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*)›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

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−
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
∈ *?T* ⟹
valid-arena x1g x1a (*set x1z*)›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,

x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
$\in$ *?T* $\Longrightarrow$
isasat-input-bounded (*all-init-atms x1a x1c*)›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹$\bigwedge$*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab.*
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
$\in$ *?T* $\Longrightarrow$
vdom-m (*all-init-atms x1a x1c*) *x2e x1a* $\subseteq$ *set x1z*›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹$\bigwedge$*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab.*
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
$\in$ *?T* $\Longrightarrow$
all-init-atms x1a x1c $\neq$ {#}›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹$\bigwedge$*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab.*
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
$\in$ *?T* $\Longrightarrow$
((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *set* (*fst x2o*)) $\in$ *vmtf* (*all-init-atms x1a x1c*) *x1*›
‹$\bigwedge$*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab.*
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1u*, *x1v*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x2e*)
$\in$ *?T* $\Longrightarrow$ (*x1j*, *x2e*) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms x1a x1c*))›
unfolding *twl-st-heur-restart-def isa-vmtf-def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
by *auto*
have *H*: ‹*vdom-m* (*all-init-atms x1a x1c*) *x2ad x1ad* $\subseteq$ *set x2af*›
if
empty: ‹$\forall$ *A*$\in$#*all-init-atms x1a x1c*. *x2ad* (*Pos A*) = [] $\wedge$ *x2ad* (*Neg A*) = []› **and**
rem: ‹*GC-remap*** (*x1a*, *Map.empty*, *fmempty*) (*fmempty*, *m*, *x1ad*)› **and**
‹*dom-m x1ad* = *mset x2af*›
for *m* :: ‹*nat* $\Rightarrow$ *nat option*› **and** *y* :: ‹*nat literal multiset*› **and** *x* :: ‹*nat*› **and**
x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab x1ac x1ad x2ad x1ae x1ag x2af x2ag
proof −
have ‹*xa* $\in$# $\mathcal{L}_{all}$ (*all-init-atms x1a x1c*) $\Longrightarrow$ *x2ad xa* = []› **for** *xa*
using *empty* **by** (*cases xa*) (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show *?thesis*
using ‹*dom-m x1ad* = *mset x2af*›
by (*auto simp*: *vdom-m-def*)
qed
have *H'*: ‹*mset x2ag* $\subseteq$# *mset x1ah* $\Longrightarrow$ *x* $\in$ *set x2ag* $\Longrightarrow$ *x* $\in$ *set x1ah*› **for** *x2ag x1ah x*

    by (*auto dest*: *mset-eq-setD*)
  show *?thesis*
    supply [[*goals-limit=1*]]
    unfolding *isasat-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl-alt-def take-0*
    apply (*intro frefI nres-relI*)
    apply (*refine-vcg isasat-GC-clauses-prog-wl2*[**where** $\mathcal{A}$ = ‹*all-init-atms - -*›]; *remove-dummy-vars*)
    subgoal
      by (*clarsimp simp add*: *twl-st-heur-restart-def*
        cdcl-GC-clauses-prog-wl-inv-def H H'
        rtranclp-GC-remap-all-init-atms
        rtranclp-GC-remap-learned-clss-l)
    subgoal
      unfolding *cdcl-GC-clauses-pre-wl-def*
      by (*drule length-watched-le''*)
        (*clarsimp-all simp add*: *twl-st-heur-restart-def*
          cdcl-GC-clauses-prog-wl-inv-def H H'
          rtranclp-GC-remap-all-init-atms
         rtranclp-GC-remap-learned-clss-l)
    subgoal
      by (*clarsimp simp add*: *twl-st-heur-restart-def*
        cdcl-GC-clauses-prog-wl-inv-def H H'
        rtranclp-GC-remap-all-init-atms
        rtranclp-GC-remap-learned-clss-l)
    done
qed

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*))›
  by (*auto simp*: *pw-eq-iff refine-pw-simps uncurry-def Bex-def split*: *prod.splits*)

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})›
  supply [[*goals-limit=1*]]
  unfolding *cdcl-GC-clauses-wl-D-def*
  by (*fastforce intro*!: *ext simp*: *RES-RES-RETURN-RES2 cdcl-remap-st-def*

RES-RES7-RETURN-RES uncurry-def image-iff
RES-RETURN-RES-RES2 RES-RETURN-RES RES-RES2-RETURN-RES rewatch-spec-def
intro!: *bind-cong-nres*)

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*) ∈ *state-wl-l None*›
  ‹*correct-watching''* *T* ∧ *cdcl-GC-clauses-pre S* ∧
   set-mset (*dom-m* (*get-clauses-wl T*)) ⊆ *clauses-pointed-to*
      (*Neg* ‘ *set-mset* (*all-init-atms* (*get-clauses-wl T*) (*get-unit-init-clss-wl T*)) ∪
       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* ≤
      ⇓ {((*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*) ∧ *N''* = *N* ∧
          (∀ *L*∈#*all-init-lits* (*get-clauses-wl T*) (*get-unit-init-clss-wl T*). *WS' L* = []) ∧
          all-init-lits (*get-clauses-wl T*) (*get-unit-init-clss-wl T*) = *all-init-lits N NE'* ∧
          (∃ *m*. *GC-remap*** (*get-clauses-wl T*, *Map.empty*, *fmempty*)
              (*fmempty*, *m*, *N*))}
     (*SPEC*(λ(*N'*::(*nat*, *'a literal list* × *bool*) *fmap*, *m*).
         *GC-remap*** (*N0'*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *N'*) ∧
   0 ∉# *dom-m N'*))›
   using *cdcl-GC-clauses-prog-wl2*[*of* ‹*get-trail-wl T*› ‹*get-clauses-wl T*› ‹*get-conflict-wl T*›
    ‹*get-unit-init-clss-wl T*› ‹*get-unit-learned-clss-wl T*› ‹*literals-to-update-wl T*›
    ‹*get-watched-wl T*› *S*] *assms*
   by (*cases T*) *auto*

lemma *correct-watching''-clauses-pointed-to*:
  assumes
    xa-xb: ‹(*xa*, *xb*) ∈ *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*))
⊆ *clauses-pointed-to*
(*Neg* ‘
set-mset
(*all-init-atms* (*get-clauses-wl xa*) (*get-unit-init-clss-wl xa*)) ∪
Pos ‘
set-mset
(*all-init-atms* (*get-clauses-wl xa*) (*get-unit-init-clss-wl xa*)))
(*get-watched-wl xa*)⟩
(**is** ⟨- ⊆ *?A*⟩)
proof
let *?𝒜* = ⟨*all-init-atms* (*get-clauses-wl xa*) (*get-unit-init-clss-wl xa*)⟩
fix *C*
assume *C*: ⟨*C* ∈# *dom-m* (*get-clauses-wl xa*)⟩
obtain *M N D NE UE Q W* **where**
xa: ⟨*xa* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)⟩
by (*cases xa*)
obtain *x* **where**
xb-x: ⟨(*xb*, *x*) ∈ *twl-st-l None*⟩ **and**
⟨*twl-list-invs xb*⟩ **and**
struct-invs: ⟨*twl-struct-invs x*⟩ **and**
⟨*get-conflict-l xb* = *None*⟩ **and**
⟨*clauses-to-update-l xb* = {#}⟩ **and**
⟨*count-decided* (*get-trail-l xb*) = *0*⟩ **and**
⟨∀ *L*∈*set* (*get-trail-l xb*). *mark-of L* = *0*⟩
using *pre* **unfolding** *cdcl-GC-clauses-pre-def* **by** *fast*
have ⟨*twl-st-inv x*⟩
using *xb-x C struct-invs*
by (*auto simp*: *twl-struct-invs-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*)
then have *le0*: ⟨*get-clauses-wl xa* ∝ *C* ≠ []⟩
using *xb-x C xa-xb*
by (*cases x*; *cases* ⟨*irred N C*⟩)
(*auto simp*: *twl-struct-invs-def twl-st-inv.simps*
twl-st-l-def state-wl-l-def xa ran-m-def conj-disj-distribR
Collect-disj-eq Collect-conv-if
dest!: *multi-member-split*)
then have *le*: ⟨*N* ∝ *C* ! *0* ∈ *set* (*watched-l* (*N* ∝ *C*))⟩
by (*cases* ⟨*N* ∝ *C*⟩) (*auto simp*: *xa*)
have *eq*: ⟨*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N NE*)) =
set-mset (*all-lits-of-mm* (*mset* ‘# *init-clss-lf N* + *NE*))⟩
by (*auto simp del*: *all-init-atms-def*[*symmetric*]
simp: *all-init-atms-def xa* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*[*symmetric*]
all-init-lits-def)
have *H*: ⟨*get-clauses-wl xa* ∝ *C* ! *0* ∈# *all-lits-of-mm* (*mset* ‘# *init-clss-lf* (*get-clauses-wl xa*) + *get-unit-init-clss-wl xa*)⟩
using *L C le0* **apply** −
by (*subst* (*asm*) *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*[*OF xa-xb xb-x struct-invs*])
(*cases* ⟨*N* ∝ *C*⟩; *auto simp*: *literals-are-*$\mathcal{L}_{in}$*-def all-lits-def ran-m-def eq*
all-lits-of-mm-add-mset is-$\mathcal{L}_{all}$*-def xa all-lits-of-m-add-mset*
dest!: *multi-member-split*)
moreover {
have ⟨{#*i* ∈# *fst* ‘# *mset* (*W* (*N* ∝ *C* ! *0*)). *i* ∈# *dom-m N*#} =
add-mset C {#*Ca* ∈# *remove1-mset C* (*dom-m N*). *N* ∝ *C* ! *0* ∈ *set* (*watched-l* (*N* ∝ *Ca*))#}⟩
using *corr H C le* **unfolding** *xa*

by (*auto simp*: *clauses-pointed-to-def correct-watching''.simps xa*
simp: *all-init-atms-def all-init-lits-def clause-to-update-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*)
from *arg-cong*[*OF this*, *of set-mset*] **have** ‹$C \in$ *fst* ‘ *set* (*W* ($N \propto C$! *0*))›
using *corr H C le* **unfolding** *xa*
by (*auto simp*: *clauses-pointed-to-def correct-watching''.simps xa*
simp: *all-init-atms-def all-init-lits-def clause-to-update-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*) **}**
ultimately show ‹$C \in$ *?A*›
by (*cases* ‹$N \propto C$! *0*›)
(*auto simp*: *clauses-pointed-to-def correct-watching''.simps xa*
simp: *all-init-atms-def all-init-lits-def clause-to-update-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*)
qed

abbreviation *isasat-GC-clauses-rel* **where**
‹*isasat-GC-clauses-rel y* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧
(∀ *L*∈#*all-init-lits* (*get-clauses-wl y*) (*get-unit-init-clss-wl y*). *get-watched-wl T L* = [])∧
all-init-lits-st y = *all-init-lits* (*get-clauses-wl y*) (*get-unit-init-clss-wl y*) ∧
get-trail-wl T = *get-trail-wl y* ∧
get-conflict-wl T = *get-conflict-wl y* ∧
get-unit-init-clss-wl T = *get-unit-init-clss-wl y* ∧
get-unit-learned-clss-wl T = *get-unit-learned-clss-wl y* ∧
(∃ *m*. *GC-remap*** (*get-clauses-wl y*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *get-clauses-wl T*)) ∧
arena-is-packed (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}›

lemma *ref-two-step''*: ‹$R \subseteq R' \Longrightarrow A \leq B \Longrightarrow \Downarrow R\ A \leq\ \Downarrow R'\ B$›
by (*simp add*: *weaken-⇓ ref-two-step'*)

lemma *isasat-GC-clauses-prog-wl-cdcl-remap-st*:
assumes
‹(*x*, *y*) ∈ *twl-st-heur-restart'''* *r*› **and**
‹*cdcl-GC-clauses-pre-wl-D y*›
shows ‹*isasat-GC-clauses-prog-wl x* ≤ ⇓ (*isasat-GC-clauses-rel y*) (*cdcl-remap-st y*)›
proof −
have *xy*: ‹(*x*, *y*) ∈ *twl-st-heur-restart*›
using *assms*(*1*) **by** *fast*
have *H*: ‹*isasat-GC-clauses-rel y* =
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}
O
{(*S*, *T*). *S* = *T* ∧ (∀ *L*∈#*all-init-lits-st y*. *get-watched-wl T L* = [])∧
all-init-lits-st y = *all-init-lits* (*get-clauses-wl y*) (*get-unit-init-clss-wl y*) ∧
get-trail-wl T = *get-trail-wl y* ∧
get-conflict-wl T = *get-conflict-wl y* ∧
get-unit-init-clss-wl T = *get-unit-init-clss-wl y* ∧
get-unit-learned-clss-wl T = *get-unit-learned-clss-wl y* ∧
(∃ *m*. *GC-remap*** (*get-clauses-wl y*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *get-clauses-wl T*))}›
by *blast*
show *?thesis*
using *assms* **apply** −
apply (*rule order-trans*[*OF isasat-GC-clauses-prog-wl*[*THEN fref-to-Down*]])
subgoal by *fast*
apply (*rule xy*)

unfolding *conc-fun-chain*[*symmetric*] *H*
apply (*rule ref-two-step'*)
unfolding *cdcl-GC-clauses-pre-wl-D-def cdcl-GC-clauses-pre-wl-def*
apply *normalize-goal*+
apply (*rule order-trans*[*OF cdcl-GC-clauses-prog-wl2-st*])
apply (*solves auto*)
subgoal for *xa xb* **by** (*simp add*: *correct-watching''-clauses-pointed-to*)
apply (*rule refl*)
subgoal by (*auto simp*: *cdcl-remap-st-def conc-fun-RES split*: *prod.splits*)
done
qed

fun *correct-watching'''* :: ‹- ⇒ 'v twl-st-wl ⇒ bool› **where**
‹*correct-watching'''* $\mathcal{A}$ (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) ⟷
(∀ *L* ∈# *all-lits-of-mm* $\mathcal{A}$.
distinct-watched (*W L*) ∧
(∀ (*i*, *K*, *b*)∈#*mset* (*W L*).
i ∈# *dom-m N* ∧ *K* ∈ *set* (*N* ∝ *i*) ∧ *K* ≠ *L* ∧
correctly-marked-as-binary N (*i*, *K*, *b*)) ∧
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* ≤ *length C*› **and**
i-notin[*simp*]: ‹*i* ∉# *dom-m aa*› **and**
dist[*iff*]: ‹*C* ! *0* ≠ *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 −
have [*iff*]: ‹*C* ! *Suc 0* ≠ *C* ! *0*›
using ‹*C* ! *0* ≠ *C* ! *Suc 0*› **by** *argo*
have [*iff*]: ‹*C* ! *Suc 0* ∈# *all-lits-of-m* (*mset C*)› ‹*C* ! *0* ∈# *all-lits-of-m* (*mset C*)›
‹*C* ! *Suc 0* ∈ *set C*› ‹ *C* ! *0* ∈ *set C*› ‹*C* ! *0* ∈ *set* (*watched-l C*)› ‹*C* ! *Suc 0* ∈ *set* (*watched-l C*)›
using *leC* **by** (*force intro*!: *in-clause-in-all-lits-of-m nth-mem simp*: *in-set-conv-iff*
intro: *exI*[*of - 0*] *exI*[*of - ‹Suc 0›*])+
have [*dest*!]: ‹⋀*L*. *L* ≠ *C* ! *0* ⟹ *L* ≠ *C* ! *Suc 0* ⟹ *L* ∈ *set* (*watched-l C*) ⟹ *False*›
by (*cases C*; *cases* ‹*tl C*›; *auto*)+
have *i*: ‹*i* ∉ *fst* ' *set* (*b L*)› **if** ‹*L*∈#*all-lits-of-mm* $\mathcal{A}$›**for** *L*
using *corr i-notin that* **unfolding** *correct-watching'''.simps*
by *force*
have [*iff*]: ‹(*i*,*c*, *d*) ∉ *set* (*b L*)› **if** ‹*L*∈#*all-lits-of-mm* $\mathcal{A}$› **for** *L c d*
using *i*[*of L*, *OF that*] **by** (*auto simp*: *image-iff*)
then show *?thesis*
using *corr*
by (*force simp*: *correct-watching'''.simps ran-m-mapsto-upd-notin*
all-lits-of-mm-add-mset all-lits-of-mm-union clause-to-update-mapsto-upd-notin correctly-marked-as-binary.simps
split: *if-splits*)
qed

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* $(N \propto x) \geq 2$› **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* $\leq$ *SPEC*(λW. *correct-watching‴* $\mathcal{A}$ (*M*, *N*, *C*, *NE*, *UE*, *Q*, *W*))›
proof −
define *I* **where**
‹$I \equiv \lambda(a :: nat\ list)\ (b :: nat\ list)\ W$.
correct-watching‴ $\mathcal{A}$ ((*M*, *fmrestrict-set* (*set a*) *N*, *C*, *NE*, *UE*, *Q*, *W*))›
have *I0*: ‹*set-mset* (*dom-m N*) $\subseteq$ *set x* $\wedge$ *distinct x* $\Longrightarrow$ *I* [] *x W*› **for** *x*
using *empty* **unfolding** *I-def* **by** (*auto simp*: *correct-watching‴.simps*
all-blits-are-in-problem-init.simps clause-to-update-def
all-lits-of-mm-union)
have *le*: ‹*length* (σ *L*) < *size* (*dom-m N*)›
if ‹*correct-watching‴* $\mathcal{A}$ (*M*, *fmrestrict-set* (*set l1*) *N*, *C*, *NE*, *UE*, *Q*, σ)› **and**
‹*set-mset* (*dom-m N*) $\subseteq$ *set x* $\wedge$ *distinct x*› **and**
‹*x* = *l1* @ *xa* # *l2*› ‹*xa* $\in\#$ *dom-m N*› ‹*L* $\in$ *set* ($N \propto xa$)›
for *L l1* σ *xa l2 x*
proof −
have *1*: ‹*card* (*set l1*) $\leq$ *length l1*›
by (*auto simp*: *card-length*)
have ‹*L* $\in\#$ *all-lits-of-mm* $\mathcal{A}$›
using *that incl in-clause-in-all-lits-of-m*[*of L* ‹*mset* ($N \propto xa$)›]
by (*auto simp*: *correct-watching‴.simps dom-m-fmrestrict-set′ ran-m-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset atm-of-all-lits-of-m
in-all-lits-of-mm-ain-atms-of-iff
dest!: *multi-member-split*)
then have ‹*distinct-watched* (σ *L*)› **and** ‹*fst* ‘ *set* (σ *L*) $\subseteq$ *set l1* $\cap$ *set-mset* (*dom-m N*)›
using *that incl*
by (*auto simp*: *correct-watching‴.simps dom-m-fmrestrict-set′ dest*!: *multi-member-split*)
then have ‹*length* (*map fst* (σ *L*)) $\leq$ *card* (*set l1* $\cap$ *set-mset* (*dom-m N*))›
using *1* **by** (*subst distinct-card*[*symmetric*])
(*auto simp*: *distinct-watched-alt-def intro*!: *card-mono intro*: *order-trans*)
also have ‹... < *card* (*set-mset* (*dom-m N*))›
using *that* **by** (*auto intro*!: *psubset-card-mono*)
also have ‹... = *size* (*dom-m N*)›
by (*simp add*: *distinct-mset-dom distinct-mset-size-eq-card*)
finally show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *rewatch-def*
apply (*refine-vcg*
nfoldli-rule[**where** *I* = ‹*I*›])
subgoal by (*rule I0*)
subgoal using *assms* **unfolding** *I-def* **by** *auto*
subgoal for *x xa l1 l2* σ **using** *H*[*of xa*] **unfolding** *I-def* **apply** −
by (*rule*, *subst* (*asm*)*nth-eq-iff-index-eq*)
linarith+
subgoal for *x xa l1 l2* σ **unfolding** *I-def* **by** (*rule le*) (*auto intro*!: *nth-mem*)
subgoal for *x xa l1 l2* σ **unfolding** *I-def* **by** (*drule le*[**where** *L* = ‹$N \propto xa\ !\ 1$›]) (*auto simp*: *I-def*
dest!: *le*)
subgoal for *x xa l1 l2* σ
unfolding *I-def*
by (*cases* ‹*the* (*fmlookup N xa*)›)
(*auto intro*!: *correct-watching‴-add-clause simp*: *dom-m-fmrestrict-set′*)
subgoal

unfolding *I-def*
by *auto*
subgoal by *auto*
subgoal unfolding *I-def*
by (*auto simp*: *fmlookup-restrict-set-id′*)
done
qed

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* ∈# *dom-m old* $\Longrightarrow$ *C* ∉# *dom-m old′* $\Longrightarrow$
m′ C ≠ *None* ∧
fmlookup new′ (*the* (*m′ C*)) = *fmlookup old C*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *a aa b ab ac ba*
apply (*cases* ‹*C* ∉# *dom-m a*›)
apply (*auto dest*: *GC-remap-ran-m-remap GC-remap-ran-m-no-rewrite-map*
GC-remap-ran-m-no-rewrite)
apply (*metis GC-remap-ran-m-no-rewrite-fmap GC-remap-ran-m-no-rewrite-map in-dom-m-lookup-iff*
option.sel)
using *GC-remap-ran-m-remap rtranclp-GC-remap-ran-m-no-rewrite* **by** *fastforce*
done

lemma *GC-remap-ran-m-exists-earlier*:
‹*GC-remap* (*old*, *m*, *new*) (*old′*, *m′*, *new′*) $\Longrightarrow$ *C* ∈# *dom-m new′* $\Longrightarrow$ *C* ∉# *dom-m new* $\Longrightarrow$
∃ *D*. *m′ D* = *Some C* ∧ *D* ∈# *dom-m old* ∧
fmlookup new′ C = *fmlookup old D*›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) *auto*

lemma *rtranclp-GC-remap-ran-m-exists-earlier*:
‹*GC-remap*** (*old*, *m*, *new*) (*old′*, *m′*, *new′*) $\Longrightarrow$ *C* ∈# *dom-m new′* $\Longrightarrow$ *C* ∉# *dom-m new* $\Longrightarrow$
∃ *D*. *m′ D* = *Some C* ∧ *D* ∈# *dom-m old* ∧
fmlookup new′ C = *fmlookup old D*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
apply (*auto dest*: *GC-remap-ran-m-exists-earlier*)
apply (*case-tac* ‹*C* ∈# *dom-m b*›)
apply (*auto elim*!: *GC-remapE split*: *if-splits*)
apply *blast*
using *rtranclp-GC-remap-ran-m-no-new-map rtranclp-GC-remap-ran-m-no-rewrite* **by** *fastforce*

lemma *rewatch-heur-st-correct-watching*:
assumes
pre: ‹*cdcl-GC-clauses-pre-wl-D y*› **and**
S-T: ‹(*S*, *T*) ∈ *isasat-GC-clauses-rel y*›
shows ‹*rewatch-heur-st S* ≤ ⇓ (*twl-st-heur-restart′′′* (*length* (*get-clauses-wl-heur S*)))
(*rewatch-spec T*)›
proof −
obtain *M N D NE UE Q W* **where**
T: ‹*T* = (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*)›
by (*cases T*) *auto*

obtain *M′ N′ D′ j W′ vm φ clvls cach lbd outl stats fast-ema slow-ema ccount*
vdom avdom lcount opts **where**
S: ‹*S* = (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,

vdom, *avdom*, *lcount*, *opts*)›
by (*cases S*) *auto*

have
valid: ‹*valid-arena* N' *N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
dom-m-vdom: ‹*set-mset* (*dom-m N*) ⊆ *set vdom*› **and**
W: ‹(W', *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms N NE*))› **and**
empty: ‹⋀*L*. *L* ∈# *all-init-lits-st y* ⟹ *W L* = []› **and**
NUE:‹*get-unit-init-clss-wl y* = *NE* ›
‹*get-unit-learned-clss-wl y* = *UE*›
‹*get-trail-wl y* = *M*›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def S T*)
obtain *m* **where**
m: ‹*GC-remap*** (*get-clauses-wl y*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *N*)›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def S T*)
obtain *x xa xb* **where**
y-x: ‹(*y*, *x*) ∈ *Id*› ‹*x* = *y*› **and**
lits-y: ‹*literals-are-*$\mathcal{L}_{in}{}'$ (*all-init-atms-st y*) *y*› **and**
x-xa: ‹(*x*, *xa*) ∈ *state-wl-l None*› **and**
‹*correct-watching*′′ *x*› **and**
xa-xb: ‹(*xa*, *xb*) ∈ *twl-st-l None*› **and**
‹*twl-list-invs xa*› **and**
struct-invs: ‹*twl-struct-invs xb*› **and**
‹*get-conflict-l xa* = *None*› **and**
‹*clauses-to-update-l xa* = {#}› **and**
‹*count-decided* (*get-trail-l xa*) = *0*› **and**
‹∀ *L*∈*set* (*get-trail-l xa*). *mark-of L* = *0*›
using *pre*
unfolding *cdcl-GC-clauses-pre-wl-D-def cdcl-GC-clauses-pre-wl-def*
cdcl-GC-clauses-pre-def
by *blast*
have [*iff*]:
‹*distinct-mset* (*mset* (*watched-l C*) + *mset* (*unwatched-l C*)) ⟷ *distinct C*› **for** *C*
unfolding *mset-append*[*symmetric*]
by *auto*

have ‹*twl-st-inv xb*›
using *xa-xb struct-invs*
by (*auto simp*: *twl-struct-invs-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*)
then have *A*:
‹⋀*C*. *C* ∈# *dom-m* (*get-clauses-wl x*) ⟹ *distinct* (*get-clauses-wl x* ∝ *C*) ∧ *2* ≤ *length* (*get-clauses-wl x* ∝ *C*)›
using *xa-xb x-xa*
by (*cases x*; *cases* ‹*irred* (*get-clauses-wl x*) *C*›)
(*auto simp*: *twl-struct-invs-def twl-st-inv.simps*
twl-st-l-def state-wl-l-def ran-m-def conj-disj-distribR
Collect-disj-eq Collect-conv-if
dest!: *multi-member-split*
split: *if-splits*)
have *struct-wf*:
‹*C* ∈# *dom-m N* ⟹ *distinct* (*N* ∝ *C*) ∧ *2* ≤ *length* (*N* ∝ *C*)› **for** *C*
using *rtranclp-GC-remap-ran-m-exists-earlier*[*OF m*, *of* ‹*C*›] *A y-x*
by (*auto simp*: *T dest*:)

have *eq-UnD*: ‹$A = A' \cup A'' \Longrightarrow A' \subseteq A$› **for** A A' A''
by *blast*

have *eq3*: ‹*all-init-lits* (*get-clauses-wl y*) *NE* = *all-init-lits N NE*›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *all-init-lits-def*)
moreover have ‹*all-lits-st y* = *all-lits-st T*›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*] *rtranclp-GC-remap-learned-clss-l-old-new*[*OF m*]
apply (*auto simp*: *all-init-lits-def T NUE all-lits-def*)
by (*metis NUE*(*1*) *NUE*(*2*) *all-clss-l-ran-m all-lits-def get-unit-clauses-wl-alt-def*)
ultimately have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-init-atms N NE*) (*mset ‘# ran-mf N*)›
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*OF x-xa xa-xb struct-invs*] *lits-y*
rtranclp-GC-remap-init-clss-l-old-new[*OF m*]
rtranclp-GC-remap-learned-clss-l-old-new[*OF m*]
apply (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def* $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*
y-x literals-are-$\mathcal{L}_{in}$*′-def literals-are-*$\mathcal{L}_{in}$*-def all-lits-def*[*of N*] *T*
get-unit-clauses-wl-alt-def all-lits-of-mm-union all-lits-def atm-of-eq-atm-of
is-$\mathcal{L}_{all}$*-def NUE all-init-atms-def all-init-lits-def all-atms-def conj-disj-distribR*
in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def atm-of-all-lits-of-mm
ex-disj-distrib Collect-disj-eq atms-of-def
dest!: *multi-member-split*[*of* - ‹*ran-m* -›]
split: *if-splits*
simp del: *all-init-atms-def*[*symmetric*] *all-atms-def*[*symmetric*])
apply (*auto dest*!: *eq-UnD dest*!: *split-list*)
done
have *eq*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N NE*)) = *set-mset* (*all-init-lits-st y*)›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *T all-init-lits-def NUE*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
then have *vd*: ‹*vdom-m* (*all-init-atms N NE*) *W N* $\subseteq$ *set-mset* (*dom-m N*)›
using *empty dom-m-vdom*
by (*auto simp*: *vdom-m-def*)
have ‹{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, {#}, {#}).
i ∈# *dom-m N*#} =
{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, {#}, {#}).
True#}› **for** *L*
by (*rule filter-mset-cong2*) (*auto simp*: *clause-to-update-def*)
then have *corr2*: ‹*correct-watching′′′*
({#*mset* (*fst x*). *x* ∈# *init-clss-l* (*get-clauses-wl y*)#} + *NE*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W′a*) $\Longrightarrow$
correct-watching′ (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W′a*)› **for** *W′a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *correct-watching′′′.simps correct-watching′.simps*)
have *eq2*: ‹*all-init-lits* (*get-clauses-wl y*) *NE* = *all-init-lits N NE*›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *T all-init-lits-def NUE*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
have ‹*i* ∈# *dom-m N* $\Longrightarrow$ *set* (*N* ∝ *i*) $\subseteq$ *set-mset* (*all-init-lits N NE*)› **for** *i*
using *lits* **by** (*auto dest*!: *multi-member-split split-list simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def ran-m-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
then have *blit2*: ‹*correct-watching′′′*
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W′a*) $\Longrightarrow$
blits-in-$\mathcal{L}_{in}$*′* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W′a*)› **for** *W′a*

using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
unfolding *correct-watching'''.simps blits-in-$\mathcal{L}_{in}$'-def*
$\mathcal{L}_{all}$-all-init-atms-all-init-lits all-init-lits-def[*symmetric*]
by (*fastforce simp*: *correct-watching'''.simps blits-in-$\mathcal{L}_{in}$'-def*
simp: *eq $\mathcal{L}_{all}$-all-init-atms-all-init-lits eq2*
dest!: *multi-member-split*[*of* - ‹*all-init-lits N NE*›]
dest: *mset-eq-setD*)
have ‹*correct-watching'''*
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W'a*) ⟹
vdom-m (*all-init-atms N NE*) *W'a N* ⊆ *set-mset* (*dom-m N*)› **for** *W'a*
unfolding *correct-watching'''.simps blits-in-$\mathcal{L}_{in}$'-def*
$\mathcal{L}_{all}$-all-init-atms-all-init-lits all-init-lits-def[*symmetric*]
using *eq eq3*
by (*force simp*: *correct-watching'''.simps vdom-m-def NUE*)
then have *st*: ‹(*x*, *W'a*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms N NE*)) ⟹
correct-watching'''
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W'a*) ⟹
((*M'*, *N'*, *D'*, *j*, *x*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*,
slow-ema, *ccount*, *vdom*, *avdom*, *lcount*, *opts*),
M, *N*, *get-conflict-wl y*, *NE*, *UE*, *Q*, *W'a*)
∈ *twl-st-heur-restart*› **for** *W'a m x*
using *S-T dom-m-vdom*
by (*auto simp*: *S T twl-st-heur-restart-def y-x NUE*)

show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *rewatch-heur-st-def T S*
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal by (*auto dest*!: *valid-arena-vdom-subset simp*: *twl-st-heur-restart-def*)
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist dom-m-vdom W lits*])
apply (*solves simp*)
apply (*rule vd*)
apply (*rule order-trans*[*OF ref-two-step'*])
apply (*rule rewatch-correctness*[**where** *M=M* **and** *N=N* **and** *NE=NE* **and** *UE=UE* **and** *C=D* **and** *Q=Q*])
apply (*rule empty*[*unfolded all-init-lits-def*]; *assumption*)
apply (*rule struct-wf*; *assumption*)
subgoal using *lits eq2* **by** (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-mm-def all-init-atms-def all-init-lits-def*
$\mathcal{L}_{all}$-atm-of-all-lits-of-mm
simp del: *all-init-atms-def*[*symmetric*])
apply (*subst conc-fun-RES*)
apply (*rule order.refl*)
apply (*fastforce simp*: *rewatch-spec-def RETURN-RES-refine-iff NUE*
intro: *corr2 blit2 st*)
done
qed

lemma *GC-remap-dom-m-subset*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›

by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m-subset*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m old'* $\subseteq$# *dom-m old*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*] **by** *auto*
done

lemma *GC-remap-mapping-unchanged*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *C* $\in$ *dom m* $\Longrightarrow$ *m' C* = *m C*›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) *auto*

lemma *rtranclp-GC-remap-mapping-unchanged*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *C* $\in$ *dom m* $\Longrightarrow$ *m' C* = *m C*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-unchanged*[*of old1 m1 new1 old2 m2 new2*, *of C*]
by (*auto dest*: *GC-remap-mapping-unchanged simp*: *dom-def intro*!: *image-mset-cong2*)
done

lemma *GC-remap-mapping-dom-extended*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom m'* = *dom m* $\cup$ *set-mset* (*dom-m old* $-$ *dom-m old'*)›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-mapping-dom-extended*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom m'* = *dom m* $\cup$ *set-mset* (*dom-m old* $-$ *dom-m old'*)›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-dom-extended*[*of old1 m1 new1 old2 m2 new2*]
GC-remap-dom-m-subset[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-dom-m-subset[*of old m new old1 m1 new1*]
by (*auto dest*: *GC-remap-mapping-dom-extended simp*: *dom-def mset-subset-eq-exists-conv*)
done

lemma *GC-remap-dom-m*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* $-$ *dom-m old'*)›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* $-$ *dom-m old'*)›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-dom-m*[*of old1 m1 new1 old2 m2 new2*] *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-dom-m-subset[*of old m new old1 m1 new1*]
GC-remap-mapping-unchanged[*of old1 m1 new1 old2 m2 new2*]

rtranclp-GC-remap-mapping-dom-extended[*of old m new old1 m1 new1*]
by (*auto dest*: *simp*: *mset-subset-eq-exists-conv intro*!: *image-mset-cong2*)
done

lemma *isasat-GC-clauses-rel-packed-le*:
assumes
xy: ‹(x, y) ∈ *twl-st-heur-restart'''* *r*› **and**
ST: ‹(S, T) ∈ *isasat-GC-clauses-rel y*›
shows ‹*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹∀ *C* ∈ *set* (*get-vdom S*). *C* < *length* (*get-clauses-wl-heur x*)›
proof −
obtain *m* **where**
‹(S, T) ∈ *twl-st-heur-restart*› **and**
‹∀ *L*∈#*all-init-lits-st y*. *get-watched-wl T L* = []› **and**
‹*get-trail-wl T* = *get-trail-wl y*› **and**
‹*get-conflict-wl T* = *get-conflict-wl y*› **and**
‹*get-unit-init-clss-wl T* = *get-unit-init-clss-wl y*› **and**
‹*get-unit-learned-clss-wl T* = *get-unit-learned-clss-wl y*› **and**
remap: ‹*GC-remap*** (*get-clauses-wl y*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *get-clauses-wl T*)› **and**
packed: ‹*arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)›
using *ST* **by** *auto*
have ‹*valid-arena* (*get-clauses-wl-heur x*) (*get-clauses-wl y*) (*set* (*get-vdom x*))›
using *xy* **unfolding** *twl-st-heur-restart-def* **by** (*cases x*; *cases y*) *auto*
from *valid-arena-ge-length-clauses*[*OF this*]
have ‹($\sum$ *C*∈#*dom-m* (*get-clauses-wl y*). *length* (*get-clauses-wl y* ∝ *C*) +
header-size (*get-clauses-wl y* ∝ *C*)) ≤ *length* (*get-clauses-wl-heur x*)›
(**is** ‹*?A* ≤ -›) **.**
moreover have ‹*?A* = ($\sum$ *C*∈#*dom-m* (*get-clauses-wl T*). *length* (*get-clauses-wl T* ∝ *C*) +
header-size (*get-clauses-wl T* ∝ *C*))›
using *rtranclp-GC-remap-ran-m-remap*[*OF remap*]
by (*auto simp*: *rtranclp-GC-remap-dom-m*[*OF remap*] *intro*!: *sum-mset-cong*)
ultimately show *le*: ‹*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur x*)›
using *packed* **unfolding** *arena-is-packed-def* **by** *simp*

have ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
using *ST* **unfolding** *twl-st-heur-restart-def* **by** (*cases S*; *cases T*) *auto*
then show ‹∀ *C* ∈ *set* (*get-vdom S*). *C* < *length* (*get-clauses-wl-heur x*)›
using *le*
by (*auto dest*: *valid-arena-in-vdom-le-arena*)
qed

lemma *isasat-GC-clauses-wl-D*:
‹(*isasat-GC-clauses-wl-D*, *cdcl-GC-clauses-wl-D*)
∈ *twl-st-heur-restart'''* *r* $\rightarrow_f$ ⟨*twl-st-heur-restart''''* *r*⟩*nres-rel*›
unfolding *isasat-GC-clauses-wl-D-def cdcl-GC-clauses-wl-D-alt-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg isasat-GC-clauses-prog-wl-cdcl-remap-st*[**where** *r*=*r*]
rewatch-heur-st-correct-watching)
subgoal unfolding *isasat-GC-clauses-pre-wl-D-def* **by** *blast*
subgoal by *fast*
subgoal by (*rule isasat-GC-clauses-rel-packed-le*)
subgoal by (*rule isasat-GC-clauses-rel-packed-le*(*2*))
apply *assumption+*
subgoal by (*auto*)
subgoal by (*auto*)

done

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* ⟹
    $(S,\ T) \in$ *twl-st-heur'''* *r* ⟷ $(S,\ T) \in$ *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* ⟹
    $(S,\ T) \in$ *twl-st-heur* ⟷ $(S,\ T) \in$ *twl-st-heur-restart*⟩ (**is** ⟨- ⟹ - *?B*⟩)
proof −
 note *cong* = *trail-pol-cong*
   option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong phase-saving-cong*
   cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
   isasat-input-bounded-cong

 show ⟨*cdcl-twl-full-restart-wl-GC-prog-pre T* ⟹ *?A*⟩
  supply [[*goals-limit=1*]]
  unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def cdcl-twl-full-restart-l-GC-prog-pre-def*
  apply *normalize-goal*+
  apply (*rule iffI*)
  subgoal for *U V*
   using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
    cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩]
 vdom-m-cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
   apply −
   apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
   apply (*cases S*; *cases T*)
   by (*simp add*: *twl-st-heur-def twl-st-heur-restart-ana-def*
    twl-st-heur-restart-def del: *isasat-input-nempty-def*)
  subgoal for *U V*
   using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
    cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩]
 vdom-m-cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
   apply −
   by (*cases S*; *cases T*)
    (*simp add*: *twl-st-heur-def twl-st-heur-restart-ana-def*

twl-st-heur-restart-def del: *isasat-input-nempty-def*)
done
show ⟨*cdcl-twl-full-restart-wl-D-GC-prog-post S0 T* ⟹ *?B*⟩
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-D-GC-prog-post-def*
cdcl-twl-full-restart-wl-GC-prog-post-def
cdcl-twl-full-restart-l-GC-prog-pre-def
apply *normalize-goal+*
subgoal for *S0′ T′ S0′′ U S0′′′*
apply (*rule iffI*)
subgoal
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U*]
cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩]
vdom-m-cong[*of* ⟨*all-atms-st T*⟩ ⟨*all-init-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
cdcl-twl-restart-l-invs[*of S0′′ S0′′′ U*]
apply −
apply (*clarsimp simp del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T′*)
apply (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
using *isa-vmtf-cong option-lookup-clause-rel-cong trail-pol-cong* **by** *presburger*
subgoal
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U*]
cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩]
vdom-m-cong[*of* ⟨*all-init-atms-st T*⟩ ⟨*all-atms-st T*⟩ ⟨*get-watched-wl T*⟩ ⟨*get-clauses-wl T*⟩]
cdcl-twl-restart-l-invs[*of S0′′ S0′′′ U*]
apply −
apply (*cases S*; *cases T*)
by (*clarsimp simp add*: *twl-st-heur-def twl-st-heur-restart-def*
simp del: *isasat-input-nempty-def*)
done
done

qed

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 $\to_f$ ⟨*twl-st-heur′′′′ r*⟩*nres-rel*⟩
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-def*
cdcl-twl-full-restart-wl-D-GC-prog-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg cdcl-twl-local-restart-wl-spec0*
remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D[**where** *r=r*, *THEN fref-to-Down*]
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl2-D[**where** *r=r*, *THEN fref-to-Down*]
isasat-GC-clauses-wl-D[**where** *r=r*, *THEN fref-to-Down*])
apply (*subst* (*asm*) *cdcl-twl-full-restart-wl-GC-prog-pre-heur*, *assumption*)
apply *assumption*
subgoal
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def*
cdcl-twl-full-restart-l-GC-prog-pre-def
by *normalize-goal+ auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply *assumption*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)

subgoal for *x y*
by (*blast dest*: *cdcl-twl-full-restart-wl-D-GC-prog-post-heur*)
done

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* $\to_f$ ⟨*bool-rel*⟩*nres-rel*›
unfolding *restart-required-heur-def restart-required-wl-def uncurry-def Let-def*
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def get-learned-clss-wl-def*)

lemma *restart-required-heur-restart-required-wl0*:
‹(*uncurry restart-required-heur*, *uncurry restart-required-wl*) ∈
twl-st-heur''' *r* $\times_f$ *nat-rel* $\to_f$ ⟨*bool-rel*⟩*nres-rel*›
unfolding *restart-required-heur-def restart-required-wl-def uncurry-def Let-def*
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def get-learned-clss-wl-def*)

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* $\to_f$ ⟨*twl-st-heur*'''' *r* $\times_f$ *nat-rel*⟩*nres-rel*›
proof −
have [*refine0*]: ‹*GC-required-heur S n* ≤ *SPEC* (λ-. *True*)› **for** *S n*
by (*auto simp*: *GC-required-heur-def*)
show *?thesis*
unfolding *restart-prog-wl-D-heur-def restart-prog-wl-D-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
restart-required-heur-restart-required-wl0[**where** *r*=*r*, *THEN fref-to-Down-curry*]
cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog[**where** *r*=*r*, *THEN fref-to-Down*]
cdcl-twl-full-restart-wl-D-GC-heur-prog[**where** *r*=*r*, *THEN fref-to-Down*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

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*›
apply (*intro frefI nres-relI*)
apply (*rule-tac r2 =* ‹*length*(*get-clauses-wl-heur* (*fst* (*fst x*)))› **and** *x'1* = ‹*y*› **in**
order-trans[*OF restart-prog-wl-D-heur-restart-prog-wl-D*[*THEN fref-to-Down*]])
apply *fast*
apply (*auto intro*!: *conc-fun-R-mono*)
done

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*)›
by (*auto simp*: *isasat-trail-nth-st-def intro*!: *ext*)

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*)›
by (*auto simp*: *get-the-propagation-reason-pol-st-def intro*!: *ext*)

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*)›
by (*auto simp*: *isasat-length-trail-st-def intro*!: *ext*)

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*)›
by (*auto simp*: *get-pos-of-level-in-trail-imp-st-def intro*!: *ext*)

definition *rewatch-heur-st-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*rewatch-heur-st-pre S* ⟷ (∀ *i* < *length* (*get-vdom S*). *get-vdom S* ! *i* ≤ *uint64-max*)›

lemma *isasat-GC-clauses-wl-D-rewatch-pre*:
assumes
‹*length* (*get-clauses-wl-heur x*) ≤ *uint64-max*› **and**
‹*length* (*get-clauses-wl-heur xc*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹∀ *i* ∈ *set* (*get-vdom xc*). *i* ≤ *length* (*get-clauses-wl-heur x*)›
shows ‹*rewatch-heur-st-pre xc*›
using *assms*

unfolding *rewatch-heur-st-pre-def all-set-conv-all-nth*
by *auto*

lemma *li-uint32-maxdiv2-le-unit32-max*: ‹$a \leq$ *uint32-max div 2* + *1* $\Longrightarrow a \leq$ *uint32-max*›
by (*auto simp*: *uint32-max-def*)

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*]:
‹(*uncurry* (*return oo clause-score-ordering*), *uncurry* (*RETURN oo clause-score-ordering*)) ∈
(*uint32-nat-assn* $*a$ *uint32-nat-assn*)^k $*_a$ (*uint32-nat-assn* $*a$ *uint32-nat-assn*)^k $\rightarrow_a$ *bool-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *clause-score-ordering-def uint32-nat-rel-def br-def*
nat-of-uint32-less-iff nat-of-uint32-le-iff)

sepref-definition *get-slow-ema-heur-fast-code*
is ‹*RETURN o get-slow-ema-heur*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *get-slow-ema-heur-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-slow-ema-heur-slow-code*
is ‹*RETURN o get-slow-ema-heur*›
:: ‹*isasat-unbounded-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *get-slow-ema-heur-alt-def isasat-unbounded-assn-def*
by *sepref*

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 ‹*RETURN o get-fast-ema-heur*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *get-fast-ema-heur-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-fast-ema-heur-slow-code*
is ‹*RETURN o get-fast-ema-heur*›
:: ‹*isasat-unbounded-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *get-fast-ema-heur-alt-def isasat-unbounded-assn-def*
by *sepref*

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 ‹*RETURN o get-conflict-count-since-last-restart-heur*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint64-assn*›
unfolding *get-counflict-count-heur-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-conflict-count-since-last-restart-heur-slow-code*
is ⟨*RETURN o get-conflict-count-since-last-restart-heur*⟩
:: ⟨*isasat-unbounded-assn*^{k} $\rightarrow_a$ *uint64-assn*⟩
unfolding *get-counflict-count-heur-alt-def isasat-unbounded-assn-def*
by *sepref*

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-bounded-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*⟩
unfolding *get-learned-count-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-learned-count-slow-code*
is ⟨*RETURN o get-learned-count*⟩
:: ⟨*isasat-unbounded-assn*^{k} $\rightarrow_a$ *nat-assn*⟩
unfolding *get-learned-count-alt-def isasat-unbounded-assn-def*
by *sepref*

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-pol-assn*^{k} $*_a$ *vmtf-remove-conc*^{k} $\rightarrow_a$ *uint32-nat-assn*⟩
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-int-inv-def*
by *sepref*

sepref-definition *find-local-restart-target-level-fast-code*
is ⟨*uncurry find-local-restart-target-level-int*⟩
:: ⟨*trail-pol-fast-assn*^{k} $*_a$ *vmtf-remove-conc*^{k} $\rightarrow_a$ *uint32-nat-assn*⟩
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-int-inv-def*
length-uint32-nat-def
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *incr-restart-stat-def isasat-unbounded-assn-def PR-CONST-def*
by *sepref*

sepref-register *incr-restart-stat*

sepref-definition *incr-restart-stat-fast-code*

is ⟨*incr-restart-stat*⟩
:: ⟨*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *incr-restart-stat-def isasat-bounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *incr-lrestart-stat-def isasat-unbounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *incr-lrestart-stat-def isasat-bounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-st-alt-def isasat-unbounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-st-alt-def isasat-bounded-assn-def PR-CONST-def*
by *sepref*

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*⟩
supply [[*goals-limit=1*]]
unfolding *empty-Q-def isasat-unbounded-assn-def*
by *sepref*

sepref-definition *empty-Q-fast-code*

is ‹*empty-Q*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-Q-def isasat-bounded-assn-def*
by *sepref*

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*›
unfolding *cdcl-twl-local-restart-wl-D-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*›
unfolding *cdcl-twl-local-restart-wl-D-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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-assn*^{k} $\rightarrow_a$ *uint64-assn*›
by *sepref-to-hoare sep-auto*

definition *two-uint64* :: ‹*uint64*› **where**
‹*two-uint64* = *2*›

lemma *two-uint64*[*sepref-fr-rules*]:
‹(*uncurry0* (*return two-uint64*), *uncurry0* (*RETURN two-uint64*))
∈ *unit-assn*^{k} $\rightarrow_a$ *uint64-assn*›
by *sepref-to-hoare sep-auto*

sepref-register *upper-restart-bound-not-reached*
sepref-definition *upper-restart-bound-not-reached-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹*isasat-unbounded-assn*^{k} $\rightarrow_a$ *bool-assn*›
unfolding *upper-restart-bound-not-reached-def PR-CONST-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *upper-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool-assn*›

unfolding *upper-restart-bound-not-reached-def PR-CONST-def isasat-bounded-assn-def*
apply (*rewrite at* ‹□ < -› *nat-of-uint64-conv-def*[*symmetric*])
supply [[*goals-limit* = *1*]]
by *sepref*

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*)›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
unfolding *lower-restart-bound-not-reached-def PR-CONST-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *lower-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o lower-restart-bound-not-reached*)›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
unfolding *lower-restart-bound-not-reached-def PR-CONST-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
apply (*rewrite at* ‹□ < -› *nat-of-uint64-conv-def*[*symmetric*])
by *sepref*

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*]$_a$
$arena\text{-}assn^k *_a nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn * a\ uint32\text{-}nat\text{-}assn$›
supply *uint32-max-uint32-nat-assn*[*sepref-fr-rules*]
unfolding *clause-score-extract-def insert-sort-inner-def valid-sort-clause-score-pre-at-def*
by *sepref*

declare *clause-score-extract-code.refine*[*sepref-fr-rules*]

sepref-definition *isa-get-clause-LBD-code2*
is ‹*uncurry isa-get-clause-LBD*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›
unfolding *isa-get-clause-LBD-def fast-minus-def*[*symmetric*] *LBD-SHIFT-hnr*[*sepref-fr-rules*]
by *sepref*

lemma *isa-get-clause-LBD-code*[*sepref-fr-rules*]:
‹(*uncurry isa-get-clause-LBD-code2*, *uncurry* (*RETURN* ∘∘ *get-clause-LBD*))
∈ [*uncurry get-clause-LBD-pre*]$_a$ $arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
using *isa-get-clause-LBD-code2.refine*[*FCOMP isa-get-clause-LBD-get-clause-LBD*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)
sepref-definition *isa-arena-act-code2*
is ‹*uncurry isa-arena-act*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}assn$›

unfolding *isa-arena-act-def ACTIVITY-SHIFT-hnr*[*sepref-fr-rules*] *fast-minus-def*[*symmetric*]
by *sepref*

lemma *isa-arena-act-code2*[*sepref-fr-rules*]:
⟨(*uncurry isa-arena-act-code2*, *uncurry* (*RETURN* ∘∘ *arena-act*))
∈ [*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *uint64-nat-assn*^{k} → *uint32-nat-assn*⟩
using *isa-arena-act-code2.refine*[*FCOMP isa-arena-act-arena-act*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

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-sort-clause-score-pre-at*]$_a$
arena-fast-assn^k $*_a$ *uint64-nat-assn*^{k} → *uint32-nat-assn* ∗*a* *uint32-nat-assn*⟩
supply *uint32-max-uint32-nat-assn*[*sepref-fr-rules*]
unfolding *clause-score-extract-def insert-sort-inner-def valid-sort-clause-score-pre-at-def*
by *sepref*

declare *clause-score-extract-fast-code.refine*[*sepref-fr-rules*]

sepref-definition (**in** −) *partition-main-clause-code*
is ⟨*uncurry3 partition-main-clause*⟩
:: ⟨[λ(((*arena*, *i*), *j*), *vdom*). *valid-sort-clause-score-pre arena vdom*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *vdom-assn*^{d} → *vdom-assn* ∗*a* *nat-assn*⟩
supply *insort-inner-clauses-by-score-invI*[*intro*]
partition-main-inv-def[*simp*]
unfolding *partition-main-clause-def partition-between-ref-def*
partition-main-def WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
by *sepref*

sepref-definition (**in** −) *partition-main-clause-fast-code*
is ⟨*uncurry3 partition-main-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*⟩
supply *insort-inner-clauses-by-score-invI*[*intro*] [[*goals-limit=1*]]
partition-main-inv-def[*simp*] *mset-eq-length*[*dest*]
unfolding *partition-main-clause-def partition-between-ref-def*
partition-main-def one-uint64-nat-def[*symmetric*]
WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
by *sepref*

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*⟩
:: ⟨[λ(((*arena*, *i*), *j*), *vdom*). *valid-sort-clause-score-pre arena vdom*]$_a$
arena-assn^k $*_a$ *nat-assn*^{k} $*_a$ *nat-assn*^{k} $*_a$ *vdom-assn*^{d} → *vdom-assn* ∗*a* *nat-assn*⟩
supply *insort-inner-clauses-by-score-invI*[*intro*] *valid-sort-clause-score-pre-swap*[

unfolded WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*], *intro*]
unfolding *partition-clause-def partition-between-ref-def*
choose-pivot3-def partition-main-clause-def[*symmetric*]
WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
by *sepref*

lemma *div2-hnr*[*sepref-fr-rules*]: ‹(*return o* (λ*n. n* >> *1*), *RETURN o div2*) $\in$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *div2-def uint64-nat-rel-def br-def nat-of-uint64-shiftl nat-shiftr-div2*)

sepref-definition (**in** −) *partition-clause-fast-code*
is ‹*uncurry3 partition-clause*›
:: ‹[λ(((*arena*, *i*), *j*), *vdom*). *length vdom* $\leq$ *uint64-max* $\wedge$ *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} $\rightarrow$ *vdom-fast-assn* $*a$ *uint64-nat-assn*›
supply *insort-inner-clauses-by-score-invI*[*intro*] *valid-sort-clause-score-pre-swap*[
unfolded WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*], *intro*] *mset-eq-length*[*dest*]
unfolding *partition-clause-def partition-between-ref-def div2-def*[*symmetric*]
choose-pivot3-def partition-main-clause-def[*symmetric*]
WB-More-Refinement-List.swap-def IICF-List.swap-def[*symmetric*]
by *sepref*

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} $\rightarrow$ *vdom-assn*›
supply *sort-clauses-by-score-invI*[*intro*]
unfolding *insert-sort-def*
quicksort-clauses-by-score-def
full-quicksort-ref-def
quicksort-ref-def
partition-clause-def[*symmetric*]
List.null-def
by *sepref*

lemma *minus-uint64-safe*:
‹(*uncurry* (*return oo safe-minus*), *uncurry* (*RETURN oo* (−))) $\in$ *uint64-nat-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare*
(*sep-auto simp*: *safe-minus-def uint64-nat-rel-def br-def nat-of-uint64-le-iff nat-of-uint64-notle-minus*)

sepref-definition (**in** −) *sort-clauses-by-score-fast-code*
is ‹*uncurry quicksort-clauses-by-score*›
:: ‹[λ(*arena*, *vdom*). *length vdom* $\leq$ *uint64-max* $\wedge$ *valid-sort-clause-score-pre arena vdom*]$_a$
arena-fast-assn^k $*_a$ *vdom-fast-assn*^{d} $\rightarrow$ *vdom-fast-assn*›
supply *sort-clauses-by-score-invI*[*intro*] [[*goals-limit=1*]] *mset-eq-length*[*dest*] *minus-uint64-safe*[*sepref-fr-rules*]
unfolding *insert-sort-def*
quicksort-clauses-by-score-def
full-quicksort-ref-def

quicksort-ref-def
partition-clause-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
List.null-def zero-uint64-nat-def[*symmetric*]
by *sepref*

lemma *arl64-take*[*sepref-fr-rules*]:
⟨(*uncurry* (*return oo arl64-take*), *uncurry* (*RETURN oo take*)) ∈
$[\lambda(n, xs).\ n \le \mathit{length}\ xs]_a$ *uint64-nat-assn*^{k} $*_a$ (*arl64-assn R*)^d → *arl64-assn R*⟩
by (*sepref-to-hoare*)
(*sep-auto simp*: *arl64-assn-def arl64-take-def is-array-list64-def hr-comp-def*
uint64-nat-rel-def br-def list-rel-def list-all2-conv-all-nth)

sepref-register *remove-deleted-clauses-from-avdom*
sepref-definition *remove-deleted-clauses-from-avdom-fast-code*
is ⟨*uncurry isa-remove-deleted-clauses-from-avdom*⟩
:: ⟨$[\lambda(N, \mathit{vdom}).\ \mathit{length}\ \mathit{vdom} \le \mathit{uint64\text{-}max}]_a$ *arena-fast-assn*^{k} $*_a$ *vdom-fast-assn*^{d} → *vdom-fast-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isa-remove-deleted-clauses-from-avdom-def swap-def*[*symmetric*]
WB-More-Refinement-List.swap-def zero-uint64-nat-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
by *sepref*

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*⟩
unfolding *isa-remove-deleted-clauses-from-avdom-def swap-def*[*symmetric*]
WB-More-Refinement-List.swap-def
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply *sort-clauses-by-score-invI*[*intro*] *sort-clauses-by-score-code.refine*[*sepref-fr-rules*]
unfolding *sort-vdom-heur-def isasat-unbounded-assn-def*
by *sepref*

sepref-definition *sort-vdom-heur-fast-code*
is ⟨*sort-vdom-heur*⟩
:: ⟨$[\lambda S.\ \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S) \le \mathit{uint64\text{-}max}]_a$*isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply *sort-clauses-by-score-invI*[*intro*] *sort-clauses-by-score-fast-code.refine*[*sepref-fr-rules*]
[[*goals-limit=1*]]
unfolding *sort-vdom-heur-def isasat-bounded-assn-def*
by *sepref*

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-unbounded-assn*^{k} $\rightarrow_a$ *bool-assn*⟩
unfolding *opts-restart-st-def isasat-unbounded-assn-def*

by *sepref*

sepref-definition *opts-restart-st-fast-code*
is ⟨*RETURN o opts-restart-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
unfolding *opts-restart-st-def isasat-bounded-assn-def*
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
unfolding *opts-reduction-st-def isasat-unbounded-assn-def*
by *sepref*

sepref-definition *opts-reduction-st-fast-code*
is ⟨*RETURN o opts-reduction-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
unfolding *opts-reduction-st-def isasat-bounded-assn-def*
by *sepref*

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*))
∈ $unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$⟩
by *sepref-to-hoare sep-auto*

lemma *max-restart-decision-lvl-code-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return max-restart-decision-lvl-code*), *uncurry0* (*RETURN max-restart-decision-lvl*)) ∈
$unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
by *sepref-to-hoare* (*sep-auto simp*: *br-def uint32-nat-rel-def max-restart-decision-lvl-def*
max-restart-decision-lvl-code-def)

lemma [*sepref-fr-rules*]:
⟨(*uncurry0* (*return GC-EVERY*), *uncurry0* (*RETURN GC-EVERY*)) ∈ $unit\text{-}assn^k \rightarrow_a uint64\text{-}assn$⟩
by *sepref-to-hoare* (*sep-auto simp*: *GC-EVERY-def*)

lemma (**in** −) *MINIMUM-DELETION-LBD-hnr*[*sepref-fr-rules*]:
⟨(*uncurry0* (*return 3*), *uncurry0* (*RETURN MINIMUM-DELETION-LBD*)) ∈ $unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
by *sepref-to-hoare* (*sep-auto simp*: *MINIMUM-DELETION-LBD-def uint32-nat-rel-def br-def*)

sepref-definition *restart-required-heur-fast-code*
is ⟨*uncurry restart-required-heur*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$⟩
supply [[*goals-limit=1*]]

shiftr-uint64 [*sepref-fr-rules*]
unfolding *restart-required-heur-def*
apply (*rewrite at* ‹*let* - = (□ > -) *in* -› *nat-of-uint64-conv-def* [*symmetric*])
by *sepref*

sepref-definition *restart-required-heur-slow-code*
is ‹*uncurry restart-required-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
supply [[*goals-limit=1*]]
shiftr-uint64 [*sepref-fr-rules*]
unfolding *restart-required-heur-def*
by *sepref*

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*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
unfolding *get-reduction-count-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-reductions-count-code*
is ‹*RETURN o get-reductions-count*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a uint64\text{-}assn$›
unfolding *get-reduction-count-alt-def isasat-unbounded-assn-def*
by *sepref*

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*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
supply [[*goals-limit=1*]]
op-eq-uint64 [*sepref-fr-rules*]
unfolding *GC-required-heur-def*
by *sepref*

sepref-definition *GC-required-heur-slow-code*
is ‹*uncurry GC-required-heur*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
supply [[*goals-limit=1*]]
op-eq-uint64 [*sepref-fr-rules*]
unfolding *GC-required-heur-def*
by *sepref*

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*›
  :: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *isasat-trail-nth-st-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-definition *isasat-trail-nth-st-slow-code*
  is ‹*uncurry isasat-trail-nth-st*›
  :: ‹$isasat\text{-}unbounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *isasat-trail-nth-st-alt-def isasat-unbounded-assn-def*
  by *sepref*

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*›
  :: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a option\text{-}assn\ uint64\text{-}nat\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *get-the-propagation-reason-pol-st-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-definition *get-the-propagation-reason-pol-st-slow-code*
  is ‹*uncurry get-the-propagation-reason-pol-st*›
  :: ‹$isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a option\text{-}assn\ nat\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *get-the-propagation-reason-pol-st-alt-def isasat-unbounded-assn-def*
  by *sepref*

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*›
  :: ‹$unat\text{-}lit\text{-}assn^k *_a (uint64\text{-}nat\text{-}assn)^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *isasat-replace-annot-in-trail-def isasat-bounded-assn-def*
    zero-uint64-nat-def[*symmetric*]
  by *sepref*

sepref-definition *isasat-replace-annot-in-trail-slow-code*
  is ‹*uncurry2 isasat-replace-annot-in-trail*›
  :: ‹$unat\text{-}lit\text{-}assn^k *_a (nat\text{-}assn)^k *_a isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
  supply [[*goals-limit=1*]]

unfolding *isasat-replace-annot-in-trail-def isasat-unbounded-assn-def*
by *sepref*

sepref-definition *mark-garbage-fast-code*
is ‹*uncurry mark-garbage*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a arl64\text{-}assn\ uint32\text{-}assn$›
supply *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *mark-garbage-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *mark-garbage-fast-hnr*[*sepref-fr-rules*]:
‹(*uncurry mark-garbage-fast-code*, *uncurry* (*RETURN oo extra-information-mark-to-delete*))
$\in [mark\text{-}garbage\text{-}pre]_a\ \ arena\text{-}fast\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$›
using *mark-garbage-fast-code.refine*[*FCOMP isa-mark-garbage*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

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*)) ∈ [λ((*l*,*i*),-). *i*<*length l*]$_a$ $(is\text{-}array\text{-}list64^d *_a nat\text{-}assn^k *_a id\text{-}assn^k) \rightarrow is\text{-}array\text{-}list64$
by (*sep-auto simp*: *arl64-set-nat-def is-array-list64-def*)
sepref-decl-impl *arl64-set-nat*: *arl64-set-hnr-aux* **.**

end

sepref-definition *mark-garbage-fast-code2*
is ‹*uncurry mark-garbage*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^d *_a nat\text{-}assn^k \rightarrow_a arl64\text{-}assn\ uint32\text{-}assn$›
unfolding *STATUS-SHIFT-def*
unfolding *mark-garbage-def fast-minus-def*[*symmetric*]
by *sepref*

lemma *mark-garbage-fast-hnr2*[*sepref-fr-rules*]:
‹(*uncurry mark-garbage-fast-code2*, *uncurry* (*RETURN oo extra-information-mark-to-delete*))
$\in [mark\text{-}garbage\text{-}pre]_a\ \ arena\text{-}fast\text{-}assn^d *_a nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$›
using *mark-garbage-fast-code2.refine*[*FCOMP isa-mark-garbage*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp*)

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*]$_a$
$uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
supply [[*goals-limit=1*]]

unfolding *mark-garbage-heur2-def isasat-bounded-assn-def*
zero-uint64-nat-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
by *sepref*

sepref-definition *mark-garbage-heur2-slow-code*
is ‹*uncurry mark-garbage-heur2*›
:: ‹$[\lambda(C, S)$. *mark-garbage-pre* (*get-clauses-wl-heur* S, C) $\wedge$ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur* S) $C]_a$
nat-assn^k $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow$ *isasat-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *mark-garbage-heur2-def isasat-unbounded-assn-def*
zero-uint64-nat-def[*symmetric*]
by *sepref*

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*›
:: ‹*uint32-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *uint32-nat-assn* $*a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def zero-uint64-nat-def*[*symmetric*]
one-uint32-nat-def[*symmetric*]
isasat-trail-nth-st-def[*symmetric*] *get-the-propagation-reason-pol-st-def*[*symmetric*]
by *sepref*

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-nat-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *uint32-nat-assn* $*a$ *isasat-unbounded-assn*›
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def*
isasat-trail-nth-st-def[*symmetric*] *get-the-propagation-reason-pol-st-def*[*symmetric*]
one-uint32-nat-def[*symmetric*]
by *sepref*

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-length-trail-pre o get-trail-wl-heur*]$_a$ *isasat-bounded-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-length-trail-st-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *isasat-length-trail-st-slow-code*
is ‹*RETURN o isasat-length-trail-st*›
:: ‹[*isa-length-trail-pre o get-trail-wl-heur*]$_a$ *isasat-unbounded-assn*^{k} $\rightarrow$ *uint32-nat-assn*›

supply [[*goals-limit=1*]]
unfolding *isasat-length-trail-st-alt-def isasat-unbounded-assn-def*
by *sepref*

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*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *get-pos-of-level-in-trail-imp-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-definition *get-pos-of-level-in-trail-imp-st-slow-code*
is ⟨*uncurry get-pos-of-level-in-trail-imp-st*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *get-pos-of-level-in-trail-imp-alt-def isasat-unbounded-assn-def*
by *sepref*

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*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-def zero-uint32-nat-def*[*symmetric*]
isasat-length-trail-st-def[*symmetric*] *get-pos-of-level-in-trail-imp-st-def*[*symmetric*]
by *sepref*

sepref-definition *remove-one-annot-true-clause-imp-wl-D-heur-slow-code*
is ⟨*remove-one-annot-true-clause-imp-wl-D-heur*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-def zero-uint32-nat-def*[*symmetric*]
isasat-length-trail-st-def[*symmetric*] *get-pos-of-level-in-trail-imp-st-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$[\lambda(((N, \text{-}), \text{-}), \text{-}).\ length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^d *_a watchlist\text{-}fast\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a$
$(arena\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn *a\ vdom\text{-}fast\text{-}assn)^d \rightarrow$

(*arena-fast-assn* ∗*a* (*arena-fast-assn* ∗*a* *vdom-fast-assn* ∗*a* *vdom-fast-assn*))›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn′*[*sepref-fr-rules*] *length-ll-def*[*simp*] *if-splits*[*split*]
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def nth-rll-def*[*symmetric*]
length-ll-def[*symmetric*] *zero-uint64-nat-def*[*symmetric*] *one-uint64-nat-def*[*symmetric*]
four-uint64-nat-def[*symmetric*] *five-uint64-nat-def*[*symmetric*]
apply (*rewrite at* ‹*let - = length-ll - ⌑ in -*› *uint64-of-uint32-conv-def*[*symmetric*])
by *sepref*

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-assn* ∗*a* (*arena-assn* ∗*a* *vdom-assn* ∗*a* *vdom-assn*))›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn′*[*sepref-fr-rules*] *length-ll-def*[*simp*]
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def nth-rll-def*[*symmetric*]
length-ll-def[*symmetric*]
apply (*rewrite at* ‹*let - = - +* (*If* (⌑ < -) - -) *in -*› *four-uint64-nat-def*[*symmetric*])
by *sepref*

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-take-ll L 0 W = W*[*L* := []]›
by (*auto simp*: *shorten-take-ll-def*)

lemma *length-shorten-take-ll*[*simp*]: ‹*length* (*shorten-take-ll a j W*) = *length W*›
by (*auto simp*: *shorten-take-ll-def*)

sepref-definition *isasat-GC-clauses-prog-single-wl-code*
is ‹*uncurry3 isasat-GC-clauses-prog-single-wl*›
:: ‹$[\lambda(((N, \text{-}), \text{-}), A).\ A \leq uint32\text{-}max\ div\ 2 \wedge length\ N \leq uint64\text{-}max]_a$
$arena\text{-}fast\text{-}assn^d *_a (arena\text{-}fast\text{-}assn * a\ vdom\text{-}fast\text{-}assn * a\ vdom\text{-}fast\text{-}assn)^d *_a watchlist\text{-}fast\text{-}assn^d$
$*_a uint32\text{-}nat\text{-}assn^k \rightarrow$
(*arena-fast-assn* ∗*a* (*arena-fast-assn* ∗*a* *vdom-fast-assn* ∗*a* *vdom-fast-assn*) ∗*a* *watchlist-fast-assn*)›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn′*[*sepref-fr-rules*]
unfolding *isasat-GC-clauses-prog-single-wl-def zero-uint64-nat-def*[*symmetric*]
shorten-take-ll-0[*symmetric*]
apply (*rewrite at* ‹*let - = shorten-take-ll ⌑ - - in -*› *nat-of-uint32-conv-def*[*symmetric*])
apply (*rewrite at* ‹*let - = shorten-take-ll ⌑ - - in RETURN -*› *nat-of-uint32-conv-def*[*symmetric*])
by *sepref*

sepref-definition *isasat-GC-clauses-prog-single-wl-slow-code*
is ‹*uncurry3 isasat-GC-clauses-prog-single-wl*›
:: ‹$[\lambda(((\text{-}, \text{-}), \text{-}), A).\ A \leq uint32\text{-}max\ div\ 2]_a$
$arena\text{-}assn^d *_a (arena\text{-}assn * a\ vdom\text{-}assn * a\ vdom\text{-}assn)^d *_a watchlist\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k \rightarrow$
(*arena-assn* ∗*a* (*arena-assn* ∗*a* *vdom-assn* ∗*a* *vdom-assn*) ∗*a* *watchlist-assn*)›
supply [[*goals-limit=1*]] *Pos-unat-lit-assn′*[*sepref-fr-rules*]
unfolding *isasat-GC-clauses-prog-single-wl-def*
shorten-take-ll-0[*symmetric*]
by *sepref*

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-GC-clauses-prog-wl2′ ns fst′* = (*isasat-GC-clauses-prog-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-max*$]_a$
(*array-assn vmtf-node-assn*)^k $*_a$ (*option-assn uint32-nat-assn*)^k $*_a$
(*arena-fast-assn* ∗*a* (*arena-fast-assn* ∗*a vdom-fast-assn* ∗*a vdom-fast-assn*) ∗*a watchlist-fast-assn*)^d
→
(*arena-fast-assn* ∗*a* (*arena-fast-assn* ∗*a vdom-fast-assn* ∗*a vdom-fast-assn*) ∗*a watchlist-fast-assn*)⟩
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl2-def isasat-GC-clauses-prog-wl2′-def*
by *sepref*

sepref-definition *isasat-GC-clauses-prog-wl2-slow-code*
is ⟨*uncurry2 isasat-GC-clauses-prog-wl2′*⟩
:: ⟨(*array-assn vmtf-node-assn*)^k $*_a$ (*option-assn uint32-nat-assn*)^k $*_a$
(*arena-assn* ∗*a* (*arena-assn* ∗*a vdom-assn* ∗*a vdom-assn*) ∗*a watchlist-assn*)^d $\rightarrow_a$
(*arena-assn* ∗*a* (*arena-assn* ∗*a vdom-assn* ∗*a vdom-assn*) ∗*a watchlist-assn*)⟩
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl2-def isasat-GC-clauses-prog-wl2′-def*
by *sepref*

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-clauses-wl-heur* S) $\leq$ *uint64-max*$]_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl-def isasat-bounded-assn-def*
isasat-GC-clauses-prog-wl2′-def[*symmetric*]
apply (*rewrite* **in** ⟨(-, -, -, -, -, *take* ⌑ -)⟩ *zero-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨(-, -, *take* ⌑ -)⟩ *zero-uint64-nat-def*[*symmetric*])
apply (*rewrite* **in** ⟨(-, *take* ⌑ -, -)⟩ *zero-uint64-nat-def*[*symmetric*])
by *sepref*

sepref-definition *isasat-GC-clauses-prog-wl-slow-code*
is ⟨*isasat-GC-clauses-prog-wl*⟩
:: ⟨*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl-def isasat-unbounded-assn-def*
IICF-Array-List.arl.fold-custom-empty isasat-GC-clauses-prog-wl2′-def[*symmetric*]
by *sepref*

sepref-definition *isa-arena-length-fast-code2*
is ⟨*uncurry isa-arena-length*⟩
:: ⟨(*arl64-assn uint32-assn*)^k $*_a$ *nat-assn*^{k} $\rightarrow_a$ *uint64-assn*⟩
supply *arena-el-assn-alt-def*[*symmetric*, *simp*] *sum-uint64-assn*[*sepref-fr-rules*]
minus-uint64-nat-assn[*sepref-fr-rules*]
unfolding *isa-arena-length-def SIZE-SHIFT-def fast-minus-def*
by *sepref*

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-fast-assn^k $*_a$ *nat-assn*^{k} → *uint64-nat-assn*⟩
using *isa-arena-length-fast-code2.refine*[*FCOMP isa-arena-length-arena-length*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *uncurry-def list-rel-compp*
by (*simp add*: *arl64-assn-comp*)

lemma *rewatch-heur-st-pre-alt-def*:
⟨*rewatch-heur-st-pre S* ⟷ (∀ *i* ∈ *set* (*get-vdom S*). *i* ≤ *uint64-max*)⟩
by (*auto simp*: *rewatch-heur-st-pre-def all-set-conv-nth*)
find-theorems ∀ *x* < *length* -. - -!- ∀ - ∈ *set* -. -
sepref-definition *rewatch-heur-st-code*
is ⟨*rewatch-heur-st*⟩
:: ⟨[λ*S*. *rewatch-heur-st-pre S* ∧ *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *append-el-aa-uint32-hnr'*[*sepref-fr-rules*] *append-ll-def*[*simp*]
unfolding *isasat-GC-clauses-prog-wl-def isasat-bounded-assn-def*
rewatch-heur-st-def Let-def two-uint64-nat-def[*symmetric*]
to-watcher-fast-def[*symmetric*] *rewatch-heur-st-pre-alt-def*
by *sepref*

sepref-definition *rewatch-heur-st-slow-code*
is ⟨*rewatch-heur-st*⟩
:: ⟨*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]] *append-el-aa-uint32-hnr'*[*sepref-fr-rules*]
unfolding *isasat-GC-clauses-prog-wl-def isasat-unbounded-assn-def*
rewatch-heur-st-def rewatch-heur-def Let-def two-uint64-nat-def[*symmetric*]
by *sepref*

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*⟩
:: ⟨[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *isasat-GC-clauses-wl-D-rewatch-pre*[*intro*!]
unfolding *isasat-GC-clauses-wl-D-def*
by *sepref*

sepref-definition *isasat-GC-clauses-wl-D-slow-code*
is ⟨*isasat-GC-clauses-wl-D*⟩
:: ⟨*isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-wl-D-def*
by *sepref*

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-max*), *uncurry0* (*RETURN uint32-max*)) ∈
$\textit{unit-assn}^k \rightarrow_a$ *nat-assn*›
by *sepref-to-hoare sep-auto*
lemma *nat-of-uint64*:
‹(*return o id*, *RETURN o nat-of-uint64*) ∈
$(\textit{uint64-assn})^k \rightarrow_a$ *uint64-nat-assn*›
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def*
nat-of-uint64-def
split: *option.splits*)

sepref-definition *number-clss-to-keep-impl*
is ‹*RETURN o number-clss-to-keep*›
:: ‹$\textit{isasat-unbounded-assn}^k \rightarrow_a$ *nat-assn*›
unfolding *number-clss-to-keep-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]] *sum-uint64-assn*[*sepref-fr-rules*]
by *sepref*

sepref-definition *number-clss-to-keep-fast-impl*
is ‹*RETURN o number-clss-to-keep*›
:: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a$ *uint64-nat-assn*›
unfolding *number-clss-to-keep-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]] *nat-of-uint64*[*sepref-fr-rules*] *sum-uint64-assn*[*sepref-fr-rules*]
by *sepref*

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*)›
:: ‹$[\textit{uncurry access-vdom-at-pre}]_a$ $\textit{isasat-unbounded-assn}^k *_a \textit{nat-assn}^k \rightarrow$ *nat-assn*›
unfolding *access-vdom-at-alt-def access-vdom-at-pre-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *access-vdom-at-fast-code*
is ‹*uncurry* (*RETURN oo access-vdom-at*)›
:: ‹$[\textit{uncurry access-vdom-at-pre}]_a$ $\textit{isasat-bounded-assn}^k *_a \textit{uint64-nat-assn}^k \rightarrow$ *uint64-nat-assn*›
unfolding *access-vdom-at-alt-def access-vdom-at-pre-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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 −
  show *?thesis*
    unfolding *cdcl-twl-stgy-restart-prog-wl-heur-def cdcl-twl-stgy-restart-prog-wl-D-def*
    apply (*intro frefI nres-relI*)
    apply (*refine-rcg*
        restart-prog-wl-D-heur-restart-prog-wl-D2[*THEN fref-to-Down-curry2*]
        cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[*THEN fref-to-Down*]
        cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D[*THEN fref-to-Down*]
        unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D[*THEN fref-to-Down*]
        WHILEIT-refine[**where** *R* = ‹*bool-rel* $\times_r$ *twl-st-heur* $\times_r$ *nat-rel*›])
    subgoal by *auto*
    subgoal unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fastforce*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    done
qed

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* {

$ebrk \leftarrow RETURN\ (\neg isasat\text{-}fast\ S_0)$;
$(ebrk,\ brk,\ T,\ n) \leftarrow$
$WHILE_T{}^{\lambda(ebrk,\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n\ \wedge}$ $(\neg ebrk \longrightarrow isasat\text{-}fast\ T) \wedge length\ (get\text{-}c$
$(\lambda(ebrk,\ brk,\ \text{-}).\ \neg brk \wedge \neg ebrk)$
$(\lambda(ebrk,\ brk,\ S,\ n).$
$do\ \{$
$ASSERT(\neg brk \wedge \neg ebrk)$;
$ASSERT(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max)$;
$T \leftarrow unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur\ S$;
$ASSERT(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ T) \leq uint64\text{-}max)$;
$ASSERT(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ T) = length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S))$;
$(brk,\ T) \leftarrow cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\text{-}heur\ T$;
$ASSERT(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ T) \leq uint64\text{-}max)$;
$(T,\ n) \leftarrow restart\text{-}prog\text{-}wl\text{-}D\text{-}heur\ T\ n\ brk$;
$ebrk \leftarrow RETURN\ (\neg isasat\text{-}fast\ T)$;
$RETURN\ (ebrk,\ brk,\ T,\ n)$
$\})$
$(ebrk,\ False,\ S_0{::}twl\text{-}st\text{-}wl\text{-}heur,\ 0)$;
$ASSERT(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ T) \leq uint64\text{-}max \wedge$
$get\text{-}old\text{-}arena\ T = [])$;
$if\ \neg brk\ then\ do\ \{$
$T \leftarrow isasat\text{-}fast\text{-}slow\ T$;
$(brk,\ T,\ \text{-}) \leftarrow WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n}$
$(\lambda(brk,\ \text{-}).\ \neg brk)$
$(\lambda(brk,\ S,\ n).$
$do\ \{$
$T \leftarrow unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur\ S$;
$(brk,\ T) \leftarrow cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\text{-}heur\ T$;
$(T,\ n) \leftarrow restart\text{-}prog\text{-}wl\text{-}D\text{-}heur\ T\ n\ brk$;
$RETURN\ (brk,\ T,\ n)$
$\})$
$(False,\ T,\ n)$;
$RETURN\ T$
$\}$
$else\ isasat\text{-}fast\text{-}slow\ T$
$\}$›

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
assumes r: ‹$r \leq uint64\text{-}max$›
shows ‹$(cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\text{-}wl\text{-}heur,\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\text{-}wl\text{-}D) \in$
$twl\text{-}st\text{-}heur'''\ r \rightarrow_f \langle twl\text{-}st\text{-}heur\rangle nres\text{-}rel$›
proof −
have *cdcl-twl-stgy-restart-prog-early-wl-D-alt-def*:
‹$cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\text{-}wl\text{-}D\ S_0 = do\ \{$
$ebrk \leftarrow RES\ UNIV$;
$(ebrk,\ brk,\ T,\ n) \leftarrow WHILE_T{}^{\lambda(\text{-},\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}D\text{-}inv\ S_0\ brk\ T\ n}$
$(\lambda(ebrk,\ brk,\ \text{-}).\ \neg brk \wedge \neg ebrk)$
$(\lambda(\text{-},\ brk,\ S,\ n).$
$do\ \{$
$T \leftarrow unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\ S$;
$(brk,\ T) \leftarrow cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\ T$;
$(T,\ n) \leftarrow restart\text{-}prog\text{-}wl\text{-}D\ T\ n\ brk$;
$ebrk \leftarrow RES\ UNIV$;
$RETURN\ (ebrk,\ brk,\ T,\ n)$

})
$(ebrk,\ False,\ S_0::nat\ twl\text{-}st\text{-}wl,\ 0)$;
if $\neg brk$ *then do* {
$T \leftarrow RETURN\ T$;
$(brk,\ T,\ \text{-}) \leftarrow WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}D\text{-}inv\ S_0\ brk\ T\ n}$
$(\lambda(brk,\ \text{-}).\ \neg brk)$
$(\lambda(brk,\ S,\ n).$
do {
$T \leftarrow unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\ S$;
$(brk,\ T) \leftarrow cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\ T$;
$(T,\ n) \leftarrow restart\text{-}prog\text{-}wl\text{-}D\ T\ n\ brk$;
$RETURN\ (brk,\ T,\ n)$
})
$(False,\ T::nat\ twl\text{-}st\text{-}wl,\ n)$;
$RETURN\ T$
}
else $RETURN\ T$
}› **for** S_0
unfolding *cdcl-twl-stgy-restart-prog-early-wl-D-def nres-monad1* **by** *auto*
have [*refine0*]: ‹$RETURN\ (\neg isasat\text{-}fast\ x) \le \Downarrow$
$\{(b,\ b').\ b = b' \wedge (b = (\neg isasat\text{-}fast\ x))\}\ (RES\ UNIV)$›
for x
by (*auto intro*: *RETURN-RES-refine*)
have [*refine0*]: ‹*isasat-fast-slow x1e*
$\le \Downarrow \{(S,\ S').\ S = x1e \wedge S' = x1b\}$
$(RETURN\ x1b)$›
for *x1e x1b*
proof −
show *?thesis*
unfolding *isasat-fast-slow-alt-def* **by** *auto*
qed
have *twl-st-heur''*: ‹$(x1e,\ x1b) \in twl\text{-}st\text{-}heur \Longrightarrow$
$(x1e,\ x1b)$
$\in twl\text{-}st\text{-}heur''$
$(dom\text{-}m\ (get\text{-}clauses\text{-}wl\ x1b))$
$(length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x1e))$›
for *x1e x1b*
by (*auto simp*: *twl-st-heur'-def*)
have *twl-st-heur'''*: ‹$(x1e,\ x1b) \in twl\text{-}st\text{-}heur''\ \mathcal{D}\ r \Longrightarrow$
$(x1e,\ x1b)$
$\in twl\text{-}st\text{-}heur'''\ r$›
for *x1e x1b r* $\mathcal{D}$
by (*auto simp*: *twl-st-heur'-def*)
have H: ‹$(xb,\ x'a)$
$\in bool\text{-}rel \times_f$
$twl\text{-}st\text{-}heur''''\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x1e) + 6 + uint\text{-}max\ div\ 2) \Longrightarrow$
$x'a = (x1f,\ x2f) \Longrightarrow$
$xb = (x1g,\ x2g) \Longrightarrow$
$(x1g,\ x1f) \in bool\text{-}rel \Longrightarrow$
$(x2e,\ x2b) \in nat\text{-}rel \Longrightarrow$
$(((x2g,\ x2e),\ x1g),\ (x2f,\ x2b),\ x1f)$
$\in twl\text{-}st\text{-}heur'''\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x2g)) \times_f$
$nat\text{-}rel \times_f$
bool-rel› **for** *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb x'a x1f x2f x1g x2g*
by *auto*

have *abs-inv*: ‹$(x, y) \in$ *twl-st-heur‴ r* $\Longrightarrow$
($ebrk$, $ebrka$) $\in \{(b, b').\ b = b' \land b = (\neg$ *isasat-fast x*$)\} \Longrightarrow$
(xb, $x'a$) $\in$ *bool-rel* $\times_f$ (*twl-st-heur* $\times_f$ *nat-rel*) $\Longrightarrow$
case x′a of
(*brk*, *xa*, *xb*) $\Rightarrow$
cdcl-twl-stgy-restart-abs-wl-D-inv y brk xa xb $\Longrightarrow$
x2f = (*x1g*, *x2g*) $\Longrightarrow$
xb = (*x1f*, *x2f*) $\Longrightarrow$
cdcl-twl-stgy-restart-abs-wl-heur-inv x x1f x1g x2g›
for *x y ebrk ebrka xa x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
x1e x2e T Ta xb x′a x1f x2f x1g x2g
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fastforce*
thm *restart-prog-wl-D-heur-restart-prog-wl-D*[*THEN fref-to-Down-curry2*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D′[*THEN fref-to-Down*]
WHILEIT-refine[**where** R = ‹*bool-rel* $\times_r$ *twl-st-heur* $\times_r$ *nat-rel*›]
WHILEIT-refine[**where** R = ‹$\{((ebrk, brk, T, n), (ebrk', brk', T', n'))$.
$(ebrk = ebrk') \land (brk = brk') \land (T, T') \in$ *twl-st-heur* $\land n = n' \land$
$(\neg ebrk \longrightarrow$ *isasat-fast T*$) \land$ *length* (*get-clauses-wl-heur T*) $\leq$ *uint64-max*$\}$›]
show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur′-def*[*simp*]
unfolding *cdcl-twl-stgy-restart-prog-early-wl-heur-def*
cdcl-twl-stgy-restart-prog-early-wl-D-alt-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry2*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D′[*THEN fref-to-Down*]
WHILEIT-refine[**where** R = ‹*bool-rel* $\times_r$ *twl-st-heur* $\times_r$ *nat-rel*›]
WHILEIT-refine[**where** R = ‹$\{((ebrk, brk, T, n), (ebrk', brk', T', n'))$.
$(ebrk = ebrk') \land (brk = brk') \land (T, T') \in$ *twl-st-heur* $\land n = n' \land$
$(\neg ebrk \longrightarrow$ *isasat-fast T*$) \land$ *length* (*get-clauses-wl-heur T*) $\leq$ *uint64-max*$\}$›])
subgoal using *r* **by** *auto*
subgoal
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur″*; *auto*; *fail*)
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur‴*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def uint64-max-def uint32-max-def*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*subst* (*asm*)(*2*) *twl-st-heur-def*) *force*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule abs-inv*)
subgoal by *auto*

apply (*rule twl-st-heur''*; *auto*; *fail*)
apply (*rule twl-st-heur'''*; *assumption*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *isasat-fast-slow-alt-def*)
done
qed

definition *length-avdom* :: ‹*twl-st-wl-heur* ⇒ *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*)›
by (*intro ext*) (*auto simp*: *length-avdom-def*)

definition *get-the-propagation-reason-heur*
:: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *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*)›
by (*intro ext*) (*auto simp*: *get-the-propagation-reason-heur-def*)

definition *clause-is-learned-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool*
where
‹*clause-is-learned-heur S C* ⟷ *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*)›
by (*intro ext*) (*auto simp*: *clause-is-learned-heur-def*)

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*)›
by (*intro ext*) (*auto simp*: *clause-lbd-heur-def get-clause-LBD-def arena-lbd-def*)

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*:

$\langle$*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*)$\rangle$
by (*intro ext*) (*auto simp*: *access-length-heur-def arena-lbd-def*)

definition *marked-as-used-st* **where**
$\langle$*marked-as-used-st T C* =
marked-as-used (*get-clauses-wl-heur T*) *C*$\rangle$

lemma *marked-as-used-st-alt-def*:
$\langle$*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*)$\rangle$
by (*intro ext*) (*auto simp*: *marked-as-used-st-def*)

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
$\langle$*D* = *Some C* $\longleftrightarrow$ *D* $\neq$ *None* $\wedge$ (*nat-of-uint64-conv* (*the D*) = *C*)$\rangle$
by *auto*

lemma (**in** −) *isasat-fast-alt-def*:
$\langle$*RETURN o isasat-fast* = (λ(*M*, *N*, -). *RETURN* (*length N* $\leq$ *uint64-max* − (*uint32-max div 2* + *6*)))$\rangle$
unfolding *isasat-fast-def*
by (*auto intro*!:*ext*)

definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
:: *twl-st-wl-heur* $\Rightarrow$ (*bool* $\times$ *twl-st-wl-heur*) *nres*
where
$\langle$*cdcl-twl-stgy-restart-prog-bounded-wl-heur* S_0 = *do* {
ebrk $\leftarrow$ *RETURN* ($\neg$*isasat-fast* S_0);
(*ebrk*, *brk*, *T*, *n*) $\leftarrow$
$WHILE_T{}^{\lambda(ebrk,\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n\ \wedge\ (\neg ebrk \longrightarrow isasat\text{-}fast\ T)\ \wedge\ length\ (get\text{-}c}$
(λ(*ebrk*, *brk*, -). $\neg$*brk* $\wedge$ $\neg$*ebrk*)
(λ(*ebrk*, *brk*, *S*, *n*).
do {
ASSERT($\neg$*brk* $\wedge$ $\neg$*ebrk*);
ASSERT(*length* (*get-clauses-wl-heur S*) $\leq$ *uint64-max*);
T $\leftarrow$ *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*)
}$\rangle$

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D*:
assumes *r*: $\langle$*r* $\leq$ *uint64-max*$\rangle$
shows $\langle$(*cdcl-twl-stgy-restart-prog-bounded-wl-heur*, *cdcl-twl-stgy-restart-prog-bounded-wl-D*) $\in$ *twl-st-heur'''* *r* $\rightarrow_f$ $\langle$*bool-rel* $\times_r$ *twl-st-heur*$\rangle$*nres-rel*$\rangle$
proof −
have *cdcl-twl-stgy-restart-prog-bounded-wl-D-alt-def*:

‹*cdcl-twl-stgy-restart-prog-bounded-wl-D* S_0 = *do* {
  ebrk ← *RES UNIV*;
  (*ebrk*, *brk*, *T*, *n*) ← $\mathit{WHILE}_T{}^{\lambda(\text{-},\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}D\text{-}inv\ S_0\ brk\ T\ n}$
    (λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
    (λ(-, *brk*, *S*, *n*).
    do {
      T ← *unit-propagation-outer-loop-wl-D S*;
      (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D T*;
      (*T*, *n*) ← *restart-prog-wl-D T n brk*;
      ebrk ← *RES UNIV*;
      RETURN (*ebrk*, *brk*, *T*, *n*)
    })
    (*ebrk*, *False*, S_0::*nat twl-st-wl*, *0*);
  RETURN (*brk*, *T*)
}› **for** S_0
  unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-D-def nres-monad1* **by** *auto*
have [*refine0*]: ‹*RETURN* (¬*isasat-fast x*) ≤ ⇓
  {(*b*, *b'*). *b* = *b'* ∧ (*b* = (¬*isasat-fast x*))} (*RES UNIV*)›
  for *x*
  by (*auto intro*: *RETURN-RES-refine*)
have [*refine0*]: ‹*isasat-fast-slow x1e*
  ≤ ⇓ {(*S*, *S'*). *S* = *x1e* ∧ *S'* = *x1b*}
  (*RETURN x1b*)›
  for *x1e x1b*
proof −
  show *?thesis*
    unfolding *isasat-fast-slow-alt-def* **by** *auto*
qed
have *twl-st-heur''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur* ⟹
  (*x1e*, *x1b*)
  ∈ *twl-st-heur''*
      (*dom-m* (*get-clauses-wl x1b*))
      (*length* (*get-clauses-wl-heur x1e*))›
  for *x1e x1b*
  by (*auto simp*: *twl-st-heur'-def*)
have *twl-st-heur'''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur''* $\mathcal{D}$ *r* ⟹
  (*x1e*, *x1b*)
  ∈ *twl-st-heur''' r*›
  for *x1e x1b r* $\mathcal{D}$
  by (*auto simp*: *twl-st-heur'-def*)
have *H*: ‹(*xb*, *x'a*)
  ∈ *bool-rel* $\times_f$
    twl-st-heur'''' (*length* (*get-clauses-wl-heur x1e*) + *6* + *uint-max div 2*) ⟹
  x'a = (*x1f*, *x2f*) ⟹
  xb = (*x1g*, *x2g*) ⟹
  (*x1g*, *x1f*) ∈ *bool-rel* ⟹
  (*x2e*, *x2b*) ∈ *nat-rel* ⟹
  (((*x2g*, *x2e*), *x1g*), (*x2f*, *x2b*), *x1f*)
  ∈ *twl-st-heur'''* (*length* (*get-clauses-wl-heur x2g*)) $\times_f$
    nat-rel $\times_f$
    bool-rel› **for** *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb*
    x'a x1f x2f x1g x2g
  by *auto*
have *abs-inv*: ‹(*x*, *y*) ∈ *twl-st-heur''' r* ⟹
  (*ebrk*, *ebrka*) ∈ {(*b*, *b'*). *b* = *b'* ∧ *b* = (¬ *isasat-fast x*)} ⟹
  (*xb*, *x'a*) ∈ *bool-rel* $\times_f$ (*twl-st-heur* $\times_f$ *nat-rel*) ⟹

case x'a of
(brk, xa, xb) ⇒
cdcl-twl-stgy-restart-abs-wl-D-inv y brk xa xb ⟹
x2f = (x1g, x2g) ⟹
xb = (x1f, x2f) ⟹
cdcl-twl-stgy-restart-abs-wl-heur-inv x x1f x1g x2g›
for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
x1e x2e T Ta xb x'a x1f x2f x1g x2g
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fastforce*
show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*]
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def*
cdcl-twl-stgy-restart-prog-bounded-wl-D-alt-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry2*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
WHILEIT-refine[**where** *R* = ‹{((*ebrk*, *brk*, *T*,*n*), (*ebrk'*, *brk'*, *T'*, *n'*)).
(*ebrk* = *ebrk'*) ∧ (*brk* = *brk'*) ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧ *n* = *n'* ∧
(¬*ebrk* ⟶ *isasat-fast T*) ∧ *length* (*get-clauses-wl-heur T*) ≤ *uint64-max*}›])
subgoal using *r* **by** *auto*
subgoal
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur'''*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def uint64-max-def uint32-max-def*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

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*›

:: ‹$isasat\text{-}unbounded\text{-}assn^k \rightarrow_a$ *nat-assn*›
unfolding *length-avdom-alt-def access-vdom-at-pre-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *length-avdom-fast-code*
is ‹*RETURN o length-avdom*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a$ *uint64-nat-assn*›
unfolding *length-avdom-alt-def access-vdom-at-pre-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*›
:: ‹$isasat\text{-}unbounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a$ *option-assn nat-assn*›
unfolding *get-the-propagation-reason-heur-alt-def access-vdom-at-pre-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *get-the-propagation-reason-heur-fast-code*
is ‹*uncurry get-the-propagation-reason-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a$ *option-assn uint64-nat-assn*›
unfolding *get-the-propagation-reason-heur-alt-def access-vdom-at-pre-def*
isasat-bounded-assn-def
supply [[*goals-limit* = *1*]]
by *sepref*

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*))›
:: ‹[λ(*S*, *C*). *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*]$_a$
$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow$ *bool-assn*›
unfolding *clause-is-learned-heur-alt-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *clause-is-learned-heur-code2*
is ‹*uncurry* (*RETURN oo* (*clause-is-learned-heur*))›
:: ‹[λ(*S*, *C*). *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*]$_a$
$isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow$ *bool-assn*›
unfolding *clause-is-learned-heur-alt-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*›
  unfolding *clause-lbd-heur-alt-def isasat-unbounded-assn-def*
  supply [[*goals-limit* = *1*]]
  by *sepref*

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*›
  unfolding *clause-lbd-heur-alt-def isasat-bounded-assn-def*
  supply [[*goals-limit* = *1*]]
  by *sepref*

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*›
  unfolding *mark-garbage-heur-def isasat-unbounded-assn-def delete-index-and-swap-alt-def*
    length-avdom-def
  supply [[*goals-limit* = *1*]]
  by *sepref*

definition *butlast-arl64* :: ‹*'a array-list64* $\Rightarrow$ -› **where**
  ‹*butlast-arl64* = (λ(*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 −
  have [*simp*]: ‹$b \leq$ *length* $l' \Longrightarrow$ (*take b* l', x) $\in$ ⟨*the-pure A*⟩*list-rel* $\Longrightarrow$
    (*take* (b − *Suc 0*) l', *take* (*length* x − *Suc 0*) x) $\in$ ⟨*the-pure A*⟩*list-rel*›
    for b l' x
    using *list-rel-take*[*of* ‹*take b* l'› x ‹*the-pure A*› ‹b −*1*›]
    by (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
      butlast-conv-take min-def
      simp del: *take-butlast-conv*)
  have [*simp*]: ‹$x \neq [] \Longrightarrow$
      nat-of-uint64 $b \leq$ *length* $l' \Longrightarrow$
      $l' \neq [] \Longrightarrow$
      length $l' \leq$ *uint64-max* $\Longrightarrow$
      (*take* (*nat-of-uint64 b*) l', x) $\in$ ⟨*the-pure A*⟩*list-rel* $\Longrightarrow$
       nat-of-uint64 (b − *1*) = *nat-of-uint64* b − *1*› **for** x b l'
    by (*metis One-nat-def Suc-leI le-0-eq list-rel-pres-neq-nil*
       nat-of-uint64-012(*3*) *nat-of-uint64-ge-minus nat-of-uint64-le-iff not-less take-eq-Nil*)
  show *?thesis*
    by *sepref-to-hoare*
    (*sep-auto simp*: *butlast-arl64-def arl64-assn-def hr-comp-def is-array-list64-def*

butlast-conv-take split: *prod.splits*
simp del: *take-butlast-conv*)
qed

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*)›
:: ‹[λ((*C*, *i*), *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *i* < *length-avdom S* ∧
get-learned-count S ≥ *1*]$_a$
uint64-nat-assn^k $*_a$ *uint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *mark-garbage-heur-def isasat-bounded-assn-def delete-index-and-swap-alt-def*
length-avdom-def one-uint64-nat-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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} → *isasat-unbounded-assn*›
unfolding *delete-index-vdom-heur-def isasat-unbounded-assn-def delete-index-and-swap-alt-def*
length-avdom-def butlast-nonresizing-def[*symmetric*] *fast-minus-def*[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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} → *isasat-bounded-assn*›
unfolding *delete-index-vdom-heur-def isasat-bounded-assn-def delete-index-and-swap-alt-def*
length-avdom-def butlast-nonresizing-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

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} → *uint64-nat-assn*›
unfolding *access-length-heur-alt-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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} → *uint64-nat-assn*›
unfolding *access-length-heur-alt-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]

by *sepref*

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*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
supply *op-eq-uint32*[*sepref-fr-rules*] *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
unfolding *isa-marked-as-used-def*
by *sepref*

lemma *isa-marked-as-used-code*[*sepref-fr-rules*]:
‹(*uncurry isa-marked-as-used-fast-code*, *uncurry* (*RETURN* ∘∘ *marked-as-used*))
$\in$ [*uncurry marked-as-used-pre*]$_a$ $arena\text{-}fast\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow bool\text{-}assn$›
using *isa-marked-as-used-fast-code.refine*[*FCOMP*
isa-marked-as-used-marked-as-used[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

sepref-definition *isa-marked-as-used-fast-code2*
is ‹*uncurry isa-marked-as-used*›
:: ‹$(arl64\text{-}assn\ uint32\text{-}assn)^k *_a nat\text{-}assn^k \rightarrow_a bool\text{-}assn$›
supply *op-eq-uint32*[*sepref-fr-rules*]
unfolding *isa-marked-as-used-def STATUS-SHIFT-def*
by *sepref*

lemma *isa-marked-as-used-code2*[*sepref-fr-rules*]:
‹(*uncurry isa-marked-as-used-fast-code2*, *uncurry* (*RETURN* ∘∘ *marked-as-used*))
$\in$ [*uncurry marked-as-used-pre*]$_a$ $arena\text{-}fast\text{-}assn^k *_a nat\text{-}assn^k \rightarrow bool\text{-}assn$›
using *isa-marked-as-used-fast-code2.refine*[*FCOMP*
isa-marked-as-used-marked-as-used[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp update-lbd-pre-def*)

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*]$_a$
$isasat\text{-}unbounded\text{-}assn^k *_a nat\text{-}assn^k \rightarrow bool\text{-}assn$›
unfolding *marked-as-used-st-alt-def isasat-unbounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*]$_a$
$isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow bool\text{-}assn$›
unfolding *marked-as-used-st-alt-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*⟩
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
apply *clarify*
apply (*rule-tac x=N* **in** *exI*)
apply (*rule-tac x=vdom* **in** *exI*)
by (*auto simp*: *arena-act-pre-def*
simp: *valid-arena-mark-unused*)

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*⟩
unfolding *mark-unused-st-heur-def isasat-unbounded-assn-def*
arena-act-pre-mark-used[*intro*!]
supply [[*goals-limit* = *1*]]
by *sepref*

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*)⟩
unfolding *isa-mark-unused-def* **supply** *STATUS-SHIFT-hnr*[*sepref-fr-rules*]
by *sepref*

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*⟩
using *isa-mark-unused-fast-code.refine*[*FCOMP isa-mark-unused-mark-unused*[*unfolded convert-fref*]]
unfolding *hr-comp-assoc*[*symmetric*] *list-rel-compp status-assn-alt-def uncurry-def*
by (*auto simp add*: *arl64-assn-comp*)

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*⟩
unfolding *mark-unused-st-heur-def isasat-bounded-assn-def*
arena-act-pre-mark-used[*intro*!]
supply [[*goals-limit* = *1*]]
by *sepref*

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-assn*^{k} $*_a$ *isasat-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
supply [[*goals-limit=1*]]

unfolding *mark-clauses-as-unused-wl-D-heur-def*
mark-unused-st-heur-def[*symmetric*]
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
arena-act-pre-mark-used[*intro*!]
by *sepref*

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*⟩
:: ⟨$[\lambda(\text{-}, S).\ length\ (get\text{-}avdom\ S) \leq uint64\text{-}max]_a$
$uint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]] *length-avdom-def*[*simp*]
unfolding *mark-clauses-as-unused-wl-D-heur-def*
mark-unused-st-heur-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
by *sepref*

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*⟩
:: ⟨$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$⟩
supply *if-splits*[*split*]
unfolding *mark-to-delete-clauses-wl-D-heur-def*
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
get-the-propagation-reason-heur-def[*symmetric*]
clause-is-learned-heur-def[*symmetric*]
clause-lbd-heur-def[*symmetric*]
access-length-heur-def[*symmetric*]
short-circuit-conv
marked-as-used-st-def[*symmetric*]
supply [[*goals-limit = 1*]]
by *sepref*

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*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
unfolding *mark-to-delete-clauses-wl-D-heur-def*
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
get-the-propagation-reason-heur-def[*symmetric*]
clause-is-learned-heur-def[*symmetric*]
clause-lbd-heur-def[*symmetric*] *nat-of-uint64-conv-def*
access-length-heur-def[*symmetric*] *zero-uint64-nat-def*[*symmetric*]

short-circuit-conv mark-to-delete-clauses-wl-D-heur-is-Some-iff
marked-as-used-st-def[*symmetric*] *one-uint64-nat-def*[*symmetric*]
supply [[*goals-limit* = *1*]] *option.splits*[*split*] *if-splits*[*split*]
length-avdom-def[*symmetric*, *simp*] *access-vdom-at-def*[*simp*]
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
unfolding *cdcl-twl-full-restart-wl-prog-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *cdcl-twl-full-restart-wl-prog-heur-fast-code*
is ⟨*cdcl-twl-full-restart-wl-prog-heur*⟩
:: ⟨[λS. *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
unfolding *cdcl-twl-full-restart-wl-prog-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
unfolding *cdcl-twl-restart-wl-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *cdcl-twl-restart-wl-heur-fast-code*
is ⟨*cdcl-twl-restart-wl-heur*⟩
:: ⟨[λS. *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
unfolding *cdcl-twl-restart-wl-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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-unbounded-assn*^{d} $\rightarrow_a$ *isasat-unbounded-assn*⟩
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-def zero-uint32-nat-def*[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code*
is ⟨*cdcl-twl-full-restart-wl-D-GC-heur-prog*⟩
:: ⟨[λS. *length* (*get-clauses-wl-heur S*) ≤ *uint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-def zero-uint32-nat-def*[*symmetric*]
supply [[*goals-limit* = *1*]]

by *sepref*

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*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d *_a nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow_a isasat\text{-}unbounded\text{-}assn *a\ nat\text{-}assn$›
unfolding *restart-prog-wl-D-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-definition *restart-prog-wl-D-heur-fast-code*
is ‹*uncurry2* (*restart-prog-wl-D-heur*)›
:: ‹$[\lambda((S, -), -).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq uint64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d *_a nat\text{-}assn^k *_a bool\text{-}assn^k \rightarrow isasat\text{-}bounded\text{-}assn *a\ nat\text{-}assn$›
unfolding *restart-prog-wl-D-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

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*›
:: ‹$isasat\text{-}unbounded\text{-}assn^d \rightarrow_a isasat\text{-}unbounded\text{-}assn$›
unfolding *cdcl-twl-stgy-restart-prog-wl-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

declare *cdcl-twl-stgy-restart-prog-wl-heur-code.refine*[*sepref-fr-rules*]

definition *isasat-fast-bound* **where**
‹*isasat-fast-bound* = *uint64-max* − (*uint32-max div 2* + *6*)›

lemma *isasat-fast-bound*[*sepref-fr-rules*]:
‹(*uncurry0* (*return 18446744071562067962*), *uncurry0* (*RETURN isasat-fast-bound*)) ∈
$unit\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
by *sepref-to-hoare* (*sep-auto simp*: *uint64-nat-rel-def br-def isasat-fast-bound-def*
uint64-max-def uint32-max-def)

sepref-register *isasat-fast*
sepref-definition *isasat-fast-code*
is ‹*RETURN o isasat-fast*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool\text{-}assn$›
unfolding *isasat-fast-alt-def isasat-bounded-assn-def isasat-fast-bound-def*[*symmetric*]
supply [[*goals-limit* = *1*]] *uint32-max-nat-hnr*[*sepref-fr-rules*]
by *sepref*

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*›
:: ‹$[\lambda S.\ \textit{isasat-fast}\ S]_a\ \textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-unbounded-assn}$›
unfolding *cdcl-twl-stgy-restart-prog-early-wl-heur-def*
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
by *sepref*

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* =
$SPEC(\lambda T.\ \exists n\ n'.\ \textit{cdcl-twl-stgy-restart-with-leftovers}^{**}\ (S,\ n)\ (T,\ n') \wedge \textit{final-twl-state}\ T)$›

To get a full CDCL run:

- either we fully apply $\textit{cdcl}_W\textit{-restart-mset.cdcl}_W\textit{-stgy}$ (up to restarts)
- or we can stop early.

definition *conclusive-CDCL-run* **where**
‹*conclusive-CDCL-run CS T U* ⟷
$(\exists n\ n'.\ \textit{cdcl}_W\textit{-restart-mset.cdcl}_W\textit{-restart-stgy}^{**}\ (T,\ n)\ (U,\ n') \wedge$
$\textit{no-step}\ \textit{cdcl}_W\textit{-restart-mset.cdcl}_W\ (U)) \vee$
$(CS \neq \{\#\} \wedge \textit{conflicting}\ U \neq \textit{None} \wedge \textit{count-decided}\ (\textit{trail}\ U) = 0 \wedge$

unsatisfiable (*set-mset CS*))›

lemma *cdcl-twl-stgy-restart-restart-prog-spec*: ‹*twl-struct-invs S* ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog S ≤ *conclusive-TWL-run S*›
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-restart-prog-spec*; *assumption?*)
unfolding *conclusive-TWL-run-def twl-restart-def*
by *auto*

lemma *cdcl-twl-stgy-restart-restart-prog-early-spec*: ‹*twl-struct-invs S* ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog-early S ≤ *conclusive-TWL-run S*›
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-prog-early-spec*; *assumption?*)
unfolding *conclusive-TWL-run-def twl-restart-def*
by *auto*

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* ≤ ⇓*Id* (*cdcl-twl-stgy-restart-prog-wl S*)›
apply (*rule cdcl-twl-stgy-restart-prog-wl-D-cdcl-twl-stgy-restart-prog-wl*[
THEN fref-to-Down, *of S S*, *THEN order-trans*])
apply *fast*
using *assms* **apply** (*auto intro*: *conc-fun-R-mono*)[]
apply (*rule conc-fun-R-mono*)
apply *auto*
done

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* ≤ ⇓*Id* (*cdcl-twl-stgy-restart-prog-early-wl S*)›
apply (*rule cdcl-twl-stgy-restart-prog-early-wl-D-cdcl-twl-stgy-restart-prog-early-wl*[
THEN fref-to-Down, *THEN order-trans*])
apply *fast*
using *assms* **apply** *auto*[]
apply (*rule conc-fun-R-mono*)
apply *auto*
done

lemma *distinct-nat-of-uint32*[*iff*]:
‹*distinct-mset* (*nat-of-uint32* ‘# *A*) ⟷ *distinct-mset A*›
‹*distinct* (*map nat-of-uint32 xs*) ⟷ *distinct xs*›
using *distinct-image-mset-inj*[*of nat-of-uint32*]
by (*auto simp*: *inj-on-def distinct-map*)

lemma $cdcl_W$*-ex-*$cdcl_W$*-stgy*:
‹$cdcl_W$*-restart-mset.*$cdcl_W$ *S T* ⟹ ∃ *U*. $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy S U*›
by (*meson* $cdcl_W$*-restart-mset.*$cdcl_W$*.cases* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.simps*)

lemma *rtranclp-*$cdcl_W$*-*$cdcl_W$*-init-state*:

‹$cdcl_W$-restart-mset.$cdcl_W{}^{**}$ (init-state {#}) S ⟷ S = init-state {#}›
unfolding *rtranclp-unfold*
by (*subst tranclp-unfold-begin*)
(*auto simp*: $cdcl_W$-stgy-$cdcl_W$-init-state-empty-no-step
$cdcl_W$-stgy-$cdcl_W$-init-state
simp del: *init-state.simps*
dest: $cdcl_W$-restart-mset.$cdcl_W$-stgy-$cdcl_W$ $cdcl_W$-ex-$cdcl_W$-stgy)

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 ⇒ 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*)›
using *dist* **apply** −
unfolding *init-dt-pre-def to-init-state-l-def init-state-l-def*
by (*rule exI*[*of* - ‹(([], {#}, {#}, *None*, {#}, {#}, {#}, {#}), {#})›])
(*auto simp*: *twl-st-l-init-def twl-init-invs*)

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-learned-clss T = {#});
          cdcl-twl-stgy-restart-prog T
        }
    }
    else do {
        let S = init-state0;
        T ← SPEC (init-dt-spec0 CS (to-init-state0 S));
        failed ← SPEC (λ- :: bool. True);
        if failed then do {
          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-learned-clss T = {#});
            cdcl-twl-stgy-restart-prog T
        }
        } else do {
          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-learned-clss T = {#});
            cdcl-twl-stgy-restart-prog-early T
          }
        }
     }
  }›
```

lemma *SAT0-SAT*:
  assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
  shows ‹*SAT0 CS* ≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*} (*SAT* (*mset* ‘# *mset CS*))›
proof −
  have *conflict-during-init*: ‹*RETURN* (*fst T*)
 ≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*}
    (*SPEC* (*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
       (*init-state* (*mset* ‘# *mset CS*))))›
    if
      spec: ‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
      confl: ‹*get-conflict* (*fst T*) ≠ *None*›
    for *T*
  proof −
    let *?CS* = ‹*mset* ‘# *mset CS*›
    have
      struct-invs: ‹*twl-struct-invs-init T*› **and**
      ‹*clauses-to-update-init T* = {#}› **and**
      count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
      ‹*get-conflict-init T* = *None* ⟶
       literals-to-update-init T =
       uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
      clss: ‹*mset* ‘# *mset CS* +
       clause ‘# *get-init-clauses-init* (*to-init-state0 init-state0*) +
       other-clauses-init (*to-init-state0 init-state0*) +
       get-unit-init-clauses-init (*to-init-state0 init-state0*) =
       clause ‘# *get-init-clauses-init T* + *other-clauses-init T* +
       get-unit-init-clauses-init T› **and**
      learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
          get-learned-clauses-init T›
        ‹*get-unit-learned-clauses-init T* =
          get-unit-learned-clauses-init (*to-init-state0 init-state0*)› **and**
      ‹*twl-stgy-invs* (*fst T*)› **and**
      ‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
      ‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
      ‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
       get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
      using *spec* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
        Set.mem-Collect-eq **apply** −
      apply *normalize-goal*+
      by *fast*+

    have *count-dec*: ‹*count-decided* (*get-trail* (*fst T*)) = *0*›
      using *count-dec* **unfolding** *count-decided-0-iff* **by** (*auto simp*: *twl-st-init*
        twl-st-wl-init)

    have *le*: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clause* ($state_W$*-of-init T*)› **and**
      all-struct-invs:
        ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* ($state_W$*-of-init T*)›
      using *struct-invs* **unfolding** *twl-struct-invs-init-def*
         $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*
      by *fast*+
    have ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-conflicting* ($state_W$*-of-init T*)›
      using *struct-invs* **unfolding** *twl-struct-invs-init-def*
        $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*
      by *fast*
    have ‹*unsatisfiable* (*set-mset* (*mset* ‘# *mset* (*rev CS*)))›

using *conflict-of-level-unsatisfiable*[*OF all-struct-invs*] *count-dec confl learned le clss*
by (*auto simp*: *clauses-def mset-take-mset-drop-mset′ twl-st-init twl-st-wl-init image-image to-init-state0-def init-state0-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def ac-simps twl-st-l-init*)
then have *unsat*[*simp*]: ‹*unsatisfiable* (*mset* ‘ *set CS*)›
by *auto*
then have [*simp*]: ‹$CS \neq []$›
by (*auto simp del*: *unsat*)
show *?thesis*
unfolding *conclusive-CDCL-run-def*
apply (*rule RETURN-SPEC-refine*)
apply (*rule exI*[*of* - ‹$state_W$*-of* (*fst T*)›])
apply (*intro conjI*)
subgoal
by *auto*
subgoal
apply (*rule disjI2*)
using *struct-invs learned count-dec clss confl*
by (*clarsimp simp*: *twl-st-init twl-st-wl-init twl-st-l-init*)
done
qed

have *empty-clauses*: ‹*RETURN* (*fst init-state0*)
$\leq \Downarrow \{(S, T).\ T = state_W\text{-}of\ S\}$
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
(*init-state* (*mset* ‘# *mset CS*))))›
if
‹$T \in$ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
‹$CS = []$›
for *T*
proof −
have [*dest*]: ‹$cdcl_W$*-restart-mset.*$cdcl_W$ ([], {#}, {#}, *None*) (*a*, *aa*, *ab*, *b*) ⟹ *False*›
for *a aa ab b*
by (*metis* $cdcl_W$*-restart-mset.*$cdcl_W$*.cases* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.conflict′* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.propagate′* $cdcl_W$*-restart-mset.other′* $cdcl_W$*-stgy-*$cdcl_W$*-init-state-empty-no-step init-state.simps*)
show *?thesis*
by (*rule RETURN-RES-refine*, *rule exI*[*of* - ‹*init-state* {#}›])
(*use that* **in** ‹*auto simp*: *conclusive-CDCL-run-def init-state0-def*›)
qed

have *extract-atms-clss-nempty*: ‹*extract-atms-clss CS* {} ≠ {}›
if
‹$T \in$ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
‹$CS \neq []$›
for *T*
proof −
show *?thesis*
using *that*
by (*cases T*; *cases CS*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*

extract-atms-clss-alt-def)
qed

have *cdcl-twl-stgy-restart-prog*: ‹*cdcl-twl-stgy-restart-prog* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$-*of S*}
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
(*init-state* (*mset* ‘# *mset CS*))))› (**is** *?G1*) **and**
cdcl-twl-stgy-restart-prog-early: ‹*cdcl-twl-stgy-restart-prog-early* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$-*of S*}
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
(*init-state* (*mset* ‘# *mset CS*))))› (**is** *?G2*)
if
spec: ‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
confl: ‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
CS-nempty[*simp*]: ‹*CS* ≠ []› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*clause* ‘# *get-clauses* (*fst T*) + *unit-clss* (*fst T*) = *mset* ‘# *mset CS*› **and**
‹*get-learned-clss* (*fst T*) = {#}›
for *T*
proof −
let *?CS* = ‹*mset* ‘# *mset CS*›
have
struct-invs: ‹*twl-struct-invs-init T*› **and**
clss-to-upd: ‹*clauses-to-update-init T* = {#}› **and**
count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
clss: ‹*mset* ‘# *mset CS* +
clause ‘# *get-init-clauses-init* (*to-init-state0 init-state0*) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) =
clause ‘# *get-init-clauses-init T* + *other-clauses-init T* +
get-unit-init-clauses-init T› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›
‹*get-unit-learned-clauses-init T* =
get-unit-learned-clauses-init (*to-init-state0 init-state0*)› **and**
stgy-invs: ‹*twl-stgy-invs* (*fst T*)› **and**
oth: ‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
using *spec* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
Set.mem-Collect-eq **apply** −
apply *normalize-goal*+
by *fast*+
have *struct-invs*: ‹*twl-struct-invs* (*fst T*)›
by (*rule twl-struct-invs-init-twl-struct-invs*)
(*use struct-invs oth confl* **in** ‹*auto simp*: *twl-st-init*›)
have *clss-to-upd*: ‹*clauses-to-update* (*fst T*) = {#}›
using *clss-to-upd* **by** (*auto simp*: *twl-st-init*)

have *conclusive-le*: ‹*conclusive-TWL-run* (*fst T*)

$\leq \Downarrow \{(S,\ T).\ T = state_W\text{-}of\ S\}$
  (*SPEC*
    (*conclusive-CDCL-run* (*mset* ‘# *mset CS*) (*init-state* (*mset* ‘# *mset CS*))))›
  unfolding *IsaSAT.conclusive-TWL-run-def*
proof (*rule RES-refine*)
  fix *Ta*
  assume *s*: ‹*Ta* ∈ {*Ta*.
      ∃ *n n′*.
        *cdcl-twl-stgy-restart-with-leftovers*** (*fst T*, *n*) (*Ta*, *n′*) ∧
        final-twl-state Ta}›
  then obtain *n n′* **where**
    twl: ‹*cdcl-twl-stgy-restart-with-leftovers*** (*fst T*, *n*) (*Ta*, *n′*)› **and**
final: ‹*final-twl-state Ta*›
by *blast*
  have *stgy-T-Ta*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst T*), *n*) ($state_W$*-of Ta*, *n′*)›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

  have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst T*), *n*) ($state_W$*-of Ta*, *n′*)›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

  have *struct-invs-x*: ‹*twl-struct-invs Ta*›
using *twl struct-invs rtranclp-cdcl-twl-stgy-restart-with-leftovers-twl-struct-invs*[*OF twl*]
by *simp*
  then have *all-struct-invs-x*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of Ta*)›
unfolding *twl-struct-invs-def*
by *blast*

  have *M-lev*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*)

  have *learned′*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* ([], *mset* ‘# *mset CS*, {#}, *None*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
by *auto*
  have *ent*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*)
  define *MW* **where** ‹*MW* ≡ *get-trail-init T*›
  have *CS-clss*: ‹$cdcl_W$*-restart-mset.clauses* ($state_W$*-of* (*fst T*)) = *mset* ‘# *mset CS*›
    using *learned clss oth confl* **unfolding** *clauses-def to-init-state0-def init-state0-def*
  $cdcl_W$*-restart-mset.clauses-def*
by (*cases T*) *auto*
  have *n-d*: ‹*no-dup MW*› **and**
propa: ‹⋀*L mark a b*. *a* @ *Propagated L mark* # *b* = *MW* ⟹
    b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*› **and**
clss-in-clss: ‹*set* (*get-all-mark-of-propagated MW*) ⊆ *set-mset ?CS*›
using *struct-invs* **unfolding** *twl-struct-invs-def twl-struct-invs-init-def*
  $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
  $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def st* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
  clauses-def MW-def clss to-init-state0-def init-state0-def CS-clss[*symmetric*]
    by ((*cases T*; *auto*)+)[*3*]

  have *count-dec′*: ‹∀ *L*∈*set MW*. ¬*is-decided L*›
using *count-dec* **unfolding** *st MW-def twl-st-init* **by** *auto*

have *st-W*: ‹$state_W$*-of* (*fst T*) = (*MW*, *?CS*, {#}, *None*)›
using *clss st learned confl oth*
by (*cases T*) (*auto simp*: *state-wl-l-init-def state-wl-l-def twl-st-l-init-def mset-take-mset-drop-mset mset-take-mset-drop-mset′ clauses-def MW-def added-only-watched-def state-wl-l-init′-def*
to-init-state0-def init-state0-def
simp del: *all-clss-l-ran-m*
simp: *all-clss-lf-ran-m*[*symmetric*])

have *0*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** ([], *?CS*, {#}, *None*)
(*MW*, *?CS*, {#}, *None*)›
using *n-d count-dec′ propa clss-in-clss*
proof (*induction MW*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons K MW*) **note** *IH* = *this*(*1*) **and** *H* = *this*(*2*−) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *propa* = *this*(*4*) **and** *clss-in-clss* = *this*(*5*)
let *?init* = ‹([], *mset* ‘# *mset CS*, {#}, *None*)›
let *?int* = ‹(*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
let *?final* = ‹(*K* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
obtain *L C* **where**
K: ‹*K* = *Propagated L C*›
using *dec* **by** (*cases K*) *auto*
term *?init*

have *1*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** *?init ?int*›
apply (*rule IH*)
subgoal using *n-d* **by** *simp*
subgoal using *dec* **by** *simp*
subgoal for *M2 L′ mark M1*
using *K propa*[*of* ‹*K* # *M2*› *L′ mark M1*]
by (*auto split*: *if-splits*)
subgoal using *clss-in-clss* **by** (*auto simp*: *K*)
done
have ‹*MW* ⊨*as CNot* (*remove1-mset L C*)› **and** ‹*L* ∈# *C*›
using *propa*[*of* ‹[]› *L C* ‹*MW*›]
by (*auto simp*: *K*)
moreover have ‹*C* ∈# $cdcl_W$*-restart-mset.clauses* (*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *clss-in-clss* **by** (*auto simp*: *K clauses-def split*: *if-splits*)
ultimately have ‹$cdcl_W$*-restart-mset.propagate ?int*
(*Propagated L C* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *n-d* **apply** −
apply (*rule* $cdcl_W$*-restart-mset.propagate-rule*[*of* - ‹*C*› *L*])
by (*auto simp*: *K*)
then have *2*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy ?int ?final*›
by (*auto simp add*: *K dest*!: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.propagate′*)

show *?case*
apply (*rule rtranclp.rtrancl-into-rtrancl*[*OF 1*])
apply (*rule 2*)
.
qed

with $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*[*OF 0*, *of n*]
have *stgy*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** (([], *mset* ‘# *mset CS*, {#}, *None*), *n*)

```
      (state_W-of Ta, n')›
    using stgy-T-Ta unfolding st-W by simp

  have entailed: ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (state_W-of Ta)›
apply (rule cdcl_W-restart-mset.rtranclp-cdcl_W-learned-clauses-entailed)
  apply (rule cdcl_W-restart-mset.rtranclp-cdcl_W-restart-stgy-cdcl_W-restart[OF stgy, unfolded fst-conv])
  apply (rule learned')
 apply (rule M-lev)
apply (rule ent)
done

  consider
    (ns) ‹no-step cdcl-twl-stgy Ta› |
    (stop) ‹get-conflict Ta ≠ None› and ‹count-decided (get-trail Ta) = 0›
    using final unfolding final-twl-state-def by auto
  then show ‹∃ s'∈Collect (conclusive-CDCL-run (mset '# mset CS)
        (init-state (mset '# mset CS))).
      (Ta, s') ∈ {(S, T). T = state_W-of S}›
  proof cases
    case ns
    from no-step-cdcl-twl-stgy-no-step-cdcl_W-stgy[OF this struct-invs-x]
    have ‹no-step cdcl_W-restart-mset.cdcl_W (state_W-of Ta)›
 by (blast dest: cdcl_W-ex-cdcl_W-stgy)
    then show ?thesis
  apply −
  apply (rule bexI[of - ‹state_W-of Ta›])
      using twl stgy s
      unfolding conclusive-CDCL-run-def
      by auto
  next
    case stop
    have ‹unsatisfiable (set-mset (init-clss (state_W-of Ta)))›
      apply (rule conflict-of-level-unsatisfiable)
        apply (rule all-struct-invs-x)
      using entailed stop by (auto simp: twl-st)
    then have ‹unsatisfiable (mset ' set CS)›
      using cdcl_W-restart-mset.rtranclp-cdcl_W-restart-init-clss[symmetric, OF
        cdcl_W-restart-mset.rtranclp-cdcl_W-restart-stgy-cdcl_W-restart[OF stgy]]
      by auto

    then show ?thesis
      using stop
      by (auto simp: twl-st-init twl-st conclusive-CDCL-run-def)
  qed
 qed
 show ?G1
  apply (rule cdcl-twl-stgy-restart-restart-prog-spec[THEN order-trans])
      apply (rule struct-invs; fail)
     apply (rule stgy-invs; fail)
    apply (rule clss-to-upd; fail)
   apply (use confl in fast; fail)
  apply (rule conclusive-le)
  done
 show ?G2
  apply (rule cdcl-twl-stgy-restart-restart-prog-early-spec[THEN order-trans])
      apply (rule struct-invs; fail)
```

    apply (*rule stgy-invs*; *fail*)
   apply (*rule clss-to-upd*; *fail*)
  apply (*use confl* **in** *fast*; *fail*)
 apply (*rule conclusive-le*)
 done
qed

show *?thesis*
 unfolding *SAT0-def SAT-def*
 apply (*refine-vcg lhs-step-If*)
 subgoal for *b T*
  by (*rule conflict-during-init*)
 subgoal by (*rule empty-clauses*)
 subgoal for *b T*
  by (*rule extract-atms-clss-nempty*)
 subgoal for *b T*
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal for *b T*
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal for *b T*
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal for *b T*
  by (*rule cdcl-twl-stgy-restart-prog*)
 subgoal for *b T*
  by (*rule conflict-during-init*)
 subgoal by (*rule empty-clauses*)
 subgoal for *b T*
  by (*rule extract-atms-clss-nempty*)
 subgoal premises *p* **for** *b - - T*
  using *p*(*6*−)
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal premises *p* **for** *b - - T*
  using *p*(*6*−)
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal premises *p* **for** *b - - T*
  using *p*(*6*−)
  by (*cases T*)
   (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
    extract-atms-clss-alt-def)
 subgoal for *b T*
  by (*rule cdcl-twl-stgy-restart-prog*)
 subgoal for *b T*
  by (*rule conflict-during-init*)
 subgoal by (*rule empty-clauses*)
 subgoal for *b T*
  by (*rule extract-atms-clss-nempty*)

subgoal for *b T*
by (*cases T*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *b T*
by (*cases T*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *b T*
by (*cases T*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *b T*
by (*rule cdcl-twl-stgy-restart-prog-early*)
done
qed

definition *SAT-l* :: ‹*nat clause-l list* ⇒ *nat twl-st-l nres*› **where**
‹*SAT-l CS* = *do*{
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state-l*;
T ← *init-dt CS* (*to-init-state-l S*);
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* {} ≠ {});
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*;
T ← *init-dt CS* (*to-init-state-l S*);
failed ← *SPEC* (λ- :: *bool*. *True*);
if failed then do {
T ← *init-dt CS* (*to-init-state-l S*);
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* {} ≠ {});
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 ≠ *None*
then RETURN T

else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
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* ≤ ⇓ {(*T*,*T*′). (*T*, *T*′) ∈ *twl-st-l None*} (*SAT0 CS*)›
proof −
have *inj*: ‹*inj* (*uminus* :: - *literal* ⇒ -)›
by (*auto simp*: *inj-on-def*)
have [*simp*]: ‹{#− *lit-of x*. *x* ∈# *A*#} = {#− *lit-of x*. *x* ∈# *B*#} ⟷
{#*lit-of x*. *x* ∈# *A*#} = {#*lit-of x*. *x* ∈# *B*#}› **for** *A B* :: ‹(*nat literal*, *nat literal*,
nat) *annotated-lit multiset*›
unfolding *multiset.map-comp*[*unfolded comp-def*, *symmetric*]
apply (*subst inj-image-mset-eq-iff* [*of uminus*])
apply (*rule inj*)
by (*auto simp*: *inj-on-def*)[]
have *get-unit-twl-st-l*: ‹(*s*, *x*) ∈ *twl-st-l-init* ⟹ *get-learned-unit-clauses-l-init s* = {#} ⟹
learned-clss-l (*get-clauses-l-init s*) = {#} ⟹
{#*mset* (*fst x*). *x* ∈# *ran-m* (*get-clauses-l-init s*)#} +
get-unit-clauses-l-init s =
clause ‘# *get-init-clauses-init x* + *get-unit-init-clauses-init x*› **for** *s x*
apply (*cases s*; *cases x*)
apply (*auto simp*: *twl-st-l-init-def mset-take-mset-drop-mset*′)
by (*metis* (*mono-tags*, *lifting*) *add.right-neutral all-clss-l-ran-m*)

have *init-dt-pre*: ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
by (*rule init-dt-pre-init*) (*use dist* **in** *auto*)

have *init-dt-spec0*: ‹*init-dt CS* (*to-init-state-l init-state-l*)
≤ ⇓{((*T*),*T*′). (*T*, *T*′) ∈ *twl-st-l-init* ∧ *twl-list-invs* (*fst T*) ∧
clauses-to-update-l (*fst T*) = {#}}
(*SPEC* (*init-dt-spec0 CS* (*to-init-state0 init-state0*)))›
apply (*rule init-dt-full*[*THEN order-trans*])
subgoal by (*rule init-dt-pre*)
subgoal
apply (*rule RES-refine*)
unfolding *init-dt-spec-def Set.mem-Collect-eq init-dt-spec0-def*
to-init-state-l-def init-state-l-def
to-init-state0-def init-state0-def
apply *normalize-goal*+
apply (*rule-tac x*=*x* **in** *bexI*)
subgoal for *s x* **by** (*auto simp*: *twl-st-l-init*)
subgoal for *s x*
unfolding *Set.mem-Collect-eq*
by (*simp-all add*: *twl-st-init twl-st-l-init twl-st-l-init-no-decision-iff get-unit-twl-st-l*)
done
done

have *init-state0*: ‹(*fst init-state-l*, *fst init-state0*) ∈ {(*T*, *T'*). (*T*, *T'*) ∈ *twl-st-l None*}›
  by (*auto simp*: *twl-st-l-def init-state0-def init-state-l-def*)
show *?thesis*
  unfolding *SAT-l-def SAT0-def*
  apply (*refine-vcg init-dt-spec0*)
  subgoal by *auto*
  subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
  subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
  subgoal by *auto*
  subgoal by (*rule init-state0*)
  subgoal for *b ba T Ta*
    unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
    by (*cases T*; *cases Ta*)
      (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*
        init-dt-spec0-def)
  subgoal for *b ba T Ta*
    unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba T Ta*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba T Ta*
    by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ‹*fst Ta*›,
        THEN order-trans])
      (*auto simp*: *twl-st-l-init-alt-def mset-take-mset-drop-mset' intro*!: *conc-fun-R-mono*)
  subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
  subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
  subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
  subgoal by *auto*
  subgoal by (*rule init-state0*)
  subgoal for *b ba - - - - T Ta*
    unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
    by (*cases T*; *cases Ta*)
      (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*
        init-dt-spec0-def)
  subgoal for *b ba - - - - T Ta*
    unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba - - - - T Ta*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba - - - - T Ta*
    by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ‹*fst Ta*›,
        THEN order-trans])
      (*auto simp*: *twl-st-l-init-alt-def intro*!: *conc-fun-R-mono*)
  subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
  subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
  subgoal by *auto*
  subgoal by (*rule init-state0*)
  subgoal by *auto*
  subgoal for *b ba T Ta*
    unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba T Ta*
   by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
  subgoal for *b ba T Ta*
    by (*rule cdcl-twl-stgy-restart-prog-early-l-cdcl-twl-stgy-restart-prog-early*[*THEN fref-to-Down*, *of* -
‹*fst Ta*›,

THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def intro*!: *conc-fun-R-mono*)
done
qed

definition *SAT-wl* :: ‹*nat clause-l list* ⇒ *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 ← *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 T
else if CS = [] *then RETURN* (([], *fmempty*, *None*, {#}, {#}, {#}, λ-. *undefined*))
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*) = {#});
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* $\mathcal{A}_{in}{}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
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* $\mathcal{A}_{in}{}'$));
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* (*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* {} ≠ {});
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;
𝒜 ← RETURN (); ~~*initialisation*~~
T ← init-dt CS (*to-init-state-l S*);
failed ← SPEC (*λ- :: bool. True*);
if failed then do {
let S = init-state-l;
𝒜 ← RETURN (); ~~*initialisation*~~
T ← init-dt CS (*to-init-state-l S*);
let T = T;
if get-conflict-l-init T ≠ None
then RETURN (*fst T*)
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
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 ≠ None
then RETURN (*fst T*)
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
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*
}
}
}
}›
unfolding *SAT-l-def* **by** (*auto cong*: *if-cong Let-def twl-st-l-init*)

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'*) ∈ *state-wl-l-init*› **and** ‹∀ *C*∈*set CS*. *distinct C*›
shows ‹*init-dt-wl-full CS S* ≤ ⇓ {(*S*, *S'*). (*fst S*, *fst S'*) ∈ *state-wl-l None*} (*init-dt CS S'*)›
proof −
have *init-dt-wl*: ‹*init-dt-wl CS S* ≤ *SPEC* (λ*T*. *RETURN T* ≤ ⇓ *state-wl-l-init* (*init-dt CS S'*) ∧
init-dt-wl-spec CS S T)›
apply (*rule SPEC-rule-conjI*)
apply (*rule order-trans*)
apply (*rule init-dt-wl-init-dt*[*of S S'*])
subgoal by (*rule assms*)
subgoal by (*rule assms*)
apply (*rule no-fail-spec-le-RETURN-itself*)
subgoal
apply (*rule SPEC-nofail*)
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule init-dt-full*)
using *assms* **by** (*auto simp*: *conc-fun-RES init-dt-wl-pre-def*)
subgoal
apply (*rule order-trans*)
apply (*rule init-dt-wl-init-dt-wl-spec*)
apply (*rule assms*)
apply *simp*
done
done

show *?thesis*
unfolding *init-dt-wl-full-def*
apply (*rule specify-left*)
apply (*rule init-dt-wl*)
subgoal for *x*
apply (*cases x*, *cases* ‹*fst x*›)
apply (*simp only*: *prod.case fst-conv*)
apply *normalize-goal*+
apply (*rule specify-left*)
apply (*rule-tac M* =*aa* **and** *N*=*ba* **and** *C*=*c* **and** *NE*=*d* **and** *UE*=*e* **and** *Q*=*f* **in**
rewatch-correctness[*OF - init-dt-wl-spec-rewatch-pre*])
subgoal by *rule*
apply (*assumption*)
apply (*auto*)[*3*]
apply (*cases* ‹*init-dt CS S'*›)
apply (*auto simp*: *RETURN-RES-refine-iff state-wl-l-def state-wl-l-init-def*
state-wl-l-init'-def)
done

done
qed

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*)›
unfolding *init-dt-wl-pre-def to-init-state-def init-state-wl-def*
apply (*rule exI*[*of* - ‹(([], *fmempty*, *None*, {#}, {#}, {#}, {#}), {#})›])
apply (*intro conjI*)
apply (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def*)[]
unfolding *init-dt-pre-def*
apply (*rule exI*[*of* - ‹(([], {#}, {#}, *None*, {#}, {#}, {#}, {#}), {#})›])
using *dist* **by** (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def twl-st-l-init-def twl-init-invs*)[]

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 –
have *extract-atms-clss*: ‹(*extract-atms-clss CS* {}, ()) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}›
by *auto*
have *init-dt-wl-pre*: ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
by (*rule init-dt-wl-pre*) (*use dist* **in** *auto*)

have *init-rel*: ‹(*to-init-state init-state-wl*, *to-init-state-l init-state-l*)
∈ *state-wl-l-init*›
by (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def twl-st-l-init-def twl-init-invs to-init-state-def init-state-wl-def init-state-l-def to-init-state-l-def*)[]

— The following stlightly strange theorem allows to reuse the definition and the correctness of *init-dt-wl-heur-full*, which was split in the definition for purely refinement-related reasons.
define *init-dt-wl-rel* **where**
‹*init-dt-wl-rel S* ≡ ({(*T*, *T'*). *RETURN T* ≤ *init-dt-wl' CS S* ∧ *T'* = ()})› **for** *S*
have *init-dt-wl'*:
‹*init-dt-wl' CS S* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ (*init-dt-wl-rel S*))›
if
‹*init-dt-wl-pre CS S*› **and**
‹(*S*, *S'*) ∈ *state-wl-l-init*› **and**
‹∀ *C*∈*set CS*. *distinct C*›
for *S S'*
proof –
have [*simp*]: ‹(*U*, *U'*) ∈ ({(*T*, *T'*). *RETURN T* ≤ *init-dt-wl' CS S* ∧ *remove-watched T* = *T'*} *O* *state-wl-l-init*) ⟷ ((*U*, *U'*) ∈ {(*T*, *T'*). *remove-watched T* = *T'*} *O* *state-wl-l-init* ∧ *RETURN U* ≤ *init-dt-wl' CS S*)› **for** *S S' U U'*
by *auto*
have *H*: ‹*A* ≤ ⇓ ({(*S*, *S'*). *P S S'*}) *B* ⟷ *A* ≤ ⇓ ({(*S*, *S'*). *RETURN S* ≤ *A* ∧ *P S S'*}) *B*›
for *A B P R*
by (*simp add*: *pw-conc-inres pw-conc-nofail pw-le-iff p2rel-def*)
have *nofail*: ‹*nofail* (*init-dt-wl' CS S*)›
apply (*rule SPEC-nofail*)
apply (*rule order-trans*)
apply (*rule init-dt-wl'-spec*[*unfolded conc-fun-RES*])

using *that* **by** *auto*
have *H*: ‹$A \le \Downarrow$ ({(*S*, *S'*). *P S S'*} *O R*) *B* $\longleftrightarrow$ $A \le \Downarrow$ ({(*S*, *S'*). *RETURN S* $\le$ *A* $\wedge$ *P S S'*} *O R*) *B*›
for *A B P R*
by (*smt Collect-cong H case-prod-cong conc-fun-chain*)
show *?thesis*
unfolding *init-dt-wl-rel-def*
using *that*
by (*auto simp*: *nofail no-fail-spec-le-RETURN-itself*)
qed

have *rewatch-st*: ‹*rewatch-st* (*from-init-state T*) $\le$
$\Downarrow$ ({(*S*, *S'*). (*S*, *fst S'*) $\in$ *state-wl-l None* $\wedge$ *correct-watching S* $\wedge$
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init S*)) (*finalise-init S*)})
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- $\le \Downarrow$ *?rewatch* -›)
if ‹(*extract-atms-clss CS* {}, $\mathcal{A}$) $\in$ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
‹(*T*, *Ta*) $\in$ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for *T Ta* **and** $\mathcal{A}$ :: *unit*
proof −
have *le-wa*: ‹$\Downarrow$ {(*T*, *T'*). *T* = *append-empty-watched T'*} *A* =
(*do* {*S* ← *A*; *RETURN* (*append-empty-watched S*)})› **for** *A R*
by (*cases A*)
(*auto simp*: *conc-fun-RES RES-RETURN-RES image-iff*)
have *init'*: ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
by (*rule init-dt-pre-init*) (*use assms* **in** *auto*)
have *H*: ‹*do* {*T* ← *RETURN T*; *rewatch-st* (*from-init-state T*)} $\le$
$\Downarrow$ {(*S'*, *T'*). *S'* = *fst T'*} (*init-dt-wl-full CS* (*to-init-state init-state-wl*))›
using *that* **unfolding** *init-dt-wl-full-def init-dt-wl-rel-def init-dt-wl'-def* **apply** −
apply (*rule bind-refine*[*of* - ‹{(*T'*, *T''*). *T'* = *append-empty-watched T''*}›])
apply (*subst le-wa*)
apply (*auto simp*: *rewatch-st-def from-init-state-def intro*!: *bind-refine*[*of* - *Id*])
done
have [*intro*]: ‹*correct-watching-init* (*af*, *ag*, *None*, *ai*, *aj*, {#}, *ba*) $\Longrightarrow$
blits-in-$\mathcal{L}_{in}$ (*af*, *ag*, *ah*, *ai*, *aj*, *ak*, *ba*)› **for** *af ag ah ai aj ak ba*
by (*auto simp*: *correct-watching-init.simps blits-in-*$\mathcal{L}_{in}$*-def*
all-blits-are-in-problem-init.simps all-lits-def
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-all-lits-of-mm-ain-atms-of-iff*
atm-of-all-lits-of-mm)

have ‹*rewatch-st* (*from-init-state T*)
$\le \Downarrow$ {(*S*, *S'*). (*S*, *fst S'*) $\in$ *state-wl-l None*}
(*init-dt CS* (*to-init-state-l init-state-l*))›
apply (*rule H*[*simplified*, *THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule init-dt-wl-full-init-dt-wl-spec-full*)
subgoal by (*rule init-dt-wl-pre*)
apply (*rule init'*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def*
init-state-l-def state-wl-l-init-def state-wl-l-init'-def)
subgoal using *assms* **by** *auto*
by (*auto intro*!: *conc-fun-R-mono simp*: *conc-fun-chain*)

moreover have ‹*rewatch-st* (*from-init-state T*) $\le$ *SPEC* (λS. *correct-watching S* $\wedge$
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init S*)) (*finalise-init S*))›

apply (*rule H*[*simplified*, *THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule Watched-Literals-Watch-List-Initialisation.init-dt-wl-full-init-dt-wl-spec-full*)
subgoal by (*rule init-dt-wl-pre*)
using *is-*$\mathcal{L}_{all}$*-all-atms-st-all-lits-st*[*of*]
by (*auto simp*: *conc-fun-RES init-dt-wl-spec-full-def correct-watching-init-correct-watching finalise-init-def literals-are-*$\mathcal{L}_{in}$*-def*)

ultimately show *?thesis*
by (*rule add-invar-refineI-P*)
qed
have *cdcl-twl-stgy-restart-prog-wl-D*: ‹*cdcl-twl-stgy-restart-prog-wl-D* (*finalise-init U*)
≤ ⇓ {(*T*, *T′*). (*T*, *T′*) ∈ *state-wl-l None*}
(*cdcl-twl-stgy-restart-prog-l* (*fst U′*))›
if
‹(*extract-atms-clss CS* {}, ($\mathcal{A}$::*unit*)) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
UU′: ‹(*U*, *U′*) ∈ *?rewatch*› **and**
‹¬ *get-conflict-wl U* ≠ *None*› **and**
‹¬ *get-conflict-l* (*fst U′*) ≠ *None*› **and**
‹*CS* ≠ []› **and**
‹*CS* ≠ []› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*clauses-to-update-l* (*fst U′*) = {#}› **and**
‹*mset* ‘# *ran-mf* (*get-clauses-l* (*fst U′*)) + *get-unit-clauses-l* (*fst U′*) =
mset ‘# *mset CS*› **and**
‹*learned-clss-l* (*get-clauses-l* (*fst U′*)) = {#}› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))› **and**
‹*mset* ‘# *ran-mf* (*get-clauses-wl U*) + *get-unit-clauses-wl U* =
mset ‘# *mset CS*› **and**
‹*learned-clss-l* (*get-clauses-wl U*) = {#}›
for $\mathcal{A}$ *T Ta U U′*
proof −
have *1*: ‹ {(*T*, *T′*). (*T*, *T′*) ∈ *state-wl-l None*} = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init U*)) (*finalise-init U*)›
using *UU′* **by** (*auto simp*: *finalise-init-def*)
show *?thesis*
apply (*rule cdcl-twl-stgy-restart-prog-wl-D-spec*[*OF lits*, *THEN order-trans*])
apply (*subst Down-id-eq*, *subst 1*)
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*])
apply *fast*
using *UU′* **by** (*auto simp*: *finalise-init-def*)
qed

have *conflict-during-init*:
‹(([], *fmempty*, *None*, {#}, {#}, {#}, λ-. *undefined*), *fst init-state-l*)
∈ {(*T*, *T′*). (*T*, *T′*) ∈ *state-wl-l None*}›
by (*auto simp*: *init-state-l-def state-wl-l-def*)

have *init-init-dt*: ‹*RETURN* (*from-init-state T*)
≤ ⇓ ({(*S*, *S′*). *S* = *fst S′*} *O* {(*S* :: *nat twl-st-wl-init-full*, *S′* :: *nat twl-st-wl-init*).
remove-watched S = *S′*} *O state-wl-l-init*)
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- ≤ ⇓ *?init-dt* - ›)

if
⟨$(extract\text{-}atms\text{-}clss\ CS\ \{\},\ (\mathcal{A}::unit)) \in \{(x,\ \text{-}).\ x = extract\text{-}atms\text{-}clss\ CS\ \{\}\}$⟩ **and**
⟨$(T,\ Ta) \in init\text{-}dt\text{-}wl\text{-}rel\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl)$⟩
for $\mathcal{A}$ T Ta
proof −
have *1*: ⟨$RETURN\ T \le init\text{-}dt\text{-}wl'\ CS\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl)$⟩
using *that* **by** (*auto simp*: *init-dt-wl-rel-def from-init-state-def*)
have *2*: ⟨$RETURN\ (from\text{-}init\text{-}state\ T) \le \Downarrow \{(S,\ S').\ S = fst\ S'\}\ (RETURN\ T)$⟩
by (*auto simp*: *RETURN-refine from-init-state-def*)
have *2*: ⟨$RETURN\ (from\text{-}init\text{-}state\ T) \le \Downarrow \{(S,\ S').\ S = fst\ S'\}\ (init\text{-}dt\text{-}wl'\ CS\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl))$⟩
apply (*rule 2*[*THEN order-trans*])
apply (*rule ref-two-step′*)
apply (*rule 1*)
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule 2*)
unfolding *conc-fun-chain*[*symmetric*]
apply (*rule ref-two-step′*)
unfolding *conc-fun-chain*
apply (*rule init-dt-wl′-init-dt*)
apply (*rule init-dt-wl-pre*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def init-state-l-def state-wl-l-init-def state-wl-l-init′-def*)
subgoal using *assms* **by** *auto*
done
qed

have *rewatch-st-fst*: ⟨$rewatch\text{-}st\ (finalise\text{-}init\ (from\text{-}init\text{-}state\ T)) \le SPEC\ (\lambda c.\ (c,\ fst\ Ta) \in \{(S,\ T).\ (S,\ T) \in state\text{-}wl\text{-}l\ None \land correct\text{-}watching\ S \land blits\text{-}in\text{-}\mathcal{L}_{in}\ S\})$⟩
(**is** ⟨$\text{-} \le SPEC\ ?rewatch$⟩)
if

⟨$(extract\text{-}atms\text{-}clss\ CS\ \{\},\ \mathcal{A}) \in \{(x,\ \text{-}).\ x = extract\text{-}atms\text{-}clss\ CS\ \{\}\}$⟩ **and**
T: ⟨$(T,\ \mathcal{A}') \in init\text{-}dt\text{-}wl\text{-}rel\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl)$⟩ **and**
T-Ta: ⟨$(from\text{-}init\text{-}state\ T,\ Ta) \in \{(S,\ S').\ S = fst\ S'\}\ O\ \{(S,\ S').\ remove\text{-}watched\ S = S'\}\ O\ state\text{-}wl\text{-}l\text{-}init$⟩ **and**
⟨$\neg\ get\text{-}conflict\text{-}wl\ (from\text{-}init\text{-}state\ T) \ne None$⟩ **and**
⟨$\neg\ get\text{-}conflict\text{-}l\text{-}init\ Ta \ne None$⟩
for $\mathcal{A}$ b ba T $\mathcal{A}'$ Ta bb bc
proof −
have *1*: ⟨$RETURN\ T \le init\text{-}dt\text{-}wl'\ CS\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl)$⟩
using *T* **unfolding** *init-dt-wl-rel-def* **by** *auto*
have *2*: ⟨$RETURN\ T \le \Downarrow \{(S,\ S').\ remove\text{-}watched\ S = S'\}\ (SPEC\ (init\text{-}dt\text{-}wl\text{-}spec\ CS\ (to\text{-}init\text{-}state\ init\text{-}state\text{-}wl)))$⟩
using *order-trans*[*OF 1 init-dt-wl′-spec*[*OF init-dt-wl-pre*]] **.**

have *empty-watched*: ⟨$get\text{-}watched\text{-}wl\ (finalise\text{-}init\ (from\text{-}init\text{-}state\ T)) = (\lambda\text{-}.\ [])$⟩
using *1 2 init-dt-wl′-spec*[*OF init-dt-wl-pre*]
by (*cases T*; *cases* ⟨$init\text{-}dt\text{-}wl\ CS\ (init\text{-}state\text{-}wl,\ \{\#\})$⟩)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff finalise-init-def from-init-state-def state-wl-l-init-def state-wl-l-init′-def to-init-state-def to-init-state-l-def init-state-l-def init-dt-wl′-def RES-RETURN-RES*)

have *1*: ‹*length* (*aa* ∝ *x*) ≥ *2*› ‹*distinct* (*aa* ∝ *x*)›
if
struct: ‹*twl-struct-invs-init*
((*af*,
{#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#},
{#}, *y*, *ac*, {#}, {#}, *ae*),
OC)› **and**
x: ‹*x* ∈# *dom-m aa*› **and**
learned: ‹*learned-clss-l aa* = {#}›
for *af aa y ac ae x OC*
proof −
have *irred*: ‹*irred aa x*›
using *that* **by** (*cases* ‹*fmlookup aa x*›) (*auto simp*: *ran-m-def dest*!: *multi-member-split*
split: *if-splits*)
have ‹*Multiset.Ball*
({#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#} +
{#})
struct-wf-twl-cls›
using *struct* **unfolding** *twl-struct-invs-init-def fst-conv twl-st-inv.simps*
by *fast*
then show ‹*length* (*aa* ∝ *x*) ≥ *2*› ‹*distinct* (*aa* ∝ *x*)›
using *x learned in-ran-mf-clause-inI*[*OF x*, *of True*] *irred*
by (*auto simp*: *mset-take-mset-drop-mset′ dest*!: *multi-member-split*[*of x*]
split: *if-splits*)
qed
have *min-len*: ‹ *x* ∈# *dom-m* (*get-clauses-wl* (*finalise-init* (*from-init-state T*))) ⟹
distinct (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*) ∧
2 ≤ *length* (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*)›
for *x*
using *2*
by (*cases T*)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init′-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl′-def RES-RETURN-RES
init-dt-spec-def init-state-wl-def twl-st-l-init-def
intro: *1*)

show *?thesis*
apply (*rule rewatch-st-correctness*[*THEN order-trans*])
subgoal by (*rule empty-watched*)
subgoal by (*rule min-len*)
subgoal using *T-Ta* **by** (*auto simp*: *finalise-init-def*
state-wl-l-init-def state-wl-l-init′-def state-wl-l-def
correct-watching-init-correct-watching
correct-watching-init-blits-in-$\mathcal{L}_{in}$)
done
qed

have *cdcl-twl-stgy-restart-prog-wl-D2*: ‹*cdcl-twl-stgy-restart-prog-wl-D U′*
≤ ⇓ {(*T*, *T′*). (*T*, *T′*) ∈ *state-wl-l None*}
(*cdcl-twl-stgy-restart-prog-l* (*fst T′*))› (**is** *?A*) **and**
cdcl-twl-stgy-restart-prog-early-wl-D2: ‹*cdcl-twl-stgy-restart-prog-early-wl-D U′*

$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-early-l* (*fst* T'))› (**is** *?B*)

if
U': ‹(U', *fst* T') $\in \{(S,\ T).\ (S,\ T) \in$ *state-wl-l None* $\wedge$ *correct-watching* S $\wedge$ *blits-in-*$\mathcal{L}_{in}$ $S\}$›
for $\mathcal{A}$ b b' T $\mathcal{A}'$ T' c c' U'
proof −
have *1*: ‹ $\{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$ = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (U')) (U')›
apply (*rule literals-are-*$\mathcal{L}_{in}$*-all-atms-st*)
using U' **by** (*auto simp*: *finalise-init-def correct-watching.simps*)
show *?A*
apply (*rule cdcl-twl-stgy-restart-prog-wl-D-spec*[*OF lits*, *THEN order-trans*])
apply (*subst Down-id-eq*, *subst 1*)
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
show *?B*
apply (*rule cdcl-twl-stgy-restart-prog-early-wl-D-spec*[*OF lits*, *THEN order-trans*])
apply (*subst Down-id-eq*, *subst 1*)
apply (*rule cdcl-twl-stgy-restart-prog-early-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
qed
have *all-le*: ‹$\forall$ $C \in$*set* CS. $\forall$ $L \in$*set* C. *nat-of-lit* $L \leq$ *uint-max*›
proof (*intro ballI*)
fix C L
assume ‹$C \in$ *set* CS› **and** ‹$L \in$ *set* C›
then have ‹$L \in\#$ $\mathcal{L}_{all}$ (*mset-set* ($\bigcup$ $C \in$*set* CS. *atm-of* ‘ *set* C))›
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit* $L \leq$ *uint-max*›
using *assms* **by** *auto*
qed
have [*simp*]: ‹(*Tc*, *fst Td*) $\in$ *state-wl-l None* $\Longrightarrow$
get-conflict-l-init Td = *get-conflict-wl Tc*› **for** *Tc Td*
by (*cases Tc*; *cases Td*; *auto simp*: *state-wl-l-def*)
show *?thesis*
unfolding *SAT-wl-def SAT-l-alt-def*
apply (*refine-vcg extract-atms-clss init-dt-wl*$'$ *init-rel*)
subgoal using *assms* **unfolding** *extract-atms-clss-alt-def* **by** *auto*
subgoal using *assms* **unfolding** *distinct-mset-set-def* **by** *auto*
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def*
simp del: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ b *ba* T *Ta* U U'
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)

subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule init-init-dt*; *assumption*)
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def simp del*: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
unfolding *twl-st-l-init*[*symmetric*]
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
by (*cases U'*; *cases U*)
(*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def state-wl-l-def*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal by (*rule conflict-during-init*)

subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def simp del*: *isasat-input-bounded-def*)
subgoal for $\mathcal{A}$ *b ba U* $\mathcal{A}'$ *T' bb bc*
by (*cases U*; *cases T'*)
(*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T* $\mathcal{A}'$ *T' bb bc*
by (*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
apply (*rule rewatch-st-fst*; *assumption*)
subgoal by (*rule cdcl-twl-stgy-restart-prog-early-wl-D2*)
done
qed

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 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 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 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 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 -
  have H: ‹A = B ⟹ A ≤ ⇓ Id B› for A B
    by auto
  have 1: ‹?A ≤ ⇓ Id ?B›
    unfolding IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto

    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    apply (rule H; solves auto)
    subgoal by auto
    done

  have 2: ‹?B ≤ ⇓ Id ?A›
    unfolding IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
```

subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
apply (*rule H*; *solves auto*)
subgoal by *auto*
done

show *?thesis*
using *1 2* **by** *simp*
qed

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-st-wl-heur-init*) ← *init-dt-wl-heur True CS S*;
      let T = convert-state $\mathcal{A}_{in}{}''$ *T*;
      T ← *rewatch-heur-st 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($\mathcal{A}_{in}{}''$ ≠ {#});

$ASSERT(isasat\text{-}input\text{-}bounded\text{-}nempty\ \mathcal{A}_{in}'')$;
- $\leftarrow$ *isasat-information-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-conflict-wl-is-None-heur U then extract-model-of-state-stat U
else extract-state-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-init-code)
else if CS = [] *then empty-conflict-code*
else do {
$ASSERT(\mathcal{A}_{in}'' \neq \{\#\})$;
$ASSERT(isasat\text{-}input\text{-}bounded\text{-}nempty\ \mathcal{A}_{in}'')$;
- $\leftarrow$ *isasat-information-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-heur-st-fast-pre T);
$T \leftarrow rewatch\text{-}heur\text{-}st\text{-}fast\ T$;
ASSERT(isasat-fast-init T);
$T \leftarrow finalise\text{-}init\text{-}code\ opts\ (T::twl\text{-}st\text{-}wl\text{-}heur\text{-}init)$;
ASSERT(isasat-fast T);
$U \leftarrow cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}prog\text{-}early\text{-}wl\text{-}heur\ T$;
RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
else extract-state-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 −
have [*simp*]: ‹$RES\ (B^{-1}\ ``\ \{g\}) = SPEC\ (\lambda c.\ (c, g) \in B)$›
by *auto*
show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

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 −
have [*simp*]: ‹$RES\ (B^{-1}\ ``\ \{g\}) = SPEC\ (\lambda c.\ (c, g) \in B)$› **for** g
by *auto*

show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *fref-to-Down-unRET-curry-SPEC*:
assumes ‹(*uncurry f*, *uncurry* (*RETURN oo g*)) $\in [P]_f$ $A \to \langle B\rangle$*nres-rel*› **and**
‹P (x, y)› **and**
‹((x', y'), (x, y)) $\in A$›
shows ‹$f\ x'\ y' \leq$ *SPEC* (λc. (c, $g\ x\ y$) $\in B$)›
proof −
have [*simp*]: ‹*RES* (B^{-1} ‘‘ $\{g\}$) = *SPEC* (λc. (c, g) $\in B$)› **for** g
by *auto*
show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *all-lits-of-mm-empty-iff*: ‹*all-lits-of-mm* A = {#} $\longleftrightarrow$ ($\forall$ C ∈# A. C = {#})›
apply (*induction* A)
subgoal by *auto*
subgoal by (*auto simp*: *all-lits-of-mm-add-mset*)
done

lemma *all-lits-of-mm-extract-atms-clss*:
‹L ∈# (*all-lits-of-mm* (*mset* ‘# *mset CS*)) $\longleftrightarrow$ *atm-of* $L \in$ *extract-atms-clss CS* {}›
by (*induction CS*)
(*auto simp*: *extract-atms-clss-alt-def all-lits-of-mm-add-mset*
in-all-lits-of-m-ain-atms-of-iff)

lemma *IsaSAT-heur-alt-def*:
‹*IsaSAT-heur opts CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT($\forall$ C∈*set CS*. $\forall$ L∈*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*;
if b
then do {
$S \leftarrow$ *init-state-wl-heur* $\mathcal{A}_{in}{}'$;
(T::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur True CS S*;
$T \leftarrow$ *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($\mathcal{A}_{in}{}'' \neq$ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}{}''$);
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*);
T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
U $\leftarrow$ *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 $\leftarrow$ *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur False CS S*;
failed $\leftarrow$ *RETURN* (*is-failed-heur-init T* $\vee$ $\neg$*isasat-fast-init T*);
if failed then do {
S $\leftarrow$ *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur True CS S*;
T $\leftarrow$ *rewatch-heur-st T*;
let T = *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(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
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*);
T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
U $\leftarrow$ *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 = *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(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
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*);
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)
}
}
}
}›
by (*auto simp*: *init-state-wl-heur-fast-def IsaSAT-heur-def isasat-init-fast-slow-alt-def convert-state-def isasat-information-banner-def cong*: *if-cong*)

lemma *rewatch-heur-st-rewatch-st*:
assumes
UV: ‹(*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 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 −
let *?𝒜* = ‹(*mset-set* (*extract-atms-clss CS* {}))›
obtain *M′ arena D′ j W′ vm φ clvls cach lbd vdom M N D NE UE Q W OC failed* **where**
U: ‹*U* = ((*M′*, *arena*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*))› **and**
V: ‹*V* = ((*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*), *OC*)›
by (*cases U*; *cases V*) *auto*
have *valid*: ‹*valid-arena arena N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
vdom-N: ‹*mset vdom* = *dom-m N*› **and**
watched: ‹(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 *?𝒜*)› **and**
lall: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm ?𝒜* (*mset* ‘# *ran-mf N*)› **and**
vdom: ‹*vdom-m ?𝒜 W N* ⊆ *set-mset* (*dom-m N*)›
using *UV* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def U V distinct-mset-dom*
empty-watched-def vdom-m-def literals-are-in-$\mathcal{L}_{in}$*-mm-def*
all-lits-of-mm-union
simp flip: *distinct-mset-mset-distinct*)

show *?thesis*
using *UV*
unfolding *rewatch-heur-st-def rewatch-st-def*
apply (*simp only*: *prod.simps from-init-state-def fst-conv nres-monad1 U V*)
apply *refine-vcg*
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def dest*: *valid-arena-vdom-subset*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist - watched lall*])
subgoal by *simp*
subgoal using *vdom-N*[*symmetric*] **by** *simp*
subgoal by (*auto simp*: *vdom-m-def*)
subgoal by (*auto simp*: *U V twl-st-heur-parsing-def Collect-eq-comp′*
twl-st-heur-parsing-no-WL-def)
done
qed

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 −
have
UV: ‹(*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-. [])}›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
unfolding *convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def*
by (*rule rewatch-heur-st-rewatch-st*[*of U V*, *THEN order-trans*])
(*auto intro*!: *conc-fun-R-mono simp*: *Collect-eq-comp′*
twl-st-heur-parsing-def)
qed

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 −
have
UV: ‹(*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-. [])}›
using *T failed* **by** (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
unfolding *convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def*
by (*rule rewatch-heur-st-rewatch-st*[*of U V*, *THEN order-trans*])
(*auto intro*!: *conc-fun-R-mono simp*: *Collect-eq-comp′*
twl-st-heur-parsing-def)
qed

lemma *IsaSAT-heur-IsaSAT*:
‹*IsaSAT-heur b CS* ≤ ⇓{((*M*, *stats*), *M′*). *M* = *map-option rev M′*} (*IsaSAT CS*)›
proof −
have *init-dt-wl-heur*: ‹*init-dt-wl-heur True CS S* ≤
⇓(*twl-st-heur-parsing-no-WL* 𝒜 *True O* {(*S*, *T*). *S* = *remove-watched T* ∧
get-watched-wl (*fst T*) = (λ-. [])})
(*init-dt-wl′ CS T*)›
if
‹*case* (*CS*, *T*) *of*

$(CS, S) \Rightarrow$
$(\forall C{\in}set\ CS.\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C)) \wedge$
distinct-mset-set (*mset* ' *set CS*)› **and**
‹$((CS, S), CS, T) \in \langle Id\rangle list\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\ \mathcal{A}\ True$›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS S*, *THEN order-trans*])
apply (*rule that*(*1*))
apply (*rule that*(*2*))
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl′-def RES-RETURN-RES conc-fun-RES Image-iff*)+
done
qed
have *init-dt-wl-heur-b*: ‹*init-dt-wl-heur False CS S* ≤
$\Downarrow(twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\ \mathcal{A}\ False\ O\ \{(S, T).\ S = remove\text{-}watched\ T \wedge$
$get\text{-}watched\text{-}wl\ (fst\ T) = (\lambda\text{-}.\ [])\})$
(*init-dt-wl′ CS T*)›
if
‹*case* (*CS*, *T*) *of*
$(CS, S) \Rightarrow$
$(\forall C{\in}set\ CS.\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ C)) \wedge$
distinct-mset-set (*mset* ' *set CS*)› **and**
‹$((CS, S), CS, T) \in \langle Id\rangle list\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\ \mathcal{A}\ True$›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS S*, *THEN order-trans*])
apply (*rule that*(*1*))
using *that*(*2*) **apply** (*force simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl′-def RES-RETURN-RES conc-fun-RES Image-iff*)+
done
qed
have *virtual-copy*: ‹$(virtual\text{-}copy\ \mathcal{A}, ()) \in \{(\mathcal{B}, c).\ c = () \wedge \mathcal{B} = \mathcal{A}\}$› **for** $\mathcal{B}\ \mathcal{A}$
by (*auto simp*: *virtual-copy-def*)
have *input-le*: ‹$\forall C{\in}set\ CS.\ \forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \le uint\text{-}max$›
if ‹*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {}))›
proof (*intro ballI*)
fix *C L*
assume ‹$C \in set\ CS$› **and** ‹$L \in set\ C$›
then have ‹$L \in\# \mathcal{L}_{all}\ (mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\}))$›
by (*auto simp*: *extract-atms-clss-alt-def* $in\text{-}\mathcal{L}_{all}\text{-}atm\text{-}of\text{-}\mathcal{A}_{in}$)
then show ‹$nat\text{-}of\text{-}lit\ L \le uint\text{-}max$›
using *that* **by** *auto*
qed
have *lits-C*: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ (mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\}))\ (mset\ C)$›
if ‹$C \in set\ CS$› **for** *C CS*
using *that*
by (*force simp*: $literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}def$ $in\text{-}\mathcal{L}_{all}\text{-}atm\text{-}of\text{-}\mathcal{A}_{in}$ *in-all-lits-of-m-ain-atms-of-iff extract-atms-clss-alt-def atm-of-eq-atm-of*)

have *init-state-wl-heur*: ‹*isasat-input-bounded* $\mathcal{A} \Longrightarrow$
init-state-wl-heur $\mathcal{A} \leq$ *SPEC* ($\lambda c.$ (*c*, *init-state-wl*) $\in$
$\{(S, S').\ (S, S') \in$ *twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *True*$\}$)› **for** $\mathcal{A}$
apply (*rule init-state-wl-heur-init-state-wl*[*THEN fref-to-Down-unRET-uncurry0-SPEC*,
of $\mathcal{A}$, *THEN order-trans*])
apply (*auto*)
done

have *get-conflict-wl-is-None-heur-init*: ‹ (*Tb*, *Tc*)
$\in$ ($\{(S,T).\ (S, T) \in$ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* $\{\}$)) *True* $\wedge$
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* $\wedge$
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* $\wedge$
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) $\wedge$
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) $\wedge$
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)$\}$ *O* $\{(S, T).\ S = (T, \{\#\})\}$) $\Longrightarrow$
($\neg$ *get-conflict-wl-is-None-heur-init Tb*) = (*get-conflict-wl Tc* $\neq$ *None*)› **for** *Tb Tc U V*
by (*cases V*) (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *get-conflict-wl-is-None-heur-init3*: ‹(*T*, *Ta*)
$\in$ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* $\{\}$)) *False O*
$\{(S, T).\ S =$ *remove-watched T* $\wedge$ *get-watched-wl* (*fst T*) = (λ-. [])$\}$ $\Longrightarrow$
(*failed*, *faileda*)
$\in \{(b, b').\ \ b = b' \wedge b =$ (*is-failed-heur-init T* $\vee$ $\neg$ *isasat-fast-init T*)$\}$ $\Longrightarrow \neg$*failed* $\Longrightarrow$
($\neg$ *get-conflict-wl-is-None-heur-init T*) = (*get-conflict-wl* (*fst Ta*) $\neq$ *None*)› **for** *T Ta failed faileda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-heur-parsing-no-WL-def*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *finalise-init-nempty*: ‹*x1i* $\neq$ *None*› ‹*x1j* $\neq$ *None*›
if
T: ‹(*Tb*, *Tc*)
$\in$ ($\{(S,T).\ (S, T) \in$ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* $\{\}$)) *True* $\wedge$
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* $\wedge$
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* $\wedge$
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) $\wedge$
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) $\wedge$
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)$\}$ *O* $\{(S, T).\ S = (T, \{\#\})\}$)› **and**
nempty: ‹*extract-atms-clss CS* $\{\} \neq \{\}$› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* $\{\}$))) *Tb* =
(*x1*, *x2*)›
for *ba S T Ta Tb Tc uu x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f*
x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k U V
proof –
show ‹*x1i* $\neq$ *None*›

using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
show ‹*x1j* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
qed

have *banner*: ‹*isasat-information-banner*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(-, -). *True*})› **for** *Tb*
by (*auto simp*: *isasat-information-banner-def*)

have *finalise-init-code*: ‹*finalise-init-code b*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?A*) **and**
finalise-init-code3: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?B*)
if
T: ‹(*Tb*, *Tc*)
∈ ({(*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*, {#})})› **and**
confl: ‹¬ *get-conflict-wl Tc* ≠ *None*› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
clss-CS: ‹*mset* ‘# *ran-mf* (*get-clauses-wl Tc*) + *get-unit-clauses-wl Tc* =
mset ‘# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl Tc*) = {#}›
for *ba S T Ta Tb Tc u v U V*
proof −
have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl* **by** *auto*
have *2*: ‹*all-atms* (*get-clauses-wl Tc*) (*get-unit-clauses-wl Tc*) ≠ {#}›
using *clss-CS nempty*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def extract-atms-clss-alt-def
all-lits-of-mm-empty-iff)
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *clss-CS nempty*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def
atm-of-all-lits-of-mm extract-atms-clss-alt-def atms-of-ms-def)
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong*

option-lookup-clause-rel-cong isa-vmtf-init-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$-*cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
show *?A*
by (*rule finalise-init-finalise-init*[*THEN fref-to-Down-unRET-curry-SPEC*, *of b*])
(*use 1 2 learned 4* **in** *auto*)
then show *?B* **unfolding** *convert-state-def* **by** *auto*
qed

have *get-conflict-wl-is-None-heur-init2*: ‹(*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-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)) =
(*get-conflict-wl* (*from-init-state V*) ≠ *None*)› **for** *U V*
by (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def twl-st-heur-parsing-no-WL-def
option-lookup-clause-rel-def convert-state-def from-init-state-def)

have *finalise-init2*: ‹*x1i* ≠ *None*› ‹*x1j* ≠ *None*›
if
T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *b O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T* =
(*x1*, *x2*)›
for *uu ba S T Ta baa uua uub x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f*
x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k b
proof −
show ‹*x1i* ≠ *None*›
using *T conv nempty*

unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
show ‹$x1j \neq None$›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
qed

have *rewatch-heur-st-fast-pre*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
∈ *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-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
for *uu ba S T Ta baa uua uub*
proof −
have ‹*valid-arena* (*get-clauses-wl-heur-init T*) (*get-clauses-wl* (*fst Ta*))
(*set* (*get-vdom-heur-init T*))›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
using *length-le* **unfolding** *rewatch-heur-st-fast-pre-def convert-state-def*
isasat-fast-init-def uint64-max-def uint32-max-def
by (*auto dest*: *valid-arena-in-vdom-le-arena*)
qed
have *rewatch-heur-st-fast-pre2*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
∈ *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**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
and
failed: ‹¬*is-failed-heur-init T*›
for *uu ba S T Ta baa uua uub*
proof −
have
Ta: ‹(*T*, *Ta*)
∈ *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-. [])}›
using *failed T* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
from *rewatch-heur-st-fast-pre*[*OF this length-le*]
show *?thesis* **.**
qed
have *finalise-init-code2*: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ {(*S'*, *T'*).
(*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S'*})›
if
Ta: ‹(*T*, *Ta*)

∈ *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**
confl: ‹¬ *get-conflict-wl* (*from-init-state Ta*) ≠ *None*› **and**
‹*CS* ≠ []› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))› **and**
clss-CS: ‹*mset* '# *ran-mf* (*get-clauses-wl* (*from-init-state Ta*)) +
get-unit-clauses-wl (*from-init-state Ta*) =
mset '# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl* (*from-init-state Ta*)) = {#}› **and**
‹*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})) ≠ {#}› **and**
‹*isasat-input-bounded-nempty*
(*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))› **and**
‹*case convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T of*
(*M'*, *N'*, *D'*, *Q'*, *W'*, *xa*, *xb*) ⇒
(*case xa of*
(*x*, *xa*) ⇒
(*case x of*
(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ⇒
λ*to-remove* (φ, *clvls*). *fst-As* ≠ *None* ∧ *lst-As* ≠ *None*)
xa)
xb› **and**
T: ‹(*Tb*, *Tc*)
∈ {(*S*, *Ta'*). (*S*, *Ta'*)
∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init T* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init T* ∧
get-clauses-wl (*fst Ta'*) = *get-clauses-wl* (*fst Ta*) ∧
get-conflict-wl (*fst Ta'*) = *get-conflict-wl* (*fst Ta*) ∧
get-unit-clauses-wl (*fst Ta'*) = *get-unit-clauses-wl* (*fst Ta*)} *O*
{(*S*, *T*). *S* = (*T*, {#})}› **and**
failed: ‹¬*is-failed-heur-init T*›
for *uu ba S T Ta baa uua uub V W b Tb Tc*
proof −
have
Ta: ‹(*T*, *Ta*)
∈ *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-. [])}›
using *failed Ta* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)

have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl T* **by** (*auto simp*: *from-init-state-def*)
have *2*: ‹*all-atms* (*get-clauses-wl Tc*) (*get-unit-clauses-wl Tc*) ≠ {#}›
using *clss-CS*[*THEN arg-cong*, *of set-mset*] *clss-CS nempty T Ta*
by (*auto 5 5 simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def extract-atms-clss-alt-def
all-lits-of-mm-empty-iff from-init-state-def)
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*all-atms-st* (*fst Ta*))›
using *T* **by** *auto*
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *clss-CS*[*THEN arg-cong*, *of set-mset*, *simplified*] *nempty T Ta*
unfolding *3*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def from-init-state-def
atm-of-all-lits-of-mm extract-atms-clss-alt-def atms-of-ms-def)

have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong*
option-lookup-clause-rel-cong isa-vmtf-init-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$-*cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])

show *?thesis*
apply (*rule finalise-init-finalise-init-full*[*unfolded conc-fun-RETURN*,
THEN order-trans])
by (*use 1 2 learned 4 T* **in** ‹*auto simp*: *from-init-state-def convert-state-def*›)
qed
have *isasat-fast*: ‹*isasat-fast Td*›
if
fast: ‹¬ ¬ *isasat-fast-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))
T)› **and**
Tb: ‹(*Tb*, *Tc*)
∈ {(*S*, *Tb*).
(*S*, *Tb*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init T* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init T* ∧
get-clauses-wl (*fst Tb*) = *get-clauses-wl* (*fst Ta*) ∧
get-conflict-wl (*fst Tb*) = *get-conflict-wl* (*fst Ta*) ∧
get-unit-clauses-wl (*fst Tb*) = *get-unit-clauses-wl* (*fst Ta*)} *O*
{(*S*, *T*). *S* = (*T*, {#})}› **and**
Td: ‹(*Td*, *Te*)
∈ {(*S'*, *T'*).
(*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S'*}›
for *uu ba S T Ta baa uua uub Tb Tc Td Te*
proof −
show *?thesis*
using *fast Td Tb*
by (*auto simp*: *convert-state-def isasat-fast-init-def isasat-fast-def*)
qed
define *init-succesfull* **where** ‹*init-succesfull T* = *RETURN* (*is-failed-heur-init T* ∨ ¬*isasat-fast-init*
T)› **for** *T*
define *init-succesfull2* **where** ‹*init-succesfull2* = *SPEC* (λ- :: *bool*. *True*)›
have [*refine*]: ‹*init-succesfull T* ≤ ⇓ {(*b*, *b'*). (*b* = *b'*) ∧ (*b* ⟷ *is-failed-heur-init T* ∨ ¬*isasat-fast-init*
T)}

init-succesfull2⟩ **for** *T*
by (*auto simp*: *init-succesfull-def init-succesfull2-def intro*!: *RETURN-RES-refine*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *IsaSAT-heur-alt-def IsaSAT-alt-def init-succesfull-def*[*symmetric*]
apply (*rewrite at* ⟨*do* {- ← *init-dt-wl′* - -; - ← (¤ :: *bool nres*); *If* - - - }⟩ *init-succesfull2-def*[*symmetric*])
apply (*refine-vcg virtual-copy init-state-wl-heur banner*
cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D[*THEN fref-to-Down*])
subgoal by (*rule input-le*)
subgoal by (*rule distinct-mset-mset-set*)
subgoal by *auto*
subgoal by *auto*
apply (*rule init-dt-wl-heur*[*of* ⟨*mset-set* (*extract-atms-clss CS* {})⟩])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
apply (*rule rewatch-heur-st-rewatch-st*; *assumption*)
subgoal unfolding *convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init*)
subgoal by (*simp add*: *empty-init-code-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by *simp*
subgoal by (*rule finalise-init-nempty*)
subgoal by (*rule finalise-init-nempty*)
apply (*rule finalise-init-code*; *assumption*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*27*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*)

apply (*rule init-dt-wl-heur-b*[*of* ⟨*mset-set* (*extract-atms-clss CS* {})⟩])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
subgoal by *fast*
apply (*rule init-dt-wl-heur*[*of* ⟨*mset-set* (*extract-atms-clss CS* {})⟩])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)

apply (*rule rewatch-heur-st-rewatch-st*; *assumption*)
subgoal unfolding *convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init*)
subgoal by (*simp add*: *empty-init-code-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by *simp*
subgoal by (*rule finalise-init-nempty*)
subgoal by (*rule finalise-init-nempty*)
apply (*rule finalise-init-code*; *assumption*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*34*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal unfolding *from-init-state-def convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init3*)
subgoal by (*simp add*: *empty-init-code-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*rule finalise-init2*)
subgoal by (*rule finalise-init2*)
subgoal for *uu ba S T Ta baa uua*
by (*rule rewatch-heur-st-fast-pre2*; *assumption?*) (*simp-all add*: *convert-state-def*)
apply (*rule rewatch-heur-st-rewatch-st3*; *assumption?*)
subgoal by *auto*
subgoal by (*clarsimp simp add*: *isasat-fast-init-def convert-state-def*)
apply (*rule finalise-init-code2*; *assumption?*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
apply (*rule-tac r1* = ⟨*length* (*get-clauses-wl-heur Td*)⟩ **in** *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-p*
THEN fref-to-Down])
subgoal by (*auto simp*: *isasat-fast-def*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*33*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done
qed

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*
by (*auto simp*: *max-def nat-of-uint32-le-iff*)

lemma *max-0L-uint32*[*simp*]: ⟨*max* (*0*::*uint32*) *a* = *a*⟩
by (*metis max.cobounded2 max-def uint32-less-than-0*)

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*))⟩
unfolding *length-get-clauses-wl-heur-init-def*
by *auto*

definition *model-if-satisfiable* :: ⟨*nat clauses* ⇒ *nat literal list option nres*⟩ **where**
⟨*model-if-satisfiable CS* = *SPEC* (λ*M*.
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*) ∈ [λ*CS*. (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id*→ ⟨*Id*⟩*nres-rel*⟩
(**is** ⟨- ∈[λ*CS*. *?P CS*]$_f$ *Id* → -⟩)
proof −
have *H*: ⟨$cdcl_W$*-restart-mset*.$cdcl_W$*-stgy-invariant* (*init-state CS*)⟩
⟨$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*init-state CS*)⟩
if ⟨*?P CS*⟩ **for** *CS*
using *that* **by** (*auto simp*:
twl-struct-invs-def twl-st-inv.simps $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.no-strange-atm-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-M-level-inv-def*
$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clause-alt-def* $cdcl_W$*-restart-mset.no-smaller-propa-def*
past-invs.simps clauses-def twl-list-invs-def twl-stgy-invs-def clause-to-update-def
$cdcl_W$*-restart-mset*.$cdcl_W$*-stgy-invariant-def*
$cdcl_W$*-restart-mset.no-smaller-confl-def*
distinct-mset-set-def)
have *H*: ⟨*s* ∈ {*M*. *if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*}⟩
if
dist: ⟨*Multiset.Ball CS distinct-mset*⟩ **and**
[*simp*]: ⟨*CS′* = *CS*⟩ **and**
s: ⟨*s* ∈ (λ*T*. *if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-run CS′* (*init-state CS′*))⟩
for *s* :: ⟨*nat literal list option*⟩ **and** *CS CS′*
proof −
obtain *T* **where**
s: ⟨(*s* = *Some* (*map lit-of* (*trail T*)) ∧ *conflicting T* = *None*) ∨
(*s* = *None* ∧ *conflicting T* ≠ *None*)⟩ **and**
conc: ⟨*conclusive-CDCL-run CS′* ([], *CS′*, {#}, *None*) *T*⟩

using *s* **by** *auto force*
consider
n n′ **where** ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*restart-stgy*** (([], *CS′*, {#}, *None*), *n*) (*T*, *n′*)›
‹*no-step* $cdcl_W$-*restart-mset*.$cdcl_W$ *T*› |
‹*CS′* ≠ {#}› **and** ‹*conflicting T* ≠ *None*› **and** ‹*backtrack-lvl T* = *0*› **and**
‹*unsatisfiable* (*set-mset CS′*)›
using *conc* **unfolding** *conclusive-CDCL-run-def*
by *auto*
then show *?thesis*
proof *cases*
case (*1 n n′*) **note** *st* = *this*(*1*) **and** *ns* = *this*(*2*)
have ‹*no-step* $cdcl_W$-*restart-mset*.$cdcl_W$-*stgy T*›
using *ns* $cdcl_W$-*restart-mset*.$cdcl_W$-*stgy*-$cdcl_W$ **by** *blast*
then have *full-T*: ‹*full* $cdcl_W$-*restart-mset*.$cdcl_W$-*stgy T T*›
unfolding *full-def* **by** *blast*

have *invs*: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*stgy-invariant T*›
‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv T*›
using *st* $cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*restart*-dcl_W-*all-struct-inv*[*OF st*]
$cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*restart*-dcl_W-*stgy-invariant*[*OF st*]
H[*OF dist*] **by** *auto*
have *res*: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*restart*** ([], *CS′*, {#}, *None*) *T*›
using $cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*restart-stgy*-$cdcl_W$-*restart*[*OF st*] **by** *simp*
have *ent*: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*learned-clauses-entailed-by-init T*›
using $cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*learned-clauses-entailed*[*OF res*] *H*[*OF dist*]
unfolding ‹*CS′* = *CS*› $cdcl_W$-*restart-mset*.$cdcl_W$-*learned-clauses-entailed-by-init-def*
$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv-def*
by *simp*
have [*simp*]: ‹*init-clss T* = *CS*›
using $cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*restart-init-clss*[*OF res*] **by** *simp*
show *?thesis*
using $cdcl_W$-*restart-mset*.*full*-$cdcl_W$-*stgy-inv-normal-form*[*OF full-T invs ent*] *s*
by (*auto simp*: *true-annots-true-cls lits-of-def*)
next
case *2*
moreover have ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*learned-clauses-entailed-by-init* (*init-state CS*)›
unfolding $cdcl_W$-*restart-mset*.$cdcl_W$-*learned-clauses-entailed-by-init-def*
by *auto*
ultimately show *?thesis*
using *H*[*OF dist*] $cdcl_W$-*restart-mset*.*full*-$cdcl_W$-*stgy-inv-normal-form*[*of* ‹*init-state CS*›
‹*init-state CS*›] *s*
by *auto*
qed
qed
show *?thesis*
unfolding *SAT′-def model-if-satisfiable-def SAT-def Let-def*
apply (*intro frefI nres-relI*)
subgoal for *CS′ CS*
unfolding *RES-RETURN-RES*
apply (*rule RES-refine*)
unfolding *pair-in-Id-conv bex-triv-one-point1 bex-triv-one-point2*
by (*rule H*)
done
qed

lemma *SAT-model-if-satisfiable′*:

‹(*uncurry* (λ-. *SAT′*), *uncurry* (λ-. *model-if-satisfiable*)) ∈
  [λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id* $\times_r$ *Id*→ ⟨*Id*⟩*nres-rel*›
 using *SAT-model-if-satisfiable* **by** (*auto simp*: *fref-def*)

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*)›
 by (*auto simp*: *twl-st-l-def convert-lits-l-map-lit-of*)

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*)›
 unfolding *IsaSAT-def SAT-l′-def*
 apply *refine-vcg*
 apply (*rule SAT-wl-SAT-l*)
 subgoal using *assms* **by** *auto*
 subgoal using *assms* **by** *auto*
 subgoal by (*auto simp*: *extract-model-of-state-def*)
 done

lemma *SAT-l′-SAT0′*:
 assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
 shows ‹*SAT-l′ CS* ≤ ⇓ *Id* (*SAT0′ CS*)›
 unfolding *SAT-l′-def SAT0′-def*
 apply *refine-vcg*
 apply (*rule SAT-l-SAT0*)
 subgoal using *assms* **by** *auto*
 subgoal by (*auto simp*: *extract-model-of-state-def*)
 done

lemma *SAT0′-SAT′*:
 assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
 shows ‹*SAT0′ CS* ≤ ⇓ *Id* (*SAT′* (*mset* ‘# *mset CS*))›
 unfolding *SAT′-def SAT0′-def*
 apply *refine-vcg*
 apply (*rule SAT0-SAT*)
 subgoal using *assms* **by** *auto*
 subgoal by (*auto simp*: *extract-model-of-state-def twl-st-l twl-st*)
 done

lemma *IsaSAT-heur-model-if-sat*:
assumes ‹∀ *C* ∈# *mset* ‘# *mset CS*. *distinct-mset C*› **and**
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-heur opts CS* ≤ ⇓ *model-stat-rel* (*model-if-satisfiable* (*mset* ‘# *mset CS*))›
apply (*rule IsaSAT-heur-IsaSAT*[*THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule ISASAT-SAT-l′*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-l′-SAT0′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT0′-SAT′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-model-if-satisfiable*[*THEN fref-to-Down*, *of* ‹*mset* ‘# *mset CS*›])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule conc-fun-R-mono*)
apply (*auto simp*: *model-stat-rel-def*)
done

lemma *IsaSAT-heur-model-if-sat′*: ‹(*uncurry IsaSAT-heur*, *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*) ∧
(∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint-max*)]$_f$
Id $\times_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨*model-stat-rel*⟩*nres-rel*›
proof −
have *H*: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
if *CS-p*: ‹∀ *C*∈#*CS′*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint-max*› **and**
‹(*CS*, *CS′*) ∈ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*›
for *CS CS′*
unfolding *isasat-input-bounded-def*
proof
fix *L*
assume *L*: ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
then obtain *C* **where**
L: ‹*C*∈*set CS* ∧ (*L* ∈*set C* ∨ − *L* ∈ *set C*)›
apply (*cases L*)
apply (*auto simp*: *extract-atms-clss-alt-def uint-max-def nat-of-uint32-uint32-of-nat-id*
$\mathcal{L}_{all}$*-def*)+
apply (*metis literal.exhaust-sel*)+
done
have ‹*nat-of-lit L* ≤ *uint-max* ∨ *nat-of-lit* (−*L*) ≤ *uint-max*›

      using *L CS-p that* **by** (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
      br-def mset-rel-def p2rel-def rel-mset-def
        rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
    then show ‹*nat-of-lit L* ≤ *uint-max*›
      using *L*
      by (*cases L*) (*auto simp*: *extract-atms-clss-alt-def uint-max-def*)
  qed
  show *?thesis*
    apply (*intro frefI nres-relI*)
    unfolding *uncurry-def*
    apply *clarify*
    subgoal for *o1 o2 o3 CS o1' o2' o3' CS'*
    apply (*rule IsaSAT-heur-model-if-sat*[*THEN order-trans*, *of CS - ‹*(*o1*, *o2*, *o3*)*›*])
    subgoal by (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
      br-def mset-rel-def p2rel-def rel-mset-def
        rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
    subgoal by (*rule H*) *auto*
    apply (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
      br-def mset-rel-def p2rel-def rel-mset-def
        rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
    done
    done
qed

definition *IsaSAT-bounded-heur* :: ‹*opts* ⇒ *nat clause-l list* ⇒ (*bool* × (*nat literal list option* × *stats*)) *nres*› **where**
  ‹*IsaSAT-bounded-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*;
    S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}$*'*;
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur False CS S*;
    let T = *convert-state* $\mathcal{A}_{in}$*'' T*;
    if ¬*get-conflict-wl-is-None-heur-init T*
    then RETURN (*True*, *empty-init-code*)
    else if CS = [] *then do* {*stat* ← *empty-conflict-code*; *RETURN* (*True*, *stat*)}
    else
    if isasat-fast-init T ∧ ¬*is-failed-heur-init T*
    then 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*);
      ASSERT(*rewatch-heur-st-fast-pre T*);
      T ← *rewatch-heur-st-fast T*;
      ASSERT(*isasat-fast-init T*);
      T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
      ASSERT(*isasat-fast T*);
      (*b*, *U*) ← *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*
}›
unfolding *nth-aa64-i32-u64-def nth-aa64-def*
nth-nat-of-uint32-nth′ nth-u-code-def[*symmetric*] **..**

lemma [*code*]: ‹*uint32-max-uint32* = *4294967295*›
by (*auto simp*: *uint32-max-uint32-def*)

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 −
have *1*: ‹*arl-assn uint32-nat-assn* ∗*a uint32-nat-assn* =
hr-comp (*arl-assn uint32-assn* ∗*a uint32-assn*) (⟨*uint32-nat-rel*⟩*list-rel* $\times_r$ *uint32-nat-rel*)›
unfolding *arl-assn-comp′ hr-comp-prod-conv*
by *auto*

have [*simp*]: ‹ *Max* (*insert 0* (*nat-of-uint32* ‘ *set aa*)) = *nat-of-uint32* (*Max* (*insert 0* (*set aa*)))›**for** *aa*
apply (*induction aa*)
subgoal by *auto*
subgoal for *a aa*
by (*cases* ‹*nat-of-uint32* ‘ *set aa* = {}›) (*auto simp*: *nat-of-uint32-max*)
done

have *2*: ‹(((⟨*uint32-nat-rel*⟩*list-rel* $\times_f$ *uint32-nat-rel*) *O lits-with-max-rel*) =
(*lits-with-max-rel O* ⟨*uint32-nat-rel*⟩*IsaSAT-Initialisation.mset-rel*)›
apply (*rule*; *rule*)
apply (*case-tac x*)
apply (*simp only*: *relcomp.simps*)
apply *normalize-goal*+
subgoal for *yx a b xa xb xc*
apply (*rule exI*[*of* - *a*])
apply (*rule exI*[*of* - ‹*uint32-of-nat* ‘# *mset* (*fst xb*)›])
apply (*rule exI*[*of* - ‹*mset* (*fst xb*)›])
apply (*cases xa*)
by (*auto simp*: *uint32-nat-rel-def IsaSAT-Initialisation.mset-rel-def p2rel-def rel2p-def*[*abs-def*]

br-def
rel-mset-def lits-with-max-rel-def list-rel-def list-all2-op-eq-map-right-iff')
apply (*case-tac x*)
apply (*simp only*: *relcomp.simps*)
apply *normalize-goal*+
subgoal for *yx a b xa xb xc*
apply (*rule exI*[*of - a*])
apply (*cases xa*)
by (*auto simp*: *uint32-nat-rel-def IsaSAT-Initialisation.mset-rel-def p2rel-def rel2p-def*[*abs-def*]
br-def
rel-mset-def lits-with-max-rel-def list-rel-def list-all2-op-eq-map-right-iff')
done

show *?thesis*
unfolding *1 hr-comp-assoc 2*
by *auto*
qed

lemma *init-state-wl-D'-code-isasat*: ‹(*hr-comp isasat-init-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-assn*›
by *auto*

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 −
have *ex-remove-xs*:
‹(∃ *xs*. *mset xs* = *mset x* ∧ {#*literal-of-nat* (*nat-of-uint32 x*). *x* ∈# *mset xs*#} = *y*) ⟷
({#*literal-of-nat* (*nat-of-uint32 x*). *x* ∈# *mset x*#} = *y*)›
for *x y*
by *auto*

show *?thesis*
unfolding *list-assn-pure-conv list-mset-assn-pure-conv*
list-rel-mset-rel-def
apply (*auto simp*: *hr-comp-def*)
apply (*auto simp*: *ent-ex-up-swap list-mset-assn-def pure-def*)
using *ex-mset*[*of* ‹*map* (λ*x*. *literal-of-nat* (*nat-of-uint32 x*)) ‘# *mset xs'*›]
by (*auto simp add*: *list-mset-rel-def br-def IsaSAT-Initialisation.mset-rel-def unat-lit-rel-def*
uint32-nat-rel-def nat-lit-rel-def WB-More-Refinement.list-mset-rel-def
p2rel-def Collect-eq-comp rel2p-def
list-all2-op-eq-map-map-right-iff rel-mset-def rel2p-def[*abs-def*]
list-all2-op-eq-map-right-iff' ex-remove-xs list-rel-def
list-all2-op-eq-map-right-iff
simp del: *literal-of-nat.simps*)
qed

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 −
have [*simp*]: ‹($\lambda a\ c.$ ↑ ((*c*, *a*) ∈ *unat-lit-rel*)) = *unat-lit-assn*›
by (*auto simp*: *unat-lit-rel-def pure-def*)
have [*simp*]: ‹*id-assn* (*an*, *ao*, *bb*) (*bs*, *bt*, *bu*) = (*id-assn an bs* ∗ *id-assn ao bt* ∗ *id-assn bb bu*)›
for *an ao bb bs bt bu* :: *uint64*
by (*auto simp*: *pure-def*)
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *twl-st-heur-def hr-comp-def trail-pol-def isasat-unbounded-assn-def*
get-trail-wl-code-def
extract-model-of-state-def extract-model-of-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*
elim!: *mod-starE*)
qed

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 −
have [*simp*]: ‹($\lambda a\ c.$ ↑ ((*c*, *a*) ∈ *unat-lit-rel*)) = *unat-lit-assn*›
by (*auto simp*: *unat-lit-rel-def pure-def*)
have [*simp*]: ‹*id-assn* (*an*, *ao*, *bb*) (*bs*, *bt*, *bu*) = (*id-assn an bs* ∗ *id-assn ao bt* ∗ *id-assn bb bu*)›
for *an ao bb bs bt bu* :: *uint64*
by (*auto simp*: *pure-def*)
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *twl-st-heur-def hr-comp-def trail-pol-def isasat-unbounded-assn-def*
get-trail-wl-code-def get-stats-code-def
extract-model-of-state-def extract-model-of-state-stat-def extract-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*
elim!: *mod-starE*)
qed

lemma *convert-state-hnr*:
‹(*uncurry* (*return oo* (λ- *S*. *S*)), *uncurry* (*RETURN oo convert-state*))
∈ *ghost-assn*^{k} $*_a$ (*isasat-init-assn*)^d $\rightarrow_a$
isasat-init-assn›
by *sepref-to-hoare* (*sep-auto simp*: *convert-state-def*)

lemma *convert-state-hnr-unb*:
‹(*uncurry* (*return oo* (λ- *S*. *S*)), *uncurry* (*RETURN oo convert-state*))
∈ *ghost-assn*^{k} $*_a$ (*isasat-init-unbounded-assn*)^d $\rightarrow_a$
isasat-init-unbounded-assn›
by *sepref-to-hoare* (*sep-auto simp*: *convert-state-def*)

lemma *IsaSAT-use-fast-mode*[*sepref-fr-rules*]:

⟨(*uncurry0* (*return IsaSAT-use-fast-mode*), *uncurry0* (*RETURN IsaSAT-use-fast-mode*))
∈ $unit\text{-}assn^k \rightarrow_a$ *bool-assn*⟩
by *sepref-to-hoare sep-auto*

sepref-definition *empty-conflict-code′*
is ⟨*uncurry0* (*empty-conflict-code*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a$ *model-stat-assn*⟩
unfolding *empty-conflict-code-def*
apply (*rewrite* **in** ⟨*let* - = ⌑ *in* -⟩ *IICF-Array-List.arl.fold-custom-empty*)
apply (*rewrite* **in** ⟨*let* - = ⌑ *in* -⟩ *annotate-assn*[**where** *A*=⟨*arl-assn unat-lit-assn*⟩])
by *sepref*

declare *empty-conflict-code′.refine*[*sepref-fr-rules*]

sepref-definition *empty-init-code′*
is ⟨*uncurry0* (*RETURN empty-init-code*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a$ *model-stat-assn*⟩
unfolding *empty-init-code-def*
by *sepref*

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*))⟩
by (*auto simp*: *isasat-fast-init-def uint64-max-def uint32-max-def isasat-fast-bound-def intro*!: *ext*)

sepref-definition *isasat-fast-init-code*
is ⟨*RETURN o isasat-fast-init*⟩
:: ⟨$isasat\text{-}init\text{-}assn^k \rightarrow_a$ *bool-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-fast-init-alt-def isasat-init-assn-def isasat-fast-bound-def*[*symmetric*]
by *sepref*

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*›
  unfolding *isasat-init-unbounded-assn-def* **by** *auto*

lemma *arena-assn-alt-def*: ‹*arl-assn* (*pure* (*uint32-nat-rel O arena-el-rel*)) = *arena-assn*›
  unfolding *hr-comp-pure*[*symmetric*] **..**

lemma [*sepref-fr-rules*]: ‹(*init-state-wl-D′-code-unb*, *init-state-wl-heur*)
∈ [λx. *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*›
  using *init-state-wl-D′-code-unb.refine*[*FCOMP init-state-wl-D′*[*unfolded convert-fref*]]
  unfolding *lits-with-max-assn-alt-def*[*symmetric*] *init-state-wl-D′-code-isasat-unb arena-assn-alt-def*
  unfolding *isasat-init-unbounded-assn-def*
  by *auto*

sepref-definition *isasat-init-fast-slow-code*
  is ‹*isasat-init-fast-slow*›
  :: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-unbounded-assn*›
  supply [[*goals-limit=1*]]
  unfolding *isasat-init-unbounded-assn-def isasat-init-assn-def isasat-init-fast-slow-def*
  apply (*rewrite at* ‹(-, -, -, -, -, -, -, -, -, -, ⌑, -)› *arl64-to-arl-conv-def*[*symmetric*])
  apply (*rewrite at* ‹(-, ⌑, -, -, -, -, -, -, -, -, -, -)› *arl64-to-arl-conv-def*[*symmetric*])
  apply (*rewrite* **in** ‹(-, -, -, -, -, -, -, -, -, -, ⌑, -)› *arl-nat-of-uint64-conv-def*[*symmetric*])
  by *sepref*

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›
  by (*sepref-to-hoare*) (*sep-auto simp*: *isasat-init-assn-def*
       elim!: *mod-starE*)

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*›
  supply [[*goals-limit=1*]] *isasat-fast-init-def*[*simp*]
  unfolding *IsaSAT-heur-def empty-conflict-def*[*symmetric*]
    get-conflict-wl-is-None extract-model-of-state-def[*symmetric*]
    extract-stats-def[*symmetric*] *init-dt-wl-heur-b-def*[*symmetric*]
    length-get-clauses-wl-heur-init-def[*symmetric*]
   init-dt-step-wl-heur-unb-def[*symmetric*] *init-dt-wl-heur-unb-def*[*symmetric*]

supply *get-conflict-wl-is-None-heur-init-def*[*simp*]
supply *id-mset-list-assn-list-mset-assn*[*sepref-fr-rules*] *get-conflict-wl-is-None-def*[*simp*]
option.splits[*split*]
extract-stats-def[*simp del*]
apply (*rewrite at* ⟨*extract-atms-clss* - ⌑⟩ *op-extract-list-empty-def*[*symmetric*])
apply (*rewrite at* ⟨*extract-atms-clss* - ⌑⟩ *op-extract-list-empty-def*[*symmetric*])
apply (*rewrite at* ⟨*extract-atms-clss* - ⌑⟩ *op-extract-list-empty-def*[*symmetric*])
by *sepref*

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*⟩
using *IsaSAT-code.refine*[*FCOMP IsaSAT-heur-model-if-sat′*[*unfolded convert-fref*],
unfolded list-assn-list-mset-rel-clauses-l-assn]
unfolding *model-assn-def*
apply *auto*
done

sepref-definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur-fast-code*
is ⟨*cdcl-twl-stgy-restart-prog-bounded-wl-heur*⟩
:: ⟨[λ*S*. *isasat-fast S*]$_a$ *isasat-bounded-assn*^{d} → *bool-assn* ∗*a* *isasat-bounded-assn*⟩
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def*
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
by *sepref*

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} →$_a$
model-stat-fast-assn⟩
proof −
have [*simp*]: ⟨(λ*a c*. ↑ ((*c*, *a*) ∈ *unat-lit-rel*)) = *unat-lit-assn*⟩
by (*auto simp*: *unat-lit-rel-def pure-def*)
have [*simp*]: ⟨*id-assn* (*an*, *ao*, *bb*) (*bs*, *bt*, *bu*) = (*id-assn an bs* ∗ *id-assn ao bt* ∗ *id-assn bb bu*)⟩
for *an ao bb bs bt bu* :: *uint64*
by (*auto simp*: *pure-def*)
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *twl-st-heur-def hr-comp-def trail-pol-def isasat-bounded-assn-def*
get-trail-wl-code-b-def
extract-model-of-state-def extract-model-of-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*
elim!: *mod-starE*)
qed

sepref-definition *empty-conflict-fast-code′*
is ⟨*uncurry0* (*empty-conflict-code*)⟩

:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-fast-assn*›
unfolding *empty-conflict-code-def*
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *arl32.fold-custom-empty*)
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *annotate-assn*[**where** *A*=‹*arl32-assn unat-lit-assn*›])
by *sepref*

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*›
unfolding *empty-init-code-def*
by *sepref*

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 −
have [*simp*]: ‹(λ*a c*. ↑ ((*c*, *a*) ∈ *unat-lit-rel*)) = *unat-lit-assn*›
by (*auto simp*: *unat-lit-rel-def pure-def*)
have [*simp*]: ‹*id-assn* (*an*, *ao*, *bb*) (*bs*, *bt*, *bu*) = (*id-assn an bs* ∗ *id-assn ao bt* ∗ *id-assn bb bu*)›
for *an ao bb bs bt bu* :: *uint64*
by (*auto simp*: *pure-def*)
show *?thesis*
by *sepref-to-hoare*
(*sep-auto simp*: *twl-st-heur-def hr-comp-def trail-pol-def isasat-bounded-assn-def*
get-trail-wl-code-def get-stats-fast-code-def
extract-model-of-state-def extract-model-of-state-stat-def extract-state-stat-def
dest!: *ann-lits-split-reasons-map-lit-of*
elim!: *mod-starE*)
qed

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*›
supply [[*goals-limit=1*]] *isasat-fast-init-def*[*simp*]
unfolding *IsaSAT-bounded-heur-def empty-conflict-def*[*symmetric*]
get-conflict-wl-is-None extract-model-of-state-def[*symmetric*]
extract-stats-def[*symmetric*]
length-get-clauses-wl-heur-init-def[*symmetric*]
init-dt-step-wl-heur-b-def[*symmetric*] *init-dt-wl-heur-b-def*[*symmetric*]
supply *get-conflict-wl-is-None-heur-init-def*[*simp*]
supply *id-mset-list-assn-list-mset-assn*[*sepref-fr-rules*] *get-conflict-wl-is-None-def*[*simp*]
option.splits[*split*]
extract-stats-def[*simp del*]
apply (*rewrite at* ‹*extract-atms-clss* - ⌑› *op-extract-list-empty-def*[*symmetric*])
apply (*rewrite at* ‹*extract-atms-clss* - ⌑› *op-extract-list-empty-def*[*symmetric*])
by *sepref*

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*) ⤜ (λ*b*. *return* (*b*, (*snd a*)))›
unfolding *arl-set-u64-def arl-set-def heap-array-set′-u64-def arl-set′-u64-def*
heap-array-set-u64-def Array.upd′-def Array-upd-u64-def
by (*cases a*) (*auto simp*: *nat-of-uint64-code*[*symmetric*])

lemma *arl-set-u-code*[*code*]: ‹*arl-set-u a i x* =
Array-upd-u i x (*fst a*) ⤜ (λ*b*. *return* (*b*, (*snd a*)))›
unfolding *arl-set-u-def arl-set-def heap-array-set′-u64-def arl-set′-u-def*
heap-array-set-u-def Array.upd′-def Array-upd-u-def
by (*cases a*) (*auto simp*: *nat-of-uint64-code*[*symmetric*])

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
```