IsaSAT: Heuristics and Code Generation

Mathias Fleury, Jasmin Blanchette, Peter Lammich

April 24, 2020

Contents

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

Chapter 1

Refinement of Literals

1.1 Literals as Natural Numbers

1.1.1 Definition

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

lemma *ex-literal-of-nat*: ‹∃ *bb*. *b* = *literal-of-nat bb*›
⟨*proof*⟩

1.1.2 Lifting to annotated literals

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*))›
⟨*proof*⟩

lemma *ann-lit-of-pair-pair-of-ann-lit*: ‹*ann-lit-of-pair* (*pair-of-ann-lit L*) = *L*›
⟨*proof*⟩

lemma *pair-of-ann-lit-ann-lit-of-pair*: ‹*pair-of-ann-lit* (*ann-lit-of-pair L*) = *L*›
⟨*proof*⟩

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ‹*literal-of-nat b* = *L* ⟷ *b* = *nat-of-lit L*›
⟨*proof*⟩

lemma *nat-of-lit-eq-iff* [*iff*]: ‹*nat-of-lit xa* = *nat-of-lit x* ⟷ *x* = *xa*›
⟨*proof*⟩

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*)}›

1.2 Conflict Clause

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

1.3 Atoms with bound

definition *uint32-max* :: *nat* **where**
‹*uint32-max* ≡ *2^32−1*›

definition *uint64-max* :: *nat* **where**
‹*uint64-max* ≡ *2^64−1*›

definition *sint32-max* :: *nat* **where**
‹*sint32-max* ≡ *2^31−1*›

definition *sint64-max* :: *nat* **where**
‹*sint64-max* ≡ *2^63−1*›

lemma *uint64-max-uint-def*: ‹*unat* (−*1* :: *64 Word.word*) = *uint64-max*›
⟨*proof*⟩

1.4 Operations with set of atoms.

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*)›

The following lemma was necessary at some point to prove the existence of some list.

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*⟩

definition *isasat-input-bounded* **where**
[*simp*]: ‹*isasat-input-bounded* = (∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* ≤ *uint32-max*)›

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

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

1.5 Set of atoms with bound

context
assumes *in-*$\mathcal{L}_{all}$*-less-uint32-max*: ‹*isasat-input-bounded*›
begin

lemma *in-*$\mathcal{L}_{all}$*-less-uint32-max'*: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$ ⟹ *nat-of-lit L* ≤ *uint32-max*›
⟨*proof*⟩

lemma *in-*$\mathcal{A}_{in}$*-less-than-uint32-max-div-2*:
‹*L* ∈# $\mathcal{A}_{in}$ ⟹ *L* ≤ *uint32-max div 2*›
⟨*proof*⟩

lemma *simple-clss-size-upper-div2'*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*› **and**

in-$\mathcal{L}_{all}$-less-uint32-max: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint32-max* − *1*›
shows ‹*size C* ≤ *uint32-max div 2*›
⟨*proof*⟩

lemma *simple-clss-size-upper-div2*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*›
shows ‹*size C* ≤ *1* + *uint32-max div 2*›
⟨*proof*⟩

lemma *clss-size-uint32-max*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*›
shows ‹*size C* ≤ *uint32-max* + *2*›
⟨*proof*⟩

lemma *clss-size-upper*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
in-$\mathcal{L}_{all}$-less-uint32-max: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint32-max* − *1*›
shows ‹*size C* ≤ *uint32-max*›
⟨*proof*⟩

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

lemma *length-trail-uint32-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* ≤ *uint32-max div 2* + *1*›
⟨*proof*⟩

end

end

1.6 Instantion for code generation

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

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

end

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

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

end

1.6.1 Literals as Natural Numbers

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

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

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

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

1.6.2 State Conversion

Functions and Types:

More Operations

1.6.3 Code Generation

More Operations

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

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

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

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

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

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

definition *find-decomp-wl-imp* :: ‹(nat, nat) ann-lits ⇒ nat clause ⇒ nat literal ⇒ (nat, nat) ann-lits nres› **where**

```
  ‹find-decomp-wl-imp = (λ$M_0$ D L. do {
    let lev = get-maximum-level $M_0$ (remove1-mset (−L) D);
    let k = count-decided $M_0$;
    (-, M) ←
      $WHILE_T{}^{λ(j, M). j = count-decided M ∧ j ≥ lev ∧            (M = [] ⟶ j = lev) ∧            (∃ M'. M_0 = M' @ M ∧ (j =}$
         (λ(j, M). j > lev)
         (λ(j, M). do {
            ASSERT(M ≠ []);
            if is-decided (hd M)
            then RETURN (j−1, tl M)
            else RETURN (j, tl M)}
         )
         (k, $M_0$);
    RETURN M
  })›
```

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

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

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

definition *find-decomp-wl'* **where**

```
  ‹find-decomp-wl' =
     (λ(M::(nat, nat) ann-lits) (D::nat clause) (L::nat literal).
        SPEC(λM1. ∃ K M2. (Decided K # M1, M2) ∈ set (get-all-ann-decomposition M) ∧
          get-level M K = get-maximum-level M (D − {#−L#}) + 1))›
```

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

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

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

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

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

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*›
⟨*proof*⟩

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* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
⟨*proof*⟩

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

Chapter 2

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 (was optional in cadical too; since sr-19, not optional);
2. the status and LBD;
3. the size;
4. 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 for the status;

In previous iteration, we had something a bit simpler:

1. the LBD was in a seperate field, allowing to store the complete LBD (which does not matter).

2. the activity was also stored and used for ties. This was beneficial on some problems (including the *eq.atree.braun* problems), but we later decided to remove it to consume less memory. This did not make a difference on the overall benchmark set. For ties, we use a pure MTF-like scheme and keep newer clauses (like CaDiCaL).

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 *uint32-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.

2.1 Status of a clause

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

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

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

end

2.2 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**
$\langle \textit{POS-SHIFT} = \textit{3} \rangle$

definition *STATUS-SHIFT* :: *nat* **where**
$\langle \textit{STATUS-SHIFT} = \textit{2} \rangle$

abbreviation *LBD-SHIFT* :: *nat* **where**
$\langle \textit{LBD-SHIFT} \equiv \textit{STATUS-SHIFT} \rangle$

lemmas *LBD-SHIFT-def* = *STATUS-SHIFT-def*

definition *SIZE-SHIFT* :: *nat* **where**
$\langle \textit{SIZE-SHIFT} = \textit{1} \rangle$

definition *MAX-LENGTH-SHORT-CLAUSE* :: *nat* **where**
[*simp*]: $\langle \textit{MAX-LENGTH-SHORT-CLAUSE} = \textit{4} \rangle$

definition *is-short-clause* **where**
[*simp*]: $\langle \textit{is-short-clause } C \longleftrightarrow \textit{length } C \leq \textit{MAX-LENGTH-SHORT-CLAUSE} \rangle$

abbreviation *is-long-clause* **where**
$\langle \textit{is-long-clause } C \equiv \neg \textit{is-short-clause } C \rangle$

abbreviation (*input*) *MAX-HEADER-SIZE* :: ‹*nat*› **where**
$\langle \textit{MAX-HEADER-SIZE} \equiv \textit{3} \rangle$

abbreviation (*input*) *MIN-HEADER-SIZE* :: ‹*nat*› **where**
$\langle \textit{MIN-HEADER-SIZE} \equiv \textit{2} \rangle$

definition *header-size* :: ‹*nat clause-l* ⇒ *nat*› **where**
$\langle \textit{header-size } C = (\textit{if is-short-clause } C \textit{ then MIN-HEADER-SIZE else MAX-HEADER-SIZE}) \rangle$

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

In an attempt to avoid unfolding definitions and to not rely on the actual value of the positions of the headers before the clauses.

lemma *arena-shift-distinct*:
$\langle i > \textit{MIN-HEADER-SIZE} \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{LBD-SHIFT} \rangle$
$\langle i > \textit{MIN-HEADER-SIZE} \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{STATUS-SHIFT} \rangle$

$\langle i > \textit{MAX-HEADER-SIZE} \Longrightarrow i - \textit{SIZE-SHIFT} \neq i - \textit{POS-SHIFT} \rangle$
$\langle i > \textit{MAX-HEADER-SIZE} \Longrightarrow i - \textit{LBD-SHIFT} \neq i - \textit{POS-SHIFT} \rangle$
$\langle i > \textit{MAX-HEADER-SIZE} \Longrightarrow i - \textit{STATUS-SHIFT} \neq i - \textit{POS-SHIFT} \rangle$

$\langle i > \textit{MIN-HEADER-SIZE} \Longrightarrow j > \textit{MIN-HEADER-SIZE} \Longrightarrow i - \textit{SIZE-SHIFT} = j - \textit{SIZE-SHIFT} \longleftrightarrow i = j \rangle$
$\langle i > \textit{MIN-HEADER-SIZE} \Longrightarrow j > \textit{MIN-HEADER-SIZE} \Longrightarrow i - \textit{LBD-SHIFT} = j - \textit{LBD-SHIFT} \longleftrightarrow i = j \rangle$
$\langle i > \textit{MIN-HEADER-SIZE} \Longrightarrow j > \textit{MIN-HEADER-SIZE} \Longrightarrow i - \textit{STATUS-SHIFT} = j - \textit{STATUS-SHIFT} \longleftrightarrow i = j \rangle$

‹$i > \textit{MAX-HEADER-SIZE} \Longrightarrow j > \textit{MAX-HEADER-SIZE} \Longrightarrow i - \textit{POS-SHIFT} = j - \textit{POS-SHIFT} \longleftrightarrow i = j$›

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

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

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

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

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

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

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

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

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

    drop (*header-size* (*N*∝*i*)) *arena* = *map ALit* (*N*∝*i*)
  ))›
⟨*proof*⟩

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

definition *arena-dead-clause* :: ‹*arena* ⇒ *bool*› **where**
‹*arena-dead-clause arena* ⟷
is-Status(*arena*!(*MIN-HEADER-SIZE* − *STATUS-SHIFT*)) ∧ *xarena-status*(*arena*!(*MIN-HEADER-SIZE* − *STATUS-SHIFT*)) = *DELETED* ∧
is-Size(*arena*!(*MIN-HEADER-SIZE* − *SIZE-SHIFT*))
›

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

definition *extra-information-mark-to-delete* **where**
‹*extra-information-mark-to-delete arena i* = *arena*[*i* − *STATUS-SHIFT* := *AStatus DELETED 0 0*]›

This extracts a single clause from the complete arena.

abbreviation *clause-slice* **where**
‹*clause-slice arena N i* ≡ *Misc.slice* (*i* − *header-size* (*N*∝*i*)) (*i* + *length*(*N*∝*i*)) *arena*›

abbreviation *dead-clause-slice* **where**
‹*dead-clause-slice arena N i* ≡ *Misc.slice* (*i* − *MIN-HEADER-SIZE*) *i arena*›

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

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

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

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

definition *arena-status* **where**
‹*arena-status arena i* = *xarena-status* (*arena*!(*i* − *STATUS-SHIFT*))›

definition *arena-used* **where**
‹*arena-used arena i* = *xarena-used* (*arena*!(*i* − *STATUS-SHIFT*))›

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

definition *arena-lbd* **where**
‹*arena-lbd arena i* = *xarena-lbd* (*arena*!(*i* − *LBD-SHIFT*))›

definition *arena-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*)⟩

2.3 Separation properties

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

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

lemma *minimal-difference-between-invalid-index*:
assumes ⟨*valid-arena arena N vdom*⟩ **and**
⟨*i* ∈# *dom-m N*⟩ **and** ⟨*j* ∉# *dom-m N*⟩ **and** ⟨*j* ≥ *i*⟩ **and** ⟨*j* ∈ *vdom*⟩
shows ⟨*j* − *i* ≥ *length* (*N*∝*i*) + *MIN-HEADER-SIZE*⟩
⟨*proof*⟩

At first we had the weaker (*1*::′*a*) ≤ *i* − *j* which we replaced by (*4*::′*a*) ≤ *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* (whi::natch 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* ∈# *dom-m N*⟩ **and** ⟨*j* ∉# *dom-m N*⟩ **and** ⟨*j* < *i*⟩ **and** ⟨*j* ∈ *vdom*⟩
shows ⟨*i* − *j* ≥ (*Suc* (*Suc 0*))⟩ **and**
⟨*is-long-clause* (*N* ∝ *i*) ⟹ *i* − *j* ≥ (*Suc* (*Suc* (*Suc 0*)))⟩
⟨*proof*⟩

lemma *valid-arena-in-vdom-le-arena*:
assumes ⟨*valid-arena arena N vdom*⟩ **and** ⟨*j* ∈ *vdom*⟩
shows ⟨*j* < *length arena*⟩ **and** ⟨*j* ≥ *MIN-HEADER-SIZE*⟩
⟨*proof*⟩

lemma *valid-minimal-difference-between-valid-index*:
assumes ⟨*valid-arena arena N vdom*⟩ **and**
⟨*i* ∈# *dom-m N*⟩ **and** ⟨*j* ∈# *dom-m N*⟩ **and** ⟨*j* > *i*⟩
shows ⟨*j* − *i* ≥ *length* (*N*∝*i*) + *header-size* (*N*∝*j*)⟩
⟨*proof*⟩

Updates

Mark to delete **lemma** *clause-slice-extra-information-mark-to-delete*:
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* (*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*∝*i*))
else clause-slice arena N ia)›
⟨*proof*⟩

lemma *clause-slice-extra-information-mark-to-delete-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* (*extra-information-mark-to-delete arena i*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
⟨*proof*⟩

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

lemma *valid-arena-mono*: ‹*valid-arena ab ar vdom1* ⟹ *vdom2* ⊆ *vdom1* ⟹ *valid-arena ab ar vdom2*›
⟨*proof*⟩

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

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

Removable from addressable space

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

Update LBD

abbreviation *MAX-LBD* :: ‹*nat*› **where**
‹*MAX-LBD* ≡ *67108863*›

lemma *MAX-LBD-alt-def*:
‹*MAX-LBD* = (*2^26−1*)›
⟨*proof*⟩

definition *shorten-lbd* :: ‹*nat* ⇒ *nat*› **where**
‹*shorten-lbd n* = (*if n* ≥ *MAX-LBD then MAX-LBD else n*)›

definition *update-lbd* **where**
‹*update-lbd C lbd arena* = *arena*[*C* − *LBD-SHIFT* := *AStatus* (*arena-status arena C*)
(*arena-used arena C*) (*shorten-lbd 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*⟩

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

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*⟩

lemma *xarena-active-clause-update-lbd-same*:
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* (*update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
⟨*proof*⟩

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

Update saved position

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

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

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

lemma *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*⟩

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*⟩

lemma *xarena-active-clause-update-pos-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**
long: ‹*is-long-clause* (*N* ∝ *i*)› **and**
‹*pos* ≤ *length* (*N* ∝ *i*) − *2*›
shows ‹*xarena-active-clause* (*update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
⟨*proof*⟩

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

lemma *valid-arena-update-pos*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)›**and**
pos: ‹*pos* ≤ *length* (*N* ∝ *i*) − *2*›
shows ‹*valid-arena* (*update-pos-direct i pos arena*) *N vdom*›
⟨*proof*⟩

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*⟩

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

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

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

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

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

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

Learning a clause

definition *append-clause-skeleton* **where**
‹*append-clause-skeleton pos st used lbd C arena* =
(*if is-short-clause C then*
arena @ (*AStatus st used lbd*) #
ASize (*length C* − *2*) # *map ALit C*
else arena @ *APos pos* # (*AStatus st used 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*) *0* (*shorten-lbd*(*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 lbd C arena*) *N i*)
(*the* (*fmlookup N i*))›
⟨*proof*⟩

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

lemma *arena-active-clause-append-clause-same*: ‹*2* ≤ *length C* ⟹ *st* ≠ *DELETED* ⟹
pos ≤ *length C* − *2* ⟹
b ⟷ (*st* = *IRRED*) ⟹
xarena-active-clause
(*Misc.slice* (*length arena*) (*length arena* + *header-size C* + *length C*)
(*append-clause-skeleton pos st used lbd C arena*))
(*the* (*fmlookup* (*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*length arena* + *header-size C*)))›
⟨*proof*⟩

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

lemma *valid-arena-append-clause-skeleton*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ *2*› **and**
b: ‹*b* ⟷ (*st* = *IRRED*)› **and** *st*: ‹*st* ≠ *DELETED*› **and**
pos: ‹*pos* ≤ *length C* − *2*›
shows ‹*valid-arena* (*append-clause-skeleton pos st used lbd C arena*)
(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*insert* (*length arena* + *header-size C*) *vdom*)›
⟨*proof*⟩

lemma *valid-arena-append-clause*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ *2*›
shows ‹*valid-arena* (*append-clause b C arena*)
(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*insert* (*length arena* + *header-size C*) *vdom*)›
⟨*proof*⟩

Refinement Relation

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

definition *bitfield-rel* **where**

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

definition arena-el-relation where
‹arena-el-relation x el  = (case el of
     AStatus n b lbd ⇒ (x AND 0b11, n) ∈ status-rel ∧ ((x AND 0b1100) >> 2, b) ∈ nat-rel ∧ (x >>
5, lbd) ∈ nat-rel
   | APos n ⇒ (x, n) ∈ nat-rel
   | ASize 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
    ‹True› and
    ‹is-Status (arena ! (i − STATUS-SHIFT))› and
    ‹SIZE-SHIFT ≤ i› and
    ‹LBD-SHIFT ≤ i›
    ‹True› and
    ‹arena-length arena i ≥ 2› and
    ‹arena-length arena i ≥ Suc 0› and
    ‹arena-length arena i ≥ 0› and
    ‹arena-length arena i > Suc 0› and
    ‹arena-length arena i > 0› and
    ‹arena-status arena i = LEARNED ⟷ ¬irred N i› and
    ‹arena-status arena i = IRRED ⟷ irred N i› and
```

‹*arena-status arena i* ≠ *DELETED*› **and**
‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* (*N* ∝ *i*)›
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

definition *arena-lit-pre2* **where**
‹*arena-lit-pre2 arena i j* ⟷
(∃ *N vdom. valid-arena arena N vdom* ∧ *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*))›

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

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

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

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

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

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

definition *mark-used-raw* **where**
‹*mark-used-raw arena i v* =
arena[*i* − *STATUS-SHIFT* := *AStatus* (*arena-status arena i*) ((*arena-used arena i*) *OR v*) (*arena-lbd arena i*)]›

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

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

definition *mark-unused* **where**
‹*mark-unused arena i* =
arena[*i* − *STATUS-SHIFT* := *AStatus* (*xarena-status* (*arena*!(*i* − *STATUS-SHIFT*)))
(*if* (*arena-used arena i*) > *0 then arena-used arena i* − *1 else 0*)
(*arena-lbd arena i*)]›

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

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*⟩

definition *marked-as-used* :: ‹*arena* ⇒ *nat* ⇒ *nat*› **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*›

lemma *valid-arena-vdom-le*:
assumes ‹*valid-arena arena N ovdm*›
shows ‹*finite ovdm*› **and** ‹*card ovdm ≤ length arena*›
⟨*proof*⟩

lemma *valid-arena-vdom-subset*:
assumes ‹*valid-arena arena N* (*set vdom*)› **and** ‹*distinct vdom*›
shows ‹*length vdom ≤ length arena*›
⟨*proof*⟩

2.4 MOP versions of operations

2.4.1 Access to literals

definition *mop-arena-lit* **where**
‹*mop-arena-lit arena s = do* {
    ASSERT(*arena-lit-pre arena s*);
    RETURN (*arena-lit arena s*)
}›

lemma *arena-lit-pre-le-lengthD*: ‹*arena-lit-pre arena C ⟹ C < length arena*›
⟨*proof*⟩

definition *mop-arena-lit2* :: ‹*arena ⇒ nat ⇒ nat ⇒ nat literal nres*› **where**
‹*mop-arena-lit2 arena i j = do* {
ASSERT(*arena-lit-pre arena* (*i+j*));
let s = i+j;
RETURN (*arena-lit arena s*)
}›

named-theorems *mop-arena-lit* ‹*Theorems on mop−forms of arena constants*›

lemma *mop-arena-lit-itself*:
‹*mop-arena-lit arena k′ ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*) *⟹ mop-arena-lit arena k′ ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*)›
‹*mop-arena-lit2 arena i′ k′ ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*) *⟹ mop-arena-lit2 arena i′ k′ ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*)›
⟨*proof*⟩

lemma [*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹*i ∈# dom-m N*›
shows
‹*k = i+j ⟹ j < length* (*N ∝ i*) *⟹ mop-arena-lit arena k ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*)›
‹*i=i′ ⟹ j=j′ ⟹j < length* (*N ∝ i*) *⟹ mop-arena-lit2 arena i′ j′ ≤ SPEC*(*λc.* (*c*, *N ∝ i*!*j*) *∈ Id*)›
⟨*proof*⟩

lemma *mop-arena-lit2*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**

i: ‹$(C,\ C') \in$ *nat-rel*› ‹$(i,\ i') \in$ *nat-rel*›
shows
‹*mop-arena-lit2 arena C i* $\le \Downarrow$*Id* (*mop-clauses-at N C′ i′*)›
⟨*proof*⟩

definition *mop-arena-lit2′* :: ‹*nat set* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*mop-arena-lit2′ vdom* = *mop-arena-lit2*›

lemma *mop-arena-lit2′*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹$(C,\ C') \in$ *nat-rel*› ‹$(i,\ i') \in$ *nat-rel*›
shows
‹*mop-arena-lit2′ vdom arena C i* $\le \Downarrow$*Id* (*mop-clauses-at N C′ i′*)›
⟨*proof*⟩

lemma *arena-lit-pre2-arena-lit*[*dest*]:
‹*arena-lit-pre2 N i j* ⟹ *arena-lit-pre N* (*i*+*j*)›
⟨*proof*⟩

2.4.2 Swapping of literals

definition *mop-arena-swap* **where**
‹*mop-arena-swap C i j arena* = *do* {
ASSERT(*swap-lits-pre C i j arena*);
RETURN (*swap-lits C i j arena*)
}›

lemma *mop-arena-swap*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹$(C,\ C') \in$ *nat-rel*› ‹$(i,\ i') \in$ *nat-rel*› ‹$(j,\ j') \in$ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* $\le \Downarrow\{(N',\ N).$ *valid-arena N′ N vdom*$\}$ (*mop-clauses-swap N C′ i′ j′*)›
⟨*proof*⟩

2.4.3 Position Saving

definition *mop-arena-pos* :: ‹*arena* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-pos arena C* = *do* {
ASSERT(*get-saved-pos-pre arena C*);
RETURN (*arena-pos arena C*)
}›

definition *mop-arena-length* :: ‹*arena-el list* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length arena C* = *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
RETURN (*arena-length arena C*)
}›

2.4.4 Clause length

lemma *mop-arena-length*:
‹(*uncurry mop-arena-length*, *uncurry* (*RETURN oo* ($\lambda N\ c.$ *length* ($N \propto c$)))) ∈
$[\lambda(N,\ i).\ i \in\#$ *dom-m N*$]_f$ $\{(N,\ N').$ *valid-arena N N′ vdom*$\}$ $\times_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

```
definition mop-arena-lbd where
  ‹mop-arena-lbd arena C = do {
    ASSERT(get-clause-LBD-pre arena C);
    RETURN(arena-lbd arena C)
  }›

definition mop-arena-update-lbd where
  ‹mop-arena-update-lbd C glue arena = do {
    ASSERT(update-lbd-pre ((C, glue), arena));
    RETURN(update-lbd C glue arena)
  }›

definition mop-arena-status where
  ‹mop-arena-status arena C = do {
    ASSERT(arena-is-valid-clause-vdom arena C);
    RETURN(arena-status arena C)
  }›

definition mop-marked-as-used where
  ‹mop-marked-as-used arena C = do {
    ASSERT(marked-as-used-pre arena C);
    RETURN(marked-as-used arena C)
  }›

definition arena-other-watched :: ‹arena ⇒ nat literal ⇒ nat ⇒ nat ⇒ nat literal nres› where
‹arena-other-watched S L C i = do {
    ASSERT(i < 2 ∧ arena-lit S (C + i) = L ∧ arena-lit-pre2 S C i ∧
      arena-lit-pre2 S C (1−i));
    mop-arena-lit2 S C (1 − i)
  }›

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

definition mark-used :: ‹arena ⇒ nat ⇒ arena› where
  mark-used-int-def: ‹mark-used arena C ≡ mark-used-raw arena C 1›

lemmas mark-used-def = mark-used-int-def[unfolded mark-used-raw-def]

lemmas length-mark-used[simp] =
  length-mark-used-raw[of - - 1, unfolded mark-used-int-def[symmetric]]

lemmas valid-arena-mark-used =
   valid-arena-mark-used-raw[of - - - - 1, unfolded mark-used-int-def[symmetric]]

definition mark-used2 :: ‹arena ⇒ nat ⇒ arena› where
  mark-used2-int-def: ‹mark-used2 arena C ≡ mark-used-raw arena C 2›

lemmas mark-used2-def = mark-used2-int-def[unfolded mark-used-raw-def]

lemmas length-mark-used2[simp] =
  length-mark-used-raw[of - - 2, unfolded mark-used2-int-def[symmetric]]

lemmas valid-arena-mark-used2 =
   valid-arena-mark-used-raw[of - - - - 2, unfolded mark-used2-int-def[symmetric]]
```

```
definition mop-arena-mark-used where
  ‹mop-arena-mark-used C arena = do {
    ASSERT(arena-act-pre C arena);
    RETURN (mark-used C arena)
  }›

definition mop-arena-mark-used2 where
  ‹mop-arena-mark-used2 C arena = do {
    ASSERT(arena-act-pre C arena);
    RETURN (mark-used2 C arena)
  }›

end
theory WB-More-Word
  imports HOL− Word.More-Word Isabelle-LLVM.Bits-Natural
begin

lemma nat-uint-XOR: ‹nat (uint (a XOR b)) = nat (uint a) XOR nat (uint b)›
  if len: ‹LENGTH('a) > 0›
  for a b :: ‹'a ::len0 Word.word›
⟨proof⟩
lemma bitXOR-1-if-mod-2-int: ‹bitOR L 1 = (if L mod 2 = 0 then L + 1 else L)› for L :: int
  ⟨proof⟩


lemma bitOR-1-if-mod-2-nat:
  ‹bitOR L 1 = (if L mod 2 = 0 then L + 1 else L)›
  ‹bitOR L (Suc 0) = (if L mod 2 = 0 then L + 1 else L)› for L :: nat
⟨proof⟩

lemma bin-pos-same-XOR3:
  ‹a XOR a XOR c = c›
  ‹a XOR c XOR a = c› for a c :: int
  ⟨proof⟩

lemma bin-pos-same-XOR3-nat:
  ‹a XOR a XOR c = c›
  ‹a XOR c XOR a = c› for a c :: nat
 ⟨proof⟩

end
theory IsaSAT-Literals-LLVM
  imports WB-More-Word IsaSAT-Literals Watched-Literals.WB-More-IICF-LLVM
begin


lemma inline-ho[llvm-inline]: ‹doM { f ← return f; m f } = m f› for f :: ‹- ⇒ -› ⟨proof⟩




lemma RETURN-comp-5-10-hnr-post[to-hnr-post]:
  ‹(RETURN ooooo f5)$a$b$c$d$e = RETURN$(f5$a$b$c$d$e)›
  ‹(RETURN oooooo f6)$a$b$c$d$e$f = RETURN$(f6$a$b$c$d$e$f)›
  ‹(RETURN ooooooo f7)$a$b$c$d$e$f$g = RETURN$(f7$a$b$c$d$e$f$g)›
  ‹(RETURN oooooooo f8)$a$b$c$d$e$f$g$h = RETURN$(f8$a$b$c$d$e$f$g$h)›
```

‹(*RETURN ooooooooo f9*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i* = *RETURN*$(*f9*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*)›
‹(*RETURN oooooooooo f10*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j* = *RETURN*$(*f10*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*)›
‹(*RETURN* o_{11} *f11*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k* = *RETURN*$(*f11*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*)›
‹(*RETURN* o_{12} *f12*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l* = *RETURN*$(*f12*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l*)›
‹(*RETURN* o_{13} *f13*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l*$*m* = *RETURN*$(*f13*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l*$*m*)›
‹(*RETURN* o_{14} *f14*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l*$*m*$*n* = *RETURN*$(*f14*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*$*j*$*k*$*l*$*m*$*n*)›
⟨*proof*⟩

definition [*simp*,*llvm-inline*]: ‹*case-prod-open* ≡ *case-prod*›
lemmas *fold-case-prod-open* = *case-prod-open-def*[*symmetric*]

lemma *case-prod-open-arity*[*sepref-monadify-arity*]:
‹*case-prod-open* ≡ λ_2*fp p. SP case-prod-open*$($\lambda_2$*a b. fp*$*a*$*b*)$*p*›
⟨*proof*⟩

lemma *case-prod-open-comb*[*sepref-monadify-comb*]:
‹⋀*fp p. case-prod-open*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$($\lambda_2$*p.* (*SP case-prod-open*$*fp*$*p*))›
⟨*proof*⟩

lemma *case-prod-open-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-prod-open*$(λ_2*a b. fp a b*)$*p*) ≡
Refine-Basic.bind$(*EVAL*$*p*)$($\lambda_2$*p. case-prod-open*$(λ_2*a b. EVAL*$(*fp a b*))$*p*)
⟨*proof*⟩

lemma *hn-case-prod-open'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*prod-assn P1 P2*) *p' p* ∗∗ Γ*1*›
assumes *Pair*: ⋀*a1 a2 a1' a2'*. ⟦*p'*=(*a1'*,*a2'*)⟧
⟹ *hn-refine* (*hn-ctxt P1 a1' a1* ∗∗ *hn-ctxt P2 a2' a2* ∗∗ Γ*1*) (*f a1 a2*)
(Γ*2 a1 a2 a1' a2'*) *R* (*f' a1' a2'*)
assumes *FR2*: ‹⋀*a1 a2 a1' a2'*. Γ*2 a1 a2 a1' a2'* ⊢ *hn-ctxt P1' a1' a1* ∗∗ *hn-ctxt P2' a2' a2* ∗∗ Γ*1'*›
shows ‹*hn-refine* Γ (*case-prod-open f p*) (*hn-ctxt* (*prod-assn P1' P2'*) *p' p* ∗∗ Γ*1'*)
R (*case-prod-open*$($\lambda_2$*a b. f' a b*)$*p'*)› (**is** ‹*?G* Γ›)
⟨*proof*⟩
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *prod-assn-pair-conv*[*THEN prod-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

lemma *ho-prod-open-move*[*sepref-preproc*]: ‹*case-prod-open* (λ*a b x. f x a b*) = (λ*p x. case-prod-open* (*f x*) *p*)›
⟨*proof*⟩

definition ‹*tuple4 a b c d* ≡ (*a*,*b*,*c*,*d*)›
definition ‹*tuple7 a b c d e f g* ≡ *tuple4 a b c* (*tuple4 d e f g*)›
definition ‹*tuple13 a b c d e f g h i j k l m* ≡ (*tuple7 a b c d e f* (*tuple7 g h i j k l m*))›

lemmas *fold-tuples* = *tuple4-def*[*symmetric*] *tuple7-def*[*symmetric*] *tuple13-def*[*symmetric*]

sepref-register *tuple4 tuple7 tuple13*

sepref-def *tuple4-impl* [*llvm-inline*] **is** ‹*uncurry3* (*RETURN oooo tuple4*)› ::

‹$A1^d *_a A2^d *_a A3^d *_a A4^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4$›
⟨*proof*⟩

sepref-def *tuple7-impl* [*llvm-inline*] **is** ‹*uncurry6* (*RETURN ooooooo tuple7*)› ::
‹$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7$›
⟨*proof*⟩

sepref-def *tuple13-impl* [*llvm-inline*] **is** ‹*uncurry12* (*RETURN* o_{13} *tuple13*)› ::
$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d *_a A8^d *_a A9^d *_a A10^d *_a A11^d *_a A12^d *_a A13^d$
$\rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7 \times_a A8 \times_a A9 \times_a A10 \times_a A11 \times_a A12 \times_a A13$
⟨*proof*⟩

lemmas *fold-tuple-optimizations* = *fold-tuples fold-case-prod-open*

lemma *sint64-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFFFFFFFFFF*, *sint64-max*)∈*snat-rel′ TYPE*(*64*)›
⟨*proof*⟩

lemma *sint32-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFF*, *sint32-max*)∈*snat-rel′ TYPE*(*32*)›
⟨*proof*⟩

lemma *uint64-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFFFFFFFFFF*, *uint64-max*)∈*unat-rel′ TYPE*(*64*)›
⟨*proof*⟩

lemma *uint32-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFF*, *uint32-max*)∈*unat-rel′ TYPE*(*32*)›
⟨*proof*⟩

lemma *convert-fref*:
‹*WB-More-Refinement.fref* = *Sepref-Rules.frefnd*›
‹*WB-More-Refinement.freft* = *Sepref-Rules.freftnd*›
⟨*proof*⟩

no-notation *WB-More-Refinement.fref* (‹$[-]_f$ - → -› [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (‹- $\rightarrow_f$ -› [*60*,*60*] *60*)

abbreviation ‹*uint32-nat-assn* ≡ *unat-assn′ TYPE*(*32*)›
abbreviation ‹*uint64-nat-assn* ≡ *unat-assn′ TYPE*(*64*)›

abbreviation ‹*sint32-nat-assn* ≡ *snat-assn′ TYPE*(*32*)›
abbreviation ‹*sint64-nat-assn* ≡ *snat-assn′ TYPE*(*64*)›

lemmas [*sepref-bounds-simps*] =
uint32-max-def sint32-max-def

uint64-max-def sint64-max-def

lemma *is-up'-32-64*[*simp,intro*!]: ‹*is-up' UCAST*(*32* → *64*)› ⟨*proof*⟩
lemma *is-down'-64-32*[*simp,intro*!]: ‹*is-down' UCAST*(*64* → *32*)› ⟨*proof*⟩

lemma *ins-idx-upcast64*:
‹*l*[*i*:=*y*] = *op-list-set l* (*op-unat-snat-upcast TYPE*(*64*) *i*) *y*›
‹*l*!*i* = *op-list-get l* (*op-unat-snat-upcast TYPE*(*64*) *i*)›
⟨*proof*⟩

type-synonym *'a array-list32* = ‹(*'a*,*32*)*array-list*›
type-synonym *'a array-list64* = ‹(*'a*,*64*)*array-list*›

abbreviation ‹*arl32-assn* ≡ *al-assn' TYPE*(*32*)›
abbreviation ‹*arl64-assn* ≡ *al-assn' TYPE*(*64*)›

type-synonym *'a larray32* = ‹(*'a*,*32*) *larray*›
type-synonym *'a larray64* = ‹(*'a*,*64*) *larray*›

abbreviation ‹*larray32-assn* ≡ *larray-assn' TYPE*(*32*)›
abbreviation ‹*larray64-assn* ≡ *larray-assn' TYPE*(*64*)›

definition ‹*unat-lit-rel* == *unat-rel' TYPE*(*32*) *O nat-lit-rel*›
lemmas [*fcomp-norm-unfold*] = *unat-lit-rel-def*[*symmetric*]

abbreviation *unat-lit-assn* :: ‹*nat literal* ⇒ *32 word* ⇒ *assn*› **where**
‹*unat-lit-assn* ≡ *pure unat-lit-rel*›

2.4.5 Atom-Of

type-synonym *atom-assn* = ‹*32 word*›

definition ‹*atom-rel* ≡ *b-rel* (*unat-rel' TYPE*(*32*)) (λ*x*. *x*<*2*^*31*)›
abbreviation ‹*atom-assn* ≡ *pure atom-rel*›

lemma *atom-rel-alt*: ‹*atom-rel* = *unat-rel' TYPE*(*32*) *O nbn-rel* (*2*^*31*)›
⟨*proof*⟩

interpretation *atom*: *dflt-pure-option-private* ‹*2*^*32*−*1*› *atom-assn* ‹*ll-icmp-eq* (*2*^*32*−*1*)›
⟨*proof*⟩

lemma *atm-of-refine*: ‹(λ*x*. *x div 2* , *atm-of*) ∈ *nat-lit-rel* → *nat-rel*›
⟨*proof*⟩

sepref-def *atm-of-impl* **is** [] ‹*RETURN o* (λ*x*::*nat*. *x div 2*)›
:: ‹$uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *atom-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *atm-of-impl.refine*[*FCOMP atm-of-refine*]

definition *Pos-rel* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*Pos-rel n* = *2* ∗ *n*›

lemma *Pos-refine-aux*: ‹(*Pos-rel*,*Pos*)∈*nat-rel* → *nat-lit-rel*›
⟨*proof*⟩

lemma *Neg-refine-aux*: ‹(λ*x*. *2*∗*x* + *1*,*Neg*)∈*nat-rel* → *nat-lit-rel*›
⟨*proof*⟩

sepref-def *Pos-impl* **is** [] ‹*RETURN o Pos-rel*› :: ‹*atom-assn*^{d} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *Neg-impl* **is** [] ‹*RETURN o* (λ*x*. *2*∗*x*+*1*)› :: ‹*atom-assn*^{d} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] =
Pos-impl.*refine*[*FCOMP Pos-refine-aux*]
Neg-impl.*refine*[*FCOMP Neg-refine-aux*]

sepref-def *atom-eq-impl* **is** ‹*uncurry* (*RETURN oo* (=))› :: ‹*atom-assn*^{d} $*_a$ *atom-assn*^{d} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

definition *value-of-atm* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*value-of-atm A* = *A*›

lemma *value-of-atm-rel*: ‹(λ*x*. *x*, *value-of-atm*) ∈ *nat-rel* → *nat-rel*›
⟨*proof*⟩

sepref-def *value-of-atm-impl*
is [] ‹*RETURN o* (λ*x*. *x*)›
:: ‹*atom-assn*^{d} $\rightarrow_a$ *unat-assn*′ *TYPE*(*32*)›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *value-of-atm-impl*.*refine*[*FCOMP value-of-atm-rel*]

definition *index-of-atm* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*index-of-atm A* = *value-of-atm A*›

lemma *index-of-atm-rel*: ‹(λ*x*. *value-of-atm x*, *index-of-atm*) ∈ *nat-rel* → *nat-rel*›
⟨*proof*⟩

sepref-def *index-of-atm-impl*
is [] ‹*RETURN o* (λ*x*. *value-of-atm x*)›
:: ‹*atom-assn*^{d} $\rightarrow_a$ *snat-assn*′ *TYPE*(*64*)›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *index-of-atm-impl*.*refine*[*FCOMP index-of-atm-rel*]

lemma *annot-index-of-atm*: ‹*xs* ! *x* = *xs* ! *index-of-atm x*›
‹*xs* [*x* := *a*] = *xs* [*index-of-atm x* := *a*]›
⟨*proof*⟩

definition *index-atm-of* **where**
[*simp*]: ‹*index-atm-of L* = *index-of-atm* (*atm-of L*)›

context fixes *x y* :: *nat* **assumes** ‹*NO-MATCH* (*index-of-atm y*) *x*› **begin**
  lemmas *annot-index-of-atm′* = *annot-index-of-atm*[**where** *x*=*x*]
end

method-setup *annot-all-atm-idxs* = ‹*Scan.succeed* (*fn ctxt* => *SIMPLE-METHOD′*
    let
      val ctxt = *put-simpset HOL-basic-ss ctxt*
      val ctxt = *ctxt addsimps* @{*thms annot-index-of-atm′*}
      val ctxt = *ctxt addsimprocs* [@{*simproc NO-MATCH*}]
    in
      simp-tac ctxt
    end
)›

lemma *annot-index-atm-of*[*def-pat-rules*]:
  ‹*nth*\$*xs*\$(*atm-of*\$*x*) ≡ *nth*\$*xs*\$(*index-atm-of*\$*x*)›
  ‹*list-update*\$*xs*\$(*atm-of*\$*x*)\$*a* ≡ *list-update*\$*xs*\$(*index-atm-of*\$*x*)\$*a*›
  ⟨*proof*⟩

sepref-def *index-atm-of-impl*
  is ‹*RETURN o index-atm-of*›
  :: ‹*unat-lit-assn*^{d} $\rightarrow_a$ *snat-assn′ TYPE*(*64*)›
  ⟨*proof*⟩

lemma *nat-of-lit-refine-aux*: ‹((λ*x*. *x*), *nat-of-lit*) ∈ *nat-lit-rel* → *nat-rel*›
  ⟨*proof*⟩

sepref-def *nat-of-lit-rel-impl* **is** [] ‹*RETURN o* (λ*x*::*nat*. *x*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
  ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *nat-of-lit-rel-impl.refine*[*FCOMP nat-of-lit-refine-aux*]

lemma *uminus-refine-aux*: ‹(λ*x*. *x XOR 1*, *uminus*) ∈ *nat-lit-rel* → *nat-lit-rel*›
  ⟨*proof*⟩

sepref-def *uminus-impl* **is** [] ‹*RETURN o* (λ*x*::*nat*. *x XOR 1*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *uminus-impl.refine*[*FCOMP uminus-refine-aux*]

lemma *lit-eq-refine-aux*: ‹((=), (=)) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
  ⟨*proof*⟩

sepref-def *lit-eq-impl* **is** [] ‹*uncurry* (*RETURN oo* (=))› :: ‹*uint32-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *lit-eq-impl.refine*[*FCOMP lit-eq-refine-aux*]

lemma *is-pos-refine-aux*: ‹(λx. x AND 1 = 0, is-pos) ∈ nat-lit-rel → bool-rel›
⟨*proof*⟩

sepref-def *is-pos-impl* **is** [] ‹RETURN o (λx. x AND 1 = 0)› :: ‹$uint32\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *is-pos-impl.refine*[*FCOMP is-pos-refine-aux*]

sepref-decl-op *nat-lit-eq*: ‹(=) :: nat literal ⇒ - ⇒ -› ::
‹(Id :: (nat literal × -) set) → (Id :: (nat literal × -) set) → bool-rel› ⟨*proof*⟩

sepref-def *nat-lit-eq-impl*
is [] ‹uncurry (RETURN oo (λx y. x = y))›
:: ‹$uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

lemma *nat-lit-rel*: ‹((=), op-nat-lit-eq) ∈ nat-lit-rel → nat-lit-rel → bool-rel›
⟨*proof*⟩

sepref-register ‹(=) :: nat literal ⇒ - ⇒ -›
declare *nat-lit-eq-impl.refine*[*FCOMP nat-lit-rel*, *sepref-fr-rules*]

end
theory *IsaSAT-Arena-LLVM*
imports *IsaSAT-Arena IsaSAT-Literals-LLVM Watched-Literals.WB-More-IICF-LLVM*
begin

2.5 Code Generation

no-notation *WB-More-Refinement.fref* (‹[-]$_f$ - → -› [0,60,60] 60)
no-notation *WB-More-Refinement.freft* (‹- →$_f$ -› [60,60] 60)

lemma *protected-bind-assoc*: ‹Refine-Basic.bind\$(Refine-Basic.bind\$m\$(λ$_2$x. f x))\$(λ$_2$y. g y) = Refine-Basic.bind\$m\$(λ$_2$
Refine-Basic.bind\$(f x)\$(λ$_2$y. g y))› ⟨*proof*⟩

lemma *convert-swap*: ‹ WB-More-Refinement-List.swap = More-List.swap›
⟨*proof*⟩

Code Generation

definition ‹arena-el-impl-rel ≡ unat-rel′ TYPE(32) O arena-el-rel›
lemmas [*fcomp-norm-unfold*] = *arena-el-impl-rel-def*[*symmetric*]
abbreviation ‹arena-el-impl-assn ≡ pure arena-el-impl-rel›

Arena Element Operations **context**

notes [*simp*] = *arena-el-rel-def*
notes [*split*] = *arena-el.splits*
notes [*intro*!] = *frefI*
begin

Literal

lemma *xarena-lit-refine1*: ‹(λeli. eli, xarena-lit) ∈ [is-Lit]$_f$ arena-el-rel → nat-lit-rel› ⟨*proof*⟩

sepref-def *xarena-lit-impl* [*llvm-inline*]
  is [] ‹*RETURN o* (λ*eli. eli*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *xarena-lit-impl.refine*[*FCOMP xarena-lit-refine1*]

lemma *ALit-refine1*: ‹(λ*x. x*,*ALit*) ∈ *nat-lit-rel* → *arena-el-rel*› ⟨*proof*⟩
sepref-def *ALit-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x. x*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *ALit-impl.refine*[*FCOMP ALit-refine1*]

LBD

lemma *xarena-lbd-refine1*: ‹(λ*eli. eli* >> *5*, *xarena-lbd*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *nat-rel*›
  ⟨*proof*⟩

sepref-def *xarena-lbd-impl* [*llvm-inline*]
  is [] ‹(*RETURN o* (λ*eli. eli* >> *5*))› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *xarena-lbd-impl.refine*[*FCOMP xarena-lbd-refine1*]

Size

lemma *xarena-length-refine1*: ‹(λ*eli. eli*, *xarena-length*) ∈ [*is-Size*]$_f$ *arena-el-rel* → *nat-rel*› ⟨*proof*⟩
sepref-def *xarena-len-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli. eli*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *xarena-len-impl.refine*[*FCOMP xarena-length-refine1*]

lemma *ASize-refine1*: ‹(λ*x. x*,*ASize*) ∈ *nat-rel* → *arena-el-rel*› ⟨*proof*⟩
sepref-def *ASize-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x. x*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *ASize-impl.refine*[*FCOMP ASize-refine1*]

Position

lemma *xarena-pos-refine1*: ‹(λ*eli. eli*, *xarena-pos*) ∈ [*is-Pos*]$_f$ *arena-el-rel* → *nat-rel*› ⟨*proof*⟩
sepref-def *xarena-pos-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli. eli*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *xarena-pos-impl.refine*[*FCOMP xarena-pos-refine1*]

lemma *APos-refine1*: ‹(λ*x. x*,*APos*) ∈ *nat-rel* → *arena-el-rel*› ⟨*proof*⟩
sepref-def *APos-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x. x*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *APos-impl.refine*[*FCOMP APos-refine1*]

Status

definition ‹*status-impl-rel* ≡ *unat-rel′ TYPE*(*32*) *O status-rel*›
lemmas [*fcomp-norm-unfold*] = *status-impl-rel-def*[*symmetric*]
abbreviation ‹*status-impl-assn* ≡ *pure status-impl-rel*›

lemma *xarena-status-refine1*: ‹(λ*eli. eli AND 0b11*, *xarena-status*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *status-rel*› ⟨*proof*⟩
sepref-def *xarena-status-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli. eli AND 0b11*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *xarena-status-impl.refine*[*FCOMP xarena-status-refine1*]

lemma *xarena-used-refine1*: ‹(λ*eli.* (*eli AND 0b1100*) >> *2*, *xarena-used*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *nat-rel*›

⟨*proof*⟩

lemma *is-down'-32-2*[*simp*]: ‹*is-down' UCAST*(*32* → *2*)›
⟨*proof*⟩

lemma *bitAND-mod*: ‹*bitAND L* (*2*^*n* − *1*) = *L mod* (*2*^*n*)› **for** *L* :: *nat*
⟨*proof*⟩

lemma *nat-ex-numeral*: ‹$\exists$ *m*. *n*=*0* ∨ *n* = *numeral m*› **for** *n* :: *nat*
⟨*proof*⟩

lemma *xarena-used-implI*: ‹*x AND 12* >> *2* < *max-unat 2*› **for** *x* :: *nat*
⟨*proof*⟩

sepref-def *xarena-used-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli*.(*eli AND 0b1100*) >> *2*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *unat-assn' TYPE*(*2*)›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *xarena-used-impl.refine*[*FCOMP xarena-used-refine1*]

lemma *status-eq-refine1*: ‹((=),(=)) ∈ *status-rel* → *status-rel* → *bool-rel*›
⟨*proof*⟩

sepref-def *status-eq-impl* [*llvm-inline*] **is** [] ‹*uncurry* (*RETURN oo* (=))›
:: ‹(*unat-assn' TYPE*(*32*))^k $*_a$ (*unat-assn' TYPE*(*32*))^k →$_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *status-eq-impl.refine*[*FCOMP status-eq-refine1*]

definition ‹*AStatus-impl1 cs used lbd* ≡
(*cs AND unat-const TYPE*(*32*) *0b11*) + (*used* << *2*) + (*lbd* << *unat-const TYPE*(*32*) *5*)›

lemma *bang-eq-int*:
fixes *x* :: *int*
shows (*x* = *y*) = (∀ *n*. *x* !! *n* = *y* !! *n*)
⟨*proof*⟩

lemma *bang-eq-nat*:
fixes *x* :: *nat*
shows (*x* = *y*) = (∀ *n*. *x* !! *n* = *y* !! *n*)
⟨*proof*⟩

lemma *sum-bitAND-shift-pow2*:
‹(*a* + (*b* << (*n* + *m*))) *AND* (*2*^*n* − *1*) = *a AND* (*2*^*n* − *1*)› **for** *a b n* :: *nat*
⟨*proof*⟩

lemma *and-bang-nat*: ‹(*x AND y*) !! *n* = (*x* !! *n* ∧ *y* !! *n*)› **for** *x y n* :: *nat*
⟨*proof*⟩

lemma *AND-12-AND-15-AND-12*: ‹*a AND 12* = (*a AND 15*) *AND 12*› **for** *a* :: *nat*
⟨*proof*⟩

lemma *AStatus-shift-safe*:
‹*c* ≥ *2* ⟹ *x42* + (*x43* << *c*) *AND 3* = *x42 AND 3*›

$\langle (x53 << 2)\ AND\ 3 = 0 \rangle$
$\langle x42 + (x43 << 4)\ AND\ 12 = x42\ AND\ 12 \rangle$
$\langle x42 + (x43 << 5)\ AND\ 12 = x42\ AND\ 12 \rangle$
$\langle Suc\ (x42 + (x43 << 5))\ AND\ 12 = (Suc\ x42)\ AND\ 12 \rangle$
$\langle Suc\ ((x42) + (x43 << 5))\ AND\ 3 = Suc\ x42\ AND\ 3 \rangle$
$\langle Suc\ (x42 << 2)\ AND\ 3 = Suc\ 0 \rangle$
$\langle x42 \leq 3 \Longrightarrow Suc\ ((x42 << 2) + (x43 << 5)) >> 5 = x43 \rangle$
for $x42\ x43\ x53 :: nat$
$\langle proof \rangle$

lemma *less-unat-AND-shift*: $\langle x42 < 2\hat{\ }n \Longrightarrow x42 >> n = 0 \rangle$ **for** $x42 :: nat$
$\langle proof \rangle$

lemma [*simp*]: $\langle (a + (w << n)) >> n = (a >> n) + w \rangle$ $\langle ((w << n)) >> n = w \rangle$
$\langle n \leq m \Longrightarrow ((w << n)) >> m = w >> (m - n) \rangle$
$\langle n \geq m \Longrightarrow ((w << n)) >> m = w << (n - m) \rangle$ **for** $w\ n :: nat$
$\langle proof \rangle$

lemma *less-numeral-pred*:
$\langle a \leq numeral\ b \longleftrightarrow a = numeral\ b \vee a \leq pred\text{-}numeral\ b \rangle$ **for** $a :: nat$
$\langle proof \rangle$

lemma *nat-shiftl-numeral* [*simp*]:
$(numeral\ w :: nat) << numeral\ w' = numeral\ (num.Bit0\ w) << pred\text{-}numeral\ w'$
$\langle proof \rangle$

lemma *nat-shiftl-numeral'* [*simp*]:
$(numeral\ w :: nat) << 1 = numeral\ (num.Bit0\ w)$
$(1 :: nat) << n = 2\ \hat{\ }\ n$
$\langle proof \rangle$

lemma *shiftr-nat-alt-def*: $\langle (a :: nat) >> b = nat\ (int\ a >> b) \rangle$
$\langle proof \rangle$

lemma *nat-shiftr-numeral* [*simp*]:
$(1 :: nat) >> numeral\ w' = 0$
$(numeral\ num.One :: nat) >> numeral\ w' = 0$
$(numeral\ (num.Bit0\ w) :: nat) >> numeral\ w' = numeral\ w >> pred\text{-}numeral\ w'$
$(numeral\ (num.Bit1\ w) :: nat) >> numeral\ w' = numeral\ w >> pred\text{-}numeral\ w'$
$\langle proof \rangle$

lemma *nat-shiftr-numeral-Suc0* [*simp*]:
$(1 :: nat) >> Suc\ 0 = 0$
$(numeral\ num.One :: nat) >> Suc\ 0 = 0$
$(numeral\ (num.Bit0\ w) :: nat) >> Suc\ 0 = numeral\ w$
$(numeral\ (num.Bit1\ w) :: nat) >> Suc\ 0 = numeral\ w$
$\langle proof \rangle$

lemma *nat-shiftr-numeral1* [*simp*]:
$(1 :: nat) >> 1 = 0$
$(numeral\ num.One :: nat) >> 1 = 0$
$(numeral\ (num.Bit0\ w) :: nat) >> 1 = numeral\ w$
$(numeral\ (num.Bit1\ w) :: nat) >> 1 = numeral\ w$
$\langle proof \rangle$

lemma *nat-numeral-and-one*: ‹*(1 :: nat) AND 1 = 1*›
  ⟨*proof*⟩

lemma *AStatus-refine1*: ‹*(AStatus-impl1, AStatus) ∈ status-rel → br id (λn. n ≤ 3) → nat-rel → arena-el-rel*›
  ⟨*proof*⟩

lemma *AStatus-implI*:
  assumes ‹*b << 5 < max-unat 32*›
  shows ‹*b << 5 < max-unat 32 − 7*› ‹*(a AND 3) + 4 + (b << 5) < max-unat 32*›
    ‹*(a AND 3) + (b << 5) < max-unat 32*›
⟨*proof*⟩

lemma *nat-shiftr-mono*: ‹*a < b ⟹ a << n < b << n*› **for** *a b :: nat*
  ⟨*proof*⟩

lemma *AStatus-implI3*:
  assumes ‹*(ac :: 2 word, ba) ∈ unat-rel*›
  shows ‹*(a AND (3::nat)) + (ba << (2::nat)) < max-unat (32::nat)*› **and**
    ‹*b << 5 < max-unat 32 ⟹ (a AND 3) + (ba << 2) + (b << 5) < max-unat 32*›
⟨*proof*⟩

lemma *AStatus-implI2*: ‹*(ac :: 2 word, ba) ∈ unat-rel ⟹ ba << (2::nat) < max-unat (32::nat)*›
  ⟨*proof*⟩

lemma *is-up-2-32*[*simp*]: ‹*is-up′ UCAST(2 → 32)*›
  ⟨*proof*⟩

sepref-def *AStatus-impl* [*llvm-inline*]
  is [] ‹*uncurry2 (RETURN ooo AStatus-impl1)*›
  :: ‹$[\lambda((a,b),\ c).\ c << 5 < max\text{-}unat\ 32]_a$
    $uint32\text{-}nat\text{-}assn^k *_a (unat\text{-}assn'\ TYPE(2))^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

lemma *Collect-eq-simps3*: ‹*P O {(c, a). a = c ∧ Q c} = {(a, b). (a, b) ∈ P ∧ Q b}*›
  ‹*P O {(c, a). c = a ∧ Q c} = {(a, b). (a, b) ∈ P ∧ Q b}*›
  ⟨*proof*⟩

lemma *unat-rel-2-br*: ‹*(((unat-rel :: (2 word × -) set) O br id (λn. n ≤ 3))) = ((unat-rel))*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *AStatus-impl.refine*[*FCOMP AStatus-refine1*, *unfolded unat-rel-2-br*]

Arena Operations

Length **abbreviation** ‹*arena-fast-assn ≡ al-assn′ TYPE(64) arena-el-impl-assn*›

lemma *arena-lengthI*:
  assumes ‹*arena-is-valid-clause-idx a b*›
  shows ‹*Suc 0 ≤ b*›
  and ‹*b < length a*›
  and ‹*is-Size (a ! (b − Suc 0))*›
  ⟨*proof*⟩

lemma *arena-length-alt*:
‹*arena-length arena i* = (
let l = *xarena-length* (*arena*!(*i* − *snat-const TYPE*(*64*) *1*))
in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
⟨*proof*⟩

sepref-register *arena-length*
sepref-def *arena-length-impl*
is ‹*uncurry* (*RETURN oo arena-length*)›
:: ‹[*uncurry arena-is-valid-clause-idx*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *snat-assn*′ *TYPE*(*64*)›
⟨*proof*⟩

Literal at given position **lemma** *arena-lit-implI*:
assumes ‹*arena-lit-pre a b*›
shows ‹*b* < *length a*› ‹*is-Lit* (*a* ! *b*)›
⟨*proof*⟩

sepref-register *arena-lit xarena-lit*
sepref-def *arena-lit-impl*
is ‹*uncurry* (*RETURN oo arena-lit*)›
:: ‹[*uncurry arena-lit-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-register *mop-arena-lit mop-arena-lit2*
sepref-def *mop-arena-lit-impl*
is ‹*uncurry* (*mop-arena-lit*)›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*›
⟨*proof*⟩

sepref-def *mop-arena-lit2-impl*
is ‹*uncurry2* (*mop-arena-lit2*)›
:: ‹[λ((*N*, -), -). *length N* ≤ *sint64-max*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

Status of the clause **lemma** *arena-status-implI*:
assumes ‹*arena-is-valid-clause-vdom a b*›
shows ‹*2* ≤ *b*› ‹*b* − *2* < *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›
⟨*proof*⟩

sepref-register *arena-status xarena-status*
sepref-def *arena-status-impl*
is ‹*uncurry* (*RETURN oo arena-status*)›
:: ‹[*uncurry arena-is-valid-clause-vdom*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *status-impl-assn*›
⟨*proof*⟩

Swap literals **sepref-register** *swap-lits*
sepref-def *swap-lits-impl* **is** ‹*uncurry3* (*RETURN oooo swap-lits*)›
:: ‹[λ(((*C*,*i*),*j*),*arena*). *C* + *i* < *length arena* ∧ *C* + *j* < *length arena*]$_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
⟨*proof*⟩

Get LBD **lemma** *get-clause-LBD-pre-implI*:
assumes ‹*get-clause-LBD-pre a b*›
shows ‹*2* ≤ *b*› ‹*b* − *2* < *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›

⟨*proof*⟩

sepref-register *arena-lbd mop-arena-lbd*
sepref-def *arena-lbd-impl*
is ‹*uncurry* (*RETURN oo arena-lbd*)›
:: ‹[*uncurry get-clause-LBD-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} →*uint32-nat-assn*›
⟨*proof*⟩

sepref-def *mop-arena-lbd-impl*
is ‹*uncurry mop-arena-lbd*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

used flag

sepref-register *arena-used*
sepref-def *arena-used-impl*
is ‹*uncurry* (*RETURN oo arena-used*)›
:: ‹[*uncurry get-clause-LBD-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *unat-assn′ TYPE*(*2*)›
⟨*proof*⟩

Get Saved Position

lemma *arena-posI*:
assumes ‹*get-saved-pos-pre a b*›
shows ‹*3* ≤ *b*›
and ‹*b* < *length a*›
and ‹*is-Pos* (*a* ! (*b* − *3*))›
⟨*proof*⟩

lemma *arena-pos-alt*:
‹*arena-pos arena i* = (
let l = *xarena-pos* (*arena*!(*i* − *snat-const TYPE*(*64*) *3*))
in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
⟨*proof*⟩

sepref-register *arena-pos*
sepref-def *arena-pos-impl*
is ‹*uncurry* (*RETURN oo arena-pos*)›
:: ‹[*uncurry get-saved-pos-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *snat-assn′ TYPE*(*64*)›
⟨*proof*⟩

Update LBD

lemma *update-lbdI*:
assumes ‹*update-lbd-pre* ((*b*, *lbd*), *a*)›
shows ‹*2* ≤ *b*›
and ‹*b* −*2* < *length a*›
and ‹*arena-is-valid-clause-vdom a b*›
and ‹*get-clause-LBD-pre a b*›
⟨*proof*⟩

lemma *shorten-lbd-le*: ‹*shorten-lbd baa* << *5* < *max-unat 32*›
⟨*proof*⟩

sepref-register *update-lbd AStatus shorten-lbd*
sepref-def *shorten-lbd-impl*
is ‹*RETURN o shorten-lbd*›
:: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *update-lbd-impl*
  is ‹*uncurry2* (*RETURN ooo update-lbd*)›
  :: ‹$[update\text{-}lbd\text{-}pre]_a$ *sint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
  ⟨*proof*⟩

sepref-def *mop-arena-update-lbd-impl*
  is ‹*uncurry2 mop-arena-update-lbd*›
  :: ‹*sint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow_a$ *arena-fast-assn*›
  ⟨*proof*⟩

Update Saved Position

lemma *update-posI*:
  assumes ‹*isa-update-pos-pre* ((*b*, *pos*), *a*)›
  shows ‹$3 \leq b$› ‹$2 \leq pos$› ‹$b-3 <$ *length a*›
  ⟨*proof*⟩

lemma *update-posI2*:
  assumes ‹*isa-update-pos-pre* ((*b*, *pos*), *a*)›
  assumes ‹*rdomp* (*al-assn arena-el-impl-assn* :: - ⇒ (*32 word*, *64*) *array-list* ⇒ *assn*) *a*›
  shows ‹*pos* − *2* < *max-unat 32*›
⟨*proof*⟩

sepref-register *arena-update-pos*
sepref-def *update-pos-impl*
  is ‹*uncurry2* (*RETURN ooo arena-update-pos*)›
  :: ‹$[isa\text{-}update\text{-}pos\text{-}pre]_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
  ⟨*proof*⟩

sepref-register *IRRED LEARNED DELETED*
lemma *IRRED-impl*[*sepref-import-param*]: ‹(*0*,*IRRED*) ∈ *status-impl-rel*›
  ⟨*proof*⟩

lemma *LEARNED-impl*[*sepref-import-param*]: ‹(*1*,*LEARNED*) ∈ *status-impl-rel*›
  ⟨*proof*⟩

lemma *DELETED-impl*[*sepref-import-param*]: ‹(*3*,*DELETED*) ∈ *status-impl-rel*›
  ⟨*proof*⟩

lemma *mark-garbageI*:
  assumes ‹*mark-garbage-pre* (*a*, *b*)›
  shows ‹$2 \leq b$› ‹$b-2 <$ *length a*›
  ⟨*proof*⟩

sepref-register *extra-information-mark-to-delete*
sepref-def *mark-garbage-impl* **is** ‹*uncurry* (*RETURN oo extra-information-mark-to-delete*)›
  :: ‹$[mark\text{-}garbage\text{-}pre]_a$ *arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
  ⟨*proof*⟩

lemma *bit-shiftr-shiftl-same-le*:
  ‹*a* << *b* >> *b* ≤ *a*› **for** *a b c* :: *nat*
  ⟨*proof*⟩

lemma *bit-shiftl-shiftr-same-le*:
‹$a >> b << b \leq a$› **for** *a b c* :: *nat*
⟨*proof*⟩

lemma *valid-arena-arena-lbd-shift-le*:
assumes
‹*rdomp* (*al-assn arena-el-impl-assn*) *a*› **and**
‹$b \in\#$ *dom-m N*› **and**
‹*valid-arena a N vdom*›
shows ‹*arena-lbd a b* $<< 5 <$ *max-unat 32*›
⟨*proof*⟩

lemma *arena-mark-used-implI*:
assumes ‹*arena-act-pre a b*›
shows ‹$2 \leq b$› ‹$b - 2 <$ *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›
‹*arena-is-valid-clause-vdom a b*›
‹*get-clause-LBD-pre a b*›
‹*rdomp* (*al-assn arena-el-impl-assn*) *a* $\Longrightarrow$ *arena-lbd a b* $<< 5 <$ *max-unat 32*›
⟨*proof*⟩

lemma *mark-used-alt-def*:
‹*RETURN oo mark-used* =
(λ*arena i. do* {
lbd ← *RETURN* (*arena-lbd arena i*); *let status* = *arena-status arena i*;
RETURN (*arena*[*i* − *STATUS-SHIFT* := *AStatus status* (*arena-used arena i OR 1*) *lbd*])})›
⟨*proof*⟩

sepref-register *mark-used mark-used2*
sepref-def *mark-used-impl* **is** ‹*uncurry* (*RETURN oo mark-used*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

sepref-def *mark-used2-impl* **is** ‹*uncurry* (*RETURN oo mark-used2*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

sepref-register *mark-unused*
sepref-def *mark-unused-impl* **is** ‹*uncurry* (*RETURN oo mark-unused*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
⟨*proof*⟩

sepref-def *mop-arena-mark-used-impl*
is ‹*uncurry mop-arena-mark-used*›
:: ‹*arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

sepref-def *mop-arena-mark-used2-impl*
is ‹*uncurry mop-arena-mark-used2*›
:: ‹*arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

Marked as used? **lemma** *arena-marked-as-used-implI*:
assumes ‹*marked-as-used-pre a b*›
shows ‹$2 \leq b$› ‹$b - 2 <$ *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›

⟨*proof*⟩

sepref-register *marked-as-used*
sepref-def *marked-as-used-impl*
is ‹*uncurry* (*RETURN oo marked-as-used*)›
:: ‹$[$*uncurry marked-as-used-pre*$]_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *unat-assn′ TYPE*(*2*)›
⟨*proof*⟩

sepref-register *MAX-LENGTH-SHORT-CLAUSE mop-arena-status*
sepref-def *MAX-LENGTH-SHORT-CLAUSE-impl* **is** ‹*uncurry0* (*RETURN MAX-LENGTH-SHORT-CLAUSE*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

definition *arena-other-watched-as-swap* :: ‹*nat list* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat nres*› **where**
‹*arena-other-watched-as-swap S L C i* = *do* {
ASSERT(*i* < *2* ∧
C + *i* < *length S* ∧
C < *length S* ∧
(*C* + *1*) < *length S*);
K ← *RETURN* (*S* ! *C*);
K′ ← *RETURN* (*S* ! (*1* + *C*));
RETURN (*L XOR K XOR K′*)
}›

lemma *arena-other-watched-as-swap-arena-other-watched*:
assumes
N: ‹(*N*, *N′*) ∈ ⟨*arena-el-rel*⟩*list-rel*› **and**
L: ‹(*L*, *L′*) ∈ *nat-lit-rel*› **and**
C: ‹(*C*, *C′*) ∈ *nat-rel*› **and**
i: ‹(*i*, *i′*) ∈ *nat-rel*›
shows
‹*arena-other-watched-as-swap N L C i* ≤ ⇓*nat-lit-rel*
(*arena-other-watched N′ L′ C′ i′*)›
⟨*proof*⟩

sepref-def *arena-other-watched-as-swap-impl*
is ‹*uncurry3 arena-other-watched-as-swap*›
:: ‹(*al-assn′* (*TYPE*(*64*)) *uint32-nat-assn*)^k $*_a$ *uint32-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
sint64-nat-assn^k $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *arena-other-watched-as-swap-arena-other-watched′*:
‹(*arena-other-watched-as-swap*, *arena-other-watched*) ∈
⟨*arena-el-rel*⟩*list-rel* → *nat-lit-rel* → *nat-rel* → *nat-rel* →
⟨*nat-lit-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *arena-fast-al-unat-assn*:
‹*hr-comp* (*al-assn unat-assn*) (⟨*arena-el-rel*⟩*list-rel*) = *arena-fast-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] =
arena-other-watched-as-swap-impl.refine[*FCOMP arena-other-watched-as-swap-arena-other-watched′*,
unfolded arena-fast-al-unat-assn]

end

sepref-def *mop-arena-length-impl*
  is ‹*uncurry mop-arena-length*›
  :: ‹$arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

sepref-def *mop-arena-status-impl*
  is ‹*uncurry mop-arena-status*›
  :: ‹$arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a status\text{-}impl\text{-}assn$›
  ⟨*proof*⟩

experiment begin
export-llvm
  arena-length-impl
  arena-lit-impl
  arena-status-impl
  swap-lits-impl
  arena-lbd-impl
  arena-pos-impl
  update-lbd-impl
  update-pos-impl
  mark-garbage-impl
  mark-used-impl
  mark-unused-impl
  marked-as-used-impl
  MAX-LENGTH-SHORT-CLAUSE-impl
  mop-arena-status-impl
end

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

Chapter 3

The memory representation: Manipulation of all clauses

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 ×$_f$ *nat-rel* ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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

type-synonym *vdom* = ‹*nat set*›

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

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

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

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

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]
definition *fmap-length-rll-u64* :: ‹(*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat*› **where**

⟨*fmap-length-rll-u64 l i = length-uint32-nat (l ∝ i)*⟩

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

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

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

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

definition *fm-add-new* **where**

```
⟨fm-add-new b C N0 = do {
   let s = length C − 2;
   let lbd = shorten-lbd s;
   let st = (if b then AStatus IRRED 0 lbd else AStatus LEARNED 0 lbd);
   let l = length N0;
   let N = (if is-short-clause C then
         (((N0 @ [st]))) @ [ASize s]
         else ((((N0 @ [APos 0]) @ [st]))) @ [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 C)
     (λ(i, N). do {
       ASSERT(i < length C);
       RETURN (i+1, N @ [ALit (C ! i)])
     })
     (0, N);
   RETURN (N, l + header-size C)
 }⟩
```

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

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

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

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

definition *AStatus-IRRED* **where**
⟨*AStatus-IRRED = AStatus IRRED 0*⟩

```
definition AStatus-IRRED2 where
  ‹AStatus-IRRED2 = AStatus IRRED 1›

definition AStatus-LEARNED where
  ‹AStatus-LEARNED = AStatus LEARNED 1›


definition AStatus-LEARNED2 where
  ‹AStatus-LEARNED2 = AStatus LEARNED 0›


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 uint32-max) ⟹
        2 ≤ length ba ⟹
        length b ≤ uint64-max − (uint32-max + 5) ⟹
        (aa, header-size ba) ∈ uint64-nat-rel ⟹
        (ab, length b) ∈ uint64-nat-rel ⟹
        length b + header-size ba ≤ uint64-max›
  ⟨proof⟩



definition (in −)four-uint64-nat where
  [simp]: ‹four-uint64-nat = (4 :: nat)›
definition (in −)five-uint64-nat where
  [simp]: ‹five-uint64-nat = (5 :: nat)›

definition append-and-length-fast-code-pre where
  ‹append-and-length-fast-code-pre ≡ λ((b, C), N). length C ≤ uint32-max+2 ∧ length C ≥ 2 ∧
           length N + length C + MAX-HEADER-SIZE ≤ sint64-max›


lemma fm-add-new-alt-def:
 ‹fm-add-new b C N0 = do {
      let s = length C − 2;
      let lbd = shorten-lbd s;
      let st = (if b then AStatus-IRRED lbd else AStatus-LEARNED2 lbd);
      let l = length N0;
      let N =
        (if is-short-clause C
          then ((N0 @ [st])) @
              [ASize s]
          else (((N0 @ [APos 0]) @ [st])) @
              [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 C)
          (λ(i, N). do {
              - ← ASSERT (i < length C);
              RETURN (i + 1, N @ [ALit (C ! i)])
            })
          (0, N);
      RETURN (N, l + header-size C)
```

```
    }›
  ⟨proof⟩
```

definition *fmap-swap-ll-u64* **where**

```
  [simp]: ‹fmap-swap-ll-u64 = fmap-swap-ll›
```

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 (arena-length old-arena C) ≤ 4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE));
   let st = C − (if (arena-length old-arena C) ≤ 4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE);
    ASSERT(C + (arena-length old-arena C) ≤ length old-arena);
    let en = C + (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 (arena-l
          (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⟩
```

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⟩
```

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

```
  ⟨proof⟩
```

lemma *valid-arena-ge-length-clauses*:

```
  assumes ‹valid-arena arena N vdom›
  shows ‹length arena ≥ (∑ C ∈# dom-m N. length (N ∝ C) + header-size (N ∝ C))›
⟨proof⟩
```

lemma *valid-arena-size-dom-m-le-arena*: ‹*valid-arena arena N vdom* ⟹ *size* (*dom-m N*) ≤ *length*

arena›
⟨*proof*⟩

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

sepref-register *is-short-clause header-size fm-add-new-fast fm-mv-clause-to-new-arena*

abbreviation *clause-ll-assn* :: ‹*nat clause-l* ⇒ - ⇒ *assn*› **where**
‹*clause-ll-assn* ≡ *larray64-assn unat-lit-assn*›

sepref-def *is-short-clause-code*
is ‹*RETURN o is-short-clause*›
:: ‹ $\mathit{clause\text{-}ll\text{-}assn}^k \rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *header-size-code*
is ‹*RETURN o header-size*›
:: ‹$\mathit{clause\text{-}ll\text{-}assn}^k \rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

lemma *header-size-bound*: ‹*header-size x* ≤ *MAX-HEADER-SIZE*› ⟨*proof*⟩

lemma *fm-add-new-bounds1*: ⟦
length a2′ < *header-size baa* + *length b* + *length baa*;
length b + *length baa* + *MAX-HEADER-SIZE* ≤ *sint64-max* ⟧
⟹ *Suc* (*length a2′*) < *max-snat 64*

‹*length b* + *length baa* + *MAX-HEADER-SIZE* ≤ *sint64-max* ⟹ *length b* + *header-size baa* < *max-snat 64*›
⟨*proof*⟩

sepref-def *append-and-length-fast-code*
is ‹*uncurry2 fm-add-new-fast*›
:: ‹$[\mathit{append\text{-}and\text{-}length\text{-}fast\text{-}code\text{-}pre}]_a$
$\mathit{bool1\text{-}assn}^k *_a \mathit{clause\text{-}ll\text{-}assn}^k *_a (\mathit{arena\text{-}fast\text{-}assn})^d \rightarrow$
$\mathit{arena\text{-}fast\text{-}assn} \times_a \mathit{sint64\text{-}nat\text{-}assn}$›
⟨*proof*⟩

sepref-def *fm-mv-clause-to-new-arena-fast-code*
is ‹*uncurry2 fm-mv-clause-to-new-arena*›
:: ‹$[\lambda((n, \mathit{arena}_o), \mathit{arena}).\ \mathit{length}\ \mathit{arena}_o \le \mathit{sint64\text{-}max} \wedge \mathit{length}\ \mathit{arena} + \mathit{arena\text{-}length}\ \mathit{arena}_o\ n +$
$(\mathit{if}\ \mathit{arena\text{-}length}\ \mathit{arena}_o\ \ n \le 4\ \mathit{then}\ \mathit{MIN\text{-}HEADER\text{-}SIZE}\ \mathit{else}\ \mathit{MAX\text{-}HEADER\text{-}SIZE}) \le \mathit{sint64\text{-}max}]_a$
$\mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{arena\text{-}fast\text{-}assn}^k *_a \mathit{arena\text{-}fast\text{-}assn}^d \rightarrow \mathit{arena\text{-}fast\text{-}assn}$›
⟨*proof*⟩

experiment begin
export-llvm
is-short-clause-code

  header-size-code
  append-and-length-fast-code
  fm-mv-clause-to-new-arena-fast-code
end

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

begin

Chapter 4

Efficient 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).

4.1 Polarities

type-synonym *tri-bool* = ⟨*bool option*⟩

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 (**in** −) *tri-bool-eq* :: ⟨*tri-bool* ⇒ *tri-bool* ⇒ *bool*⟩ **where**
⟨*tri-bool-eq* = (=)⟩

4.2 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 SET-TRUE*
else if Neg L ∈ *lits-of-l M then SET-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*›

4.3 Control Stack

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

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

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

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

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

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

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

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

definition *control-stack'* **where**

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

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

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

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

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*⟩

4.4 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 $\mathcal{A}$}›

4.5 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*⟩

4.6 Code generation

4.6.1 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*, *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*, *reason*, *k*, *cs*))›

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

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

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

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

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

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

4.6.2 Level of a literal

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

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

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

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

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

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

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

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*›
⟨*proof*⟩

4.6.3 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*›
⟨*proof*⟩

4.6.4 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*)) ∈
[λ(*M*, *L*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ $\times_f$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

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

4.6.5 Length of the trail

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

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

lemma *isa-length-trail-pre*:
‹(*M*, *M′*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *isa-length-trail-pre M*›
⟨*proof*⟩

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

definition *mop-isa-length-trail* **where**
‹*mop-isa-length-trail* = (λ(*M*). *do* {
  ASSERT(*isa-length-trail-pre M*);
  RETURN (*isa-length-trail M*)
})›

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

4.6.6 Consing elements

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

definition *cons-trail-Propagated-tr* :: ‹*nat literal* ⇒ *nat* ⇒ *trail-pol* ⇒ *trail-pol nres*› **where**
‹*cons-trail-Propagated-tr* = (λ*L C* (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*). *do* {
  ASSERT(*cons-trail-Propagated-tr-pre* ((*L*, *C*), (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*)));
  RETURN (*M′* @ [*L*], *let xs* = *xs*[*nat-of-lit L* := *SET-TRUE*] *in xs*[*nat-of-lit* (−*L*) := *SET-FALSE*],
   lvls[*atm-of L* := *k*], *reasons*[*atm-of L*:= *C*], *k*, *cs*)})›

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

⟨*proof*⟩

lemma *cons-trail-Propagated-tr-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* $((L, C), M')$›
⟨*proof*⟩

lemma *cons-trail-Propagated-tr*:
‹(*uncurry2* (*cons-trail-Propagated-tr*), *uncurry2* (*cons-trail-propagate-l*)) $\in$
$[\lambda((L, C), M).\ L \in\# \mathcal{L}_{all}\ \mathcal{A} \wedge C \neq$ *DECISION-REASON*$]_f$
Id $\times_f$ *nat-rel* $\times_f$ *trail-pol* $\mathcal{A} \rightarrow \langle$*trail-pol* $\mathcal{A}\rangle$*nres-rel*›
⟨*proof*⟩

lemma *cons-trail-Propagated-tr2*:
‹$(((L, C), M), ((L', C'), M')) \in$ *Id* $\times_f$ *Id* $\times_f$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$
$C \neq$ *DECISION-REASON* $\Longrightarrow$
cons-trail-Propagated-tr L C M
$\leq \Downarrow (\{(M'', M''').\ (M'', M''') \in$ *trail-pol* $\mathcal{A} \wedge M''' =$ *Propagated* L C $\#$ $M' \wedge$ *no-dup* $M'''\})$
(*cons-trail-propagate-l* L' C' M')›
⟨*proof*⟩

lemma *undefined-lit-count-decided-uint32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹$\forall L \in$*set* M. *lit-of* $L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** *n-d*: ‹*no-dup* M› **and**
‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹*undefined-lit* M L› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*Suc* (*count-decided* M) $\leq$ *uint32-max*›
⟨*proof*⟩

lemma *length-trail-uint32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹$\forall L \in$*set* M. *lit-of* $L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** *n-d*: ‹*no-dup* M› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*length* $M \leq$ *uint32-max*›
⟨*proof*⟩

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* ⇒ (*nat literal* × *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))›

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 := 0],
    reasons, if reasons ! atm-of L = DECISION-REASON then k−1 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')›
  ⟨proof⟩

lemma control-stack-dec-butlast:
  ‹control-stack b (Decided x1 # M's) ⟹ control-stack (butlast b) M's›
  ⟨proof⟩

lemma tl-trail-tr:
  ‹((RETURN o tl-trailt-tr), (RETURN o tl)) ∈
    [λM. M ≠ []]_f trail-pol 𝒜 → ⟨trail-pol 𝒜⟩nres-rel›
⟨proof⟩

lemma tl-trailt-tr-pre:
  assumes ‹M ≠ []›
    ‹(M', M) ∈ trail-pol 𝒜›
  shows ‹tl-trailt-tr-pre M'›
⟨proof⟩

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 := 0],
    reasons, k, cs))›

definition tl-trail-propedt-tr-pre where
  ‹tl-trail-propedt-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)›

lemma tl-trail-propedt-tr-pre:
  assumes ‹(M', M) ∈ trail-pol 𝒜› and
    ‹M ≠ []›
  shows ‹tl-trail-propedt-tr-pre M'›
  ⟨proof⟩


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

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

lemma *lit-of-last-trail-pol-lit-of-last-trail*:
$\langle(\mathit{RETURN\ o\ lit\text{-}of\text{-}last\text{-}trail\text{-}pol},\ \mathit{RETURN\ o\ lit\text{-}of\text{-}hd\text{-}trail}) \in$
$[\lambda S.\ S \neq []]_f\ \mathit{trail\text{-}pol}\ \mathcal{A} \rightarrow \langle Id\rangle \mathit{nres\text{-}rel}\rangle$
$\langle proof\rangle$

4.6.7 Setting a new literal

definition *cons-trail-Decided* :: $\langle \mathit{nat\ literal} \Rightarrow (\mathit{nat},\ \mathit{nat})\ \mathit{ann\text{-}lits} \Rightarrow (\mathit{nat},\ \mathit{nat})\ \mathit{ann\text{-}lits}\rangle$ **where**
$\langle \mathit{cons\text{-}trail\text{-}Decided}\ L\ M' = \mathit{Decided}\ L\ \#\ M'\rangle$

definition *cons-trail-Decided-tr* :: $\langle \mathit{nat\ literal} \Rightarrow \mathit{trail\text{-}pol} \Rightarrow \mathit{trail\text{-}pol}\rangle$ **where**
$\langle \mathit{cons\text{-}trail\text{-}Decided\text{-}tr} = (\lambda L\ (M',\ xs,\ lvls,\ reasons,\ k,\ cs).\ do\{$
$\quad let\ n = length\ M'\ in$
$\quad (M'\ @\ [L],\ let\ xs = xs[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{SET\text{-}TRUE}]\ in\ xs[\mathit{nat\text{-}of\text{-}lit}\ (-L) := \mathit{SET\text{-}FALSE}],$
$\quad\ lvls[\mathit{atm\text{-}of}\ L := k{+}1],\ reasons[\mathit{atm\text{-}of}\ L := \mathit{DECISION\text{-}REASON}],\ k{+}1,\ cs\ @\ [n])\})\rangle$

definition *cons-trail-Decided-tr-pre* **where**
$\langle \mathit{cons\text{-}trail\text{-}Decided\text{-}tr\text{-}pre} =$
$\quad (\lambda(L,\ (M,\ xs,\ lvls,\ reason,\ k,\ cs)).\ \mathit{nat\text{-}of\text{-}lit}\ L < length\ xs \wedge \mathit{nat\text{-}of\text{-}lit}\ (-L) < length\ xs \wedge$
$\quad\ \mathit{atm\text{-}of}\ L < length\ lvls \wedge \mathit{atm\text{-}of}\ L < length\ reason\ \ \wedge length\ cs < \mathit{uint32\text{-}max} \wedge$
$\quad\ Suc\ k \leq \mathit{uint32\text{-}max} \wedge length\ M < \mathit{uint32\text{-}max})\rangle$

lemma *length-cons-trail-Decided*[*simp*]:
$\langle length\ (\mathit{cons\text{-}trail\text{-}Decided}\ L\ M) = Suc\ (length\ M)\rangle$
$\langle proof\rangle$

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

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

4.6.8 Polarity: Defined or Undefined

definition (**in** $-$) *defined-atm-pol-pre* **where**
$\langle \mathit{defined\text{-}atm\text{-}pol\text{-}pre} = (\lambda(M,\ xs,\ lvls,\ k)\ L.\ 2{*}L < length\ xs \wedge$
$\quad 2{*}L \leq \mathit{uint32\text{-}max})\rangle$

definition (**in** $-$) *defined-atm-pol* **where**
$\langle \mathit{defined\text{-}atm\text{-}pol} = (\lambda(M,\ xs,\ lvls,\ k)\ L.\ \neg((xs!(2{*}L)) = None))\rangle$

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

4.6.9 Reasons

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

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

definition *get-propagation-reason-raw-pol* :: ‹*trail-pol* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat nres*› **where**
‹*get-propagation-reason-raw-pol* = (λ(-, -, -, *reasons*, -) *L*. *do* {
    ASSERT(*atm-of L* < *length reasons*);
    let r = *reasons* ! *atm-of L*;
    RETURN r})›

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

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

definition *get-the-propagation-reason-pol* :: ‹*trail-pol* $\Rightarrow$ *nat literal* $\Rightarrow$ *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* $\wedge$ *r* $\neq$ *DECISION-REASON then Some r else None*)})›

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

4.7 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)) ∈
    [λ(M, i). i < length M]_f trail-pol 𝒜 ×_r nat-rel → ⟨Id⟩nres-rel›
  ⟨proof⟩
```

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

```
definition trail-pol-no-CS :: ‹nat multiset ⇒ (trail-pol × (nat, nat) ann-lits) set›
where
  ‹trail-pol-no-CS 𝒜 =
   {((M', xs, lvls, reasons, k, cs), M). ((M', reasons), M) ∈ ann-lits-split-reasons 𝒜 ∧
    no-dup M ∧
    (∀ L ∈# ℒ_all 𝒜. nat-of-lit L < length xs ∧ xs ! (nat-of-lit L) = polarity M L) ∧
    (∀ L ∈# ℒ_all 𝒜. atm-of L < length lvls ∧ lvls ! (atm-of L) = get-level M L) ∧
    (∀ L∈set M. lit-of L ∈# ℒ_all 𝒜) ∧
    isasat-input-bounded 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 := 0],
    reasons, k, cs))›

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

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

lemma tl-trailt-tr-no-CS-pre:
  assumes ‹(M', M) ∈ trail-pol-no-CS 𝒜› and ‹M ≠ []›
  shows ‹tl-trailt-tr-no-CS-pre M'›
⟨proof⟩

lemma tl-trail-tr-no-CS:
  ‹((RETURN o tl-trailt-tr-no-CS), (RETURN o tl)) ∈
    [λM. M ≠ []]_f trail-pol-no-CS 𝒜 → ⟨trail-pol-no-CS 𝒜⟩nres-rel›
  ⟨proof⟩

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

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

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

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

definition (**in** −) *trail-conv-back-imp* :: ‹nat ⇒ trail-pol ⇒ trail-pol nres› **where**
‹trail-conv-back-imp j = (λ(M, xs, lvls, reason, -, cs). do {
    ASSERT(j ≤ length cs); RETURN (M, xs, lvls, reason, j, take (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›
⟨proof⟩

definition (**in** −)*take-arl* **where**
‹take-arl = (λi (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›
⟨proof⟩

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

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

lemma *control-stack-is-decided*:
‹control-stack cs M ⟹ c∈set cs ⟹ is-decided ((rev M)!c)›
⟨proof⟩

lemma *control-stack-distinct*:
 ‹*control-stack cs M* ⟹ *distinct cs*›
 ⟨*proof*⟩

lemma *control-stack-level-control-stack*:
 assumes
  cs: ‹*control-stack cs M*› **and**
  n-d: ‹*no-dup M*› **and**
  i: ‹*i* < *length cs*›
 shows ‹*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) = *Suc i*›
⟨*proof*⟩

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*)
  })›
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*›
 ⟨*proof*⟩

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*›
 ⟨*proof*⟩

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*) ∈
   $[\lambda S.\ S \neq []]_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
 ⟨*proof*⟩

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

4.7.1 Variable-Move-to-Front

Variants around head and last

definition *option-hd* :: ‹*'a list* ⇒ *'a option*› **where**
 ‹*option-hd xs* = (*if xs* = [] *then None else Some* (*hd xs*))›

lemma *option-hd-None-iff*[*iff*]: ‹*option-hd zs* = *None* ⟷ *zs* = []› ‹*None* = *option-hd zs* ⟷ *zs* = []›
 ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

Specification

type-synonym *'v abs-vmtf-ns* = ‹$'v\ \mathit{set} \times 'v\ \mathit{set}$›
type-synonym *'v abs-vmtf-ns-remove* = ‹$'v\ \mathit{abs\text{-}vmtf\text{-}ns} \times 'v\ \mathit{set}$›

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* :: ‹$\mathit{nat}\ \mathit{list} \Rightarrow \mathit{nat} \Rightarrow \mathit{nat\text{-}vmtf\text{-}node}\ \mathit{list} \Rightarrow \mathit{bool}$› **where**
Nil: ‹$\mathit{vmtf\text{-}ns}\ []\ \mathit{st}\ xs$› |
Cons1: ‹$a < \mathit{length}\ xs \Longrightarrow m \geq n \Longrightarrow xs\ !\ a = \mathit{VMTF\text{-}Node}\ (n{::}\mathit{nat})\ \mathit{None}\ \mathit{None} \Longrightarrow \mathit{vmtf\text{-}ns}\ [a]\ m\ xs$›
|
Cons: ‹$\mathit{vmtf\text{-}ns}\ (b\ \#\ l)\ m\ xs \Longrightarrow a < \mathit{length}\ xs \Longrightarrow xs\ !\ a = \mathit{VMTF\text{-}Node}\ n\ \mathit{None}\ (\mathit{Some}\ b) \Longrightarrow$
$a \neq b \Longrightarrow a \notin \mathit{set}\ l \Longrightarrow n > m \Longrightarrow$
$xs' = xs[b := \mathit{VMTF\text{-}Node}\ (\mathit{stamp}\ (xs!b))\ (\mathit{Some}\ a)\ (\mathit{get\text{-}next}\ (xs!b))] \Longrightarrow n' \geq n \Longrightarrow$
$\mathit{vmtf\text{-}ns}\ (a\ \#\ b\ \#\ l)\ n'\ xs'$›

inductive-cases *vmtf-nsE*: ‹$\mathit{vmtf\text{-}ns}\ xs\ \mathit{st}\ zs$›

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

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

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

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

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

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

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

lemma vmtf-ns-append-rebuild:
  assumes
    ‹(vmtf-ns (axs @ [ax]) an ns) › and
    ‹vmtf-ns (ay # azs) (stamp (ns!ay)) ns› and
    ‹stamp (ns!ax) > stamp (ns!ay)› and
    ‹distinct (axs @ [ax, ay] @ azs)›
  shows ‹vmtf-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))])›
  ⟨proof⟩
```

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

```
definition ns-vmtf-dequeue :: ‹nat ⇒ 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*›
⟨*proof*⟩

lemma *vmtf-ns-last-next*:
‹*vmtf-ns* (*xs* @ [*x*]) *m ns* ⟹ *get-next* (*ns* ! *x*) = *None*›
⟨*proof*⟩

lemma *vmtf-ns-hd-prev*:
‹*vmtf-ns* (*x* # *xs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *None*›
⟨*proof*⟩

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

lemma *vmtf-ns-last-mid-get-next-option-hd*:
‹*vmtf-ns* (*xs* @ *x* # *zs*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *option-hd zs*›
⟨*proof*⟩

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

lemma *vmtf-ns-last-mid-get-prev-option-last*:
‹*vmtf-ns* (*xs* @ *x* # *zs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *option-last xs*›
⟨*proof*⟩

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

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *stamp-ns-vmtf-dequeue*:
‹*axs* < *length zs* ⟹ *stamp* (*ns-vmtf-dequeue x zs* ! *axs*) = *stamp* (*zs* ! *axs*)›
⟨*proof*⟩

lemma *sorted-many-eq-append*: ‹*sorted* (*xs* @ [*x*, *y*]) ⟷ *sorted* (*xs* @ [*x*]) ∧ *x* ≤ *y*›
⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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*⟩

lemma *vmtf-ns-hd-next*:
  ‹*vmtf-ns* (*x* # *a* # *list*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *Some a*›
  ⟨*proof*⟩

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*⟩

lemma *vmtf-ns-stamp-distinct*:
  assumes ‹*vmtf-ns l m ns*›
  shows ‹*distinct* (*map* (λ*a*. *stamp* (*ns*!*a*)) *l*)›
  ⟨*proof*⟩

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*⟩

lemma *vmtf-ns-rescale*:
  assumes
    ‹*vmtf-ns l m xs*› **and**
    ‹*sorted* (*map* (λ*a*. *st* ! *a*) (*rev l*))› **and** ‹*distinct* (*map* (λ*a*. *st* ! *a*) *l*)›
    ‹∀ *a* ∈ *set l*. *get-prev* (*zs* ! *a*) = *get-prev* (*xs* ! *a*)› **and**
    ‹∀ *a* ∈ *set l*. *get-next* (*zs* ! *a*) = *get-next* (*xs* ! *a*)› **and**
    ‹∀ *a* ∈ *set l*. *stamp* (*zs* ! *a*) = *st* ! *a*› **and**
    ‹*length xs* ≤ *length zs*› **and**
    ‹∀ *a*∈*set l*. *a* < *length st*› **and**
    m′: ‹∀ *a* ∈ *set l*. *st* ! *a* < *m′*›
  shows ‹*vmtf-ns l m′ zs*›
  ⟨*proof*⟩

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

Abstract Invariants Invariants

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

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

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

Implementation

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

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

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

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

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

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

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

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

definition (**in** −) *vmtf-en-dequeue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf*› **where**

‹vmtf-en-dequeue = (λM L vm. vmtf-enqueue M L (vmtf-dequeue L vm))›

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

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

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

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

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

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

lemma vmtf-atm-of-ys-iff:
assumes
vmtf-ns: ‹vmtf-ns (ys′ @ xs′) m ns› **and**

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

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

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

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

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

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

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

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

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

```
  shows ‹vmtf-en-dequeue-pre 𝒜 ((M, A), vm)›
  ⟨proof⟩

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

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›
  ⟨proof⟩


definition vmtf-find-next-undef :: ‹nat multiset ⇒ vmtf-remove-int ⇒ (nat, nat) ann-lits ⇒ (nat option)
nres› where
‹vmtf-find-next-undef 𝒜 = (λ((ns, m, fst-As, lst-As, next-search), to-remove) M. do {
   WHILE_T^λnext-search. ((ns, m, fst-As, lst-As, next-search), to-remove) ∈ vmtf 𝒜 M ∧          (next-search ≠ None ⟶ Pos (t
     (λnext-search. next-search ≠ None ∧ defined-lit M (Pos (the next-search)))
     (λnext-search. do {
        ASSERT(next-search ≠ None);
        let n = the next-search;
        ASSERT(Pos n ∈# ℒ_all  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) ∈ vmtf 𝒜 M›
  shows ‹vmtf-find-next-undef 𝒜 ((ns, m, fst-As, lst-As, next-search), to-remove) M
     ≤ ⇓ Id (SPEC (λL. ((ns, m, fst-As, lst-As, L), to-remove) ∈ vmtf 𝒜 M ∧
        (L = None ⟶ (∀L∈#ℒ_all 𝒜. defined-lit M L)) ∧
        (L ≠ None ⟶ Pos (the L) ∈# ℒ_all 𝒜 ∧ undefined-lit M (Pos (the L)))))›
⟨proof⟩

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⟩

lemma vmtf-unset-vmtf-tl:
```

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

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*⟩

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

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

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

lemma *vmtf-ns-Cons*:
assumes
vmtf: ‹*vmtf-ns* (*b* # *l*) *m xs*› **and**
a-xs: ‹*a* < *length xs*› **and**
ab: ‹*a* ≠ *b*› **and**
a-l: ‹*a* ∉ *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'* ≥ *n*›
shows ‹*vmtf-ns* (*a* # *b* # *l*) *n' xs'*›
⟨*proof*⟩

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⟩

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⟩

lemma length-vmtf-cons[simp]: ‹length (vmtf-cons ns L n m) = length ns›
  ⟨proof⟩

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⟩

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 𝒜›
  shows
```

‹*vmtf-rescale vm ≤ SPEC (λvm. (vm, to-remove) ∈ vmtf 𝒜 M ∧ fst (snd vm) ≤ uint32-max)*›
(**is** ‹*?A ≤ ?R*›)
⟨*proof*⟩

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

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

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

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

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

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

definition *atoms-hash-del-pre* **where**
‹*atoms-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* $\mathcal{A}_{in}$ *= (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) ← $WHILE_T{}^{\lambda(i,\ vm',\ h).\ i \leq length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge\quad (i < length\ to\text{-}remove}$
(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*))
    ≤ ⇓(*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*⟩

lemma *vmtf-change-to-remove-order′*:
  ‹(*uncurry* (*vmtf-flush-int* $\mathcal{A}_{in}$), *uncurry* (*vmtf-flush* $\mathcal{A}_{in}$)) ∈
   [λ(*M*, *vm*). *vm* ∈ *vmtf* $\mathcal{A}_{in}$ *M* ∧ *isasat-input-bounded* $\mathcal{A}_{in}$ ∧ *isasat-input-nempty* $\mathcal{A}_{in}$]$_f$
     Id $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$) → ⟨(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$)⟩ *nres-rel*›
  ⟨*proof*⟩

4.7.2 Phase saving

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

definition *phase-saving* :: ‹*nat multiset* ⇒ *phase-saver* ⇒ *bool*› **where**
‹*phase-saving* $\mathcal{A}$ φ ⟷ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length* φ)›

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

definition *save-phase* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
  ‹*save-phase L* φ = φ[*atm-of L* := *is-pos L*]›

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

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

definition *save-phase-inv* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
  ‹*save-phase-inv L* φ = φ[*atm-of L* := ¬*is-pos L*]›

end
theory *LBD*
  imports *IsaSAT-Literals*
begin

Chapter 5

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.

Remark that we combine the LBD with a MTF scheme.

5.1 Types and relations

type-synonym *lbd* = ‹*bool list*›
type-synonym *lbd-ref* = ‹*nat list* × *nat* × *nat*›

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.

```
definition lbd-ref :: ‹(lbd-ref × lbd) set› where
  ‹lbd-ref = {((lbd, stamp, m), lbd').
       length lbd' ≤ Suc (Suc (uint32-max div 2)) ∧
       m = length (filter id lbd') ∧
       stamp > 0 ∧
       length lbd = length lbd' ∧
      (∀ v ∈ set lbd. v ≤ stamp) ∧
      (∀ i < length lbd'. lbd' ! i ⟷ lbd ! i = stamp)
}›
```

5.2 Testing if a level is marked

```
definition level-in-lbd :: ‹nat ⇒ lbd ⇒ bool› where
  ‹level-in-lbd i = (λlbd. i < length lbd ∧ lbd!i)›
```

```
definition level-in-lbd-ref :: ‹nat ⇒ lbd-ref ⇒ bool› where
  ‹level-in-lbd-ref = (λi (lbd, stamp, -). i < length-uint32-nat lbd ∧ lbd!i = stamp)›
```

lemma *level-in-lbd-ref-level-in-lbd*:

```
  ‹(uncurry (RETURN oo level-in-lbd-ref), uncurry (RETURN oo level-in-lbd)) ∈
    nat-rel ×_r lbd-ref →_f ⟨bool-rel⟩nres-rel›
  ⟨proof⟩
```

5.3 Marking more levels

```
definition list-grow where
  ‹list-grow xs n x = xs @ replicate (n − length xs) x›

definition lbd-write :: ‹lbd ⇒ nat ⇒ lbd› where
  ‹lbd-write = (λlbd i.
    (if i < length lbd then (lbd[i := True])
     else ((list-grow lbd (i + 1) False)[i := True])))›


definition lbd-ref-write :: ‹lbd-ref ⇒ nat ⇒ lbd-ref nres›  where
  ‹lbd-ref-write = (λ(lbd, stamp, n) i. do {
    ASSERT(length lbd ≤ uint32-max ∧ n + 1 ≤ uint32-max);
    (if i < length-uint32-nat lbd then
       let n = if lbd ! i = stamp then n else n+1 in
       RETURN (lbd[i := stamp], stamp, n)
     else do {
        ASSERT(i + 1 ≤ uint32-max);
        RETURN ((list-grow lbd (i + 1) 0)[i := stamp], stamp, n + 1)
     })
  })›

lemma length-list-grow[simp]:
  ‹length (list-grow xs n a) = max (length xs) n›
  ⟨proof⟩

lemma list-update-append2: ‹i ≥ length xs ⟹ (xs @ ys)[i := x] = xs @ ys[i − length xs := x]›
  ⟨proof⟩

lemma lbd-ref-write-lbd-write:
  ‹(uncurry (lbd-ref-write), uncurry (RETURN oo lbd-write)) ∈
    [λ(lbd, i). i ≤ Suc (uint32-max div 2)]_f
     lbd-ref ×_f nat-rel → ⟨lbd-ref⟩nres-rel›
  ⟨proof⟩
```

5.4 Cleaning the marked levels

```
definition lbd-emtpy-inv :: ‹nat list ⇒ nat list × nat ⇒ bool› where
  ‹lbd-emtpy-inv ys = (λ(xs, i). (∀j < i. xs ! j = 0) ∧ i ≤ length xs ∧ length ys = length xs)›

definition lbd-empty-loop-ref where
  ‹lbd-empty-loop-ref = (λ(xs, -, -). do {
    (xs, i) ←
       WHILE_T^lbd-emtpy-inv xs
        (λ(xs, i). i < length xs)
        (λ(xs, i). do {
           ASSERT(i < length xs);
           ASSERT(i + 1 < uint32-max);
           RETURN (xs[i := 0], i + 1)})
        (xs, 0);
```

    RETURN (*xs*, *1*, *0*)
  })›

definition *lbd-empty* **where**
  ‹*lbd-empty xs* = *RETURN* (*replicate* (*length xs*) *False*)›

lemma *lbd-empty-loop-ref*:
  assumes ‹((*xs*, *m*, *n*), *ys*) ∈ *lbd-ref*›
  shows
    ‹*lbd-empty-loop-ref* (*xs*, *m*, *n*) ≤ ⇓ *lbd-ref* (*RETURN* (*replicate* (*length ys*) *False*))›
⟨*proof*⟩

definition *lbd-empty-cheap-ref* **where**
  ‹*lbd-empty-cheap-ref* = (λ(*xs*, *stamp*, *n*). *RETURN* (*xs*, *stamp* + *1*, *0*))›

lemma *lbd-empty-cheap-ref*:
  assumes ‹((*xs*, *m*, *n*), *ys*) ∈ *lbd-ref*›
  shows
    ‹*lbd-empty-cheap-ref* (*xs*, *m*, *n*) ≤ ⇓ *lbd-ref* (*RETURN* (*replicate* (*length ys*) *False*))›
  ⟨*proof*⟩

definition *lbd-empty-ref* :: ‹*lbd-ref* ⇒ *lbd-ref nres*› **where**
  ‹*lbd-empty-ref* = (λ(*xs*, *m*, *n*). *if m* = *uint32-max then lbd-empty-loop-ref* (*xs*,*m*,*n*)
    else lbd-empty-cheap-ref (*xs*, *m*, *n*)) ›

lemma *lbd-empty-ref*:
  assumes ‹((*xs*, *m*, *n*), *ys*) ∈ *lbd-ref*›
  shows
    ‹*lbd-empty-ref* (*xs*, *m*, *n*) ≤ ⇓ *lbd-ref* (*RETURN* (*replicate* (*length ys*) *False*))›
  ⟨*proof*⟩

lemma *lbd-empty-ref-lbd-empty*:
  ‹(*lbd-empty-ref*, *lbd-empty*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
  ⟨*proof*⟩

definition (**in** −)*empty-lbd* :: ‹*lbd*› **where**
  ‹*empty-lbd* = (*replicate 32 False*)›

definition *empty-lbd-ref* :: ‹*lbd-ref*› **where**
  ‹*empty-lbd-ref* = (*replicate 32 0*, *1*, *0*)›

lemma *empty-lbd-ref-empty-lbd*:
  ‹(λ-. (*RETURN empty-lbd-ref*), λ-. (*RETURN empty-lbd*)) ∈ *unit-rel* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
  ⟨*proof*⟩

5.5 Extracting the LBD

We do not prove correctness of our algorithm, as we don't care about the actual returned value (for correctness).

definition *get-LBD* :: ‹*lbd* ⇒ *nat nres*› **where**
  ‹*get-LBD lbd* = *SPEC*(λ-. *True*)›

definition *get-LBD-ref* :: ‹*lbd-ref* ⇒ *nat nres*› **where**
  ‹*get-LBD-ref* = (λ(*xs*, *m*, *n*). *RETURN n*)›

lemma *get-LBD-ref*:
‹((*lbd*, *m*), *lbd′*) ∈ *lbd-ref* ⟹ *get-LBD-ref* (*lbd*, *m*) ≤ ⇓ *nat-rel* (*get-LBD lbd′*)›
⟨*proof*⟩

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

end
theory *LBD-LLVM*
imports *LBD IsaSAT-Literals-LLVM*
begin

no-notation *WB-More-Refinement.fref* (‹[-]$_f$ - → -› [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (‹- $\rightarrow_f$ -› [*60*,*60*] *60*)

type-synonym *′a larray64* = ‹(*′a*,*64*) *larray*›
type-synonym *lbd-assn* = ‹(*32 word*) *larray64* × *32 word* × *32 word*›

abbreviation *lbd-int-assn* :: ‹*lbd-ref* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-int-assn* ≡ *larray64-assn uint32-nat-assn* $\times_a$ *uint32-nat-assn* $\times_a$ *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-def** *level-in-lbd-code*
is [] ‹*uncurry* (*RETURN oo level-in-lbd-ref*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *lbd-int-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *lbd-empty-loop-code*
is ‹*lbd-empty-loop-ref*›
:: ‹*lbd-int-assn*^{d} $\rightarrow_a$ *lbd-int-assn*›
⟨*proof*⟩

sepref-def *lbd-empty-cheap-code*
is ‹*lbd-empty-cheap-ref*›
:: ‹[λ(-, *stamp*, -). *stamp* < *uint32-max*]$_a$ *lbd-int-assn*^{d} → *lbd-int-assn*›
⟨*proof*⟩

lemma *uint32-max-alt-def*: *uint32-max* = *4294967295*
⟨*proof*⟩
sepref-register *lbd-empty-cheap-ref lbd-empty-loop-ref*

sepref-def *lbd-empty-code*
is ‹*lbd-empty-ref*›
:: ‹*lbd-int-assn*^{d} $\rightarrow_a$ *lbd-int-assn*›
⟨*proof*⟩

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
‹(*lbd-empty-code*, *lbd-empty*) ∈ *lbd-assn*^{d} $\rightarrow_a$ *lbd-assn*›
⟨*proof*⟩

sepref-def *empty-lbd-code*
is [] ‹*uncurry0* (*RETURN empty-lbd-ref*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *lbd-int-assn*›
⟨*proof*⟩

lemma *empty-lbd-ref-empty-lbd*:
‹(*uncurry0* (*RETURN empty-lbd-ref*), *uncurry0* (*RETURN empty-lbd*)) ∈ *unit-rel* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *get-LBD-code*
is [] ‹*get-LBD-ref*›
:: ‹*lbd-int-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) ∈ *lbd-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

Marking more levels

lemmas *list-grow-alt* = *list-grow-def*[*unfolded op-list-grow-init*′*-def*[*symmetric*]]

sepref-def *lbd-write-code*
is [] ‹*uncurry lbd-ref-write*›
:: ‹ [λ(*lbd*, *i*). *i* ≤ *Suc* (*uint32-max div 2*)]$_a$
lbd-int-assn^d $*_a$ *uint32-nat-assn*^{k} → *lbd-int-assn*›
⟨*proof*⟩

lemma *lbd-write-hnr-*[*sepref-fr-rules*]:
‹(*uncurry lbd-write-code*, *uncurry* (*RETURN* ∘∘ *lbd-write*))
∈ [λ(*lbd*, *i*). *i* ≤ *Suc* (*uint32-max div 2*)]$_a$
lbd-assn^d $*_a$ *uint32-nat-assn*^{k} → *lbd-assn*›
⟨*proof*⟩

experiment begin

export-llvm
level-in-lbd-code
lbd-empty-code
empty-lbd-code
get-LBD-code
lbd-write-code

end

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
begin
```

Chapter 6

Refinement of the Watched Function

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 $(\mathit{nat} \times \mathit{uint32})$ to $(\mathit{nat} \times \mathit{uint32} \times \mathit{bool})$ to enable special handling of binary clauses, 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 $(2{::}\mathit{uint64})\ \widehat{}\ 32$ 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).

None of the problems appears in the LLVM code.

6.1 Definition

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

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

definition *mop-append-ll* :: $\langle {'}a\ \mathit{list\ list} \Rightarrow \mathit{nat\ literal} \Rightarrow {'}a \Rightarrow {'}a\ \mathit{list\ list\ nres}\rangle$ **where**

```
⟨mop-append-ll xs i x = do {
    ASSERT(nat-of-lit i < length xs);
    RETURN (append-ll xs (nat-of-lit i) x)
  }⟩
```

6.2 Operations

lemma *length-ll-length-ll-f*:
  $\langle(\mathit{uncurry}\ (\mathit{RETURN\ oo\ length\text{-}ll}), \mathit{uncurry}\ (\mathit{RETURN\ oo\ length\text{-}ll\text{-}f})) \in$
    $[\lambda(W, L).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}_{in}]_f\ ((\langle \mathit{Id}\rangle \mathit{map\text{-}fun\text{-}rel}\ (D_0\ \mathcal{A}_{in})) \times_r \mathit{nat\text{-}lit\text{-}rel}) \to$
      $\langle \mathit{nat\text{-}rel}\rangle\ \mathit{nres\text{-}rel}\rangle$
  ⟨*proof*⟩

lemma *mop-append-ll*:
‹(*uncurry2 mop-append-ll*, *uncurry2* (*RETURN ooo* (λ*W i x*. *W*(*i* := *W i* @ [*x*])))) ∈
[λ((*W*, *i*), *x*). *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) $\times_f$ *Id* $\times_f$ *Id* → ⟨⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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

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*, *NS*, *US*, *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 o snd o snd*) *S*) *K w*›
⟨*proof*⟩

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*›
⟨*proof*⟩

end
theory *IsaSAT-Watch-List-LLVM*
imports *IsaSAT-Watch-List IsaSAT-Literals-LLVM*
begin

type-synonym *watched-wl-uint32*
= ‹(*64*,(*64 word* × *32 word* × *1 word*),*64*)*array-array-list*›

abbreviation ‹*watcher-fast-assn* ≡ *sint64-nat-assn* $\times_a$ *unat-lit-assn* $\times_a$ *bool1-assn* ›

end
theory *IsaSAT-Lookup-Conflict*

```
  imports
    IsaSAT-Literals
    Watched-Literals.CDCL-Conflict-Minimisation
    LBD
    IsaSAT-Clauses
    IsaSAT-Watch-List
    IsaSAT-Trail
begin
```

Chapter 7

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 ∉# P› and ‹−L ∉# P› and
    ‹xs' = xs[atm-of L := Some (is-pos L)]›

lemma mset-as-position-distinct-mset:
  ‹mset-as-position xs P ⟹ distinct-mset P›
  ⟨proof⟩

lemma mset-as-position-atm-le-length:
  ‹mset-as-position xs P ⟹ L ∈# P ⟹ atm-of L < length xs›
  ⟨proof⟩

lemma mset-as-position-nth:
  ‹mset-as-position xs P ⟹ L ∈# P ⟹ xs ! (atm-of L) = Some (is-pos L)›
  ⟨proof⟩

lemma mset-as-position-in-iff-nth:
  ‹mset-as-position xs P ⟹ atm-of L < length xs ⟹ L ∈# P ⟷ xs ! (atm-of L) = Some (is-pos L)›
  ⟨proof⟩

lemma mset-as-position-tautology: ‹mset-as-position xs C ⟹ ¬tautology C›
  ⟨proof⟩

lemma mset-as-position-right-unique:
  assumes
    map: ‹mset-as-position xs D› and
    map': ‹mset-as-position xs D'›
  shows ‹D = D'›
⟨proof⟩

lemma mset-as-position-mset-union:
  fixes P xs
  defines ‹xs' ≡ fold (λL 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')›
  ⟨proof⟩

lemma mset-as-position-empty-iff: ‹mset-as-position xs {#} ⟷ (∃ n. xs = replicate n None)›
  ⟨proof⟩

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 𝒜 = {((n, xs), C). n = size C ∧ mset-as-position xs C ∧
   (∀ L∈atms-of (ℒ_all 𝒜). L < length xs)}›

lemma lookup-clause-rel-empty-iff: ‹((n, xs), C) ∈ lookup-clause-rel 𝒜 ⟹ n = 0 ⟷ C = {#}›
  ⟨proof⟩

lemma conflict-atm-le-length: ‹((n, xs), C) ∈ lookup-clause-rel 𝒜 ⟹ L ∈ atms-of (ℒ_all 𝒜) ⟹
   L < length xs›
  ⟨proof⟩


lemma conflict-le-length:
  assumes
```

c-rel: ‹$((n,\ xs),\ C) \in$ *lookup-clause-rel* $\mathcal{A}$› **and**
L-$\mathcal{L}_{all}$: ‹$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›
shows ‹*atm-of* $L <$ *length xs*›
⟨*proof*⟩

lemma *lookup-clause-rel-atm-in-iff*:
‹$((n,\ xs),\ C) \in$ *lookup-clause-rel* $\mathcal{A} \Longrightarrow L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow L \in\#\ C \longleftrightarrow xs!(\textit{atm-of}\ L) = \textit{Some}\ (\textit{is-pos}$ $L)$›
⟨*proof*⟩

lemma
assumes
c: ‹$((n,xs),\ C) \in$ *lookup-clause-rel* $\mathcal{A}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
lookup-clause-rel-not-tautolgy: ‹$\neg$*tautology* C› **and**
lookup-clause-rel-distinct-mset: ‹*distinct-mset* C› **and**
lookup-clause-rel-size: ‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}\ C \Longrightarrow \textit{size}\ C \leq 1 + \textit{uint32-max div 2}$›
⟨*proof*⟩

definition *option-bool-rel* :: ‹$(\textit{bool} \times {}'a\ \textit{option})\ \textit{set}$› **where**
‹*option-bool-rel* $= \{(b,\ x).\ b \longleftrightarrow \neg(\textit{is-None}\ x)\}$›

definition *NOTIN* :: ‹*bool option*› **where**
[*simp*]: ‹*NOTIN* = *None*›

definition *ISIN* :: ‹$\textit{bool} \Rightarrow \textit{bool option}$› **where**
[*simp*]: ‹*ISIN* b = *Some* b›

definition *is-NOTIN* :: ‹$\textit{bool option} \Rightarrow \textit{bool}$› **where**
[*simp*]: ‹*is-NOTIN* $x \longleftrightarrow x =$ *NOTIN*›

lemma *is-NOTIN-alt-def*:
‹*is-NOTIN* $x \longleftrightarrow$ *is-None* x›
⟨*proof*⟩

definition *option-lookup-clause-rel* **where**
‹*option-lookup-clause-rel* $\mathcal{A} = \{((b,(n,xs)),\ C).\ b = (C = \textit{None})\ \wedge$
$(C = \textit{None} \longrightarrow ((n,xs),\ \{\#\}) \in \textit{lookup-clause-rel}\ \mathcal{A})\ \wedge$
$(C \neq \textit{None} \longrightarrow ((n,xs),\ \textit{the}\ C) \in \textit{lookup-clause-rel}\ \mathcal{A})\}$
›

lemma *option-lookup-clause-rel-lookup-clause-rel-iff*:
‹$((\textit{False},\ (n,\ xs)),\ \textit{Some}\ C) \in \textit{option-lookup-clause-rel}\ \mathcal{A} \longleftrightarrow$
$((n,\ xs),\ C) \in \textit{lookup-clause-rel}\ \mathcal{A}$›
⟨*proof*⟩

type-synonym (in $-$**)** *conflict-option-rel* = ‹$\textit{bool} \times \textit{nat} \times \textit{bool option list}$›

definition (in $-$**)** *lookup-clause-assn-is-None* :: ‹$- \Rightarrow \textit{bool}$› **where**
‹*lookup-clause-assn-is-None* $= (\lambda(b,\ -,\ -).\ b)$›

lemma *lookup-clause-assn-is-None-is-None*:

‹(*RETURN o lookup-clause-assn-is-None*, *RETURN o is-None*) ∈
option-lookup-clause-rel $\mathcal{A} \rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *lookup-clause-assn-is-empty* :: ‹*-* ⇒ *bool*› **where**
‹*lookup-clause-assn-is-empty* = (λ(-, *n*, -). *n* = *0*)›

lemma *lookup-clause-assn-is-empty-is-empty*:
‹(*RETURN o lookup-clause-assn-is-empty*, *RETURN o* (λ*D*. *Multiset.is-empty*(*the D*))) ∈
[λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *size-lookup-conflict* :: ‹*-* ⇒ *nat*› **where**
‹*size-lookup-conflict* = (λ(-, *n*, -). *n*)›

definition *size-conflict-wl-heur* :: ‹*-* ⇒ *nat*› **where**
‹*size-conflict-wl-heur* = (λ(*M*, *N*, *U*, *D*, -, -, -, -). *size-lookup-conflict D*)›

lemma (**in** −) *mset-as-position-length-not-None*:
‹*mset-as-position x2 C* ⟹ *size C* = *length* (*filter* ((≠) *None*) *x2*)›
⟨*proof*⟩

definition (**in** −) *is-in-lookup-conflict* **where**
‹*is-in-lookup-conflict* = (λ(*n*, *xs*) *L*. ¬*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*⟩

lemma *mset-as-position-remove2*:
‹*mset-as-position xs D* ⟹ *atm-of L* < *length xs* ⟹
mset-as-position (*xs*[*atm-of L* := *None*]) (*D* − {#*L*, −*L*#})›
⟨*proof*⟩

definition (**in** −) *delete-from-lookup-conflict*
:: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel nres*› **where**
‹*delete-from-lookup-conflict* = (λ*L* (*n*, *xs*). *do* {
    ASSERT(*n*≥*1*);
    ASSERT(*atm-of L* < *length xs*);
    RETURN (*n* − *1*, *xs*[*atm-of L* := *None*])
  })›

lemma *delete-from-lookup-conflict-op-mset-delete*:
‹(*uncurry delete-from-lookup-conflict*, *uncurry* (*RETURN oo remove1-mset*)) ∈
[λ(*L*, *D*). −*L* ∉# *D* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *L* ∈# *D*]$_f$ *Id* $\times_f$ *lookup-clause-rel* $\mathcal{A}$ →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *delete-from-lookup-conflict-pre* **where**
‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ = (λ(*a*, *b*). − *a* ∉# *b* ∧ *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *a* ∈# *b*)›

definition *set-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒

out-learned ⇒ (*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*set-conflict-m M N i - - -* =
  SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*N*∝*i*)) ∧ *n* = *card-max-lvl M* (*mset* (*N*∝*i*)) ∧
   out-learned M C outl)›

definition *merge-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*merge-conflict-m M N i D - -* =
  SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
    n = *card-max-lvl M* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
    out-learned M C outl)›

definition *merge-conflict-m-g*
:: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*merge-conflict-m-g init M Ni D* =
  SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
    n = *card-max-lvl M* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
    out-learned M C outl)›

definition *add-to-lookup-conflict* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*add-to-lookup-conflict* = (λ*L* (*n*, *xs*). (*if xs* ! *atm-of L* = *NOTIN then n* + *1 else n*,
  xs[*atm-of L* := *ISIN* (*is-pos L*)]))›

definition *lookup-conflict-merge′-step*
:: ‹*nat multiset* ⇒ *nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *nat* ⇒ *lookup-clause-rel* ⇒ *nat clause-l* ⇒
  nat clause ⇒ *out-learned* ⇒ *bool*›
where
‹*lookup-conflict-merge′-step* $\mathcal{A}$ *init M i clvls zs D C outl* = (
  let D′ = *mset* (*take* (*i* − *init*) (*drop init D*));
    E = *remdups-mset* (*D′* + *C*) *in*
  ((*False*, *zs*), *Some E*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
  out-learned M (*Some E*) *outl* ∧
  literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ *E* ∧ *clvls* = *card-max-lvl M E*)›

lemma *option-lookup-clause-rel-update-None*:
assumes ‹((*False*, (*n*, *xs*)), *Some D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and** *L-xs* : ‹*L* < *length xs*›
shows ‹((*False*, (*if xs*!*L* = *None then n else n* − *1*, *xs*[*L* := *None*])),
  Some (*D* − {# *Pos L*, *Neg L* #})) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

lemma *add-to-lookup-conflict-lookup-clause-rel*:
assumes
  confl: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
  uL-C: ‹−*L* ∉# *C*› **and**
  L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹(*add-to-lookup-conflict L* (*n*, *xs*), {#*L*#} ∪# *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

```
definition outlearned-add
  :: ‹(nat,nat)ann-lits ⇒ nat literal ⇒ nat × bool option list ⇒ out-learned ⇒ out-learned› where
  ‹outlearned-add = (λM 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 = (λM 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 ⇒
        out-learned ⇒ (conflict-option-rel × nat × out-learned) nres›
where
  ‹lookup-conflict-merge init M D  = (λ(b, xs) clvls outl. do {
   (-, clvls, zs, outl) ← WHILE_T^λ(i::nat, clvls :: nat, zs, outl).          length (snd zs) = length (snd xs) ∧                Suc i ≤ uin
       (λ(i :: nat, clvls, zs, outl). i < length-uint32-nat D)
       (λ(i :: nat, clvls, zs, outl). do {
           ASSERT(i < length-uint32-nat D);
           ASSERT(Suc i ≤ uint32-max);
           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, outl)
        })
       (init, clvls, xs, outl);
     RETURN ((False, zs), clvls, outl)
   })›
```

```
definition resolve-lookup-conflict-aa
  :: ‹(nat,nat)ann-lits ⇒ nat clauses-l ⇒ nat ⇒ conflict-option-rel ⇒ nat ⇒
     out-learned ⇒ (conflict-option-rel × nat × out-learned) nres›
where
  ‹resolve-lookup-conflict-aa M N i xs clvls outl =
     lookup-conflict-merge 1 M (N ∝ i) xs clvls outl›
```

```
definition set-lookup-conflict-aa
  :: ‹(nat,nat)ann-lits ⇒ nat clauses-l ⇒ nat ⇒ conflict-option-rel ⇒ nat ⇒
  out-learned ⇒(conflict-option-rel × nat × out-learned) nres›
where
  ‹set-lookup-conflict-aa M C i xs clvls outl =
     lookup-conflict-merge 0 M (C∝i) xs clvls outl›
```

```
definition isa-outlearned-add
  :: ‹trail-pol ⇒ nat literal ⇒ nat × bool option list ⇒ out-learned ⇒ out-learned› where
  ‹isa-outlearned-add = (λM 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 𝒜 ⟹ L ∈# ℒ_all 𝒜 ⟹
      isa-outlearned-add M' L zs outl = outlearned-add M L zs outl›
```

⟨*proof*⟩

definition *isa-clvls-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*isa-clvls-add* = (λ*M L zs clvls*.
(*if get-level-pol M L* = *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
else clvls))›

lemma *isa-clvls-add-clvls-add*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-clvls-add M' L zs outl = *clvls-add M L zs outl*›
⟨*proof*⟩

definition *isa-lookup-conflict-merge*
:: ‹*nat* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*›
where

```
‹isa-lookup-conflict-merge init M N i  = (λ(b, xs) clvls outl. do {
    ASSERT( arena-is-valid-clause-idx N i);
  (-, clvls, zs, outl) ← $WHILE_T^{λ(i::nat, clvls :: nat, zs, outl).            length (snd zs) = length (snd xs) ∧                    Suc (fst zs)}$
      (λ(j :: nat, clvls, zs, outl). j < i + arena-length N i)
      (λ(j :: nat, clvls, zs, 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));
          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, outl)
       })
      (i+init, clvls, xs, outl);
    RETURN ((False, zs), clvls, outl)
  })›
```

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 outl* ≤ ⇓ *Id*
(*lookup-conflict-merge init M* (*N* ∝ *i*) (*b*, *xs*) *clvls outl*)›
⟨*proof*⟩

lemma (**in** −) *arena-is-valid-clause-idx-le-uint64-max*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *uint64-max* ⟹
bd + *arena-length be bd* ≤ *uint64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *uint64-max* ⟹
bd ≤ *uint64-max*›
⟨*proof*⟩

definition *isa-set-lookup-conflict-aa* **where**
‹*isa-set-lookup-conflict-aa = isa-lookup-conflict-merge 0*›

definition *isa-set-lookup-conflict-aa-pre* **where**
‹*isa-set-lookup-conflict-aa-pre =*
$(\lambda(((((M, N), i), (\text{-}, xs)), \text{-}), out).\ i < \mathit{length}\ N)$›

lemma *lookup-conflict-merge′-spec*:
assumes
o: ‹$((b, n, xs), \mathit{Some}\ C) \in \mathit{option\text{-}lookup\text{-}clause\text{-}rel}\ \mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\ \mathcal{A}\ (\mathit{mset}\ D)$› **and**
tauto: ‹$\neg\mathit{tautology}\ (\mathit{mset}\ D)$› **and**
lits-C: ‹$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\ \mathcal{A}\ C$› **and**
‹*clvls = card-max-lvl M C*› **and**
CD: ‹$\bigwedge L.\ L \in \mathit{set}\ (\mathit{drop}\ \mathit{init}\ D) \Longrightarrow -L \notin\#\ C$› **and**
‹$\mathit{Suc}\ \mathit{init} \le \mathit{uint32\text{-}max}$› **and**
‹*out-learned M* (*Some C*) *outl*› **and**
bounded: ‹$\mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}$›
shows
‹*lookup-conflict-merge init M D* (*b*, *n*, *xs*) *clvls outl* $\le$
$\Downarrow(\mathit{option\text{-}lookup\text{-}clause\text{-}rel}\ \mathcal{A} \times_r \mathit{Id} \times_r \mathit{Id})$
(*merge-conflict-m-g init M D* (*Some C*))›
(**is** ‹- $\le$ $\Downarrow$ *?Ref ?Spec*›)
⟨*proof*⟩

lemma $\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\mathit{\text{-}mm\text{-}literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}$:
assumes *lits*: ‹$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\mathit{\text{-}mm}\ \mathcal{A}\ (\mathit{mset}\ `\#\ \mathit{ran\text{-}mf}\ N)$› **and**
i: ‹$i \in\#\ \mathit{dom\text{-}m}\ N$›
shows ‹$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\ \mathcal{A}\ (\mathit{mset}\ (N \propto i))$›
⟨*proof*⟩

lemma *isa-set-lookup-conflict*:
‹(*uncurry5 isa-set-lookup-conflict-aa*, *uncurry5 set-conflict-m*) $\in$
$[\lambda(((((M, N), i), xs), \mathit{clvls}), \mathit{outl}).\ i \in\#\ \mathit{dom\text{-}m}\ N \land xs = \mathit{None} \land \mathit{distinct}\ (N \propto i) \land$
$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\mathit{\text{-}mm}\ \mathcal{A}\ (\mathit{mset}\ `\#\ \mathit{ran\text{-}mf}\ N) \land$
$\neg\mathit{tautology}\ (\mathit{mset}\ (N \propto i)) \land \mathit{clvls} = 0 \land$
$\mathit{out\text{-}learned}\ M\ \mathit{None}\ \mathit{outl} \land$
$\mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}]_f$
$\mathit{trail\text{-}pol}\ \mathcal{A} \times_f \{(\mathit{arena}, N).\ \mathit{valid\text{-}arena}\ \mathit{arena}\ N\ \mathit{vdom}\} \times_f \mathit{nat\text{-}rel} \times_f$
$\mathit{option\text{-}lookup\text{-}clause\text{-}rel}\ \mathcal{A} \times_f \mathit{nat\text{-}rel}\ \times_f \mathit{Id} \rightarrow$
$\langle\mathit{option\text{-}lookup\text{-}clause\text{-}rel}\ \mathcal{A} \times_r \mathit{nat\text{-}rel} \times_r \mathit{Id}\rangle\mathit{nres\text{-}rel}$›
⟨*proof*⟩

definition *merge-conflict-m-pre* **where**
‹$\mathit{merge\text{-}conflict\text{-}m\text{-}pre}\ \mathcal{A} =$
$(\lambda((((((M, N), i), xs), \mathit{clvls}), \mathit{out}).\ i \in\#\ \mathit{dom\text{-}m}\ N \land xs \ne \mathit{None} \land \mathit{distinct}\ (N \propto i) \land$
$\neg\mathit{tautology}\ (\mathit{mset}\ (N \propto i)) \land$
$(\forall L \in \mathit{set}\ (\mathit{tl}\ (N \propto i)).\ -\ L \notin\#\ \mathit{the}\ xs) \land$
$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\ \mathcal{A}\ (\mathit{the}\ xs) \land \mathit{clvls} = \mathit{card\text{-}max\text{-}lvl}\ M\ (\mathit{the}\ xs) \land$
$\mathit{out\text{-}learned}\ M\ xs\ \mathit{out} \land \mathit{no\text{-}dup}\ M \land$
$\mathit{literals\text{-}are\text{-}in\text{-}}\mathcal{L}_{in}\mathit{\text{-}mm}\ \mathcal{A}\ (\mathit{mset}\ `\#\ \mathit{ran\text{-}mf}\ N) \land$
$\mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A})$›

definition *isa-resolve-merge-conflict-gt2* **where**
‹*isa-resolve-merge-conflict-gt2* = *isa-lookup-conflict-merge 1*›

lemma *isa-resolve-merge-conflict-gt2*:
‹(*uncurry5 isa-resolve-merge-conflict-gt2*, *uncurry5 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* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

lemma *lookup-clause-rel-exists-le-uint32-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* ≥ *i* ∧ *j* < *length xs* ∧ *j* < *uint32-max* ∧ *xs* ! *j* ≠ *None*›
⟨*proof*⟩

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**
‹$\mathit{iterate\text{-}over\text{-}conflict\text{-}inv}\ M\ D_0' = (\lambda(D, D').\ D \subseteq\#\ D_0' \wedge D' \subseteq\#\ D)$›

definition *is-literal-redundant-spec* **where**
‹$\mathit{is\text{-}literal\text{-}redundant\text{-}spec}\ K\ NU\ UNE\ D\ L = SPEC(\lambda b.\ b \longrightarrow$
$NU + UNE \models pm\ \mathit{remove1\text{-}mset}\ L\ (\mathit{add\text{-}mset}\ K\ D))$›

definition *iterate-over-conflict*
:: ‹$'v\ literal \Rightarrow ('v, 'mark)\ ann\text{-}lits \Rightarrow 'v\ clauses \Rightarrow 'v\ clauses \Rightarrow\ 'v\ clause \Rightarrow$
$'v\ clause\ nres$›
where

```
‹iterate-over-conflict K M NU UNE $D_0'$ = do {
  $(D, \text{-}) \leftarrow$
     $WHILE_T{}^{iterate\text{-}over\text{-}conflict\text{-}inv\ M\ D_0'}$
     $(\lambda(D, D').\ D' \neq \{\#\})$
     $(\lambda(D, D').\ do\{$
        $x \leftarrow SPEC\ (\lambda x.\ x \in\#\ D')$;
        $red \leftarrow is\text{-}literal\text{-}redundant\text{-}spec\ K\ NU\ UNE\ D\ x$;
        if $\neg red$
        then RETURN $(D, remove1\text{-}mset\ x\ D')$
        else RETURN $(remove1\text{-}mset\ x\ D, remove1\text{-}mset\ x\ D')$
      })
     $(D_0', D_0')$;
    RETURN D
}›
```

definition *minimize-and-extract-highest-lookup-conflict-inv* **where**
‹$\mathit{minimize\text{-}and\text{-}extract\text{-}highest\text{-}lookup\text{-}conflict\text{-}inv} = (\lambda(D, i, s, outl).$
$length\ outl \leq uint32\text{-}max \wedge mset\ (tl\ outl) = D \wedge outl \neq [] \wedge i \geq 1)$›

type-synonym *'v conflict-highest-conflict* = ‹$('v\ literal \times nat)\ option$›

definition (**in** −) *atm-in-conflict* **where**
‹$\mathit{atm\text{-}in\text{-}conflict}\ L\ D \longleftrightarrow L \in atms\text{-}of\ D$›

definition *atm-in-conflict-lookup* :: ‹$nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool$› **where**
‹$\mathit{atm\text{-}in\text{-}conflict\text{-}lookup} = (\lambda L\ (\text{-}, xs).\ xs\ !\ L \neq None)$›

definition *atm-in-conflict-lookup-pre* :: ‹$nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool$› **where**
‹$\mathit{atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre}\ L\ xs \longleftrightarrow L < length\ (snd\ xs)$›

lemma *atm-in-conflict-lookup-atm-in-conflict*:
‹$(uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict\text{-}lookup), uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict)) \in$
$[\lambda(L, xs).\ L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})]_f\ Id \times_f lookup\text{-}clause\text{-}rel\ \mathcal{A} \rightarrow \langle bool\text{-}rel\rangle nres\text{-}rel$›
⟨*proof*⟩

lemma *atm-in-conflict-lookup-pre*:
fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
assumes
‹$x1n \in\#\ \mathcal{L}_{all}\ \mathcal{A}$› **and**
‹$(x2f, x2a) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}$›
shows ‹$atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre\ (atm\text{-}of\ x1n)\ x2f$›
⟨*proof*⟩

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*))›
⟨*proof*⟩

abbreviation *ana-lookups-rel* **where**
‹*ana-lookups-rel NU* ≡ ⟨*ana-lookup-rel NU*⟩*list-rel*›

definition *ana-lookup-conv* :: ‹*nat clauses-l* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv NU* = (λ(*C*, *i*, *b*). (*C*, (*if b then 1 else 0*), *i*, (*if b then 1 else length* (*NU* ∝ *C*))))›

definition *get-literal-and-remove-of-analyse-wl2*
:: ‹′*v clause-l* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ ′*v literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*get-literal-and-remove-of-analyse-wl2 C analyse* =
(*let* (*i*, *j*, *b*) = *last analyse in*
(*C* ! *j*, *analyse*[*length analyse* − *1* := (*i*, *j* + *1*, *b*)]))›

definition *lit-redundant-rec-wl-inv2* **where**
‹*lit-redundant-rec-wl-inv2 M NU D* =
(λ(*cach*, *analyse*, *b*). ∃ *analyse′*. (*analyse*, *analyse′*) ∈ *ana-lookups-rel NU* ∧

```
    lit-redundant-rec-wl-inv M NU D (cach, analyse′, b))›

definition mark-failed-lits-stack-inv2 where
  ‹mark-failed-lits-stack-inv2 NU analyse = (λcach.
      ∃ 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 𝒜 M NU D cach analysis lbd =
      WHILE_T^{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 ∝ C ! k ∈ lits-of-l M);
            ASSERT(NU ∝ C ! k ∈# ℒ_all 𝒜);
    ASSERT(literals-are-in-ℒ_in 𝒜 (mset (NU ∝ C)));
    ASSERT(length (NU ∝ C) ≤ Suc (uint32-max div 2));
    ASSERT(len ≤ length (NU ∝ C)); — makes the refinement easier
            let C = NU ∝ C;
            if i ≥ len
            then
              RETURN(cach (atm-of (C ! k) := SEEN-REMOVABLE), butlast analyse, True)
            else do {
              let (L, analyse) = get-literal-and-remove-of-analyse-wl2 C analyse;
              ASSERT(L ∈# ℒ_all 𝒜);
              let b = ¬level-in-lbd (get-level M L) lbd;
              if (get-level M L = 0 ∨
                  conflict-min-cach cach (atm-of L) = SEEN-REMOVABLE ∨
                  atm-in-conflict (atm-of L) D)
              then RETURN (cach, analyse, False)
              else if b ∨ conflict-min-cach cach (atm-of L) = SEEN-FAILED
              then do {
                 ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
                 cach ← mark-failed-lits-wl NU analyse cach;
                 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(length (NU ∝ C) ≤ Suc (uint32-max div 2));
      RETURN (cach, analyse @ [lit-redundant-reason-stack2 (−L) NU C], False)
    }
                 | None ⇒ do {
                     ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
```

```
                    cach ← mark-failed-lits-wl NU analyse cach;
                    RETURN (cach, [], False)
                }
            }
        }
     })
    (cach, analysis, False)⟩

lemma lit-redundant-rec-wl-ref-butlast:
  ⟨lit-redundant-rec-wl-ref NU x ⟹ lit-redundant-rec-wl-ref NU (butlast x)⟩
  ⟨proof⟩

lemma lit-redundant-rec-wl-lookup-mark-failed-lits-stack-inv:
  assumes
    ⟨(x, x′) ∈ Id⟩ and
    ⟨case x of (cach, analyse, b) ⇒ analyse ≠ []⟩ and
    ⟨lit-redundant-rec-wl-inv M NU D x′⟩ and
    ⟨¬ snd (snd (snd (last x1a))) ≤ fst (snd (snd (last x1a)))⟩ and
    ⟨get-literal-and-remove-of-analyse-wl (NU ∝ fst (last x1c)) x1c = (x1e, x2e)⟩ and
    ⟨x2 = (x1a, x2a)⟩ and
    ⟨x′ = (x1, x2)⟩ and
    ⟨x2b = (x1c, x2c)⟩ and
    ⟨x = (x1b, x2b)⟩
  shows ⟨mark-failed-lits-stack-inv NU x2e x1b⟩
⟨proof⟩

context
  fixes M D 𝒜 NU analysis analysis′
  assumes
    M-D: ⟨M ⊨as CNot D⟩ and
    n-d: ⟨no-dup M⟩ and
    lits: ⟨literals-are-in-ℒin-trail 𝒜 M⟩ and
    ana: ⟨(analysis, analysis′) ∈ ana-lookups-rel NU⟩ and
    lits-NU: ⟨literals-are-in-ℒin-mm 𝒜 ((mset ∘ fst) ‘# ran-m NU)⟩ and
    bounded: ⟨isasat-input-bounded 𝒜⟩
begin
lemma ccmin-rel:
  assumes ⟨lit-redundant-rec-wl-inv M NU D (cach, analysis′, False)⟩
  shows ⟨((cach, analysis, False), cach, analysis′, False)
        ∈ {((cach, ana, b), cach′, ana′, b′).
          (ana, ana′) ∈ ana-lookups-rel NU ∧
          b = b′ ∧ cach = cach′ ∧ lit-redundant-rec-wl-inv M NU D (cach, ana′, b)}⟩
⟨proof⟩


context
  fixes x :: ⟨(nat ⇒ minimize-status) × (nat × nat × bool) list × bool⟩ and
  x′ :: ⟨(nat ⇒ minimize-status) × (nat × nat × nat × nat) list × bool⟩
  assumes x-x′: ⟨(x, x′) ∈ {((cach, ana, b), (cach′, ana′, b′)).
    (ana, ana′) ∈ ana-lookups-rel NU ∧ b = b′ ∧ cach = cach′ ∧
    lit-redundant-rec-wl-inv M NU D (cach, ana′, b)}⟩
begin

lemma ccmin-lit-redundant-rec-wl-inv2:
  assumes ⟨lit-redundant-rec-wl-inv M NU D x′⟩
  shows ⟨lit-redundant-rec-wl-inv2 M NU D x⟩
```

⟨*proof*⟩

context
assumes
‹*lit-redundant-rec-wl-inv2 M NU D x*› **and**
‹*lit-redundant-rec-wl-inv M NU D x′*›
begin

lemma *ccmin-cond*:
fixes *x1* :: ‹*nat* ⇒ *minimize-status*› **and**
x2 :: ‹(*nat* × *nat* × *bool*) *list* × *bool*› **and**
x1a :: ‹(*nat* × *nat* × *bool*) *list*› **and**
x2a :: ‹*bool*› **and** *x1b* :: ‹*nat* ⇒ *minimize-status*› **and**
x2b :: ‹(*nat* × *nat* × *nat* × *nat*) *list* × *bool*› **and**
x1c :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and** *x2c* :: ‹*bool*›
assumes
‹*x2* = (*x1a*, *x2a*)›
‹*x* = (*x1*, *x2*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x′* = (*x1b*, *x2b*)›
shows ‹(*x1a* ≠ []) = (*x1c* ≠ [])›
⟨*proof*⟩

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 ≠ []›
  ⟨proof⟩

lemma ccmin-nempty:
  shows ‹x1c ≠ []›
  ⟨proof⟩

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›
  ⟨proof⟩ definition j where
  ‹j = fst (snd (last x1c))›

private definition b where
  ‹b = snd (snd (last x1c))›

private lemma last-x1c[simp]:
  ‹last x1c = (x1d, x1f, b)›
  ⟨proof⟩ lemma
  ana: ‹(x1d, (if b then 1 else 0), x1f, (if b then 1 else length (NU ∝ x1d))) = (x1d, x1e, x1f, x2i)› and
  st3:
    ‹x1e = (if b then 1 else 0)›
    ‹x1f = j›
    ‹x2f = (if b then 1 else length (NU ∝ x1d))›
    ‹x2d = (if b then 1 else 0, j, if b then 1 else length (NU ∝ x1d))› and
    ‹j ≤ (if b then 1 else length (NU ∝ x1d))› and
    ‹x1d ∈# dom-m NU› and
    ‹0 < x1d› and
    ‹(if b then 1 else length (NU ∝ x1d)) ≤ length (NU ∝ x1d)› and
    ‹(if b then 1 else 0) < length (NU ∝ x1d)› and
```

dist: ‹*distinct* (*NU* ∝ *x1d*)› **and**
tauto: ‹¬ *tautology* (*mset* (*NU* ∝ *x1d*))›
⟨*proof*⟩

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
⟨*proof*⟩

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
⟨*proof*⟩

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
⟨*proof*⟩

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›
⟨*proof*⟩

lemma *ccmin-le-uint32-max*:
‹*length* (*NU* ∝ *x1g*) ≤ *Suc* (*uint32-max div 2*)›
⟨*proof*⟩

lemma *ccmin-in-all-lits*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
⟨*proof*⟩

lemma *ccmin-less-length*:
shows ‹*x2i* ≤ *length* (*NU* ∝ *x1g*)›
⟨*proof*⟩

lemma *ccmin-same-cond*:
shows ‹(*x2i* ≤ *x1i*) = (*x2f* ≤ *x1f*)›
⟨*proof*⟩

lemma *list-rel-butlast*:
assumes *rel*: ‹(*xs*, *ys*) ∈ ⟨*R*⟩*list-rel*›
shows ‹(*butlast xs*, *butlast ys*) ∈ ⟨*R*⟩*list-rel*›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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*›
  ⟨*proof*⟩ **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*›
  ⟨*proof*⟩ **lemma** *x1k-x1j*: ‹*x1k* = *x1j*› ‹*x1j* = *NU* ∝ *x1d* ! *x1f*› **and**
  x2k-x2j: ‹(*x2k*, *x2j*) ∈ *ana-lookups-rel NU*›
  ⟨*proof*⟩

lemma *ccmin-x1k-all*:
  shows ‹*x1k* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
  ⟨*proof*⟩

context
  notes -[*simp*]= *x1k-x1j*
  fixes *b* :: ‹*bool*› **and** *lbd*
  assumes *b*: ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd*, *b*) ∈ *bool-rel*›
begin

private lemma *in-conflict-atm-in*:
  ‹− *x1e*′ ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e*′) *D* ⟷ *x1e*′ ∈# *D*› **for** *x1e*′
  ⟨*proof*⟩

lemma *ccmin-already-seen*:
  shows ‹(*get-level M x1k* = *0* ∨
        conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
        atm-in-conflict (*atm-of x1k*) *D*) =
       (*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
  ⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
    (*x1*, *x2j*, *False*)›
  ⟨*proof*⟩

lemma *ccmin-already-seen-rel*:
  assumes
    ‹*get-level M x1k* = *0* ∨
     conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
     atm-in-conflict (*atm-of x1k*) *D*› **and**
    ‹*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*›
  shows ‹((*x1b*, *x2k*, *False*), *x1*, *x2j*, *False*)
       ∈ {((*cach*, *ana*, *b*), *cach*′, *ana*′, *b*′).
        (*ana*, *ana*′) ∈ *ana-lookups-rel NU* ∧
        b = *b*′ ∧ *cach* = *cach*′ ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana*′, *b*)}›
  ⟨*proof*⟩

```
context
  assumes
    ‹¬ (get-level M x1k = 0 ∨
        conflict-min-cach x1b (atm-of x1k) = SEEN-REMOVABLE ∨
        atm-in-conflict (atm-of x1k) D)› and
    ‹¬ (get-level M x1j = 0 ∨ x1 (atm-of x1j) = SEEN-REMOVABLE ∨ x1j ∈# D)›
begin
lemma ccmin-already-failed:
  shows ‹(¬ level-in-lbd (get-level M x1k) lbd ∨
          conflict-min-cach x1b (atm-of x1k) = SEEN-FAILED) =
         (b ∨ x1 (atm-of x1j) = SEEN-FAILED)›
  ⟨proof⟩


context
  assumes
    ‹¬ level-in-lbd (get-level M x1k) lbd ∨
     conflict-min-cach x1b (atm-of x1k) = SEEN-FAILED› and
    ‹b ∨ x1 (atm-of x1j) = SEEN-FAILED›
begin

lemma ccmin-mark-failed-lits-stack-inv2-lbd:
  shows ‹mark-failed-lits-stack-inv2 NU x2k x1b›
  ⟨proof⟩

lemma ccmin-mark-failed-lits-wl-lbd:
  shows ‹mark-failed-lits-wl NU x2k x1b
         ≤ ⇓ Id
            (mark-failed-lits-wl NU x2j x1)›
  ⟨proof⟩


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)}›
  ⟨proof⟩

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›
  ⟨proof⟩

lemma ccmin-lit-eq:
  ‹− x1k = − x1j›
  ⟨proof⟩


context
  fixes xa :: ‹nat option› and x'a :: ‹nat option›
  assumes xa-x'a: ‹(xa, x'a) ∈ ⟨nat-rel⟩option-rel›
begin

lemma ccmin-lit-eq2:
  ‹(xa, x'a) ∈ Id›
  ⟨proof⟩

context
  assumes
    [simp]: ‹xa = None› ‹x'a = None›
begin

lemma ccmin-mark-failed-lits-stack-inv2-dec:
  ‹mark-failed-lits-stack-inv2 NU x2k x1b›
  ⟨proof⟩

lemma ccmin-mark-failed-lits-stack-wl-dec:
  shows ‹mark-failed-lits-wl NU x2k x1b
         ≤ ⇓ Id
            (mark-failed-lits-wl NU x2j x1)›
  ⟨proof⟩


lemma ccmin-rel-dec:
  fixes cach :: ‹nat ⇒ minimize-status› and cacha :: ‹nat ⇒ minimize-status›
  assumes ‹(cach, cacha)  ∈ Id›
  shows ‹((cach, [], False), cacha, [], False)
        ∈  {((cach, ana, b), cach', ana', b').
       (ana, ana') ∈ ana-lookups-rel NU ∧
       b = b' ∧ cach = cach' ∧ lit-redundant-rec-wl-inv M NU D (cach, ana', b)}›
  ⟨proof⟩

end


context
  fixes xb :: ‹nat› and x'b :: ‹nat›
  assumes H:
    ‹xa = Some xb›
    ‹x'a = Some x'b›
    ‹(xb, x'b) ∈ nat-rel›
    ‹x'b ∈# dom-m NU›
    ‹2 ≤ length (NU ∝ x'b)›
    ‹x'b > 0›
    ‹distinct (NU ∝ x'b) ∧ ¬ tautology (mset (NU ∝ x'b))›
begin

lemma ccmin-stack-pre:
  shows ‹xb ∈# dom-m NU› ‹2 ≤ length (NU ∝ xb)›
  ⟨proof⟩
```

lemma *ccmin-literals-are-in-$\mathcal{L}_{in}$-NU-xb*:
shows ‹*literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$* (*mset* (*NU* ∝ *xb*))›
⟨*proof*⟩

lemma *ccmin-le-uint32-max-xb*:
‹*length* (*NU* ∝ *xb*) ≤ *Suc* (*uint32-max div 2*)›
⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv3*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j* @ [*lit-redundant-reason-stack* (− *NU* ∝ *x1d* ! *x1f*) *NU x′b*], *False*)›
⟨*proof*⟩

lemma *ccmin-stack-rel*:
shows ‹((*x1b*, *x2k* @ [*lit-redundant-reason-stack2* (− *x1k*) *NU xb*], *False*), *x1*,
x2j @ [*lit-redundant-reason-stack* (− *x1j*) *NU x′b*], *False*)
∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
(*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
⟨*proof*⟩

end
end
end
end
end
end
end
end
end
end
end
end

lemma *lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*:
assumes
M-D: ‹*M* ⊨*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*⟩

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-$\mathcal{L}_{in}$ $\mathcal{A}$ (mset (NU ∝ C)));
      ASSERT(distinct (NU ∝ C) ∧ ¬tautology (mset (NU ∝ C)));
      ASSERT(length (NU ∝ C) ≤ Suc (uint32-max div 2));
      lit-redundant-rec-wl-lookup $\mathcal{A}$ M NU D cach [lit-redundant-reason-stack2 (−L) NU C] lbd
  }
        | None ⇒ do {
            RETURN (cach, [], False)
        }
      }
  }›
```

lemma *literal-redundant-wl-lookup-literal-redundant-wl*:
  assumes ‹*M* ⊨*as CNot D*› ‹*no-dup M*› ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
    ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**
    ‹*isasat-input-bounded* $\mathcal{A}$›
  shows
    ‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* ≤
      ⇓ (*Id* $\times_f$ (*ana-lookups-rel NU* $\times_f$ *bool-rel*)) (*literal-redundant-wl M NU D cach L lbd*)›
⟨*proof*⟩

definition (**in** −) *lookup-conflict-nth* **where**
  [*simp*]: ‹*lookup-conflict-nth* = (λ(-, *xs*) *i*. *xs* ! *i*)›

definition (**in** −) *lookup-conflict-size* **where**
  [*simp*]: ‹*lookup-conflict-size* = (λ(*n*, *xs*). *n*)›

definition (**in** −) *lookup-conflict-upd-None* **where**
  [*simp*]: ‹*lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. (*n*−*1*, *xs* [*i* :=*None*]))›

definition *minimize-and-extract-highest-lookup-conflict*
  :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒ (*nat* ⇒ *minimize-status*) ⇒ *lbd*
⇒
    out-learned ⇒ (*nat clause* × (*nat* ⇒ *minimize-status*) × *out-learned*) *nres*›
where

```
  ‹minimize-and-extract-highest-lookup-conflict $\mathcal{A}$ = (λM NU nxs s lbd outl. do {
    (D, -, s, outl) ←
        $WHILE_T^{minimize-and-extract-highest-lookup-conflict-inv}$
          (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, 1, s, outl);
    RETURN (D, s, outl)
  })⟩

lemma entails-uminus-filter-to-poslev-can-remove:
  assumes NU-uL-E: ⟨NU ⊨p add-mset (− L) (filter-to-poslev M′ L E)⟩ and
     NU-E: ⟨NU ⊨p E⟩ and L-E: ⟨L ∈# E⟩
   shows ⟨NU ⊨p remove1-mset L E⟩
⟨proof⟩

lemma minimize-and-extract-highest-lookup-conflict-iterate-over-conflict:
  fixes D :: ⟨nat clause⟩ and S′ :: ⟨nat twl-st-l⟩ and NU :: ⟨nat clauses-l⟩ and S :: ⟨nat twl-st-wl⟩
     and S′′ :: ⟨nat twl-st⟩
   defines
    ⟨S′′′ ≡ state_W-of S′′⟩
  defines
    ⟨M ≡ get-trail-wl S⟩ and
    NU: ⟨NU ≡ get-clauses-wl S⟩ and
    NU′-def: ⟨NU′ ≡ mset ‘# ran-mf NU⟩ and
    NUE: ⟨NUE ≡ get-unit-learned-clss-wl S + get-unit-init-clss-wl S⟩ and
    NUS: ⟨NUS ≡ get-subsumed-learned-clauses-wl S + get-subsumed-init-clauses-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-ℒ_in-trail 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 + NUS) D⟩ and
    NU-P-D: ⟨NU′ + NUE + NUS ⊨pm add-mset K D⟩ and
    lits-D: ⟨literals-are-in-ℒ_in 𝒜 D⟩ and
    lits-NU: ⟨literals-are-in-ℒ_in-mm 𝒜 (mset ‘# ran-mf NU)⟩ and
    K: ⟨K = outl ! 0⟩ and
    outl-nempty: ⟨outl ≠ []⟩ and
    bounded: ⟨isasat-input-bounded 𝒜⟩
  shows
    ⟨minimize-and-extract-highest-lookup-conflict 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 + NUS) D)⟩
    (is ⟨- ≤ ⇓ ?R -⟩)
⟨proof⟩

definition cach-refinement-list
  :: ⟨nat multiset ⇒ (minimize-status list × (nat conflict-min-cach)) set⟩
where
  ⟨cach-refinement-list 𝒜_in = ⟨Id⟩map-fun-rel {(a, a′). a = a′ ∧ a ∈# 𝒜_in}⟩

definition cach-refinement-nonull
  :: ⟨nat multiset ⇒ ((minimize-status list × nat list) × minimize-status list) set⟩
where
  ⟨cach-refinement-nonull 𝒜 = {((cach, support), cach′). cach = cach′ ∧
       (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟷ L ∈ set support) ∧
```

($\forall$ *L* $\in$ *set support*. *L* < *length cach*) $\wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}$}›

definition *cach-refinement*
:: ‹*nat multiset* $\Rightarrow$ ((*minimize-status list* $\times$ *nat list*) $\times$ (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement* $\mathcal{A}_{in}$ = *cach-refinement-nonull* $\mathcal{A}_{in}$ *O* *cach-refinement-list* $\mathcal{A}_{in}$›

lemma *cach-refinement-alt-def*:
‹*cach-refinement* $\mathcal{A}_{in}$ = {((*cach*, *support*), *cach'*).
($\forall$ *L* < *length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longleftrightarrow$ *L* $\in$ *set support*) $\wedge$
($\forall$ *L* $\in$ *set support*. *L* < *length cach*) $\wedge$
($\forall$ *L* $\in\#$ $\mathcal{A}_{in}$. *L* < *length cach* $\wedge$ *cach* ! *L* = *cach' L*) $\wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}_{in}$}›
⟨*proof*⟩

lemma *in-cach-refinement-alt-def*:
‹((*cach*, *support*), *cach'*) $\in$ *cach-refinement* $\mathcal{A}_{in}$ $\longleftrightarrow$
(*cach*, *cach'*) $\in$ *cach-refinement-list* $\mathcal{A}_{in}$ $\wedge$
($\forall$ *L*<*length cach*. *cach* ! *L* $\neq$ *SEEN-UNKNOWN* $\longleftrightarrow$ *L* $\in$ *set support*) $\wedge$
($\forall$ *L* $\in$ *set support*. *L* < *length cach*) $\wedge$
distinct support $\wedge$ *set support* $\subseteq$ *set-mset* $\mathcal{A}_{in}$›
⟨*proof*⟩

definition (**in** −) *conflict-min-cach-l* :: ‹*conflict-min-cach-l* $\Rightarrow$ *nat* $\Rightarrow$ *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* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*x1l*, *x1j*) $\in$ *cach-refinement* $\mathcal{A}$›
shows ‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1n*)›
⟨*proof*⟩

lemma *nth-conflict-min-cach*:
‹(*uncurry* (*RETURN oo conflict-min-cach-l*), *uncurry* (*RETURN oo conflict-min-cach*)) $\in$
[λ(*cach*, *L*). *L* $\in\#$ $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* $\rightarrow$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *conflict-min-cach-set-failed*
:: ‹*nat conflict-min-cach* $\Rightarrow$ *nat* $\Rightarrow$ *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* $\Rightarrow$ *nat* $\Rightarrow$ *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 𝒜› and ‹distinct n› and ‹set n ⊆ set-mset 𝒜›
   shows ‹length n ≤ Suc (uint32-max div 2)›
⟨proof⟩
```

```
lemma conflict-min-cach-set-failed:
  ‹(uncurry conflict-min-cach-set-failed-l, uncurry (RETURN oo conflict-min-cach-set-failed)) ∈
    [λ(cach, L). L ∈# 𝒜_in ∧ isasat-input-bounded 𝒜_in]_f cach-refinement 𝒜_in ×_r nat-rel → ⟨cach-refinement
𝒜_in⟩nres-rel›
  ⟨proof⟩
```

```
definition (in −) conflict-min-cach-set-removable
  :: ‹nat conflict-min-cach ⇒ nat ⇒ nat conflict-min-cach›
where
  [simp]: ‹conflict-min-cach-set-removable cach L = cach(L:= SEEN-REMOVABLE)›
```

```
lemma conflict-min-cach-set-removable:
  ‹(uncurry conflict-min-cach-set-removable-l,
    uncurry (RETURN oo conflict-min-cach-set-removable)) ∈
    [λ(cach, L). L ∈# 𝒜_in ∧ isasat-input-bounded 𝒜_in]_f cach-refinement 𝒜_in ×_r nat-rel → ⟨cach-refinement
𝒜_in⟩nres-rel›
  ⟨proof⟩
```

```
definition isa-mark-failed-lits-stack where
  ‹isa-mark-failed-lits-stack NU analyse cach = do {
     let l = length analyse;
     ASSERT(length analyse ≤ 1 + uint32-max div 2);
     (-, cach) ← WHILE_T^λ(-, cach). True
       (λ(i, cach). i < l)
       (λ(i, cach). do {
          ASSERT(i < length analyse);
          let (cls-idx, idx, -) = (analyse ! i);
          ASSERT(cls-idx + idx ≥ 1);
          ASSERT(cls-idx + idx − 1 < length NU);
 ASSERT(arena-lit-pre NU (cls-idx + idx − 1));
 cach ← conflict-min-cach-set-failed-l cach (atm-of (arena-lit NU (cls-idx + idx − 1)));
          RETURN (i+1, cach)
       })
       (0, cach);
     RETURN cach
   }›
```

```
context
begin
lemma mark-failed-lits-stack-inv-helper1: ‹mark-failed-lits-stack-inv a ba a2' ⟹
       a1' < length ba ⟹
       (a1'a, a2'a) = ba ! a1' ⟹
       a1'a ∈# dom-m a›
```

⟨*proof*⟩

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2′* ⟹
a1′ < *length ba* ⟹
(*a1′a*, *xx*, *a2′a*, *yy*) = *ba* ! *a1′* ⟹
a2′a − *Suc 0* < *length* (*a* ∝ *a1′a*)›
⟨*proof*⟩

lemma *isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*:
assumes ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
shows ‹(*uncurry2 isa-mark-failed-lits-stack*, *uncurry2* (*mark-failed-lits-stack* $\mathcal{A}_{in}$)) ∈
[λ((*N*, *ana*), *cach*). *length ana* ≤ *1* + *uint32-max div 2*]$_f$
{(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *ana-lookups-rel NU* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$ →
⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

definition *isa-get-literal-and-remove-of-analyse-wl*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *nat literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl C analyse* =
(*let* (*i*, *j*, *b*) = (*last analyse*) *in*
(*arena-lit C* (*i* + *j*), *analyse*[*length analyse* − *1* := (*i*, *j* + *1*, *b*)]))›

definition *isa-get-literal-and-remove-of-analyse-wl-pre*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *bool*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ⟷
(*let* (*i*, *j*, *b*) = *last analyse in*
analyse ≠ [] ∧ *arena-lit-pre arena* (*i*+*j*) ∧ *j* < *uint32-max*)›

lemma *arena-lit-pre-le*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* ≤ *uint64-max*›
⟨*proof*⟩

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* < *uint64-max*›
⟨*proof*⟩

definition *lit-redundant-reason-stack-wl-lookup-pre* :: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *bool*› **where**
‹*lit-redundant-reason-stack-wl-lookup-pre L NU C* ⟷
arena-lit-pre NU C ∧
arena-is-valid-clause-idx NU C›

definition *lit-redundant-reason-stack-wl-lookup*
:: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *nat* × *nat* × *bool*›
where
‹*lit-redundant-reason-stack-wl-lookup L NU C* =
(*if arena-length NU C* > *2 then* (*C*, *1*, *False*)
else if arena-lit NU C = *L*
then (*C*, *1*, *False*)
else (*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^{λ-. 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 ((last analyse)));
     let (C, k, i, len) = ana-lookup-conv-lookup NU ((last analyse));
            ASSERT(C < length NU);
            ASSERT(arena-is-valid-clause-idx NU C);
            ASSERT(arena-lit-pre NU (C + k));
            if i ≥ 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 = 0 ∨
    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, take 0 analyse, 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, take 0 analyse, 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^λ-. 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 (last analyse));
  let (C, k, i, len) = ana-lookup-conv-lookup NU ((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 ≥ 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 = 0 ∨
                conflict-min-cach-l cach (atm-of L) = SEEN-REMOVABLE ∨
                atm-in-conflict-lookup (atm-of L) D)
            then RETURN (cach, analyse, False)
            else if b ∨ conflict-min-cach-l cach (atm-of L) = SEEN-FAILED
            then do {
              cach ← isa-mark-failed-lits-stack NU analyse cach;
              RETURN (cach, [], False)
            }
            else do {
              C ← get-propagation-reason-pol M (−L);
              case C of
                Some C ⇒ do {
    ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
    RETURN (cach, analyse @ [lit-redundant-reason-stack-wl-lookup (−L) NU C], False)
  }
              | None ⇒ do {
                  cach ← isa-mark-failed-lits-stack NU analyse cach;
                  RETURN (cach, [], False)
              }
          }
      }
    })
    (cach, analysis, False)›
```

⟨*proof*⟩

lemma *lit-redundant-rec-wl-lookup-alt-def*:

```
‹lit-redundant-rec-wl-lookup 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 ∝ C) > k); — >= 2 would work too
          ASSERT (NU ∝ C ! k ∈ lits-of-l M);
          ASSERT(NU ∝ C ! k ∈# $\mathcal{L}_{all}$ 𝒜);
  ASSERT(literals-are-in-$\mathcal{L}_{in}$ 𝒜 (mset (NU ∝ C)));
  ASSERT(length (NU ∝ C) ≤ Suc (uint32-max div 2));
  ASSERT(len ≤ length (NU ∝ C)); — makes the refinement easier
  let (C,k, i, len) = (C,k,i,len);
          let C = NU ∝ C;
          if i ≥ len
          then
             RETURN(cach (atm-of (C ! k) := SEEN-REMOVABLE), butlast analyse, True)
          else do {
             let (L, analyse) = get-literal-and-remove-of-analyse-wl2 C analyse;
             ASSERT(L ∈# $\mathcal{L}_{all}$ 𝒜);
             let b = ¬level-in-lbd (get-level M L) lbd;
             if (get-level M L = 0 ∨
                 conflict-min-cach cach (atm-of L) = SEEN-REMOVABLE ∨
                 atm-in-conflict (atm-of L) D)
             then RETURN (cach, analyse, False)
             else if b ∨ conflict-min-cach cach (atm-of L) = SEEN-FAILED
             then do {
                ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
                cach ← mark-failed-lits-wl NU analyse cach;
                RETURN (cach, [], False)
             }
             else do {
        ASSERT(− L ∈ lits-of-l M);
                C ← get-propagation-reason M (−L);
                case C of
                  Some C ⇒ do {
      ASSERT(C ∈# dom-m NU);
      ASSERT(length (NU ∝ C) ≥ 2);
      ASSERT(literals-are-in-$\mathcal{L}_{in}$ 𝒜 (mset (NU ∝ C)));
                    ASSERT(length (NU ∝ C) ≤ Suc (uint32-max div 2));
      RETURN (cach, analyse @ [lit-redundant-reason-stack2 (−L) NU C], False)
    }
                | None ⇒ do {
                    ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
                    cach ← mark-failed-lits-wl NU analyse cach;
                    RETURN (cach, [], False)
                }
             }
          }
      })
     (cach, analysis, False)›
```

⟨*proof*⟩

lemma *valid-arena-nempty*:
‹*valid-arena arena N vdom* $\Longrightarrow$ *i* ∈# *dom-m N* $\Longrightarrow$ *N* ∝ *i* ≠ []›
⟨*proof*⟩

lemma *isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-lit-redundant-rec-wl-lookup*, *uncurry5* (*lit-redundant-rec-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *iterate-over-conflict-spec*:
fixes *D* :: ‹*'v clause*›
assumes ‹*NU* + *NUE* ⊨*pm add-mset K D*› **and** *dist*: ‹*distinct-mset D*›
shows
‹*iterate-over-conflict K M NU NUE D* ≤ ⇓ *Id* (*SPEC*(λ*D'*. *D'* ⊆# *D* ∧
NU + *NUE* ⊨*pm add-mset K D'*))›
⟨*proof*⟩

end

lemma
fixes *D* :: ‹*nat clause*› **and** *s* **and** *s'* **and** *NU* :: ‹*nat clauses-l*› **and**
S :: ‹*nat twl-st-wl*› **and** *S'* :: ‹*nat twl-st-l*› **and** *S''* :: ‹*nat twl-st*›
defines
‹*S'''* ≡ *state*$_W$*-of S''*›
defines
‹*M* ≡ *get-trail-wl S*› **and**
NU: ‹*NU* ≡ *get-clauses-wl S*› **and**
NU'-def: ‹*NU'* ≡ *mset* ‘# *ran-mf NU*› **and**
NUE: ‹*NUE* ≡ *get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*› **and**
NUE: ‹*NUS* ≡ *get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-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* + *NUS*) *D*› **and**
NU-P-D: ‹*NU'* + *NUE* + *NUS* ⊨*pm add-mset K D*› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf NU*)› **and**
K: ‹*K* = *outl* ! *0*› **and**
outl-nempty: ‹*outl* ≠ []› **and**
‹*isasat-input-bounded* $\mathcal{A}$›

shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ *M NU D s′ lbd outl* ≤
⇓ ({((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧ *outl*!*0* = *K* ∧
E′ ⊆# *D*})
(*SPEC* (λ*D′*. *D′* ⊆# *D* ∧ *NU′* + *NUE* + *NUS* ⊨*pm add-mset K D′*))›
⟨*proof*⟩

lemma (**in** −) *lookup-conflict-upd-None-RETURN-def*:
‹*RETURN oo lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. *RETURN* (*n*− *1*, *xs* [*i* := *NOTIN*]))›
⟨*proof*⟩

definition *isa-literal-redundant-wl-lookup* ::
trail-pol ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l*
⇒ *nat literal* ⇒ *lbd* ⇒ (*conflict-min-cach-l* × (*nat* × *nat* × *bool*) *list* × *bool*) *nres*
where

```
‹isa-literal-redundant-wl-lookup M NU D cach L lbd = do {
    ASSERT(get-level-pol-pre (M, L));
    ASSERT(conflict-min-cach-l-pre (cach, atm-of L));
    if get-level-pol M L = 0 ∨ conflict-min-cach-l cach (atm-of L) = SEEN-REMOVABLE
    then RETURN (cach, [], True)
    else if conflict-min-cach-l cach (atm-of L) = SEEN-FAILED
    then RETURN (cach, [], False)
    else do {
      C ← get-propagation-reason-pol M (−L);
      case C of
        Some C ⇒ do {
          ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
          isa-lit-redundant-rec-wl-lookup M NU D cach
     [lit-redundant-reason-stack-wl-lookup (−L) NU C] lbd}
      | None ⇒ do {
          RETURN (cach, [], False)
      }
    }
}›
```

lemma *in*-$\mathcal{L}_{all}$-*atm-of*-$\mathcal{A}_{in}$*D*[*intro*]: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of L* ∈# $\mathcal{A}$›
⟨*proof*⟩

lemma *isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-literal-redundant-wl-lookup*, *uncurry5* (*literal-redundant-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in*-$\mathcal{L}_{in}$-*mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$ *cach-refinement* $\mathcal{A}$
$\times_f$ *Id* $\times_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *lookup-conflict-remove1* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*lookup-conflict-remove1* =
(λ*L* (*n*,*xs*). (*n*−*1*, *xs* [*atm-of L* := *NOTIN*]))›

lemma *lookup-conflict-remove1*:
‹(*uncurry* (*RETURN oo lookup-conflict-remove1*), *uncurry* (*RETURN oo remove1-mset*))
∈ [λ(*L*,*C*). *L* ∈# *C* ∧ −*L* ∉# *C* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id $\times_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›

⟨*proof*⟩

definition (**in** −) *lookup-conflict-remove1-pre* :: ‹*nat literal* × *nat* × *bool option list* ⇒ *bool*› **where**
‹*lookup-conflict-remove1-pre* = (λ(*L*,(*n*,*xs*)). *n* > *0* ∧ *atm-of L* < *length xs*)›

definition *isa-minimize-and-extract-highest-lookup-conflict*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l* ⇒ *lbd* ⇒
out-learned ⇒ (*lookup-clause-rel* × *conflict-min-cach-l* × *out-learned*) *nres*›
where

```
‹isa-minimize-and-extract-highest-lookup-conflict = (λM 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, 1, s, outl);
    RETURN (D, s, outl)
})›
```

lemma *isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict*:
assumes ‹*isasat-input-bounded* 𝒜›
shows ‹(*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*,
uncurry5 (*minimize-and-extract-highest-lookup-conflict* 𝒜)) ∈
[λ(((((-, *N*), *D*), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* 𝒜 ((*mset* ∘ *fst*) ‘# *ran-m N*) ∧
¬*tautology D*]$_f$
trail-pol 𝒜 $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* 𝒜 $\times_f$
cach-refinement 𝒜 $\times_f$ *Id* $\times_f$ *Id* →
⟨*lookup-clause-rel* 𝒜 $\times_r$ *cach-refinement* 𝒜 $\times_r$ *Id*⟩*nres-rel*›
⟨*proof*⟩

definition *set-empty-conflict-to-none* **where**
‹*set-empty-conflict-to-none D* = *None*›

definition *set-lookup-empty-conflict-to-none* **where**
‹*set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). (*True*, *n*, *xs*))›

lemma *set-empty-conflict-to-none-hnr*:
‹(*RETURN o set-lookup-empty-conflict-to-none*, *RETURN o set-empty-conflict-to-none*) ∈
[λ*D*. *D* = {#}]$_f$ *lookup-clause-rel* 𝒜 → ⟨*option-lookup-clause-rel* 𝒜⟩*nres-rel*›
⟨*proof*⟩

definition *lookup-merge-eq2*
:: ‹*nat literal* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*› **where**

```
‹lookup-merge-eq2 L M N = (λ(-, zs) clvls 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));
    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, outl)
  })›
```

definition *merge-conflict-m-eq2*

```
  :: ‹nat literal ⇒ (nat, nat) ann-lits ⇒ nat clause-l ⇒ nat clause option ⇒
  (nat clause option × nat × out-learned) nres›
```

where

```
‹merge-conflict-m-eq2 L M Ni D =
    SPEC (λ(C, n, 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: ‹$\bigwedge$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 outl ≤
⇓(option-lookup-clause-rel $\mathcal{A}$ $\times_r$ Id $\times_r$ Id)
(merge-conflict-m-eq2 L M D (Some C))›
(**is** ‹- ≤ ⇓ ?Ref ?Spec›)

⟨*proof*⟩

definition *isasat-lookup-merge-eq2*

```
  :: ‹nat literal ⇒ trail-pol ⇒ arena ⇒ nat ⇒ conflict-option-rel ⇒ nat ⇒
       out-learned ⇒ (conflict-option-rel × nat × out-learned) nres› where
‹isasat-lookup-merge-eq2 L M N C = (λ(-, zs) clvls 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));
    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*, *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 outl* ≤ ⇓ *Id*
      (*lookup-merge-eq2 L M* (*N* ∝ *i*) (*b*, *xs*) *clvls outl*)›
⟨*proof*⟩

definition *merge-conflict-m-eq2-pre* **where**
  ‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ =
  (λ((((((*L*, *M*), *N*), *i*), *xs*), *clvls*), *out*). *i* ∈# *dom-m N* ∧ *xs* ≠ *None* ∧ *distinct* (*N* ∝ *i*) ∧
      ¬*tautology* (*mset* (*N* ∝ *i*)) ∧
      (∀ *K* ∈ *set* (*remove1 L* (*N* ∝ *i*)). − *K* ∉# *the xs*) ∧
      literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) ∧ *clvls* = *card-max-lvl M* (*the xs*) ∧
      out-learned M xs out ∧ *no-dup M* ∧
      literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) ∧
      isasat-input-bounded $\mathcal{A}$ ∧
      length (*N* ∝ *i*) = *2* ∧
      L ∈ *set* (*N* ∝ *i*))›

definition *merge-conflict-m-g-eq2* :: ‹-› **where**
‹*merge-conflict-m-g-eq2 L M N i D* - - = *merge-conflict-m-eq2 L M* (*N* ∝ *i*) *D*›

lemma *isasat-lookup-merge-eq2*:
  ‹(*uncurry6 isasat-lookup-merge-eq2*, *uncurry6 merge-conflict-m-g-eq2*) ∈
    [*merge-conflict-m-eq2-pre* $\mathcal{A}$]$_f$
    Id $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel*
$\mathcal{A}$
        $\times_f$ *nat-rel* $\times_f$ *Id* →
      ⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id*⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-Setup*
  imports
    Watched-Literals-VMTF
    Watched-Literals.Watched-Literals-Watch-List-Initialisation
    IsaSAT-Lookup-Conflict
    IsaSAT-Clauses IsaSAT-Arena IsaSAT-Watch-List LBD
begin

Chapter 8

Complete state

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 use of *nat* leads to worse performance than using *sint64*. 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) if we generate Standard ML code.

We have successfully killed all natural numbers when generating LLVM. However, the LLVM binding does not have a binding to GMP integers.

8.1 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 = ‹64 word × 64 word × 64 word × 64 word × 64 word›

definition ema-bitshifting where
  ‹ema-bitshifting = (1 << 32)›


definition (in −) ema-update :: ‹nat ⇒ ema ⇒ ema› where
  ‹ema-update = (λlbd (value, α, β, wait, period).
     let lbd = (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-init :: ‹64 word ⇒ 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* ⇒ *64 word*› **where**
‹*ema-get-value* (*v*, -) = *v*›

fun *ema-extract-value* :: ‹*ema* ⇒ *64 word*› **where**
‹*ema-extract-value* (*v*, -) = *v* >> *32*›

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*›

8.2 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* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *ema*›

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* :: ‹*32 word* ⇒ *stats* ⇒ *stats*› **where**
‹*add-lbd lbd* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbds*). (*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *ema-update* (*unat lbd*) *lbds*))›

8.3 Information related to restarts

definition *NORMAL-PHASE* :: ⟨*64 word*⟩ **where**
⟨*NORMAL-PHASE = 0*⟩

definition *QUIET-PHASE* :: ⟨*64 word*⟩ **where**
⟨*QUIET-PHASE = 1*⟩

definition *DEFAULT-INIT-PHASE* :: ⟨*64 word*⟩ **where**
⟨*DEFAULT-INIT-PHASE = 10000*⟩

type-synonym *restart-info* = ⟨*64 word × 64 word × 64 word × 64 word × 64 word*⟩

definition *incr-conflict-count-since-last-restart* :: ⟨*restart-info ⇒ restart-info*⟩ **where**
⟨*incr-conflict-count-since-last-restart = (λ(ccount, ema-lvl, restart-phase, end-of-phase, length-phase).*
(ccount + 1, ema-lvl, restart-phase, end-of-phase, length-phase))⟩

definition *restart-info-update-lvl-avg* :: ⟨*32 word ⇒ 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, NORMAL-PHASE, DEFAULT-INIT-PHASE, 1000)*⟩

definition *restart-info-restart-done* :: ⟨*restart-info ⇒ restart-info*⟩ **where**
⟨*restart-info-restart-done = (λ(ccount, lvl-avg). (0, lvl-avg))*⟩

8.4 Phase saving

type-synonym *phase-save-heur* = ⟨*phase-saver × nat × phase-saver × nat × phase-saver × 64 word × 64 word × 64 word*⟩

definition *phase-save-heur-rel* :: ⟨*nat multiset ⇒ phase-save-heur ⇒ bool*⟩ **where**
⟨*phase-save-heur-rel 𝒜 = (λ(φ, target-assigned, target, best-assigned, best,*
end-of-phase, curr-phase). phase-saving 𝒜 φ ∧
phase-saving 𝒜 target ∧ phase-saving 𝒜 best ∧ length φ = length best ∧
length target = length best)⟩

definition *end-of-rephasing-phase* :: ⟨*phase-save-heur ⇒ 64 word*⟩ **where**
⟨*end-of-rephasing-phase = (λ(φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase,*
length-phase). end-of-phase)⟩

definition *phase-current-rephasing-phase* :: ⟨*phase-save-heur ⇒ 64 word*⟩ **where**
⟨*phase-current-rephasing-phase =*
(λ(φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase, length-phase). curr-phase)⟩

8.5 Heuristics

type-synonym *restart-heuristics* = ⟨*ema × ema × restart-info × 64 word × phase-save-heur*⟩

fun *fast-ema-of* :: ⟨*restart-heuristics ⇒ ema*⟩ **where**
⟨*fast-ema-of (fast-ema, slow-ema, restart-info, wasted, φ) = fast-ema*⟩

fun *slow-ema-of* :: ⟨*restart-heuristics ⇒ ema*⟩ **where**

⟨*slow-ema-of* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ) = *slow-ema*⟩

fun *restart-info-of* :: ⟨*restart-heuristics* ⇒ *restart-info*⟩ **where**
⟨*restart-info-of* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ) = *restart-info*⟩

fun *current-restart-phase* :: ⟨*restart-heuristics* ⇒ *64 word*⟩ **where**
⟨*current-restart-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ)
=
restart-phase⟩

fun *incr-restart-phase* :: ⟨*restart-heuristics* ⇒ *restart-heuristics*⟩ **where**
⟨*incr-restart-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ) =
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*, φ)⟩

fun *incr-wasted* :: ⟨*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*⟩ **where**
⟨*incr-wasted waste* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) =
(*fast-ema*, *slow-ema*, *res-info*, *wasted* + *waste*, φ)⟩

fun *set-zero-wasted* :: ⟨*restart-heuristics* ⇒ *restart-heuristics*⟩ **where**
⟨*set-zero-wasted* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) =
(*fast-ema*, *slow-ema*, *res-info*, *0*, φ)⟩

fun *wasted-of* :: ⟨*restart-heuristics* ⇒ *64 word*⟩ **where**
⟨*wasted-of* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) = *wasted*⟩

definition *heuristic-rel* :: ⟨*nat multiset* ⇒ *restart-heuristics* ⇒ *bool*⟩ **where**
⟨*heuristic-rel* $\mathcal{A}$ = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *phase-save-heur-rel* $\mathcal{A}$ φ)⟩

definition *save-phase-heur* :: ⟨*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics*⟩ **where**
⟨*save-phase-heur L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, *target*, *best*)).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ[*L* := *b*], *target*, *best*)))⟩

definition *save-phase-heur-pre* :: ⟨*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *bool*⟩ **where**
⟨*save-phase-heur-pre L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). *L* < *length* φ)⟩

definition *mop-save-phase-heur* :: ⟨*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*⟩ **where**

```
⟨mop-save-phase-heur L b heur = do {
   ASSERT(save-phase-heur-pre L b heur);
   RETURN (save-phase-heur L b heur)
}⟩
```

definition *get-saved-phase-heur-pre* :: ⟨*nat* ⇒ *restart-heuristics* ⇒ *bool*⟩ **where**
⟨*get-saved-phase-heur-pre L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). *L* < *length* φ)⟩

definition *get-saved-phase-heur* :: ⟨*nat* ⇒ *restart-heuristics* ⇒ *bool*⟩ **where**
⟨*get-saved-phase-heur L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). φ!*L*)⟩

definition *current-rephasing-phase* :: ⟨*restart-heuristics* ⇒ *64 word*⟩ **where**
⟨*current-rephasing-phase* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *phase-current-rephasing-phase* φ)⟩

definition *mop-get-saved-phase-heur* :: ⟨*nat* ⇒ *restart-heuristics* ⇒ *bool nres*⟩ **where**

```
⟨mop-get-saved-phase-heur L heur = do {
   ASSERT(get-saved-phase-heur-pre L heur);
   RETURN (get-saved-phase-heur L heur)
}⟩
```

definition *end-of-rephasing-phase-heur* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase-heur* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*). *end-of-rephasing-phase phasing*)›

lemma *heuristic-relI*[*intro*!]:
‹*heuristic-rel* 𝒜 *heur* ⟹ *heuristic-rel* 𝒜 (*incr-wasted wast heur*)›
‹*heuristic-rel* 𝒜 *heur* ⟹ *heuristic-rel* 𝒜 (*set-zero-wasted heur*)›
‹*heuristic-rel* 𝒜 *heur* ⟹ *heuristic-rel* 𝒜 (*incr-restart-phase heur*)›
‹*heuristic-rel* 𝒜 *heur* ⟹ *heuristic-rel* 𝒜 (*save-phase-heur L b heur*)›
⟨*proof*⟩

lemma *save-phase-heur-preI*:
‹*heuristic-rel* 𝒜 *heur* ⟹ *a* ∈# 𝒜 ⟹ *save-phase-heur-pre a b heur*›
⟨*proof*⟩

8.6 VMTF

type-synonym (**in** −) *isa-vmtf-remove-int* = ‹*vmtf* × (*nat list* × *bool list*)›

8.7 Options

type-synonym *opts* = ‹*bool* × *bool* × *bool*›

definition *opts-restart* **where**
‹*opts-restart* = (λ(*a*, *b*, *c*). *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*)›

type-synonym *out-learned* = ‹*nat clause-l*›

type-synonym *vdom* = ‹*nat list*›

8.7.1 Conflict

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*)›

8.8 Full state

heur stands for heuristic.

Definition **type-synonym** *twl-st-wl-heur* =
‹*trail-pol* × *arena* ×
conflict-option-rel × *nat* × (*nat watcher*) *list list* × *isa-vmtf-remove-int* ×
nat × *conflict-min-cach-l* × *lbd* × *out-learned* × *stats* × *restart-heuristics* ×
vdom × *vdom* × *nat* × *opts* × *arena*›

Accessors **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*›

definition (**in** −) *mop-watched-by-app-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher nres*›
where
‹*mop-watched-by-app-heur S L K* = *do* {
ASSERT(*K* < *length* (*watched-by-int S L*));
ASSERT(*nat-of-lit L* < *length* (*get-watched-wl-heur S*));
RETURN (*watched-by-int S L* ! *K*)}›

lemma *watched-by-app-heur-alt-def*:
‹*watched-by-app-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L K*. *W* ! *nat-of-lit L* ! *K*)›
⟨*proof*⟩

definition *watched-by-app* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app S L K* = *watched-by S L* ! *K*›

fun *get-vmtf-heur* :: ‹*twl-st-wl-heur* ⇒ *isa-vmtf-remove-int*› **where**
‹*get-vmtf-heur* (-, -, -, -, -, *vm*, -) = *vm*›

fun *get-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* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *fast-ema-of heur*⟩

fun *get-slow-ema-heur* :: ⟨*twl-st-wl-heur* ⇒ *ema*⟩ **where**
⟨*get-slow-ema-heur* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *slow-ema-of heur*⟩

fun *get-conflict-count-heur* :: ⟨*twl-st-wl-heur* ⇒ *restart-info*⟩ **where**
⟨*get-conflict-count-heur* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *restart-info-of heur*⟩

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*⟩

8.9 Virtual domain

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-m* $\mathcal{A}$ *W N* = ⋃(((‘) *fst*) ‘ *set* ‘ *W* ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) ∪ *set-mset* (*dom-m N*)⟩

lemma *vdom-m-simps*[*simp*]:
⟨*bh* ∈# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ *W* (*N*(*bh* ↪ *C*)) = *vdom-m* $\mathcal{A}$ *W N*⟩
⟨*bh* ∉# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ *W* (*N*(*bh* ↪ *C*)) = *insert bh* (*vdom-m* $\mathcal{A}$ *W N*)⟩
⟨*proof*⟩

lemma *vdom-m-simps2*[*simp*]:
⟨*i* ∈# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ (*W*(*L* := *W L* @ [(*i*, *C*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*⟩
⟨*bi* ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ (*bp*(*L*:= *bp L* @ [(*bi*, *av*′)])) *ax* = *vdom-m* $\mathcal{A}$ *bp ax*⟩
⟨*proof*⟩

lemma *vdom-m-simps3*[*simp*]:

‹*fst biav′* ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ (*bp*(*L*:= *bp L* @ [*biav′*])) *ax* = *vdom-m* $\mathcal{A}$ *bp ax*›
⟨*proof*⟩

What is the difference with the next lemma?

lemma [*simp*]:
‹*bf* ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ *bj* (*ax*(*bf* ↪ *C′*)) = *vdom-m* $\mathcal{A}$ *bj* (*ax*)›
⟨*proof*⟩

lemma *vdom-m-simps4* [*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m $\mathcal{A}$ (*W* (*L1* := *W L1* @ [(*i*, *C1*)], *L2* := *W L2* @ [(*i*, *C2*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*›
⟨*proof*⟩

This is *?i* ∈# *dom-m ?N* ⟹ *vdom-m ?*$\mathcal{A}$ (*?W*(*?L1.0* := *?W ?L1.0* @ [(*?i*, *?C1.0*)], *?L2.0* := *?W ?L2.0* @ [(*?i*, *?C2.0*)])) *?N* = *vdom-m ?*$\mathcal{A}$ *?W ?N* if the assumption of distinctness is not present in the context.

lemma *vdom-m-simps4′*[*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m $\mathcal{A}$ (*W* (*L1* := *W L1* @ [(*i*, *C1*), (*i*, *C2*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*›
⟨*proof*⟩

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-watched* $\mathcal{A}$ = (λ-. [])›

lemma *vdom-m-empty-watched*[*simp*]:
‹*vdom-m* $\mathcal{A}$ (*empty-watched* $\mathcal{A}'$) *N* = *set-mset* (*dom-m N*)›
⟨*proof*⟩

The following rule makes the previous one not applicable. Therefore, we do not mark this lemma as simp.

lemma *vdom-m-simps5*:
‹*i* ∉# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ *W* (*fmupd i C N*) = *insert i* (*vdom-m* $\mathcal{A}$ *W N*)›
⟨*proof*⟩

lemma *in-watch-list-in-vdom*:
assumes ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*w* < *length* (*watched-by S L*)›
shows ‹*fst* (*watched-by S L* ! *w*) ∈ *vdom-m* $\mathcal{A}$ (*get-watched-wl S*) (*get-clauses-wl S*)›
⟨*proof*⟩

lemma *in-watch-list-in-vdom′*:
assumes ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*A* ∈ *set* (*watched-by S L*)›
shows ‹*fst A* ∈ *vdom-m* $\mathcal{A}$ (*get-watched-wl S*) (*get-clauses-wl S*)›
⟨*proof*⟩

lemma *in-dom-in-vdom*[*simp*]:
‹*x* ∈# *dom-m N* ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ *W N*›
⟨*proof*⟩

lemma *in-vdom-m-upd*:
‹*x1f* ∈ *vdom-m* $\mathcal{A}$ (*g*(*x1e* := (*g x1e*)[*x2* := (*x1f*, *x2f*)])) *b*›
if ‹*x2* < *length* (*g x1e*)› **and** ‹*x1e* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
⟨*proof*⟩

lemma *in-vdom-m-fmdropD*:
‹$x \in$ *vdom-m* $\mathcal{A}$ *ga* (*fmdrop C baa*) $\Longrightarrow x \in$ (*vdom-m* $\mathcal{A}$ *ga baa*)›
⟨*proof*⟩

definition *cach-refinement-empty* **where**
‹*cach-refinement-empty* $\mathcal{A}$ *cach* $\longleftrightarrow$
(*cach*, λ-. *SEEN-UNKNOWN*) $\in$ *cach-refinement* $\mathcal{A}$›

VMTF **definition** *isa-vmtf* **where**
‹*isa-vmtf* $\mathcal{A}$ M =
((*Id* $\times_r$ *nat-rel* $\times_r$ *nat-rel* $\times_r$ *nat-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel*) $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)$^{-1}$
‘‘ *vmtf* $\mathcal{A}$ M›

lemma *isa-vmtfI*:
‹(*vm*, *to-remove*$'$) $\in$ *vmtf* $\mathcal{A}$ M $\Longrightarrow$ (*to-remove*, *to-remove*$'$) $\in$ *distinct-atoms-rel* $\mathcal{A}$ $\Longrightarrow$
(*vm*, *to-remove*) $\in$ *isa-vmtf* $\mathcal{A}$ M›
⟨*proof*⟩

lemma *isa-vmtf-consD*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *isa-vmtf* $\mathcal{A}$ M $\Longrightarrow$
((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *isa-vmtf* $\mathcal{A}$ (L # M)›
⟨*proof*⟩

lemma *isa-vmtf-consD2*:
‹$f \in$ *isa-vmtf* $\mathcal{A}$ M $\Longrightarrow$
$f \in$ *isa-vmtf* $\mathcal{A}$ (L # M)›
⟨*proof*⟩

vdom is an upper bound on all the address of the clauses that are used in the state. *avdom* includes the active clauses.

definition *twl-st-heur* :: ‹(*twl-st-wl-heur* $\times$ *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur* =
{((M', N', D', j, W', *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*),
(M, N, D, *NE*, *UE*, *NS*, *US*, Q, W)).
(M', M) $\in$ *trail-pol* (*all-atms* N (*NE* + *UE* + *NS* + *US*)) $\wedge$
valid-arena N' N (*set vdom*) $\wedge$
(D', D) $\in$ *option-lookup-clause-rel* (*all-atms* N (*NE* + *UE* + *NS* + *US*)) $\wedge$
(D = *None* $\longrightarrow$ $j \le$ *length* M) $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) $\wedge$
(W', W) $\in$ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms* N (*NE* + *UE* + *NS* + *US*))) $\wedge$
vm $\in$ *isa-vmtf* (*all-atms* N (*NE* + *UE* + *NS* + *US*)) M $\wedge$
no-dup M $\wedge$
clvls $\in$ *counts-maximum-level* M D $\wedge$
cach-refinement-empty (*all-atms* N (*NE* + *UE* + *NS* + *US*)) *cach* $\wedge$
out-learned M D *outl* $\wedge$
lcount = *size* (*learned-clss-lf* N) $\wedge$
vdom-m (*all-atms* N (*NE* + *UE* + *NS* + *US*)) W N $\subseteq$ *set vdom* $\wedge$
mset avdom $\subseteq$# *mset vdom* $\wedge$
distinct vdom $\wedge$
isasat-input-bounded (*all-atms* N (*NE* + *UE* + *NS* + *US*)) $\wedge$
isasat-input-nempty (*all-atms* N (*NE* + *UE* + *NS* + *US*)) $\wedge$
old-arena = [] $\wedge$
heuristic-rel (*all-atms* N (*NE* + *UE* + *NS* + *US*)) *heur*

}›

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'*)))› **and**
    twl-st-heur-state-simp-watched2: ‹*C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S'*) ⟹
      nat-of-lit C < *length*(*get-watched-wl-heur S*)›
  ⟨*proof*⟩

abbreviation *twl-st-heur'''*
   :: ‹*nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur''' r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
          length (*get-clauses-wl-heur S*) = *r*}›

definition *twl-st-heur'* :: ‹*nat multiset* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur' N* = {(*S*, *S'*). (*S*, *S'*) ∈ *twl-st-heur* ∧ *dom-m* (*get-clauses-wl S'*) = *N*}›

definition *twl-st-heur-conflict-ana*
  :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur-conflict-ana* =
  {((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
      avdom, *lcount*, *opts*, *old-arena*),
     (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)).
    (*M'*, *M*) ∈ *trail-pol* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
    valid-arena N' N (*set vdom*) ∧
    (*D'*, *D*) ∈ *option-lookup-clause-rel* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
    (*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE* + *NS* + *US*))) ∧
    vm ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *M* ∧
    no-dup M ∧
    clvls ∈ *counts-maximum-level M D* ∧
    cach-refinement-empty (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *cach* ∧
    out-learned M D outl ∧
    lcount = *size* (*learned-clss-lf N*) ∧
    vdom-m (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *W N* ⊆ *set vdom* ∧
    mset avdom ⊆# *mset vdom* ∧
    distinct vdom ∧
    isasat-input-bounded (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
    isasat-input-nempty (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
    old-arena = [] ∧
    heuristic-rel (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *heur*
  }›

lemma *twl-st-heur-twl-st-heur-conflict-ana*:
  ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ (*S*, *T*) ∈ *twl-st-heur-conflict-ana*›
  ⟨*proof*⟩

lemma *twl-st-heur-ana-state-simp*:
  assumes ‹(*S*, *S'*) ∈ *twl-st-heur-conflict-ana*›
  shows

‹(*get-trail-wl-heur S*, *get-trail-wl S'*) ∈ *trail-pol* (*all-atms-st S'*)› **and**
‹$C \in\# \mathcal{L}_{all}$ (*all-atms-st S'*) ⟹ *watched-by-int S C* = *watched-by S' C*›
⟨*proof*⟩

This relations decouples the conflict that has been minimised and appears abstractly from the refined state, where the conflict has been removed from the data structure to a separate array.

definition *twl-st-heur-bt* :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-bt* =
{((*M'*, *N'*, *D'*, *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*,
old-arena),
(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)).
(*M'*, *M*) ∈ *trail-pol* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
valid-arena N' N (*set vdom*) ∧
(*D'*, *None*) ∈ *option-lookup-clause-rel* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE* + *NS* + *US*))) ∧
vm ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M None* ∧
cach-refinement-empty (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *cach* ∧
out-learned M None outl ∧
lcount = *size* (*learned-clss-l N*) ∧
vdom-m (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *W N* ⊆ *set vdom* ∧
mset avdom ⊆# *mset vdom* ∧
distinct vdom ∧
isasat-input-bounded (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
isasat-input-nempty (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
old-arena = [] ∧
heuristic-rel (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *heur*
}›

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*) ≤ *sint64-max* − (*uint32-max div 2* + *MAX-HEADER-SIZE*+*1*))›

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* ⟹ *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*›
⟨*proof*⟩

8.10 Lift Operations to State

definition *polarity-st* :: ‹*'v twl-st-wl* ⇒ *'v literal* ⇒ *bool option*› **where**
‹*polarity-st S* = *polarity* (*get-trail-wl S*)›

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

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-conflict-wl-is-None-heur-alt-def*:
‹*RETURN o get-conflict-wl-is-None-heur* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)›
⟨*proof*⟩

definition *count-decided-st* :: ‹*nat twl-st-wl* ⇒ *nat*› **where**
‹*count-decided-st* = (λ(*M*, -). *count-decided M*)›

definition *isa-count-decided-st* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*isa-count-decided-st* = (λ(*M*, -). *count-decided-pol M*)›

lemma *count-decided-st-count-decided-st*:
‹(*RETURN o isa-count-decided-st*, *RETURN o count-decided-st*) ∈ *twl-st-heur* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *count-decided-st-alt-def*: ‹*count-decided-st S* = *count-decided* (*get-trail-wl S*)›
⟨*proof*⟩

definition (**in** −) *is-in-conflict-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*› **where**
‹*is-in-conflict-st L S* ⟷ *is-in-conflict L* (*get-conflict-wl S*)›

definition *atm-is-in-conflict-st-heur* :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *bool nres*› **where**
‹*atm-is-in-conflict-st-heur L* = (λ(*M*, *N*, (-, *D*), -). *do* {
ASSERT (*atm-in-conflict-lookup-pre* (*atm-of L*) *D*); *RETURN* (¬*atm-in-conflict-lookup* (*atm-of L*) *D*) })›

lemma *atm-is-in-conflict-st-heur-alt-def*:
‹*atm-is-in-conflict-st-heur* = (λ*L* (*M*, *N*, (-, (-, *D*)), -). *do* {*ASSERT* ((*atm-of L*) < *length D*); *RETURN* (*D* ! (*atm-of L*) = *None*)})›
⟨*proof*⟩

lemma *atm-of-in-atms-of-iff*: ‹*atm-of x* ∈ *atms-of D* ⟷ *x* ∈# *D* ∨ −*x* ∈# *D*›
⟨*proof*⟩

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*atm-is-in-conflict-st-heur*), *uncurry* (*mop-lit-notin-conflict-wl*)) ∈
$[\lambda(L,\ S).\ True]_f$
Id $\times_r$ *twl-st-heur* → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

abbreviation *nat-lit-lit-rel* **where**
‹*nat-lit-lit-rel* ≡ *Id* :: (*nat literal* × -) *set*›

8.11 More theorems

lemma *valid-arena-DECISION-REASON*:
‹*valid-arena arena NU vdom* ⟹ *DECISION-REASON* ∉# *dom-m NU*›
⟨*proof*⟩

definition *count-decided-st-heur* :: ‹- ⇒ -› **where**
‹*count-decided-st-heur* = (λ((-,-,-,-,*n*, -), -). *n*)›

lemma *twl-st-heur-count-decided-st-alt-def*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
⟨*proof*⟩

lemma *twl-st-heur-isa-length-trail-get-trail-wl*:
fixes S :: *twl-st-wl-heur*
shows ‹$(S,\ T) \in$ *twl-st-heur* $\Longrightarrow$ *isa-length-trail* (*get-trail-wl-heur* S) = *length* (*get-trail-wl* T)›
⟨*proof*⟩

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *trail-pol* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *trail-pol* $\mathcal{B}$›
⟨*proof*⟩

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *distinct-atoms-rel* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *distinct-atoms-rel* $\mathcal{B}$›
⟨*proof*⟩

lemma *phase-save-heur-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *phase-save-heur-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *phase-save-heur-rel* $\mathcal{B}$ *heur*›
⟨*proof*⟩

lemma *heuristic-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{B}$ *heur*›
⟨*proof*⟩

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *vmtf* $\mathcal{A}$ M $\Longrightarrow$ $L \in$ *vmtf* $\mathcal{B}$ M›
⟨*proof*⟩

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›
⟨*proof*⟩

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* $\mathcal{B}$›
⟨*proof*⟩

lemma D_0*-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $D_0\ \mathcal{A} = D_0\ \mathcal{B}$›
⟨*proof*⟩

lemma *phase-saving-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *phase-saving* $\mathcal{A}$ = *phase-saving* $\mathcal{B}$›
⟨*proof*⟩

lemma *cach-refinement-empty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *cach-refinement-empty* $\mathcal{A}$ = *cach-refinement-empty* $\mathcal{B}$›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *isasat-input-bounded-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ = *isasat-input-bounded* $\mathcal{B}$›
⟨*proof*⟩

lemma *isasat-input-nempty-cong*:
‹*set-mset 𝒜 = set-mset ℬ ⟹ isasat-input-nempty 𝒜 = isasat-input-nempty ℬ*›
⟨*proof*⟩

8.12 Shared Code Equations

definition *clause-not-marked-to-delete* **where**
‹*clause-not-marked-to-delete S C ⟷ 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 ⟷*
  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 ×$_f$ nat-rel → ⟨bool-rel⟩nres-rel›
⟨*proof*⟩

definition (**in** −) *access-lit-in-clauses-heur-pre* **where**
‹*access-lit-in-clauses-heur-pre =*
  (λ((S, i), j).
    arena-lit-pre (get-clauses-wl-heur S) (i+j))›

definition (**in** −) *access-lit-in-clauses-heur* **where**
‹*access-lit-in-clauses-heur S i j = arena-lit (get-clauses-wl-heur S) (i + j)*›

lemma *access-lit-in-clauses-heur-alt-def*:
‹*access-lit-in-clauses-heur = (λ(M, N, -) i j. arena-lit N (i + j))*›
⟨*proof*⟩

definition (**in** −) *mop-access-lit-in-clauses-heur* **where**
‹*mop-access-lit-in-clauses-heur S i j = mop-arena-lit2 (get-clauses-wl-heur S) i j*›

lemma *mop-access-lit-in-clauses-heur-alt-def*:
‹*mop-access-lit-in-clauses-heur = (λ(M, N, -) i j. mop-arena-lit2 N i j)*›
⟨*proof*⟩

lemma *access-lit-in-clauses-heur-fast-pre*:
‹*arena-lit-pre (get-clauses-wl-heur a) (ba + b) ⟹*
  isasat-fast a ⟹ ba + b ≤ sint64-max›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$-*add-mset*:
‹*set-mset* ($\mathcal{L}_{all}$ (*add-mset L C*)) = *insert* (*Pos L*) (*insert* (*Neg L*) (*set-mset* ($\mathcal{L}_{all}$ *C*)))›
⟨*proof*⟩

lemma *correct-watching-dom-watched*:
assumes ‹*correct-watching S*› **and** ‹$\bigwedge$*C*. *C* ∈# *ran-mf* (*get-clauses-wl S*) ⟹ *C* ≠ []›
shows ‹*set-mset* (*dom-m* (*get-clauses-wl S*)) ⊆
$\bigcup$(((\`) *fst*) \` *set* \` (*get-watched-wl S*) \` *set-mset* ($\mathcal{L}_{all}$ (*all-atms-st S*)))›
(**is** ‹*?A* ⊆ *?B*›)
⟨*proof*⟩

8.13 Rewatch

definition *rewatch-heur* **where**

```
‹rewatch-heur vdom arena W = do {
  let - = vdom;
  nfoldli [0..<length vdom] (λ-. True)
   (λi W. do {
      ASSERT(i < length vdom);
      let C = vdom ! i;
      ASSERT(arena-is-valid-clause-vdom arena C);
      if arena-status arena C ≠ DELETED
      then do {
        L1 ← mop-arena-lit2 arena C 0;
        L2 ← mop-arena-lit2 arena C 1;
        n ← mop-arena-length arena C;
        let b = (n = 2);
        ASSERT(length (W ! (nat-of-lit L1)) < length arena);
        W ← mop-append-ll W L1 (C, L2, b);
        ASSERT(length (W ! (nat-of-lit L2)) < length arena);
        W ← mop-append-ll W L2 (C, L1, b);
        RETURN W
      }
      else RETURN W
    })
   W
  }›
```

lemma *rewatch-heur-rewatch*:
assumes
valid: ‹*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*⟩

lemma *rewatch-heur-alt-def*:
‹*rewatch-heur vdom arena W* = *do* {

```
  let - = vdom;
  nfoldli [0..<length vdom] (λ-. True)
   (λi W. do {
      ASSERT(i < length vdom);
      let C = vdom ! i;
      ASSERT(arena-is-valid-clause-vdom arena C);
      if arena-status arena C ≠ DELETED
      then do {
        L1 ← mop-arena-lit2 arena C 0;
        L2 ← mop-arena-lit2 arena C 1;
        n ← mop-arena-length arena C;
        let b = (n = 2);
        ASSERT(length (W ! (nat-of-lit L1)) < length arena);
        W ← mop-append-ll W L1 (C, L2, b);
        ASSERT(length (W ! (nat-of-lit L2)) < length arena);
        W ← mop-append-ll W L2 (C, L1, b);
        RETURN W
      }
      else RETURN W
    })
   W
  }›
  ⟨proof⟩

lemma arena-lit-pre-le-sint64-max:
 ‹length ba ≤ sint64-max ⟹
       arena-lit-pre ba a ⟹ a ≤ sint64-max›
  ⟨proof⟩

definition rewatch-heur-st
 :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
where
‹rewatch-heur-st = (λ(M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
       stats, heur, vdom, avdom, ccount, lcount). do {
  ASSERT(length vdom ≤ length N0);
  W ← rewatch-heur vdom N0 W;
  RETURN (M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
       stats, heur, 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 ≤ sint64-max) ∧ length (get-clauses-wl-heur S) ≤ sint64-max)›

definition rewatch-st :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres› where
  ‹rewatch-st S = do{
     (M, N, D, NE, UE, NS, US, Q, W) ← RETURN S;
     W ← rewatch N W;
     RETURN ((M, N, D, NE, UE, NS, US, Q, W))
  }›


fun remove-watched-wl :: ‹'v twl-st-wl ⇒ -› where
```

⟨*remove-watched-wl* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, -) = (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *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)⟩
  ⟨*proof*⟩

8.14 Fast to slow conversion

Setup to convert a list from *64 word* to *nat*.

definition *convert-wlists-to-nat-conv* :: ⟨*'a list list* ⇒ *'a list list*⟩ **where**
  ⟨*convert-wlists-to-nat-conv* = *id*⟩

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* ∧ *s* ≤ *r*}⟩

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* − *MIN-HEADER-SIZE*⟩
⟨*proof*⟩

lemma *length-watched-le2*:
  assumes
    prop-inv: ⟨*correct-watching-except i j L x1*⟩ **and**
    xb-x'a: ⟨(*x1a*, *x1*) ∈ *twl-st-heur''* $\mathcal{D}1$ *r*⟩ **and**
    x2: ⟨*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)⟩ **and** *diff*: ⟨*L* ≠ *x2*⟩
  shows ⟨*length* (*watched-by x1 x2*) ≤ *r* − *MIN-HEADER-SIZE*⟩
⟨*proof*⟩

lemma *atm-of-all-lits-of-m*: ⟨*atm-of* '# (*all-lits-of-m C*) = *atm-of* '# *C* + *atm-of* '# *C*⟩
  ⟨*atm-of* ' *set-mset* (*all-lits-of-m C*) = *atm-of* '*set-mset C* ⟩
  ⟨*proof*⟩

lemma *mop-watched-by-app-heur-mop-watched-by-at*:
  ⟨(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
    twl-st-heur $\times_f$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*⟩
  ⟨*proof*⟩

lemma *mop-watched-by-app-heur-mop-watched-by-at''*:
‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
twl-st-heur-up'' $\mathcal{D}$ *r s K* $\times_f$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *mop-polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-pol* = (λ*M L*. *do* {
ASSERT(*polarity-pol-pre M L*);
RETURN (*polarity-pol M L*)
})›

definition *polarity-st-pre* :: ‹*nat twl-st-wl* × *nat literal* ⇒ *bool*› **where**
‹*polarity-st-pre* ≡ λ(*S*, *L*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›

definition *mop-polarity-st-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-st-heur S L* = *do* {
mop-polarity-pol (*get-trail-wl-heur S*) *L*
}›

lemma *mop-polarity-st-heur-alt-def*: ‹*mop-polarity-st-heur* = (λ(*M*, -) *L*. *do* {
mop-polarity-pol M L
})›
⟨*proof*⟩

lemma *mop-polarity-st-heur-mop-polarity-wl*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur* $\times_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *mop-polarity-st-heur-mop-polarity-wl''*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\times_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma [*simp*,*iff*]: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ⟷ *blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*). *length avdom*)›
⟨*proof*⟩

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*,

heur, *vdom*, *lcount*) *C* . *arena-status N′ C* = *LEARNED*)›
⟨*proof*⟩

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *lcount*) *L* . *get-the-propagation-reason-pol M′ L*)›
⟨*proof*⟩

definition *clause-lbd-heur* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat*›
where
‹*clause-lbd-heur S C* = *arena-lbd* (*get-clauses-wl-heur S*) *C*›

definition (**in** −) *access-length-heur* **where**
‹*access-length-heur S i* = *arena-length* (*get-clauses-wl-heur S*) *i*›

lemma *access-length-heur-alt-def*:
‹*access-length-heur* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
lcount) *C*. *arena-length N′ C*)›
⟨*proof*⟩

definition *marked-as-used-st* **where**
‹*marked-as-used-st T C* =
marked-as-used (*get-clauses-wl-heur T*) *C*›

lemma *marked-as-used-st-alt-def*:
‹*marked-as-used-st* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
lcount) *C*.
marked-as-used N′ C)›
⟨*proof*⟩

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*)
i. *avdom* ! *i*)›
⟨*proof*⟩

definition *access-vdom-at-pre* **where**
‹*access-vdom-at-pre S i* ⟷ *i* < *length* (*get-avdom S*)›

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*).
(*M′*, *extra-information-mark-to-delete N′ C*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,

vdom, *delete-index-and-swap avdom i*, *lcount* − *1*, *opts*, *old-arena*))›

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *if st then lcount else lcount* − *1*, *opts*) })›

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*).
(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *delete-index-and-swap avdom i*, *lcount*))›

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-unused arena C*) *C*›
⟨*proof*⟩

definition *mop-mark-garbage-heur* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mop-mark-garbage-heur C i* = (λ*S*. *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *get-learned-count S* ≥ *1* ∧ *i* < *length* (*get-avdom S*));
RETURN (*mark-garbage-heur C i S*)
})›

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*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*).
(*M'*, *mark-unused N' C*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*,
lbd, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*))›

definition *mop-mark-unused-st-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mop-mark-unused-st-heur C T* = *do* {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*mark-unused-st-heur C T*)
}›

lemma *mop-mark-garbage-heur-alt-def*:
‹*mop-mark-garbage-heur C i* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur* (*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*), *C*) ∧ *lcount* ≥ *1* ∧ *i* < *length avdom*);
RETURN (*M'*, *extra-information-mark-to-delete N' C*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*,
vdom, *delete-index-and-swap avdom i*, *lcount* − *1*, *opts*, *old-arena*)
})›
⟨*proof*⟩

lemma *mark-unused-st-heur-simp*[*simp*]:
‹*get-avdom* (*mark-unused-st-heur C T*) = *get-avdom T*›

‹*get-vdom* (*mark-unused-st-heur C T*) = *get-vdom T*›
⟨*proof*⟩

lemma *get-slow-ema-heur-alt-def*:
‹*RETURN o get-slow-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, -), *lcount*). *RETURN sema*)›
⟨*proof*⟩

lemma *get-fast-ema-heur-alt-def*:
‹*RETURN o get-fast-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, *ccount*), *lcount*). *RETURN fema*)›
⟨*proof*⟩

fun *get-conflict-count-since-last-restart-heur* :: ‹*twl-st-wl-heur* ⇒ *64 word*› **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*, *clvls*, *cach*, *lbd*,
outl, *stats*, (-, -, (*ccount*, -), -), *lcount*). *RETURN ccount*)›
⟨*proof*⟩

lemma *get-learned-count-alt-def*:
‹*RETURN o get-learned-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, -, *vdom*, *avdom*, *lcount*, *opts*). *RETURN lcount*)›
⟨*proof*⟩

I also played with *ema-reinit fast-ema* and *ema-reinit slow-ema*. Currently removed, to test the performance, I remove it.

definition *incr-restart-stat* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*incr-restart-stat* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*), *vdom*, *avdom*, *lcount*). *do*{
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *incr-restart stats*,
(*fast-ema*, *slow-ema*,
restart-info-restart-done res-info, *wasted*), *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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*), *vdom*, *avdom*, *lcount*). *do*{
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *incr-lrestart stats*,
(*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*),
vdom, *avdom*, *lcount*)
})›

definition *incr-wasted-st* :: ‹*64 word* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*incr-wasted-st* = (λ*waste* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*, φ), *vdom*, *avdom*, *lcount*). *do*{
(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
(*fast-ema*, *slow-ema*, *res-info*, *wasted*+*waste*, φ),
vdom, *avdom*, *lcount*)
})›

```
definition wasted-bytes-st :: ‹twl-st-wl-heur ⇒ 64 word› where
  ‹wasted-bytes-st = (λ(M, N, D, Q, W, vm, clvls, cach, lbd, outl, stats, (fast-ema, slow-ema,
      res-info, wasted, φ), vdom, avdom, lcount).
      wasted)›
```

```
definition opts-restart-st :: ‹twl-st-wl-heur ⇒ bool› where
  ‹opts-restart-st = (λ(M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
        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, clvls, cach, lbd, outl,
        stats, heur, vdom, avdom, lcount, opts, -). (opts-reduce opts))›
```

```
definition isasat-length-trail-st :: ‹twl-st-wl-heur ⇒ nat› where
  ‹isasat-length-trail-st S = isa-length-trail (get-trail-wl-heur S)›
```

```
lemma isasat-length-trail-st-alt-def:
  ‹isasat-length-trail-st = (λ(M, -). isa-length-trail M)›
  ⟨proof⟩
```

```
definition mop-isasat-length-trail-st :: ‹twl-st-wl-heur ⇒ nat nres› where
  ‹mop-isasat-length-trail-st S = do {
    ASSERT(isa-length-trail-pre (get-trail-wl-heur S));
    RETURN (isa-length-trail (get-trail-wl-heur S))
  }›
```

```
lemma mop-isasat-length-trail-st-alt-def:
  ‹mop-isasat-length-trail-st = (λ(M, -). do {
    ASSERT(isa-length-trail-pre M);
    RETURN (isa-length-trail M)
  })›
  ⟨proof⟩
```

```
definition get-pos-of-level-in-trail-imp-st :: ‹twl-st-wl-heur ⇒ nat ⇒ nat nres› where
‹get-pos-of-level-in-trail-imp-st S = get-pos-of-level-in-trail-imp (get-trail-wl-heur S)›
```

```
lemma get-pos-of-level-in-trail-imp-alt-def:
  ‹get-pos-of-level-in-trail-imp-st = (λ(M, -) L.  do {k ← get-pos-of-level-in-trail-imp M L; RETURN
k})›
  ⟨proof⟩
```

```
definition mop-clause-not-marked-to-delete-heur :: ‹- ⇒ nat ⇒ bool nres›
where
  ‹mop-clause-not-marked-to-delete-heur S C = do {
    ASSERT(clause-not-marked-to-delete-heur-pre (S, C));
    RETURN (clause-not-marked-to-delete-heur S C)
  }›
```

```
definition mop-arena-lbd-st where
  ‹mop-arena-lbd-st S =
    mop-arena-lbd (get-clauses-wl-heur S)›
```

lemma *mop-arena-lbd-st-alt-def*:
‹*mop-arena-lbd-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*get-clause-LBD-pre arena C*);
RETURN(*arena-lbd arena C*)
})›
⟨*proof*⟩

definition *mop-arena-status-st* **where**
‹*mop-arena-status-st S* =
mop-arena-status (*get-clauses-wl-heur S*)›

lemma *mop-arena-status-st-alt-def*:
‹*mop-arena-status-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*arena-is-valid-clause-vdom arena C*);
RETURN(*arena-status arena C*)
})›
⟨*proof*⟩

definition *mop-marked-as-used-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-marked-as-used-st S* =
mop-marked-as-used (*get-clauses-wl-heur S*)›

lemma *mop-marked-as-used-st-alt-def*:
‹*mop-marked-as-used-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*marked-as-used-pre arena C*);
RETURN(*marked-as-used arena C*)
})›
⟨*proof*⟩

definition *mop-arena-length-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length-st S* =
mop-arena-length (*get-clauses-wl-heur S*)›

lemma *mop-arena-length-st-alt-def*:
‹*mop-arena-length-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
RETURN (*arena-length arena C*)
})›
⟨*proof*⟩

definition *full-arena-length-st* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*full-arena-length-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *length arena*)›

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*) ⟹

$\mathit{get\text{-}clauses\text{-}wl}\ T \propto C\ !\ i = \mathit{access\text{-}lit\text{-}in\text{-}clauses\text{-}heur}\ S\ C\ i$›
for $S\ T\ C\ i$
⟨*proof*⟩

In an attempt to avoid using $?a + ?b + ?c = ?a + (?b + ?c)$

$?a + ?b = ?b + ?a$

$?b + (?a + ?c) = ?a + (?b + ?c)$

$?a * ?b * ?c = ?a * (?b * ?c)$

$?a * ?b = ?b * ?a$

$?b * (?a * ?c) = ?a * (?b * ?c)$

$\mathit{inf}\ (\mathit{inf}\ ?a\ ?b)\ ?c = \mathit{inf}\ ?a\ (\mathit{inf}\ ?b\ ?c)$

$\mathit{inf}\ ?a\ ?b = \mathit{inf}\ ?b\ ?a$

$\mathit{inf}\ ?b\ (\mathit{inf}\ ?a\ ?c) = \mathit{inf}\ ?a\ (\mathit{inf}\ ?b\ ?c)$

$\mathit{sup}\ (\mathit{sup}\ ?a\ ?b)\ ?c = \mathit{sup}\ ?a\ (\mathit{sup}\ ?b\ ?c)$

$\mathit{sup}\ ?a\ ?b = \mathit{sup}\ ?b\ ?a$

$\mathit{sup}\ ?b\ (\mathit{sup}\ ?a\ ?c) = \mathit{sup}\ ?a\ (\mathit{sup}\ ?b\ ?c)$

$\mathit{min}\ (\mathit{min}\ ?a\ ?b)\ ?c = \mathit{min}\ ?a\ (\mathit{min}\ ?b\ ?c)$

$\mathit{min}\ ?a\ ?b = \mathit{min}\ ?b\ ?a$

$\mathit{min}\ ?b\ (\mathit{min}\ ?a\ ?c) = \mathit{min}\ ?a\ (\mathit{min}\ ?b\ ?c)$

$\mathit{max}\ (\mathit{max}\ ?a\ ?b)\ ?c = \mathit{max}\ ?a\ (\mathit{max}\ ?b\ ?c)$

$\mathit{max}\ ?a\ ?b = \mathit{max}\ ?b\ ?a$

$\mathit{max}\ ?b\ (\mathit{max}\ ?a\ ?c) = \mathit{max}\ ?a\ (\mathit{max}\ ?b\ ?c)$

$\mathit{coprime}\ ?b\ ?a = \mathit{coprime}\ ?a\ ?b$

$(?a\ \mathit{dvd}\ ?c - ?b) = (?a\ \mathit{dvd}\ ?b - ?c)$

$(?a\ @\ ?b)\ @\ ?c = ?a\ @\ ?b\ @\ ?c$

$\mathit{gcd}\ (\mathit{gcd}\ ?a\ ?b)\ ?c = \mathit{gcd}\ ?a\ (\mathit{gcd}\ ?b\ ?c)$

$\mathit{gcd}\ ?a\ ?b = \mathit{gcd}\ ?b\ ?a$

$\mathit{gcd}\ ?b\ (\mathit{gcd}\ ?a\ ?c) = \mathit{gcd}\ ?a\ (\mathit{gcd}\ ?b\ ?c)$

$\mathit{lcm}\ (\mathit{lcm}\ ?a\ ?b)\ ?c = \mathit{lcm}\ ?a\ (\mathit{lcm}\ ?b\ ?c)$

$\mathit{lcm}\ ?a\ ?b = \mathit{lcm}\ ?b\ ?a$

$\mathit{lcm}\ ?b\ (\mathit{lcm}\ ?a\ ?c) = \mathit{lcm}\ ?a\ (\mathit{lcm}\ ?b\ ?c)$

$?a \cap\# ?b \cap\# ?c = ?a \cap\# (?b \cap\# ?c)$

$?a \cap\# ?b = ?b \cap\# ?a$

$?b \cap\# (?a \cap\# ?c) = ?a \cap\# (?b \cap\# ?c)$

$?a \cup\# ?b \cup\# ?c = ?a \cup\# (?b \cup\# ?c)$

$?a \cup\# ?b = ?b \cup\# ?a$

$?b \cup\# (?a \cup\# ?c) = ?a \cup\# (?b \cup\# ?c)$

$\mathit{signed.min}\ (\mathit{signed.min}\ ?a\ ?b)\ ?c = \mathit{signed.min}\ ?a\ (\mathit{signed.min}\ ?b\ ?c)$

$\mathit{signed.min}\ ?a\ ?b = \mathit{signed.min}\ ?b\ ?a$

$\mathit{signed.min}\ ?b\ (\mathit{signed.min}\ ?a\ ?c) = \mathit{signed.min}\ ?a\ (\mathit{signed.min}\ ?b\ ?c)$

$\mathit{signed.max}\ (\mathit{signed.max}\ ?a\ ?b)\ ?c = \mathit{signed.max}\ ?a\ (\mathit{signed.max}\ ?b\ ?c)$

$\mathit{signed.max}\ ?a\ ?b = \mathit{signed.max}\ ?b\ ?a$

$\mathit{signed.max}\ ?b\ (\mathit{signed.max}\ ?a\ ?c) = \mathit{signed.max}\ ?a\ (\mathit{signed.max}\ ?b\ ?c)$

(*?a* && *?b*) && *?c* = *?a* && *?b* && *?c*

?a && *?b* = *?b* && *?a*

?b && *?a* && *?c* = *?a* && *?b* && *?c*

(*?a* || *?b*) || *?c* = *?a* || *?b* || *?c*

?a || *?b* = *?b* || *?a*

?b || *?a* || *?c* = *?a* || *?b* || *?c*

(*?a xor ?b*) *xor ?c* = *?a xor ?b xor ?c*

?a xor ?b = *?b xor ?a*

?b xor ?a xor ?c = *?a xor ?b xor ?c* everywhere.

lemma *all-lits-simps*[*simp*]:
  ‹*all-lits N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-lits N* (*NE* + *UE* + *NS* + *US*)›
  ‹*all-atms N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-atms N* (*NE* + *UE* + *NS* + *US*)›
  ⟨*proof*⟩

lemma *clause-not-marked-to-delete-heur-alt-def*:
  ‹*RETURN* ∘∘ *clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*.
    RETURN (*arena-status arena C* ≠ *DELETED*))›
  ⟨*proof*⟩

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

type-synonym *tri-bool-assn* = ‹*8 word*›

definition ‹*tri-bool-rel-aux* ≡ { (*0*::*nat*,*None*), (*2*,*Some True*), (*3*,*Some False*) }›
definition ‹*tri-bool-rel* ≡ *unat-rel′ TYPE*(*8*) *O tri-bool-rel-aux*›
abbreviation ‹*tri-bool-assn* ≡ *pure tri-bool-rel*›
lemmas [*fcomp-norm-unfold*] = *tri-bool-rel-def*[*symmetric*]

lemma *tri-bool-UNSET-refine-aux*: ‹(*0*,*UNSET*)∈*tri-bool-rel-aux*›
  and *tri-bool-SET-TRUE-refine-aux*: ‹(*2*,*SET-TRUE*)∈*tri-bool-rel-aux*›
  and *tri-bool-SET-FALSE-refine-aux*: ‹(*3*,*SET-FALSE*)∈*tri-bool-rel-aux*›
  and *tri-bool-eq-refine-aux*: ‹((=),*tri-bool-eq*) ∈ *tri-bool-rel-aux*→*tri-bool-rel-aux*→*bool-rel*›
  ⟨*proof*⟩

sepref-def *tri-bool-UNSET-impl* **is** [] ‹*uncurry0* (*RETURN 0*)› :: ‹$\textit{unit-assn}^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)›
  ⟨*proof*⟩

sepref-def *tri-bool-SET-TRUE-impl* **is** [] ‹*uncurry0* (*RETURN 2*)› :: ‹$\textit{unit-assn}^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)›
  ⟨*proof*⟩

sepref-def *tri-bool-SET-FALSE-impl* **is** [] ‹*uncurry0* (*RETURN 3*)› :: ‹$\textit{unit-assn}^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)›
  ⟨*proof*⟩

sepref-def *tri-bool-eq-impl* [*llvm-inline*] **is** [] ‹*uncurry* (*RETURN oo* (=))› :: ‹$(\textit{unat-assn}'\ \textit{TYPE}(8))^k$ $*_a$ $(\textit{unat-assn}'\ \textit{TYPE}(8))^k \rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] =
  tri-bool-UNSET-impl.refine[*FCOMP tri-bool-UNSET-refine-aux*]

tri-bool-SET-TRUE-impl.refine[*FCOMP tri-bool-SET-TRUE-refine-aux*]
tri-bool-SET-FALSE-impl.refine[*FCOMP tri-bool-SET-FALSE-refine-aux*]
tri-bool-eq-impl.refine[*FCOMP tri-bool-eq-refine-aux*]

type-synonym *trail-pol-fast-assn* =
⟨*32 word array-list64* × *tri-bool-assn larray64* × *32 word larray64* ×
64 word larray64 × *32 word* ×
32 word array-list64⟩

sepref-def *DECISION-REASON-impl* **is** ⟨*uncurry0* (*RETURN DECISION-REASON*)⟩
:: ⟨*unit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*⟩
⟨*proof*⟩

definition *trail-pol-fast-assn* :: ⟨*trail-pol* ⇒ *trail-pol-fast-assn* ⇒ *assn*⟩ **where**
⟨*trail-pol-fast-assn* ≡
arl64-assn unat-lit-assn $\times_a$ *larray64-assn* (*tri-bool-assn*) $\times_a$
larray64-assn uint32-nat-assn $\times_a$
larray64-assn sint64-nat-assn $\times_a$ *uint32-nat-assn* $\times_a$
arl64-assn uint32-nat-assn⟩

Code generation

Conversion between incomplete and complete mode **sepref-def** *count-decided-pol-impl* **is** ⟨*RETURN o count-decided-pol*⟩ :: ⟨*trail-pol-fast-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*⟩
⟨*proof*⟩

sepref-def *get-level-atm-fast-code*
is ⟨*uncurry* (*RETURN oo get-level-atm-pol*)⟩
:: ⟨[*get-level-atm-pol-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *atom-assn*^{k} → *uint32-nat-assn*⟩
⟨*proof*⟩

sepref-def *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*⟩
⟨*proof*⟩

sepref-def *polarity-pol-fast-code*
is ⟨*uncurry* (*RETURN oo polarity-pol*)⟩
:: ⟨[*uncurry polarity-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} → *tri-bool-assn*⟩
⟨*proof*⟩

sepref-register *isa-length-trail*
sepref-def *isa-length-trail-fast-code*
is ⟨*RETURN o isa-length-trail*⟩
:: ⟨[λ-. *True*]$_a$ *trail-pol-fast-assn*^{k} → *snat-assn′ TYPE*(*64*)⟩
⟨*proof*⟩

sepref-def *mop-isa-length-trail-fast-code*

is ‹*mop-isa-length-trail*›
:: ‹*trail-pol-fast-assn*^{k} $\rightarrow_a$ *snat-assn′ TYPE(64)*›
⟨*proof*⟩

sepref-def *cons-trail-Propagated-tr-fast-code*
is ‹*uncurry2* (*cons-trail-Propagated-tr*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *trail-pol-fast-assn*^{d} $\rightarrow_a$ *trail-pol-fast-assn*›
⟨*proof*⟩

sepref-def *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*›
⟨*proof*⟩

sepref-def *tl-trail-proped-tr-fast-code*
is ‹*RETURN o tl-trail-propedt-tr*›
:: ‹[*tl-trail-propedt-tr-pre*]$_a$
trail-pol-fast-assn^d $\rightarrow$ *trail-pol-fast-assn*›
⟨*proof*⟩

sepref-def *lit-of-last-trail-fast-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹[$\lambda(M,$ -). $M \neq []$]$_a$ *trail-pol-fast-assn*^{k} $\rightarrow$ *unat-lit-assn*›
⟨*proof*⟩

sepref-def *cons-trail-Decided-tr-fast-code*
is ‹*uncurry* (*RETURN oo cons-trail-Decided-tr*)›
:: ‹[*cons-trail-Decided-tr-pre*]$_a$
unat-lit-assn^k $*_a$ *trail-pol-fast-assn*^{d} $\rightarrow$ *trail-pol-fast-assn*›
⟨*proof*⟩

sepref-def *defined-atm-fast-code*
is ‹*uncurry* (*RETURN oo defined-atm-pol*)›
:: ‹[*uncurry defined-atm-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow$ *bool1-assn*›
⟨*proof*⟩

sepref-register *get-propagation-reason-raw-pol*
sepref-def *get-propagation-reason-fast-code*
is ‹*uncurry get-propagation-reason-raw-pol*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-register *isa-trail-nth*

sepref-def *isa-trail-nth-fast-code*
  is ‹*uncurry isa-trail-nth*›
  :: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
  ⟨*proof*⟩

sepref-def *tl-trail-tr-no-CS-fast-code*
  is ‹*RETURN o tl-trailt-tr-no-CS*›
  :: ‹$[tl\text{-}trailt\text{-}tr\text{-}no\text{-}CS\text{-}pre]_a$
      $trail\text{-}pol\text{-}fast\text{-}assn^d \rightarrow trail\text{-}pol\text{-}fast\text{-}assn$›
  ⟨*proof*⟩

sepref-def *trail-conv-back-imp-fast-code*
  is ‹*uncurry trail-conv-back-imp*›
  :: ‹$uint32\text{-}nat\text{-}assn^k *_a trail\text{-}pol\text{-}fast\text{-}assn^d \rightarrow_a trail\text{-}pol\text{-}fast\text{-}assn$›
  ⟨*proof*⟩

sepref-def *get-pos-of-level-in-trail-imp-fast-code*
  is ‹*uncurry get-pos-of-level-in-trail-imp*›
  :: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
  ⟨*proof*⟩

sepref-def *get-the-propagation-reason-fast-code*
  is ‹*uncurry get-the-propagation-reason-pol*›
  :: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a snat\text{-}option\text{-}assn'\ TYPE(64)$›
  ⟨*proof*⟩

experiment begin

export-llvm
  tri-bool-UNSET-impl
  tri-bool-SET-TRUE-impl
  tri-bool-SET-FALSE-impl
  DECISION-REASON-impl
  count-decided-pol-impl
  get-level-atm-fast-code
  get-level-fast-code
  polarity-pol-fast-code
  isa-length-trail-fast-code
  cons-trail-Propagated-tr-fast-code
  tl-trail-tr-fast-code
  tl-trail-proped-tr-fast-code
  lit-of-last-trail-fast-code
  cons-trail-Decided-tr-fast-code
  defined-atm-fast-code
  get-propagation-reason-fast-code
  isa-trail-nth-fast-code
  tl-trail-tr-no-CS-fast-code
  trail-conv-back-imp-fast-code
  get-pos-of-level-in-trail-imp-fast-code
  get-the-propagation-reason-fast-code

end

end
theory *IsaSAT-Lookup-Conflict-LLVM*
imports
  IsaSAT-Lookup-Conflict
  IsaSAT-Trail-LLVM
  IsaSAT-Clauses-LLVM
  LBD-LLVM
begin

sepref-register *set-lookup-conflict-aa*
type-synonym *lookup-clause-assn* = ‹*32 word* × (*1 word*) *ptr*›

type-synonym (**in** −) *option-lookup-clause-assn* = ‹*1 word* × *lookup-clause-assn*›

type-synonym (**in** −) *out-learned-assn* = ‹*32 word array-list64*›

abbreviation (**in** −) *out-learned-assn* :: ‹*out-learned* ⇒ *out-learned-assn* ⇒ *assn*› **where**
  ‹*out-learned-assn* ≡ *arl64-assn unat-lit-assn*›

definition *minimize-status-int-rel* :: ‹(*nat* × *minimize-status*) *set*› **where**
‹*minimize-status-int-rel* = {(*0*, *SEEN-UNKNOWN*), (*1*, *SEEN-FAILED*), (*2*, *SEEN-REMOVABLE*)}›

abbreviation *minimize-status-ref-rel* **where**
‹*minimize-status-ref-rel* ≡ *snat-rel′ TYPE*(*8*)›

abbreviation *minimize-status-ref-assn* **where**
  ‹*minimize-status-ref-assn* ≡ *pure minimize-status-ref-rel*›

definition *minimize-status-rel* :: ‹-› **where**
‹*minimize-status-rel* = *minimize-status-ref-rel O minimize-status-int-rel*›

abbreviation *minimize-status-assn* :: ‹-› **where**
‹*minimize-status-assn* ≡ *pure minimize-status-rel*›

lemma *minimize-status-assn-alt-def*:
  ‹*minimize-status-assn* = *pure* (*snat-rel O minimize-status-int-rel*)›
  ⟨*proof*⟩

lemmas [*fcomp-norm-unfold*] = *minimize-status-assn-alt-def*[*symmetric*]

definition *minimize-status-rel-eq* :: ‹*minimize-status* ⇒ *minimize-status* ⇒ *bool*› **where**
 [*simp*]: ‹*minimize-status-rel-eq* = (=)›

lemma *minimize-status-rel-eq*:
   ‹((=), *minimize-status-rel-eq*) ∈ *minimize-status-int-rel* → *minimize-status-int-rel* → *bool-rel*›
  ⟨*proof*⟩

sepref-def *minimize-status-rel-eq-impl*
  is [] ‹*uncurry* (*RETURN oo* (=))›
  :: ‹$minimize\text{-}status\text{-}ref\text{-}assn^k *_a minimize\text{-}status\text{-}ref\text{-}assn^k \rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

sepref-register *minimize-status-rel-eq*

lemmas [*sepref-fr-rules*] = *minimize-status-rel-eq-impl.refine*[*unfolded convert-fref*, *FCOMP minimize-status-rel-eq*]

lemma
  SEEN-FAILED-rel: ‹(*1*, *SEEN-FAILED*) ∈ *minimize-status-int-rel*› **and**
  SEEN-UNKNOWN-rel: ‹(*0*, *SEEN-UNKNOWN*) ∈ *minimize-status-int-rel*› **and**
  SEEN-REMOVABLE-rel: ‹(*2*, *SEEN-REMOVABLE*) ∈ *minimize-status-int-rel*›
  ⟨*proof*⟩

sepref-def *SEEN-FAILED-impl*
  is [] ‹*uncurry0* (*RETURN 1*)›
  :: ‹*unit-assn*^{k} $\rightarrow_a$ *minimize-status-ref-assn*›
  ⟨*proof*⟩

sepref-def *SEEN-UNKNOWN-impl*
  is [] ‹*uncurry0* (*RETURN 0*)›
  :: ‹*unit-assn*^{k} $\rightarrow_a$ *minimize-status-ref-assn*›
  ⟨*proof*⟩

sepref-def *SEEN-REMOVABLE-impl*
  is [] ‹*uncurry0* (*RETURN 2*)›
  :: ‹*unit-assn*^{k} $\rightarrow_a$ *minimize-status-ref-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *SEEN-FAILED-impl.refine*[*FCOMP SEEN-FAILED-rel*]
  SEEN-UNKNOWN-impl.refine[*FCOMP SEEN-UNKNOWN-rel*]
  SEEN-REMOVABLE-impl.refine[*FCOMP SEEN-REMOVABLE-rel*]

definition *option-bool-impl-rel* **where**
  ‹*option-bool-impl-rel* = *bool1-rel O option-bool-rel*›

abbreviation *option-bool-impl-assn* :: ‹-› **where**
‹*option-bool-impl-assn* ≡ *pure* (*option-bool-impl-rel*)›

lemma *option-bool-impl-assn-alt-def*:
  ‹*option-bool-impl-assn* = *hr-comp bool1-assn option-bool-rel*›
  ⟨*proof*⟩

lemmas [*fcomp-norm-unfold*] = *option-bool-impl-assn-alt-def*[*symmetric*]
  option-bool-impl-rel-def[*symmetric*]

lemma *Some-rel*: ‹(λ-. *True*, *ISIN*) ∈ *bool-rel* → *option-bool-rel*›
  ⟨*proof*⟩

sepref-def *Some-impl*
  is [] ‹*RETURN o* (λ-. *True*)›
  :: ‹*bool1-assn*^{k} $\rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *Some-impl.refine*[*FCOMP Some-rel*]

lemma *is-Notin-rel*: ‹(λ*x*. ¬*x*, *is-NOTIN*) ∈ *option-bool-rel* → *bool-rel*›
  ⟨*proof*⟩

sepref-def *is-Notin-impl*
  is [] ‹*RETURN o* (λ*x*. ¬*x*)›
  :: ‹*bool1-assn*^{k} $\rightarrow_a$ *bool1-assn*›

⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *is-Notin-impl.refine*[*FCOMP is-Notin-rel*]

lemma *NOTIN-rel*: ‹(*False*, *NOTIN*) ∈ *option-bool-rel*›
⟨*proof*⟩

sepref-def *NOTIN-impl*
is [] ‹*uncurry0* (*RETURN False*)›
:: ‹$unit\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *NOTIN-impl.refine*[*FCOMP NOTIN-rel*]

definition (**in** −) *lookup-clause-rel-assn*
:: ‹*lookup-clause-rel* ⇒ *lookup-clause-assn* ⇒ *assn*›
where
‹*lookup-clause-rel-assn* ≡ ($uint32\text{-}nat\text{-}assn \times_a array\text{-}assn\ option\text{-}bool\text{-}impl\text{-}assn$)›

definition (**in** −)*conflict-option-rel-assn*
:: ‹*conflict-option-rel* ⇒ *option-lookup-clause-assn* ⇒ *assn*›
where
‹*conflict-option-rel-assn* ≡ ($bool1\text{-}assn \times_a lookup\text{-}clause\text{-}rel\text{-}assn$)›

lemmas [*fcomp-norm-unfold*] = *conflict-option-rel-assn-def*[*symmetric*]
lookup-clause-rel-assn-def[*symmetric*]

definition (**in** −)*ana-refinement-fast-rel* **where**
‹*ana-refinement-fast-rel* ≡ $snat\text{-}rel'\ TYPE(64) \times_r unat\text{-}rel'\ TYPE(32) \times_r bool1\text{-}rel$›

abbreviation (**in** −)*ana-refinement-fast-assn* **where**
‹*ana-refinement-fast-assn* ≡ $sint64\text{-}nat\text{-}assn \times_a uint32\text{-}nat\text{-}assn \times_a bool1\text{-}assn$›

lemma *ana-refinement-fast-assn-def*:
‹*ana-refinement-fast-assn* = *pure ana-refinement-fast-rel*›
⟨*proof*⟩

abbreviation (**in** −)*analyse-refinement-fast-assn* **where**
‹*analyse-refinement-fast-assn* ≡
arl64-assn ana-refinement-fast-assn›

lemma *lookup-clause-assn-is-None-alt-def*:
‹*RETURN o lookup-clause-assn-is-None* = (λ(*b*, -, -). *RETURN b*)›
⟨*proof*⟩

sepref-def *lookup-clause-assn-is-None-impl*
is ‹*RETURN o lookup-clause-assn-is-None*›
:: ‹$conflict\text{-}option\text{-}rel\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

lemma *size-lookup-conflict-alt-def*:
‹*RETURN o size-lookup-conflict* = (λ(-, *b*, -). *RETURN b*)›

⟨*proof*⟩

sepref-def *size-lookup-conflict-impl*
is ‹*RETURN o size-lookup-conflict*›
:: ‹$\mathit{conflict\text{-}option\text{-}rel\text{-}assn}^k \rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *is-in-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-lookup-conflict*)›
:: ‹$[\lambda((n, xs), L).\ \mathit{atm\text{-}of}\ L < \mathit{length}\ xs]_a$
$\mathit{lookup\text{-}clause\text{-}rel\text{-}assn}^k *_a \mathit{unat\text{-}lit\text{-}assn}^k \rightarrow$ *bool1-assn*›
⟨*proof*⟩

lemma *lookup-clause-assn-is-empty-alt-def*:
‹*lookup-clause-assn-is-empty* = (λS. *size-lookup-conflict S* = *0*)›
⟨*proof*⟩

sepref-def *lookup-clause-assn-is-empty-impl*
is ‹*RETURN o lookup-clause-assn-is-empty*›
:: ‹$\mathit{conflict\text{-}option\text{-}rel\text{-}assn}^k \rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

definition *the-lookup-conflict* :: ‹*conflict-option-rel* ⇒ -› **where**
‹*the-lookup-conflict* = *snd*›

lemma *the-lookup-conflict-alt-def*:
‹*RETURN o the-lookup-conflict* = (λ(-, (*n*, *xs*)). *RETURN* (*n*, *xs*))›
⟨*proof*⟩

sepref-def *the-lookup-conflict-impl*
is ‹*RETURN o the-lookup-conflict*›
:: ‹$\mathit{conflict\text{-}option\text{-}rel\text{-}assn}^d \rightarrow_a$ *lookup-clause-rel-assn*›
⟨*proof*⟩

definition *Some-lookup-conflict* :: ‹- ⇒ *conflict-option-rel*› **where**
‹*Some-lookup-conflict xs* = (*False*, *xs*)›

lemma *Some-lookup-conflict-alt-def*:
‹*RETURN o Some-lookup-conflict* = (λ*xs*. *RETURN* (*False*, *xs*))›
⟨*proof*⟩

sepref-def *Some-lookup-conflict-impl*
is ‹*RETURN o Some-lookup-conflict*›
:: ‹$\mathit{lookup\text{-}clause\text{-}rel\text{-}assn}^d \rightarrow_a$ *conflict-option-rel-assn*›
⟨*proof*⟩
sepref-register *Some-lookup-conflict*

type-synonym *cach-refinement-l-assn* = ‹*8 word ptr* × *32 word array-list64*›

definition (**in** −) *cach-refinement-l-assn* :: ‹- ⇒ *cach-refinement-l-assn* ⇒ -› **where**
‹*cach-refinement-l-assn* ≡ *array-assn minimize-status-assn* $\times_a$ *arl64-assn atom-assn*›

```
sepref-register conflict-min-cach-l
sepref-def delete-from-lookup-conflict-code
  is ‹uncurry delete-from-lookup-conflict›
  :: ‹unat-lit-assn^k *_a lookup-clause-rel-assn^d →_a lookup-clause-rel-assn›
  ⟨proof⟩

lemma arena-is-valid-clause-idx-le-uint64-max:
  ‹arena-is-valid-clause-idx be bd ⟹
    length be ≤ sint64-max ⟹
   bd + arena-length be bd ≤ sint64-max›
  ‹arena-is-valid-clause-idx be bd ⟹ length be ≤ sint64-max ⟹
   bd ≤ sint64-max›
  ⟨proof⟩

lemma add-to-lookup-conflict-alt-def:
  ‹RETURN oo add-to-lookup-conflict = (λL (n, xs). RETURN (if xs ! atm-of L = NOTIN then n + 1
else n,
      xs[atm-of L := ISIN (is-pos L)]))›
  ⟨proof⟩

sepref-register ISIN NOTIN atm-of add-to-lookup-conflict

sepref-def add-to-lookup-conflict-impl
  is ‹uncurry (RETURN oo add-to-lookup-conflict)›
  :: ‹[λ(L, (n, xs)). atm-of L < length xs ∧ n + 1 ≤ uint32-max]_a
      unat-lit-assn^k *_a (lookup-clause-rel-assn)^d → lookup-clause-rel-assn›
  ⟨proof⟩



lemma isa-lookup-conflict-merge-alt-def:
  ‹isa-lookup-conflict-merge i0 = (λM N i zs clvls outl.
 do {
     let xs = the-lookup-conflict zs;
    ASSERT( arena-is-valid-clause-idx N i);
   (-, clvls, zs, outl) ← WHILE_T^λ(i::nat, clvls :: nat, zs, outl).     length (snd zs) = length (snd xs) ∧          Suc (fst zs)
       (λ(j :: nat, clvls, zs, outl). j < i + arena-length N i)
       (λ(j :: nat, clvls, zs, 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));
            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, outl)
         })
       (i + i0, clvls, xs, outl);
     RETURN (Some-lookup-conflict zs, clvls, outl)
   })›
  ⟨proof⟩

sepref-def resolve-lookup-conflict-merge-fast-code
```

is ‹*uncurry5 isa-set-lookup-conflict-aa*›
:: ‹$[\lambda(((((M, N), i), (\text{-}, xs)), \text{-}), out).$
$\quad length\ N \leq sint64\text{-}max]_a$
$\quad trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d *_a$
$\quad\quad uint32\text{-}nat\text{-}assn^k *_a out\text{-}learned\text{-}assn^d \rightarrow$
$\quad conflict\text{-}option\text{-}rel\text{-}assn \times_a uint32\text{-}nat\text{-}assn \times_a out\text{-}learned\text{-}assn$›
⟨*proof*⟩

sepref-register *isa-resolve-merge-conflict-gt2*

lemma *arena-is-valid-clause-idx-le-uint64-max2*:
‹$arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ be\ bd \Longrightarrow$
$\quad length\ be \leq sint64\text{-}max \Longrightarrow$
$\quad bd + arena\text{-}length\ be\ bd \leq sint64\text{-}max$›
‹$arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ be\ bd \Longrightarrow length\ be \leq sint64\text{-}max \Longrightarrow$
$\quad bd < sint64\text{-}max$›
⟨*proof*⟩

sepref-def *resolve-merge-conflict-fast-code*
is ‹*uncurry5 isa-resolve-merge-conflict-gt2*›
:: ‹$[uncurry5\ (\lambda M\ N\ i\ (b, xs)\ clvls\ outl.\ length\ N \leq sint64\text{-}max)]_a$
$\quad trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d *_a$
$\quad\quad uint32\text{-}nat\text{-}assn^k *_a out\text{-}learned\text{-}assn^d \rightarrow$
$\quad conflict\text{-}option\text{-}rel\text{-}assn \times_a uint32\text{-}nat\text{-}assn \times_a out\text{-}learned\text{-}assn$›
⟨*proof*⟩

sepref-def *atm-in-conflict-code*
is ‹$uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict\text{-}lookup)$›
:: ‹$[uncurry\ atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre]_a$
$\quad atom\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k \rightarrow bool1\text{-}assn$›
⟨*proof*⟩

sepref-def *conflict-min-cach-l-code*
is ‹$uncurry\ (RETURN\ oo\ conflict\text{-}min\text{-}cach\text{-}l)$›
:: ‹$[conflict\text{-}min\text{-}cach\text{-}l\text{-}pre]_a\ cach\text{-}refinement\text{-}l\text{-}assn^k *_a atom\text{-}assn^k \rightarrow minimize\text{-}status\text{-}assn$›
⟨*proof*⟩

lemma *conflict-min-cach-set-failed-l-alt-def*:
‹$conflict\text{-}min\text{-}cach\text{-}set\text{-}failed\text{-}l = (\lambda(cach, sup)\ L.\ do\ \{$
$\quad ASSERT(L < length\ cach);$
$\quad ASSERT(length\ sup \leq 1 + uint32\text{-}max\ div\ 2);$
$\quad let\ b = (cach\ !\ L = SEEN\text{-}UNKNOWN);$
$\quad RETURN\ (cach[L := SEEN\text{-}FAILED],\ if\ b\ \ then\ sup\ @\ [L]\ else\ sup)$
$\})$›
⟨*proof*⟩

lemma *le-uint32-max-div2-le-uint32-max*: ‹$a2' \leq Suc\ (uint32\text{-}max\ div\ 2) \Longrightarrow a2' < uint32\text{-}max$›
⟨*proof*⟩

sepref-def *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹$cach\text{-}refinement\text{-}l\text{-}assn^d *_a atom\text{-}assn^k \rightarrow_a cach\text{-}refinement\text{-}l\text{-}assn$›
⟨*proof*⟩

lemma *conflict-min-cach-set-removable-l-alt-def*:
‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
    ASSERT(*L* < *length cach*);
    ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
    let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
    RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if b then sup* @ [*L*] *else sup*)
  })›
⟨*proof*⟩

sepref-def *conflict-min-cach-set-removable-l-code*
is ‹*uncurry conflict-min-cach-set-removable-l*›
:: ‹*cach-refinement-l-assn*^{d} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *cach-refinement-l-assn*›
⟨*proof*⟩

lemma *lookup-conflict-size-impl-alt-def*:
‹*RETURN o* (λ(*n*, *xs*). *n*) = (λ(*n*, *xs*). *RETURN n*)›
⟨*proof*⟩

sepref-def *lookup-conflict-size-impl*
is [] ‹*RETURN o* (λ(*n*, *xs*). *n*)›
:: ‹*lookup-clause-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
⟨*proof*⟩

sepref-register *lookup-conflict-remove1*

sepref-register *isa-lit-redundant-rec-wl-lookup*

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-alt-def*:
‹*RETURN o set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). *RETURN* (*True*, *n*, *xs*))›
⟨*proof*⟩

sepref-def *set-lookup-empty-conflict-to-none-imple*
is ‹*RETURN o set-lookup-empty-conflict-to-none*›
:: ‹*lookup-clause-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*›
⟨*proof*⟩

lemma *isa-mark-failed-lits-stackI*:
assumes
‹*length ba* ≤ *Suc* (*uint32-max div 2*)› **and**
‹*a1'* < *length ba*›
shows ‹*Suc a1'* ≤ *uint32-max*›

⟨*proof*⟩

sepref-register *conflict-min-cach-set-failed-l*
sepref-def *isa-mark-failed-lits-stack-fast-code*
  is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
  :: ‹[λ((*N*, -), -). *length N* ≤ *sint64-max*]$_a$
    arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{k} $*_a$ *cach-refinement-l-assn*^{d} →
    cach-refinement-l-assn›
  ⟨*proof*⟩

sepref-def *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 ≤ *sint64-max*]$_a$
      arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{d} →
      unat-lit-assn $\times_a$ *analyse-refinement-fast-assn*›
  ⟨*proof*⟩

sepref-def *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} $*_a$
    (*ana-refinement-fast-assn*)^k
     → *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

sepref-register *arena-lit*
sepref-def *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} $*_a$ *sint64-nat-assn*^{k} →
      ana-refinement-fast-assn›
  ⟨*proof*⟩

lemma *isa-lit-redundant-rec-wl-lookupI*:
  assumes
    ‹*length ba* ≤ *Suc* (*uint32-max div 2*)›
  shows ‹*length ba* < *uint32-max*›
  ⟨*proof*⟩

lemma *arena-lit-pre-le*: ‹
      arena-lit-pre a i ⟹ *length a* ≤ *sint64-max* ⟹ *i* ≤ *sint64-max*›
   ⟨*proof*⟩

lemma *get-propagation-reason-pol-get-propagation-reason-pol-raw*: ‹*do* {
   C ← *get-propagation-reason-pol M* (−*L*);
   case C of
     Some C ⇒ *f C*
   | *None* ⇒ *g*
         } = *do* {
   C ← *get-propagation-reason-raw-pol M* (−*L*);
   if C ≠ *DECISION-REASON then f C else g*
         }›
  ⟨*proof*⟩

sepref-register *atm-in-conflict-lookup*
sepref-def *lit-redundant-rec-wl-lookup-fast-code*
  is ‹*uncurry5* (*isa-lit-redundant-rec-wl-lookup*)›
  :: ‹[λ(((((*M*, *NU*), *D*), *cach*), *analysis*), *lbd*). *length NU* ≤ *sint64-max*]$_a$
      trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ (*lookup-clause-rel-assn*)^k $*_a$
        cach-refinement-l-assn^d $*_a$ *analyse-refinement-fast-assn*^{d} $*_a$ *lbd-assn*^{k} →
      cach-refinement-l-assn $\times_a$ *analyse-refinement-fast-assn* $\times_a$ *bool1-assn*›
  ⟨*proof*⟩

sepref-def *delete-index-and-swap-code*
  is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
  :: ‹[λ(*xs*, *i*). *i* < *length xs*]$_a$
      (*arl64-assn unat-lit-assn*)^d $*_a$ *sint64-nat-assn*^{k} → *arl64-assn unat-lit-assn*›
  ⟨*proof*⟩

sepref-def *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 $*_a$ *sint32-nat-assn*^{k} → *lookup-clause-rel-assn*›
  ⟨*proof*⟩

lemma *uint32-max-ge0*: ‹*0* < *uint32-max*› ⟨*proof*⟩

sepref-def *literal-redundant-wl-lookup-fast-code*
  is ‹*uncurry5 isa-literal-redundant-wl-lookup*›
  :: ‹[λ(((((*M*, *NU*), *D*), *cach*), *L*), *lbd*). *length NU* ≤ *sint64-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 $\times_a$ *analyse-refinement-fast-assn* $\times_a$ *bool1-assn*›
  ⟨*proof*⟩

sepref-def *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›
  ⟨*proof*⟩

sepref-def *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* ≤ *sint64-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 $\times_a$ *cach-refinement-l-assn* $\times_a$ *out-learned-assn*›
  ⟨*proof*⟩

lemma *isasat-lookup-merge-eq2-alt-def*:
  ‹*isasat-lookup-merge-eq2 L M N C* = (λ*zs clvls outl. do* {
    let zs = *the-lookup-conflict zs*;
    ASSERT(*arena-lit-pre N C*);
    ASSERT(*arena-lit-pre N* (*C*+*1*));

let L0 = arena-lit N C;
let L′ = (if L0 = L then arena-lit N (C + 1) else L0);
ASSERT(get-level-pol-pre (M, L′));
ASSERT(get-level-pol M L′ ≤ Suc (uint32-max div 2));
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(Some-lookup-conflict zs, clvls, outl)
})›
⟨*proof*⟩

sepref-def *isasat-lookup-merge-eq2-fast-code*
is ‹*uncurry6 isasat-lookup-merge-eq2*›
:: ‹$[\lambda((((((L, M), NU), \text{-}), \text{-}), \text{-}), \text{-}).\ length\ NU \le sint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a trail\text{-}pol\text{-}fast\text{-}assn^k\ *_a arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a$
$conflict\text{-}option\text{-}rel\text{-}assn^d *_a uint32\text{-}nat\text{-}assn^k *_a out\text{-}learned\text{-}assn^d \rightarrow$
$conflict\text{-}option\text{-}rel\text{-}assn \times_a uint32\text{-}nat\text{-}assn \times_a out\text{-}learned\text{-}assn$›
⟨*proof*⟩

experiment begin

export-llvm
nat-lit-eq-impl
minimize-status-rel-eq-impl
SEEN-FAILED-impl
SEEN-UNKNOWN-impl
SEEN-REMOVABLE-impl
Some-impl
is-Notin-impl
NOTIN-impl
lookup-clause-assn-is-None-impl
size-lookup-conflict-impl
is-in-conflict-code
lookup-clause-assn-is-empty-impl
the-lookup-conflict-impl
Some-lookup-conflict-impl
delete-from-lookup-conflict-code
add-to-lookup-conflict-impl
resolve-lookup-conflict-merge-fast-code
resolve-merge-conflict-fast-code
atm-in-conflict-code
conflict-min-cach-l-code
conflict-min-cach-set-failed-l-code
conflict-min-cach-set-removable-l-code
lookup-conflict-size-impl
set-lookup-empty-conflict-to-none-imple
isa-mark-failed-lits-stack-fast-code
isa-get-literal-and-remove-of-analyse-wl-fast-code
ana-lookup-conv-lookup-fast-code
lit-redundant-reason-stack-wl-lookup-fast-code
lit-redundant-rec-wl-lookup-fast-code
delete-index-and-swap-code

lookup-conflict-upd-None-code
literal-redundant-wl-lookup-fast-code
conflict-remove1-code
minimize-and-extract-highest-lookup-conflict-fast-code
isasat-lookup-merge-eq2-fast-code

end

end
theory *IsaSAT-Setup-LLVM*
imports *IsaSAT-Setup IsaSAT-Watch-List-LLVM IsaSAT-Lookup-Conflict-LLVM*
More-Sepref.WB-More-Refinement IsaSAT-Clauses-LLVM LBD-LLVM
begin

no-notation *WB-More-Refinement.fref* (‹$[-]_f$ - → -› [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (‹- $\rightarrow_f$ -› [*60*,*60*] *60*)

abbreviation ‹*word32-rel* ≡ *word-rel* :: (*32 word* × -) *set*›
abbreviation ‹*word64-rel* ≡ *word-rel* :: (*64 word* × -) *set*›
abbreviation ‹*word32-assn* ≡ *word-assn* :: *32 word* ⇒ -›
abbreviation ‹*word64-assn* ≡ *word-assn* :: *64 word* ⇒ -›

abbreviation *ema-rel* :: ‹(*ema*×*ema*) *set*› **where**
‹*ema-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*›

abbreviation *ema-assn* :: ‹*ema* ⇒ *ema* ⇒ *assn*› **where**
‹*ema-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*›

abbreviation *stats-rel* :: ‹(*stats* × *stats*) *set*› **where**
‹*stats-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*
$\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *ema-rel*›

abbreviation *stats-assn* :: ‹*stats* ⇒ *stats* ⇒ *assn*› **where**
‹*stats-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$
word64-assn $\times_a$ *word64-assn* $\times_a$ *ema-assn*›

lemma [*sepref-import-param*]:
‹(*ema-get-value*, *ema-get-value*) ∈ *ema-rel* → *word64-rel*›
‹(*ema-bitshifting*,*ema-bitshifting*) ∈ *word64-rel*›
‹(*ema-reinit*,*ema-reinit*) ∈ *ema-rel* → *ema-rel*›
‹(*ema-init*,*ema-init*) ∈ *word-rel* → *ema-rel*›
⟨*proof*⟩

lemma *ema-bitshifting-inline*[*llvm-inline*]:
‹*ema-bitshifting* = (*0x100000000*::-::*len word*)› ⟨*proof*⟩

lemma *ema-reinit-inline*[*llvm-inline*]:
ema-reinit = (λ(*value*, α, β, *wait*, *period*).
(*value*, α, *0x100000000*::-::*len word*, *0*::- *word*, *0*:: - *word*))
⟨*proof*⟩

lemmas [*llvm-inline*] = *ema-init-def*

sepref-def *ema-update-impl* **is** ‹*uncurry* (*RETURN oo ema-update*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *ema-assn*^{k} $\rightarrow_a$ *ema-assn*›
⟨*proof*⟩

lemma [*sepref-import-param*]:
‹(*incr-propagation*,*incr-propagation*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-conflict*,*incr-conflict*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-decision*,*incr-decision*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-restart*,*incr-restart*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-lrestart*,*incr-lrestart*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-uset*,*incr-uset*) ∈ *stats-rel* → *stats-rel*›
‹(*incr-GC*,*incr-GC*) ∈ *stats-rel* → *stats-rel*›
‹(*add-lbd*,*add-lbd*) ∈ *word32-rel* → *stats-rel* → *stats-rel*›
⟨*proof*⟩

lemmas [*llvm-inline*] =
incr-propagation-def
incr-conflict-def
incr-decision-def
incr-restart-def
incr-lrestart-def
incr-uset-def
incr-GC-def

abbreviation (*input*) ‹*restart-info-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*›

abbreviation (*input*) *restart-info-assn* **where**
‹*restart-info-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*›

lemma *restart-info-params*[*sepref-import-param*]:
(*incr-conflict-count-since-last-restart*,*incr-conflict-count-since-last-restart*) ∈
restart-info-rel → *restart-info-rel*
(*restart-info-update-lvl-avg*,*restart-info-update-lvl-avg*) ∈
word32-rel → *restart-info-rel* → *restart-info-rel*
‹(*restart-info-init*,*restart-info-init*) ∈ *restart-info-rel*›
‹(*restart-info-restart-done*,*restart-info-restart-done*) ∈ *restart-info-rel* → *restart-info-rel*›
⟨*proof*⟩

lemmas [*llvm-inline*] =
incr-conflict-count-since-last-restart-def
restart-info-update-lvl-avg-def
restart-info-init-def
restart-info-restart-done-def

type-synonym *vmtf-node-assn* = ‹(*64 word* × *32 word* × *32 word*)›

definition ‹*vmtf-node1-rel* ≡ { ((*a*,*b*,*c*),(*VMTF-Node a b c*)) | *a b c*. *True*}›
definition ‹*vmtf-node2-assn* ≡ *uint64-nat-assn* $\times_a$ *atom.option-assn* $\times_a$ *atom.option-assn*›

definition ‹*vmtf-node-assn* ≡ *hr-comp vmtf-node2-assn vmtf-node1-rel*›

lemmas [*fcomp-norm-unfold*] = *vmtf-node-assn-def*[*symmetric*]

lemma *vmtf-node-assn-pure*[*safe-constraint-rules*]: ‹*CONSTRAINT is-pure vmtf-node-assn*›
⟨*proof*⟩

lemmas [*sepref-frame-free-rules*] = *mk-free-is-pure*[*OF vmtf-node-assn-pure*[*unfolded CONSTRAINT-def*]]

lemma
vmtf-Node-refine1: ‹(λ*a b c.* (*a*,*b*,*c*), *VMTF-Node*) ∈ *Id* → *Id* → *Id* → *vmtf-node1-rel*›
and *vmtf-stamp-refine1*: ‹(λ(*a*,*b*,*c*). *a*, *stamp*) ∈ *vmtf-node1-rel* → *Id*›
and *vmtf-get-prev-refine1*: ‹(λ(*a*,*b*,*c*). *b*, *get-prev*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*›
and *vmtf-get-next-refine1*: ‹(λ(*a*,*b*,*c*). *c*, *get-next*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*›
⟨*proof*⟩

sepref-def *VMTF-Node-impl* **is** []
‹*uncurry2* (*RETURN ooo* (λ*a b c.* (*a*,*b*,*c*)))›
:: ‹$\mathit{uint64\text{-}nat\text{-}assn}^k *_a (\mathit{atom.option\text{-}assn})^k *_a (\mathit{atom.option\text{-}assn})^k \rightarrow_a \mathit{vmtf\text{-}node2\text{-}assn}$›
⟨*proof*⟩

sepref-def *VMTF-stamp-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *a*)›
:: ‹$\mathit{vmtf\text{-}node2\text{-}assn}^k \rightarrow_a \mathit{uint64\text{-}nat\text{-}assn}$›
⟨*proof*⟩

sepref-def *VMTF-get-prev-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *b*)›
:: ‹$\mathit{vmtf\text{-}node2\text{-}assn}^k \rightarrow_a \mathit{atom.option\text{-}assn}$›
⟨*proof*⟩

sepref-def *VMTF-get-next-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *c*)›
:: ‹$\mathit{vmtf\text{-}node2\text{-}assn}^k \rightarrow_a \mathit{atom.option\text{-}assn}$›
⟨*proof*⟩

lemma *workaround-hrcomp-id-norm*[*fcomp-norm-unfold*]: ‹*hr-comp R* (⟨*nat-rel*⟩*option-rel*) = *R*› ⟨*proof*⟩

lemmas [*sepref-fr-rules*] =
VMTF-Node-impl.refine[*FCOMP vmtf-Node-refine1*]
VMTF-stamp-impl.refine[*FCOMP vmtf-stamp-refine1*]
VMTF-get-prev-impl.refine[*FCOMP vmtf-get-prev-refine1*]
VMTF-get-next-impl.refine[*FCOMP vmtf-get-next-refine1*]

type-synonym *vmtf-assn* = ‹*vmtf-node-assn ptr* × *64 word* × *32 word* × *32 word* × *32 word*›

type-synonym *vmtf-remove-assn* = ‹*vmtf-assn* × (*32 word array-list64* × *1 word ptr*)›

abbreviation *vmtf-assn* :: ‹*-* ⇒ *vmtf-assn* ⇒ *assn*› **where**
‹*vmtf-assn* ≡ (*array-assn vmtf-node-assn* $\times_a$ *uint64-nat-assn* $\times_a$ *atom-assn* $\times_a$ *atom-assn* $\times_a$ *atom.option-assn*)›

abbreviation *atoms-hash-assn* :: ‹*bool list* ⇒ *1 word ptr* ⇒ *assn*› **where**
‹*atoms-hash-assn* ≡ *array-assn bool1-assn*›

abbreviation *distinct-atoms-assn* **where**
‹*distinct-atoms-assn* ≡ *arl64-assn atom-assn* $\times_a$ *atoms-hash-assn*›

definition *vmtf-remove-assn*
:: ‹*isa-vmtf-remove-int* ⇒ *vmtf-remove-assn* ⇒ *assn*›
where
‹*vmtf-remove-assn* ≡ *vmtf-assn* $\times_a$ *distinct-atoms-assn*›

Options

type-synonym *opts-assn* = ‹*1 word* × *1 word* × *1 word*›

definition *opts-assn*
:: ‹*opts* ⇒ *opts-assn* ⇒ *assn*›
where
‹*opts-assn* ≡ *bool1-assn* $\times_a$ *bool1-assn* $\times_a$ *bool1-assn*›

lemma *workaround-opt-assn*: ‹*RETURN o* (λ(*a*,*b*,*c*). *f a b c*) = (λ(*a*,*b*,*c*). *RETURN* (*f a b c*))› ⟨*proof*⟩

sepref-register *opts-restart opts-reduce opts-unbounded-mode*

sepref-def *opts-restart-impl* **is** ‹*RETURN o opts-restart*› :: ‹$opts\text{-}assn^k \rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *opts-reduce-impl* **is** ‹*RETURN o opts-reduce*› :: ‹$opts\text{-}assn^k \rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *opts-unbounded-mode-impl* **is** ‹*RETURN o opts-unbounded-mode*› :: ‹$opts\text{-}assn^k \rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

abbreviation ‹*watchlist-fast-assn* ≡ *aal-assn′ TYPE*(*64*) *TYPE*(*64*) *watcher-fast-assn*›

type-synonym *vdom-fast-assn* = ‹*64 word array-list64*›
abbreviation *vdom-fast-assn* :: ‹*vdom* ⇒ *vdom-fast-assn* ⇒ *assn*› **where**
‹*vdom-fast-assn* ≡ *arl64-assn sint64-nat-assn*›

type-synonym *phase-saver-assn* = ‹*1 word larray64*›
abbreviation *phase-saver-assn* :: ‹*phase-saver* ⇒ *phase-saver-assn* ⇒ *assn*› **where**
‹*phase-saver-assn* ≡ *larray64-assn bool1-assn*›

type-synonym *phase-saver′-assn* = ‹*1 word ptr*›

abbreviation *phase-saver′-assn* :: ‹*phase-saver* ⇒ *phase-saver′-assn* ⇒ *assn*› **where**
‹*phase-saver′-assn* ≡ *array-assn bool1-assn*›

type-synonym *arena-assn* = ‹(*32 word*, *64*) *array-list*›
type-synonym *heur-assn* = ‹(*ema* × *ema* × *restart-info* × *64 word* × *phase-saver-assn* × *64 word* × *phase-saver′-assn* × *64 word* × *phase-saver′-assn* × *64 word* × *64 word* × *64 word*)›

type-synonym *twl-st-wll-trail-fast* =
⟨*trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × *vmtf-remove-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *out-learned-assn* × *stats* ×
heur-assn ×
vdom-fast-assn × *vdom-fast-assn* × *64 word* × *opts-assn* × *arena-assn*⟩

abbreviation *phase-heur-assn* **where**
⟨*phase-heur-assn* ≡ *phase-saver-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *phase-saver'-assn* $\times_a$ *sint64-nat-assn* $\times_a$
phase-saver'-assn $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*⟩

definition *heuristic-assn* :: ⟨*restart-heuristics* ⇒ *heur-assn* ⇒ *assn*⟩ **where**
⟨*heuristic-assn* = *ema-assn* $\times_a$
ema-assn $\times_a$
restart-info-assn $\times_a$
word64-assn $\times_a$ *phase-heur-assn*⟩

definition *isasat-bounded-assn* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wll-trail-fast* ⇒ *assn*⟩ **where**
⟨*isasat-bounded-assn* =
trail-pol-fast-assn $\times_a$ *arena-fast-assn* $\times_a$
conflict-option-rel-assn $\times_a$
sint64-nat-assn $\times_a$
watchlist-fast-assn $\times_a$
vmtf-remove-assn $\times_a$
uint32-nat-assn $\times_a$
cach-refinement-l-assn $\times_a$
lbd-assn $\times_a$
out-learned-assn $\times_a$
stats-assn $\times_a$
heuristic-assn $\times_a$
vdom-fast-assn $\times_a$
vdom-fast-assn $\times_a$
uint64-nat-assn $\times_a$
opts-assn $\times_a$ *arena-fast-assn*⟩

sepref-register *NORMAL-PHASE QUIET-PHASE DEFAULT-INIT-PHASE*

sepref-def *NORMAL-PHASE-impl*
is ⟨*uncurry0* (*RETURN NORMAL-PHASE*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a$ *word-assn*⟩
⟨*proof*⟩

sepref-def *QUIET-PHASE-impl*
is ⟨*uncurry0* (*RETURN QUIET-PHASE*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a$ *word-assn*⟩
⟨*proof*⟩

Lift Operations to State

sepref-def *get-conflict-wl-is-None-fast-code*
is ⟨*RETURN o get-conflict-wl-is-None-heur*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a$ *bool1-assn*⟩
⟨*proof*⟩

sepref-def *isa-count-decided-st-fast-code*
is ‹*RETURN o isa-count-decided-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *polarity-pol-fast*
is ‹*uncurry* (*mop-polarity-pol*)›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
⟨*proof*⟩

sepref-def *polarity-st-heur-pol-fast*
is ‹*uncurry* (*mop-polarity-st-heur*)›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*›
⟨*proof*⟩

8.14.1 More theorems

lemma *count-decided-st-heur-alt-def*:
‹*count-decided-st-heur* = (λ(*M*, -). *count-decided-pol M*)›
⟨*proof*⟩

sepref-def *count-decided-st-heur-pol-fast*
is ‹*RETURN o count-decided-st-heur*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *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*) ∧
length (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$
isasat-bounded-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-register ‹(=) :: *clause-status* ⇒ *clause-status* ⇒ -›

lemma [*def-pat-rules*]: ‹*append-ll* ≡ *op-list-list-push-back*›
⟨*proof*⟩

sepref-register *rewatch-heur mop-append-ll mop-arena-length*

sepref-def *mop-append-ll-impl*
is ‹*uncurry2 mop-append-ll*›
:: ‹[λ((*W*, *i*), -). *length* (*W* ! (*nat-of-lit i*)) < *sint64-max*]$_a$
watchlist-fast-assn^d $*_a$ *unat-lit-assn*^{k} $*_a$ *watcher-fast-assn*^{k} → *watchlist-fast-assn*›
⟨*proof*⟩

sepref-def *rewatch-heur-fast-code*
is ‹*uncurry2* (*rewatch-heur*)›
:: ‹[λ((*vdom*, *arena*), *W*). (∀ *x* ∈ *set vdom*. *x* ≤ *sint64-max*) ∧ *length arena* ≤ *sint64-max* ∧
length vdom ≤ *sint64-max*]$_a$
vdom-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *watchlist-fast-assn*^{d} → *watchlist-fast-assn*›

⟨*proof*⟩

sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
⟨*proof*⟩

sepref-register *length-avdom*

sepref-def *length-avdom-fast-code*
is ‹*RETURN o length-avdom*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-register *get-the-propagation-reason-heur*

sepref-def *get-the-propagation-reason-heur-fast-code*
is ‹*uncurry get-the-propagation-reason-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *snat-option-assn' TYPE(64)*›
⟨*proof*⟩

sepref-def *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-bounded-assn^k $*_a$ *sint64-nat-assn*^{k} → *bool1-assn*›
⟨*proof*⟩

sepref-register *clause-lbd-heur*

lemma *clause-lbd-heur-alt-def*:
‹*clause-lbd-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
lcount) *C*.
arena-lbd N' C)›
⟨*proof*⟩

sepref-def *clause-lbd-heur-code2*
is ‹*uncurry* (*RETURN oo clause-lbd-heur*)›
:: ‹[λ(*S*, *C*). *get-clause-LBD-pre* (*get-clauses-wl-heur S*) *C*]$_a$
isasat-bounded-assn^k $*_a$ *sint64-nat-assn*^{k} → *uint32-nat-assn*›
⟨*proof*⟩

sepref-register *mark-garbage-heur*

sepref-def *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*]$_a$
sint64-nat-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›

⟨*proof*⟩

sepref-register *delete-index-vdom-heur*

sepref-def *delete-index-vdom-heur-fast-code2*
is ‹*uncurry* (*RETURN oo delete-index-vdom-heur*)›
:: ‹$[\lambda(i, S).\ i < \textit{length-avdom}\ S]_a$
$\textit{sint64-nat-assn}^k *_a \textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-bounded-assn}$›
⟨*proof*⟩

sepref-register *access-length-heur*

sepref-def *access-length-heur-fast-code2*
is ‹*uncurry* (*RETURN oo access-length-heur*)›
:: ‹$[\lambda(S, C).\ \textit{arena-is-valid-clause-idx}\ (\textit{get-clauses-wl-heur}\ S)\ C]_a$
$\textit{isasat-bounded-assn}^k *_a \textit{sint64-nat-assn}^k \rightarrow \textit{sint64-nat-assn}$›
⟨*proof*⟩

sepref-register *marked-as-used-st*

sepref-def *marked-as-used-st-fast-code*
is ‹*uncurry* (*RETURN oo marked-as-used-st*)›
:: ‹$[\lambda(S, C).\ \textit{marked-as-used-pre}\ (\textit{get-clauses-wl-heur}\ S)\ C]_a$
$\textit{isasat-bounded-assn}^k *_a \textit{sint64-nat-assn}^k \rightarrow \textit{unat-assn}'\ \textit{TYPE}(2)$›
⟨*proof*⟩

sepref-register *mark-unused-st-heur*
sepref-def *mark-unused-st-fast-code*
is ‹*uncurry* (*RETURN oo mark-unused-st-heur*)›
:: ‹$[\lambda(C, S).\ \textit{arena-act-pre}\ (\textit{get-clauses-wl-heur}\ S)\ C]_a$
$\textit{sint64-nat-assn}^k *_a \textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-bounded-assn}$›
⟨*proof*⟩

sepref-def *get-slow-ema-heur-fast-code*
is ‹*RETURN o get-slow-ema-heur*›
:: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a \textit{ema-assn}$›
⟨*proof*⟩

sepref-def *get-fast-ema-heur-fast-code*
is ‹*RETURN o get-fast-ema-heur*›
:: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a \textit{ema-assn}$›
⟨*proof*⟩

sepref-def *get-conflict-count-since-last-restart-heur-fast-code*
is ‹*RETURN o get-conflict-count-since-last-restart-heur*›
:: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a \textit{word64-assn}$›
⟨*proof*⟩

sepref-def *get-learned-count-fast-code*
is ‹*RETURN o get-learned-count*›
:: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a \textit{uint64-nat-assn}$›
⟨*proof*⟩

sepref-register *incr-restart-stat*

sepref-def *incr-restart-stat-fast-code*
is ‹*incr-restart-stat*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-register *incr-lrestart-stat*

sepref-def *incr-lrestart-stat-fast-code*
is ‹*incr-lrestart-stat*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-def *opts-restart-st-fast-code*
is ‹*RETURN o opts-restart-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

sepref-def *opts-reduction-st-fast-code*
is ‹*RETURN o opts-reduction-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

sepref-register *opts-reduction-st opts-restart-st*

lemma *emag-get-value-alt-def*:
‹*ema-get-value* = (λ(*a*, *b*, *c*, *d*). *a*)›
⟨*proof*⟩

sepref-def *ema-get-value-impl*
is ‹*RETURN o ema-get-value*›
:: ‹$ema\text{-}assn^k \rightarrow_a word\text{-}assn$›
⟨*proof*⟩

definition *ema-extract-value-coeff* :: ‹*nat*› **where**
[*simp*]: ‹*ema-extract-value-coeff* = *32*›

sepref-register *ema-extract-value-coeff*

lemma *ema-extract-value-32*[*sepref-fr-rules*]:
‹(*uncurry0* (*return* (*32* :: *64 word*)), *uncurry0* (*RETURN ema-extract-value-coeff*)) ∈ $unit\text{-}assn^k \rightarrow_a$ *unat-assn*›
⟨*proof*⟩

lemmas [*llvm-inline*] = *ema-extract-value-coeff-def*

lemma *emag-extract-value-alt-def*:
‹*ema-extract-value* = (λ(*a*, *b*, *c*, *d*). *a* >> *ema-extract-value-coeff*)›
⟨*proof*⟩

sepref-def *ema-extract-value-impl*
is ‹*RETURN o ema-extract-value*›

:: ‹*ema-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-register *isasat-length-trail-st*

sepref-def *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} $\rightarrow$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *mop-isasat-length-trail-st-code*
is ‹*mop-isasat-length-trail-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-register *get-pos-of-level-in-trail-imp-st*

sepref-def *get-pos-of-level-in-trail-imp-st-code*
is ‹*uncurry get-pos-of-level-in-trail-imp-st*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-register *neq* : ‹(*op-neq* :: *clause-status* ⇒ - ⇒ -)›
lemma *status-neq-refine1*: ‹((≠),*op-neq*) ∈ *status-rel* → *status-rel* → *bool-rel*›
⟨*proof*⟩

sepref-def *status-neq-impl* **is** [] ‹*uncurry* (*RETURN oo* (≠))›
:: ‹(*unat-assn′ TYPE(32)*)^k $*_a$ (*unat-assn′ TYPE(32)*)^k $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *status-neq-impl.refine*[*FCOMP status-neq-refine1*]

lemma *clause-not-marked-to-delete-heur-alt-def*:
‹*RETURN oo clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*.
RETURN (*arena-status arena C* ≠ *DELETED*))›
⟨*proof*⟩

sepref-def *clause-not-marked-to-delete-heur-fast-code*
is ‹*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*)›
:: ‹[*clause-not-marked-to-delete-heur-pre*]$_a$ *isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} → *bool1-assn*›
⟨*proof*⟩

lemma *mop-clause-not-marked-to-delete-heur-alt-def*:
‹*mop-clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*. *do* {
ASSERT(*clause-not-marked-to-delete-heur-pre* ((*M*, *arena*, *D*, *oth*), *C*));
RETURN (*arena-status arena C* ≠ *DELETED*)
})›
⟨*proof*⟩

sepref-def *mop-clause-not-marked-to-delete-heur-impl*
is ‹*uncurry mop-clause-not-marked-to-delete-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *delete-index-and-swap-code2*
  is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
  :: ‹$[\lambda(xs,\ i).\ i < length\ xs]_a$
    vdom-fast-assn^d $*_a$ *sint64-nat-assn*^{k} → *vdom-fast-assn*›
  ⟨*proof*⟩

sepref-def *mop-mark-garbage-heur-impl*
  is ‹*uncurry2 mop-mark-garbage-heur*›
  :: ‹$[\lambda((C,\ i),\ S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
    sint64-nat-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  ⟨*proof*⟩

sepref-def *mop-mark-unused-st-heur-impl*
  is ‹*uncurry mop-mark-unused-st-heur*›
  :: ‹ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

sepref-def *mop-arena-lbd-st-impl*
  is ‹*uncurry mop-arena-lbd-st*›
  :: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

sepref-def *mop-arena-status-st-impl*
  is ‹*uncurry mop-arena-status-st*›
  :: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *status-impl-assn*›
  ⟨*proof*⟩

sepref-def *mop-marked-as-used-st-impl*
  is ‹*uncurry mop-marked-as-used-st*›
  :: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-assn′ TYPE(2)*›
  ⟨*proof*⟩

sepref-def *mop-arena-length-st-impl*
  is ‹*uncurry mop-arena-length-st*›
  :: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

sepref-register *incr-wasted-st full-arena-length-st wasted-bytes-st*
sepref-def *incr-wasted-st-impl*
  is ‹*uncurry* (*RETURN oo incr-wasted-st*)›
  :: ‹*word64-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

sepref-def *full-arena-length-st-impl*
  is ‹*RETURN o full-arena-length-st*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

sepref-def *wasted-bytes-st-impl*
  is ‹*RETURN o wasted-bytes-st*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *word64-assn*›
  ⟨*proof*⟩

lemma *set-zero-wasted-def*:

‹*set-zero-wasted* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *target*, *best*).
(*fast-ema*, *slow-ema*, *res-info*, *0*, φ, *target*, *best*))›
⟨*proof*⟩

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

lemma *mop-save-phase-heur-alt-def*:
‹*mop-save-phase-heur* = (λ *L b* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *target*, *best*). *do* {
ASSERT(*L* < *length* φ);
RETURN (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ[*L* := *b*], *target*,
best)})›
⟨*proof*⟩

sepref-def *mop-save-phase-heur-impl*
is ‹*uncurry2* (*mop-save-phase-heur*)›
:: ‹*atom-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

lemma *id-unat*[*sepref-fr-rules*]:
‹(*return o id*, *RETURN o unat*) ∈ *word32-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-register *set-zero-wasted mop-save-phase-heur add-lbd*

sepref-def *add-lbd-impl*
is ‹*uncurry* (*RETURN oo add-lbd*)›
:: ‹*word32-assn*^{k} $*_a$ *stats-assn*^{d} $\rightarrow_a$ *stats-assn*›
⟨*proof*⟩

experiment begin

export-llvm
ema-update-impl
VMTF-Node-impl
VMTF-stamp-impl
VMTF-get-prev-impl
VMTF-get-next-impl
opts-restart-impl
opts-reduce-impl
opts-unbounded-mode-impl
get-conflict-wl-is-None-fast-code
isa-count-decided-st-fast-code
polarity-st-heur-pol-fast
count-decided-st-heur-pol-fast
access-lit-in-clauses-heur-fast-code
rewatch-heur-fast-code
rewatch-heur-st-fast-code
set-zero-wasted-impl

end

```
end
theory IsaSAT-Inner-Propagation
  imports IsaSAT-Setup
    IsaSAT-Clauses
begin
```

Chapter 9

Propagation: Inner Loop

declare *all-atms-def*[*symmetric*,*simp*]

9.1 Find replacement

lemma *literals-are-in-*$\mathcal{L}_{in}$*-nth2*:
  fixes *C* :: *nat*
  assumes *dom*: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›
  shows ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* ∝ *C*))›
⟨*proof*⟩

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 ≤ length NU);
  ASSERT(C+b ≤ length NU);
  (x, -) ← WHILE_T^{λ(found, i). True}
    (λ(found, i). found = None ∧ i < C + b)
    (λ(-, i). do {
      ASSERT(i < C + (arena-length NU C));
      ASSERT(i ≥ C);
      ASSERT(i < C + b);
      ASSERT(arena-lit-pre NU i);
      L ← mop-arena-lit NU i;
      ASSERT(polarity-pol-pre M′ L);
      if P L 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* ≤ *length* (*N* ∝ *C*)› **and** ‹*b* ≤ *length* (*N* ∝ *C*)› **and** ‹*a* ≤ *b*› **and**

```
    ‹valid-arena arena N vdom› and ‹C ∈# dom-m N› and eq: ‹a' = a› ‹b' = b›  ‹C' = C› and
    ‹⋀L. L ∈# ℒ_all 𝒜 ⟹ P' L = P L› and
    M'M: ‹(M', M) ∈ trail-pol 𝒜›
  assumes lits: ‹literals-are-in-ℒ_in 𝒜 (mset (N ∝ C))›
  shows
    ‹isa-find-unwatched-between P' M' arena a' b' C' ≤ ⇓ Id (find-in-list-between P a b (N ∝ C))›
⟨proof⟩
```

```
definition isa-find-non-false-literal-between where
  ‹isa-find-non-false-literal-between M arena a b C =
     isa-find-unwatched-between (λL. polarity-pol M L ≠ Some False) M arena a b C›
```

```
definition find-unwatched
  :: ‹(nat literal ⇒ bool) ⇒ (nat, nat literal list × bool) fmap ⇒ nat ⇒ (nat option) nres› where
‹find-unwatched M N C = do {
    ASSERT(C ∈# dom-m N);
    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 (N ∝ C)) (N ∝ C)
    else do {
      pos ← SPEC (λi. i ≤ length (N ∝ C) ∧ i ≥ 2);
      n ← find-in-list-between M pos (length (N ∝ C)) (N ∝ C);
      if n = None then find-in-list-between M 2 pos (N ∝ 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 {
    l ← mop-arena-length arena C;
    b ← RETURN(l ≤ MAX-LENGTH-SHORT-CLAUSE);
    if b then isa-find-unwatched-between P M' arena 2 l C
    else do {
      ASSERT(get-saved-pos-pre arena C);
      pos ← mop-arena-pos arena C;
      n ← isa-find-unwatched-between P M' arena pos l C;
      if n = None then isa-find-unwatched-between P M' arena 2 pos C
      else RETURN n
    }
  }
```

›

lemma *find-unwatched-alt-def*:
```
‹find-unwatched M N C = do {
    ASSERT(C ∈# dom-m N);
    - ← RETURN(length (N ∝ C));
    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 (N ∝ C)) (N ∝ C)
    else do {
      pos ← SPEC (λi. i ≤ length (N ∝ C) ∧ i ≥ 2);
      n ← find-in-list-between M pos (length (N ∝ C)) (N ∝ C);
      if n = None then find-in-list-between M 2 pos (N ∝ C)
      else RETURN n
    }
  }
›
  ⟨proof⟩
```

lemma *isa-find-unwatched-find-unwatched*:
  assumes *valid*: ‹*valid-arena arena N vdom*› **and**
    ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))› **and**
    ge2: ‹*2* ≤ *length* (*N* ∝ *C*)› **and**
    M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
  shows ‹*isa-find-unwatched P M' arena C* ≤ ⇓ *Id* (*find-unwatched P N C*)›
⟨*proof*⟩

definition *isa-find-unwatched-wl-st-heur*
  :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*isa-find-unwatched-wl-st-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ) *i*. *do* {
    isa-find-unwatched (λ*L*. *polarity-pol M L* ≠ *Some False*) *M N i*
  })›

lemma *find-unwatched*:
  assumes *n-d*: ‹*no-dup M*› **and** ‹*length* (*N* ∝ *C*) ≥ *2*› **and** ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))›
  shows ‹*find-unwatched* (λ*L*. *polarity M L* ≠ *Some False*) *N C* ≤ ⇓ *Id* (*find-unwatched-l M N C*)›
⟨*proof*⟩

definition *find-unwatched-wl-st-pre* **where**
  ‹*find-unwatched-wl-st-pre* = (λ(*S*, *i*).
    i ∈# *dom-m* (*get-clauses-wl S*) ∧ *2* ≤ *length* (*get-clauses-wl S* ∝ *i*) ∧
    literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* ∝ *i*))
)›

theorem *find-unwatched-wl-st-heur-find-unwatched-wl-s*:
  ‹(*uncurry isa-find-unwatched-wl-st-heur*, *uncurry find-unwatched-wl-st'*)
    ∈ [*find-unwatched-wl-st-pre*]$_f$
      twl-st-heur $\times_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *isa-save-pos* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
  ‹*isa-save-pos C i* = (λ(*M*, *N*, *oth*). *do* {

```
      ASSERT(arena-is-valid-clause-idx N C);
      if arena-length N C > MAX-LENGTH-SHORT-CLAUSE then do {
        ASSERT(isa-update-pos-pre ((C, i), N));
        RETURN (M, arena-update-pos C i N, oth)
      } else RETURN (M, N, oth)
    })
  ›
```

```
lemma isa-save-pos-is-Id:
  assumes
    ‹(S, T) ∈ twl-st-heur›
    ‹C ∈# dom-m (get-clauses-wl T)› and
    ‹i ≤ length (get-clauses-wl T ∝ C)› and
    ‹i ≥ 2›
  shows ‹isa-save-pos C i S ≤ ⇓ {(S', T'). (S', T') ∈ twl-st-heur ∧ length (get-clauses-wl-heur S') =
length (get-clauses-wl-heur S) ∧
       get-watched-wl-heur S' = get-watched-wl-heur S ∧ get-vdom S' = get-vdom S} (RETURN T)›
⟨proof⟩
```

9.2 Updates

```
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, clvls, cach, lbd, outl, stats, fema, sema). do {
    let n = 0;
    ASSERT(curry5 isa-set-lookup-conflict-aa-pre M N C D n outl);
    (D, clvls, outl) ← isa-set-lookup-conflict-aa M N C D n outl;
    j ← mop-isa-length-trail M;
    RETURN (M, N, D, j, W, vmtf, 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).
      w < length (get-watched-wl-heur S ! nat-of-lit L) )›

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 {
     K' ← mop-arena-lit2' (set (get-vdom (M, N, D, Q, W, vm))) N C f;
     ASSERT(w < length N);
     N' ← mop-arena-swap C i f N;
     ASSERT(nat-of-lit K' < length W);
     ASSERT(length (W ! (nat-of-lit K')) < length N);
     let W = W[nat-of-lit K':= W ! (nat-of-lit K') @ [(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*).
    L = *K*)›

lemma *arena-lit-pre*:
‹*valid-arena NU N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *arena-lit-pre NU* (*C* + *i*)›
⟨*proof*⟩

lemma *all-atms-swap*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *j* < *length* (*N* ∝ *C*) ⟹
all-atms (*N*(*C* ↪ *swap* (*N* ∝ *C*) *i j*)) = *all-atms N*›
⟨*proof*⟩

lemma *mop-arena-swap*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C′*) ∈ *nat-rel*› ‹(*i*, *i′*) ∈ *nat-rel*› ‹(*j*, *j′*) ∈ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* ≤ ⇓{(*N″*, *N′*). *valid-arena N″ N′ vdom* ∧ *N″* = *swap-lits C′ i′ j′ arena*
∧ *N′* = *op-clauses-swap N C′ i′ j′* ∧ *all-atms N′* = *all-atms N*} (*mop-clauses-swap N C′ i′ j′*)›
⟨*proof*⟩

lemma *update-clause-wl-alt-def*:
‹*update-clause-wl* = (λ(*L*::′*v literal*) *C b j w i f* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*). *do* {
  ASSERT(*C* ∈# *dom-m N* ∧ *j* ≤ *w* ∧ *w* < *length* (*W L*) ∧ *correct-watching-except* (*Suc j*) (*Suc w*) *L* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
  ASSERT(*L* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
  K′ ← *mop-clauses-at N C f*;
  ASSERT(*K′* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) ∧ *L* ≠ *K′*);
  N′ ← *mop-clauses-swap N C i f*;
  RETURN (*j*, *w*+*1*, (*M*, *N′*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*(*K′* := *W K′* @ [(*C*, *L*, *b*)])))
})›
⟨*proof*⟩

lemma *update-clause-wl-heur-update-clause-wl*:
‹(*uncurry7 update-clause-wl-heur*, *uncurry7* (*update-clause-wl*)) ∈
[*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″ 𝒟 r s K* →
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up″ 𝒟 r s K*⟩*nres-rel*›
⟨*proof*⟩

definition *propagate-lit-wl-heur-pre* **where**
‹*propagate-lit-wl-heur-pre* =
  (λ((*L*, *C*), *S*). *C* ≠ *DECISION-REASON*)›

definition *propagate-lit-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-heur* = (λ*L′ C i* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
  heur, *sema*). *do* {
    ASSERT(*i* ≤ *1*);
    M ← *cons-trail-Propagated-tr L′ C M*;
    N′ ← *mop-arena-swap C 0* (*1* − *i*) *N*;

let stats = *incr-propagation* (*if count-decided-pol M* = *0 then incr-uset stats else stats*);
heur ← *mop-save-phase-heur* (*atm-of* L') (*is-pos* L') *heur*;
RETURN (*M*, N', *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *sema*)
})›

definition *propagate-lit-wl-pre* **where**
‹*propagate-lit-wl-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧
C ∈# *dom-m* (*get-clauses-wl S*) ∧ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
1 − *i* < *length* (*get-clauses-wl S* ∝ *C*) ∧
0 < *length* (*get-clauses-wl S* ∝ *C*))›

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

lemma *propagate-lit-wl-heur-propagate-lit-wl*:
‹(*uncurry3 propagate-lit-wl-heur*, *uncurry3* (*propagate-lit-wl*)) ∈
[λ-. *True*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩*nres-rel*›
⟨*proof*⟩

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* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-bin-heur* = (λL' *C* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *sema*). *do* {
M ← *cons-trail-Propagated-tr* L' *C M*;
let stats = *incr-propagation* (*if count-decided-pol M* = *0 then incr-uset stats else stats*);
heur ← *mop-save-phase-heur* (*atm-of* L') (*is-pos* L') *heur*;
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *sema*)
})›

lemma *propagate-lit-wl-bin-heur-propagate-lit-wl-bin*:
‹(*uncurry2 propagate-lit-wl-bin-heur*, *uncurry2* (*propagate-lit-wl-bin*)) ∈
[λ-. *True*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩*nres-rel*›
⟨*proof*⟩

definition *unit-prop-body-wl-heur-inv* **where**
‹*unit-prop-body-wl-heur-inv S j w L* ⟷
(∃ S'. (*S*, S') ∈ *twl-st-heur* ∧ *unit-prop-body-wl-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* ⟷
(∃ S'. (*S*, S') ∈ *twl-st-heur* ∧ *unit-prop-body-wl-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*:= (*C*, *K*, *b*)]], *vm*))
  })›

definition *pos-of-watched-heur* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*pos-of-watched-heur S C L* = *do* {
  L′ ← *mop-access-lit-in-clauses-heur S C 0*;
  RETURN (*if L* = *L*′ *then 0 else 1*)
} ›

lemma *pos-of-watched-alt*:
‹*pos-of-watched N C L* = *do* {
    ASSERT(*length* (*N* ∝ *C*) > *0* ∧ *C* ∈# *dom-m N*);
    let L′ = (*N* ∝ *C*) ! *0*;
    RETURN (*if L*′ = *L then 0 else 1*)
  }›
⟨*proof*⟩

lemma *pos-of-watched-heur*:
‹(*S*, *S*′) ∈ {(*T*, *T*′). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T*′) ∈ *twl-st-heur-up*′′ 𝒟 *r s t*} ⟹
  ((*C*, *L*), (*C*′, *L*′)) ∈ *Id* $\times_r$ *Id* ⟹
  pos-of-watched-heur S C L ≤ ⇓ *nat-rel* (*pos-of-watched* (*get-clauses-wl S*′) *C*′ *L*′)›
⟨*proof*⟩

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-inv L* (*j*, *w*, *S*) ∧
    length (*watched-by S L*) ≤ *length* (*get-clauses-wl-heur S*′) − *MIN-HEADER-SIZE*)›

definition *other-watched-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*›
where
‹*other-watched-wl-heur S L C i* = *do* {
    ASSERT(*i* < *2* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C i* ∧
      arena-lit (*get-clauses-wl-heur S*) (*C* + *i*) = *L* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C* (*1* − *i*));
    mop-access-lit-in-clauses-heur S C (*1* − *i*)
  }›

lemma *other-watched-heur*:

```
‹(S, S′) ∈ {(T, T′). get-vdom T = get-vdom x2e ∧ (T, T′) ∈ twl-st-heur-up′′ 𝒟 r s t} ⟹
 ((L, C, i), (L′, C′, i′)) ∈ Id ×_r Id ⟹
 other-watched-wl-heur S L C i ≤ ⇓ Id (other-watched-wl S′ L′ C′ i′)›
 ⟨proof⟩
```

9.3 Full inner loop

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));
     (C, K, b) ← mop-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));
     val-K ← mop-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 {
            S ← set-conflict-wl-heur C S;
            RETURN (j+1, w+1, S)}
          else do {
            S ← propagate-lit-wl-bin-heur K C 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 {
    i ← pos-of-watched-heur S C L;
           ASSERT(i ≤ 1);
    L′ ← other-watched-wl-heur S L C i;
    val-L′ ← mop-polarity-st-heur S L′;
    if val-L′ = Some True
    then update-blit-wl-heur L C b j w L′ S
    else do {
      f ← isa-find-unwatched-wl-st-heur S C;
      case f of
 None ⇒ do {
   if val-L′ = Some False
   then do {
     S ← set-conflict-wl-heur C S;
     RETURN (j+1, w+1, S)}
   else do {
     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));
```

```
    K ← mop-access-lit-in-clauses-heur S C f;
    val-L′ ← mop-polarity-st-heur S K;
    if val-L′ = Some True
    then update-blit-wl-heur L C b j w K S
    else do {
      update-clause-wl-heur L C b j w i f S
    }
         }
     }
          }
        }
     }
   }›
```

declare *RETURN-as-SPEC-refine*[*refine2 del*]

```
definition set-conflict-wl′-pre where
  ‹set-conflict-wl′-pre i S ⟷
    get-conflict-wl S = None ∧ i ∈# dom-m (get-clauses-wl S) ∧
    literals-are-in-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-ℒ_in-trail (all-atms-st S) (get-trail-wl S)›
```

```
lemma literals-are-in-ℒ_in-mm-clauses[simp]: ‹literals-are-in-ℒ_in-mm (all-atms-st S) (mset ‘# ran-mf
(get-clauses-wl S))›
   ‹literals-are-in-ℒ_in-mm (all-atms-st S) ((λx. mset (fst x)) ‘# ran-m (get-clauses-wl S))›
  ⟨proof⟩
```

```
lemma set-conflict-wl-alt-def:
  ‹set-conflict-wl = (λC (M, N, D, NE, UE, NS, US, Q, W). do {
     ASSERT(set-conflict-wl-pre C (M, N, D, NE, UE, NS, US, Q, W));
     let D = Some (mset (N ∝ C));
     j ← RETURN (length M);
     RETURN (M, N, D, NE, UE, NS, US, {#}, W)
    })›
  ⟨proof⟩
```

```
lemma set-conflict-wl-pre-set-conflict-wl′-pre:
  assumes ‹set-conflict-wl-pre C S›
  shows ‹set-conflict-wl′-pre C S›
⟨proof⟩
```

```
lemma set-conflict-wl-heur-set-conflict-wl′:
  ‹(uncurry set-conflict-wl-heur, uncurry (set-conflict-wl)) ∈
    [λ-. True]_f
    nat-rel ×_r twl-st-heur-up″ 𝒟 r s K → ⟨twl-st-heur-up″ 𝒟 r s K⟩nres-rel›
⟨proof⟩
```

```
lemma in-Id-in-Id-option-rel[refine]:
  ‹(f, f′) ∈ Id ⟹ (f, f′) ∈ ⟨Id⟩ option-rel›
  ⟨proof⟩
```

The assumption that that accessed clause is active has not been checked at this point!

definition *keep-watch-heur-pre* **where**

‹*keep-watch-heur-pre* =
  (λ(((*L*, *j*), *w*), *S*).
    L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›

lemma *vdom-m-update-subset'*:
‹*fst C* ∈ *vdom-m* $\mathcal{A}$ *bh N* ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *C*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
⟨*proof*⟩

lemma *vdom-m-update-subset*:
‹*bg* < *length* (*bh ap*) ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *bh ap* ! *bg*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
⟨*proof*⟩

lemma *keep-watch-heur-keep-watch*:
‹(*uncurry3 keep-watch-heur*, *uncurry3* (*mop-keep-watch*)) ∈
  [λ-. *True*]$_f$
  Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩ *nres-rel*›
⟨*proof*⟩

This is a slightly stronger version of the previous lemma:

lemma *keep-watch-heur-keep-watch'*:
‹((((*L'*, *j'*), *w'*), *S'*), ((*L*, *j*), *w*), *S*)
    ∈ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* ⟹
keep-watch-heur L' j' w' S' ≤ ⇓ {(*T*, *T'*). *get-vdom T* = *get-vdom S'* ∧
  (*T*, *T'*) ∈ *twl-st-heur-up''* $\mathcal{D}$ *r s K*}
  (*mop-keep-watch L j w S*)›
⟨*proof*⟩

definition *update-blit-wl-heur-pre* **where**
‹*update-blit-wl-heur-pre r K'* = (λ((((((*L*, *C*), *b*), *j*), *w*), *K*), *S*). *L* = *K'*)›

lemma *update-blit-wl-heur-update-blit-wl*:
‹(*uncurry6 update-blit-wl-heur*, *uncurry6 update-blit-wl*) ∈
  [*update-blit-wl-heur-pre r K*]$_f$
  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*→
  ⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩ *nres-rel*›
⟨*proof*⟩

lemma *mop-access-lit-in-clauses-heur*:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ (*i*, *i'*) ∈ *Id* ⟹ (*j*, *j'*) ∈ *Id* ⟹ *mop-access-lit-in-clauses-heur S i j*
  ≤ ⇓ *Id*
    (*mop-clauses-at* (*get-clauses-wl T*) *i' j'*)›
⟨*proof*⟩

lemma *isa-find-unwatched-wl-st-heur-find-unwatched-wl-st*:
  ‹*isa-find-unwatched-wl-st-heur x' y'*
    ≤ ⇓ *Id* (*find-unwatched-l* (*get-trail-wl x*) (*get-clauses-wl x*) *y*)›
  if
    xy: ‹((*x'*, *y'*), *x*, *y*) ∈ *twl-st-heur* $\times_f$ *nat-rel*›
    for *x y x' y'*
⟨*proof*⟩

lemma *unit-propagation-inner-loop-body-wl-alt-def*:
‹*unit-propagation-inner-loop-body-wl L j w S* = *do* {

```
    ASSERT(unit-propagation-inner-loop-wl-loop-pre L (j, w, S));
    (C, K, b) ← mop-watched-by-at S L w;
    S ← mop-keep-watch L j w S;
    ASSERT(is-nondeleted-clause-pre C L S);
    val-K ← mop-polarity-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 do {S ← set-conflict-wl C S;
           RETURN (j+1, w+1, S)}
         else do {
           S ← propagate-lit-wl-bin K C S;
           RETURN (j+1, w+1, S)}
      }  — Now the costly operations:
      else if C ∉# dom-m (get-clauses-wl S)
      then RETURN (j, w+1, S)
      else do {
        ASSERT(unit-prop-body-wl-inv S j w L);
        i ← pos-of-watched (get-clauses-wl S) C L;
        ASSERT(i ≤ 1);
        L′ ← other-watched-wl S L C i;
        val-L′ ← mop-polarity-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-find-unwatched-inv f C S);
          case f of
            None ⇒ do {
              if val-L′ = Some False
              then do {S ← set-conflict-wl C S;
                 RETURN (j+1, w+1, S)}
              else do {S ← propagate-lit-wl L′ C i S; RETURN (j+1, w+1, S)}
            }
          | Some f ⇒ do {
              ASSERT(C ∈# dom-m (get-clauses-wl S) ∧ f < length (get-clauses-wl S ∝ C) ∧ f ≥ 2);
              let S = S; — position saving
              K ← mop-clauses-at (get-clauses-wl S) C f;
              val-L′ ← mop-polarity-wl S K;
              if val-L′ = Some True
              then update-blit-wl L C b j w K S
              else update-clause-wl L C b j w i f S
            }
        }
      }
    }
  })›
  ⟨proof⟩
```

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)
∈ [λ(((*L*, *i*), *j*), *S*). *length* (*watched-by S L*) ≤ *r* − *MIN-HEADER-SIZE* ∧ *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 $\to$
$\langle$*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ r s $K\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

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-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 *mop-length-watched-by-int* :: ‹*twl-st-wl-heur* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat nres*› **where**
‹*mop-length-watched-by-int* S L = *do* {
    ASSERT(*nat-of-lit* L < *length* (*get-watched-wl-heur* S));
    RETURN (*length* (*watched-by-int* S L))
}›

lemma *mop-length-watched-by-int-alt-def*:
‹*mop-length-watched-by-int* = (λ(M, N, D, Q, W, -) L. *do* {
    ASSERT(*nat-of-lit* L < *length* (W));
    RETURN (*length* (W ! *nat-of-lit* L))
})›
$\langle$*proof*$\rangle$

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(*length* (*watched-by-int* S_0 L) $\le$ *length* (*get-clauses-wl-heur* S_0));
    n $\leftarrow$ *mop-length-watched-by-int* S_0 L;
    $\mathit{WHILE}_T{}^{\mathit{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv}\ S_0\ L}$
      (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)
  $\in$ [λ(L, S). *length* (*watched-by* S L) $\le$ r $-$ *MIN-HEADER-SIZE* $\wedge$ L = K $\wedge$ *length* (*watched-by* S L) = s $\wedge$
      length (*watched-by* S L) $\le$ $r]_f$
  nat-lit-lit-rel $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ r s K $\to$
  $\langle$*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ r s $K\rangle$*nres-rel*›
$\langle$*proof*$\rangle$

definition *cut-watch-list-heur*
:: ‹*nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat literal* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*cut-watch-list-heur* j w L =(λ(M, N, D, Q, W, *oth*). *do* {
    ASSERT(j $\le$ *length* (W!*nat-of-lit* L) $\wedge$ j $\le$ w $\wedge$ *nat-of-lit* L < *length* W $\wedge$

```
       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 = (λj 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^{λ(j, w, W). j ≤ w ∧ w ≤ n ∧ nat-of-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⟩
```

```
lemma vdom-m-cut-watch-list:
  ‹set xs ⊆ set (W L) ⟹ vdom-m 𝒜 (W(L := xs)) d ⊆ vdom-m 𝒜 W d›
  ⟨proof⟩
```

The following order allows the rule to be used as a destruction rule, make it more useful for refinement proofs.

```
lemma vdom-m-cut-watch-listD:
  ‹x ∈ vdom-m 𝒜 (W(L := xs)) d ⟹ set xs ⊆ set (W L) ⟹ x ∈ vdom-m 𝒜 W d›
  ⟨proof⟩
```

```
lemma cut-watch-list-heur-cut-watch-list-heur:
  ‹(uncurry3 cut-watch-list-heur, uncurry3 cut-watch-list) ∈
  [λ(((j, w), L), S). True]_f
    nat-rel ×_f nat-rel ×_f nat-lit-lit-rel ×_f twl-st-heur'' 𝒟 r → ⟨twl-st-heur'' 𝒟 r⟩nres-rel›
  ⟨proof⟩
```

```
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) − MIN-HEADER-SIZE);
     cut-watch-list-heur2 j w L 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*) ∈
  [λ(*L*, *S*). *length*(*watched-by S L*) ≤ *r*−*MIN-HEADER-SIZE*]$_f$
  nat-lit-lit-rel $\times_f$ *twl-st-heur*'' 𝒟 *r* → ⟨*twl-st-heur*'' 𝒟 *r*⟩ *nres-rel*›
⟨*proof*⟩

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-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 =
  $\mathit{WHILE}_T{}^{\mathit{unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv}\ S_0}$
    (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* ≠ {#} ⟹
(*x*, *y*) ∈ *twl-st-heur*'' 𝒟*1 r1* ⟹
select-and-remove-from-literals-to-update-wl-heur x
  ≤ ⇓{((*S*, *L*), (*S*′, *L*′)). ((*S*, *L*), (*S*′, *L*′)) ∈ *twl-st-heur*'' 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*)›
⟨*proof*⟩

lemma *outer-loop-length-watched-le-length-arena*:
assumes
  xa-x′: ‹(*xa*, *x*′) ∈ *twl-st-heur*'' 𝒟 *r*› **and**
  prop-heur-inv: ‹*unit-propagation-outer-loop-wl-D-heur-inv x xa*› **and**
  prop-inv: ‹*unit-propagation-outer-loop-wl-inv x*′› **and**
  xb-x′*a*: ‹(*xb*, *x*′*a*) ∈ {((*S*, *L*), (*S*′, *L*′)). ((*S*, *L*), (*S*′, *L*′)) ∈ *twl-st-heur*'' 𝒟*1 r* $\times_f$ *nat-lit-lit-rel* ∧

```
          S' = set-literals-to-update-wl (literals-to-update-wl x' − {#L#}) x' ∧
          get-clauses-wl-heur S = get-clauses-wl-heur xa}› and
    st: ‹x'a = (x1, x2)›
      ‹xb = (x1a, x2a)› and
    x2: ‹x2 ∈# all-lits-st x1› and
    st': ‹(x2, x1) = (x1b, x2b)›
  shows ‹length (watched-by x2b x1b) ≤ r−MIN-HEADER-SIZE›
⟨proof⟩

theorem unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D':
  ‹(unit-propagation-outer-loop-wl-D-heur, unit-propagation-outer-loop-wl) ∈
    twl-st-heur'' 𝒟 r →_f ⟨twl-st-heur'' 𝒟 r⟩ nres-rel›
  ⟨proof⟩

lemma twl-st-heur'D-twl-st-heurD:
  assumes H: ‹(⋀𝒟. f ∈ twl-st-heur' 𝒟 →_f ⟨twl-st-heur' 𝒟⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →_f ⟨twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
⟨proof⟩

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 ∈# ℒ_all (all-atms-st S) ∧ K < length (get-watched-wl S L)]_f
     twl-st-heur ×_f Id ×_f Id → ⟨Id⟩ nres-rel›
  ⟨proof⟩


lemma case-tri-bool-If:
  ‹(case a of
       None ⇒ f1
     | Some v ⇒
        (if v then f2 else f3)) =
   (let b = a in if b = UNSET
    then f1
    else if b = SET-TRUE then f2 else f3)›
  ⟨proof⟩

definition isa-find-unset-lit :: ‹trail-pol ⇒ arena ⇒ nat ⇒ nat ⇒ nat ⇒ nat option nres› where
  ‹isa-find-unset-lit M = isa-find-unwatched-between (λL. polarity-pol M L ≠ Some False) M›

lemma update-clause-wl-heur-pre-le-sint64:
  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)) ≤ sint64-max› and
    ‹arena-lit-pre a1'a (bf + baa)›
  shows ‹bf + baa ≤ sint64-max›
       ‹length a1'a ≤ sint64-max›
  ⟨proof⟩


end
theory IsaSAT-Inner-Propagation-LLVM
  imports IsaSAT-Setup-LLVM
    IsaSAT-Inner-Propagation
begin
```

sepref-register *isa-save-pos*

sepref-def *isa-save-pos-fast-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

lemma [*def-pat-rules*]: ‹*nth-rll* ≡ *op-list-list-idx*›
⟨*proof*⟩

sepref-def *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} $*_a$ *sint64-nat-assn*^{k} → *watcher-fast-assn*›
⟨*proof*⟩

sepref-register *isa-find-unwatched-wl-st-heur isa-find-unwatched-between isa-find-unset-lit polarity-pol*

sepref-register *0 1*

sepref-def *isa-find-unwatched-between-fast-code*
is ‹*uncurry4 isa-find-unset-lit*›
:: ‹[λ((((*M*, *N*), -), -), -). *length N* ≤ *sint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k}
→
snat-option-assn' TYPE(*64*)›
⟨*proof*⟩

sepref-register *mop-arena-pos mop-arena-lit2*
sepref-def *mop-arena-pos-impl*
is ‹*uncurry mop-arena-pos*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *swap-lits-impl* **is** ‹*uncurry3 mop-arena-swap*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

sepref-def *find-unwatched-wl-st-heur-fast-code*
is ‹*uncurry isa-find-unwatched-wl-st-heur*›
:: ‹[(λ(*S*, *C*). *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*)]$_a$
isasat-bounded-assn^k $*_a$ *sint64-nat-assn*^{k} → *snat-option-assn' TYPE*(*64*)›
⟨*proof*⟩

sepref-register *mop-access-lit-in-clauses-heur mop-watched-by-app-heur*
sepref-def *mop-access-lit-in-clauses-heur-impl*
is ‹*uncurry2 mop-access-lit-in-clauses-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*›

⟨*proof*⟩

lemma *other-watched-wl-heur-alt-def*:
⟨*other-watched-wl-heur* = (λS. *arena-other-watched* (*get-clauses-wl-heur* S))⟩
⟨*proof*⟩

lemma *other-watched-wl-heur-alt-def2*:
⟨*other-watched-wl-heur* = (λ(-, N, -). *arena-other-watched* N)⟩
⟨*proof*⟩

sepref-def *other-watched-wl-heur-impl*
is ⟨*uncurry3 other-watched-wl-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*⟩
⟨*proof*⟩

sepref-register *update-clause-wl-heur*
setup ⟨*map-theory-claset* (*fn ctxt* => *ctxt delSWrapper split-all-tac*)⟩

lemma *arena-lit-pre-le2*: ⟨
arena-lit-pre a i $\Longrightarrow$ *length a* $\leq$ *sint64-max* $\Longrightarrow$ *i* < *max-snat 64*⟩
⟨*proof*⟩

lemma *sint64-max-le-max-snat64*: ⟨*a* < *sint64-max* $\Longrightarrow$ *Suc a* < *max-snat 64*⟩
⟨*proof*⟩

sepref-def *update-clause-wl-fast-code*
is ⟨*uncurry7 update-clause-wl-heur*⟩
:: ⟨[λ((((((((L, C), b), j), w), i), f), S). *length* (*get-clauses-wl-heur* S) $\leq$ *sint64-max*]$_a$
unat-lit-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
sint64-nat-assn^k
$*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*⟩
⟨*proof*⟩

sepref-register *mop-arena-swap*

sepref-def *propagate-lit-wl-fast-code*
is ⟨*uncurry3 propagate-lit-wl-heur*⟩
:: ⟨[λ(((L, C), i), S). *length* (*get-clauses-wl-heur* S) $\leq$ *sint64-max*]$_a$
unat-lit-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*⟩
⟨*proof*⟩

sepref-def *propagate-lit-wl-bin-fast-code*
is ⟨*uncurry2 propagate-lit-wl-bin-heur*⟩
:: ⟨[λ((L, C), S). *length* (*get-clauses-wl-heur* S) $\leq$ *sint64-max*]$_a$
unat-lit-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$
isasat-bounded-assn⟩
⟨*proof*⟩

lemma *op-list-list-upd-alt-def*: ⟨*op-list-list-upd xss i j x* = *xss*[*i* := (*xss* ! *i*)[*j* := *x*]]⟩
⟨*proof*⟩

sepref-def *update-blit-wl-heur-fast-code*
is ‹*uncurry6 update-blit-wl-heur*›
:: ‹$[\lambda((((((\text{-}, \text{-}), \text{-}), \text{-}), C), i), S).\ \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S) \le \mathit{sint64\text{-}max}]_a$
$\mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{bool1\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a$
$\mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow$
$\mathit{sint64\text{-}nat\text{-}assn} \times_a \mathit{sint64\text{-}nat\text{-}assn} \times_a \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

sepref-register *keep-watch-heur*

lemma *op-list-list-take-alt-def*: ‹*op-list-list-take xss i l* = *xss*[*i* := *take l* (*xss* ! *i*)]›
⟨*proof*⟩

sepref-def *keep-watch-heur-fast-code*
is ‹*uncurry3 keep-watch-heur*›
:: ‹$\mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

sepref-register *isa-set-lookup-conflict-aa set-conflict-wl-heur*

sepref-def *set-conflict-wl-heur-fast-code*
is ‹*uncurry set-conflict-wl-heur*›
:: ‹$[\lambda(C, S).$
$\mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S) \le \mathit{sint64\text{-}max}]_a$
$\mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

sepref-register *update-blit-wl-heur clause-not-marked-to-delete-heur*
lemma *mop-watched-by-app-heur-alt-def*:
‹*mop-watched-by-app-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, *vmtf*, *φ*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*) *L K*. *do* {
ASSERT(*K* < *length* (*W* ! *nat-of-lit L*));
ASSERT(*nat-of-lit L* < *length* (*W*));
RETURN (*W* ! *nat-of-lit L* ! *K*)})›
⟨*proof*⟩

sepref-def *mop-watched-by-app-heur-code*
is ‹*uncurry2 mop-watched-by-app-heur*›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k *_a \mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k \rightarrow_a \mathit{watcher\text{-}fast\text{-}assn}$›
⟨*proof*⟩

lemma *unit-propagation-inner-loop-wl-loop-D-heur-inv0D*:
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* (*j*, *w*, *S0*) ⟹
j ≤ *length* (*get-clauses-wl-heur S0*) − *MIN-HEADER-SIZE* ∧
w ≤ *length* (*get-clauses-wl-heur S0*) − *MIN-HEADER-SIZE*›
⟨*proof*⟩

sepref-def *pos-of-watched-heur-impl*
is ‹*uncurry2 pos-of-watched-heur*›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{unat\text{-}lit\text{-}assn}^k \rightarrow_a \mathit{sint64\text{-}nat\text{-}assn}$›
⟨*proof*⟩

sepref-def *unit-propagation-inner-loop-body-wl-fast-heur-code*
  is ‹*uncurry3 unit-propagation-inner-loop-body-wl-heur*›
  :: ‹$[\lambda((L,\ w),\ S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
    $unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k\ *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow$
     $sint64\text{-}nat\text{-}assn \times_a sint64\text{-}nat\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
  ⟨*proof*⟩

sepref-register *unit-propagation-inner-loop-body-wl-heur*

lemmas [*llvm-inline*] =
  other-watched-wl-heur-impl-def
  pos-of-watched-heur-impl-def
  propagate-lit-wl-heur-def
  clause-not-marked-to-delete-heur-fast-code-def
  mop-watched-by-app-heur-code-def
  keep-watch-heur-fast-code-def
  nat-of-lit-rel-impl-def

experiment begin

export-llvm
  isa-save-pos-fast-code
  watched-by-app-heur-fast-code
  isa-find-unwatched-between-fast-code
  find-unwatched-wl-st-heur-fast-code
  update-clause-wl-fast-code
  propagate-lit-wl-fast-code
  propagate-lit-wl-bin-fast-code
  status-neq-impl
  clause-not-marked-to-delete-heur-fast-code
  update-blit-wl-heur-fast-code
  keep-watch-heur-fast-code
  set-conflict-wl-heur-fast-code
  unit-propagation-inner-loop-body-wl-fast-heur-code

end

end
theory *IsaSAT-VMTF*
imports *Watched-Literals.WB-Sort IsaSAT-Setup*
begin

Chapter 10

Decision heuristic

10.1 Code generation for the VMTF decision heuristic and the trail

```
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 = (λM L (ns, m, fst-As, lst-As, next-search). do {
  ASSERT(defined-atm-pol-pre M L);
  de ← RETURN (defined-atm-pol M L);
  case fst-As of
    None ⇒RETURN ((ns[L := VMTF-Node m fst-As None], m+1, L, L,
          (if de then None else Some L)))
  | Some fst-As ⇒ do {
      let fst-As' = VMTF-Node (stamp (ns!fst-As)) (Some L) (get-next (ns!fst-As));
      RETURN (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 = (λM L (ns, m, fst-As, lst-As, next-search). do {
    let de = defined-lit M (Pos L);
    case fst-As of
      None ⇒ RETURN (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
       RETURN (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))})›
  ⟨proof⟩
```

```
lemma isa-vmtf-enqueue:
```

‹(*uncurry2 isa-vmtf-enqueue*, *uncurry2* (*RETURN ooo vmtf-enqueue*)) ∈
  [λ((*M*, *L*), -). *L* ∈# $\mathcal{A}$]$_f$ (*trail-pol* $\mathcal{A}$) $\times_f$ *nat-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *partition-vmtf-nth-code-helper3*:
‹∀ *x*∈*set b*. *x* < *length a* ⟹
  x'e < *length a2'* ⟹
  mset a2' = *mset b* ⟹
  a2' ! *x'e* < *length a*›
⟨*proof*⟩

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*)) ∈
  [λ((*M*, *L*), -). *L* ∈# $\mathcal{A}$]$_f$ (*trail-pol* $\mathcal{A}$) $\times_f$ *nat-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma *insert-sort-reorder-list*:
assumes *trans*: ‹⋀ *x y z*. ⟦*R* (*h x*) (*h y*); *R* (*h y*) (*h z*)⟧ ⟹ *R* (*h x*) (*h z*)› **and** *lin*: ‹⋀*x y*. *R* (*h x*) (*h y*) ∨ *R* (*h y*) (*h x*)›
shows ‹(*full-quicksort-ref R h*, *reorder-list vm*) ∈ ⟨*Id*⟩*list-rel* $\rightarrow_f$ ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *quicksort-vmtf-nth-reorder*:
‹(*uncurry quicksort-vmtf-nth*, *uncurry reorder-list*) ∈
Id $\times_r$ ⟨*Id*⟩*list-rel* $\rightarrow_f$ ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) ∈
nat-rel $\times_r$ *atoms-hash-rel* $\mathcal{A}$ $\rightarrow_f$ ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›
⟨*proof*⟩

lemma *vmtf-rescale-alt-def*:

```
‹vmtf-rescale = (λ(ns, m, fst-As, lst-As :: nat, next-search). do {
    (ns, m, -) ← $WHILE_T{}^{\lambda\text{-. True}}$
      (λ(ns, n, lst-As). lst-As ≠None)
      (λ(ns, n, a). do {
         ASSERT(a ≠ None);
         ASSERT(n+1 ≤ uint32-max);
         ASSERT(the a < length ns);
         let m = the a;
         let c = ns ! m;
         let nc = get-next c;
         let pc = get-prev c;
         RETURN (ns[m := VMTF-Node n pc nc], n + 1, pc)
      })
      (ns, 0, Some lst-As);
    RETURN ((ns, m, fst-As, lst-As, next-search))
  })›
```

⟨*proof*⟩

definition *vmtf-reorder-list-raw* **where**

```
‹vmtf-reorder-list-raw = (λvm to-remove. do {
    ASSERT(∀ x∈set to-remove. x < length vm);
    reorder-list vm to-remove
  })›
```

```
definition vmtf-reorder-list where
  ‹vmtf-reorder-list = (λ(vm, -) to-remove. do {
    vmtf-reorder-list-raw vm to-remove
  })›

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' ← vmtf-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 ×_f Id →_f ⟨Id⟩nres-rel›
⟨proof⟩


definition atms-hash-insert-pre :: ‹nat ⇒ nat list × bool list ⇒ bool› where
‹atms-hash-insert-pre i = (λ(n, xs). i < length xs ∧ (¬xs!i ⟶ length n < 2 + uint32-max div 2))›

definition atoms-hash-insert :: ‹nat ⇒ nat list × bool list ⇒ (nat list × bool list)› where
‹atoms-hash-insert i  = (λ(n, xs). if xs ! i then (n, xs) else (n @ [i], xs[i := True]))›

lemma bounded-included-le:
   assumes bounded: ‹isasat-input-bounded 𝒜› and ‹distinct n› and
   ‹set n ⊆ set-mset 𝒜 ›
  shows ‹length n < uint32-max›  ‹length n ≤ 1 + uint32-max div 2›
⟨proof⟩


lemma atms-hash-insert-pre:
  assumes ‹L ∈# 𝒜› and ‹(x, x') ∈ distinct-atoms-rel 𝒜› and ‹isasat-input-bounded 𝒜›
  shows ‹atms-hash-insert-pre L x›
  ⟨proof⟩


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* $\mathcal{A}_{in} \rightarrow \langle$*distinct-atoms-rel* $\mathcal{A}_{in}\rangle$*nres-rel*›
⟨*proof*⟩

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*)) ∈
$[\lambda(L,\ vm).\ L \in\!\#\ \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *Id* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$) →
⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *isa-vmtf-unset* :: ‹*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*› **where**
‹*isa-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*)))›

definition *vmtf-unset-pre* **where**
‹*vmtf-unset-pre* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
L < *length ns* ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length ns*))›

lemma *vmtf-unset-pre-vmtf*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in\!\#\ \mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
⟨*proof*⟩

lemma *vmtf-unset-pre*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in\!\#\ \mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
⟨*proof*⟩

lemma *vmtf-unset-pre*′:
assumes
‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* $\in\!\#\ \mathcal{A}$›
shows ‹*vmtf-unset-pre L vm*›
⟨*proof*⟩

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

definition *isa-vmtf-mark-to-rescore-and-unset-pre* **where**
‹*isa-vmtf-mark-to-rescore-and-unset-pre* = (λ(*L*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*)).
vmtf-unset-pre L ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*) ∧
atms-hash-insert-pre L tor)›

lemma *size-conflict-int-size-conflict*:
‹(*RETURN o size-conflict-int*, *RETURN o size-conflict*) ∈ $[\lambda D.\ D \neq None]_f$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

definition *rescore-clause*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *vmtf-remove-int* ⇒
(*vmtf-remove-int*) *nres*›
where
‹*rescore-clause* $\mathcal{A}$ *C M vm* = *SPEC* (λ(*vm′*). *vm′* ∈ *vmtf* $\mathcal{A}$ *M*)›

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*›
⟨*proof*⟩

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*⟩

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*⟩

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 {
    $WHILE_T{}^{\lambda next\text{-}search.\ next\text{-}search \neq None \longrightarrow defined\text{-}atm\text{-}pol\text{-}pre\ M\ (the\ next\text{-}search)}$
      (λnext-search. next-search ≠ None ∧ defined-atm-pol M (the next-search))
      (λnext-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 𝒜)) ∈
    (Id ×_r distinct-atoms-rel 𝒜) ×_r trail-pol 𝒜 →_f ⟨⟨nat-rel⟩option-rel⟩nres-rel ›
⟨proof⟩
```

10.2 Bumping

definition *vmtf-rescore-body*

```
:: ‹nat multiset ⇒ nat clause-l ⇒ (nat,nat) ann-lits ⇒ vmtf-remove-int ⇒
    (nat × vmtf-remove-int) nres›
where
  ‹vmtf-rescore-body 𝒜_in C - vm = do {
        WHILE_T^{λ(i, vm). i ≤ length C ∧ (∀ c ∈ set C. atm-of c < length (fst (fst vm)))}
          (λ(i, vm). i < length C)
          (λ(i, vm). do {
              ASSERT(i < length C);
              ASSERT(atm-of (C!i) ∈# 𝒜_in);
              let vm' = vmtf-mark-to-rescore (atm-of (C!i)) vm;
              RETURN(i+1, vm')
            })
          (0, vm)
    }›
```

definition *vmtf-rescore*

```
:: ‹nat multiset ⇒ nat clause-l ⇒ (nat,nat) ann-lits ⇒ vmtf-remove-int ⇒
      (vmtf-remove-int) nres›
where
  ‹vmtf-rescore 𝒜_in C M vm = do {
      (-, vm) ← vmtf-rescore-body 𝒜_in C M vm;
      RETURN (vm)
   }›
```

find-theorems *isa-vmtf-mark-to-rescore*

definition *isa-vmtf-rescore-body*

```
:: ‹nat clause-l ⇒ trail-pol ⇒ isa-vmtf-remove-int ⇒
    (nat × isa-vmtf-remove-int) nres›
where
  ‹isa-vmtf-rescore-body C - vm = do {
        WHILE_T^{λ(i, vm). i ≤ length C ∧ (∀ c ∈ set C. atm-of c < length (fst (fst vm)))}
          (λ(i, vm). i < length C)
          (λ(i, vm). 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')
            })
          (0, vm)
    }›
```

definition *isa-vmtf-rescore*

```
:: ‹nat clause-l ⇒ trail-pol ⇒ isa-vmtf-remove-int ⇒
      (isa-vmtf-remove-int) nres›
where
```

```
‹isa-vmtf-rescore C M vm = do {
    (-, vm) ← isa-vmtf-rescore-body C M vm;
    RETURN (vm)
  }›
```

lemma *vmtf-rescore-score-clause*:

```
‹(uncurry2 (vmtf-rescore 𝒜), uncurry2 (rescore-clause 𝒜)) ∈
    [λ((C, M), vm). literals-are-in-ℒ_in 𝒜 (mset C) ∧ vm ∈ vmtf 𝒜 M]_f
    (⟨Id⟩list-rel ×_f Id ×_f Id) → ⟨Id⟩ nres-rel›
```
⟨*proof*⟩

lemma *isa-vmtf-rescore-body*:

```
‹(uncurry2 (isa-vmtf-rescore-body), uncurry2 (vmtf-rescore-body 𝒜)) ∈ [λ-. isasat-input-bounded 𝒜]_f
  (Id ×_f trail-pol 𝒜 ×_f (Id ×_f distinct-atoms-rel 𝒜)) → ⟨Id ×_r (Id ×_f distinct-atoms-rel 𝒜)⟩ nres-rel›
```
⟨*proof*⟩

lemma *isa-vmtf-rescore*:

```
‹(uncurry2 (isa-vmtf-rescore), uncurry2 (vmtf-rescore 𝒜)) ∈ [λ-. isasat-input-bounded 𝒜]_f
    (Id ×_f trail-pol 𝒜 ×_f (Id ×_f distinct-atoms-rel 𝒜)) → ⟨(Id ×_f distinct-atoms-rel 𝒜)⟩ nres-rel›
```
⟨*proof*⟩

definition *vmtf-mark-to-rescore-clause* **where**

```
‹vmtf-mark-to-rescore-clause 𝒜_in arena C vm = do {
    ASSERT(arena-is-valid-clause-idx arena C);
    nfoldli
      ([C..<C + (arena-length arena C)])
      (λ-. True)
      (λi vm. do {
        ASSERT(i < length arena);
        ASSERT(arena-lit-pre arena i);
        ASSERT(atm-of (arena-lit arena i) ∈# 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 + (arena-length arena C)])
      (λ-. True)
      (λi 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 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*›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore-clause-spec*:
‹*vm* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹
 (∀ *C* ∈ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
  vmtf-mark-to-rescore-clause $\mathcal{A}$ *arena C vm* ≤ *RES* (*vmtf* $\mathcal{A}$ *M*)›
⟨*proof*⟩

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)
      (λi 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)
      (λi vm. do {
        ASSERT(i < length outl); ASSERT(length outl≤ uint32-max);
        C ← get-the-propagation-reason-pol M (−(outl ! i));
        case C of
          None ⇒ do {
            ASSERT (isa-vmtf-mark-to-rescore-pre (atm-of (outl ! i)) vm);
            RETURN (isa-vmtf-mark-to-rescore (atm-of (outl ! i)) vm)
   }
        | Some C ⇒ if C = 0 then RETURN vm else isa-vmtf-mark-to-rescore-clause arena C vm
      })
      vm
  }›
```

lemma *isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons*:
‹(*uncurry3 isa-vmtf-mark-to-rescore-also-reasons*, *uncurry3* (*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$)) ∈
  [λ-. *isasat-input-bounded* $\mathcal{A}]_f$
  trail-pol $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) → ⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore*′:

```
‹$L \in$ atms-of $(\mathcal{L}_{all}\ \mathcal{A}) \Longrightarrow vm \in$ vmtf $\mathcal{A}$ $M \Longrightarrow$ vmtf-mark-to-rescore $L$ $vm \in$ vmtf $\mathcal{A}$ $M$›
  ⟨proof⟩
```

lemma *vmtf-mark-to-rescore-also-reasons-spec*:

```
  ‹$vm \in$ vmtf $\mathcal{A}$ $M \Longrightarrow$ valid-arena arena $N$ vdom $\Longrightarrow$ length outl $\leq$ uint32-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-m $N \wedge$
      $(\forall C \in$ set $[C..<C +$ arena-length arena $C]$. arena-lit arena $C \in\# \mathcal{L}_{all}\ \mathcal{A}))))) \Longrightarrow$
   vmtf-mark-to-rescore-also-reasons $\mathcal{A}$ $M$ arena outl $vm \leq$ RES (vmtf $\mathcal{A}$ $M$)›
  ⟨proof⟩
```

10.3 Backtrack level for Restarts

We here find out how many decisions can be reused. Remark that since VMTF does not reuse many levels anyway, the implementation might be mostly useless, but I was not aware of that when I implemented it.

definition *find-decomp-w-ns-pre* **where**

```
  ‹find-decomp-w-ns-pre $\mathcal{A}$ = ($\lambda$((M, highest), vm).
      no-dup $M$ $\wedge$
      highest < count-decided $M$ $\wedge$
      isasat-input-bounded $\mathcal{A}$ $\wedge$
      literals-are-in-$\mathcal{L}_{in}$-trail $\mathcal{A}$ $M$ $\wedge$
      $vm \in$ vmtf $\mathcal{A}$ $M$)›
```

definition *find-decomp-wl-imp*

```
  :: ‹nat multiset ⇒ (nat, nat) ann-lits ⇒ nat ⇒ vmtf-remove-int ⇒
      ((nat, nat) ann-lits × vmtf-remove-int) nres›
```

where

```
  ‹find-decomp-wl-imp $\mathcal{A}$ = ($\lambda M_0$ lev vm. do {
    let k = count-decided $M_0$;
    let $M_0$ = trail-conv-to-no-CS $M_0$;
    let n = length $M_0$;
    pos ← get-pos-of-level-in-trail $M_0$ lev;
    ASSERT((n − pos) ≤ uint32-max);
    ASSERT(n ≥ pos);
    let target = n − pos;
    (-, M, vm') ←
     $WHILE_T^{\lambda(j,\ M,\ vm').\ j \leq target\ \wedge\ M = drop\ j\ M_0\ \wedge\ target \leq length\ M_0\ \wedge\ vm' \in vmtf\ \mathcal{A}\ M\ \wedge\ literals\text{-}ar}$
        ($\lambda$(j, M, vm). j < target)
        ($\lambda$(j, M, vm). do {
           ASSERT(M ≠ []);
           ASSERT(Suc j ≤ uint32-max);
           let L = atm-of (lit-of-hd-trail M);
           ASSERT($L \in\# \mathcal{A}$);
           RETURN (j + 1, tl M, vmtf-unset L vm)
        })
        (0, $M_0$, vm);
    ASSERT(lev = count-decided M);
    let M = trail-conv-back lev M;
    RETURN (M, vm')
  })›
```

definition *isa-find-decomp-wl-imp*

```
  :: ‹trail-pol ⇒ nat ⇒ isa-vmtf-remove-int ⇒ (trail-pol × isa-vmtf-remove-int) nres›
```

```
where
  ‹isa-find-decomp-wl-imp = (λM₀ lev vm. do {
    let k = count-decided-pol M₀;
    let M₀ = trail-pol-conv-to-no-CS M₀;
    ASSERT(isa-length-trail-pre M₀);
    let n = isa-length-trail M₀;
    pos ← get-pos-of-level-in-trail-imp M₀ lev;
    ASSERT((n − pos) ≤ uint32-max);
    ASSERT(n ≥ pos);
    let target = n − pos;
    (-, M, vm') ←
        WHILE_T^λ(j, M, vm'). j ≤ target
          (λ(j, M, vm). j < target)
          (λ(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 + 1, tl-trailt-tr-no-CS M, isa-vmtf-unset L vm)
          })
          (0, M₀, vm);
    M ← trail-conv-back-imp lev M;
    RETURN (M, vm')
  })›
```

```
abbreviation find-decomp-w-ns-prop where
  ‹find-decomp-w-ns-prop 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 𝒜 M1))›
```

```
definition find-decomp-w-ns where
  ‹find-decomp-w-ns 𝒜 =
     (λ(M::(nat, nat) ann-lits) highest vm.
        SPEC(find-decomp-w-ns-prop 𝒜 M highest vm))›
```

```
lemma isa-find-decomp-wl-imp-find-decomp-wl-imp:
  ‹(uncurry2 isa-find-decomp-wl-imp, uncurry2 (find-decomp-wl-imp 𝒜)) ∈
      [λ((M, lev), vm). lev < count-decided M]_f trail-pol 𝒜 ×_f nat-rel ×_f (Id ×_r distinct-atoms-rel 𝒜)
→
      ⟨trail-pol 𝒜 ×_r (Id ×_r distinct-atoms-rel 𝒜)⟩nres-rel›
⟨proof⟩
```

```
definition (in −) find-decomp-wl-st :: ‹nat literal ⇒ nat twl-st-wl ⇒ nat twl-st-wl nres› where
  ‹find-decomp-wl-st = (λL (M, N, D, oth).  do{
     M' ← find-decomp-wl' M (the D) L;
    RETURN (M', N, D, oth)
  })›
```

```
definition find-decomp-wl-st-int :: ‹nat ⇒ twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹find-decomp-wl-st-int = (λhighest (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*)
})›

lemma
assumes
vm: ‹*vm* ∈ *vmtf* $\mathcal{A}$ M_0› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M_0› **and**
target: ‹*highest* < *count-decided* M_0› **and**
n-d: ‹*no-dup* M_0› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
find-decomp-wl-imp-le-find-decomp-wl':
‹*find-decomp-wl-imp* $\mathcal{A}$ M_0 *highest vm* ≤ *find-decomp-w-ns* $\mathcal{A}$ M_0 *highest vm*›
(**is** *?decomp*)
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

end
theory *IsaSAT-Sorting*
imports *IsaSAT-Setup*
begin

Chapter 11

Sorting of clauses

We use the sort function developped by Peter Lammich.

```
definition clause-score-ordering where
  ‹clause-score-ordering = (λ(lbd, act) (lbd', act'). lbd < lbd' ∨ (lbd = lbd' ∧ act < act'))›

definition (in −) clause-score-extract :: ‹arena ⇒ nat ⇒ nat × nat› where
  ‹clause-score-extract arena C = (
     if arena-status arena C = DELETED
     then (uint32-max, 0) — deleted elements are the largest possible
     else
       let lbd = arena-lbd arena C in
       (lbd, C)
  )›

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 clause-score-less :: ‹arena ⇒ nat ⇒ nat ⇒ bool› where
  clause-score-less arena i j ⟷
     clause-score-ordering (clause-score-extract arena i) (clause-score-extract arena j)

definition idx-cdom :: ‹arena ⇒ nat set› where
 ‹idx-cdom arena ≡ {i. valid-sort-clause-score-pre-at arena i}›

definition mop-clause-score-less where
  ‹mop-clause-score-less arena i j = do {
    ASSERT(valid-sort-clause-score-pre-at arena i);
    ASSERT(valid-sort-clause-score-pre-at arena j);
    RETURN (clause-score-ordering (clause-score-extract arena i) (clause-score-extract arena j))
  }›
```

end
theory *IsaSAT-Sorting-LLVM*
  imports *IsaSAT-Sorting IsaSAT-Setup-LLVM*
    Isabelle-LLVM.Sorting-Introsort
begin

no-notation *WB-More-Refinement.fref* (‹[-]$_f$ - → -› [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (‹- →$_f$ -› [*60*,*60*] *60*)
declare *α-butlast*[*simp del*]

locale *pure-eo-adapter* =
  fixes *elem-assn* :: ‹*'a* ⇒ *'ai*::*llvm-rep* ⇒ *assn*›
    and *wo-assn* :: ‹*'a list* ⇒ *'oi*::*llvm-rep* ⇒ *assn*›
    and *wo-get-impl* :: ‹*'oi* ⇒ *'size*::*len2 word* ⇒ *'ai llM*›
    and *wo-set-impl* :: ‹*'oi* ⇒ *'size*::*len2 word* ⇒ *'ai* ⇒ *'oi llM*›
  assumes *pure*[*safe-constraint-rules*]: ‹*is-pure elem-assn*›
    and *get-hnr*: ‹(*uncurry wo-get-impl*,*uncurry mop-list-get*) ∈ *wo-assn*^{k} $*_a$ *snat-assn*^{k} $\rightarrow_a$ *elem-assn*›
    and *set-hnr*: ‹(*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ *wo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{k} $\rightarrow_{ad}$ (λ- ((*ai*,-),-). *cnc-assn* (λ*x*. *x*=*ai*) *wo-assn*)›
begin

  lemmas [*sepref-fr-rules*] = *get-hnr set-hnr*

  definition ‹*only-some-rel* ≡ {(*a*, *Some a*) | *a*. *True*} ∪ {(*x*, *None*) | *x*. *True*}›

  definition ‹*eo-assn* ≡ *hr-comp wo-assn* (⟨*only-some-rel*⟩*list-rel*)›

  definition ‹*eo-extract1 p i* ≡ *doN* { *r* ← *mop-list-get p i*; *RETURN* (*r*,*p*) }›
  sepref-definition *eo-extract-impl* **is** ‹*uncurry eo-extract1*›
    :: ‹*wo-assn*^{d} $*_a$ (*snat-assn' TYPE*(*'size*))^k $\rightarrow_a$ *elem-assn* $\times_a$ *wo-assn*›
    ⟨*proof*⟩

  lemma *mop-eo-extract-aux*: ‹*mop-eo-extract p i* = *doN* { *r* ← *mop-list-get p i*; *ASSERT* (*r*≠*None* ∧ *i*<*length p*); *RETURN* (*the r*, *p*[*i*:=*None*]) }›
    ⟨*proof*⟩

  lemma *assign-none-only-some-list-rel*:
    assumes *SR*[*param*]: ‹(*a*, *a'*) ∈ ⟨*only-some-rel*⟩*list-rel*› **and** *L*: ‹*i* < *length a'*›
      shows ‹(*a*, *a'*[*i* := *None*]) ∈ ⟨*only-some-rel*⟩*list-rel*›
  ⟨*proof*⟩

  lemma *eo-extract1-refine*: ‹(*eo-extract1*, *mop-eo-extract*) ∈ ⟨*only-some-rel*⟩*list-rel* → *nat-rel* → ⟨*Id* $\times_r$ ⟨*only-some-rel*⟩*list-rel*⟩*nres-rel*›
    ⟨*proof*⟩

  lemma *eo-list-set-refine*: ‹(*mop-list-set*, *mop-eo-set*) ∈ ⟨*only-some-rel*⟩*list-rel* → *Id* → *Id* → ⟨⟨*only-some-rel*⟩*list-rel*⟩*nres-*
    ⟨*proof*⟩

  lemma *set-hnr'*: ‹(*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ *wo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{k} $\rightarrow_a$ *wo-assn*›
    ⟨*proof*⟩

context
notes [*fcomp-norm-unfold*] = *eo-assn-def*[*symmetric*]
begin
lemmas *eo-extract-refine-aux* = *eo-extract-impl.refine*[*FCOMP eo-extract1-refine*]

lemma *eo-extract-refine*: (*uncurry eo-extract-impl*, *uncurry mop-eo-extract*) ∈ *eo-assn*^{d} $*_a$ *snat-assn*^{k} $\rightarrow_{ad}$ (λ- (*ai*,-). *elem-assn* $\times_a$ *cnc-assn* (λ*x*. *x*=*ai*) *eo-assn*)
⟨*proof*⟩

lemmas *eo-set-refine-aux* = *set-hnr*′[*FCOMP eo-list-set-refine*]

lemma *pure-part-cnc-imp-eq*: ‹*pure-part* (*cnc-assn* (λ*x*. *x* = *cc*) *wo-assn a c*) ⟹ *c*=*cc*›
⟨*proof*⟩

lemma *pure-entails-empty*: ‹*is-pure A* ⟹ *A a c* ⊢ □›
⟨*proof*⟩

lemma *eo-set-refine*: ‹(*uncurry2 wo-set-impl*, *uncurry2 mop-eo-set*) ∈ *eo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{d} $\rightarrow_{ad}$ (λ- ((*ai*, -), -). *cnc-assn* (λ*x*. *x* = *ai*) *eo-assn*)›
⟨*proof*⟩

end

lemma *id-Some-only-some-rel*: ‹(*id*, *Some*) ∈ *Id* → *only-some-rel*›
⟨*proof*⟩

lemma *map-some-only-some-rel-iff*: ‹(*xs*, *map Some ys*) ∈ ⟨*only-some-rel*⟩*list-rel* ⟷ *xs*=*ys*›
⟨*proof*⟩

lemma *wo-assn-conv*: ‹*wo-assn xs ys* = *eo-assn* (*map Some xs*) *ys*›
⟨*proof*⟩

lemma *to-eo-conv-refine*: ‹(*return*, *mop-to-eo-conv*) ∈ *wo-assn*^{d} $\rightarrow_{ad}$ (λ- *ai*. *cnc-assn* (λ*x*. *x* = *ai*) *eo-assn*)›
⟨*proof*⟩

lemma ‹*None* ∉ *set xs* ⟷ (∃ *ys*. *xs* = *map Some ys*)›
⟨*proof*⟩

lemma *to-wo-conv-refine*: ‹(*return*, *mop-to-wo-conv*) ∈ *eo-assn*^{d} $\rightarrow_{ad}$ (λ- *ai*. *cnc-assn* (λ*x*. *x* = *ai*) *wo-assn*)›
⟨*proof*⟩

lemma *random-access-iterator*: *random-access-iterator wo-assn eo-assn elem-assn*
return return
eo-extract-impl
wo-set-impl
⟨*proof*⟩

sublocale *random-access-iterator wo-assn eo-assn elem-assn*

return return
eo-extract-impl
wo-set-impl
⟨*proof*⟩

end

lemma *al-pure-eo*: ‹*is-pure A* ⟹ *pure-eo-adapter A* (*al-assn A*) *arl-nth arl-upd*›
⟨*proof*⟩

end
theory *IsaSAT-VMTF-LLVM*
imports *Watched-Literals.WB-Sort IsaSAT-VMTF IsaSAT-Setup-LLVM*
Isabelle-LLVM.Sorting-Introsort
IsaSAT-Sorting-LLVM
begin

definition *valid-atoms* :: ‹*nat-vmtf-node list* ⇒ *nat set*› **where**
‹*valid-atoms xs* ≡ {*i*. *i* < *length xs*}›

definition *VMTF-score-less* **where**
‹*VMTF-score-less xs i j* ⟷ *stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*)›

definition *mop-VMTF-score-less* **where**
‹*mop-VMTF-score-less xs i j* = *do* {
ASSERT(*i* < *length xs*);
ASSERT(*j* < *length xs*);
RETURN (*stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*))
}›

sepref-register *VMTF-score-less*

sepref-def (**in** −) *mop-VMTF-score-less-impl*
is ‹*uncurry2* (*mop-VMTF-score-less*)›
:: ‹(*array-assn vmtf-node-assn*)^k $*_a$ *atom-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

interpretation *VMTF*: *weak-ordering-on-lt* **where**
C = ‹*valid-atoms vs*› **and**
less = ‹*VMTF-score-less vs*›
⟨*proof*⟩

interpretation *VMTF*: *parameterized-weak-ordering valid-atoms VMTF-score-less*
mop-VMTF-score-less
⟨*proof*⟩

```
global-interpretation VMTF: parameterized-sort-impl-context
  ‹woarray-assn atom-assn› ‹eoarray-assn atom-assn› atom-assn
  return return
  eo-extract-impl
  array-upd
  valid-atoms VMTF-score-less mop-VMTF-score-less mop-VMTF-score-less-impl
  ‹array-assn vmtf-node-assn›
  defines
          VMTF-is-guarded-insert-impl = VMTF.is-guarded-param-insert-impl
      and VMTF-is-unguarded-insert-impl = VMTF.is-unguarded-param-insert-impl
      and VMTF-unguarded-insertion-sort-impl = VMTF.unguarded-insertion-sort-param-impl
      and VMTF-guarded-insertion-sort-impl = VMTF.guarded-insertion-sort-param-impl
      and VMTF-final-insertion-sort-impl = VMTF.final-insertion-sort-param-impl


      and VMTF-pcmpo-idxs-impl  = VMTF.pcmpo-idxs-impl
      and VMTF-pcmpo-v-idx-impl  = VMTF.pcmpo-v-idx-impl
      and VMTF-pcmpo-idx-v-impl  = VMTF.pcmpo-idx-v-impl
      and VMTF-pcmp-idxs-impl  = VMTF.pcmp-idxs-impl

      and VMTF-mop-geth-impl    = VMTF.mop-geth-impl
      and VMTF-mop-seth-impl    = VMTF.mop-seth-impl
      and VMTF-sift-down-impl   = VMTF.sift-down-impl
      and VMTF-heapify-btu-impl = VMTF.heapify-btu-impl
      and VMTF-heapsort-impl    = VMTF.heapsort-param-impl
      and VMTF-qsp-next-l-impl       = VMTF.qsp-next-l-impl
      and VMTF-qsp-next-h-impl       = VMTF.qsp-next-h-impl
      and VMTF-qs-partition-impl     = VMTF.qs-partition-impl

      and VMTF-partition-pivot-impl  = VMTF.partition-pivot-impl
      and VMTF-introsort-aux-impl = VMTF.introsort-aux-param-impl
      and VMTF-introsort-impl        = VMTF.introsort-param-impl
      and VMTF-move-median-to-first-impl = VMTF.move-median-to-first-param-impl

  ⟨proof⟩



global-interpretation
  VMTF-it: pure-eo-adapter atom-assn ‹arl64-assn atom-assn› arl-nth arl-upd
  defines VMTF-it-eo-extract-impl = VMTF-it.eo-extract-impl
  ⟨proof⟩



global-interpretation VMTF-it: parameterized-sort-impl-context
  where
    wo-assn = ‹arl64-assn atom-assn›
    and eo-assn = VMTF-it.eo-assn
    and elem-assn = atom-assn
    and to-eo-impl = return
    and to-wo-impl = return
    and extract-impl = VMTF-it-eo-extract-impl
    and set-impl = arl-upd
    and cdom = valid-atoms
    and pless = VMTF-score-less
```

and *pcmp* = *mop-VMTF-score-less*
and *pcmp-impl* = *mop-VMTF-score-less-impl*
and *cparam-assn* = ‹*array-assn vmtf-node-assn*›
defines
VMTF-it-is-guarded-insert-impl = *VMTF-it.is-guarded-param-insert-impl*
and *VMTF-it-is-unguarded-insert-impl* = *VMTF-it.is-unguarded-param-insert-impl*
and *VMTF-it-unguarded-insertion-sort-impl* = *VMTF-it.unguarded-insertion-sort-param-impl*
and *VMTF-it-guarded-insertion-sort-impl* = *VMTF-it.guarded-insertion-sort-param-impl*
and *VMTF-it-final-insertion-sort-impl* = *VMTF-it.final-insertion-sort-param-impl*

and *VMTF-it-pcmpo-idxs-impl* = *VMTF-it.pcmpo-idxs-impl*
and *VMTF-it-pcmpo-v-idx-impl* = *VMTF-it.pcmpo-v-idx-impl*
and *VMTF-it-pcmpo-idx-v-impl* = *VMTF-it.pcmpo-idx-v-impl*
and *VMTF-it-pcmp-idxs-impl* = *VMTF-it.pcmp-idxs-impl*

and *VMTF-it-mop-geth-impl* = *VMTF-it.mop-geth-impl*
and *VMTF-it-mop-seth-impl* = *VMTF-it.mop-seth-impl*
and *VMTF-it-sift-down-impl* = *VMTF-it.sift-down-impl*
and *VMTF-it-heapify-btu-impl* = *VMTF-it.heapify-btu-impl*
and *VMTF-it-heapsort-impl* = *VMTF-it.heapsort-param-impl*
and *VMTF-it-qsp-next-l-impl* = *VMTF-it.qsp-next-l-impl*
and *VMTF-it-qsp-next-h-impl* = *VMTF-it.qsp-next-h-impl*
and *VMTF-it-qs-partition-impl* = *VMTF-it.qs-partition-impl*

and *VMTF-it-partition-pivot-impl* = *VMTF-it.partition-pivot-impl*
and *VMTF-it-introsort-aux-impl* = *VMTF-it.introsort-aux-param-impl*
and *VMTF-it-introsort-impl* = *VMTF-it.introsort-param-impl*
and *VMTF-it-move-median-to-first-impl* = *VMTF-it.move-median-to-first-param-impl*

⟨*proof*⟩

lemmas [*llvm-inline*] = *VMTF-it.eo-extract-impl-def*[*THEN meta-fun-cong*, *THEN meta-fun-cong*]

print-named-simpset *llvm-inline*
export-llvm
‹*VMTF-heapsort-impl* :: - ⇒ - ⇒ -›
‹*VMTF-introsort-impl* :: - ⇒ - ⇒ -›

definition *VMTF-sort-scores-raw* :: ‹-› **where**
‹*VMTF-sort-scores-raw* = *pslice-sort-spec valid-atoms VMTF-score-less*›

definition *VMTF-sort-scores* :: ‹-› **where**
‹*VMTF-sort-scores xs ys* = *VMTF-sort-scores-raw xs ys 0* (*length ys*)›

lemmas *VMTF-introsort*[*sepref-fr-rules*] =
VMTF-it.introsort-param-impl-correct[*unfolded VMTF-sort-scores-raw-def*[*symmetric*] *PR-CONST-def*]

sepref-register *VMTF-sort-scores-raw vmtf-reorder-list-raw*

lemma *VMTF-sort-scores-vmtf-reorder-list-raw*:
‹(*VMTF-sort-scores*, *vmtf-reorder-list-raw*) ∈ *Id* → *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

sepref-def *VMTF-sort-scores-raw-impl*

is ‹*uncurry VMTF-sort-scores*›
:: ‹(*IICF-Array.array-assn vmtf-node-assn*)^k $*_a$ *VMTF-it.arr-assn*^{d} $\rightarrow_a$ *VMTF-it.arr-assn*›
⟨*proof*⟩

lemmas[*sepref-fr-rules*] =
VMTF-sort-scores-raw-impl.refine[*FCOMP VMTF-sort-scores-vmtf-reorder-list-raw*]

sepref-def *VMTF-sort-scores-impl*
is ‹*uncurry vmtf-reorder-list*›
:: ‹(*vmtf-assn*)^k $*_a$ *VMTF-it.arr-assn*^{d} $\rightarrow_a$ *VMTF-it.arr-assn*›
⟨*proof*⟩

sepref-def *atoms-hash-del-code*
is ‹*uncurry* (*RETURN oo atoms-hash-del*)›
:: ‹[*uncurry atoms-hash-del-pre*]$_a$ *atom-assn*^{k} $*_a$ (*atoms-hash-assn*)^d → *atoms-hash-assn*›
⟨*proof*⟩

sepref-def *atoms-hash-insert-code*
is ‹*uncurry* (*RETURN oo atoms-hash-insert*)›
:: ‹[*uncurry atms-hash-insert-pre*]$_a$
atom-assn^k $*_a$ (*distinct-atoms-assn*)^d → *distinct-atoms-assn*›
⟨*proof*⟩

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-def *update-next-search-impl*
is ‹*uncurry* (*RETURN oo update-next-search*)›
:: ‹(*atom.option-assn*)^k $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$ *vmtf-remove-assn*›
⟨*proof*⟩

lemma *case-option-split*:
‹(*case a of None* ⇒ *x* | *Some y* ⇒ *f y*) =
(*if is-None a then x else let y* = *the a in f y*)›
⟨*proof*⟩

sepref-def *ns-vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo ns-vmtf-dequeue*)›
:: ‹[*vmtf-dequeue-pre*]$_a$
atom-assn^k $*_a$ (*array-assn vmtf-node-assn*)^d → *array-assn vmtf-node-assn*›
⟨*proof*⟩

sepref-register *get-next get-prev stamp*
lemma *eq-Some-iff*: ‹*x* = *Some b* ⟷ (¬*is-None x* ∧ *the x* = *b*)›
⟨*proof*⟩

```
lemma hfref-refine-with-pre:
  assumes ‹⋀x. P x ⟹ g' x ≤ g x›
  assumes ‹(f,g') ∈ [P]ad A → R›
  shows ‹(f,g) ∈ [P]ad A → R›
  ⟨proof⟩
```

```
lemma isa-vmtf-en-dequeue-preI:
  assumes ‹isa-vmtf-en-dequeue-pre ((M,L),(ns, m, fst-As, lst-As, next-search))›
  shows ‹fst-As < length ns› ‹L < length ns› ‹Suc m < max-unat 64›
    and ‹get-next (ns!L) = Some i ⟶ i < length ns›
    and ‹fst-As ≠ lst-As ⟶ get-prev (ns ! lst-As) ≠ None›
    and ‹get-next (ns ! fst-As) ≠ None ⟶ get-prev (ns ! lst-As) ≠ None›
  ⟨proof⟩
```

```
find-theorems ‹- ≠ None ⟷ -›
```

```
lemma isa-vmtf-en-dequeue-alt-def2:
   ‹isa-vmtf-en-dequeue-pre x ⟹ uncurry2 (λM L vm.
    case vm of (ns, m, fst-As, lst-As, next-search) ⇒ doN {
      ASSERT(L<length ns);
      nsL ← mop-list-get ns (index-of-atm L);
      let fst-As = (if fst-As = L then get-next nsL else (Some fst-As));

      let next-search = (if next-search = (Some L) then get-next nsL
                         else next-search);
      let lst-As = (if lst-As = L then get-prev nsL else (Some lst-As));
      ASSERT (vmtf-dequeue-pre (L,ns));
      let ns = ns-vmtf-dequeue L ns;
      ASSERT (defined-atm-pol-pre M L);
      let de = (defined-atm-pol M L);
      ASSERT (Suc m < max-unat 64);
      case fst-As of
        None ⇒ RETURN
          (ns[L := VMTF-Node m fst-As None], m + 1, L, L,
           if de then None else Some L)
      | Some fst-As ⇒ doN {
          ASSERT (L < length ns ∧ fst-As < length ns ∧ lst-As ≠ None);
          let fst-As' =
                VMTF-Node (stamp (ns ! fst-As)) (Some L)
                 (get-next (ns ! fst-As));
          RETURN (
            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)
      }
    }) x
  ≤ uncurry2 (isa-vmtf-en-dequeue) x
   ›
  ⟨proof⟩
```

```
sepref-register 1 0
```

lemma *vmtf-en-dequeue-fast-codeI*:
  assumes ‹*isa-vmtf-en-dequeue-pre* ((*M*, *L*),(*ns*,*m*,*fst-As*, *lst-As*, *next-search*))›
  shows ‹*Suc m* < *max-unat 64*›
  ⟨*proof*⟩

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE trail-pol-fast-assn ?fr*›
  ⟨*proof*⟩

sepref-def *vmtf-en-dequeue-fast-code*
  is ‹*uncurry2 isa-vmtf-en-dequeue*›
  :: ‹[*isa-vmtf-en-dequeue-pre*]$_a$
    trail-pol-fast-assn^k $*_a$ *atom-assn*^{k} $*_a$ *vmtf-assn*^{d} → *vmtf-assn*›
  ⟨*proof*⟩

sepref-register *vmtf-rescale*
sepref-def *vmtf-rescale-code*
  is ‹*vmtf-rescale*›
  :: ‹*vmtf-assn*^{d} $\rightarrow_a$ *vmtf-assn*›
  ⟨*proof*⟩

sepref-register *partition-between-ref*

sepref-register *isa-vmtf-enqueue*

lemma *emptied-list-alt-def*: ‹*emptied-list xs* = *take 0 xs*›
  ⟨*proof*⟩

sepref-def *current-stamp-impl*
  is ‹*RETURN o current-stamp*›
  :: ‹*vmtf-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
  ⟨*proof*⟩

sepref-register *isa-vmtf-en-dequeue*

sepref-def *isa-vmtf-flush-fast-code*
  is ‹*uncurry isa-vmtf-flush-int*›
  :: ‹*trail-pol-fast-assn*^{k} $*_a$ (*vmtf-remove-assn*)^d $\rightarrow_a$
    vmtf-remove-assn›
  ⟨*proof*⟩

sepref-register *isa-vmtf-mark-to-rescore*
sepref-def *isa-vmtf-mark-to-rescore-code*
  is ‹*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore*)›
  :: ‹[*uncurry isa-vmtf-mark-to-rescore-pre*]$_a$
    atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*›

⟨*proof*⟩

sepref-register *isa-vmtf-unset*
sepref-def *isa-vmtf-unset-code*
is ‹*uncurry* (*RETURN oo isa-vmtf-unset*)›
:: ‹$[$*uncurry vmtf-unset-pre*$]_a$
atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*›
⟨*proof*⟩

lemma *isa-vmtf-mark-to-rescore-and-unsetI*: ‹
atms-hash-insert-pre ak (*ad*, *ba*) ⟹
isa-vmtf-mark-to-rescore-pre ak ((*a*, *aa*, *ab*, *ac*, *Some ak'*), *ad*, *ba*)›
⟨*proof*⟩

sepref-def *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*$]_a$
atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*›
⟨*proof*⟩

sepref-def *find-decomp-wl-imp-fast-code*
is ‹*uncurry2* (*isa-find-decomp-wl-imp*)›
:: ‹$[\lambda((M, lev), vm).\ True]_a$ *trail-pol-fast-assn*^{d} $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-remove-assn*^{d}
→ *trail-pol-fast-assn* $\times_a$ *vmtf-remove-assn*›
⟨*proof*⟩

sepref-def *vmtf-rescore-fast-code*
is ‹*uncurry2 isa-vmtf-rescore*›
:: ‹*clause-ll-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$
vmtf-remove-assn›
⟨*proof*⟩

sepref-def *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›
⟨*proof*⟩

lemma (**in** −) *arena-is-valid-clause-idx-le-uint64-max*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *sint64-max* ⟹
bd + *arena-length be bd* < *max-snat 64*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *sint64-max* ⟹
bd < *max-snat 64*›
⟨*proof*⟩

sepref-def *vmtf-mark-to-rescore-clause-fast-code*
is ‹*uncurry2* (*isa-vmtf-mark-to-rescore-clause*)›
:: ‹$[\lambda((N, \text{-}), \text{-}).\ length\ N \le sint64\text{-}max]_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*›

⟨*proof*⟩

sepref-def *vmtf-mark-to-rescore-also-reasons-fast-code*
  is ‹*uncurry3* (*isa-vmtf-mark-to-rescore-also-reasons*)›
  :: ‹$[\lambda(((\text{-}, N), \text{-}), \text{-}).\ \mathit{length}\ N \leq \mathit{sint64\text{-}max}]_a$
     $\mathit{trail\text{-}pol\text{-}fast\text{-}assn}^k *_a \mathit{arena\text{-}fast\text{-}assn}^k *_a \mathit{out\text{-}learned\text{-}assn}^k *_a \mathit{vmtf\text{-}remove\text{-}assn}^d \rightarrow$
     vmtf-remove-assn›
  ⟨*proof*⟩

experiment begin

export-llvm
  ns-vmtf-dequeue-code
  atoms-hash-del-code
  atoms-hash-insert-code
  update-next-search-impl
  ns-vmtf-dequeue-code
  vmtf-en-dequeue-fast-code
  vmtf-rescale-code
  current-stamp-impl
  isa-vmtf-flush-fast-code
  isa-vmtf-mark-to-rescore-code
  isa-vmtf-unset-code
  vmtf-mark-to-rescore-and-unset-code
  find-decomp-wl-imp-fast-code
  vmtf-rescore-fast-code
  find-decomp-wl-imp′-fast-code
  vmtf-mark-to-rescore-clause-fast-code
  vmtf-mark-to-rescore-also-reasons-fast-code

end

end
theory *IsaSAT-Show*
  imports
    Show.Show-Instances
    IsaSAT-Setup
begin

Chapter 12

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.

For the LLVM version code equations are not supported and hence we replace the function by hand.

definition *println-string* :: ‹*String.literal* ⇒ *unit*› **where**
  ‹*println-string* - = ()›

definition *print-c* :: ‹*64 word* ⇒ *unit*› **where**
  ‹*print-c* - = ()›

definition *print-char* :: ‹*64 word* ⇒ *unit*› **where**
  ‹*print-char* - = ()›

definition *print-uint64* :: ‹*64 word* ⇒ *unit*› **where**
  ‹*print-uint64* - = ()›

12.0.1 Print Information for IsaSAT

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 *print-open-colour* :: ‹*64 word* ⇒ *unit*› **where**
  ‹*print-open-colour* - = ()›

```
definition print-close-colour :: ⟨64 word ⇒ unit⟩ where
  ⟨print-close-colour - = ()⟩

definition isasat-current-information :: ⟨64 word ⇒ stats ⇒ - ⇒ stats⟩ where
⟨isasat-current-information =
   (λcurr-phase (propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds) lcount.
     if confl AND 8191 = 8191 — (8191::'a) = (8192::'a) − (1::'a), i.e., we print when all first bits are
1.
     then do{
       let - = print-c propa;
         - = if curr-phase = 1 then print-open-colour 33 else ();
         - = print-char 126;
         - = print-uint64 propa;
         - = print-uint64 confl;
         - = print-uint64 (of-nat lcount);
         - = print-uint64 frestarts;
         - = print-uint64 lrestarts;
         - = print-uint64 uset;
         - = print-uint64 gcs;
         - = print-uint64 (ema-extract-value lbds);
         - = print-close-colour 0
       in
         (propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds)}
      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, clvls, cach, lbd, outl, stats,
       heur, avdom,
       vdom, lcount, opts, old-arena).
     let curr-phase = current-restart-phase heur;
        stats = (isasat-current-information curr-phase stats lcount)
     in RETURN (M', N', D', j, W', vm, clvls, cach, lbd, outl, stats,
       heur, 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}  →f
   ⟨{(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ r}⟩nres-rel⟩
  ⟨proof⟩

definition isasat-print-progress :: ⟨64 word ⇒ 64 word ⇒ stats ⇒ - ⇒ unit⟩ where
⟨isasat-print-progress c curr-phase =
   (λ(propa, confl, decs, frestarts, lrestarts, uset, gcs, lbds) lcount.
     let
         - = print-c propa;
         - = if curr-phase = 1 then print-open-colour 33 else ();
         - = print-char (48 + c);
         - = print-uint64 propa;
         - = print-uint64 confl;
         - = print-uint64 (of-nat lcount);
         - = print-uint64 frestarts;
         - = print-uint64 lrestarts;
         - = print-uint64 uset;
         - = print-uint64 gcs;
         - = print-uint64 (ema-extract-value lbds);
         - = print-close-colour 0
     in
       ())›


definition isasat-current-progress :: ‹64 word ⇒ twl-st-wl-heur ⇒ unit nres› where
‹isasat-current-progress =
   (λc (M', N', D', j, W', vm, clvls, cach, lbd, outl, stats,
       heur, avdom,
       vdom, lcount, opts, old-arena).
     let
       curr-phase = current-restart-phase heur;
       - = isasat-print-progress c curr-phase stats lcount
     in RETURN ())›


end
theory IsaSAT-Rephase
  imports IsaSAT-Setup IsaSAT-Show
begin
```

Chapter 13

Rephasing

We implement the idea in CaDiCaL of rephasing:

- We remember the best model found so far. It is used as base.
- We flip the phase saving heuristics between *True*, *False*, and random.

```
definition rephase-init :: ‹bool ⇒ bool list ⇒ bool list nres› where
‹rephase-init b φ = do {
  let n = length φ;
  nfoldli [0..<n]
    (λ-. True)
    (λ a φ. do {
        ASSERT(a < length φ);
        RETURN (φ[a := b])
    })
    φ
 }›
```

```
lemma rephase-init-spec:
  ‹rephase-init b φ ≤ SPEC(λψ. length ψ = length φ)›
⟨proof⟩
```

```
definition copy-phase :: ‹bool list ⇒ bool list ⇒ bool list nres› where
‹copy-phase φ φ' = do {
  ASSERT(length φ = length φ');
  let n = length φ';
  nfoldli [0..<n]
    (λ-. True)
    (λ a φ'. do {
        ASSERT(a < length φ);
        ASSERT(a < length φ');
        RETURN (φ'[a := φ!a])
    })
    φ'
 }›
```

```
lemma copy-phase-alt-def:
‹copy-phase φ φ' = do {
  ASSERT(length φ = length φ');
  let n = length φ;
  nfoldli [0..<n]
    (λ-. True)
    (λ a φ'. do {
       ASSERT(a < length φ);
       ASSERT(a < length φ');
       RETURN (φ'[a := φ!a])
    })
    φ'
 }›
  ⟨proof⟩
```

lemma *copy-phase-spec*:

```
  ‹length φ = length φ' ⟹ copy-phase φ φ' ≤ SPEC(λψ. length ψ = length φ)›
  ⟨proof⟩
```

definition *rephase-random* :: ‹*64 word ⇒ bool list ⇒ bool list nres*› **where**

```
‹rephase-random b φ = do {
  let n = length φ;
  (-, φ) ← nfoldli [0..<n]
      (λ-. True)
      (λa (state, φ). do {
        ASSERT(a < length φ);
       let state = state * 6364136223846793005 + 1442695040888963407;
       RETURN (state, φ[a := (state < 2147483648)])
     })
     (b, φ);
  RETURN φ
 }›
```

lemma *rephase-random-spec*:

```
  ‹rephase-random b φ ≤ SPEC(λψ. length ψ = length φ)›
  ⟨proof⟩
```

definition *phase-rephase* :: ‹*64 word ⇒ phase-save-heur ⇒ phase-save-heur nres*› **where**

```
‹phase-rephase = (λb (φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase, length-phase).
    if b = 0
    then do {
      if curr-phase = 0
      then do {
         φ ← rephase-init False φ;
           RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 1,
length-phase)
      }
      else if curr-phase = 1
      then do {
         φ ← copy-phase best φ;
           RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 2,
length-phase)
      }
      else if curr-phase = 2
      then do {
         φ ← rephase-init True φ;
```

```
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 3, length-phase)
      }
      else if curr-phase = 3
      then do {
        φ ← rephase-random end-of-phase φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 4, length-phase)
      }
      else do {
        φ ← copy-phase best φ;
         RETURN (φ, target-assigned, target, best-assigned, best, (1+length-phase)*100+end-of-phase, 0,
           length-phase+1)
      }
    }
    else do {
      if curr-phase = 0
      then do {
        φ ← rephase-init False φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 1, length-phase)
      }
      else if curr-phase = 1
      then do {
        φ ← copy-phase best φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 2, length-phase)
      }
      else if curr-phase = 2
      then do {
        φ ← rephase-init True φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase*100+end-of-phase, 3, length-phase)
      }
      else do {
        φ ← copy-phase best φ;
         RETURN (φ, target-assigned, target, best-assigned, best, (1+length-phase)*100+end-of-phase, 0,
          length-phase+1)
     }
    })›
```

lemma *phase-rephase-spec*:
  assumes ‹*phase-save-heur-rel 𝒜 φ*›
  shows ‹*phase-rephase b φ ≤ ⇓Id (SPEC(phase-save-heur-rel 𝒜))*›
⟨*proof*⟩

definition *rephase-heur* :: ‹*64 word ⇒ restart-heuristics ⇒ restart-heuristics nres*› **where**

```
  ‹rephase-heur = (λb (fast-ema, slow-ema, restart-info, wasted, φ).
    do {
      φ ← phase-rephase b φ;
      RETURN (fast-ema, slow-ema, restart-info, wasted, φ)
   })›
```

lemma *rephase-heur-spec*:

```
  ‹heuristic-rel 𝒜 heur ⟹ rephase-heur b heur ≤ ⇓Id (SPEC(heuristic-rel 𝒜))›
  ⟨proof⟩

definition rephase-heur-st :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹rephase-heur-st = (λ(M', arena, D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena). do {
      let b = current-restart-phase heur;
      heur ← rephase-heur b heur;
      let - = isasat-print-progress (current-rephasing-phase heur) b stats lcount;
      RETURN (M', arena, D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena)
   })›

lemma rephase-heur-st-spec:
  ‹(S, S') ∈ twl-st-heur ⟹ rephase-heur-st S ≤ SPEC(λS. (S, S') ∈ twl-st-heur)›
  ⟨proof⟩

definition phase-save-phase :: ‹nat ⇒ phase-save-heur ⇒ phase-save-heur nres› where
‹phase-save-phase = (λn (φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase). do {
       target ← (if n > target-assigned
          then copy-phase φ target else RETURN target);
       target-assigned ← (if n > target-assigned
          then RETURN n else RETURN target-assigned);
       best ← (if n > best-assigned
          then copy-phase φ best else RETURN best);
       best-assigned ← (if n > best-assigned
          then RETURN n else RETURN best-assigned);
       RETURN (φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase)
   })›

lemma phase-save-phase-spec:
  assumes ‹phase-save-heur-rel 𝒜 φ›
  shows ‹phase-save-phase n φ ≤ ⇓Id (SPEC(phase-save-heur-rel 𝒜))›
⟨proof⟩

definition save-rephase-heur :: ‹nat ⇒ restart-heuristics ⇒ restart-heuristics nres› where
  ‹save-rephase-heur = (λn (fast-ema, slow-ema, restart-info, wasted, φ).
     do {
      φ ← phase-save-phase n φ;
      RETURN (fast-ema, slow-ema, restart-info, wasted, φ)
   })›

lemma save-phase-heur-spec:
  ‹heuristic-rel 𝒜 heur ⟹ save-rephase-heur n heur ≤ ⇓Id (SPEC(heuristic-rel 𝒜))›
  ⟨proof⟩


definition save-phase-st :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹save-phase-st = (λ(M', arena, D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena). do {
      ASSERT(isa-length-trail-pre M');
      let n = isa-length-trail M';
      heur ← save-rephase-heur n heur;
      RETURN (M', arena, D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena)
   })›
```

```
lemma save-phase-st-spec:
  ‹(S, S′) ∈ twl-st-heur ⟹ save-phase-st S ≤ SPEC(λS. (S, S′) ∈ twl-st-heur)›
  ⟨proof⟩
```

```
end
theory IsaSAT-LBD
  imports IsaSAT-Setup
begin
```

```
definition mark-lbd-from-clause-heur :: ‹trail-pol ⇒ arena ⇒ nat ⇒ lbd ⇒ lbd nres› where
  ‹mark-lbd-from-clause-heur M N C lbd = do {
  n ← mop-arena-length N C;
  nfoldli [0..<n] (λ-. True)
    (λi lbd. do {
        L ← mop-arena-lit2 N C i;
        ASSERT(get-level-pol-pre (M, L));
        let lev = get-level-pol M L;
        ASSERT(lev ≤ Suc (uint32-max div 2));
        RETURN (if lev = 0 then lbd else lbd-write lbd lev)})
    lbd}›
```

```
lemma count-decided-le-length: ‹count-decided M ≤ length M›
  ⟨proof⟩
```

```
lemma mark-lbd-from-clause-heur-correctness:
  assumes ‹(M, M′) ∈ trail-pol 𝒜› and ‹valid-arena N N′ vdom› ‹C ∈# dom-m N′› and
    ‹literals-are-in-ℒin 𝒜 (mset (N′ ∝ C))›
  shows ‹mark-lbd-from-clause-heur M N C lbd ≤ ⇓ Id (SPEC(λ-::bool list. True))›
  ⟨proof⟩
```

```
definition calculate-LBD-st :: ‹(nat, nat) ann-lits ⇒ nat clauses-l ⇒ nat ⇒ nat clauses-l nres› where
  ‹calculate-LBD-st = (λM N C. RETURN N)›
```

```
abbreviation TIER-ONE-MAXIMUM where
  ‹TIER-ONE-MAXIMUM ≡ 6›
definition calculate-LBD-heur-st :: ‹- ⇒ arena ⇒ lbd ⇒ nat ⇒ (arena × lbd) nres› where
  ‹calculate-LBD-heur-st = (λM N lbd C. do{
      old-glue ← mop-arena-lbd N C;
      st ← mop-arena-status N C;
      if st = IRRED then RETURN (N, lbd)
      else if old-glue < TIER-ONE-MAXIMUM then do {
        N ← mop-arena-mark-used2 N C;
        RETURN (N, lbd)
      }
      else do {
        lbd ← mark-lbd-from-clause-heur M N C lbd;
        glue ← get-LBD lbd;
        lbd ← lbd-empty lbd;
        N ← (if glue < old-glue then mop-arena-update-lbd C glue N else RETURN N);
      N ← (if glue < TIER-ONE-MAXIMUM ∨ old-glue < TIER-ONE-MAXIMUM then mop-arena-mark-used2
N C else mop-arena-mark-used N C);
        RETURN (N, lbd)
    }})›
```

```
lemma calculate-LBD-st-alt-def:
  ‹calculate-LBD-st = (λM N C. do {
      old-glue :: nat ← SPEC(λ- . True);
      st :: clause-status ← SPEC(λ- . True);
      if st = IRRED then RETURN N
      else if old-glue < 6 then  do {
         - ← RETURN N;
         RETURN N
      }
      else do {
       lbd::bool list ← SPEC(λ-. True);
       glue::nat ← get-LBD lbd;
       -::bool list ← lbd-empty lbd;
       - ← RETURN N;
       - ← RETURN N;
      RETURN N
    }})› (is ‹?A = ?B›)
  ⟨proof⟩
```

```
lemma RF-COME-ON: ‹(x, y) ∈ Id ⟹ f x ≤ ⇓ Id (f y)›
  ⟨proof⟩

lemma mop-arena-update-lbd:
  ‹C ∈# dom-m N ⟹ valid-arena arena N vdom ⟹
     mop-arena-update-lbd C glue arena ≤ SPEC(λc. (c, N) ∈ {(c, N′). N′=N ∧ valid-arena c N vdom
∧
       length c = length arena})›
  ⟨proof⟩

lemma mop-arena-mark-used-valid:
  ‹C ∈# dom-m N ⟹ valid-arena arena N vdom ⟹
     mop-arena-mark-used arena C ≤ SPEC(λc. (c, N) ∈ {(c, N′). N′=N ∧ valid-arena c N vdom ∧
       length c = length arena})›
  ⟨proof⟩

lemma mop-arena-mark-used2-valid:
  ‹C ∈# dom-m N ⟹ valid-arena arena N vdom ⟹
     mop-arena-mark-used2 arena C ≤ SPEC(λc. (c, N) ∈ {(c, N′). N′=N ∧ valid-arena c N vdom ∧
       length c = length arena})›
  ⟨proof⟩

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 calculate-LBD-heur-st-calculate-LBD-st:
  assumes ‹valid-arena arena N vdom›
    ‹(M, M′) ∈ trail-pol 𝒜›
    ‹C ∈# dom-m N›
    ‹literals-are-in-ℒin 𝒜 (mset (N ∝ C))› ‹(C, C′) ∈ nat-rel›
  shows ‹calculate-LBD-heur-st M arena lbd C ≤
     ⇓{((arena′, lbd), N′). valid-arena arena′ N′ vdom ∧ N = N′ ∧ length arena = length arena′}
```

```
      (calculate-LBD-st M' N C')›
⟨proof⟩


definition mark-lbd-from-list :: ‹-› where
  ‹mark-lbd-from-list M C lbd = do {
    nfoldli (drop 1 C) (λ-. True)
      (λL lbd. RETURN (lbd-write lbd (get-level M L))) lbd
  }›

definition mark-lbd-from-list-heur :: ‹trail-pol ⇒ nat clause-l ⇒ lbd ⇒ lbd nres› where
  ‹mark-lbd-from-list-heur M C lbd = do {
  let n = length C;
  nfoldli [1..<n] (λ-. True)
    (λi lbd. do {
       ASSERT(i < length C);
       let L = C ! i;
       ASSERT(get-level-pol-pre (M, L));
       let lev = get-level-pol M L;
       ASSERT(lev ≤ Suc (uint32-max div 2));
       RETURN (if lev = 0 then lbd else lbd-write lbd lev)})
    lbd}›

definition mark-lbd-from-conflict :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹mark-lbd-from-conflict = (λ(M, N, D, Q, W, vm, clvls, cach, lbd, outl, stats, heur, vdom, avdom,
        lcount). do{
     lbd ← mark-lbd-from-list-heur M outl lbd;
     RETURN (M, N, D, Q, W, vm, clvls, cach, lbd, outl, stats,
         heur, vdom, avdom, lcount)
    })›


lemma mark-lbd-from-list-heur-correctness:
  assumes ‹(M, M') ∈ trail-pol 𝒜› and ‹literals-are-in-ℒin 𝒜 (mset (tl C))›
  shows ‹mark-lbd-from-list-heur M C lbd ≤ ⇓ Id (SPEC(λ-::bool list. True))›
  ⟨proof⟩


definition mark-LBD-st :: ‹'v twl-st-wl ⇒ ('v twl-st-wl) nres› where
  ‹mark-LBD-st = (λS. SPEC (λ(T). S = T))›

lemma mark-LBD-st-alt-def:
  ‹mark-LBD-st S = do {n :: bool list ← SPEC (λ-. True); SPEC (λ(T). S = T)}›
  ⟨proof⟩

lemma mark-lbd-from-conflict-mark-LBD-st:
  ‹(mark-lbd-from-conflict, mark-LBD-st) ∈
    [λS. get-conflict-wl S ≠ None ∧ literals-are-in-ℒin (all-atms-st S) (the (get-conflict-wl S))]f
     twl-st-heur-conflict-ana → ⟨twl-st-heur-conflict-ana⟩nres-rel›
  ⟨proof⟩

end
theory IsaSAT-Backtrack
  imports IsaSAT-Setup IsaSAT-VMTF IsaSAT-Rephase IsaSAT-LBD
begin
```

Chapter 14

Backtrack

The backtrack function is highly complicated and tricky to maintain.

14.1 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 𝒜 M D outl = do {
     let C = replicate (length outl) (outl!0);
     (D, C, -) ← 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 {
           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 ∈# ℒ_all 𝒜 ∧ C!1 ∈# ℒ_all 𝒜 ∧ 1 < length C);
           let C = (if get-level M (C!i) > get-level M (C!1) then swap C 1 i else C);
```

```
          ASSERT(i+1 ≤ uint32-max);
          RETURN (D, C, i+1)
        })
        (D, C, 1);
      ASSERT(length outl ≠ 1 ⟶ length C > 1);
      ASSERT(length outl ≠ 1 ⟶ C!1 ∈# ℒ_all 𝒜);
      RETURN ((True, D), C, if length outl = 1 then 0 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-ℒ_in 𝒜 (mset outl)› and
    DD': ‹(D', D) ∈ lookup-clause-rel 𝒜› and
    consistent: ‹¬ tautology (mset outl)› and
    bounded: ‹isasat-input-bounded 𝒜›
  shows
    ‹empty-conflict-and-extract-clause-heur 𝒜 M D' outl ≤ ⇓ (option-lookup-clause-rel 𝒜 ×_r Id ×_r Id)
       (empty-conflict-and-extract-clause M D outl)›
⟨proof⟩
```

definition *isa-empty-conflict-and-extract-clause-heur* ::

```
  ‹trail-pol ⇒ lookup-clause-rel ⇒ nat literal list ⇒ (- × nat literal list × nat) nres›
  where
    ‹isa-empty-conflict-and-extract-clause-heur M D outl = do {
      let C = replicate (length outl) (outl!0);
      (D, C, -) ← WHILE_T
          (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!1));
     ASSERT(1 < length C);
            let C = (if get-level-pol M (C!i) > get-level-pol M (C!1) then swap C 1 i else C);
            ASSERT(i+1 ≤ uint32-max);
            RETURN (D, C, i+1)
          })
        (D, C, 1);
      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 0 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
𝒜)) ∈
     trail-pol 𝒜 ×_f Id ×_f Id →_f ⟨Id⟩nres-rel ›
⟨proof⟩
```

```
definition extract-shorter-conflict-wl-nlit where
  ‹extract-shorter-conflict-wl-nlit K M NU D NE UE =
    SPEC(λD'. D' ≠ None ∧ the D' ⊆# the D ∧ K ∈# the D' ∧
      mset ‘# ran-mf NU + NE + UE ⊨pm the D')›

definition extract-shorter-conflict-wl-nlit-st
  :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres›
  where
    ‹extract-shorter-conflict-wl-nlit-st =
     (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-inv S')›

definition extract-shorter-conflict-heur where
  ‹extract-shorter-conflict-heur = (λM 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 ≤ uint32-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-cach-ref-set-inv cach0 support
      (λ(i, cach). i < length support)
      (λ(i, cach). do {
          ASSERT(i < length support);
          ASSERT(support ! i < length cach);
          RETURN(i+1, cach[support ! i := SEEN-UNKNOWN])
      })
     (0, cach0);
    RETURN (cach, emptied-list support)
  })›
```

```
lemma empty-cach-ref-set-empty-cach-ref:
  ‹(empty-cach-ref-set, RETURN o empty-cach-ref) ∈
    [λ(cach, supp). (∀ L ∈ set supp. L < length cach) ∧ length supp ≤ Suc (uint32-max div 2) ∧
      (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set supp)]_f
    Id → ⟨Id⟩ nres-rel›
⟨proof⟩
```

```
lemma empty-cach-ref-empty-cach:
  ‹isasat-input-bounded 𝒜 ⟹ (RETURN o empty-cach-ref, RETURN o empty-cach) ∈ cach-refinement
𝒜 →_f ⟨cach-refinement 𝒜⟩ nres-rel›
  ⟨proof⟩
```

```
definition empty-cach-ref-pre where
  ‹empty-cach-ref-pre = (λ(cach :: minimize-status list, supp :: nat list).
          (∀ L∈set supp. L < length cach) ∧
          length supp ≤ Suc (uint32-max div 2) ∧
          (∀ L<length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set supp))›
```

Minimisation of the conflict

```
definition extract-shorter-conflict-list-heur-st
  :: ‹twl-st-wl-heur ⇒ (twl-st-wl-heur × - × -) nres›
  where
    ‹extract-shorter-conflict-list-heur-st = (λ(M, N, (-, D), Q', W', vm, clvls, cach, lbd, outl,
        stats, ccont, vdom). do {
     lbd ← mark-lbd-from-list-heur M outl lbd;
     ASSERT(fst M ≠ []);
     let K = lit-of-last-trail-pol M;
     ASSERT(0 < length outl);
     ASSERT(lookup-conflict-remove1-pre (−K, D));
     let D = lookup-conflict-remove1 (−K) D;
     let outl = outl[0 := −K];
     vm ← isa-vmtf-mark-to-rescore-also-reasons M N outl vm;
     (D, cach, outl) ← isa-minimize-and-extract-highest-lookup-conflict M N D cach lbd outl;
     ASSERT(empty-cach-ref-pre cach);
     let cach = empty-cach-ref cach;
     ASSERT(outl ≠ [] ∧ length outl ≤ uint32-max);
     (D, C, n) ← isa-empty-conflict-and-extract-clause-heur M D outl;
     RETURN ((M, N, D, Q', W', vm, clvls, cach, lbd, take 1 outl, stats, ccont, vdom), n, C)
  })›
```

```
lemma the-option-lookup-clause-assn:
```

⟨(*RETURN o snd*, *RETURN o the*) ∈ [*λD*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*⟩
⟨*proof*⟩

definition *update-heuristics* **where**

```
⟨update-heuristics = (λglue (fema, sema, res-info, wasted).
   (ema-update glue fema, ema-update glue sema,
      incr-conflict-count-since-last-restart res-info, wasted))⟩
```

lemma *heuristic-rel-update-heuristics*[*intro*!]:
⟨*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*update-heuristics glue heur*)⟩
⟨*proof*⟩

definition *propagate-bt-wl-D-heur*
:: ⟨*nat literal* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*propagate-bt-wl-D-heur* = (*λL C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *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 ← isa-vmtf-rescore C M vm0;
    glue ← get-LBD lbd;
    let b = False;
    let b' = (length C = 2);
    ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
      vdom, avdom, lcount, opts) ⟶ append-and-length-fast-code-pre ((b, C), N0));
    ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
      vdom, avdom, lcount, opts) ⟶ lcount < sint64-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, y, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts) ⟶ length-ll W0 (nat-of-lit (−L)) < sint64-max);
    let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [(i, L', b')]];
    ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts) ⟶ length-ll W (nat-of-lit L') < sint64-max);
    let W = W[nat-of-lit L' := W!nat-of-lit L' @ [(i, −L, b')]];
    lbd ← lbd-empty lbd;
    j ← mop-isa-length-trail M;
    ASSERT(i ≠ DECISION-REASON);
    ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
    M ← cons-trail-Propagated-tr (− L) i M;
    vm ← isa-vmtf-flush-int M vm;
    heur ← mop-save-phase-heur (atm-of L') (is-neg L') heur;
    RETURN (M, N, D, j, W, vm, 0,
       cach, lbd, outl, add-lbd (of-nat glue) stats, update-heuristics glue heur, vdom @ [ i],
        avdom @ [i],
        lcount + 1, opts)
  })⟩
```

definition (**in** −) *lit-of-hd-trail-st-heur* :: ⟨*twl-st-wl-heur* ⇒ *nat literal nres*⟩ **where**
⟨*lit-of-hd-trail-st-heur S* = *do* {*ASSERT* (*fst* (*get-trail-wl-heur S*) ≠ []); *RETURN* (*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, clvls, cach, lbd, outl, stats,
        heur, vdom). do {
      vm ← isa-vmtf-flush-int M vm;
      glue ← get-LBD lbd;
      lbd ← lbd-empty lbd;
      j ← mop-isa-length-trail M;
      ASSERT(0 ≠ DECISION-REASON);
      ASSERT(cons-trail-Propagated-tr-pre ((− L, 0::nat), M));
      M ← cons-trail-Propagated-tr (− L) 0 M;
      let stats = incr-uset stats;
      RETURN (M, N, D, j, W, vm, clvls, cach, lbd, outl, stats,
        (update-heuristics glue heur), vdom)})›
```

Full function

```
definition backtrack-wl-D-nlit-heur
  :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
  where
    ‹backtrack-wl-D-nlit-heur S0 =
    do {
      ASSERT(backtrack-wl-D-heur-inv S0);
      ASSERT(fst (get-trail-wl-heur S0) ≠ []);
      L ← lit-of-hd-trail-st-heur S0;
      (S, n, C) ← extract-shorter-conflict-list-heur-st S0;
      ASSERT(get-clauses-wl-heur S = get-clauses-wl-heur S0);
      S ← find-decomp-wl-st-int n S;

      ASSERT(get-clauses-wl-heur S = get-clauses-wl-heur S0);
      if size C > 1
      then do {
        S ← propagate-bt-wl-D-heur L C S;
        save-phase-st S
      }
      else do {
        propagate-unit-bt-wl-D-int L S
     }
  }›
```

```
lemma get-all-ann-decomposition-get-level:
  assumes
    L': ‹L' = lit-of (hd M')› and
    nd: ‹no-dup M'› and
    decomp: ‹(Decided K # a, M2) ∈ set (get-all-ann-decomposition M')› and
    lev-K: ‹get-level M' K = Suc (get-maximum-level M' (remove1-mset (− L') y))› and
    L: ‹L ∈# remove1-mset (− lit-of (hd M')) y›
  shows ‹get-level a L = get-level M' L›
⟨proof⟩
```

```
definition del-conflict-wl :: ‹'v twl-st-wl ⇒ 'v twl-st-wl› where
```

```
  ‹del-conflict-wl = (λ(M, N, D, NE, UE, Q, W). (M, N, None, NE, UE, Q, W))›

lemma [simp]:
  ‹get-clauses-wl (del-conflict-wl S) = get-clauses-wl S›
  ⟨proof⟩

lemma lcount-add-clause[simp]: ‹i ∉# dom-m N ⟹
    size (learned-clss-l (fmupd i (C, False) N)) = Suc (size (learned-clss-l N))›
  ⟨proof⟩

lemma length-watched-le:
  assumes
    prop-inv: ‹correct-watching x1› and
    xb-x'a: ‹(x1a, x1) ∈ twl-st-heur-conflict-ana› and
    x2: ‹x2 ∈# 𝓛all (all-atms-st x1)›
  shows ‹length (watched-by x1 x2) ≤ length (get-clauses-wl-heur x1a) − MIN-HEADER-SIZE›
⟨proof⟩

definition single-of-mset where
  ‹single-of-mset D = SPEC(λL. D = mset [L])›

lemma backtrack-wl-D-nlit-backtrack-wl-D:
  ‹(backtrack-wl-D-nlit-heur, backtrack-wl) ∈
  {(S, T). (S, T) ∈ twl-st-heur-conflict-ana ∧ length (get-clauses-wl-heur S) = r} →f
  ⟨{(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ MAX-HEADER-SIZE+1 + r +
uint32-max div 2}⟩nres-rel›
  (is ‹- ∈ ?R →f ⟨?S⟩nres-rel›)
⟨proof⟩
```

14.2 Backtrack with direct extraction of literal if highest level

```
lemma le-uint32-max-div-2-le-uint32-max: ‹a ≤ uint32-max div 2 + 1 ⟹ a ≤ uint32-max›
  ⟨proof⟩


lemma propagate-bt-wl-D-heur-alt-def:
  ‹propagate-bt-wl-D-heur = (λL C (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
         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 ← isa-vmtf-rescore C M vm0;
      glue ← get-LBD lbd;
      let b = False;
      let b' = (length C = 2);
      ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts) ⟶ append-and-length-fast-code-pre ((b, C), N0));
      ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts) ⟶ lcount < sint64-max);
      (N, i) ← fm-add-new-fast b C N0;
      ASSERT(update-lbd-pre ((i, glue), N));
      let N = update-lbd i glue N;
      ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts) ⟶ length-ll W0 (nat-of-lit (−L)) < sint64-max);
      let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [(i, L′, b′)]];
      ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts) ⟶ length-ll W (nat-of-lit L′) < sint64-max);
      let W = W[nat-of-lit L′ := W!nat-of-lit L′ @ [(i, −L, b′)]];
      lbd ← lbd-empty lbd;
      j ← mop-isa-length-trail M;
      ASSERT(i ≠ DECISION-REASON);
      ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
      M ← cons-trail-Propagated-tr (− L) i M;
      vm ← isa-vmtf-flush-int M vm;
      heur ← mop-save-phase-heur (atm-of L′) (is-neg L′) heur;
      RETURN (M, N, D, j, W, vm, 0,
         cach, lbd, outl, add-lbd (of-nat glue) stats, update-heuristics glue heur, vdom @ [i],
          avdom @ [i],
          lcount + 1, opts)
   })›
  ⟨proof⟩


lemma propagate-bt-wl-D-fast-code-isasat-fastI2: ‹isasat-fast b ⟹
       b = (a1′, a2′) ⟹
       a2′ = (a1′a, a2′a) ⟹
       a < length a1′a ⟹ a ≤ sint64-max›
  ⟨proof⟩

lemma propagate-bt-wl-D-fast-code-isasat-fastI3: ‹isasat-fast b ⟹
       b = (a1′, a2′) ⟹
       a2′ = (a1′a, a2′a) ⟹
       a ≤ length a1′a ⟹ a < sint64-max›
  ⟨proof⟩

lemma lit-of-hd-trail-st-heur-alt-def:
 ‹lit-of-hd-trail-st-heur = (λ(M, N, D, Q, W, vm, φ). do {ASSERT (fst M ≠ []); RETURN (lit-of-last-trail-pol
M)})›
  ⟨proof⟩

end
theory IsaSAT-Show-LLVM
  imports
    IsaSAT-Show
    IsaSAT-Setup-LLVM
begin


sepref-register isasat-current-information print-c print-uint64

sepref-def print-c-impl
  is ‹RETURN o print-c›
  :: ‹word-assn^k →_a unit-assn›
  ⟨proof⟩

sepref-def print-uint64-impl
  is ‹RETURN o print-uint64›
  :: ‹word-assn^k →_a unit-assn›
```

⟨*proof*⟩

sepref-def *print-open-colour-impl*
  is ‹*RETURN o print-open-colour*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-close-colour-impl*
  is ‹*RETURN o print-close-colour*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-char-impl*
  is ‹*RETURN o print-char*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *isasat-current-information-impl* [*llvm-code*]
  is ‹*uncurry2* (*RETURN ooo isasat-current-information*)›
  :: ‹*word-assn*^{k} $*_a$ *stats-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *stats-assn*›
  ⟨*proof*⟩

declare *isasat-current-information-impl.refine*[*sepref-fr-rules*]

lemma *current-restart-phase-alt-def*:
  ‹*current-restart-phase* =
    (λ(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ).
      restart-phase)›
  ⟨*proof*⟩

sepref-def *current-restart-phase-impl*
  is ‹*RETURN o current-restart-phase*›
  :: ‹*heuristic-assn*^{k} $\rightarrow_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *isasat-current-status-fast-code*
  is ‹*isasat-current-status*›
  :: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

sepref-def *isasat-print-progress-impl*
  is ‹*uncurry3* (*RETURN oooo isasat-print-progress*)›
  :: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *stats-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

term *isasat-current-progress*

sepref-def *isasat-current-progress-impl*
  is ‹*uncurry isasat-current-progress*›
  :: ‹*word-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} $\rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

end
theory *IsaSAT-Rephase-LLVM*

imports *IsaSAT-Rephase IsaSAT-Show-LLVM*
begin

sepref-def *rephase-random-impl*
is ‹*uncurry rephase-random*›
:: ‹$word\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
⟨*proof*⟩

sepref-def *rephase-init-impl*
is ‹*uncurry rephase-init*›
:: ‹$bool1\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
⟨*proof*⟩

sepref-def *copy-phase-impl*
is ‹*uncurry copy-phase*›
:: ‹$phase\text{-}saver\text{-}assn^k *_a phase\text{-}saver'\text{-}assn^d \rightarrow_a phase\text{-}saver'\text{-}assn$›
⟨*proof*⟩

definition *copy-phase2* **where**
‹*copy-phase2* = *copy-phase*›

sepref-def *copy-phase-impl2*
is ‹*uncurry copy-phase2*›
:: ‹$phase\text{-}saver'\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
⟨*proof*⟩

sepref-register *rephase-init rephase-random copy-phase*

sepref-def *phase-save-phase-impl*
is ‹*uncurry phase-save-phase*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a phase\text{-}heur\text{-}assn^d \rightarrow_a phase\text{-}heur\text{-}assn$›
⟨*proof*⟩

sepref-def *save-phase-heur-impl*
is ‹*uncurry save-rephase-heur*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$›
⟨*proof*⟩

sepref-def *save-phase-heur-st*
is *save-phase-st*
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-def *phase-save-rephase-impl*
is ‹*uncurry phase-rephase*›
:: ‹$word\text{-}assn^k *_a phase\text{-}heur\text{-}assn^d \rightarrow_a phase\text{-}heur\text{-}assn$›
⟨*proof*⟩

sepref-def *rephase-heur-impl*
is ‹*uncurry rephase-heur*›
:: ‹$word\text{-}assn^k *_a heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$›

⟨*proof*⟩

lemma *current-rephasing-phase-alt-def*:
‹*RETURN o current-rephasing-phase* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*,
(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*)).
RETURN curr-phase)›
⟨*proof*⟩

sepref-def *current-rephasing-phase*
is ‹*RETURN o current-rephasing-phase*›
:: ‹*heuristic-assn*^{k} $\to_a$ *word64-assn*›
⟨*proof*⟩

sepref-register *rephase-heur*
sepref-def *rephase-heur-st-impl*
is *rephase-heur-st*
:: ‹*isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
⟨*proof*⟩

experiment
begin
export-llvm *rephase-heur-st-impl*
save-phase-heur-st
end

end
theory *IsaSAT-LBD-LLVM*
imports *IsaSAT-LBD IsaSAT-Setup-LLVM*
begin

sepref-register *mark-lbd-from-clause-heur get-level-pol mark-lbd-from-list-heur*
mark-lbd-from-conflict mop-arena-status

sepref-def *mark-lbd-from-clause-heur-impl*
is ‹*uncurry3 mark-lbd-from-clause-heur*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *lbd-assn*^{d} $\to_a$ *lbd-assn*›
⟨*proof*⟩

sepref-def *calculate-LBD-heur-st-impl*
is ‹*uncurry3 calculate-LBD-heur-st*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{d} $*_a$ *lbd-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $\to_a$
arena-fast-assn $\times_a$ *lbd-assn*›
⟨*proof*⟩

sepref-def *mark-lbd-from-list-heur-impl*
is ‹*uncurry2 mark-lbd-from-list-heur*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *out-learned-assn*^{k} $*_a$ *lbd-assn*^{d} $\to_a$ *lbd-assn*›
⟨*proof*⟩

sepref-def *mark-lbd-from-conflict-impl*
is ‹*mark-lbd-from-conflict*›
:: ‹*isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
⟨*proof*⟩

```
end
theory IsaSAT-Backtrack-LLVM
  imports IsaSAT-Backtrack IsaSAT-VMTF-LLVM IsaSAT-Lookup-Conflict-LLVM
    IsaSAT-Rephase-LLVM IsaSAT-LBD-LLVM
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 (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!1));
    ASSERT(1 < length C);
           let L1 = C!i;
           let L2 = C!1;
           let C = (if get-level-pol M L1 > get-level-pol M L2 then swap C 1 i else C);
           ASSERT(i+1 ≤ uint32-max);
           RETURN (D, C, i+1)
         })
        (D, C, 1);
     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 0 else get-level-pol M (C!1))
  }›
  ⟨proof⟩

sepref-def empty-conflict-and-extract-clause-heur-fast-code
  is ‹uncurry2 (isa-empty-conflict-and-extract-clause-heur)›
  :: ‹[λ((M, D), outl). outl ≠ [] ∧ length outl ≤ uint32-max]_a
      trail-pol-fast-assn^k *_a lookup-clause-rel-assn^d *_a out-learned-assn^k →
       (conflict-option-rel-assn) ×_a clause-ll-assn ×_a uint32-nat-assn›
  ⟨proof⟩


lemma emptied-list-alt-def: ‹emptied-list xs = take 0 xs›
  ⟨proof⟩

sepref-def empty-cach-code
  is ‹empty-cach-ref-set›
  :: ‹cach-refinement-l-assn^d →_a cach-refinement-l-assn›
  ⟨proof⟩



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*⟩

sepref-register *fm-add-new-fast*

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* $\Longrightarrow$ *Suc* (*length* (*get-clauses-wl-heur S*)) < *max-snat 64*›
⟨*proof*⟩

sepref-register *update-heuristics*
sepref-def *update-heuristics-impl*
is [*llvm-inline*,*sepref-fr-rules*] ‹*uncurry* (*RETURN oo update-heuristics*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

sepref-register *cons-trail-Propagated-tr*
sepref-def *propagate-unit-bt-wl-D-fast-code*
is ‹*uncurry propagate-unit-bt-wl-D-int*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *propagate-bt-wl-D-fast-codeXX*
is ‹*uncurry2 propagate-bt-wl-D-heur*›
:: ‹[λ((*L*, *C*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k $*_a$ *clause-ll-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›

⟨*proof*⟩

lemma *extract-shorter-conflict-list-heur-st-alt-def*:
‹*extract-shorter-conflict-list-heur-st* = (λ(*M*, *N*, (*bD*), *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *ccont*, *vdom*). *do* {
lbd ← *mark-lbd-from-list-heur M outl lbd*;
let D = *the-lookup-conflict bD*;
ASSERT(*fst M* ≠ []);
let K = *lit-of-last-trail-pol M*;
ASSERT(*0* < *length outl*);
ASSERT(*lookup-conflict-remove1-pre* (−*K*, *D*));
let D = *lookup-conflict-remove1* (−*K*) *D*;
let outl = *outl*[*0* := −*K*];
vm ← *isa-vmtf-mark-to-rescore-also-reasons M N outl vm*;
(*D*, *cach*, *outl*) ← *isa-minimize-and-extract-highest-lookup-conflict M N D cach lbd outl*;
ASSERT(*empty-cach-ref-pre cach*);
let cach = *empty-cach-ref cach*;
ASSERT(*outl* ≠ [] ∧ *length outl* ≤ *uint32-max*);
(*D*, *C*, *n*) ← *isa-empty-conflict-and-extract-clause-heur M D outl*;
RETURN ((*M*, *N*, *D*, *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *take 1 outl*, *stats*, *ccont*, *vdom*), *n*, *C*)
})›
⟨*proof*⟩

sepref-register *isa-minimize-and-extract-highest-lookup-conflict*
empty-conflict-and-extract-clause-heur

sepref-def *extract-shorter-conflict-list-heur-st-fast*
is ‹*extract-shorter-conflict-list-heur-st*›
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn* $\times_a$ *uint32-nat-assn* $\times_a$ *clause-ll-assn*›
⟨*proof*⟩

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*

sepref-def *lit-of-hd-trail-st-heur-fast-code*
is ‹*lit-of-hd-trail-st-heur*›
:: ‹$[\lambda S.\ True]_a$ *isasat-bounded-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

sepref-register *save-phase-st*
sepref-def *backtrack-wl-D-fast-code*
is ‹*backtrack-wl-D-nlit-heur*›
:: ‹$[isasat\text{-}fast]_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

lemmas [*llvm-inline*] = *add-lbd-def*

experiment
begin

```
  export-llvm
    empty-conflict-and-extract-clause-heur-fast-code
    empty-cach-code
    update-heuristics-impl
    update-heuristics-impl
      isa-vmtf-flush-fast-code
      get-LBD-code
      mop-isa-length-trail-fast-code
    cons-trail-Propagated-tr-fast-code
      update-heuristics-impl
vmtf-rescore-fast-code
append-and-length-fast-code
    update-lbd-impl
```

thm *propagate-bt-wl-D-fast-codeXX-def*

```
  export-llvm
    empty-conflict-and-extract-clause-heur-fast-code
    empty-cach-code
    propagate-bt-wl-D-fast-codeXX
    propagate-unit-bt-wl-D-fast-code
    extract-shorter-conflict-list-heur-st-fast
    lit-of-hd-trail-st-heur-fast-code
    backtrack-wl-D-fast-code
```

end

end
theory *IsaSAT-Initialisation*
imports *Watched-Literals.Watched-Literals-Watch-List-Initialisation IsaSAT-Setup IsaSAT-VMTF*
Automatic-Refinement.Relators — for more lemmas
begin

Chapter 15

Initialisation

lemma *bitXOR-1-if-mod-2-int*: ‹*bitOR L 1* = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *int*
⟨*proof*⟩

lemma *bitOR-1-if-mod-2-nat*:
‹*bitOR L 1* = (*if L mod 2* = *0 then L* + *1 else L*)›
‹*bitOR L* (*Suc 0*) = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *nat*
⟨*proof*⟩

15.1 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.

15.1.1 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*) ∈
[$\lambda n.$ (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *n*)]$_f$ *nat-rel* → ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

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*) ∈
$[\lambda n.\ (\forall L{\in}\#\mathcal{L}_{all}\ \mathcal{A}.\ \textit{atm-of}\ L < n)]_f$ *nat-rel* → ⟨*distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

type-synonym *vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat set*›

type-synonym *isa-vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat list* × *bool list*›

definition *vmtf-init*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int-option-fst-As set*›
where
‹*vmtf-init* $\mathcal{A}_{in}$ *M* = {((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
$\mathcal{A}_{in}$ ≠ {#} ⟶ (*fst-As* ≠ *None* ∧ *lst-As* ≠ *None* ∧ ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*),
to-remove) ∈ *vmtf* $\mathcal{A}_{in}$ *M*)}›

definition *isa-vmtf-init* **where**
‹*isa-vmtf-init* $\mathcal{A}$ *M* =
$((\textit{Id} \times_r \textit{nat-rel} \times_r \langle\textit{nat-rel}\rangle\textit{option-rel} \times_r \langle\textit{nat-rel}\rangle\textit{option-rel} \times_r \langle\textit{nat-rel}\rangle\textit{option-rel}) \times_f$
$\textit{distinct-atoms-rel}\ \mathcal{A})^{-1}$
“ *vmtf-init* $\mathcal{A}$ *M*›

lemma *isa-vmtf-initI*:
‹(*vm*, *to-remove′*) ∈ *vmtf-init* $\mathcal{A}$ *M* ⟹ (*to-remove*, *to-remove′*) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹
(*vm*, *to-remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ *M*›
⟨*proof*⟩

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

lemma *vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf-init* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf-init* $\mathcal{B}$ *M*›
⟨*proof*⟩

lemma *isa-vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ⟹ *L* ∈ *isa-vmtf-init* $\mathcal{B}$ *M*›
⟨*proof*⟩

type-synonym (**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 𝒜 unbdd =
  {((M', N', D', j, W', vm, φ, clvls, cach, lbd, vdom, failed), ((M, N, D, NE, UE, NS, US, 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 + NS + US)) ⊆ set-mset (ℒ_all 𝒜) ∧
       mset vdom = dom-m N)) ∧
    (M', M) ∈ trail-pol 𝒜 ∧
    (D',  D) ∈ option-lookup-clause-rel 𝒜 ∧
    j ≤ length M ∧
    Q = uminus ‘# lit-of ‘# mset (drop j (rev M)) ∧
    vm ∈ isa-vmtf-init 𝒜 M ∧
    phase-saving 𝒜 φ ∧
    no-dup M ∧
    cach-refinement-empty 𝒜 cach ∧
    (W', empty-watched 𝒜) ∈ ⟨Id⟩map-fun-rel (D₀ 𝒜) ∧
    isasat-input-bounded 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 𝒜  unbdd =
  {((M', N', D', j, W', vm, φ, clvls, cach, lbd, vdom, failed), ((M, N, D, NE, UE, NS, US, Q, W),
OC)).
    (unbdd ⟶ ¬failed) ∧
    ((unbdd ∨ ¬failed) ⟶
    ((M', M) ∈ trail-pol 𝒜 ∧
    valid-arena N' N (set vdom) ∧
    (D',  D) ∈ option-lookup-clause-rel 𝒜 ∧
    j ≤ length M ∧
    Q = uminus ‘# lit-of ‘# mset (drop j (rev M)) ∧
    vm ∈ isa-vmtf-init 𝒜 M ∧
    phase-saving 𝒜 φ ∧
    no-dup M ∧
    cach-refinement-empty 𝒜 cach ∧
    mset vdom = dom-m N ∧
    vdom-m 𝒜 W N = set-mset (dom-m N) ∧
    set-mset
     (all-lits-of-mm
       ({#mset (fst x). x ∈# ran-m N#} + NE + UE + NS + US)) ⊆ set-mset (ℒ_all 𝒜) ∧
```

(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 𝒜) ∧
isasat-input-bounded 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* 𝒜 *unbdd* =
{((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*), (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena N′ N* (*set vdom*) ∧ *set-mset* (*dom-m N*) ⊆ *set vdom*)) ∧
(*M′*, *M*) ∈ *trail-pol* 𝒜 ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* 𝒜 ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *isa-vmtf-init* 𝒜 *M* ∧
phase-saving 𝒜 φ ∧
no-dup M ∧
cach-refinement-empty 𝒜 *cach* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *UE* + *NS* + *US*))
⊆ *set-mset* ($\mathcal{L}_{all}$ 𝒜) ∧
(*W′*, *empty-watched* 𝒜) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 𝒜) ∧
isasat-input-bounded 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* 𝒜 *unbdd* =
{((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*), ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena N′ N* (*set vdom*) ∧ *set-mset* (*dom-m N*) ⊆ *set vdom*)) ∧ (*M′*, *M*) ∈ *trail-pol* 𝒜 ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* 𝒜 ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *isa-vmtf-init* 𝒜 *M* ∧
phase-saving 𝒜 φ ∧
no-dup M ∧
cach-refinement-empty 𝒜 *cach* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *UE* + *NS* + *US*))
⊆ *set-mset* ($\mathcal{L}_{all}$ 𝒜) ∧
(*W′*, *empty-watched* 𝒜) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 𝒜) ∧
isasat-input-bounded 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*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*), (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
((*M′*, *M*) ∈ *trail-pol* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) ∧
set-mset (*dom-m N*) ⊆ *set vdom* ∧
valid-arena N′ N (*set vdom*))) ∧

```
    (D', D) ∈ option-lookup-clause-rel (all-atms N (NE + UE + NS + US)) ∧
    j ≤ length M ∧
    Q = uminus ‘# lit-of ‘# mset (drop j (rev M)) ∧
    vm ∈ isa-vmtf-init (all-atms N (NE + UE + NS + US)) M ∧
    phase-saving (all-atms N (NE + UE + NS + US)) φ ∧
    no-dup M ∧
    cach-refinement-empty (all-atms N (NE + UE + NS + US)) cach ∧
    vdom-m (all-atms N (NE + UE + NS + US)) W N ⊆ set vdom ∧
    set-mset (all-lits-of-mm ({#mset (fst x). x ∈# ran-m N#} + NE + UE + NS + US))
      ⊆ set-mset (ℒ_all (all-atms N (NE + UE + NS + US))) ∧
    (W', W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-atms N (NE + UE + NS + US))) ∧
    isasat-input-bounded (all-atms N (NE + UE + NS + US)) ∧
    distinct vdom
  }›
```

VMTF

```
definition initialise-VMTF :: ‹nat list ⇒ nat ⇒ isa-vmtf-remove-int-option-fst-As nres› where
‹initialise-VMTF N n = do {
   let A = replicate n (VMTF-Node 0 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 = (N ! i);
        ASSERT(L < length A);
        ASSERT(cnext ≠ None ⟶ the cnext < length A);
        ASSERT(i + 1 ≤ uint32-max);
        RETURN (i + 1, vmtf-cons A L cnext (i), Some L)
      })
      (0, A, None);
   RETURN ((A, n, cnext, (if N = [] then None else Some ((N!0))), cnext), to-remove)
  }›
```

```
lemma initialise-VMTF:
  shows ‹(uncurry initialise-VMTF, uncurry (λN n. RES (vmtf-init N []))) ∈
      [λ(N,n). (∀ L∈# N. L < n) ∧ (distinct-mset N) ∧ size N < uint32-max ∧ set-mset N = set-mset
𝒜]_f
      (⟨nat-rel⟩list-rel-mset-rel) ×_f nat-rel →
      ⟨((⟨Id⟩list-rel ×_r nat-rel ×_r ⟨nat-rel⟩ option-rel ×_r ⟨nat-rel⟩ option-rel ×_r ⟨nat-rel⟩ option-rel)
        ×_r distinct-atoms-rel 𝒜⟩nres-rel›
    (is ‹(?init, ?R) ∈ -›)
⟨proof⟩
```

15.1.2 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* {
M ← *cons-trail-Propagated-tr L 0 M*;
RETURN (*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*⟩

fun *get-subsumed-clauses-init-wl* :: ⟨′*v twl-st-wl-init* ⇒ ′*v clauses*⟩ **where**
⟨*get-subsumed-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*) = *NS* + *US*⟩

fun *get-subsumed-init-clauses-init-wl* :: ⟨′*v twl-st-wl-init* ⇒ ′*v clauses*⟩ **where**
⟨*get-subsumed-init-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*) = *NS*⟩

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* + *get-subsumed-init-clauses-init-wl S*)⟩

definition *all-atms-init* :: ⟨- ⇒ - ⇒ ′*v multiset*⟩ **where**
⟨*all-atms-init N NUE* = *atm-of* ‘# *all-lits N NUE*⟩

abbreviation *all-atms-st-init* :: ⟨′*v twl-st-wl-init* ⇒ ′*v multiset*⟩ **where**
⟨*all-atms-st-init S* ≡ *atm-of* ‘# *all-lits-st-init S*⟩

lemma *DECISION-REASON0*[*simp*]: ⟨*DECISION-REASON* ≠ *0*⟩
⟨*proof*⟩

lemma *propagate-unit-cls-heur-propagate-unit-cls*:
⟨(*uncurry propagate-unit-cls-heur*, *uncurry* (*propagate-unit-init-wl*)) ∈
[λ(*L*, *S*). *undefined-lit* (*get-trail-init-wl S*) *L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*⟩
⟨*proof*⟩

definition *already-propagated-unit-cls*
:: ⟨*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*⟩
where
⟨*already-propagated-unit-cls* = (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*›
⟨*proof*⟩

definition (**in** −) *set-conflict-unit* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-unit L - = Some* {#*L*#}›

definition *set-conflict-unit-heur* **where**
‹*set-conflict-unit-heur* = (λ *L* (*b*, *n*, *xs*). *RETURN* (*False*, *1*, *xs*[*atm-of L* := *Some* (*is-pos L*)]))›

lemma *set-conflict-unit-heur-set-conflict-unit*:
‹(*uncurry set-conflict-unit-heur*, *uncurry* (*RETURN oo set-conflict-unit*)) ∈
[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*›
⟨*proof*⟩

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*, *NS*, *US*, *Q*), *OC*).
((*M*, *N*, *set-conflict-unit L D*, *add-mset* {#*L*#} *NE*, *UE*, *NS*, *US*, {#}), *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 = (λL (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*⟩

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 ≤ uint32-max + 2);
    ASSERT(length C ≥ 2);
    if unbdd ∨ (length N ≤ sint64-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 @ [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*⟩

definition *add-init-cls-heur-b'* :: ⟨*nat literal list list* ⇒ *nat* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩ **where**
⟨*add-init-cls-heur-b' C i* = *add-init-cls-heur False* (*C*!*i*)⟩

lemma *length-C-nempty-iff*: ⟨*length C* ≥ *2* ⟷ *C* ≠ [] ∧ *tl C* ≠ []⟩
  ⟨*proof*⟩

context
  fixes *unbdd* :: *bool* **and** $\mathcal{A}$ :: ⟨*nat multiset*⟩ **and**
    CT :: ⟨*nat clause-l* × *twl-st-wl-heur-init*⟩ **and**
    CSOC :: ⟨*nat clause-l* × *nat twl-st-wl-init*⟩ **and**
    SOC :: ⟨*nat twl-st-wl-init*⟩ **and**
    C C' :: ⟨*nat clause-l*⟩ **and**
    S :: ⟨*nat twl-st-wl-init'*⟩ **and** *x1a* **and** *N* :: ⟨*nat clauses-l*⟩ **and**
    D :: ⟨*nat cconflict*⟩ **and** *x2b* **and** *NE UE NS US* :: ⟨*nat clauses*⟩ **and**
    M :: ⟨(*nat*,*nat*) *ann-lits*⟩ **and**
    a b c d e f m p q r s t u v w x y **and**
    Q **and**
    x2e :: ⟨*nat lit-queue-wl*⟩ **and** *OC* :: ⟨*nat clauses*⟩ **and**
    T :: *twl-st-wl-heur-init* **and**
    M' :: ⟨*trail-pol*⟩ **and** *N'* :: *arena* **and**
    D' :: *conflict-option-rel* **and**
    j' :: *nat* **and**
    W' :: ⟨-⟩ **and**
    vm :: ⟨*isa-vmtf-remove-int-option-fst-As*⟩ **and**
    clvls :: *nat* **and**
    cach :: *conflict-min-cach-l* **and**
    lbd :: *lbd* **and**
    vdom :: *vdom* **and**
    failed :: *bool* **and**
    φ :: *phase-saver*
  assumes
    pre: ⟨*case CSOC of*
    (*C*, *S*) ⇒ *2* ≤ *length C* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*⟩ **and**
    xy: ⟨(*CT*, *CSOC*) ∈ *Id* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ **and**
    st:
      ⟨*CSOC* = (*C*, *SOC*)⟩
      ⟨*SOC* = (*S*, *OC*)⟩
      ⟨*S* = (*M*, *a*)⟩
      ⟨*a* = (*N*, *b*)⟩
      ⟨*b* = (*D*, *c*)⟩
      ⟨*c* = (*NE*, *d*)⟩
      ⟨*d* = (*UE*, *e*)⟩
      ⟨*e* = (*NS*, *f*)⟩
      ⟨*f* = (*US*, *Q*)⟩

‹*CT* = (*C′*, *T*)›
‹*T* = (*M′*, *m*)›
‹*m* = (*N′*, *p*)›
‹*p* = (*D′*, *q*)›
‹*q* = (*j′*, *r*)›
‹*r* = (*W′*, *s*)›
‹*s* = (*vm*, *t*)›
‹*t* = (*φ*, *u*)›
‹*u* = (*clvls*, *v*)›
‹*v* = (*cach*, *w*)›
‹*w* = (*lbd*, *x*)›
‹*x* = (*vdom*, *failed*)›
begin

lemma *add-init-pre1*: ‹*length C′* ≤ *uint32-max* + *2*›
⟨*proof*⟩

lemma *add-init-pre2*: ‹*2* ≤ *length C′*›
⟨*proof*⟩ **lemma**
x1g-x1: ‹*C′* = *C*› **and**
‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
valid: ‹*valid-arena N′ N* (*set vdom*)› **and**
‹(*D′*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
‹*j′* ≤ *length M*› **and**
Q: ‹*Q* = {#− *lit-of x*. *x* ∈# *mset* (*drop j′* (*rev M*))#}› **and**
‹*vm* ∈ *isa-vmtf-init* $\mathcal{A}$ *M*› **and**
‹*phase-saving* $\mathcal{A}$ *φ*› **and**
‹*no-dup M*› **and**
‹*cach-refinement-empty* $\mathcal{A}$ *cach*› **and**
vdom: ‹*mset vdom* = *dom-m N*› **and**
var-incl:
‹*set-mset* (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *NS* + *UE* + *US*))
⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
watched: ‹(*W′*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
if ‹¬*failed* ∨ *unbdd*›
⟨*proof*⟩

lemma *init-fm-add-new*:
‹¬*failed* ∨ *unbdd* ⟹ *fm-add-new True C′ N′*
≤ ⇓ {((*arena*, *i*), (*N″*, *i′*)). *valid-arena arena N″* (*insert i* (*set vdom*)) ∧ *i* = *i′* ∧
i ∉# *dom-m N* ∧ *i* = *length N′* + *header-size C* ∧
i ∉ *set vdom*}
(*SPEC*
(λ(*N′*, *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ *N′* = *fmupd ia* (*C*, *True*) *N*))›
(**is** ‹- ⟹ - ≤ ⇓ *?qq* -›)
⟨*proof*⟩

lemma *add-init-cls-final-rel*:
fixes *nN′j′* :: ‹*arena-el list* × *nat*› **and**
nNj :: ‹(*nat*, *nat literal list* × *bool*) *fmap* × *nat*› **and**
nN :: ‹-› **and**
k :: ‹*nat*› **and** *nN′* :: ‹*arena-el list*› **and**
k′ :: ‹*nat*›
assumes

‹(*nN'j'*, *nNj*) ∈ {((*arena*, *i*), (*N''*, *i'*)). *valid-arena arena N''* (*insert i* (*set vdom*)) ∧ *i* = *i'* ∧
i ∉# *dom-m N* ∧ *i* = *length N'* + *header-size C* ∧
i ∉ *set vdom*}› **and**
‹*nNj* ∈ *Collect* (λ(*N'*, *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*C*, *True*) *N*)›
‹*nN'j'* = (*nN'*, *k'*)› **and**
‹*nNj* = (*nN*, *k*)›
shows ‹((*M'*, *nN'*, *D'*, *j'*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom* @ [*k'*], *failed*),
(*M*, *nN*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)
∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
⟨*proof*⟩
end

lemma *add-init-cls-heur-add-init-cls*:
‹(*uncurry* (*add-init-cls-heur unbdd*), *uncurry* (*add-to-clauses-init-wl*)) ∈
[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*⟩

definition *already-propagated-unit-cls-conflict*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls-conflict* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*).
((*M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *NS*, *US*, {#}), *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* = (λ*L* (*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*›
⟨*proof*⟩

definition (**in** −) *set-conflict-empty* :: ‹*nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-empty* - = *Some* {#}›

definition (**in** −) *lookup-set-conflict-empty* :: ‹*conflict-option-rel* ⇒ *conflict-option-rel*› **where**
‹*lookup-set-conflict-empty* = (λ(*b*, *s*) . (*False*, *s*))›

lemma *lookup-set-conflict-empty-set-conflict-empty*:
‹(*RETURN o lookup-set-conflict-empty*, *RETURN o set-conflict-empty*) ∈
[λ*D*. *D* = *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

definition *set-empty-clause-as-conflict-heur*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*set-empty-clause-as-conflict-heur* = (λ (*M*, *N*, (-, (*n*, *xs*)), *Q*, *WS*). *do* {

```
    ASSERT(isa-length-trail-pre M);
    RETURN (M, N, (False, (n, xs)), isa-length-trail M, WS)})›
```

```
lemma set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict:
  ‹(set-empty-clause-as-conflict-heur, RETURN o add-empty-conflict-init-wl) ∈
  [λS. get-conflict-init-wl S = None]f
   twl-st-heur-parsing-no-WL 𝒜 unbdd → ⟨twl-st-heur-parsing-no-WL 𝒜 unbdd⟩ nres-rel›
  ⟨proof⟩
```

```
definition (in −) add-clause-to-others-heur
   :: ‹nat clause-l ⇒ twl-st-wl-heur-init ⇒ twl-st-wl-heur-init nres› where
  ‹add-clause-to-others-heur = (λ - (M, N, D, Q, NS, US, WS).
      RETURN (M, N, D, Q, NS, US, 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 ×r twl-st-heur-parsing-no-WL 𝒜 unbdd →f ⟨twl-st-heur-parsing-no-WL 𝒜 unbdd⟩ nres-rel›
  ⟨proof⟩
```

```
definition (in −)list-length-1 where
  [simp]: ‹list-length-1 C ⟷ length C = 1›
```

```
definition (in −)list-length-1-code where
  ‹list-length-1-code C ⟷ (case C of [-] ⇒ True | - ⇒ False)›
```

```
definition (in −) get-conflict-wl-is-None-heur-init :: ‹twl-st-wl-heur-init ⇒ bool› where
  ‹get-conflict-wl-is-None-heur-init = (λ(M, N, (b, -), Q, -). b)›
```

```
definition init-dt-step-wl-heur
  :: ‹bool ⇒ nat clause-l ⇒ twl-st-wl-heur-init ⇒ (twl-st-wl-heur-init) nres›
where
  ‹init-dt-step-wl-heur unbdd C S = do {
    if get-conflict-wl-is-None-heur-init S
    then do {
       if is-Nil C
       then set-empty-clause-as-conflict-heur S
       else if list-length-1 C
       then do {
         ASSERT (C ≠ []);
         let L = C ! 0;
         ASSERT(polarity-pol-pre (get-trail-wl-heur-init S) L);
         let val-L = polarity-pol (get-trail-wl-heur-init S) L;
         if val-L = None
         then propagate-unit-cls-heur L S
         else
           if val-L = Some True
           then already-propagated-unit-cls-heur C S
           else conflict-propagated-unit-cls-heur L S
       }
       else do {
         ASSERT(length C ≥ 2);
         add-init-cls-heur unbdd C S
```

      }
    }
    else add-clause-to-others-heur C S
  }›

named-theorems *twl-st-heur-parsing-no-WL*
lemma [*twl-st-heur-parsing-no-WL*]:
  assumes ‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
  shows ‹(*get-trail-wl-heur-init S*, *get-trail-init-wl T*) ∈ *trail-pol* $\mathcal{A}$›
  ⟨*proof*⟩

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

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

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*:
    ‹(*RETURN o get-conflict-wl-is-None-heur-init*, *RETURN o get-conflict-wl-is-None-init*) ∈
    twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
  ⟨*proof*⟩

definition (**in** −) *get-conflict-wl-is-None-init′* **where**
  ‹*get-conflict-wl-is-None-init′* = *get-conflict-wl-is-None*›

lemma *init-dt-step-wl-heur-init-dt-step-wl*:
  ‹(*uncurry* (*init-dt-step-wl-heur unbdd*), *uncurry init-dt-step-wl*) ∈
    [λ(*C*, *S*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*]$_f$
      Id $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
  ⟨*proof*⟩

lemma (**in** −) *get-conflict-wl-is-None-heur-init-alt-def*:
  ‹*RETURN o get-conflict-wl-is-None-heur-init* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)›
  ⟨*proof*⟩

definition *polarity-st-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ - ⇒ *bool option*› **where**
  ‹*polarity-st-heur-init* = (λ(*M*, -) *L*. *polarity-pol M L*)›

lemma *polarity-st-heur-init-alt-def*:
  ‹*polarity-st-heur-init S L* = *polarity-pol* (*get-trail-wl-heur-init S*) *L*›
  ⟨*proof*⟩

definition *polarity-st-init* :: ‹′*v twl-st-wl-init* ⇒ ′*v literal* ⇒ *bool option*› **where**
  ‹*polarity-st-init S* = *polarity* (*get-trail-init-wl S*)›

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

definition *init-dt-wl-heur*
 :: ‹*bool* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where

‹*init-dt-wl-heur unbdd CS S = nfoldli CS* (λ-. *True*)
(λ*C S*. *do* {
init-dt-step-wl-heur unbdd C S}) *S*›

definition *init-dt-step-wl-heur-unb* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-step-wl-heur-unb* = *init-dt-step-wl-heur True*›

definition *init-dt-wl-heur-unb* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-unb* = *init-dt-wl-heur True*›

definition *init-dt-step-wl-heur-b* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-step-wl-heur-b* = *init-dt-step-wl-heur False*›

definition *init-dt-wl-heur-b* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*init-dt-wl-heur-b* = *init-dt-wl-heur False*›

15.1.3 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 < *uint32-max* ∧
(∀ *s*∈*set xs*. *s* ≤ *uint64-max*) ∧
finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0*
}›

lemma *distinct-length-le-Suc-Max*:
assumes ‹*distinct* (*b* :: *nat list*)›
shows ‹*length b* ≤ *Suc* (*Max* (*insert 0* (*set b*)))›
⟨*proof*⟩

lemma *isasat-atms-ext-rel-alt-def*:
‹*isasat-atms-ext-rel* = {((*xs*, *n*, *atms*), *l*).
init-valid-rep xs l ∧
n = *Max* (*insert 0 l*) ∧
length xs < *uint32-max* ∧
(∀ *s*∈*set xs*. *s* ≤ *uint64-max*) ∧
finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0* ∧
length atms ≤ *Suc n*

```
  }›
  ⟨proof⟩


definition in-map-atm-of :: ‹'a ⇒ 'a list ⇒ bool› where
  ‹in-map-atm-of L N ⟷ L ∈ set N›

definition (in −) init-next-size where
  ‹init-next-size L = 2 ∗ L›

lemma init-next-size: ‹L ≠ 0 ⟹ L + 1 ≤ uint32-max ⟹ L < init-next-size L›
  ⟨proof⟩

definition add-to-atms-ext where
  ‹add-to-atms-ext = (λi (xs, n, atms). do {
    ASSERT(i ≤ uint32-max div 2);
    ASSERT(length xs ≤ uint32-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 1 = 1 then atms else atms @ [i]);
       RETURN (xs[i := 1], n, atms)
     }
     else do {
        ASSERT(i + 1 ≤ uint32-max);
        ASSERT(length-uint32-nat xs ≠ 0);
        ASSERT(i < init-next-size i);
        RETURN ((list-grow xs (init-next-size i) 0)[i := 1], n,
            atms @ [i])
     })
    })›

lemma init-valid-rep-upd-OR:
  ‹init-valid-rep (x1b[x1a := a OR 1]) x2 ⟷
    init-valid-rep (x1b[x1a := 1]) x2 › (is ‹?A ⟷ ?B›)
⟨proof⟩

lemma init-valid-rep-insert:
  assumes val: ‹init-valid-rep x1b x2› and le: ‹x1a < length x1b›
  shows ‹init-valid-rep (x1b[x1a := Suc 0]) (insert x1a x2)›
⟨proof⟩

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⟩

lemma init-valid-rep-in-set-iff:
  ‹init-valid-rep x1b x2  ⟹ x ∈ x2 ⟷ (x < length x1b ∧ (x1b!x) mod 2 = 1)›
  ⟨proof⟩

lemma add-to-atms-ext-op-set-insert:
  ‹(uncurry add-to-atms-ext, uncurry (RETURN oo Set.insert))
   ∈ [λ(n, l). n ≤ uint32-max div 2]_f nat-rel ×_f isasat-atms-ext-rel → ⟨isasat-atms-ext-rel⟩nres-rel›
⟨proof⟩
```

definition *extract-atms-cls* :: ‹*'a clause-l* ⇒ *'a set* ⇒ *'a set*› **where**
‹*extract-atms-cls C* $\mathcal{A}_{in}$ = *fold* (λ*L* $\mathcal{A}_{in}$. *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*)
  (λ*L* $\mathcal{A}_{in}$. *do* {
    ASSERT(*atm-of L* ≤ *uint32-max div 2*);
    RETURN(*insert* (*atm-of L*) $\mathcal{A}_{in}$)})
 $\mathcal{A}_{in}$›

lemma *fild-insert-insert-swap*:
‹*fold* (λ*L*. *insert* (*f L*)) *C* (*insert a* $\mathcal{A}_{in}$) = *insert a* (*fold* (λ*L*. *insert* (*f L*)) *C* $\mathcal{A}_{in}$)›
⟨*proof*⟩

lemma *extract-atms-cls-alt-def*: ‹*extract-atms-cls C* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ *atm-of* ‘ *set C*›
⟨*proof*⟩

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* ≤ *uint32-max*]$_f$
  ⟨*Id*⟩*list-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

definition *extract-atms-clss*:: ‹*'a clause-l list* ⇒ *'a set* ⇒ *'a set*› **where**
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = *fold extract-atms-cls N* $\mathcal{A}_{in}$›

definition *extract-atms-clss-i* :: ‹*nat clause-l list* ⇒ *nat set* ⇒ *nat set nres*› **where**
‹*extract-atms-clss-i N* $\mathcal{A}_{in}$ = *nfoldli N* (λ-. *True*) *extract-atms-cls-i* $\mathcal{A}_{in}$›

lemma *extract-atms-clss-i-extract-atms-clss*:
‹(*uncurry extract-atms-clss-i*, *uncurry* (*RETURN oo extract-atms-clss*))
 ∈ [λ(*N*, $\mathcal{A}_{in}$). ∀ *C*∈*set N*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*]$_f$
  ⟨*Id*⟩*list-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *fold-extract-atms-cls-union-swap*:
‹*fold extract-atms-cls N* ($\mathcal{A}_{in}$ ∪ *a*) = *fold extract-atms-cls N* $\mathcal{A}_{in}$ ∪ *a*›
⟨*proof*⟩

lemma *extract-atms-clss-alt-def*:
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ ((⋃ *C*∈*set N*. *atm-of* ‘ *set C*))›
⟨*proof*⟩

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss CS'* {})› **for** *CS'*
⟨*proof*⟩

definition *op-extract-list-empty* **where**
‹*op-extract-list-empty* = {}›

definition *extract-atms-clss-imp-empty-rel* **where**
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*replicate 1024 0*, *0*, []))›

lemma *extract-atms-clss-imp-empty-rel*:
‹(λ-. *extract-atms-clss-imp-empty-rel*, λ-. (*RETURN op-extract-list-empty*)) ∈
unit-rel $\rightarrow_f$ ⟨*isasat-atms-ext-rel*⟩ *nres-rel*›
⟨*proof*⟩

lemma *extract-atms-cls-Nil*[*simp*]:
‹*extract-atms-cls* [] $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$›
⟨*proof*⟩

lemma *extract-atms-clss-Cons*[*simp*]:
‹*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)›
⟨*proof*⟩

definition (**in** −) *all-lits-of-atms-m* :: ‹*'a multiset* ⇒ *'a clause*› **where**
‹*all-lits-of-atms-m N* = *poss N* + *negs N*›

lemma (**in** −) *all-lits-of-atms-m-nil*[*simp*]: ‹*all-lits-of-atms-m* {#} = {#}›
⟨*proof*⟩

definition (**in** −) *all-lits-of-atms-mm* :: ‹*'a multiset multiset* ⇒ *'a clause*› **where**
‹*all-lits-of-atms-mm N* = *poss* (⋃# *N*) + *negs* (⋃# *N*)›

lemma *all-lits-of-atms-m-all-lits-of-m*:
‹*all-lits-of-atms-m N* = *all-lits-of-m* (*poss N*)›
⟨*proof*⟩

Creation of an initial state

definition *init-dt-wl-heur-spec*
:: ‹*bool* ⇒ *nat multiset* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init* ⇒ *bool*›
where
‹*init-dt-wl-heur-spec unbdd* $\mathcal{A}$ *CS T TOC* ⟷
(∃ *T' TOC'*. (*TOC*, *TOC'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧ (*T*, *T'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧
init-dt-wl-spec CS T' TOC')›

definition *init-state-wl* :: ‹*nat twl-st-wl-init'*› **where**
‹*init-state-wl* = ([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#})›

definition *init-state-wl-heur* :: ‹*nat multiset* ⇒ *twl-st-wl-heur-init nres*› **where**

```
‹init-state-wl-heur 𝒜 = do {
  M ← SPEC(λM. (M, []) ∈ trail-pol 𝒜);
  D ← SPEC(λD. (D, None) ∈ option-lookup-clause-rel 𝒜);
  W ← SPEC (λW. (W, empty-watched 𝒜) ∈ ⟨Id⟩map-fun-rel (D₀ 𝒜));
  vm ← RES (isa-vmtf-init 𝒜 []);
  φ ← SPEC (phase-saving 𝒜);
  cach ← SPEC (cach-refinement-empty 𝒜);
  let lbd = empty-lbd;
  let vdom = [];
  RETURN (M, [], D, 0, W, vm, φ, 0, cach, lbd, vdom, False)}›
```

definition *init-state-wl-heur-fast* **where**
‹*init-state-wl-heur-fast* = *init-state-wl-heur*›

lemma *init-state-wl-heur-init-state-wl*:
‹(λ-. (*init-state-wl-heur* $\mathcal{A}$), λ-. (*RETURN init-state-wl*)) $\in$
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$ *unit-rel* $\rightarrow$ ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −)*to-init-state* :: ‹*nat twl-st-wl-init′* $\Rightarrow$ *nat twl-st-wl-init*› **where**
‹*to-init-state S* = (*S*, {#})›

definition (**in** −) *from-init-state* :: ‹*nat twl-st-wl-init-full* $\Rightarrow$ *nat twl-st-wl*› **where**
‹*from-init-state* = *fst*›

definition (**in** −) *to-init-state-code* **where**
‹*to-init-state-code* = *id*›

definition *from-init-state-code* **where**
‹*from-init-state-code* = *id*›

definition (**in** −) *conflict-is-None-heur-wl* **where**
‹*conflict-is-None-heur-wl* = (λ(*M*, *N*, *U*, *D*, -). *is-None D*)›

definition (**in** −) *finalise-init* **where**
‹*finalise-init* = *id*›

15.1.4 Parsing

lemma *init-dt-wl-heur-init-dt-wl*:
‹(*uncurry* (*init-dt-wl-heur unbdd*), *uncurry init-dt-wl*) $\in$
[λ(*CS*, *S*). ($\forall$ *C* $\in$ *set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)) $\wedge$ *distinct-mset-set* (*mset* ‘ *set CS*)]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
⟨*proof*⟩

definition *rewatch-heur-st*
:: ‹*twl-st-wl-heur-init* $\Rightarrow$ *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* $\leq$ *length N′*);
W $\leftarrow$ *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*) $\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*› **and** *failed*: ‹$\neg$*is-failed-heur-init S*›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* (*get-clauses-init-wl T*))› **and**
‹$\bigwedge$*x*. *x* $\in$# *dom-m* (*get-clauses-init-wl T*) $\Longrightarrow$ *distinct* (*get-clauses-init-wl T* $\propto$ *x*) $\wedge$
2 $\leq$ *length* (*get-clauses-init-wl T* $\propto$ *x*)›
shows ‹*rewatch-heur-st S* $\leq$ $\Downarrow$ (*twl-st-heur-parsing* $\mathcal{A}$ *unbdd*)
(*SPEC* (λ((*M*,*N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*), *OC*). *T* = ((*M*,*N*,*D*,*NE*,*UE*,*NS*, *US*, *Q*), *OC*)$\wedge$
correct-watching (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)))›
⟨*proof*⟩

Full Initialisation

definition *rewatch-heur-st-fast* **where**
⟨*rewatch-heur-st-fast* = *rewatch-heur-st*⟩

definition *rewatch-heur-st-fast-pre* **where**
⟨*rewatch-heur-st-fast-pre S* =
((∀ *x* ∈ *set* (*get-vdom-heur-init S*). *x* ≤ *sint64-max*) ∧ *length* (*get-clauses-wl-heur-init S*) ≤ *sint64-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*⟩

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*))⟩
⟨*proof*⟩

15.1.5 Conversion to normal state

definition *extract-lits-sorted* **where**
⟨*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {
vars ← — insert_sort_nth2 xs vars*RETURN vars*;
RETURN (*vars*, *n*)
})⟩

definition *lits-with-max-rel* **where**
⟨*lits-with-max-rel* = {((*xs*, *n*), $\mathcal{A}_{in}$). *mset xs* = $\mathcal{A}_{in}$ ∧ *n* = *Max* (*insert 0* (*set xs*)) ∧

length xs < *uint32-max*}›

lemma *extract-lits-sorted-mset-set*:
‹(*extract-lits-sorted*, *RETURN o mset-set*)
∈ *isasat-atms-ext-rel* $\rightarrow_f$ ⟨*lits-with-max-rel*⟩*nres-rel*›
⟨*proof*⟩

TODO Move

The value 160 is random (but larger than the default 16 for array lists).

definition *finalise-init-code* :: ‹*opts* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur nres*› **where**
‹*finalise-init-code opts* =
(λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), *φ*, *clvls*, *cach*,
lbd, *vdom*, -). *do* {
ASSERT(*lst-As* ≠ *None* ∧ *fst-As* ≠ *None*);
let init-stats = (*0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *ema-fast-init*);
let fema = *ema-fast-init*;
let sema = *ema-slow-init*;
let ccount = *restart-info-init*;
let lcount = *0*;
RETURN (*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*), *to-remove*),
clvls, *cach*, *lbd*, *take 1*(*replicate 160* (*Pos 0*)), *init-stats*,
(*fema*, *sema*, *ccount*, *0*, *φ*, *0*, *replicate* (*length φ*) *False*, *0*, *replicate* (*length φ*) *False*, *10000*, *1000*, *1*), *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*›
⟨*proof*⟩

lemma *isa-vmtf-init-isa-vmtf*: ‹𝒜 ≠ {#} ⟹ ((*ak*, *al*, *Some am*, *Some an*, *bc*), *ao*, *bd*)
∈ *isa-vmtf-init* 𝒜 *au* ⟹ ((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*)
∈ *isa-vmtf* 𝒜 *au*›
⟨*proof*⟩

lemma *heuristic-rel-initI*:
‹*phase-saving* 𝒜 *φ* ⟹ *length φ'* = *length φ* ⟹ *length φ''* = *length φ* ⟹ *heuristic-rel* 𝒜 (*fema*, *sema*, *ccount*, *0*, (*φ*,*a*, *φ'*,*b*,*φ''*,*c*,*d*))›
⟨*proof*⟩

lemma *finalise-init-finalise-init-full*:
‹*get-conflict-wl S* = *None* ⟹
all-atms-st S ≠ {#} ⟹ *size* (*learned-clss-l* (*get-clauses-wl S*)) = *0* ⟹
((*ops'*, *T*), *ops*, *S*) ∈ *Id* $\times_f$ *twl-st-heur-post-parsing-wl True* ⟹
finalise-init-code ops' T ≤ ⇓ {(*S'*, *T'*). (*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init T = *get-clauses-wl-heur S'*} (*RETURN* (*finalise-init S*))›
⟨*proof*⟩

lemma *finalise-init-finalise-init*:
‹(*uncurry finalise-init-code*, *uncurry* (*RETURN oo* (λ-. *finalise-init*))) ∈
[λ(-, *S*::*nat twl-st-wl*). *get-conflict-wl S* = *None* ∧ *all-atms-st S* ≠ {#} ∧
size (*learned-clss-l* (*get-clauses-wl S*)) = *0*]$_f$ *Id* $\times_r$
twl-st-heur-post-parsing-wl True → ⟨*twl-st-heur*⟩*nres-rel*›

⟨*proof*⟩

definition (**in** −) *init-rll* :: ‹*nat* ⇒ (*nat*, ′*v clause-l* × *bool*) *fmap*› **where**
‹*init-rll n* = *fmempty*›

definition (**in** −) *init-aa* :: ‹*nat* ⇒ ′*v list*› **where**
‹*init-aa n* = []›

definition (**in** −) *init-aa′* :: ‹*nat* ⇒ (*clause-status* × *nat* × *nat*) *list*› **where**
‹*init-aa′ n* = []›

definition *init-trail-D* :: ‹*nat 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 0*;
    let M″ = *replicate n 1*;
    RETURN ((*M0*, *M*, *M′*, *M″*, *0*, *cs*))
}›

definition *init-trail-D-fast* **where**
‹*init-trail-D-fast* = *init-trail-D*›

definition *init-state-wl-D′* :: ‹*nat list* × *nat* ⇒ (*trail-pol* × - × -) *nres*› **where**
‹*init-state-wl-D′* = (λ($\mathcal{A}_{in}$, *n*). *do* {
    ASSERT(*Suc* (*2* ∗ (*n*)) ≤ *uint32-max*);
    let n = *Suc* (*n*);
    let m = *2* ∗ *n*;
    M ← *init-trail-D* $\mathcal{A}_{in}$ *n m*;
    let N = [];
    let D = (*True*, *0*, *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*, *0*, *WS*, *vm*, φ, *0*, *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* $\mathcal{A}_{in}$]$_f$
  ⟨*Id*⟩*list-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* →
⟨*trail-pol* $\mathcal{A}_{in}$⟩ *nres-rel*›
⟨*proof*⟩

definition [*to-relAPP*]: ‹*mset-rel A* ≡ *p2rel* (*rel-mset* (*rel2p A*))›
lemma *in-mset-rel-eq-f-iff*:
‹(*a*, *b*) ∈ ⟨{(*c*, *a*). *a* = *f c*}⟩*mset-rel* ⟷ *b* = *f* ‘# *a*›
⟨*proof*⟩

lemma *in-mset-rel-eq-f-iff-set*:
$\langle\langle\{(c,\ a).\ a = f\ c\}\rangle\mathit{mset\text{-}rel} = \{(b,\ a).\ a = f\ \text{‘}\#\ b\}\rangle$
$\langle\mathit{proof}\rangle$

lemma *init-state-wl-D0*:
$\langle(\mathit{init\text{-}state\text{-}wl\text{-}D}',\ \mathit{init\text{-}state\text{-}wl\text{-}heur}) \in$
$[\lambda N.\ N = \mathcal{A}_{in} \wedge \mathit{distinct\text{-}mset}\ \mathcal{A}_{in} \wedge \mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}_{in}]_f$
$\mathit{lits\text{-}with\text{-}max\text{-}rel}\ O\ \langle Id\rangle\mathit{mset\text{-}rel} \rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r \mathit{nat\text{-}rel} \times_r \langle\langle Id\rangle\mathit{list\text{-}rel}\rangle\mathit{list\text{-}rel} \times_r$
$Id \times_r \langle\mathit{bool\text{-}rel}\rangle\mathit{list\text{-}rel} \times_r Id \times_r Id \times_r Id\rangle\mathit{nres\text{-}rel}\rangle$
(**is** $\langle ?C \in [?Pre]_f\ \ ?arg \rightarrow \langle ?im\rangle\mathit{nres\text{-}rel}\rangle$)
$\langle\mathit{proof}\rangle$

lemma *init-state-wl-D'*:
$\langle(\mathit{init\text{-}state\text{-}wl\text{-}D}',\ \mathit{init\text{-}state\text{-}wl\text{-}heur}) \in$
$[\lambda\mathcal{A}_{in}.\ \mathit{distinct\text{-}mset}\ \mathcal{A}_{in} \wedge \mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}_{in}]_f$
$\mathit{lits\text{-}with\text{-}max\text{-}rel}\ O\ \langle Id\rangle\mathit{mset\text{-}rel} \rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r \mathit{nat\text{-}rel} \times_r \langle\langle Id\rangle\mathit{list\text{-}rel}\rangle\mathit{list\text{-}rel} \times_r$
$Id \times_r \langle\mathit{bool\text{-}rel}\rangle\mathit{list\text{-}rel} \times_r Id \times_r Id \times_r Id \times_r Id\rangle\mathit{nres\text{-}rel}\rangle$
$\langle\mathit{proof}\rangle$

lemma *init-state-wl-heur-init-state-wl'*:
$\langle(\mathit{init\text{-}state\text{-}wl\text{-}heur},\ \mathit{RETURN}\ o\ (\lambda\text{-}.\ \mathit{init\text{-}state\text{-}wl}))$
$\in [\lambda N.\ N = \mathcal{A}_{in} \wedge \mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}_{in}]_f\ Id \rightarrow \langle\mathit{twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl}\ \mathcal{A}_{in}\ \mathit{True}\rangle\mathit{nres\text{-}rel}\rangle$
$\langle\mathit{proof}\rangle$

lemma *all-blits-are-in-problem-init-blits-in*: $\langle\mathit{all\text{-}blits\text{-}are\text{-}in\text{-}problem\text{-}init}\ S \Longrightarrow \mathit{blits\text{-}in\text{-}}\mathcal{L}_{in}\ S\rangle$
$\langle\mathit{proof}\rangle$

lemma *correct-watching-init-blits-in-$\mathcal{L}_{in}$*:
assumes $\langle\mathit{correct\text{-}watching\text{-}init}\ S\rangle$
shows $\langle\mathit{blits\text{-}in\text{-}}\mathcal{L}_{in}\ S\rangle$
$\langle\mathit{proof}\rangle$

fun *append-empty-watched* **where**
$\langle\mathit{append\text{-}empty\text{-}watched}\ ((M,\ N,\ D,\ NE,\ UE,\ NS,\ US,\ Q),\ OC) = ((M,\ N,\ D,\ NE,\ UE,\ NS,\ US,\ Q,$
$(\lambda\text{-}.\ [])),\ OC)\rangle$

fun *remove-watched* :: $\langle'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init\text{-}full} \Rightarrow\ 'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init}\rangle$ **where**
$\langle\mathit{remove\text{-}watched}\ ((M,\ N,\ D,\ NE,\ UE,\ NS,\ US,\ Q,\ \text{-}),\ OC) = ((M,\ N,\ D,\ NE,\ UE,\ NS,\ US,\ Q),\ OC)\rangle$

definition *init-dt-wl'* :: $\langle'v\ \mathit{clause\text{-}l}\ \mathit{list} \Rightarrow\ 'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init} \Rightarrow\ 'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init\text{-}full}\ \mathit{nres}\rangle$ **where**
$\langle\mathit{init\text{-}dt\text{-}wl}'\ CS\ S = do\{$
$S \leftarrow \mathit{init\text{-}dt\text{-}wl}\ CS\ S;$
$\mathit{RETURN}\ (\mathit{append\text{-}empty\text{-}watched}\ S)$
$\}\rangle$

lemma *init-dt-wl'-spec*: $\langle\mathit{init\text{-}dt\text{-}wl\text{-}pre}\ CS\ S \Longrightarrow \mathit{init\text{-}dt\text{-}wl}'\ CS\ S \le \Downarrow$
$(\{(S :: 'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init\text{-}full},\ S' :: 'v\ \mathit{twl\text{-}st\text{-}wl\text{-}init}).$

remove-watched S = S'}) (*SPEC* (*init-dt-wl-spec CS S*))›
⟨*proof*⟩

lemma *init-dt-wl'-init-dt*:
‹*init-dt-wl-pre CS S* ⟹ (*S*, *S'*) ∈ *state-wl-l-init* ⟹ ∀ *C*∈*set CS*. *distinct C* ⟹
init-dt-wl' CS S ≤ ⇓
({(*S* :: *'v twl-st-wl-init-full*, *S'* :: *'v twl-st-wl-init*).
remove-watched S = S'} *O state-wl-l-init*) (*init-dt CS S'*)›
⟨*proof*⟩

definition *isasat-init-fast-slow* :: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*isasat-init-fast-slow* =
(λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*).
RETURN (*trail-pol-slow-of-fast M'*, *N'*, *D'*, *j*, *convert-wlists-to-nat-conv W'*, *vm*, *φ*,
clvls, *cach*, *lbd*, *vdom*, *failed*))›

lemma *isasat-init-fast-slow-alt-def*:
‹*isasat-init-fast-slow S* = *RETURN S*›
⟨*proof*⟩

end
theory *IsaSAT-Initialisation-LLVM*
imports *IsaSAT-Setup-LLVM IsaSAT-VMTF-LLVM Watched-Literals.Watched-Literals-Watch-List-Initialisation*
Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Initialisation
begin

abbreviation *unat-rel32* :: ‹(*32 word* × *nat*) *set*› **where** ‹*unat-rel32* ≡ *unat-rel*›
abbreviation *unat-rel64* :: ‹(*64 word* × *nat*) *set*› **where** ‹*unat-rel64* ≡ *unat-rel*›
abbreviation *snat-rel32* :: ‹(*32 word* × *nat*) *set*› **where** ‹*snat-rel32* ≡ *snat-rel*›
abbreviation *snat-rel64* :: ‹(*64 word* × *nat*) *set*› **where** ‹*snat-rel64* ≡ *snat-rel*›

type-synonym (**in** −)*vmtf-assn-option-fst-As* =
‹*vmtf-node-assn ptr* × *64 word* × *32 word* × *32 word* × *32 word*›

type-synonym (**in** −)*vmtf-remove-assn-option-fst-As* =
‹*vmtf-assn-option-fst-As* × (*32 word array-list64*) × *1 word ptr*›

abbreviation (**in** −) *vmtf-conc-option-fst-As* :: ‹- ⇒ - ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*vmtf-conc-option-fst-As* ≡ (*array-assn vmtf-node-assn* $\times_a$ *uint64-nat-assn* $\times_a$
atom.option-assn $\times_a$ *atom.option-assn* $\times_a$ *atom.option-assn*)›

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* $\times_a$ *distinct-atoms-assn*›

sepref-register *atoms-hash-empty*
sepref-def (**in** −) *atoms-hash-empty-code*
is ‹*atoms-hash-int-empty*›
:: ‹*sint32-nat-assn*^{k} $\rightarrow_a$ *atoms-hash-assn*›
⟨*proof*⟩

sepref-def *distinct-atms-empty-code*
is ‹*distinct-atms-int-empty*›

:: ⟨*sint64-nat-assn*^{k} $\rightarrow_a$ *distinct-atoms-assn*⟩
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *distinct-atms-empty-code.refine atoms-hash-empty-code.refine*

type-synonym (**in** −)*twl-st-wll-trail-init* =
⟨*trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × *vmtf-remove-assn-option-fst-As* × *phase-saver-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *vdom-fast-assn* × *1 word*⟩

definition *isasat-init-assn*
:: ⟨*twl-st-wl-heur-init* ⇒ *trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × - × *phase-saver-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *vdom-fast-assn* × *1 word* ⇒ *assn*⟩
where
⟨*isasat-init-assn* =
trail-pol-fast-assn $\times_a$ *arena-fast-assn* $\times_a$
conflict-option-rel-assn $\times_a$
sint64-nat-assn $\times_a$
watchlist-fast-assn $\times_a$
vmtf-remove-conc-option-fst-As $\times_a$ *phase-saver-assn* $\times_a$
uint32-nat-assn $\times_a$
cach-refinement-l-assn $\times_a$
lbd-assn $\times_a$
vdom-fast-assn $\times_a$
bool1-assn⟩

sepref-def *initialise-VMTF-code*
is ⟨*uncurry initialise-VMTF*⟩
:: ⟨$[\lambda(N, n).\ True]_a$ (*arl64-assn atom-assn*)^k $*_a$ *sint64-nat-assn*^{k} → *vmtf-remove-conc-option-fst-As*⟩
⟨*proof*⟩

declare *initialise-VMTF-code.refine*[*sepref-fr-rules*]
sepref-register *cons-trail-Propagated-tr*
sepref-def *propagate-unit-cls-code*
is ⟨*uncurry* (*propagate-unit-cls-heur*)⟩
:: ⟨*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*⟩
⟨*proof*⟩

declare *propagate-unit-cls-code.refine*[*sepref-fr-rules*]

definition *already-propagated-unit-cls-heur'* **where**
⟨*already-propagated-unit-cls-heur'* = (λ(*M*, *N*, *D*, *Q*, *oth*).
RETURN (*M*, *N*, *D*, *Q*, *oth*))⟩

lemma *already-propagated-unit-cls-heur'-alt*:
⟨*already-propagated-unit-cls-heur L* = *already-propagated-unit-cls-heur'*⟩
⟨*proof*⟩

sepref-def *already-propagated-unit-cls-code*
is ⟨*already-propagated-unit-cls-heur'*⟩
:: ⟨*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*⟩
⟨*proof*⟩

declare *already-propagated-unit-cls-code.refine*[*sepref-fr-rules*]

sepref-def *set-conflict-unit-code*
is ‹*uncurry set-conflict-unit-heur*›
:: ‹$[\lambda(L, (b, n, xs)).\ atm\text{-}of\ L < length\ xs]_a$
$unat\text{-}lit\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d \rightarrow conflict\text{-}option\text{-}rel\text{-}assn$›
⟨*proof*⟩

declare *set-conflict-unit-code.refine*[*sepref-fr-rules*]

sepref-def *conflict-propagated-unit-cls-code*
is ‹*uncurry* (*conflict-propagated-unit-cls-heur*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

declare *conflict-propagated-unit-cls-code.refine*[*sepref-fr-rules*]

sepref-register *fm-add-new*

lemma *add-init-cls-code-bI*:
assumes
‹$length\ at' \leq Suc\ (Suc\ uint32\text{-}max)$› **and**
‹$2 \leq length\ at'$› **and**
‹$length\ a1'j \leq length\ a1'a$› **and**
‹$length\ a1'a \leq sint64\text{-}max - length\ at' - 5$›
shows ‹*append-and-length-fast-code-pre* ((*True*, *at'*), *a1'a*)› ‹$5 \leq sint64\text{-}max - length\ at'$›
⟨*proof*⟩

lemma *add-init-cls-code-bI2*:
assumes
‹$length\ at' \leq Suc\ (Suc\ uint32\text{-}max)$›
shows ‹$5 \leq sint64\text{-}max - length\ at'$›
⟨*proof*⟩

lemma *add-init-clss-codebI*:
assumes
‹$length\ at' \leq Suc\ (Suc\ uint32\text{-}max)$› **and**
‹$2 \leq length\ at'$› **and**
‹$length\ a1'j \leq length\ a1'a$› **and**
‹$length\ a1'a \leq uint64\text{-}max - (length\ at' + 5)$›
shows ‹$length\ a1'j < uint64\text{-}max$›
⟨*proof*⟩

abbreviation *clauses-ll-assn* **where**
‹$clauses\text{-}ll\text{-}assn \equiv aal\text{-}assn'\ TYPE(64)\ TYPE(64)\ unat\text{-}lit\text{-}assn$›

definition *fm-add-new-fast'* **where**
‹$fm\text{-}add\text{-}new\text{-}fast'\ b\ C\ i = fm\text{-}add\text{-}new\text{-}fast\ b\ (C!i)$›

lemma *op-list-list-llen-alt-def*: ‹$op\text{-}list\text{-}list\text{-}llen\ xss\ i = length\ (xss\ !\ i)$›
⟨*proof*⟩

lemma *op-list-list-idx-alt-def*: ‹$op\text{-}list\text{-}list\text{-}idx\ xs\ i\ j = xs\ !\ i\ !\ j$›
⟨*proof*⟩

sepref-def *append-and-length-fast-code*
  is ‹*uncurry3 fm-add-new-fast'*›
  :: ‹[λ(((*b*, *C*), *i*), *N*). *i* < *length C* ∧ *append-and-length-fast-code-pre* ((*b*, *C*!*i*), *N*)]$_a$
    bool1-assn^k $*_a$ *clauses-ll-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ (*arena-fast-assn*)^d →
      arena-fast-assn $\times_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

sepref-register *fm-add-new-fast'*

sepref-def *add-init-cls-code-b*
  is ‹*uncurry2 add-init-cls-heur-b'*›
  :: ‹[λ((*xs*, *i*), *S*). *i* < *length xs*]$_a$
    (*clauses-ll-assn*)^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-init-assn*^{d} → *isasat-init-assn*›
  ⟨*proof*⟩

declare
  add-init-cls-code-b.refine[*sepref-fr-rules*]

sepref-def *already-propagated-unit-cls-conflict-code*
  is ‹*uncurry already-propagated-unit-cls-conflict-heur*›
  :: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
  ⟨*proof*⟩

declare *already-propagated-unit-cls-conflict-code.refine*[*sepref-fr-rules*]

sepref-def (**in** −) *set-conflict-empty-code*
  is ‹*RETURN o lookup-set-conflict-empty*›
  :: ‹*conflict-option-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*›
  ⟨*proof*⟩

declare *set-conflict-empty-code.refine*[*sepref-fr-rules*]

sepref-def *set-empty-clause-as-conflict-code*
  is ‹*set-empty-clause-as-conflict-heur*›
  :: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
  ⟨*proof*⟩

declare *set-empty-clause-as-conflict-code.refine*[*sepref-fr-rules*]

definition (**in** −) *add-clause-to-others-heur'*
  :: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
  ‹*add-clause-to-others-heur'* = (λ (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*).
    RETURN (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*))›

lemma *add-clause-to-others-heur'-alt*: ‹*add-clause-to-others-heur L* = *add-clause-to-others-heur'*›
  ⟨*proof*⟩
sepref-def *add-clause-to-others-code*
  is ‹*add-clause-to-others-heur'*›
  :: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
  ⟨*proof*⟩

declare *add-clause-to-others-code.refine*[*sepref-fr-rules*]

sepref-def *get-conflict-wl-is-None-init-code*
  is ‹*RETURN o get-conflict-wl-is-None-heur-init*›

:: ‹*isasat-init-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

declare *get-conflict-wl-is-None-init-code.refine*[*sepref-fr-rules*]

sepref-def *polarity-st-heur-init-code*
is ‹*uncurry* (*RETURN oo polarity-st-heur-init*)›
:: ‹[λ(*S*, *L*). *polarity-pol-pre* (*get-trail-wl-heur-init S*) *L*]$_a$ *isasat-init-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *tri-bool-assn*›
⟨*proof*⟩

declare *polarity-st-heur-init-code.refine*[*sepref-fr-rules*]

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*

lemma *is-Nil-length*: ‹*is-Nil xs* $\longleftrightarrow$ *length xs* = *0*›
⟨*proof*⟩

definition *init-dt-step-wl-heur-b′*
:: ‹*nat clause-l list* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*› **where**
‹*init-dt-step-wl-heur-b′ C i* = *init-dt-step-wl-heur-b* (*C*!*i*)›

sepref-def *init-dt-step-wl-code-b*
is ‹*uncurry2* (*init-dt-step-wl-heur-b′*)›
:: ‹[λ((*xs*, *i*), *S*). *i* < *length xs*]$_a$ (*clauses-ll-assn*)^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow$
isasat-init-assn›
⟨*proof*⟩

declare
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* ≡ *larray64-assn uint64-nat-assn* $\times_a$ *uint32-nat-assn* $\times_a$
arl64-assn atom-assn›

abbreviation *nat-lit-list-hm-assn* **where**
‹*nat-lit-list-hm-assn* ≡ *hr-comp isasat-atms-ext-rel-assn isasat-atms-ext-rel*›

sepref-def *init-next-size-impl*
is ‹*RETURN o init-next-size*›
:: ‹[λ*L*. *L* $\leq$ *uint32-max div 2*]$_a$ *sint64-nat-assn*^{k} $\rightarrow$ *sint64-nat-assn*›
⟨*proof*⟩

find-in-thms *op-list-grow-init* **in** *sepref-fr-rules*

sepref-def *nat-lit-lits-init-assn-assn-in*
is ‹*uncurry add-to-atms-ext*›
:: ‹$\mathit{atom\text{-}assn}^k *_a \mathit{isasat\text{-}atms\text{-}ext\text{-}rel\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}atms\text{-}ext\text{-}rel\text{-}assn}$›
⟨*proof*⟩

find-theorems *nfoldli WHILET*
lemma [*sepref-fr-rules*]:
‹(*uncurry nat-lit-lits-init-assn-assn-in*, *uncurry* (*RETURN* ∘∘ *op-set-insert*))
$\in [\lambda(a, b).\ a \le \mathit{uint32\text{-}max\ div\ 2}]_a$
$\mathit{atom\text{-}assn}^k *_a \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}^d \rightarrow \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}$›
⟨*proof*⟩

lemma *while-nfoldli*:
do {
(-,σ) ← WHILE_T (*FOREACH-cond c*) (λx. *do* {*ASSERT* (*FOREACH-cond c x*); *FOREACH-body f x*}) (*l*,σ);
RETURN σ
} ≤ *nfoldli l c f* σ
⟨*proof*⟩

definition *extract-atms-cls-i'* **where**
‹*extract-atms-cls-i' C i* = *extract-atms-cls-i* (*C*!*i*)›

lemma *aal-assn-boundsD'*:
assumes *A*: ‹*rdomp* (*aal-assn' TYPE*(*'l*::*len2*) *TYPE*(*'ll*::*len2*) *A*) *xss*› **and** ‹*i* < *length xss*›
shows ‹*length* (*xss* ! *i*) < *max-snat LENGTH*(*'ll*)›
⟨*proof*⟩

sepref-def *extract-atms-cls-imp*
is ‹*uncurry2 extract-atms-cls-i'*›
:: ‹$[\lambda((N, i), \text{-}).\ i < \mathit{length}\ N]_a$
$(\mathit{clauses\text{-}ll\text{-}assn})^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}^d \rightarrow \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}$›
⟨*proof*⟩

declare *extract-atms-cls-imp.refine*[*sepref-fr-rules*]

sepref-def *extract-atms-clss-imp*
is ‹*uncurry extract-atms-clss-i*›
:: ‹$(\mathit{clauses\text{-}ll\text{-}assn})^k *_a \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}^d \rightarrow_a \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}$›
⟨*proof*⟩

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
‹(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* ∘∘ *extract-atms-clss*))
$\in [\lambda(a, b).\ \forall C\in set\ a.\ \forall L\in set\ C.\ \mathit{nat\text{-}of\text{-}lit}\ L \le \mathit{uint32\text{-}max}]_a$
$(\mathit{clauses\text{-}ll\text{-}assn})^k *_a \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}^d \rightarrow \mathit{nat\text{-}lit\text{-}list\text{-}hm\text{-}assn}$›
⟨*proof*⟩

sepref-def *extract-atms-clss-imp-empty-assn*
is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
:: ‹$\mathit{unit\text{-}assn}^k \rightarrow_a \mathit{isasat\text{-}atms\text{-}ext\text{-}rel\text{-}assn}$›
⟨*proof*⟩

lemma *extract-atms-clss-imp-empty-assn*[*sepref-fr-rules*]:

‹(*uncurry0 extract-atms-clss-imp-empty-assn*, *uncurry0* (*RETURN op-extract-list-empty*))
∈ *unit-assn*^{k} $\rightarrow_a$ *nat-lit-list-hm-assn*›
⟨*proof*⟩

lemma *extract-atms-clss-imp-empty-rel-alt-def*:
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*op-larray-custom-replicate 1024 0*, *0*, []))›
⟨*proof*⟩

Full Initialisation

sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-init-assn^d → *isasat-init-assn*›
⟨*proof*⟩

declare
rewatch-heur-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *rewatch-heur-st init-dt-step-wl-heur*

sepref-def *init-dt-wl-heur-code-b*
is ‹*uncurry* (*init-dt-wl-heur-b*)›
:: ‹(*clauses-ll-assn*)^k $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$
isasat-init-assn›
⟨*proof*⟩

declare
init-dt-wl-heur-code-b.refine[*sepref-fr-rules*]

definition *extract-lits-sorted′* **where**
‹*extract-lits-sorted′ xs n vars* = *extract-lits-sorted* (*xs*, *n*, *vars*)›

lemma *extract-lits-sorted-extract-lits-sorted′*:
‹*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {*res* ← *extract-lits-sorted′ xs n vars*; *mop-free xs*; *RETURN res*})›
⟨*proof*⟩

sepref-def (**in** −) *extract-lits-sorted′-impl*
is ‹*uncurry2 extract-lits-sorted′*›
:: ‹[λ((*xs*, *n*), *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
(*larray64-assn uint64-nat-assn*)^k $*_a$ *uint32-nat-assn*^{k} $*_a$
(*arl64-assn atom-assn*)^d →
arl64-assn atom-assn $\times_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *extract-lits-sorted′-impl.refine*

sepref-def (**in** −) *extract-lits-sorted-code*
is ‹*extract-lits-sorted*›
:: ‹[λ(*xs*, *n*, *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
isasat-atms-ext-rel-assn^d →
arl64-assn atom-assn $\times_a$ *uint32-nat-assn*›

⟨*proof*⟩

declare *extract-lits-sorted-code.refine*[*sepref-fr-rules*]

abbreviation *lits-with-max-assn* **where**
‹*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* $\times_a$ *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*⟩

definition *INITIAL-OUTL-SIZE* :: ‹*nat*› **where**
[*simp*]: ‹*INITIAL-OUTL-SIZE* = *160*›

sepref-def *INITIAL-OUTL-SIZE-impl*
is ‹*uncurry0* (*RETURN INITIAL-OUTL-SIZE*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

definition *atom-of-value* :: ‹*nat* ⇒ *nat*› **where** [*simp*]: ‹*atom-of-value x* = *x*›

lemma *atom-of-value-simp-hnr*:
‹(∃ *x*. (↑(*x* = *unat xi* ∧ *P x*) ∧* ↑(*x* = *unat xi*)) *s*) =
(∃ *x*. (↑(*x* = *unat xi* ∧ *P x*)) *s*)›
‹(∃ *x*. (↑(*x* = *unat xi* ∧ *P x*)) *s*) = (↑(*P* (*unat xi*))) *s*›
⟨*proof*⟩

lemma *atom-of-value-hnr*[*sepref-fr-rules*]:
‹(*return o* (λ*x*. *x*), *RETURN o atom-of-value*) ∈ [λ*n*. *n* < *2* ^*31*]$_a$ (*uint32-nat-assn*)^d → *atom-assn*›
⟨*proof*⟩

sepref-register *atom-of-value*

lemma [*sepref-gen-algo-rules*]: ‹*GEN-ALGO* (*Pos 0*) (*is-init unat-lit-assn*)›
⟨*proof*⟩

sepref-def *finalise-init-code'*
is ‹*uncurry finalise-init-code*›
:: ‹[λ(-, *S*). *length* (*get-clauses-wl-heur-init S*) ≤ *sint64-max*]$_a$
opts-assn^d $*_a$ *isasat-init-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

declare *finalise-init-code'.refine*[*sepref-fr-rules*]

sepref-register *initialise-VMTF*
abbreviation *snat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*snat64-assn* ≡ *snat-assn*›
abbreviation *snat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*snat32-assn* ≡ *snat-assn*›

abbreviation *unat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*unat64-assn* ≡ *unat-assn*›
abbreviation *unat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*unat32-assn* ≡ *unat-assn*›

sepref-def *init-trail-D-fast-code*
  is ‹*uncurry2 init-trail-D-fast*›
  :: ‹$(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a trail\text{-}pol\text{-}fast\text{-}assn$›
  ⟨*proof*⟩

declare *init-trail-D-fast-code.refine*[*sepref-fr-rules*]

sepref-def *init-state-wl-D′-code*
  is ‹*init-state-wl-D′*›
  :: ‹$(arl64\text{-}assn\ atom\text{-}assn \times_a uint32\text{-}nat\text{-}assn)^k \rightarrow_a isasat\text{-}init\text{-}assn$›
  ⟨*proof*⟩

declare *init-state-wl-D′-code.refine*[*sepref-fr-rules*]

lemma *to-init-state-code-hnr*:
  ‹$(return\ o\ to\text{-}init\text{-}state\text{-}code,\ RETURN\ o\ id) \in isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
  ⟨*proof*⟩

abbreviation (**in** −)*lits-with-max-assn-clss* **where**
  ‹*lits-with-max-assn-clss* ≡ *hr-comp lits-with-max-assn* (⟨*nat-rel*⟩*mset-rel*)›

experiment
begin
  export-llvm *init-state-wl-D′-code*
    rewatch-heur-st-fast-code
    init-dt-wl-heur-code-b

end

end
theory *IsaSAT-Conflict-Analysis*
  imports *IsaSAT-Setup IsaSAT-VMTF IsaSAT-LBD*
begin

Skip and resolve   **definition** *maximum-level-removed-eq-count-dec* **where**
  ‹*maximum-level-removed-eq-count-dec L S* ⟷
      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-pre* **where**
  ‹*maximum-level-removed-eq-count-dec-pre* =
     (λ(*L*, *S*). *L* = −*lit-of* (*hd* (*get-trail-wl S*)) ∧ *L* ∈# *the* (*get-conflict-wl S*) ∧
      get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧ *count-decided* (*get-trail-wl S*) ≥ *1*)›

definition *maximum-level-removed-eq-count-dec-heur* **where**
  ‹*maximum-level-removed-eq-count-dec-heur L S* =
      RETURN (*get-count-max-lvls-heur S* > *1*)›

lemma *maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec*:
  ‹(*uncurry maximum-level-removed-eq-count-dec-heur*,

uncurry mop-maximum-level-removed-wl) ∈
[λ-. *True*]$_f$
Id $\times_r$ *twl-st-heur-conflict-ana* → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-trail-wl-heur-def*: ‹*get-trail-wl-heur* = (λ(*M*, *S*). *M*)›
⟨*proof*⟩

definition *lit-and-ann-of-propagated-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* × *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* ⇒ (*nat literal* × *nat*) *nres*›
where
‹*lit-and-ann-of-propagated-st-heur* = (λ((*M*, -, -, *reasons*, -), -). *do* {
ASSERT(*M* ≠ [] ∧ *atm-of* (*last M*) < *length reasons*);
RETURN (*last M*, *reasons* ! (*atm-of* (*last M*)))})›

lemma *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*:
‹(*lit-and-ann-of-propagated-st-heur*, *mop-hd-trail-wl*) ∈
[λ*S*. *True*]$_f$ *twl-st-heur-conflict-ana* → ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
⟨*proof*⟩

definition *tl-state-wl-heur-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*tl-state-wl-heur-pre* =
(λ(*M*, *N*, *D*, *WS*, *Q*, ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), -). *fst M* ≠ [] ∧
tl-trailt-tr-pre M ∧
vmtf-unset-pre (*atm-of* (*last* (*fst M*))) ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∧
atm-of (*last* (*fst M*)) < *length A* ∧
(*next-search* ≠ *None* ⟶ *the next-search* < *length A*))›

definition *tl-state-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*› **where**
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*));
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of* (*lit-of-last-trail-pol M*))
vmtf, *clvls*))
})›

lemma *tl-state-wl-heur-alt-def*:
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*));
let L = *lit-of-last-trail-pol M*;
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of L*) *vmtf*, *clvls*))
})›
⟨*proof*⟩

definition *tl-state-wl-pre* **where**
‹*tl-state-wl-pre S* ⟷ *get-trail-wl S* ≠ [] ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
−(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧

¬*is-decided* (*hd* (*get-trail-wl S*)) ∧
count-decided (*get-trail-wl S*) > *0*›

lemma *tl-state-out-learned*:
‹*lit-of* (*hd a*) ∉# *the at* ⟹
− *lit-of* (*hd a*) ∉# *the at* ⟹
¬ *is-decided* (*hd a*) ⟹
out-learned (*tl a*) *at an* ⟷ *out-learned a at an*›
⟨*proof*⟩

lemma *mop-tl-state-wl-pre-tl-state-wl-heur-pre*:
‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹ *mop-tl-state-wl-pre y* ⟹ *tl-state-wl-heur-pre x*›
⟨*proof*⟩

lemma *mop-tl-state-wl-pre-simps*:
‹*mop-tl-state-wl-pre* ([], *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟷ *False*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹
lit-of (*hd xa*) ∈# $\mathcal{L}_{all}$ (*all-atms ax* (*az* + *bga* + *NS* + *US*))›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ *lit-of* (*hd xa*) ∉# *the ay*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ −*lit-of* (*hd xa*) ∉# *the ay*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ *lit-of* (*hd xa*) ∉# *ay'*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ −*lit-of* (*hd xa*) ∉# *ay'*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ *is-proped* (*hd xa*)›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) ⟹ *count-decided xa* > *0*›
⟨*proof*⟩

lemma *tl-state-wl-heur-tl-state-wl*:
‹(*tl-state-wl-heur*, *mop-tl-state-wl*) ∈
$[\lambda$-. *True*$]_f$ *twl-st-heur-conflict-ana' r* → ⟨*bool-rel* $\times_f$ *twl-st-heur-conflict-ana' r*⟩*nres-rel*›
⟨*proof*⟩

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-used arena C*) *C*›
⟨*proof*⟩

definition (**in** −) *get-max-lvl-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*get-max-lvl-st S L* = *get-maximum-level-remove* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L*›

definition *update-confl-tl-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where

```
‹update-confl-tl-wl-heur = (λL C (M, N, (b, (n, xs)), Q, W, vm, clvls, cach, lbd, outl, stats). do {
    (N, lbd) ← calculate-LBD-heur-st M N lbd C;
    ASSERT (clvls ≥ 1);
    let L' = atm-of L;
    ASSERT(arena-is-valid-clause-idx N C);
    ((b, (n, xs)), clvls, outl) ←
      if arena-length N C = 2 then isasat-lookup-merge-eq2 L M N C (b, (n, xs)) clvls outl
      else isa-resolve-merge-conflict-gt2 M N C (b, (n, xs)) clvls outl;
    ASSERT(curry lookup-conflict-remove1-pre L (n, xs) ∧ clvls ≥ 1);
    let (n, xs) = lookup-conflict-remove1 L (n, xs);
    ASSERT(arena-act-pre N C);
    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*,
clvls − *1*, *cach*, *lbd*, *outl*, *stats*))
})›

lemma *card-max-lvl-remove1-mset-hd*:
‹−*lit-of* (*hd M*) ∈# *y* ⟹ *is-proped* (*hd M*) ⟹
card-max-lvl M (*remove1-mset* (−*lit-of* (*hd M*)) *y*) = *card-max-lvl M y* − *1*›
⟨*proof*⟩

lemma *update-confl-tl-wl-heur-state-helper*:
‹(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ⟹ *get-trail-wl S* ≠ [] ⟹
is-proped (*hd* (*get-trail-wl S*)) ⟹ *L* = *lit-of* (*hd* (*get-trail-wl S*))›
⟨*proof*⟩

lemma (**in** −) *not-ge-Suc0*: ‹¬*Suc 0* ≤ *n* ⟷ *n* = *0*›
⟨*proof*⟩

definition *update-confl-tl-wl-pre′* :: ‹((*nat literal* × *nat*) × *nat twl-st-wl*) ⇒ *bool*› **where**
‹*update-confl-tl-wl-pre′* = (λ((*L*, *C*), *S*).
C ∈# *dom-m* (*get-clauses-wl S*) ∧
get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧
− *L* ∈# *the* (*get-conflict-wl S*) ∧
L ∉# *the* (*get-conflict-wl S*) ∧
(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
is-proped (*hd* (*get-trail-wl S*)) ∧
C > *0* ∧
card-max-lvl (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1* ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
− *L* ∉ *set* (*get-clauses-wl S* ∝ *C*) ∧
(*length* (*get-clauses-wl S* ∝ *C*) ≠ *2* ⟶
L ∉ *set* (*tl* (*get-clauses-wl S* ∝ *C*)) ∧
get-clauses-wl S ∝ *C* ! *0* = *L* ∧
mset (*tl* (*get-clauses-wl S* ∝ *C*)) = *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
(∀ *L*∈*set* (*tl*(*get-clauses-wl S* ∝ *C*)). − *L* ∉# *the* (*get-conflict-wl S*)) ∧
card-max-lvl (*get-trail-wl S*) (*mset* (*tl* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*))) ∧
L ∈ *set* (*watched-l* (*get-clauses-wl S* ∝ *C*)) ∧
distinct (*get-clauses-wl S* ∝ *C*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
¬*tautology* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
¬*tautology* (*remove1-mset L* (*remove1-mset* (− *L*)
((*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*))))) ∧
count-decided (*get-trail-wl S*) > *0* ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*)) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(∀ *K*. *K* ∈# *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ⟶ − *K* ∉# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) > *0* ∧
Suc 0 ≤ *card-max-lvl* (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
size (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
lit-of (*hd* (*get-trail-wl S*)) = *L* ∧
card-max-lvl (*get-trail-wl S*) ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪#

the (*get-conflict-wl S*)) =
card-max-lvl (*tl* (*get-trail-wl S*)) (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
out-learned (*tl* (*get-trail-wl S*)) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) =
out-learned (*get-trail-wl S*) (*Some* ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)))
)›

lemma *remove1-mset-union-distrib1*:
‹*L* ∉# *B* ⟹ *remove1-mset L* (*A* ∪# *B*) = *remove1-mset L A* ∪# *B*› **and**
remove1-mset-union-distrib2:
‹*L* ∉# *A* ⟹ *remove1-mset L* (*A* ∪# *B*) = *A* ∪# *remove1-mset L B*›
⟨*proof*⟩

lemma *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*:
assumes ‹*update-confl-tl-wl-pre L C S*›
shows ‹*update-confl-tl-wl-pre'* ((*L*, *C*), *S*)›
⟨*proof*⟩

lemma (**in** −)*out-learned-add-mset-highest-level*:
‹*L* = *lit-of* (*hd M*) ⟹ *out-learned M* (*Some* (*add-mset* (− *L*) *A*)) *outl* ⟷
out-learned M (*Some A*) *outl*›
⟨*proof*⟩

lemma (**in** −)*out-learned-tl-Some-notin*:
‹*is-proped* (*hd M*) ⟹ *lit-of* (*hd M*) ∉# *C* ⟹ −*lit-of* (*hd M*) ∉# *C* ⟹
out-learned M (*Some C*) *outl* ⟷ *out-learned* (*tl M*) (*Some C*) *outl*›
⟨*proof*⟩

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*#}›
⟨*proof*⟩

lemma *mset-as-position-remove3*:
‹*mset-as-position xs* (*D* − {#*L*#}) ⟹ *atm-of L* < *length xs* ⟹ *distinct-mset D* ⟹
mset-as-position (*xs*[*atm-of L* := *None*]) (*D* − {#*L*, −*L*#})›
⟨*proof*⟩

lemma *imply-itself*: ‹*P* ⟹ *P*›
⟨*proof*⟩

lemma *update-confl-tl-wl-heur-update-confl-tl-wl*:
‹(*uncurry2* (*update-confl-tl-wl-heur*), *uncurry2 mop-update-confl-tl-wl*) ∈
$[\lambda\text{-}.\ True]_f$
Id $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-conflict-ana' r* → ⟨*bool-rel* $\times_f$ *twl-st-heur-conflict-ana' r*⟩*nres-rel*›
⟨*proof*⟩

lemma *phase-saving-le*: ‹*phase-saving* 𝒜 φ ⟹ *A* ∈# 𝒜 ⟹ *A* < *length* φ›
‹*phase-saving* 𝒜 φ ⟹ *B* ∈# $\mathcal{L}_{all}$ 𝒜 ⟹ *atm-of B* < *length* φ›
⟨*proof*⟩

lemma *isa-vmtf-le*:
‹((*a*, *b*), *M*) ∈ *isa-vmtf* 𝒜 *M'* ⟹ *A* ∈# 𝒜 ⟹ *A* < *length a*›

‹((*a*, *b*), *M*) ∈ *isa-vmtf* $\mathcal{A}$ *M'* ⟹ *B* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of B* < *length a*›
⟨*proof*⟩

lemma *isa-vmtf-next-search-le*:
‹((*a*, *b*, *c*, *c'*, *Some d*), *M*) ∈ *isa-vmtf* $\mathcal{A}$ *M'* ⟹ *d* < *length a*›
⟨*proof*⟩

lemma *trail-pol-nempty*: ‹¬(([], *aa*, *ab*, *ac*, *ad*, *b*), *L* # *ys*) ∈ *trail-pol* $\mathcal{A}$›
⟨*proof*⟩

definition *is-decided-hd-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*is-decided-hd-trail-wl-heur* = (λ*S*. *is-None* (*snd* (*last-trail-pol* (*get-trail-wl-heur S*))))›

lemma *is-decided-hd-trail-wl-heur-hd-get-trail*:
‹(*RETURN o is-decided-hd-trail-wl-heur*, *RETURN o* (λ*M*. *is-decided* (*hd* (*get-trail-wl M*))))
∈ [λ*M*. *get-trail-wl M* ≠ []]$_f$ *twl-st-heur-conflict-ana' r* → ⟨*bool-rel*⟩ *nres-rel*›
⟨*proof*⟩

definition *is-decided-hd-trail-wl-heur-pre* **where**
‹*is-decided-hd-trail-wl-heur-pre* =
(λ*S*. *fst* (*get-trail-wl-heur S*) ≠ [] ∧ *last-trail-pol-pre* (*get-trail-wl-heur S*))›

definition *skip-and-resolve-loop-wl-D-heur-inv* **where**
‹*skip-and-resolve-loop-wl-D-heur-inv* S_0' =
(λ(*brk*, *S'*). ∃*S* S_0. (*S'*, *S*) ∈ *twl-st-heur-conflict-ana* ∧ (S_0', S_0) ∈ *twl-st-heur-conflict-ana* ∧
skip-and-resolve-loop-wl-inv S_0 *brk S* ∧
length (*get-clauses-wl-heur S'*) = *length* (*get-clauses-wl-heur* S_0'))›

definition *update-confl-tl-wl-heur-pre*
:: ‹(*nat* × *nat literal*) × *twl-st-wl-heur* ⇒ *bool*›
where
‹*update-confl-tl-wl-heur-pre* =
(λ((*i*, *L*), (*M*, *N*, *D*, *W*, *Q*, ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), -), *clvls*, *cach*, *lbd*,
outl, -)).
i > *0* ∧
(*fst M*) ≠ [] ∧
atm-of ((*last* (*fst M*))) < *length A* ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length A*) ∧
L = (*last* (*fst M*))
)›

definition *lit-and-ann-of-propagated-st-heur-pre* **where**
‹*lit-and-ann-of-propagated-st-heur-pre* = (λ((*M*, -, -, *reasons*, -), -). *atm-of* (*last M*) < *length reasons* ∧ *M* ≠ [])›

definition *atm-is-in-conflict-st-heur-pre*
:: ‹*nat literal* × *twl-st-wl-heur* ⇒ *bool*›
where
‹*atm-is-in-conflict-st-heur-pre* = (λ(*L*, (*M*,*N*,(-, (-, *D*)), -)). *atm-of L* < *length D*)›

definition *skip-and-resolve-loop-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*skip-and-resolve-loop-wl-D-heur* S_0 =
do {
(-, *S*) ←

```
      $WHILE_T{}^{skip\text{-}and\text{-}resolve\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
      $(\lambda(brk, S).\ \neg brk \wedge \neg is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur\ S)$
      $(\lambda(brk, S).$
        do {
          $ASSERT(\neg brk \wedge \neg is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur\ S)$;
          $(L, C) \leftarrow lit\text{-}and\text{-}ann\text{-}of\text{-}propagated\text{-}st\text{-}heur\ S$;
          $b \leftarrow atm\text{-}is\text{-}in\text{-}conflict\text{-}st\text{-}heur\ (-L)\ S$;
          if b then
     tl-state-wl-heur S
          else do {
            $b \leftarrow maximum\text{-}level\text{-}removed\text{-}eq\text{-}count\text{-}dec\text{-}heur\ L\ S$;
            if b
            then do {
              update-confl-tl-wl-heur L C S
            }
            else
              RETURN (True, S)
          }
        }
      )
      (False, $S_0$);
    RETURN S
  }
›
```

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*atm-is-in-conflict-st-heur*), *uncurry* (*mop-lit-notin-conflict-wl*)) ∈
$[\lambda(L, S).\ True]_f$
Id $\times_r$ *twl-st-heur-conflict-ana* → ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *skip-and-resolve-loop-wl-alt-def*:

```
‹skip-and-resolve-loop-wl $S_0$ =
    do {
      $ASSERT(get\text{-}conflict\text{-}wl\ S_0 \neq None)$;
      $(\text{-}, S) \leftarrow$
        $WHILE_T{}^{\lambda(brk, S).\ skip\text{-}and\text{-}resolve\text{-}loop\text{-}wl\text{-}inv\ S_0\ brk\ S}$
        $(\lambda(brk, S).\ \neg brk \wedge \neg is\text{-}decided\ (hd\ (get\text{-}trail\text{-}wl\ S)))$
        $(\lambda(\text{-}, S).$
          do {
            $(L, C) \leftarrow mop\text{-}hd\text{-}trail\text{-}wl\ S$;
            $b \leftarrow mop\text{-}lit\text{-}notin\text{-}conflict\text{-}wl\ (-L)\ S$;
            if b then
              mop-tl-state-wl S
            else do {
              $b \leftarrow mop\text{-}maximum\text{-}level\text{-}removed\text{-}wl\ L\ S$;
              if b
              then do {
                mop-update-confl-tl-wl L C S
              }
              else
                do {RETURN (True, S)}
           }
          }
      )
      (False, S0);
    RETURN S
  }›
⟨proof⟩
```

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*)
  $\in$ *twl-st-heur-conflict-ana′ r* $\rightarrow_f$ ⟨*twl-st-heur-conflict-ana′ r*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *get-count-max-lvls-code* **where**
‹*get-count-max-lvls-code* = (λ(-, -, -, -, -, -, -, *clvls*, -). *clvls*)›

lemma *is-decided-hd-trail-wl-heur-alt-def*:
‹*is-decided-hd-trail-wl-heur* = (λ(*M*, -). *is-None* (*snd* (*last-trail-pol M*)))›
⟨*proof*⟩

lemma *atm-of-in-atms-of*: ‹*atm-of x* ∈ *atms-of C* ⟷ *x* ∈# *C* ∨ −*x* ∈# *C*›
⟨*proof*⟩

definition *atm-is-in-conflict* **where**
‹*atm-is-in-conflict L D* ⟷ *atm-of L* ∈ *atms-of* (*the D*)›

fun *is-in-option-lookup-conflict* **where**
is-in-option-lookup-conflict-def[*simp del*]:
‹*is-in-option-lookup-conflict L* (*a*, *n*, *xs*) ⟷ *is-in-lookup-conflict* (*n*, *xs*) *L*›

lemma *is-in-option-lookup-conflict-atm-is-in-conflict-iff*:
assumes
‹*ba* ≠ *None*› **and** *aa*: ‹*aa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *uaa*: ‹− *aa* ∉# *the ba*› **and**
‹((*b*, *c*, *d*), *ba*) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
shows ‹*is-in-option-lookup-conflict aa* (*b*, *c*, *d*) =
atm-is-in-conflict aa ba›
⟨*proof*⟩

lemma *is-in-option-lookup-conflict-atm-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-option-lookup-conflict*), *uncurry* (*RETURN oo atm-is-in-conflict*))
∈ [λ(*L*, *D*). *D* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ −*L* ∉# *the D*]$_f$
Id $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *is-in-option-lookup-conflict-alt-def*:
‹*RETURN oo is-in-option-lookup-conflict* =
RETURN oo (λ*L* (-, *n*, *xs*). *is-in-lookup-conflict* (*n*, *xs*) *L*)›
⟨*proof*⟩

lemma *skip-and-resolve-loop-wl-DI*:
assumes
‹*skip-and-resolve-loop-wl-D-heur-inv S* (*b*, *T*)›
shows ‹*is-decided-hd-trail-wl-heur-pre T*›
⟨*proof*⟩

lemma *isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv*:
  ‹*isasat-fast x* $\Longrightarrow$
        skip-and-resolve-loop-wl-D-heur-inv x
         (*False*, *a2'*) $\Longrightarrow$ *isasat-fast a2'*›
  ⟨*proof*⟩

end
theory *IsaSAT-Conflict-Analysis-LLVM*
imports *IsaSAT-Conflict-Analysis IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM IsaSAT-LBD-LLVM*
begin
thm *fold-tuple-optimizations*

lemma *get-count-max-lvls-heur-def*:
   ‹*get-count-max-lvls-heur* = (λ(-, -, -, -, -, -, *clvls*, -). *clvls*)›
  ⟨*proof*⟩

sepref-def *get-count-max-lvls-heur-impl*
  is ‹*RETURN o get-count-max-lvls-heur*›
  :: ‹$\textit{isasat-bounded-assn}^k \rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *get-count-max-lvls-heur-impl.refine*

sepref-def *maximum-level-removed-eq-count-dec-fast-code*
  is ‹*uncurry* (*maximum-level-removed-eq-count-dec-heur*)›
  :: ‹$\textit{unat-lit-assn}^k *_a \textit{isasat-bounded-assn}^k \rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

declare
  maximum-level-removed-eq-count-dec-fast-code.refine[*sepref-fr-rules*]

lemma *is-decided-hd-trail-wl-heur-alt-def*:
  ‹*is-decided-hd-trail-wl-heur* = (λ((*M*, *xs*, *lvls*, *reasons*, *k*), -).
      let r = *reasons* ! (*atm-of* (*last M*)) *in*
      r = *DECISION-REASON*)›
  ⟨*proof*⟩

sepref-def *is-decided-hd-trail-wl-fast-code*
  is ‹*RETURN o is-decided-hd-trail-wl-heur*›
  :: ‹$[\textit{is-decided-hd-trail-wl-heur-pre}]_a\ \textit{isasat-bounded-assn}^k \rightarrow$ *bool1-assn*›
  ⟨*proof*⟩

declare
  is-decided-hd-trail-wl-fast-code.refine[*sepref-fr-rules*]

sepref-def *lit-and-ann-of-propagated-st-heur-fast-code*
  is ‹*lit-and-ann-of-propagated-st-heur*›
  :: ‹$[\lambda\text{-}.\ \textit{True}]_a$
       $\textit{isasat-bounded-assn}^k \rightarrow (\textit{unat-lit-assn} \times_a \textit{sint64-nat-assn})$›
  ⟨*proof*⟩

declare
  lit-and-ann-of-propagated-st-heur-fast-code.refine[*sepref-fr-rules*]

definition *is-UNSET* **where** [*simp*]: ‹*is-UNSET x* ⟷ *x* = *UNSET*›
lemma *tri-bool-is-UNSET-refine-aux*:
‹($\lambda x.\ x = 0$, *is-UNSET*) ∈ *tri-bool-rel-aux* → *bool-rel* ›
⟨*proof*⟩

sepref-definition *is-UNSET-impl*
is ‹*RETURN o* ($\lambda x.\ x= 0$)›
:: ‹(*unat-assn′ TYPE*(*8*))^k $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *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} → *bool1-assn*›
⟨*proof*⟩

sepref-def *atm-is-in-conflict-st-heur-fast-code*
is ‹*uncurry* (*atm-is-in-conflict-st-heur*)›
:: ‹[λ-. *True*]$_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} → *bool1-assn*›
⟨*proof*⟩

declare *atm-is-in-conflict-st-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-def (**in** −) *lit-of-last-trail-fast-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹[λ(*M*). *fst M* ≠ []]$_a$ *trail-pol-fast-assn*^{k} → *unat-lit-assn*›
⟨*proof*⟩

declare *lit-of-last-trail-fast-code.refine*[*sepref-fr-rules*]

lemma *tl-state-wl-heurI*: ‹*tl-state-wl-heur-pre* (*a*, *b*) ⟹ *fst a* ≠ []›
‹*tl-state-wl-heur-pre* (*a*, *b*) ⟹ *tl-trailt-tr-pre a*›
‹*tl-state-wl-heur-pre* (*a1′*, *a1′a*, *a1′b*, *a1′c*, *a1′d*, *a1′e*, *a1′f*, *a2′f*) ⟹
vmtf-unset-pre (*atm-of* (*lit-of-last-trail-pol a1′*)) *a1′e*›
⟨*proof*⟩

lemma *tl-state-wl-heur-alt-def*:
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*));
let L = (*atm-of* (*lit-of-last-trail-pol M*));
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset L vmtf*, φ, *clvls*))
})›
⟨*proof*⟩

sepref-def *tl-state-wl-heur-fast-code*
is ‹*tl-state-wl-heur*›
:: ‹[λ-. *True*]$_a$ *isasat-bounded-assn*^{d} → *bool1-assn* $\times_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare

tl-state-wl-heur-fast-code.refine[*sepref-fr-rules*]

definition *None-lookup-conflict* :: ‹- ⇒ - ⇒ *conflict-option-rel*› **where**
‹*None-lookup-conflict b xs* = (*b*, *xs*)›

sepref-def *None-lookup-conflict-impl*
is ‹*uncurry* (*RETURN oo None-lookup-conflict*)›
:: ‹$bool1\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^d \rightarrow_a conflict\text{-}option\text{-}rel\text{-}assn$›
⟨*proof*⟩

sepref-register *None-lookup-conflict*
declare *None-lookup-conflict-impl.refine*[*sepref-fr-rules*]

definition *extract-values-of-lookup-conflict* :: ‹*conflict-option-rel* ⇒ *bool*› **where**
‹*extract-values-of-lookup-conflict* = (λ(*b*, (-, *xs*)). *b*)›

sepref-def *extract-values-of-lookup-conflict-impl*
is ‹*RETURN o extract-values-of-lookup-conflict*›
:: ‹$conflict\text{-}option\text{-}rel\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

sepref-register *extract-values-of-lookup-conflict*
declare *extract-values-of-lookup-conflict-impl.refine*[*sepref-fr-rules*]

sepref-register *isasat-lookup-merge-eq2 update-confl-tl-wl-heur*

lemma *update-confl-tl-wl-heur-alt-def*:
‹*update-confl-tl-wl-heur* = (λ*L C* (*M*, *N*, *bnxs*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*). *do* {
    (*N*, *lbd*) ← *calculate-LBD-heur-st M N lbd C*;
    ASSERT (*clvls* ≥ *1*);
    let L′ = *atm-of L*;
    ASSERT(*arena-is-valid-clause-idx N C*);
    (*bnxs*, *clvls*, *outl*) ←
      if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C bnxs clvls outl*
      else isa-resolve-merge-conflict-gt2 M N C bnxs clvls outl;
    let b = *extract-values-of-lookup-conflict bnxs*;
    let nxs = *the-lookup-conflict bnxs*;
    ASSERT(*curry lookup-conflict-remove1-pre L nxs* ∧ *clvls* ≥ *1*);
    let nxs = *lookup-conflict-remove1 L nxs*;
    ASSERT(*arena-act-pre N C*);
    ASSERT(*vmtf-unset-pre L′ vm*);
    ASSERT(*tl-trailt-tr-pre M*);
    RETURN (*False*, (*tl-trailt-tr M*, *N*, (*None-lookup-conflict b nxs*), *Q*, *W*, *isa-vmtf-unset L′ vm*,
      clvls − *1*, *cach*, *lbd*, *outl*, *stats*))
})›
⟨*proof*⟩

sepref-def *update-confl-tl-wl-fast-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹$[\lambda((i, L), S).\ isasat\text{-}fast\ S]_a$
$unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow bool1\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

declare *update-confl-tl-wl-fast-code.refine*[*sepref-fr-rules*]

sepref-register *is-in-conflict-st atm-is-in-conflict-st-heur*
sepref-def *skip-and-resolve-loop-wl-D-fast*
  is ‹*skip-and-resolve-loop-wl-D-heur*›
  :: ‹$[\lambda S.\ isasat\text{-}fast\ S]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
  ⟨*proof*⟩

declare *skip-and-resolve-loop-wl-D-fast.refine*[*sepref-fr-rules*]

experiment
begin
  export-llvm
    get-count-max-lvls-heur-impl
    maximum-level-removed-eq-count-dec-fast-code
    is-decided-hd-trail-wl-fast-code
    lit-and-ann-of-propagated-st-heur-fast-code
    is-in-option-lookup-conflict-code
    atm-is-in-conflict-st-heur-fast-code
    lit-of-last-trail-fast-code
    tl-state-wl-heur-fast-code
    None-lookup-conflict-impl
    extract-values-of-lookup-conflict-impl
    update-confl-tl-wl-fast-code
    skip-and-resolve-loop-wl-D-fast

end

end
theory *IsaSAT-Propagate-Conflict*
  imports *IsaSAT-Setup IsaSAT-Inner-Propagation*
begin

Chapter 16

Propagation Loop And Conflict

16.1 Unit Propagation, Inner Loop

definition (**in** −) *length-ll-fs* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs* = (λ(-, -, -, -, -, -, -, -, *W*) *L*. *length* (*W L*))›

definition (**in** −) *length-ll-fs-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs-heur S L* = *length* (*watched-by-int S L*)›

lemma *length-ll-fs-heur-alt-def*:
‹*length-ll-fs-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L*. *length* (*W* ! *nat-of-lit L*))›
⟨*proof*⟩

lemma (**in** −) *get-watched-wl-heur-def*: ‹*get-watched-wl-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -). *W*)›
⟨*proof*⟩

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur b*) ≤ *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*›
⟨*proof*⟩

lemma *unit-propagation-inner-loop-wl-loop-D-heur-alt-def*:

```
‹unit-propagation-inner-loop-wl-loop-D-heur L $S_0$ = do {
  ASSERT (length (watched-by-int $S_0$ L) ≤ length (get-clauses-wl-heur $S_0$));
  n ← mop-length-watched-by-int $S_0$ L;
  let b = (0, 0, $S_0$);
  $WHILE_T{}^{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv\ S_0\ L}$
    (λ(j, w, S). w < n ∧ get-conflict-wl-is-None-heur S)
    (λ(j, w, S). do {
      unit-propagation-inner-loop-body-wl-heur L j w S
    })
    b
}›
⟨proof⟩
```

16.2 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)
   })
  ›
 ⟨proof⟩
```

```
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')›
 ⟨proof⟩
```

```
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)›
 ⟨proof⟩
```

```
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›
 ⟨proof⟩
```

```
end
theory IsaSAT-Propagate-Conflict-LLVM
 imports IsaSAT-Propagate-Conflict IsaSAT-Inner-Propagation-LLVM
begin
```

```
lemma length-ll[def-pat-rules]: ‹length-ll$xs$i ≡ op-list-list-llen$xs$i›
 ⟨proof⟩
```

```
sepref-def 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 → sint64-nat-assn›
 ⟨proof⟩
```

```
sepref-def mop-length-watched-by-int-impl [llvm-inline]
 is ‹uncurry mop-length-watched-by-int›
 :: ‹isasat-bounded-assn^k *_a unat-lit-assn^k →_a sint64-nat-assn›
 ⟨proof⟩
```

```
sepref-register unit-propagation-inner-loop-body-wl-heur


lemma unit-propagation-inner-loop-wl-loop-D-heur-fast:
  ‹length (get-clauses-wl-heur b) ≤ sint64-max ⟹
    unit-propagation-inner-loop-wl-loop-D-heur-inv b a (a1', a1'a, a2'a) ⟹
     length (get-clauses-wl-heur a2'a) ≤ sint64-max›
  ⟨proof⟩

sepref-def unit-propagation-inner-loop-wl-loop-D-fast
  is ‹uncurry unit-propagation-inner-loop-wl-loop-D-heur›
  :: ‹[λ(L, S). length (get-clauses-wl-heur S) ≤ sint64-max]_a
    unat-lit-assn^k *_a isasat-bounded-assn^d → sint64-nat-assn ×_a sint64-nat-assn ×_a isasat-bounded-assn›
  ⟨proof⟩

lemma le-uint64-max-minus-4-uint64-max: ‹a ≤ sint64-max − MIN-HEADER-SIZE ⟹ Suc a <
max-snat 64›
  ⟨proof⟩

definition cut-watch-list-heur2-inv where
  ‹cut-watch-list-heur2-inv L n = (λ(j, w, W). j ≤ w ∧ w ≤ n ∧ nat-of-lit L < length W)›

lemma cut-watch-list-heur2-alt-def:
‹cut-watch-list-heur2 = (λj 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^cut-watch-list-heur2-inv L n
    (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)
})›
  ⟨proof⟩

lemma cut-watch-list-heur2I:
  ‹length (a1'd ! nat-of-lit baa) ≤ sint64-max − MIN-HEADER-SIZE ⟹
       cut-watch-list-heur2-inv baa (length (a1'd ! nat-of-lit baa))
        (a1'e, a1'f, a2'f) ⟹
       a1'f < length-ll a2'f (nat-of-lit baa) ⟹
       ez ≤ bba ⟹
       Suc a1'e < max-snat 64›
  ‹length (a1'd ! nat-of-lit baa) ≤ sint64-max − MIN-HEADER-SIZE ⟹
       cut-watch-list-heur2-inv baa (length (a1'd ! nat-of-lit baa))
        (a1'e, a1'f, a2'f) ⟹
       a1'f < length-ll a2'f (nat-of-lit baa) ⟹
       ez ≤ bba ⟹
       Suc a1'f < max-snat 64›
  ‹cut-watch-list-heur2-inv baa (length (a1'd ! nat-of-lit baa))
        (a1'e, a1'f, a2'f) ⟹ nat-of-lit baa < length a2'f›
  ‹cut-watch-list-heur2-inv baa (length (a1'd ! nat-of-lit baa))
```

$(a1'e, a1'f, a2'f) \Longrightarrow a1'f < \textit{length-ll } a2'f (\textit{nat-of-lit baa}) \Longrightarrow$
$a1'e < \textit{length } (a2'f ! \textit{ nat-of-lit baa})$›
⟨*proof*⟩

sepref-def *cut-watch-list-heur2-fast-code*
is ‹*uncurry3 cut-watch-list-heur2*›
:: ‹$[\lambda(((j, w), L), S).\ \textit{length } (\textit{watched-by-int } S\ L) \leq \textit{sint64-max}-\textit{MIN-HEADER-SIZE}]_a$
$\textit{sint64-nat-assn}^k *_a \textit{sint64-nat-assn}^k *_a \textit{unat-lit-assn}^k *_a$
$\textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-bounded-assn}$›
⟨*proof*⟩

sepref-def *unit-propagation-inner-loop-wl-D-fast-code*
is ‹*uncurry unit-propagation-inner-loop-wl-D-heur*›
:: ‹$[\lambda(L, S).\ \textit{length } (\textit{get-clauses-wl-heur } S) \leq \textit{sint64-max}]_a$
$\textit{unat-lit-assn}^k *_a \textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-bounded-assn}$›
⟨*proof*⟩

sepref-def *select-and-remove-from-literals-to-update-wlfast-code*
is ‹*select-and-remove-from-literals-to-update-wl-heur*›
:: ‹$\textit{isasat-bounded-assn}^d \rightarrow_a \textit{isasat-bounded-assn} \times_a \textit{unat-lit-assn}$›
⟨*proof*⟩

sepref-def *literals-to-update-wl-literals-to-update-wl-empty-fast-code*
is ‹*RETURN o literals-to-update-wl-literals-to-update-wl-empty*›
:: ‹$[\lambda S.\ \textit{isa-length-trail-pre } (\textit{get-trail-wl-heur } S)]_a\ \textit{isasat-bounded-assn}^k \rightarrow \textit{bool1-assn}$›
⟨*proof*⟩

sepref-register *literals-to-update-wl-literals-to-update-wl-empty*
select-and-remove-from-literals-to-update-wl-heur

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
‹$\textit{length } (\textit{get-clauses-wl-heur } x) \leq \textit{sint64-max} \Longrightarrow$
$\textit{unit-propagation-outer-loop-wl-D-heur-inv } x\ s' \Longrightarrow$
$\textit{length } (\textit{get-clauses-wl-heur } a1') =$
$\textit{length } (\textit{get-clauses-wl-heur } s') \Longrightarrow$
$\textit{length } (\textit{get-clauses-wl-heur } s') \leq \textit{sint64-max}$›
⟨*proof*⟩

sepref-def *unit-propagation-outer-loop-wl-D-fast-code*
is ‹*unit-propagation-outer-loop-wl-D-heur*›
:: ‹$[\lambda S.\ \textit{length } (\textit{get-clauses-wl-heur } S) \leq \textit{sint64-max}]_a\ \textit{isasat-bounded-assn}^d \rightarrow \textit{isasat-bounded-assn}$›
⟨*proof*⟩

experiment begin

export-llvm
length-ll-fs-heur-fast-code
unit-propagation-inner-loop-wl-loop-D-fast
cut-watch-list-heur2-fast-code
unit-propagation-inner-loop-wl-D-fast-code
isa-trail-nth-fast-code

```
  select-and-remove-from-literals-to-update-wlfast-code
  literals-to-update-wl-literals-to-update-wl-empty-fast-code
  unit-propagation-outer-loop-wl-D-fast-code

end

end
theory IsaSAT-Decide
  imports IsaSAT-Setup IsaSAT-VMTF
begin
```

Chapter 17

Decide

lemma (**in** $-$)*not-is-None-not-None*: $\langle \neg \mathit{is\text{-}None}\ s \Longrightarrow s \neq \mathit{None}\rangle$
$\langle \mathit{proof}\rangle$

definition *vmtf-find-next-undef-upd*
$:: \langle \mathit{nat\ multiset} \Rightarrow (\mathit{nat},\mathit{nat})\mathit{ann\text{-}lits} \Rightarrow \mathit{vmtf\text{-}remove\text{-}int} \Rightarrow$
$(((\mathit{nat},\mathit{nat})\mathit{ann\text{-}lits} \times \mathit{vmtf\text{-}remove\text{-}int}) \times \mathit{nat\ option})\mathit{nres}\rangle$
where
$\langle \mathit{vmtf\text{-}find\text{-}next\text{-}undef\text{-}upd}\ \mathcal{A} = (\lambda M\ \mathit{vm}.\ \mathit{do}\{$
$L \leftarrow \mathit{vmtf\text{-}find\text{-}next\text{-}undef}\ \mathcal{A}\ \mathit{vm}\ M;$
$\mathit{RETURN}\ ((M,\ \mathit{update\text{-}next\text{-}search}\ L\ \mathit{vm}),\ L)$
$\})\rangle$

definition *isa-vmtf-find-next-undef-upd*
$:: \langle \mathit{trail\text{-}pol} \Rightarrow \mathit{isa\text{-}vmtf\text{-}remove\text{-}int} \Rightarrow$
$((\mathit{trail\text{-}pol} \times \mathit{isa\text{-}vmtf\text{-}remove\text{-}int}) \times \mathit{nat\ option})\mathit{nres}\rangle$
where
$\langle \mathit{isa\text{-}vmtf\text{-}find\text{-}next\text{-}undef\text{-}upd} = (\lambda M\ \mathit{vm}.\ \mathit{do}\{$
$L \leftarrow \mathit{isa\text{-}vmtf\text{-}find\text{-}next\text{-}undef}\ \mathit{vm}\ M;$
$\mathit{RETURN}\ ((M,\ \mathit{update\text{-}next\text{-}search}\ L\ \mathit{vm}),\ L)$
$\})\rangle$

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
$\langle(\mathit{uncurry\ isa\text{-}vmtf\text{-}find\text{-}next\text{-}undef\text{-}upd},\ \mathit{uncurry}\ (\mathit{vmtf\text{-}find\text{-}next\text{-}undef\text{-}upd}\ \mathcal{A})) \in$
$\mathit{trail\text{-}pol}\ \mathcal{A}\ \times_r\ (\mathit{Id} \times_r \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}) \rightarrow_f$
$\langle \mathit{trail\text{-}pol}\ \mathcal{A} \times_f\ (\mathit{Id} \times_r \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}) \times_f\ \langle \mathit{nat\text{-}rel}\rangle \mathit{option\text{-}rel}\rangle \mathit{nres\text{-}rel}\ \rangle$
$\langle \mathit{proof}\rangle$

definition *lit-of-found-atm* **where**
$\langle \mathit{lit\text{-}of\text{-}found\text{-}atm}\ \varphi\ L = \mathit{SPEC}\ (\lambda K.\ (L = \mathit{None} \longrightarrow K = \mathit{None}) \wedge$
$(L \neq \mathit{None} \longrightarrow K \neq \mathit{None} \wedge \mathit{atm\text{-}of}\ (\mathit{the}\ K) = \mathit{the}\ L))\rangle$

definition *find-undefined-atm*
$:: \langle \mathit{nat\ multiset} \Rightarrow (\mathit{nat},\mathit{nat})\ \mathit{ann\text{-}lits} \Rightarrow \mathit{vmtf\text{-}remove\text{-}int} \Rightarrow$
$(((\mathit{nat},\mathit{nat})\ \mathit{ann\text{-}lits} \times \mathit{vmtf\text{-}remove\text{-}int}) \times \mathit{nat\ option})\ \mathit{nres}\rangle$
where
$\langle \mathit{find\text{-}undefined\text{-}atm}\ \mathcal{A}\ M\ \text{-} = \mathit{SPEC}(\lambda((M',\ \mathit{vm}),\ L).$
$(L \neq \mathit{None} \longrightarrow \mathit{Pos}\ (\mathit{the}\ L) \in\!\#\ \mathcal{L}_{all}\ \mathcal{A} \wedge \mathit{undefined\text{-}atm}\ M\ (\mathit{the}\ L)) \wedge$
$(L = \mathit{None} \longrightarrow (\forall K\in\!\#\ \mathcal{L}_{all}\ \mathcal{A}.\ \mathit{defined\text{-}lit}\ M\ K)) \wedge M = M' \wedge \mathit{vm} \in \mathit{vmtf}\ \mathcal{A}\ M)\rangle$

definition *lit-of-found-atm-D-pre* **where**
$\langle \mathit{lit\text{-}of\text{-}found\text{-}atm\text{-}D\text{-}pre} = (\lambda(\varphi,\ L).\ L \neq \mathit{None} \longrightarrow (\mathit{the}\ L < \mathit{length}\ \varphi \wedge \mathit{the}\ L \leq \mathit{uint32\text{-}max\ div\ 2}))\rangle$

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', j, W', vm, clvls, cach, lbd, outl, stats, heur,
     vdom, avdom, lcount, opts, old-arena). do {
    ((M, vm), L) ← isa-vmtf-find-next-undef-upd M vm;
    ASSERT(L ≠ None ⟶ get-saved-phase-heur-pre (the L) heur);
    L ← lit-of-found-atm heur L;
    RETURN ((M, N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
     vdom, avdom, lcount, opts, old-arena), L)
  })›
```

lemma *lit-of-found-atm-D-pre*:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹ (*L* ≠ *None* ⟹ *the L* ∈# $\mathcal{A}$) ⟹
L ≠ *None* ⟹ *get-saved-phase-heur-pre* (*the L*) *heur*›
⟨*proof*⟩

definition *find-unassigned-lit-wl-D-heur-pre* **where**

```
‹find-unassigned-lit-wl-D-heur-pre S ⟷
  (
    ∃ T U.
      (S, T) ∈ state-wl-l None ∧
      (T, U) ∈ twl-st-l None ∧
      twl-struct-invs U ∧
      literals-are-$\mathcal{L}_{in}$ (all-atms-st S) S ∧
      get-conflict-wl S = None
  )›
```

lemma *vmtf-find-next-undef-upd*:
‹(*uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$), *uncurry* (*find-undefined-atm* $\mathcal{A}$)) ∈
[λ(*M*, *vm*). *vm* ∈ *vmtf* $\mathcal{A}$ *M*]$_f$ *Id* ×$_f$ *Id* → ⟨*Id* ×$_f$ *Id* ×$_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D*:
‹(*find-unassigned-lit-wl-D-heur*, *find-unassigned-lit-wl*) ∈
[*find-unassigned-lit-wl-D-heur-pre*]$_f$
twl-st-heur''' *r* → ⟨{((*T*, *L*), (*T'*, *L'*)). (*T*, *T'*) ∈ *twl-st-heur'''* *r* ∧ *L* = *L'* ∧
(*L* ≠ *None* ⟶ *undefined-lit* (*get-trail-wl T'*) (*the L*) ∧ *the L* ∈# $\mathcal{L}_{all}$ (*all-atms-st T'*)) ∧
get-conflict-wl T' = *None*}⟩*nres-rel*›
⟨*proof*⟩

definition *lit-of-found-atm-D*
:: ‹*bool list* ⇒ *nat option* ⇒ (*nat literal option*)*nres*› **where**

```
‹lit-of-found-atm-D = (λ(φ::bool list) L. do{
    case L of
      None ⇒ RETURN None
    | Some L ⇒ do {
        ASSERT (L<length φ);
        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 ×_f Id → ⟨Id⟩nres-rel›
  ⟨proof⟩
```

```
definition decide-lit-wl-heur :: ‹nat literal ⇒ twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹decide-lit-wl-heur = (λL' (M, N, D, Q, W, vmtf, 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, clvls, cach, lbd, outl, incr-decision stats,
         fema, sema)})›
```

```
definition mop-get-saved-phase-heur-st :: ‹nat ⇒ twl-st-wl-heur ⇒ bool nres› where
   ‹mop-get-saved-phase-heur-st =
     (λL (M', N', D', Q', W', vm, clvls, cach, lbd, outl, stats, heur, vdom, avdom, lcount, opts,
       old-arena).
      mop-get-saved-phase-heur L heur)›
```

```
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) ∈ twl-st-heur''' r →_f ⟨bool-rel ×_f twl-st-heur''' r⟩ nres-rel›
⟨proof⟩
```

```
lemma bind-triple-unfold:
  ‹do {
    ((M, vm), L) ← (P :: - nres);
    f ((M, vm), L)
} =
do {
    x ← P;
    f x
}›
  ⟨proof⟩
```

```
definition decide-wl-or-skip-D-heur' where
  ‹decide-wl-or-skip-D-heur' = (λ(M, N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena). do {
      ((M, vm), L) ← isa-vmtf-find-next-undef-upd M vm;
      ASSERT(L ≠ None ⟶ get-saved-phase-heur-pre (the L) heur);
      case L of
       None ⇒ RETURN (True, (M, N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts, old-arena))
```

```
    | Some L ⇒ do {
       b ← mop-get-saved-phase-heur L heur;
       let L = (if b then Pos L else Neg L);
       T ← decide-lit-wl-heur L (M, N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
         vdom, avdom, lcount, opts, old-arena);
       RETURN (False, T)
     }
   })
⟩
```

lemma *decide-wl-or-skip-D-heur'-decide-wl-or-skip-D-heur*:
⟨*decide-wl-or-skip-D-heur' S* $\leq$ ⇓*Id* (*decide-wl-or-skip-D-heur S*)⟩
⟨*proof*⟩

lemma *decide-wl-or-skip-D-heur'-decide-wl-or-skip-D-heur2*:
⟨(*decide-wl-or-skip-D-heur'*, *decide-wl-or-skip-D-heur*) ∈ *Id* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*⟩
⟨*proof*⟩

end
theory *IsaSAT-Decide-LLVM*
imports *IsaSAT-Decide IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM IsaSAT-Rephase-LLVM*
begin

sepref-def *decide-lit-wl-fast-code*
is ⟨*uncurry decide-lit-wl-heur*⟩
:: ⟨*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
⟨*proof*⟩

sepref-register *find-unassigned-lit-wl-D-heur decide-lit-wl-heur*

sepref-register *isa-vmtf-find-next-undef*

sepref-def *isa-vmtf-find-next-undef-code* **is**
⟨*uncurry isa-vmtf-find-next-undef*⟩ :: ⟨*vmtf-remove-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *atom.option-assn*⟩
⟨*proof*⟩

sepref-register *update-next-search*
sepref-def *update-next-search-code* **is**
⟨*uncurry* (*RETURN oo update-next-search*)⟩ :: ⟨*atom.option-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$ *vmtf-remove-assn*⟩
⟨*proof*⟩

sepref-register *isa-vmtf-find-next-undef-upd mop-get-saved-phase-heur*
sepref-def *isa-vmtf-find-next-undef-upd-code* **is**
⟨*uncurry isa-vmtf-find-next-undef-upd*⟩
:: ⟨*trail-pol-fast-assn*^{d} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$ (*trail-pol-fast-assn* $\times_a$ *vmtf-remove-assn*) $\times_a$ *atom.option-assn*⟩
⟨*proof*⟩

lemma *mop-get-saved-phase-heur-alt-def*:

```
⟨mop-get-saved-phase-heur = (λL (fast-ema, slow-ema, res-info, wasted, φ, target, best). do {
          ASSERT (L < length φ);
          RETURN (φ ! L)
        })⟩
```

⟨*proof*⟩

sepref-def *mop-get-saved-phase-heur-impl*
  is ‹*uncurry mop-get-saved-phase-heur*›
  :: ‹*atom-assn*^{k} $*_a$ *heuristic-assn*^{k} $\rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

sepref-def *decide-wl-or-skip-D-fast-code*
  is ‹*decide-wl-or-skip-D-heur*›
  :: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

experiment begin

export-llvm
  decide-lit-wl-fast-code
  isa-vmtf-find-next-undef-code
  update-next-search-code
  isa-vmtf-find-next-undef-upd-code
  decide-wl-or-skip-D-fast-code

end

end
theory *IsaSAT-CDCL*
  imports *IsaSAT-Propagate-Conflict IsaSAT-Conflict-Analysis IsaSAT-Backtrack*
    IsaSAT-Decide IsaSAT-Show
begin

Chapter 18

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 > 0
        then do {
          T ← skip-and-resolve-loop-wl-D-heur S;
          ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur T));
          U ← backtrack-wl-D-nlit-heur T;
          U ← isasat-current-status U; — Print some information every once in a while
          RETURN (False, U)
        }
        else RETURN (True, S)
      }
    }
  ›
```

```
lemma twl-st-heur''D-twl-st-heurD:
  assumes H: ‹(⋀𝒟 r. f ∈ twl-st-heur'' 𝒟 r →f ⟨twl-st-heur'' 𝒟 r⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →f ⟨twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
⟨proof⟩
```

```
lemma twl-st-heur'''D-twl-st-heurD:
  assumes H: ‹(⋀r. f ∈ twl-st-heur''' r →f ⟨twl-st-heur''' r⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →f ⟨twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
⟨proof⟩
```

```
lemma twl-st-heur'''D-twl-st-heurD-prod:
  assumes H: ‹(⋀r. f ∈ twl-st-heur''' r →f ⟨A ×r twl-st-heur''' r⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →f ⟨A ×r twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
⟨proof⟩
```

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*:

```
 ‹(cdcl-twl-o-prog-wl-D-heur, cdcl-twl-o-prog-wl) ∈
  {(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) = r} →f
    ⟨bool-rel ×f {(S, T). (S, T) ∈ twl-st-heur ∧
       length (get-clauses-wl-heur S) ≤ r + MAX-HEADER-SIZE+1 + uint32-max div 2}⟩nres-rel›
⟨proof⟩
```

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2*:

```
 ‹(cdcl-twl-o-prog-wl-D-heur, cdcl-twl-o-prog-wl) ∈
  {(S, T). (S, T) ∈ twl-st-heur} →f
    ⟨bool-rel ×f {(S, T). (S, T) ∈ twl-st-heur}⟩nres-rel›
 ⟨proof⟩
```

Combining Together: Full Strategy **definition** *cdcl-twl-stgy-prog-wl-D-heur*

```
 :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
where
 ‹cdcl-twl-stgy-prog-wl-D-heur S0 =
 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, S0);
     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) ∈
    twl-st-heur →f ⟨twl-st-heur⟩ nres-rel›
 ⟨proof⟩
```

lemma *cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D*:

```
 ‹(cdcl-twl-stgy-prog-wl-D-heur, cdcl-twl-stgy-prog-wl) ∈ twl-st-heur →f ⟨twl-st-heur⟩nres-rel›
⟨proof⟩
```

definition *cdcl-twl-stgy-prog-break-wl-D-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›

```
where
 ‹cdcl-twl-stgy-prog-break-wl-D-heur S0 =
 do {
   b ← RETURN (isasat-fast S0);
   (b, brk, T) ← WHILE_T^λ(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, S₀);
  if brk then RETURN T
  else cdcl-twl-stgy-prog-wl-D-heur T
})›
```

definition *cdcl-twl-stgy-prog-bounded-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where

```
‹cdcl-twl-stgy-prog-bounded-wl-heur S₀ =
do {
  b ← RETURN (isasat-fast S₀);
  (b, brk, T) ← WHILE_T^λ(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, S₀);
  RETURN (brk, T)
})›
```

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
  assumes *r*: ‹*r* ≤ *sint64-max*›
  shows ‹(*cdcl-twl-stgy-prog-bounded-wl-heur*, *cdcl-twl-stgy-prog-early-wl*) ∈
    twl-st-heur''' *r* $\rightarrow_f$ ⟨*bool-rel* $\times_r$ *twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-CDCL-LLVM*
  imports *IsaSAT-CDCL IsaSAT-Propagate-Conflict-LLVM IsaSAT-Conflict-Analysis-LLVM*
    IsaSAT-Backtrack-LLVM
    IsaSAT-Decide-LLVM IsaSAT-Show-LLVM
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-def *cdcl-twl-o-prog-wl-D-fast-code*
  is ‹*cdcl-twl-o-prog-wl-D-heur*›
  :: ‹$[isasat\text{-}fast]_a$
    $isasat\text{-}bounded\text{-}assn^d$ → *bool1-assn* $\times_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

declare
  cdcl-twl-o-prog-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-register *unit-propagation-outer-loop-wl-D-heur*
cdcl-twl-o-prog-wl-D-heur

definition *length-clauses-heur* **where**
‹*length-clauses-heur S* = *length* (*get-clauses-wl-heur S*)›

lemma *length-clauses-heur-alt-def*: ‹*length-clauses-heur* = (λ(*M*, *N*, -). *length N*)›
⟨*proof*⟩

sepref-def *length-clauses-heur-impl*
is ‹*RETURN o length-clauses-heur*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

declare *length-clauses-heur-impl.refine* [*sepref-fr-rules*]

lemma *isasat-fast-alt-def*: ‹*isasat-fast S* = (*length-clauses-heur S* ≤ *9223372034707292156*)›
⟨*proof*⟩

sepref-def *isasat-fast-impl*
is ‹*RETURN o isasat-fast*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

declare *isasat-fast-impl.refine*[*sepref-fr-rules*]

sepref-def *cdcl-twl-stgy-prog-wl-D-code*
is ‹*cdcl-twl-stgy-prog-bounded-wl-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *isasat-bounded-assn*›
⟨*proof*⟩

declare *cdcl-twl-stgy-prog-wl-D-code.refine*[*sepref-fr-rules*]

export-llvm *cdcl-twl-stgy-prog-wl-D-code* **file** ‹*code/isasat.ll*›

end
theory *IsaSAT-Restart-Heuristics*
imports
Watched-Literals.WB-Sort Watched-Literals.Watched-Literals-Watch-List-Restart IsaSAT-Rephase
IsaSAT-Setup IsaSAT-VMTF IsaSAT-Sorting
begin

Chapter 19

Restarts

lemma *twl-st-heur-change-subsumed-clauses*:
  assumes ‹((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
      vdom, *avdom*, *lcount*, *opts*, *old-arena*),
    (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)) ∈ *twl-st-heur*›
  ‹*set-mset* (*all-atms N* ((*NE*+*UE*)+(*NS*+*US*))) = *set-mset* (*all-atms N* ((*NE*+*UE*)+(*NS′*+*US′*)))›
  shows ‹((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
      vdom, *avdom*, *lcount*, *opts*, *old-arena*),
    (*M*, *N*, *D*, *NE*, *UE*, *NS′*, *US′*, *Q*, *W*)) ∈ *twl-st-heur*›
⟨*proof*⟩

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*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
      vdom, *avdom*, *lcount*, *opts*, *old-arena*),
    (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)).
   (*M*′, *M*) ∈ *trail-pol* (*all-init-atms N* (*NE*+*NS*)) ∧
   valid-arena N′ *N* (*set vdom*) ∧
   (*D*′, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N* (*NE*+*NS*)) ∧
   (*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*+*NS*))) ∧
   vm ∈ *isa-vmtf* (*all-init-atms N* (*NE*+*NS*)) *M* ∧
   no-dup M ∧
   clvls ∈ *counts-maximum-level M D* ∧
   cach-refinement-empty (*all-init-atms N* (*NE*+*NS*)) *cach* ∧
   out-learned M D outl ∧
   lcount = *size* (*learned-clss-lf N*) ∧
   vdom-m (*all-init-atms N* (*NE*+*NS*)) *W N* ⊆ *set vdom* ∧
   mset avdom ⊆# *mset vdom* ∧
   isasat-input-bounded (*all-init-atms N* (*NE*+*NS*)) ∧
   isasat-input-nempty (*all-init-atms N* (*NE*+*NS*)) ∧
   distinct vdom ∧ *old-arena* = [] ∧
   heuristic-rel (*all-init-atms N* (*NE*+*NS*)) *heur*
  }›

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*›
  ⟨*proof*⟩

lemma *twl-st-heur-restartD*:
  ‹*x* ∈ *twl-st-heur-restart* ⟹ *x* ∈ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur* (*fst x*)))›
  ⟨*proof*⟩

definition *clause-score-ordering2* **where**
  ‹*clause-score-ordering2* = (λ(*lbd*, *act*) (*lbd*′, *act*′). *lbd* < *lbd*′ ∨ (*lbd* = *lbd*′ ∧ *act* ≤ *act*′))›

lemma *unbounded-id*: ‹*unbounded* (*id* :: *nat* ⇒ *nat*)›
  ⟨*proof*⟩

global-interpretation *twl-restart-ops id*
  ⟨*proof*⟩

```
global-interpretation twl-restart id
  ⟨proof⟩
```

We first fix the function that proves termination. We don't take the "smallest" function possible (other possibilites that are growing slower include $\lambda n.\ n >> 50$). Remark that this scheme is not compatible with Luby (TODO: use Luby restart scheme every once in a while like CryptoMinisat?)

```
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-local-restart-target-level-int-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+1)
         })
        (False, 0);
    RETURN i
   })›

definition find-local-restart-target-level where
  ‹find-local-restart-target-level M - = SPEC(λi. i ≤ count-decided M)›

lemma find-local-restart-target-level-alt-def:
  ‹find-local-restart-target-level M vm = do {
      (b, i) ← SPEC(λ(b::bool, i). i ≤ count-decided M);
       RETURN i
    }›
  ⟨proof⟩


lemma find-local-restart-target-level-int-find-local-restart-target-level:
   ‹(uncurry find-local-restart-target-level-int, uncurry find-local-restart-target-level) ∈
      [λ(M, vm). vm ∈ isa-vmtf 𝒜 M]_f trail-pol 𝒜 ×_r Id → ⟨nat-rel⟩nres-rel›
  ⟨proof⟩

definition empty-Q :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹empty-Q = (λ(M, N, D, Q, W, vm, clvls, cach, lbd, outl, stats, (fema, sema, ccount, wasted), vdom,
      lcount). do{
    j ← mop-isa-length-trail M;
    RETURN (M, N, D, j, W, vm, clvls, cach, lbd, outl, stats, (fema, sema,
       restart-info-restart-done ccount, wasted), vdom, 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-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, clvls, cach, lbd, stats). do {
      find-local-restart-target-level-int M vm})›
 ⟨proof⟩

definition cdcl-twl-local-restart-wl-D-heur
   :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
where
  ‹cdcl-twl-local-restart-wl-D-heur = (λS. do {
      ASSERT(restart-abs-wl-heur-pre S False);
      lvl ← find-local-restart-target-level-st S;
      if lvl = count-decided-st-heur S
      then RETURN S
      else do {
        S ← find-decomp-wl-st-int lvl S;
        S ← empty-Q S;
        incr-lrestart-stat S
      }
   })›


named-theorems twl-st-heur-restart

lemma [twl-st-heur-restart]:
  assumes ‹(S, T) ∈ twl-st-heur-restart›
  shows ‹(get-trail-wl-heur S, get-trail-wl T) ∈ trail-pol (all-init-atms-st T)›
  ⟨proof⟩

lemma trail-pol-literals-are-in-ℒin-trail:
  ‹(M', M) ∈ trail-pol 𝒜 ⟹ literals-are-in-ℒin-trail 𝒜 M›
  ⟨proof⟩

lemma refine-generalise1: ‹A ≤ B ⟹ do {x ← B; C x} ≤ D ⟹ do {x ← A; C x} ≤ (D:: 'a nres)›
  ⟨proof⟩

lemma refine-generalise2: A ≤ B ⟹ do {x ← do {x ← B; A' x}; C x} ≤ D ⟹
  do {x ← do {x ← A; A' x}; C x} ≤ (D:: 'a nres)
  ⟨proof⟩

lemma cdcl-twl-local-restart-wl-D-spec-int:
  ‹cdcl-twl-local-restart-wl-spec (M, N, D, NE, UE, NS, US, Q, W) ≥ ( do {
      ASSERT(restart-abs-wl-pre (M, N, D, NE, UE, NS, US, Q, W) False);
      i ← SPEC(λ-. True);
      if i
      then RETURN (M, N, D, NE, UE, NS, {#}, Q, W)
      else do {
        (M, Q') ← SPEC(λ(M', Q'). (∃K M2. (Decided K # M', M2) ∈ set (get-all-ann-decomposition
M) ∧
              Q' = {#}) ∨ (M' = M ∧ Q' = Q));
        RETURN (M, N, D, NE, UE, NS, {#}, Q', W)
      }
    })›
⟨proof⟩
```

lemma *trail-pol-no-dup*: ‹$(M, M') \in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *no-dup* M'›
  ⟨*proof*⟩

lemma *heuristic-rel-restart-info-done*[*intro*!, *simp*]:
  ‹*heuristic-rel* $\mathcal{A}$ (*fema*, *sema*, *ccount*, *wasted*) $\Longrightarrow$
    heuristic-rel $\mathcal{A}$ ((*fema*, *sema*, *restart-info-restart-done ccount*, *wasted*))›
  ⟨*proof*⟩

lemma *cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec*:
  ‹(*cdcl-twl-local-restart-wl-D-heur*, *cdcl-twl-local-restart-wl-spec*) $\in$
    twl-st-heur''' $r \rightarrow_f$ ⟨*twl-st-heur'''* r⟩*nres-rel*›
⟨*proof*⟩

definition *remove-all-annot-true-clause-imp-wl-D-heur-inv*
  :: ‹*twl-st-wl-heur* ⇒ *nat watcher list* ⇒ *nat* × *twl-st-wl-heur* ⇒ *bool*›
where

```
  ‹remove-all-annot-true-clause-imp-wl-D-heur-inv S xs = (λ(i, T).
      ∃S' T'. (S, S') ∈ twl-st-heur-restart ∧ (T, T') ∈ twl-st-heur-restart ∧
        remove-all-annot-true-clause-imp-wl-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-pre*
  :: ‹*nat multiset* ⇒ *nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*›
where
  ‹*remove-all-annot-true-clause-imp-wl-D-pre* $\mathcal{A}$ L $S \longleftrightarrow (L \in\# \mathcal{L}_{all}\ \mathcal{A})$›

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 = (λL (M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
      stats, heur, vdom, avdom, lcount, opts). do {
    ASSERT(remove-all-annot-true-clause-imp-wl-D-heur-pre L (M, N0, D, Q, W, vm, clvls,
      cach, lbd, outl, stats, heur,
      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, clvls, cach, lbd, outl, stats,
   heur, 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, clvls, cach, lbd, outl, stats,
       heur, vdom, avdom, lcount, opts) xs
           (i, M, N, D, Q, W, vm, clvls, cach, lbd, outl, stats,
       heur, 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, clvls, cach, lbd, outl, stats,
   heur, vdom, avdom, lcount', opts)
  })›
```

```
definition minimum-number-between-restarts :: ‹64 word› where
  ‹minimum-number-between-restarts = 50›
```

```
definition five-uint64 :: ‹64 word› 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, clvls, cach, lbd, outl,
    (props, decs, confl, restarts, -), heur, vdom, avdom, lcount, opts).
    of-nat lcount < 3000 + 1000 * 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, clvls, cach, lbd, outl,
       (props, decs, confl, restarts, -), heur,
      vdom, avdom, lcount, opts, old).
    (¬opts-reduce opts ∨ (opts-restart opts ∧ (of-nat lcount < 2000 + 1000 * restarts))))›
```

```
definition reorder-vdom-wl :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres› where
  ‹reorder-vdom-wl S = RETURN S›
```

```
definition sort-clauses-by-score :: ‹arena ⇒ nat list ⇒ nat list nres› where
  ‹sort-clauses-by-score arena vdom = do {
      ASSERT(∀ i∈set vdom. valid-sort-clause-score-pre-at arena i);
      SPEC(λvdom'. mset vdom = mset vdom')
  }›
```

```
definition (in −) quicksort-clauses-by-score :: ‹arena ⇒ nat list ⇒ nat list nres› where
  ‹quicksort-clauses-by-score arena =
    full-quicksort-ref clause-score-ordering2 (clause-score-extract arena)›
```

```
lemma quicksort-clauses-by-score-sort:
 ‹(quicksort-clauses-by-score, sort-clauses-by-score) ∈
   Id → Id → ⟨Id⟩nres-rel›
   ⟨proof⟩
```

```
definition remove-deleted-clauses-from-avdom :: ‹-› where
‹remove-deleted-clauses-from-avdom N avdom0 = do {
  let n = length avdom0;
 (i, j, avdom) ← WHILE_T λ(i, j, avdom). i ≤ j ∧ j ≤ n ∧ length avdom = length avdom0 ∧          mset (take i avdom @ dro
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < length avdom);
      if (avdom ! j) ∈# dom-m N then RETURN (i+1, j+1, swap avdom i j)
      else RETURN (i, j+1, avdom)
    })
    (0, 0, avdom0);
  ASSERT(i ≤ length avdom);
  RETURN (take i avdom)
}›
```

```
lemma remove-deleted-clauses-from-avdom:
  ‹remove-deleted-clauses-from-avdom N avdom0 ≤ SPEC(λavdom. mset avdom ⊆# mset avdom0)›
  ⟨proof⟩
```

```
definition isa-remove-deleted-clauses-from-avdom :: ‹-› where
‹isa-remove-deleted-clauses-from-avdom arena avdom0 = do {
  ASSERT(length avdom0 ≤ length arena);
  let n = length avdom0;
  (i, j, avdom) ← WHILE_T λ(i, j, -). i ≤ j ∧ j ≤ n
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < n);
      ASSERT(arena-is-valid-clause-vdom arena (avdom!j) ∧ j < length avdom ∧ i < length avdom);
      if arena-status arena (avdom ! j) ≠ DELETED then RETURN (i+1, j+1, swap avdom i j)
      else RETURN (i, j+1, avdom)
    }) (0, 0, avdom0);
  ASSERT(i ≤ length avdom);
  RETURN (take i avdom)
}›
```

```
lemma isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom:
   ‹valid-arena arena N (set vdom) ⟹ mset avdom0 ⊆# mset vdom ⟹ distinct vdom ⟹
    isa-remove-deleted-clauses-from-avdom arena avdom0 ≤ ⇓Id (remove-deleted-clauses-from-avdom N
avdom0)›
  ⟨proof⟩
```

```
definition (in −) sort-vdom-heur :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
  ‹sort-vdom-heur = (λ(M', arena, D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       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 ← *sort-clauses-by-score arena avdom*;
RETURN (*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*)
})›

lemma *sort-clauses-by-score-reorder*:
‹*valid-arena arena N* (*set vdom'*) ⟹ *set vdom* ⊆ *set vdom'* ⟹
sort-clauses-by-score arena vdom ≤ *SPEC*(λ*vdom'*. *mset vdom* = *mset vdom'*)›
⟨*proof*⟩

lemma *sort-vdom-heur-reorder-vdom-wl*:
‹(*sort-vdom-heur*, *reorder-vdom-wl*) ∈ *twl-st-heur-restart-ana r* $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

lemma (**in** −) *insort-inner-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a ba* ⟹
mset ba = *mset a2'* ⟹
a1' < *length a2'* ⟹
valid-sort-clause-score-pre-at a (*a2'* ! *a1'*)›
⟨*proof*⟩

lemma *sort-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a b* ⟹
mset b = *mset a2'* ⟹ *valid-sort-clause-score-pre a a2'*›
⟨*proof*⟩

definition *partition-main-clause* **where**
‹*partition-main-clause arena* = *partition-main clause-score-ordering* (*clause-score-extract arena*)›

definition *partition-clause* **where**
‹*partition-clause arena* = *partition-between-ref clause-score-ordering* (*clause-score-extract arena*)›

lemma *valid-sort-clause-score-pre-swap*:
‹*valid-sort-clause-score-pre a b* ⟹ *x* < *length b* ⟹
ba < *length b* ⟹ *valid-sort-clause-score-pre a* (*swap b x ba*)›
⟨*proof*⟩

definition *div2* **where** [*simp*]: ‹*div2 n* = *n div 2*›

definition *safe-minus* **where** ‹*safe-minus a b* = (*if b* ≥ *a then 0 else a* − *b*)›

definition *max-restart-decision-lvl* :: *nat* **where**
‹*max-restart-decision-lvl* = *300*›

definition *max-restart-decision-lvl-code* :: ‹*32 word*› **where**
‹*max-restart-decision-lvl-code* = *300*›

fun (**in** −) *get-reductions-count* :: ‹*twl-st-wl-heur* ⇒ *64 word*› **where**
‹*get-reductions-count* (-, -, -, -, -, -, -,-,-,-,
(-, -, -, *lres*, -, -), -)
= *lres*›

```
definition get-restart-phase :: ‹twl-st-wl-heur ⇒ 64 word› where
  ‹get-restart-phase = (λ(-, -, -, -, -, -, -, -, -, -, -, heur, -).
     current-restart-phase heur)›
```

```
definition GC-required-heur :: ‹twl-st-wl-heur ⇒ nat ⇒ bool nres› where
  ‹GC-required-heur S n = do {
    n ← RETURN (full-arena-length-st S);
    wasted ← RETURN (wasted-bytes-st S);
    RETURN (3∗wasted > ((of-nat n)>>2))
 }›
```

```
definition FLAG-no-restart :: ‹8 word› where
  ‹FLAG-no-restart = 0›
```

```
definition FLAG-restart :: ‹8 word› where
  ‹FLAG-restart = 1›
```

```
definition FLAG-GC-restart :: ‹8 word› where
  ‹FLAG-GC-restart = 2›
```

```
definition restart-flag-rel :: ‹(8 word × restart-type) set› where
 ‹restart-flag-rel = {(FLAG-no-restart, NO-RESTART), (FLAG-restart, RESTART), (FLAG-GC-restart,
GC)}›
```

```
definition restart-required-heur :: ‹twl-st-wl-heur ⇒ nat ⇒ 8 word nres› where
  ‹restart-required-heur S n = do {
    let opt-red = opts-reduction-st S;
    let opt-res = opts-restart-st S;
    let curr-phase = get-restart-phase S;
    let lcount = get-learned-count S;
    let can-res = (lcount > n);

    if ¬can-res ∨ ¬opt-res ∨ ¬opt-red then RETURN FLAG-no-restart
    else if curr-phase = QUIET-PHASE
    then do {
      GC-required ← GC-required-heur S n;
      let upper = upper-restart-bound-not-reached S;
      if (opt-res ∨ opt-red) ∧ ¬upper
      then RETURN FLAG-GC-restart
      else RETURN FLAG-no-restart
    }
    else do {
      let sema = ema-get-value (get-slow-ema-heur S);
      let limit = (shiftr (11 ∗ sema) (4::nat));
      let fema = ema-get-value (get-fast-ema-heur S);
      let ccount = get-conflict-count-since-last-restart-heur S;
      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);
      let should-reduce = ((opt-res ∨ opt-red) ∧
         (should-not-reduce ⟶ limit > fema) ∧ min-reached ∧ can-res ∧
          level > 2 ∧ ~~This comment from Marijn Heule seems not to help: term ‹level ≤
max-restart-decision-lvl›~~
        of-nat level > (shiftr fema 32));
      GC-required ← GC-required-heur S n;
      if should-reduce
      then if GC-required
        then RETURN FLAG-GC-restart
        else RETURN FLAG-restart
      else RETURN FLAG-no-restart
    }
  }⟩
```

```
lemma (in −) get-reduction-count-alt-def:
  ⟨RETURN o get-reductions-count = (λ(M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
      (-, -, -, lres, -, -), heur, lcount). RETURN lres)⟩
  ⟨proof⟩
```

```
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-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 ∧
        blits-in-ℒ_in S0 ∧
        blits-in-ℒ_in S ∧
        (∃ U0 U. (T0, U0) ∈ twl-st-l None ∧
              (T, U) ∈ twl-st-l None ∧
              remove-one-annot-true-clause** T0 T ∧
              twl-list-invs T0 ∧
              twl-struct-invs U0 ∧
              twl-list-invs T ∧
              twl-struct-invs U ∧
              get-conflict-l T0 = None ∧
        clauses-to-update-l T0 = {#}) ∧
        correct-watching S0 ∧ correct-watching S)⟩
  ⟨proof⟩
```

```
lemma mark-to-delete-clauses-wl-D-heur-pre-alt-def:
    ⟨mark-to-delete-clauses-wl-D-heur-pre S ⟷
      (∃S'. (S, S') ∈ twl-st-heur ∧ mark-to-delete-clauses-wl-pre S')⟩ (is ?A) and
    mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur:
      ⟨mark-to-delete-clauses-wl-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⟩
```

```
lemma mark-garbage-heur-wl:
  assumes
    ⟨(S, T) ∈ twl-st-heur-restart⟩ and
    ⟨C ∈# dom-m (get-clauses-wl T)⟩ and
    ⟨¬ irred (get-clauses-wl T) C⟩ and ⟨i < length (get-avdom S)⟩
  shows ⟨(mark-garbage-heur C i S, mark-garbage-wl C T) ∈ twl-st-heur-restart⟩
```

⟨*proof*⟩

lemma *mark-garbage-heur-wl-ana*:
  assumes
    ‹$(S, T) \in$ *twl-st-heur-restart-ana r*› **and**
    ‹$C \in\#$ *dom-m* (*get-clauses-wl T*)› **and**
    ‹¬ *irred* (*get-clauses-wl T*) *C*› **and** ‹$i <$ *length* (*get-avdom S*)›
  shows ‹(*mark-garbage-heur C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart-ana r*›
  ⟨*proof*⟩

lemma *mark-unused-st-heur-ana*:
  assumes
    ‹$(S, T) \in$ *twl-st-heur-restart-ana r*› **and**
    ‹$C \in\#$ *dom-m* (*get-clauses-wl T*)›
  shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart-ana r*›
  ⟨*proof*⟩

lemma *twl-st-heur-restart-valid-arena*[*twl-st-heur-restart*]:
  assumes
    ‹$(S, T) \in$ *twl-st-heur-restart*›
  shows ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
  ⟨*proof*⟩

lemma *twl-st-heur-restart-get-avdom-nth-get-vdom*[*twl-st-heur-restart*]:
  assumes
    ‹$(S, T) \in$ *twl-st-heur-restart*› ‹$i <$ *length* (*get-avdom S*)›
  shows ‹*get-avdom S* ! *i* ∈ *set* (*get-vdom S*)›
  ⟨*proof*⟩

lemma [*twl-st-heur-restart*]:
  assumes
    ‹$(S, T) \in$ *twl-st-heur-restart*› **and**
    ‹$C \in$ *set* (*get-avdom S*)›
  shows ‹*clause-not-marked-to-delete-heur S C* ⟷
        ($C \in\#$ *dom-m* (*get-clauses-wl T*))› **and**
    ‹$C \in\#$ *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›**and**
     ‹$C \in\#$ *dom-m* (*get-clauses-wl T*) ⟹ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
    ‹$C \in\#$ *dom-m* (*get-clauses-wl T*) ⟹ *arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
⟨*proof*⟩

definition *number-clss-to-keep* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
  ‹*number-clss-to-keep* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
      (*props*, *decs*, *confl*, *restarts*, -), *heur*,
        vdom, *avdom*, *lcount*).
    RES UNIV)›

definition *number-clss-to-keep-impl* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
  ‹*number-clss-to-keep-impl* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
      (*props*, *decs*, *confl*, *restarts*, -), *heur*,
        vdom, *avdom*, *lcount*).
    let n = *unat* (*1000* + *150* ∗ *restarts*) *in RETURN* (*if n* ≥ *sint64-max then sint64-max else n*))›

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹(*number-clss-to-keep-impl*, *number-clss-to-keep*) ∈ *Id* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *MINIMUM-DELETION-LBD* :: *nat* **where**
‹*MINIMUM-DELETION-LBD = 3*›

lemma *in-set-delete-index-and-swapD*:
‹*x* ∈ *set* (*delete-index-and-swap xs i*) ⟹ *x* ∈ *set xs*›
⟨*proof*⟩

lemma *delete-index-vdom-heur-twl-st-heur-restart*:
‹(*S*, *T*) ∈ *twl-st-heur-restart* ⟹ *i* < *length* (*get-avdom S*) ⟹
  (*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart*›
⟨*proof*⟩

lemma *delete-index-vdom-heur-twl-st-heur-restart-ana*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *i* < *length* (*get-avdom S*) ⟹
  (*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart-ana r*›
⟨*proof*⟩

definition *mark-clauses-as-unused-wl-D-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where

```
‹mark-clauses-as-unused-wl-D-heur  = (λi S. do {
    (-, T) ← WHILE_T
      (λ(i, S). i < length (get-avdom S))
      (λ(i, T). do {
        ASSERT(i < length (get-avdom T));
        ASSERT(length (get-avdom T) ≤ length (get-avdom S));
        ASSERT(access-vdom-at-pre T i);
        let C = get-avdom T ! i;
        ASSERT(clause-not-marked-to-delete-heur-pre (T, C));
        if ¬clause-not-marked-to-delete-heur T C then RETURN (i, delete-index-vdom-heur i T)
        else do {
          ASSERT(arena-act-pre (get-clauses-wl-heur T) C);
          RETURN (i+1, (mark-unused-st-heur C T))
        }
      })
      (i, S);
    RETURN T
  })›
```

lemma *avdom-delete-index-vdom-heur*[*simp*]:
‹*get-avdom* (*delete-index-vdom-heur i S*) =
  delete-index-and-swap (*get-avdom S*) *i*›
⟨*proof*⟩

lemma *incr-wasted-st*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*›
shows ‹(*incr-wasted-st C S*, *T*) ∈ *twl-st-heur-restart-ana r*›
⟨*proof*⟩

lemma *incr-wasted-st-twl-st*[*simp*]:
⟨*get-avdom* (*incr-wasted-st w T*) = *get-avdom T*⟩
⟨*get-vdom* (*incr-wasted-st w T*) = *get-vdom T*⟩
⟨*get-trail-wl-heur* (*incr-wasted-st w T*) = *get-trail-wl-heur T*⟩
⟨*get-clauses-wl-heur* (*incr-wasted-st C T*) = *get-clauses-wl-heur T*⟩
⟨*get-conflict-wl-heur* (*incr-wasted-st C T*) = *get-conflict-wl-heur T*⟩
⟨*get-learned-count* (*incr-wasted-st C T*) = *get-learned-count T*⟩
⟨*get-conflict-count-heur* (*incr-wasted-st C T*) = *get-conflict-count-heur T*⟩
⟨*proof*⟩

lemma *mark-clauses-as-unused-wl-D-heur*:
assumes ⟨(*S*, *T*) ∈ *twl-st-heur-restart-ana r*⟩
shows ⟨*mark-clauses-as-unused-wl-D-heur i S* ≤ ⇓ (*twl-st-heur-restart-ana r*) (*SPEC* ((=) *T*))⟩
⟨*proof*⟩

definition *mark-to-delete-clauses-wl-D-heur*
:: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩
where

```
⟨mark-to-delete-clauses-wl-D-heur  = (λS0. do {
    ASSERT(mark-to-delete-clauses-wl-D-heur-pre S0);
    S ← sort-vdom-heur S0;
    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));
        b ← mop-clause-not-marked-to-delete-heur T C;
        if ¬b 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));
          L ← mop-access-lit-in-clauses-heur T C 0;
          D ← get-the-propagation-reason-pol (get-trail-wl-heur T) L;
          lbd ← mop-arena-lbd (get-clauses-wl-heur T) C;
          length ← mop-arena-length (get-clauses-wl-heur T) C;
          status ← mop-arena-status (get-clauses-wl-heur T) C;
          used ← mop-marked-as-used (get-clauses-wl-heur T) C;
          let can-del = (D ≠ Some C) ∧
      lbd > MINIMUM-DELETION-LBD ∧
             status = LEARNED ∧
             length ≠ 2 ∧
      used > 0;
          if can-del
          then
            do {
              wasted ← mop-arena-length-st T C;
              T ← mop-mark-garbage-heur C i (incr-wasted-st (of-nat wasted) T);
              RETURN (i, T)
            }
          else do {
```

```
    T ← mop-mark-unused-st-heur C T;
          RETURN (i+1, T)
  }
      }
     })
     (l, S);
   ASSERT(length (get-avdom T) ≤ length (get-clauses-wl-heur S0));
   T ← mark-clauses-as-unused-wl-D-heur i T;
   incr-restart-stat T
  })›
```

lemma *twl-st-heur-restart-same-annotD*:

```
  ‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
     Propagated L C' ∈ set (get-trail-wl T) ⟹ C = C'›
  ‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
     Decided L ∈ set (get-trail-wl T) ⟹ False›
  ⟨proof⟩
```

lemma $\mathcal{L}_{all}$*-mono*:

```
  ‹set-mset $\mathcal{A}$ ⊆ set-mset $\mathcal{B}$ ⟹ L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ L ∈# $\mathcal{L}_{all}$ $\mathcal{B}$›
  ⟨proof⟩
```

lemma *all-lits-of-mm-mono2*:

```
  ‹x ∈# (all-lits-of-mm A) ⟹ set-mset A ⊆ set-mset B ⟹ x ∈# (all-lits-of-mm B)›
  ⟨proof⟩
```

lemma $\mathcal{L}_{all}$*-init-all*:

```
  ‹L ∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a) ⟹ L ∈# $\mathcal{L}_{all}$ (all-atms-st x1a)›
  ⟨proof⟩
```

lemma *get-vdom-mark-garbage*[*simp*]:

```
  ‹get-vdom (mark-garbage-heur C i S) = get-vdom S›
  ‹get-avdom (mark-garbage-heur C i S) = delete-index-and-swap (get-avdom S) i›
  ⟨proof⟩
```

lemma *mark-to-delete-clauses-wl-D-heur-alt-def*:

```
    ‹mark-to-delete-clauses-wl-D-heur  = (λS0. do {
          ASSERT (mark-to-delete-clauses-wl-D-heur-pre S0);
          S ← sort-vdom-heur S0;
          - ← RETURN (get-avdom S);
          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);
                   ASSERT
                        (clause-not-marked-to-delete-heur-pre
                          (T, get-avdom T ! i));
                   b ← mop-clause-not-marked-to-delete-heur T
                        (get-avdom T ! i);
                   if ¬b then RETURN (i, delete-index-vdom-heur i T)
                   else do {
```

```
          ASSERT
              (access-lit-in-clauses-heur-pre
                ((T, get-avdom T ! i), 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));
          L ← mop-access-lit-in-clauses-heur T
              (get-avdom T ! i) 0;
          D ← get-the-propagation-reason-pol
              (get-trail-wl-heur T) L;
          ASSERT
              (get-clause-LBD-pre (get-clauses-wl-heur T)
                (get-avdom T ! i));
          ASSERT
              (arena-is-valid-clause-idx
                (get-clauses-wl-heur T) (get-avdom T ! i));
          ASSERT
              (arena-is-valid-clause-vdom
                (get-clauses-wl-heur T) (get-avdom T ! i));
          ASSERT
              (marked-as-used-pre
                (get-clauses-wl-heur T) (get-avdom T ! i));
          let can-del = (D ≠ Some (get-avdom T ! i) ∧
            MINIMUM-DELETION-LBD
            < arena-lbd (get-clauses-wl-heur T)
               (get-avdom T ! i) ∧
            arena-status (get-clauses-wl-heur T)
             (get-avdom T ! i) =
            LEARNED ∧
            arena-length (get-clauses-wl-heur T)
             (get-avdom T ! i) ≠
            2 ∧
            marked-as-used (get-clauses-wl-heur T)
               (get-avdom T ! i) > 0);
          if can-del
          then do {
               wasted ← mop-arena-length-st T (get-avdom T ! i);
                ASSERT(mark-garbage-pre
                  (get-clauses-wl-heur T, get-avdom T ! i) ∧
                  1 ≤ get-learned-count T ∧ i < length (get-avdom T));
                RETURN
              (i, mark-garbage-heur (get-avdom T ! i) i (incr-wasted-st (of-nat wasted) T))
              }
          else do {
                ASSERT(arena-act-pre (get-clauses-wl-heur T) (get-avdom T ! i));
                RETURN
                 (i + 1,
                  mark-unused-st-heur (get-avdom T ! i) T)
              }
        }
    })
  (l, S);
ASSERT
```

```
            (length (get-avdom T) ≤ length (get-clauses-wl-heur S0));
        mark-clauses-as-unused-wl-D-heur i T ≫= incr-restart-stat
      })›
  ⟨proof⟩

lemma mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D:
  ‹(mark-to-delete-clauses-wl-D-heur, mark-to-delete-clauses-wl) ∈
     twl-st-heur-restart-ana r →f ⟨twl-st-heur-restart-ana r⟩nres-rel›
⟨proof⟩

definition cdcl-twl-full-restart-wl-prog-heur where
‹cdcl-twl-full-restart-wl-prog-heur S = do {
  - ← 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) ∈
     twl-st-heur''' r →f ⟨twl-st-heur''' r⟩nres-rel›
  ⟨proof⟩

definition cdcl-twl-restart-wl-heur where
‹cdcl-twl-restart-wl-heur S = do {
    let b = lower-restart-bound-not-reached S;
    if b then cdcl-twl-local-restart-wl-D-heur S
    else cdcl-twl-full-restart-wl-prog-heur S
  }›


lemma cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog:
  ‹(cdcl-twl-restart-wl-heur, cdcl-twl-restart-wl-prog) ∈
    twl-st-heur''' r →f ⟨twl-st-heur''' r⟩nres-rel›
  ⟨proof⟩


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 ℒall-atm-of-all-init-lits-of-mm:
  ‹set-mset (ℒall (atm-of ‘# all-init-lits N NUE)) = set-mset (all-init-lits N NUE)›
  ⟨proof⟩

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⟩

lemmas *trail-pol-replace-annot-in-trail-spec2* =
trail-pol-replace-annot-in-trail-spec[*of* ⟨– -⟩, *simplified*]

lemma $\mathcal{L}_{all}$-*ball-all*:
⟨(∀ L ∈# $\mathcal{L}_{all}$ (*all-atms N NUE*). *P L*) = (∀ L ∈# *all-lits N NUE*. *P L*)⟩
⟨(∀ L ∈# $\mathcal{L}_{all}$ (*all-init-atms N NUE*). *P L*) = (∀ L ∈# *all-init-lits N NUE*. *P L*)⟩
⟨proof⟩

lemma *twl-st-heur-restart-ana-US-empty*:
⟨NO-MATCH {#} US ⟹ (S, M, N, D, NE, UE, NS, US, W, Q) ∈ twl-st-heur-restart-ana r ⟷
 (S, M, N, D, NE, UE, NS, {#}, W, Q)
  ∈ twl-st-heur-restart-ana r⟩
 ⟨proof⟩

fun *equality-except-trail-empty-US-wl* :: ⟨'v twl-st-wl ⇒ 'v twl-st-wl ⇒ bool⟩ **where**
⟨equality-except-trail-empty-US-wl (M, N, D, NE, UE, NS, US, WS, Q)
  (M', N', D', NE', UE', NS', US', WS', Q') ⟷
 N = N' ∧ D = D' ∧ NE = NE' ∧ NS = NS' ∧ US = {#} ∧ UE = UE' ∧ WS = WS' ∧ Q = Q'⟩

lemma *equality-except-conflict-wl-get-clauses-wl*:
 ⟨equality-except-conflict-wl S Y ⟹ get-clauses-wl S = get-clauses-wl Y⟩ **and**
equality-except-conflict-wl-get-trail-wl:
 ⟨equality-except-conflict-wl S Y ⟹ get-trail-wl S = get-trail-wl Y⟩ **and**
equality-except-trail-empty-US-wl-get-conflict-wl:
 ⟨equality-except-trail-empty-US-wl S Y ⟹ get-conflict-wl S = get-conflict-wl Y⟩ **and**
equality-except-trail-empty-US-wl-get-clauses-wl:
 ⟨equality-except-trail-empty-US-wl S Y⟹ get-clauses-wl S = get-clauses-wl Y⟩
⟨proof⟩

lemma *isasat-replace-annot-in-trail-replace-annot-in-trail-spec*:
⟨(((L, C), S), ((L', C'), S')) ∈ Id $\times_f$ Id $\times_f$ twl-st-heur-restart-ana r ⟹
isasat-replace-annot-in-trail L C S ≤
 ⇓{(U, U'). (U, U') ∈ twl-st-heur-restart-ana r ∧
  get-clauses-wl-heur U = get-clauses-wl-heur S ∧
  get-vdom U = get-vdom S ∧
  equality-except-trail-empty-US-wl U' S'}
 (replace-annot-wl L' C' S')⟩
⟨proof⟩

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 = (λi 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 ≤ Suc (uint32-max div 2));
    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 ← 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-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-inv S′ (*i*, *T′*)))›

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where

```
‹remove-one-annot-true-clause-imp-wl-D-heur = (λS. do {
    ASSERT((isa-length-trail-pre o get-trail-wl-heur) S);
    k ← (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) ← $WHILE_T^{remove-one-annot-true-clause-imp-wl-D-heur-inv\ S}$
      (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
           (λk. ∃ M1. (∃ M2 K.
                          (Decided K # M1, M2)
                          ∈ set (get-all-ann-decomposition (get-trail-wl T))) ∧
                     count-decided M1 = 0 ∧ k = length M1)›
```
  ⟨*proof*⟩

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
  ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *mop-isa-length-trail* (*get-trail-wl-heur S*) = *RETURN* (*length* (*get-trail-wl T*))›
  ⟨*proof*⟩

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
  fixes *S* :: *twl-st-wl-heur*
  shows ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
  ⟨*proof*⟩

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-st T*)›
⟨*proof*⟩

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup M* ⟹ *k* < *length M* ⟹ *M* ! *k* = *Propagated L C* ⟹ *Decided L* ∉ *set M*›
⟨*proof*⟩

lemma *remove-all-annot-true-clause-imp-wl-inv-length-cong*:
‹*remove-all-annot-true-clause-imp-wl-inv S xs T* ⟹
length xs = *length ys* ⟹ *remove-all-annot-true-clause-imp-wl-inv S ys T*›
⟨*proof*⟩

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-pre k' T*› **and**
proped: ‹*is-proped* (*rev* (*get-trail-wl T*) ! *k'*)›
shows ‹*do* {
L ← *isa-trail-nth* (*get-trail-wl-heur S*) *k*;
C ← *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *L*;
RETURN (*L*, *C*)
} ≤ ⇓ {((*L*, *C*), *L'*, *C'*). *L* = *L'* ∧ *C'* = *the C* ∧ *C* ≠ *None*}
(*SPEC* (λ*p*. *rev* (*get-trail-wl T*) ! *k'* = *Propagated* (*fst p*) (*snd p*)))›
⟨*proof*⟩

lemma *red-in-dom-number-of-learned-ge1*: ‹*C'* ∈# *dom-m baa* ⟹ ¬ *irred baa C'* ⟹ *Suc 0* ≤ *size* (*learned-clss-l baa*)›
⟨*proof*⟩

lemma *mark-garbage-heur2-remove-and-add-cls-l*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ (*C*, *C'*) ∈ *Id* ⟹
mark-garbage-heur2 C S
≤ ⇓ (*twl-st-heur-restart-ana r*) (*remove-and-add-cls-wl C' T*)›
⟨*proof*⟩

lemma *remove-one-annot-true-clause-one-imp-wl-pre-fst-le-uint32*:
assumes ‹(*x*, *y*) ∈ *nat-rel* $\times_f$ *twl-st-heur-restart-ana r*› **and**
‹*remove-one-annot-true-clause-one-imp-wl-pre* (*fst y*) (*snd y*)›
shows ‹*fst x* + *1* ≤ *Suc* (*uint32-max div 2*)›
⟨*proof*⟩

lemma *remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D*:
‹(*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*,
uncurry remove-one-annot-true-clause-one-imp-wl) ∈
nat-rel $\times_f$ *twl-st-heur-restart-ana r* $\rightarrow_f$ ⟨*nat-rel* $\times_f$ *twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

definition *find-decomp-wl0* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl* ⇒ *bool*› **where**
‹*find-decomp-wl0* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) (*M'*, *N'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*, *W'*).
(∃ *K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧

```
    count-decided M′ = 0) ∧
  (N′, D′, NE′, UE′, NS, US, Q′, W′) = (N, D, NE, UE, NS′, US′, Q, W))›

definition empty-Q-wl :: ‹′v twl-st-wl ⇒ ′v twl-st-wl› where
‹empty-Q-wl = (λ(M′, N, D, NE, UE, NS, US, -, W). (M′, N, D, NE, UE, NS, {#}, {#}, W))›

definition empty-US-wl :: ‹′v twl-st-wl ⇒ ′v twl-st-wl› where
‹empty-US-wl = (λ(M′, N, D, NE, UE, NS, US, Q, W). (M′, N, D, NE, UE, NS, {#}, Q, W))›

lemma cdcl-twl-local-restart-wl-spec0-alt-def:
  ‹cdcl-twl-local-restart-wl-spec0 = (λS. do {
    ASSERT(restart-abs-wl-pre2 S False);
    if count-decided (get-trail-wl S) > 0
    then do {
      T ← SPEC(find-decomp-wl0 S);
      RETURN (empty-Q-wl T)
    } else RETURN (empty-US-wl S)})›
  ⟨proof⟩

lemma cdcl-twl-local-restart-wl-spec0:
  assumes Sy: ‹(S, y) ∈ 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
    }
         ≤ ⇓ (twl-st-heur-restart-ana r) (cdcl-twl-local-restart-wl-spec0 y)›
⟨proof⟩

lemma no-get-all-ann-decomposition-count-dec0:
  ‹(∀M1. (∀M2 K. (Decided K # M1, M2) ∉ set (get-all-ann-decomposition M))) ⟷
  count-decided M = 0›
  ⟨proof⟩

lemma get-pos-of-level-in-trail-decomp-iff:
  assumes ‹no-dup M›
  shows ‹((∃M1 M2 K.
                (Decided K # M1, M2)
                ∈ set (get-all-ann-decomposition M) ∧
                count-decided M1 = 0 ∧ k = length M1)) ⟷
    k < length M ∧ count-decided M > 0 ∧ is-decided (rev M ! k) ∧ get-level M (lit-of (rev M ! k)) =
1›
  (is ‹?A ⟷ ?B›)
⟨proof⟩

lemma remove-all-learned-subsumed-clauses-wl-id:
  ‹(x2a, x2) ∈ twl-st-heur-restart-ana r ⟹
   RETURN x2a
   ≤ ⇓ (twl-st-heur-restart-ana r)
      (remove-all-learned-subsumed-clauses-wl x2)›
   ⟨proof⟩

lemma remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D:
```

‹(*remove-one-annot-true-clause-imp-wl-D-heur*, *remove-one-annot-true-clause-imp-wl*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl2-D*:
‹(*mark-to-delete-clauses-wl-D-heur*, *mark-to-delete-clauses-wl2*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
⟨*proof*⟩

definition *iterate-over-VMTF* **where**

```
‹iterate-over-VMTF ≡ (λf (I :: 'a ⇒ bool) (ns :: (nat, nat) vmtf-node list, n) x. do {
    (-, x) ← WHILE_T^λ(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-ℒ_all = (λf 𝒜₀ I x. do {
  𝒜 ← SPEC(λ𝒜. set-mset 𝒜 = set-mset 𝒜₀ ∧ distinct-mset 𝒜);
  (-, x) ← WHILE_T^λ(-, x). I x
    (λ(ℬ, -). ℬ ≠ {#})
    (λ(ℬ, x). do {
      ASSERT(ℬ ≠ {#});
      A ← SPEC (λA. A ∈# ℬ);
      x ← f A x;
      RETURN (remove1-mset A ℬ, x)
    })
    (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-*$\mathcal{L}_{all}$ *f* $\mathcal{A}$ *I x*)›
⟨*proof*⟩

definition *arena-is-packed* :: ‹*arena* ⇒ *nat clauses-l* ⇒ *bool*› **where**
‹*arena-is-packed arena N* ⟷ *length arena* = ($\sum$ *C* ∈# *dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*))›

lemma *arena-is-packed-empty*[*simp*]: ‹*arena-is-packed* [] *fmempty*›

  ⟨*proof*⟩

```
lemma sum-mset-cong:
  ‹(⋀A. A ∈# M ⟹ f A = g A) ⟹ (∑ A ∈# M. f A) = (∑ A ∈# M. g A)›
  ⟨proof⟩
lemma arena-is-packed-append:
  assumes ‹arena-is-packed (arena) N› and
    [simp]: ‹length C = length (fst C′) + header-size (fst C′)› and
    [simp]: ‹a ∉# dom-m N›
  shows ‹arena-is-packed (arena @ C) (fmupd a C′ N)›
⟨proof⟩
```

```
lemma arena-is-packed-append-valid:
  assumes
    in-dom: ‹fst C ∈# 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-length x1c (fst C)) x1c)
          (fmupd (length x1d + n) (the (fmlookup x1a (fst C))) x2a)›
⟨proof⟩

definition move-is-packed :: ‹arena ⇒ - ⇒ arena ⇒ - ⇒ bool› where
‹move-is-packed arena_o N_o arena N ⟷
   ((∑ C∈#dom-m N_o. length (N_o ∝ C) + header-size (N_o ∝ C)) +
   (∑ C∈#dom-m N. length (N ∝ C) + header-size (N ∝ C)) ≤ 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 = (λN0 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 MAX-HEADER-SIZE else MIN-HEADER-SIZE) +
            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 MAX-HEADER-SIZE else MIN-HEADER-SIZE);
N′ ← fm-mv-clause-to-new-arena C N N′;
ASSERT(mark-garbage-pre (N, C));
RETURN (i+1, extra-information-mark-to-delete N C, N′, vdm @ [D],
(if st = LEARNED then avdm @ [D] else avdm))
} else RETURN (i+1, N, (N′, vdm, avdm))
}) (0, N0, (N′, vdm, avdm));
RETURN (N, (N′, vdm, avdm))
})›

definition *isasat-GC-entry* :: ‹-› **where**
‹*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena-old W′ = {((arena$_o$, (arena, vdom, avdom)), (N$_o$, N)). valid-arena arena$_o$ N$_o$ vdom0* ∧ *valid-arena arena N (set vdom)* ∧ *vdom-m* $\mathcal{A}$ *W′ N$_o$* ⊆ *vdom0* ∧ *dom-m N = mset vdom* ∧ *distinct vdom* ∧
arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena$_o$ = length arena-old* ∧
move-is-packed arena$_o$ N$_o$ arena N}›

definition *isasat-GC-refl* :: ‹-› **where**
‹*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena-old = {((arena$_o$, (arena, vdom, avdom), W), (N$_o$, N, W′)). valid-arena arena$_o$ N$_o$ vdom0* ∧ *valid-arena arena N (set vdom)* ∧
(W, W′) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W′ N$_o$* ⊆ *vdom0* ∧ *dom-m N = mset vdom* ∧ *distinct vdom* ∧
arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena$_o$ = length arena-old* ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length (W′ L)* ≤ *length arena$_o$*) ∧*move-is-packed arena$_o$ N$_o$ arena N}*›

lemma *move-is-packed-empty[simp]*: ‹*valid-arena arena N vdom* ⟹ *move-is-packed arena N [] fmempty*›
⟨*proof*⟩

lemma *move-is-packed-append*:
assumes
dom: ‹*C* ∈# *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*⟩

definition *arena-header-size* :: ‹*arena* ⇒ *nat* ⇒ *nat*› **where**
‹*arena-header-size arena C =*
(if arena-length arena C > 4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE)›

lemma *valid-arena-header-size*:
‹*valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *arena-header-size arena C = header-size (N* ∝ *C)*›
⟨*proof*⟩

lemma *isasat-GC-clauses-prog-copy-wl-entry*:
assumes ‹*valid-arena arena N vdom0*› **and**
‹*valid-arena arena′ N′ (set vdom)*› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W N* ⊆ *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 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 ∈# ℒ_all 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 𝒜 vdom0 arena W)
         (cdcl-GC-clauses-prog-copy-wl-entry N (W A) A N')›
     (is ‹- ≤ ⇓ (?R) -›)
⟨proof⟩
```

```
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 𝒜 vdom0 arena0› and
    X: ‹X = (arena, (arena', vdom, avdom), W)› ‹X' = (N, N', W')› and
    L: ‹A ∈# 𝒜› and
    st: ‹(A, A') ∈ Id› and st': ‹narena = (arena', vdom, avdom)› and
    ae: ‹length arena0 = length arena› and
    le-all: ‹∀ L ∈# ℒ_all 𝒜. length (W' L) ≤ length arena›
  shows ‹isasat-GC-clauses-prog-single-wl arena narena  W A
     ≤ ⇓ (isasat-GC-refl 𝒜 vdom0 arena0)
         (cdcl-GC-clauses-prog-single-wl N W' A' N')›
     (is ‹- ≤ ⇓ ?R -›)
⟨proof⟩
```

```
definition isasat-GC-clauses-prog-wl2 where
  ‹isasat-GC-clauses-prog-wl2 ≡ (λ(ns :: (nat, nat) vmtf-node list, n) x0. do {
      (-, x) ← WHILE_T^{λ(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 = (λN0 𝒜0 WS. do {*
𝒜 ← SPEC(λ𝒜. set-mset 𝒜 = set-mset 𝒜0);
(-, (N, N′, WS)) ← $WHILE_T{}^{cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl\text{-}inv\ \mathcal{A}\ N0}$
(λ(ℬ, -). ℬ ≠ {#})
(λ(ℬ, (N, N′, WS)). do {
ASSERT(ℬ ≠ {#});
A ← SPEC (λA. A ∈# ℬ);
(N, N′, WS) ← cdcl-GC-clauses-prog-single-wl N WS A N′;
RETURN (remove1-mset A ℬ, (N, N′, WS))
})
(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′ -*›)
⟨*proof*⟩

lemma *cdcl-GC-clauses-prog-wl-inv-cong-empty*:
‹*set-mset 𝒜 = set-mset ℬ ⟹*
cdcl-GC-clauses-prog-wl-inv 𝒜 N ({#}, x) ⟹ cdcl-GC-clauses-prog-wl-inv ℬ N ({#}, x)›
⟨*proof*⟩

lemma *isasat-GC-clauses-prog-wl2*:
assumes ‹*valid-arena* $arena_o$ N_o *vdom0*› **and**
‹*valid-arena arena N (set vdom)*› **and**
vdom: ‹*vdom-m 𝒜 W′* N_o *⊆ vdom0*› **and**
vmtf: ‹*((ns, m, n, lst-As1, next-search1), to-remove1) ∈ vmtf 𝒜 M*› **and**
nempty: ‹*𝒜 ≠ {#}*› **and**
W-W′: ‹*(W, W′) ∈ ⟨Id⟩map-fun-rel* (D_0 *𝒜)*› **and**
bounded: ‹*isasat-input-bounded 𝒜*› **and** *old*: ‹*old-arena = []*› **and**
le-all: ‹*∀L ∈#* $\mathcal{L}_{all}$ *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 *𝒜) ∧ vdom-m 𝒜 W′* $N_o{}'$ *⊆ vdom0 ∧*
cdcl-GC-clauses-prog-wl-inv 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 *𝒜 W′)*›
⟨*proof*⟩

lemma *cdcl-GC-clauses-prog-wl-alt-def*:
‹*cdcl-GC-clauses-prog-wl* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*). *do* {
  ASSERT(*cdcl-GC-clauses-pre-wl* (*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*));
  (*N*, *N′*, *WS*) ← *cdcl-GC-clauses-prog-wl2 N0* (*all-init-atms N0* (*NE*+*NS*)) *WS*;
  RETURN (*M*, *N′*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*)
  })›
⟨*proof*⟩

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*), *clvls*, *cach*, *lbd*, *outl*, *stats*,
  heur, *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*), *clvls*, *cach*, *lbd*, *outl*, *incr-GC stats*, *set-zero-wasted heur*,
    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*⟩

lemma *isasat-GC-clauses-prog-wl*:
‹(*isasat-GC-clauses-prog-wl*, *cdcl-GC-clauses-prog-wl*) ∈
twl-st-heur-restart $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}⟩*nres-rel*›
(**is** ‹- ∈ *?T* $\rightarrow_f$ -›)
⟨*proof*⟩

definition *cdcl-remap-st* :: ‹*′v twl-st-wl* ⇒ *′v twl-st-wl nres*› **where**
‹*cdcl-remap-st* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*).
SPEC (λ(*M′*, *N′*, *D′*, *NE′*, *UE′*, *NS′*, *US′*, *Q′*, *WS′*).
  (*M′*, *D′*, *NE′*, *UE′*, *NS′*, *US′*, *Q′*) = (*M*, *D*, *NE*, *UE*, *NS*, *US*, *Q*) ∧
  (∃ *m*. $GC\text{-}remap^{**}$ (*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*, *NS*, *US*, *Q*, *WS*).
SPEC (λ(*M′*, *N′*, *D′*, *NE′*, *UE′*, *NS′*, *US′*, *Q′*, *WS′*).
  (*M′*, *N′*, *D′*, *NE′*, *UE′*, *NS′*, *US′*, *Q′*) = (*M*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *Q*) ∧
  correct-watching′ (*M*, *N′*, *D*, *NE*, *UE*, *NS′*, *US*, *Q′*, *WS′*) ∧
  literals-are-$\mathcal{L}_{in}$*′* (*M*, *N′*, *D*, *NE*, *UE*, *NS′*, *US*, *Q′*, *WS′*)))›

lemma *blits-in-*$\mathcal{L}_{in}$*′-restart-wl-spec0′*:
‹*literals-are-*$\mathcal{L}_{in}$*′* (*a*, *aq*, *ab*, *ac*, *ad*, *ae*, *af*, *Q*, *b*) ⟹
  literals-are-$\mathcal{L}_{in}$*′* (*a*, *aq*, *ab*, *ac*, *ad*, *ae*, *af*, {#}, *b*)›
⟨*proof*⟩

lemma *cdcl-GC-clauses-wl-D-alt-def*:

```
  ‹cdcl-GC-clauses-wl = (λS. do {
    ASSERT(cdcl-GC-clauses-pre-wl S);
    let b = True;
    if b then do {
      S ← cdcl-remap-st S;
      S ← rewatch-spec S;
      RETURN S
    }
    else remove-all-learned-subsumed-clauses-wl S})›
  ⟨proof⟩

definition isasat-GC-clauses-pre-wl-D :: ‹twl-st-wl-heur ⇒ bool› where
‹isasat-GC-clauses-pre-wl-D S ⟷ (
  ∃ T. (S, T) ∈ twl-st-heur-restart ∧ cdcl-GC-clauses-pre-wl T
  )›


definition isasat-GC-clauses-wl-D :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
‹isasat-GC-clauses-wl-D = (λS. 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-st T) ∪
       Pos ‘ set-mset (all-init-atms-st T))
      (get-watched-wl T) ∧
    literals-are-ℒin' T› and
    ‹get-clauses-wl T = N0'›
  shows
    ‹cdcl-GC-clauses-prog-wl T ≤
      ⇓ {((M', N'', D', NE', UE', NS', US', Q', WS'), (N, N')).
      (M', D', NE', UE', NS', US', Q') = (get-trail-wl T, get-conflict-wl T, get-unit-init-clss-wl T,
          get-unit-learned-clss-wl T, get-subsumed-init-clauses-wl T, get-subsumed-learned-clauses-wl T,
          literals-to-update-wl T) ∧ N'' = N ∧
          (∀ L∈#all-init-lits-st T. WS' L = []) ∧
          all-init-lits-st T = all-init-lits N (NE'+NS') ∧
          (∃ 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'))›
    ⟨proof⟩

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-ℒin' xa›
  shows ‹set-mset (dom-m (get-clauses-wl xa))
        ⊆ clauses-pointed-to
           (Neg ‘
            set-mset
             (all-init-atms-st xa) ∪
            Pos ‘
            set-mset
             (all-init-atms-st xa))
           (get-watched-wl xa)›
       (is ‹- ⊆ ?A›)
⟨proof⟩
```

```
abbreviation isasat-GC-clauses-rel where
  ‹isasat-GC-clauses-rel y ≡ {(S, T). (S, T) ∈ twl-st-heur-restart ∧
           (∀ L∈#all-init-lits-st y. get-watched-wl T L = [])∧
           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 ∧
           get-subsumed-init-clauses-wl T = get-subsumed-init-clauses-wl y ∧
           get-subsumed-learned-clauses-wl T = get-subsumed-learned-clauses-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 ⊆ R' ⟹ A ≤ B ⟹ ⇓ R A ≤ ⇓ R' B›
  ⟨proof⟩
```

```
lemma isasat-GC-clauses-prog-wl-cdcl-remap-st:
  assumes
    ‹(x, y) ∈ twl-st-heur-restart''' r› and
    ‹cdcl-GC-clauses-pre-wl y›
  shows ‹isasat-GC-clauses-prog-wl x ≤ ⇓ (isasat-GC-clauses-rel y) (cdcl-remap-st y)›
⟨proof⟩
```

```
fun correct-watching''' :: ‹- ⇒ 'v twl-st-wl ⇒ bool› where
  ‹correct-watching''' 𝒜 (M, N, D, NE, UE, NS, US, Q, W) ⟷
    (∀ L ∈# all-lits-of-mm 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, NS, US, {#}, {#}))›
```

```
declare correct-watching'''.simps[simp del]
```

```
lemma correct-watching'''-add-clause:
  assumes
    corr: ‹correct-watching''' 𝒜 ((a, aa, CD, ac, ad, NS, US, 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, NS, US, Q, b
(C ! 0 := b (C ! 0) @ [(i, C ! Suc 0, length C = 2)],
C ! Suc 0 := b (C ! Suc 0) @ [(i, C ! 0, length C = 2)]))))›
⟨*proof*⟩

lemma *rewatch-correctness*:
assumes *empty*: ‹⋀*L. L* ∈# *all-lits-of-mm* $\mathcal{A}$ ⟹ *W L* = []› **and**
H[*dest*]: ‹⋀*x. x* ∈# *dom-m N* ⟹ *distinct* (*N* ∝ *x*) ∧ *length* (*N* ∝ *x*) ≥ *2*› **and**
incl: ‹*set-mset* (*all-lits-of-mm* (*mset* ‘# *ran-mf N*)) ⊆ *set-mset* (*all-lits-of-mm* $\mathcal{A}$)›
shows
‹*rewatch N W* ≤ *SPEC*(λ*W. correct-watching*''' $\mathcal{A}$ (*M, N, C, NE, UE, NS, US, Q, W*))›
⟨*proof*⟩

inductive-cases *GC-remapE*: ‹*GC-remap* (*a, aa, b*) (*ab, ac, ba*)›
lemma *rtranclp-GC-remap-ran-m-remap*:
‹*GC-remap*** (*old, m, new*) (*old', m', new'*) ⟹ *C* ∈# *dom-m old* ⟹ *C* ∉# *dom-m old'* ⟹
m' C ≠ *None* ∧
fmlookup new' (*the* (*m' C*)) = *fmlookup old C*›
⟨*proof*⟩

lemma *GC-remap-ran-m-exists-earlier*:
‹*GC-remap* (*old, m, new*) (*old', m', new'*) ⟹ *C* ∈# *dom-m new'* ⟹ *C* ∉# *dom-m new* ⟹
∃ *D. m' D* = *Some C* ∧ *D* ∈# *dom-m old* ∧
fmlookup new' C = *fmlookup old D*›
⟨*proof*⟩

lemma *rtranclp-GC-remap-ran-m-exists-earlier*:
‹*GC-remap*** (*old, m, new*) (*old', m', new'*) ⟹ *C* ∈# *dom-m new'* ⟹ *C* ∉# *dom-m new* ⟹
∃ *D. m' D* = *Some C* ∧ *D* ∈# *dom-m old* ∧
fmlookup new' C = *fmlookup old D*›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*:
‹*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N NE*)) = *set-mset* (*all-init-lits N NE*)›
⟨*proof*⟩

lemma *rewatch-heur-st-correct-watching*:
assumes
pre: ‹*cdcl-GC-clauses-pre-wl 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*⟩

lemma *GC-remap-dom-m-subset*:
‹*GC-remap* (*old, m, new*) (*old', m', new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›
⟨*proof*⟩

lemma *rtranclp-GC-remap-dom-m-subset*:
‹*rtranclp GC-remap* (*old, m, new*) (*old', m', new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›
⟨*proof*⟩

lemma *GC-remap-mapping-unchanged*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
⟨*proof*⟩

lemma *rtranclp-GC-remap-mapping-unchanged*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
⟨*proof*⟩

lemma *GC-remap-mapping-dom-extended*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)›
⟨*proof*⟩

lemma *rtranclp-GC-remap-mapping-dom-extended*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)›
⟨*proof*⟩

lemma *GC-remap-dom-m*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* − *dom-m old'*)›
⟨*proof*⟩

lemma *rtranclp-GC-remap-dom-m*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* − *dom-m old'*)›
⟨*proof*⟩

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*⟩

lemma *isasat-GC-clauses-wl-D*:
‹(*isasat-GC-clauses-wl-D*, *cdcl-GC-clauses-wl*)
  ∈ *twl-st-heur-restart'''* *r* $\rightarrow_f$ ⟨*twl-st-heur-restart''''* *r*⟩*nres-rel*›
⟨*proof*⟩

definition *cdcl-twl-full-restart-wl-D-GC-heur-prog* **where**

```
‹cdcl-twl-full-restart-wl-D-GC-heur-prog S0 = do {
    S ← do {
      if count-decided-st-heur S0 > 0
      then do {
        S ← find-decomp-wl-st-int 0 S0;
        empty-Q S
      } else RETURN S0
    };
    ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
    T ← remove-one-annot-true-clause-imp-wl-D-heur S;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
    U ← mark-to-delete-clauses-wl-D-heur T;
    ASSERT(length (get-clauses-wl-heur U) = length (get-clauses-wl-heur S0));
    V ← isasat-GC-clauses-wl-D U;
    RETURN V
  }›
```

lemma

```
    cdcl-twl-full-restart-wl-GC-prog-pre-heur:
      ‹cdcl-twl-full-restart-wl-GC-prog-pre T ⟹
        (S, T) ∈ twl-st-heur''' r ⟷ (S, T) ∈ twl-st-heur-restart-ana r› (is ‹- ⟹ - ?A›) and
      cdcl-twl-full-restart-wl-D-GC-prog-post-heur:
       ‹cdcl-twl-full-restart-wl-GC-prog-post S0 T ⟹
        (S, T) ∈ twl-st-heur ⟷ (S, T) ∈ twl-st-heur-restart›  (is ‹- ⟹ - ?B›)
⟨proof⟩
```

lemma *cdcl-twl-full-restart-wl-D-GC-heur-prog*:

```
  ‹(cdcl-twl-full-restart-wl-D-GC-heur-prog, cdcl-twl-full-restart-wl-GC-prog) ∈
    twl-st-heur''' r →_f ⟨twl-st-heur'''' r⟩nres-rel›
  ⟨proof⟩
```

definition *end-of-restart-phase* :: ‹*restart-heuristics* ⇒ *64 word*› **where**

```
  ‹end-of-restart-phase = (λ(-, -, (restart-phase,- ,- , end-of-phase, -), -).
    end-of-phase)›
```

definition *end-of-restart-phase-st* :: ‹*twl-st-wl-heur* ⇒ *64 word*› **where**

```
  ‹end-of-restart-phase-st = (λ(M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena).
      end-of-restart-phase heur)›
```

definition *end-of-rephasing-phase-st* :: ‹*twl-st-wl-heur* ⇒ *64 word*› **where**

```
  ‹end-of-rephasing-phase-st = (λ(M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena).
      end-of-rephasing-phase-heur heur)›
```

Using $a + (1::'a)$ ensures that we do not get stuck with 0.

fun *incr-restart-phase-end* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**

```
 ‹incr-restart-phase-end (fast-ema, slow-ema, (ccount, ema-lvl, restart-phase, end-of-phase, length-phase),
wasted) =
   (fast-ema, slow-ema, (ccount, ema-lvl, restart-phase, end-of-phase + length-phase, (length-phase * 3)
>> 1), wasted)›
```

definition *update-restart-phases* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**

```
  ‹update-restart-phases = (λ(M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
       vdom, avdom, lcount, opts, old-arena). do {
     heur ← RETURN (incr-restart-phase heur);
     heur ← RETURN (incr-restart-phase-end heur);
     RETURN (M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur,
        vdom, avdom, lcount, opts, old-arena)
  })›
```

definition *update-all-phases* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ (*twl-st-wl-heur* × *nat*) *nres*› **where**

```
  ‹update-all-phases = (λS n. do {
     let lcount = get-learned-count S;
     end-of-restart-phase ← RETURN (end-of-restart-phase-st S);
```

```
    S ← (if end-of-restart-phase > of-nat lcount then RETURN S else update-restart-phases S);
    S ← (if end-of-rephasing-phase-st S > of-nat lcount then RETURN S else rephase-heur-st S);
    RETURN (S, n)
  })›
```

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;
    if ¬brk ∧ b = FLAG-GC-restart
    then do {
       T ← cdcl-twl-full-restart-wl-D-GC-heur-prog S;
       RETURN (T, n+1)
    }
    else if ¬brk ∧ b = FLAG-restart
    then do {
       T ← cdcl-twl-restart-wl-heur S;
       RETURN (T, n+1)
    }
    else update-all-phases S n
  }›
```

lemma *restart-required-heur-restart-required-wl*:

```
  ‹(uncurry restart-required-heur, uncurry restart-required-wl) ∈
    twl-st-heur ×f nat-rel →f ⟨restart-flag-rel⟩nres-rel›
    ⟨proof⟩
```

lemma *restart-required-heur-restart-required-wl0*:

```
  ‹(uncurry restart-required-heur, uncurry restart-required-wl) ∈
    twl-st-heur''' r ×f nat-rel →f ⟨restart-flag-rel⟩nres-rel›
    ⟨proof⟩
```

lemma *heuristic-rel-incr-restartI*[*intro*!]:

```
  ‹heuristic-rel 𝒜 heur ⟹ heuristic-rel 𝒜 (incr-restart-phase-end heur)›
  ⟨proof⟩
```

lemma *update-all-phases-Pair*:

```
  ‹(uncurry update-all-phases, uncurry (RETURN oo Pair)) ∈
     twl-st-heur'''' r ×f nat-rel →f ⟨twl-st-heur'''' r ×f nat-rel⟩nres-rel›
⟨proof⟩
```

lemma *restart-prog-wl-D-heur-restart-prog-wl-D*:

```
  ‹(uncurry2 restart-prog-wl-D-heur, uncurry2 restart-prog-wl) ∈
     twl-st-heur''' r ×f nat-rel ×f bool-rel  →f ⟨twl-st-heur'''' r ×f nat-rel⟩nres-rel›
⟨proof⟩
```

lemma *restart-prog-wl-D-heur-restart-prog-wl-D2*:

```
  ‹(uncurry2 restart-prog-wl-D-heur, uncurry2 restart-prog-wl) ∈
  twl-st-heur ×f nat-rel ×f bool-rel →f ⟨twl-st-heur ×f nat-rel⟩nres-rel›
  ⟨proof⟩
```

definition *isasat-trail-nth-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat literal nres*› **where**

```
‹isasat-trail-nth-st S i = isa-trail-nth (get-trail-wl-heur S) i›

lemma isasat-trail-nth-st-alt-def:
  ‹isasat-trail-nth-st = (λ(M, -) i.  isa-trail-nth M i)›
  ⟨proof⟩

definition get-the-propagation-reason-pol-st :: ‹twl-st-wl-heur ⇒ nat literal ⇒ nat option nres› where
‹get-the-propagation-reason-pol-st S i = get-the-propagation-reason-pol (get-trail-wl-heur S) i›

lemma get-the-propagation-reason-pol-st-alt-def:
  ‹get-the-propagation-reason-pol-st = (λ(M, -) i.  get-the-propagation-reason-pol M i)›
  ⟨proof⟩


definition rewatch-heur-st-pre :: ‹twl-st-wl-heur ⇒ bool› where
‹rewatch-heur-st-pre S ⟷ (∀ i < length (get-vdom S). get-vdom S ! i ≤ sint64-max)›

lemma isasat-GC-clauses-wl-D-rewatch-pre:
  assumes
    ‹length (get-clauses-wl-heur x) ≤ sint64-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›
  ⟨proof⟩

lemma li-uint32-maxdiv2-le-unit32-max: ‹a ≤ uint32-max div 2 + 1 ⟹ a ≤ uint32-max›
  ⟨proof⟩

end
theory IsaSAT-Arena-Sorting-LLVM
  imports IsaSAT-Sorting-LLVM
begin
definition idx-cdom :: ‹arena ⇒ nat set› where
 ‹idx-cdom arena ≡ {i. valid-sort-clause-score-pre-at arena i}›

definition mop-clause-score-less where
  ‹mop-clause-score-less arena i j = do {
    ASSERT(valid-sort-clause-score-pre-at arena i);
    ASSERT(valid-sort-clause-score-pre-at arena j);
    RETURN (clause-score-ordering (clause-score-extract arena i) (clause-score-extract arena j))
  }›

sepref-register clause-score-extract

sepref-def (in −) clause-score-extract-code
  is ‹uncurry (RETURN oo clause-score-extract)›
  :: ‹[uncurry valid-sort-clause-score-pre-at]_a
      arena-fast-assn^k *_a sint64-nat-assn^k → uint32-nat-assn ×_a sint64-nat-assn›
  ⟨proof⟩

sepref-def (in −) clause-score-ordering-code
  is ‹uncurry (RETURN oo clause-score-ordering)›
  :: ‹(uint32-nat-assn ×_a sint64-nat-assn)^k *_a (uint32-nat-assn ×_a sint64-nat-assn)^k →_a bool1-assn›
  ⟨proof⟩

sepref-register mop-clause-score-less clause-score-less clause-score-ordering
```

sepref-def *mop-clause-score-less-impl*
is ‹*uncurry2 mop-clause-score-less*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

interpretation *LBD*: *weak-ordering-on-lt* **where**
C = ‹*idx-cdom vs*› **and**
less = ‹*clause-score-less vs*›
⟨*proof*⟩

interpretation *LBD*: *parameterized-weak-ordering idx-cdom clause-score-less*
mop-clause-score-less
⟨*proof*⟩

global-interpretation *LBD*: *parameterized-sort-impl-context*
‹*woarray-assn snat-assn*› ‹*eoarray-assn snat-assn*› *snat-assn*
return return
eo-extract-impl
array-upd
idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
‹*arena-fast-assn*›
defines
LBD-is-guarded-insert-impl = *LBD.is-guarded-param-insert-impl*
and *LBD-is-unguarded-insert-impl* = *LBD.is-unguarded-param-insert-impl*
and *LBD-unguarded-insertion-sort-impl* = *LBD.unguarded-insertion-sort-param-impl*
and *LBD-guarded-insertion-sort-impl* = *LBD.guarded-insertion-sort-param-impl*
and *LBD-final-insertion-sort-impl* = *LBD.final-insertion-sort-param-impl*

and *LBD-pcmpo-idxs-impl* = *LBD.pcmpo-idxs-impl*
and *LBD-pcmpo-v-idx-impl* = *LBD.pcmpo-v-idx-impl*
and *LBD-pcmpo-idx-v-impl* = *LBD.pcmpo-idx-v-impl*
and *LBD-pcmp-idxs-impl* = *LBD.pcmp-idxs-impl*

and *LBD-mop-geth-impl* = *LBD.mop-geth-impl*
and *LBD-mop-seth-impl* = *LBD.mop-seth-impl*
and *LBD-sift-down-impl* = *LBD.sift-down-impl*
and *LBD-heapify-btu-impl* = *LBD.heapify-btu-impl*
and *LBD-heapsort-impl* = *LBD.heapsort-param-impl*
and *LBD-qsp-next-l-impl* = *LBD.qsp-next-l-impl*
and *LBD-qsp-next-h-impl* = *LBD.qsp-next-h-impl*
and *LBD-qs-partition-impl* = *LBD.qs-partition-impl*

and *LBD-partition-pivot-impl* = *LBD.partition-pivot-impl*
and *LBD-introsort-aux-impl* = *LBD.introsort-aux-param-impl*
and *LBD-introsort-impl* = *LBD.introsort-param-impl*
and *LBD-move-median-to-first-impl* = *LBD.move-median-to-first-param-impl*

⟨*proof*⟩

global-interpretation
LBD-it: *pure-eo-adapter sint64-nat-assn vdom-fast-assn arl-nth arl-upd*
defines *LBD-it-eo-extract-impl* = *LBD-it.eo-extract-impl*
⟨*proof*⟩

```
global-interpretation LBD-it: parameterized-sort-impl-context
  vdom-fast-assn ‹LBD-it.eo-assn› sint64-nat-assn
  return return
  LBD-it-eo-extract-impl
  arl-upd
  idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
  ‹arena-fast-assn›
  defines
          LBD-it-is-guarded-insert-impl = LBD-it.is-guarded-param-insert-impl
      and LBD-it-is-unguarded-insert-impl = LBD-it.is-unguarded-param-insert-impl
      and LBD-it-unguarded-insertion-sort-impl = LBD-it.unguarded-insertion-sort-param-impl
      and LBD-it-guarded-insertion-sort-impl = LBD-it.guarded-insertion-sort-param-impl
      and LBD-it-final-insertion-sort-impl = LBD-it.final-insertion-sort-param-impl


      and LBD-it-pcmpo-idxs-impl  = LBD-it.pcmpo-idxs-impl
      and LBD-it-pcmpo-v-idx-impl  = LBD-it.pcmpo-v-idx-impl
      and LBD-it-pcmpo-idx-v-impl  = LBD-it.pcmpo-idx-v-impl
      and LBD-it-pcmp-idxs-impl  = LBD-it.pcmp-idxs-impl

      and LBD-it-mop-geth-impl    = LBD-it.mop-geth-impl
      and LBD-it-mop-seth-impl    = LBD-it.mop-seth-impl
      and LBD-it-sift-down-impl   = LBD-it.sift-down-impl
      and LBD-it-heapify-btu-impl = LBD-it.heapify-btu-impl
      and LBD-it-heapsort-impl    = LBD-it.heapsort-param-impl
      and LBD-it-qsp-next-l-impl       = LBD-it.qsp-next-l-impl
      and LBD-it-qsp-next-h-impl       = LBD-it.qsp-next-h-impl
      and LBD-it-qs-partition-impl     = LBD-it.qs-partition-impl

      and LBD-it-partition-pivot-impl  = LBD-it.partition-pivot-impl
      and LBD-it-introsort-aux-impl = LBD-it.introsort-aux-param-impl
      and LBD-it-introsort-impl        = LBD-it.introsort-param-impl
      and LBD-it-move-median-to-first-impl = LBD-it.move-median-to-first-param-impl

  ⟨proof⟩



lemmas [llvm-inline] = LBD-it.eo-extract-impl-def[THEN meta-fun-cong, THEN meta-fun-cong]

print-named-simpset llvm-inline
export-llvm
  ‹LBD-heapsort-impl :: - ⇒ - ⇒ -›
  ‹LBD-introsort-impl :: - ⇒ - ⇒ -›



end
theory IsaSAT-Restart-Heuristics-LLVM
  imports IsaSAT-Restart-Heuristics IsaSAT-Setup-LLVM
     IsaSAT-VMTF-LLVM IsaSAT-Rephase-LLVM
     IsaSAT-Arena-Sorting-LLVM
begin
```

hide-fact (**open**) *Sepref-Rules.frefI*
no-notation *Sepref-Rules.fref* (‹[-]$_{fd}$ - → -› [*0*,*60*,*60*] *60*)
no-notation *Sepref-Rules.freft* (‹- →$_{fd}$ -› [*60*,*60*] *60*)
no-notation *Sepref-Rules.freftnd* (‹- →$_{f}$ -› [*60*,*60*] *60*)
no-notation *Sepref-Rules.frefnd* (‹[-]$_{f}$ - → -› [*0*,*60*,*60*] *60*)

sepref-def *FLAG-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *FLAG-no-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-no-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *FLAG-GC-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-GC-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

lemma *current-restart-phase-alt-def*:
  ‹*current-restart-phase* = (λ(*fast-ema*, *slow-ema*,
    (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), -).
    restart-phase)›
  ⟨*proof*⟩

sepref-def *current-restart-phase-impl*
  is ‹*RETURN o current-restart-phase*›
  :: ‹*heuristic-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *get-restart-phase-imp*
  is ‹(*RETURN o get-restart-phase*)›
  :: ‹*isasat-bounded-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *end-of-restart-phase-impl*
  is ‹*RETURN o end-of-restart-phase*›
  :: ‹*heuristic-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *end-of-restart-phase-st-impl*
  is ‹*RETURN o end-of-restart-phase-st*›
  :: ‹*isasat-bounded-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *end-of-rephasing-phase-impl*
  is ‹*RETURN o end-of-rephasing-phase*›
  :: ‹*phase-heur-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *end-of-rephasing-phase-heur-impl*
  is ‹*RETURN o end-of-rephasing-phase-heur*›
  :: ‹*heuristic-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *end-of-rephasing-phase-st-impl*
is ‹*RETURN o end-of-rephasing-phase-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

lemma *incr-restart-phase-end-alt-def*:
‹*incr-restart-phase-end* = (λ(*fast-ema*, *slow-ema*,
(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*), *wasted*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase* + *length-phase*,
(*length-phase* ∗ *3*) >> *1*), *wasted*))›
⟨*proof*⟩

sepref-def *incr-restart-phase-end-impl*
is ‹*RETURN o incr-restart-phase-end*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

lemma *incr-restart-phase-alt-def*:
‹*incr-restart-phase* = (λ(*fast-ema*, *slow-ema*,
(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*))›
⟨*proof*⟩

sepref-def *incr-restart-phase-impl*
is ‹*RETURN o incr-restart-phase*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

sepref-register *incr-restart-phase incr-restart-phase-end*
update-restart-phases update-all-phases

sepref-def *update-restart-phases-impl*
is ‹*update-restart-phases*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *update-all-phases-impl*
is ‹*uncurry update-all-phases*›
:: ‹*isasat-bounded-assn*^{d} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$
isasat-bounded-assn $\times_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-def *find-local-restart-target-level-fast-code*
is ‹*uncurry find-local-restart-target-level-int*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *vmtf-remove-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *find-local-restart-target-level-st-fast-code*
is ‹*find-local-restart-target-level-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *empty-Q-fast-code*

is ‹*empty-Q*›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

sepref-register *cdcl-twl-local-restart-wl-D-heur*
empty-Q find-decomp-wl-st-int

find-theorems *count-decided-st-heur name*:*refine*
sepref-def *cdcl-twl-local-restart-wl-D-heur-fast-code*
is ‹*cdcl-twl-local-restart-wl-D-heur*›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow_a \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

sepref-register *upper-restart-bound-not-reached*

sepref-def *upper-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k \rightarrow_a \mathit{bool1\text{-}assn}$›
⟨*proof*⟩

sepref-register *lower-restart-bound-not-reached*
sepref-def *lower-restart-bound-not-reached-impl*
is ‹(*RETURN o lower-restart-bound-not-reached*)›
:: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k \rightarrow_a \mathit{bool1\text{-}assn}$›
⟨*proof*⟩

definition *lbd-sort-clauses-raw* :: ‹*arena* ⇒ *vdom* ⇒ *nat* ⇒ *nat* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses-raw arena N* = *pslice-sort-spec idx-cdom clause-score-less arena N*›

definition *lbd-sort-clauses* :: ‹*arena* ⇒ *vdom* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses arena N* = *lbd-sort-clauses-raw arena N 0* (*length N*)›

lemmas *LBD-introsort*[*sepref-fr-rules*] =
LBD-it.introsort-param-impl-correct[*unfolded lbd-sort-clauses-raw-def*[*symmetric*] *PR-CONST-def*]

lemma *quicksort-clauses-by-score-sort*:
‹(*lbd-sort-clauses*, *sort-clauses-by-score*) ∈
Id → *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

sepref-register *lbd-sort-clauses-raw*
sepref-def *lbd-sort-clauses-impl*
is ‹*uncurry lbd-sort-clauses*›
:: ‹$\mathit{arena\text{-}fast\text{-}assn}^k *_a \mathit{vdom\text{-}fast\text{-}assn}^d \rightarrow_a \mathit{vdom\text{-}fast\text{-}assn}$›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] =
lbd-sort-clauses-impl.refine[*FCOMP quicksort-clauses-by-score-sort*]

sepref-register *remove-deleted-clauses-from-avdom arena-status DELETED*

sepref-def *remove-deleted-clauses-from-avdom-fast-code*
is ‹*uncurry isa-remove-deleted-clauses-from-avdom*›
:: ‹$[\lambda(N,\ \mathit{vdom}).\ \mathit{length}\ \mathit{vdom} \leq \mathit{sint64\text{-}max}]_a\ \mathit{arena\text{-}fast\text{-}assn}^k *_a \mathit{vdom\text{-}fast\text{-}assn}^d \rightarrow \mathit{vdom\text{-}fast\text{-}assn}$›

⟨*proof*⟩

sepref-def *sort-vdom-heur-fast-code*
is ‹*sort-vdom-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-register *max-restart-decision-lvl*

sepref-def *minimum-number-between-restarts-impl*
is ‹*uncurry0* (*RETURN minimum-number-between-restarts*)›
:: ‹$unit\text{-}assn^k \rightarrow_a word\text{-}assn$›
⟨*proof*⟩

sepref-def *uint32-nat-assn-impl*
is ‹*uncurry0* (*RETURN max-restart-decision-lvl*)›
:: ‹$unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-def *get-reductions-count-fast-code*
is ‹*RETURN o get-reductions-count*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a word\text{-}assn$›
⟨*proof*⟩

sepref-register *get-reductions-count*

lemma *of-nat-snat*:
‹(*id*,*of-nat*) ∈ *snat-rel' TYPE*(*'a*::*len2*) → *word-rel*›
⟨*proof*⟩

sepref-def *GC-required-heur-fast-code*
is ‹*uncurry GC-required-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
⟨*proof*⟩

sepref-register *ema-get-value get-fast-ema-heur get-slow-ema-heur*
sepref-def *restart-required-heur-fast-code*
is ‹*uncurry restart-required-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a word\text{-}assn$›
⟨*proof*⟩

sepref-register *isa-trail-nth isasat-trail-nth-st*

sepref-def *isasat-trail-nth-st-code*
is ‹*uncurry isasat-trail-nth-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

sepref-register *get-the-propagation-reason-pol-st*

sepref-def *get-the-propagation-reason-pol-st-code*
is ‹*uncurry get-the-propagation-reason-pol-st*›

:: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a snat\text{-}option\text{-}assn'$ *TYPE*(*64*)›
⟨*proof*⟩

sepref-register *isasat-replace-annot-in-trail*
sepref-def *isasat-replace-annot-in-trail-code*
is ‹*uncurry2 isasat-replace-annot-in-trail*›
:: ‹$unat\text{-}lit\text{-}assn^k *_a (sint64\text{-}nat\text{-}assn)^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-register *mark-garbage-heur2*
sepref-def *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*]$_a$
$sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-register *remove-one-annot-true-clause-one-imp-wl-D-heur*
term *mark-garbage-heur2*
sepref-def *remove-one-annot-true-clause-one-imp-wl-D-heur-code*
is ‹*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a sint64\text{-}nat\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩
sepref-register *mark-clauses-as-unused-wl-D-heur*

sepref-def *access-vdom-at-fast-code*
is ‹*uncurry* (*RETURN oo access-vdom-at*)›
:: ‹[*uncurry access-vdom-at-pre*]$_a$ $isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow sint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

sepref-register *remove-one-annot-true-clause-imp-wl-D-heur*

sepref-def *remove-one-annot-true-clause-imp-wl-D-heur-code*
is ‹*remove-one-annot-true-clause-imp-wl-D-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

lemma *length-ll*[*def-pat-rules*]: ‹*length-ll*\$*xs*\$*i* ≡ *op-list-list-llen*\$*xs*\$*i*›
⟨*proof*⟩

lemma [*def-pat-rules*]: ‹*nth-rll* ≡ *op-list-list-idx*›
⟨*proof*⟩

sepref-register *length-ll extra-information-mark-to-delete nth-rll*
LEARNED

lemma *isasat-GC-clauses-prog-copy-wl-entry-alt-def*:
‹*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, -, -) = (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 MAX-HEADER-SIZE else MIN-HEADER-SIZE)
+ 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 MAX-HEADER-SIZE else MIN-HEADER-SIZE);
          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))
  })›
⟨proof⟩

sepref-def isasat-GC-clauses-prog-copy-wl-entry-code
  is ‹uncurry3 isasat-GC-clauses-prog-copy-wl-entry›
  :: ‹[λ(((N, -), -), -). length N ≤ sint64-max]_a
     arena-fast-assn^d *_a watchlist-fast-assn^k *_a unat-lit-assn^k *_a
         (arena-fast-assn ×_a vdom-fast-assn ×_a vdom-fast-assn)^d →
     (arena-fast-assn ×_a (arena-fast-assn ×_a vdom-fast-assn ×_a vdom-fast-assn))›
  ⟨proof⟩

sepref-register isasat-GC-clauses-prog-copy-wl-entry

lemma shorten-taken-op-list-list-take:
  ‹W[L := []] = op-list-list-take W L 0›
  ⟨proof⟩

sepref-def isasat-GC-clauses-prog-single-wl-code
  is ‹uncurry3 isasat-GC-clauses-prog-single-wl›
  :: ‹[λ(((N, -), -), A). A ≤ uint32-max div 2 ∧ length N ≤ sint64-max]_a
     arena-fast-assn^d *_a (arena-fast-assn ×_a vdom-fast-assn ×_a vdom-fast-assn)^d *_a watchlist-fast-assn^d
*_a atom-assn^k →
     (arena-fast-assn ×_a (arena-fast-assn ×_a vdom-fast-assn ×_a vdom-fast-assn) ×_a watchlist-fast-assn)›
  ⟨proof⟩


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 isasat-GC-clauses-prog-single-wl
sepref-def isasat-GC-clauses-prog-wl2-code
  is ‹uncurry2 isasat-GC-clauses-prog-wl2′›
  :: ‹[λ((-, -), (N, -)). length N ≤ sint64-max]_a
     (array-assn vmtf-node-assn)^k *_a (atom.option-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* $\times_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*) $\times_a$ *watchlist-fast-assn*)›
⟨*proof*⟩

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

sepref-register *isasat-GC-clauses-prog-wl isasat-GC-clauses-prog-wl2' rewatch-heur-st*
sepref-def *isasat-GC-clauses-prog-wl-code*
is ‹*isasat-GC-clauses-prog-wl*›
:: ‹[λS. *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

lemma *rewatch-heur-st-pre-alt-def*:
‹*rewatch-heur-st-pre S* $\longleftrightarrow$ ($\forall$ *i* $\in$ *set* (*get-vdom S*). *i* $\leq$ *sint64-max*)›
⟨*proof*⟩

sepref-def *rewatch-heur-st-code*
is ‹*rewatch-heur-st*›
:: ‹[λS. *rewatch-heur-st-pre S* $\wedge$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-register *isasat-GC-clauses-wl-D*

sepref-def *isasat-GC-clauses-wl-D-code*
is ‹*isasat-GC-clauses-wl-D*›
:: ‹[λS. *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-register *number-clss-to-keep*

sepref-register *access-vdom-at*

lemma [*sepref-fr-rules*]:
‹(*return o id*, *RETURN o unat*) $\in$ *word64-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-def *number-clss-to-keep-fast-code*
is ‹*number-clss-to-keep-impl*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹(*number-clss-to-keep-impl*, *number-clss-to-keep*) $\in$ *Sepref-Rules.freft Id* (λ-. ⟨*nat-rel*⟩*nres-rel*)›
⟨*proof*⟩

lemma *number-clss-to-keep-fast-code-refine*[*sepref-fr-rules*]:
‹(*number-clss-to-keep-fast-code*, *number-clss-to-keep*) $\in$ (*isasat-bounded-assn*)^k $\rightarrow_a$ *snat-assn*›
⟨*proof*⟩

sepref-def *mark-clauses-as-unused-wl-D-heur-fast-code*
is ‹*uncurry mark-clauses-as-unused-wl-D-heur*›

:: ‹$[\lambda(\text{-}, S).\ \mathit{length}\ (\mathit{get\text{-}avdom}\ S) \leq \mathit{sint64\text{-}max}]_a$
  $\mathit{sint64\text{-}nat\text{-}assn}^k *_a \mathit{isasat\text{-}bounded\text{-}assn}^d \rightarrow \mathit{isasat\text{-}bounded\text{-}assn}$›
⟨*proof*⟩

experiment
begin
  export-llvm *restart-required-heur-fast-code*
    access-vdom-at-fast-code
    isasat-GC-clauses-wl-D-code
end

end
theory *IsaSAT-Restart*
  imports *IsaSAT-Restart-Heuristics IsaSAT-CDCL*
begin

Chapter 20

Full CDCL with Restarts

```
definition cdcl-twl-stgy-restart-abs-wl-heur-inv where
  ‹cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ brk T n ⟷
    (∃S₀' T'. (S₀, S₀') ∈ twl-st-heur ∧ (T, T') ∈ twl-st-heur ∧
      cdcl-twl-stgy-restart-abs-wl-inv S₀' 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 S₀ = do {
    (brk, T, -) ← WHILE_T^λ(brk, T, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ 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, S₀::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) ∈
    twl-st-heur →_f ⟨twl-st-heur⟩nres-rel›
⟨proof⟩
```

```
definition fast-number-of-iterations :: ‹- ⇒ bool› where
‹fast-number-of-iterations n ⟷ n < uint64-max >> 1›
```

```
definition isasat-fast-slow :: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres› where
   [simp]: ‹isasat-fast-slow S = RETURN S›
```

```
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 S₀ = do {
    ebrk ← RETURN (¬isasat-fast S₀);
    (ebrk, brk, T, n) ←
    WHILE_T^λ(ebrk, brk, T, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ brk T n ∧        (¬ebrk ⟶isasat-fast T) ∧ length (get-c
      (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
      (λ(ebrk, brk, S, n).
      do {
        ASSERT(¬brk ∧ ¬ebrk);
        ASSERT(length (get-clauses-wl-heur S) ≤ uint64-max);
        T ← unit-propagation-outer-loop-wl-D-heur S;
        ASSERT(length (get-clauses-wl-heur T) ≤ uint64-max);
        ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S));
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        ASSERT(length (get-clauses-wl-heur T) ≤ uint64-max);
        (T, n) ← restart-prog-wl-D-heur T n brk;
ebrk ← RETURN (¬isasat-fast T);
        RETURN (ebrk, brk, T, n)
      })
      (ebrk, False, S₀::twl-st-wl-heur, 0);
    ASSERT(length (get-clauses-wl-heur T) ≤ uint64-max ∧
        get-old-arena T = []);
    if ¬brk then do {
       T ← isasat-fast-slow T;
       (brk, T, -) ← WHILE_T^{λ(brk, T, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ 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, T, n);
       RETURN T
    }
    else isasat-fast-slow T
  })
```

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
  assumes *r*: ‹$r \leq$ *uint64-max*›
  shows ‹(*cdcl-twl-stgy-restart-prog-early-wl-heur*, *cdcl-twl-stgy-restart-prog-early-wl*) ∈
   twl-st-heur''' *r* $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
  ‹*D* = *Some* *C* ⟷ *D* ≠ *None* ∧ ((*the* *D*) = *C*)›
  ⟨*proof*⟩

lemma (**in** −) *isasat-fast-alt-def*:
  ‹*RETURN* *o* *isasat-fast* = (λ(*M*, *N*, -). *RETURN* (*length* *N* ≤ *sint64-max* − (*uint32-max* *div* *2* + *MAX-HEADER-SIZE* + *1*)))›
  ⟨*proof*⟩

definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
  :: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where

```
 ‹cdcl-twl-stgy-restart-prog-bounded-wl-heur S₀ = do {
   ebrk ← RETURN (¬isasat-fast S₀);
   (ebrk, brk, T, n) ←
   WHILE_T^λ(ebrk, brk, T, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ brk T n ∧         (¬ebrk ⟶isasat-fast T ∧ n < uint64-m
     (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
     (λ(ebrk, brk, S, n).
     do {
       ASSERT(¬brk ∧ ¬ebrk);
       ASSERT(length (get-clauses-wl-heur S) ≤ sint64-max);
       T ← unit-propagation-outer-loop-wl-D-heur S;
       ASSERT(length (get-clauses-wl-heur T) ≤ sint64-max);
       ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S));
       (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
       ASSERT(length (get-clauses-wl-heur T) ≤ sint64-max);
       (T, n) ← restart-prog-wl-D-heur T n brk;
 ebrk ← RETURN (¬(isasat-fast T ∧ n < uint64-max));
       RETURN (ebrk, brk, T, n)
     })
     (ebrk, False, S₀::twl-st-wl-heur, 0);
   RETURN (brk, T)
 }›
```

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D*:
  assumes *r*: ‹$r \leq$ *uint64-max*›
  shows ‹(*cdcl-twl-stgy-restart-prog-bounded-wl-heur*, *cdcl-twl-stgy-restart-prog-bounded-wl*) ∈
   twl-st-heur''' $r \rightarrow_f$ ⟨*bool-rel* $\times_r$ *twl-st-heur*⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-Restart-LLVM*
  imports *IsaSAT-Restart IsaSAT-Restart-Heuristics-LLVM IsaSAT-CDCL-LLVM*
begin

sepref-register *mark-to-delete-clauses-wl-D-heur*

sepref-def *MINIMUM-DELETION-LBD-impl*
  is ‹*uncurry0* (*RETURN MINIMUM-DELETION-LBD*)›
  :: ‹*unit-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

sepref-register *delete-index-and-swap mop-mark-garbage-heur*

sepref-def *mark-to-delete-clauses-wl-D-heur-fast-impl*
  is ‹*mark-to-delete-clauses-wl-D-heur*›
  :: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  ⟨*proof*⟩

sepref-register *cdcl-twl-full-restart-wl-prog-heur*

sepref-def *cdcl-twl-full-restart-wl-prog-heur-fast-code*
is ‹*cdcl-twl-full-restart-wl-prog-heur*›
:: ‹$[\lambda S.$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *cdcl-twl-restart-wl-heur-fast-code*
is ‹*cdcl-twl-restart-wl-heur*›
:: ‹$[\lambda S.$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code*
is ‹*cdcl-twl-full-restart-wl-D-GC-heur-prog*›
:: ‹$[\lambda S.$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-register *restart-required-heur cdcl-twl-restart-wl-heur*

sepref-def *restart-prog-wl-D-heur-fast-code*
is ‹*uncurry2* (*restart-prog-wl-D-heur*)›
:: ‹$[\lambda((S, n), \text{-}).$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max* $\wedge$ $n <$ *uint64-max*$]_a$
isasat-bounded-assn^d $*_a$ *uint64-nat-assn*^{k} $*_a$ *bool1-assn*^{k} $\rightarrow$ *isasat-bounded-assn* $\times_a$ *uint64-nat-assn*›
⟨*proof*⟩

definition *isasat-fast-bound* **where**
‹*isasat-fast-bound* = *uint64-max* − (*uint32-max div 2* + *6*)›

lemma *isasat-fast-bound-alt-def*:
‹*isasat-fast-bound* = *18446744071562067962*›
⟨*proof*⟩

sepref-register *isasat-fast*
sepref-def *isasat-fast-code*
is ‹*RETURN o isasat-fast*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-register *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
sepref-def *cdcl-twl-stgy-restart-prog-wl-heur-fast-code*
is ‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur*›
:: ‹$[\lambda S.$ *isasat-fast S*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *bool1-assn* $\times_a$ *isasat-bounded-assn*›
⟨*proof*⟩

experiment
begin
export-llvm *opts-reduction-st-fast-code*
opts-restart-st-fast-code
get-conflict-count-since-last-restart-heur-fast-code
get-fast-ema-heur-fast-code
get-slow-ema-heur-fast-code
get-learned-count-fast-code
count-decided-st-heur-pol-fast
upper-restart-bound-not-reached-fast-impl
minimum-number-between-restarts-impl

restart-required-heur-fast-code
cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code
cdcl-twl-restart-wl-heur-fast-code
cdcl-twl-full-restart-wl-prog-heur-fast-code
cdcl-twl-local-restart-wl-D-heur-fast-code

end

end
theory *IsaSAT*
imports *IsaSAT-Restart IsaSAT-Initialisation*
begin

Chapter 21

Full IsaSAT

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.

21.1 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(λ*T*. ∃ *n n'*. *cdcl-twl-stgy-restart-with-leftovers*** (*S*, *n*) (*T*, *n'*) ∧ *final-twl-state T*)›

definition *conclusive-TWL-run-bounded* :: ‹*'v twl-st* ⇒ (*bool* × *'v twl-st*) *nres*› **where**
‹*conclusive-TWL-run-bounded S* =
SPEC(λ(*brk*, *T*). ∃ *n n'*. *cdcl-twl-stgy-restart-with-leftovers*** (*S*, *n*) (*T*, *n'*) ∧
(*brk* ⟶ *final-twl-state T*))›

To get a full CDCL run:

- either we fully apply *cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy* (up to restarts)
- or we can stop early.

definition *conclusive-CDCL-run* **where**
‹*conclusive-CDCL-run CS T U* ⟷
(∃ *n n'*. *cdcl*$_W$*-restart-mset.cdcl*$_W$*-restart-stgy*** (*T*, *n*) (*U*, *n'*) ∧

no-step cdcl$_W$*-restart-mset.cdcl*$_W$ (*U*)) ∨
(*CS* ≠ {#} ∧ *conflicting U* ≠ *None* ∧ *count-decided* (*trail U*) = *0* ∧
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*›
⟨*proof*⟩

lemma *cdcl-twl-stgy-restart-prog-bounded-spec*: ‹*twl-struct-invs S* ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog-bounded S ≤ *conclusive-TWL-run-bounded S*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-cdcl*$_W$*-init-state*:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$** (*init-state* {#}) *S* ⟷ *S* = *init-state* {#}›
⟨*proof*⟩

definition *init-state-l* :: ‹*'v twl-st-l-init*› **where**
‹*init-state-l* = (([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›

definition *to-init-state-l* :: ‹*nat twl-st-l-init* ⇒ *nat twl-st-l-init*› **where**
‹*to-init-state-l S* = *S*›

definition *init-state0* :: ‹*'v twl-st-init*› **where**
‹*init-state0* = (([], {#}, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›

definition *to-init-state0* :: ‹*nat twl-st-init* ⇒ *nat twl-st-init*› **where**
‹*to-init-state0 S* = *S*›

lemma *init-dt-pre-init*:
assumes *dist*: ‹*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*›
shows ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
⟨*proof*⟩

This is the specification of the SAT solver:

definition *SAT* :: ‹*nat clauses* ⇒ *nat cdcl*$_W$*-restart-mset nres*› **where**
‹*SAT CS* = *do*{
let T = *init-state CS*;

```
  SPEC (conclusive-CDCL-run CS T)
 }⟩
```

```
definition init-dt-spec0 :: ⟨′v clause-l list ⇒ ′v twl-st-init ⇒ ′v twl-st-init ⇒ bool⟩ where
 ⟨init-dt-spec0 CS SOC T′ ⟷
    (
     twl-struct-invs-init T′ ∧
     clauses-to-update-init T′ = {#} ∧
     (∀ s∈set (get-trail-init T′). ¬is-decided s) ∧
     (get-conflict-init T′ = None ⟶
 literals-to-update-init T′ = uminus ‘# lit-of ‘# mset (get-trail-init T′)) ∧
     (mset ‘# mset CS + clause ‘# (get-init-clauses-init SOC) + other-clauses-init SOC +
    get-unit-init-clauses-init SOC + get-subsumed-init-clauses-init SOC =
      clause ‘# (get-init-clauses-init T′) + other-clauses-init T′ +
    get-unit-init-clauses-init T′ + get-subsumed-init-clauses-init T′) ∧
     get-learned-clauses-init SOC = get-learned-clauses-init T′ ∧
     get-subsumed-learned-clauses-init SOC = get-subsumed-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′))⟩
```

21.2 Refinements of the Whole SAT Solver

We do not 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 + subsumed-clauses T = mset ‘# mset CS);
        ASSERT(get-learned-clss T = {#});
        ASSERT(subsumed-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 + subsumed-clauses 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 + subsumed-clauses 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*⟩

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 +
              get-subsumed-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 +
              get-subsumed-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 +
              get-subsumed-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*⟩

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 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 𝒜_in′));
  ASSERT(mset `# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T = mset `# mset CS);
  ASSERT(learned-clss-l (get-clauses-wl T) = {#});
  cdcl-twl-stgy-restart-prog-wl (finalise-init T)
        }
    }
```

```
   else do {
       let S = init-state-wl;
       T ← init-dt-wl′ CS (to-init-state S);
       let T = from-init-state T;
       failed ← SPEC (λ- :: bool. True);
       if failed then do {
         let S = init-state-wl;
         T ← init-dt-wl′ CS (to-init-state S);
         T ← rewatch-st (from-init-state T);
         if get-conflict-wl T ≠ None
         then RETURN T
         else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, λ-. undefined))
         else do {
           ASSERT (extract-atms-clss CS {} ≠ {});
           ASSERT(isasat-input-bounded-nempty (mset-set 𝒜ᵢₙ′));
           ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
            get-subsumed-clauses-wl T  = mset ‘# mset CS);
           ASSERT(learned-clss-l (get-clauses-wl T) = {#});
           cdcl-twl-stgy-restart-prog-wl (finalise-init T)
         }
       } else do {
         if get-conflict-wl T ≠ None
         then RETURN T
         else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, λ-. undefined))
         else do {
           ASSERT (extract-atms-clss CS {} ≠ {});
           ASSERT(isasat-input-bounded-nempty (mset-set 𝒜ᵢₙ′));
           ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
            get-subsumed-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 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 +
             get-subsumed-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) +
              get-subsumed-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) +
              get-subsumed-clauses-l (fst T) = mset `# mset CS);
            ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
            let T = fst T;
            cdcl-twl-stgy-restart-prog-early-l T
          }
        }
      }
    }›
    ⟨proof⟩
```

lemma *init-dt-wl-full-init-dt-wl-spec-full*:
  assumes ‹*init-dt-wl-pre CS S*› **and** ‹*init-dt-pre CS S'*› **and**
    ‹$(S, S') \in$ *state-wl-l-init*› **and** ‹$\forall C \in$ *set CS. distinct C*›
  shows ‹*init-dt-wl-full CS S* $\leq \Downarrow$ {(S, S'). (*fst S*, *fst S'*) $\in$ *state-wl-l None*} (*init-dt CS S'*)›
⟨*proof*⟩

lemma *init-dt-wl-pre*:
  assumes *dist*: ‹*Multiset.Ball* (*mset `# mset CS*) *distinct-mset*›
  shows ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
  ⟨*proof*⟩

```
lemma SAT-wl-SAT-l:
  assumes
    dist: ‹Multiset.Ball (mset ‘# mset CS) distinct-mset› and
    bounded: ‹isasat-input-bounded (mset-set (⋃ C∈set CS. atm-of ‘ set C))›
  shows ‹SAT-wl CS ≤ ⇓ {(T,T′). (T, T′) ∈ state-wl-l None} (SAT-l CS)›
⟨proof⟩

definition extract-model-of-state where
  ‹extract-model-of-state U = Some (map lit-of (get-trail-wl U))›

definition extract-model-of-state-heur where
  ‹extract-model-of-state-heur U = Some (fst (get-trail-wl-heur U))›

definition extract-stats where
  [simp]: ‹extract-stats U = None›

definition extract-stats-init where
  [simp]: ‹extract-stats-init = None›

definition IsaSAT :: ‹nat clause-l list ⇒ nat literal list option nres› where
  ‹IsaSAT CS = do{
    S ← SAT-wl CS;
    RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
  }›


lemma IsaSAT-alt-def:
  ‹IsaSAT CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    ASSERT(distinct-mset-set (mset ‘ set CS));
    let 𝒜in′ = 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 𝒜in′));
           ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
              get-subsumed-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 (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 +
            get-subsumed-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 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 +
            get-subsumed-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 T;
          RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
        }
      }
    }
  }› (is ‹?A = ?B›) for CS opts
⟨proof⟩
```

definition *extract-model-of-state-stat* :: ‹*twl-st-wl-heur* ⇒ *bool* × *nat literal list* × *stats*› **where**

```
  ‹extract-model-of-state-stat U =
     (False, (fst (get-trail-wl-heur U)),
       (λ(M, -,  -, -, - ,- ,- ,-, -, -,  stat, -, -). stat) U)›
```

definition *extract-state-stat* :: ‹*twl-st-wl-heur* ⇒ *bool* × *nat literal list* × *stats*› **where**

```
  ‹extract-state-stat U =
     (True, [],
       (λ(M, -, -, -, - ,- ,- ,-, -, -, stat, -, -). stat) U)›
```

definition *empty-conflict* :: ‹*nat literal list option*› **where**

```
  ‹empty-conflict = Some []›
```

definition *empty-conflict-code* :: ‹(*bool* × - *list* × *stats*) *nres*› **where**

```
  ‹empty-conflict-code = do{
     let M0 = [];
     RETURN (False, M0, (0, 0, 0, 0, 0, 0, 0, ema-fast-init))}›
```

definition *empty-init-code* :: ‹*bool* × - *list* × *stats*› **where**

```
⟨empty-init-code = (True, [], (0, 0, 0, 0, 0, 0, 0, ema-fast-init))⟩
```

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) ≤ sint64-max − (uint32-max div 2 +
MAX-HEADER-SIZE+1))⟩
```

definition *IsaSAT-heur* :: ⟨*opts* ⇒ *nat clause-l list* ⇒ (*bool* × *nat literal list* × *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 ≤ uint32-max);
   let 𝒜_in' = mset-set (extract-atms-clss CS {});
   ASSERT(isasat-input-bounded 𝒜_in');
   ASSERT(distinct-mset 𝒜_in');
   let 𝒜_in'' = virtual-copy 𝒜_in';
   let b = opts-unbounded-mode opts;
   if b
   then do {
       S ← init-state-wl-heur 𝒜_in';
       (T::twl-st-wl-heur-init) ←  init-dt-wl-heur True CS S;
 T ← rewatch-heur-st T;
       let T = convert-state 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(𝒜_in'' ≠ {#});
          ASSERT(isasat-input-bounded-nempty 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 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 𝒜_in' = mset-set (extract-atms-clss CS {});
         S ← init-state-wl-heur 𝒜_in';
         (T::twl-st-wl-heur-init) ← init-dt-wl-heur True CS S;
         let T = convert-state 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($\mathcal{A}_{in}'' \neq \{\#\}$);
           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 {
          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 $\mathcal{A}_{in}''$);
             - ← 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);
             U ← 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)
           }
         }
       }
    }›
```

lemma *fref-to-Down-unRET-uncurry0-SPEC*:
  assumes ‹(λ-. (*f*), λ-. (*RETURN g*)) ∈ $[P]_f$ *unit-rel* → ⟨*B*⟩*nres-rel*› **and** ‹*P* ()›
  shows ‹*f* ≤ *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)›
⟨*proof*⟩

lemma *fref-to-Down-unRET-SPEC*:
  assumes ‹(*f*, *RETURN o g*) ∈ $[P]_f$ *A* → ⟨*B*⟩*nres-rel*› **and**
    ‹*P y*› **and**
    ‹(*x*, *y*) ∈ *A*›
  shows ‹*f x* ≤ *SPEC* (λ*c*. (*c*, *g y*) ∈ *B*)›
⟨*proof*⟩

lemma *fref-to-Down-unRET-curry-SPEC*:
  assumes ‹(*uncurry f*, *uncurry* (*RETURN oo g*)) ∈ $[P]_f$ *A* → ⟨*B*⟩*nres-rel*› **and**
    ‹*P* (*x*, *y*)› **and**
    ‹((*x*′, *y*′), (*x*, *y*)) ∈ *A*›

shows ‹*f x′ y′ ≤ SPEC (λc. (c, g x y) ∈ B)*›
⟨*proof*⟩

lemma *all-lits-of-mm-empty-iff*: ‹*all-lits-of-mm A = {#} ⟷ (∀ C ∈# A. C = {#})*›
⟨*proof*⟩

lemma *all-lits-of-mm-extract-atms-clss*:
‹*L ∈# (all-lits-of-mm (mset ‘# mset CS)) ⟷ atm-of L ∈ extract-atms-clss CS {}*›
⟨*proof*⟩

lemma *IsaSAT-heur-alt-def*:

```
‹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 ≤ uint32-max);
   let 𝒜in′ = mset-set (extract-atms-clss CS {});
   ASSERT(isasat-input-bounded 𝒜in′);
   ASSERT(distinct-mset 𝒜in′);
   let 𝒜in″ = virtual-copy 𝒜in′;
   let b = opts-unbounded-mode opts;
   if b
   then do {
       S ← init-state-wl-heur 𝒜in′;
       (T::twl-st-wl-heur-init) ←  init-dt-wl-heur True CS S;
       T ← rewatch-heur-st T;
       let T = convert-state 𝒜in″ 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(𝒜in″ ≠ {#});
          ASSERT(isasat-input-bounded-nempty 𝒜in″);
           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 𝒜in′;
       (T::twl-st-wl-heur-init) ← init-dt-wl-heur False CS S;
       failed ← RETURN (is-failed-heur-init T ∨ ¬isasat-fast-init T);
       if failed then do {
          S ← init-state-wl-heur 𝒜in′;
         (T::twl-st-wl-heur-init) ← init-dt-wl-heur True CS S;
         T ← rewatch-heur-st T;
         let T = convert-state 𝒜in″ 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(𝒜in″ ≠ {#});
          ASSERT(isasat-input-bounded-nempty 𝒜in″);
```

```
            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 {
          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 $\mathcal{A}_{in}''$);
             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);
             U ← cdcl-twl-stgy-restart-prog-early-wl-heur T;
             RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
               else extract-state-stat U)
           }
        }
      }
    }⟩
  ⟨proof⟩
```

abbreviation *rewatch-heur-st-rewatch-st-rel* **where**

```
  ⟨rewatch-heur-st-rewatch-st-rel CS U V ≡
    {(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-subsumed-init-clauses-wl (fst T) = get-subsumed-init-clauses-wl (fst V) ∧
  get-subsumed-learned-clauses-wl (fst T) = get-subsumed-learned-clauses-wl (fst V) ∧
  get-unit-init-clss-wl (fst T) = get-unit-init-clss-wl (fst V) ∧
  get-unit-learned-clss-wl (fst T) = get-unit-learned-clss-wl (fst V) ∧
  get-unit-clauses-wl (fst T) = get-unit-clauses-wl (fst V)} O {(S, T). S = (T, {#})}⟩
```

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 ≤
    ⇓(rewatch-heur-st-rewatch-st-rel CS U V)
          (rewatch-st (from-init-state V))⟩
⟨proof⟩
```

lemma *rewatch-heur-st-rewatch-st2*:
  assumes
    T: ‹(*U*, *V*)
      ∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
        {(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
  shows ‹*rewatch-heur-st-fast*
          (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
        ≤ ⇓ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
        get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
  get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
        get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
  get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
  get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})})
            (*rewatch-st* (*from-init-state V*))›
⟨*proof*⟩

lemma *rewatch-heur-st-rewatch-st3*:
  assumes
    T: ‹(*U*, *V*)
      ∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
        {(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
    failed: ‹¬*is-failed-heur-init U*›
  shows ‹*rewatch-heur-st-fast*
          (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
        ≤ ⇓ (*rewatch-heur-st-rewatch-st-rel CS U V*)
            (*rewatch-st* (*from-init-state V*))›
⟨*proof*⟩

abbreviation *option-with-bool-rel* :: ‹((*bool* × *'a*) × *'a option*) *set*› **where**
  ‹*option-with-bool-rel* ≡ {((*b*, *s*), *s'*). (*b* = *is-None s'*) ∧ (¬*b* ⟶ *s* = *the s'*)}›

definition *model-stat-rel* :: ‹((*bool* × *nat literal list* × *'a*) × *nat literal list option*) *set*› **where**
  ‹*model-stat-rel* = {((*b*, *M'*, *s*), *M*). ((*b*, *rev M'*), *M*) ∈ *option-with-bool-rel*}›

lemma *IsaSAT-heur-IsaSAT*:
  ‹*IsaSAT-heur b CS* ≤ ⇓*model-stat-rel* (*IsaSAT CS*)›
⟨*proof*⟩

definition *length-get-clauses-wl-heur-init* **where**
  ‹*length-get-clauses-wl-heur-init S* = *length* (*get-clauses-wl-heur-init S*)›

lemma *length-get-clauses-wl-heur-init-alt-def*:
  ‹*RETURN o length-get-clauses-wl-heur-init* = (λ(-, *N*,-). *RETURN* (*length N*))›
  ⟨*proof*⟩

definition *model-if-satisfiable* :: ‹*nat clauses* ⇒ *nat literal list option nres*› **where**
  ‹*model-if-satisfiable CS* = *SPEC* (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) ∈ [λCS. (∀ C ∈# CS. distinct-mset C)]_f Id→ ⟨Id⟩nres-rel›
    (is ‹- ∈[λCS. ?P CS]_f Id → -›)
⟨proof⟩

lemma SAT-model-if-satisfiable′:
  ‹(uncurry (λ-. SAT′), uncurry (λ-. model-if-satisfiable)) ∈
    [λ(-, CS). (∀ C ∈# CS. distinct-mset C)]_f Id ×_r Id→ ⟨Id⟩nres-rel›
  ⟨proof⟩

definition SAT-l′ where
  ‹SAT-l′ CS = do{
    S ← SAT-l CS;
    RETURN (if get-conflict-l S = None then Some (map lit-of (get-trail-l S)) else None)
  }›


definition SAT0′ where
  ‹SAT0′ CS = do{
    S ← SAT0 CS;
    RETURN (if get-conflict S = None then Some (map lit-of (get-trail S)) else None)
  }›


lemma twl-st-l-map-lit-of[twl-st-l, simp]:
  ‹(S, T) ∈ twl-st-l b ⟹ map lit-of (get-trail-l S) = map lit-of (get-trail T)›
  ⟨proof⟩


lemma ISASAT-SAT-l′:
  assumes ‹Multiset.Ball (mset ‘# mset CS) distinct-mset› and
    ‹isasat-input-bounded (mset-set (⋃ C∈set CS. atm-of ‘ set C))›
  shows ‹IsaSAT CS ≤ ⇓ Id (SAT-l′ CS)›
  ⟨proof⟩

lemma SAT-l′-SAT0′:
  assumes ‹Multiset.Ball (mset ‘# mset CS) distinct-mset›
  shows ‹SAT-l′ CS ≤ ⇓ Id (SAT0′ CS)›
  ⟨proof⟩

lemma SAT0′-SAT′:
  assumes ‹Multiset.Ball (mset ‘# mset CS) distinct-mset›
  shows ‹SAT0′ CS ≤ ⇓ Id (SAT′ (mset ‘# mset CS))›
  ⟨proof⟩


lemma IsaSAT-heur-model-if-sat:
  assumes ‹∀ 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))›
  ⟨proof⟩
```

```
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 ≤ uint32-max)]_f
    Id ×_r list-mset-rel O ⟨list-mset-rel⟩mset-rel → ⟨model-stat-rel⟩nres-rel›
⟨proof⟩
```

21.3 Refinements of the Whole Bounded SAT Solver

This is the specification of the SAT solver:

```
definition SAT-bounded :: ‹nat clauses ⇒ (bool × nat cdcl_W-restart-mset) nres› where
  ‹SAT-bounded CS = do{
    T ← SPEC(λT. T = init-state CS);
    finished ← SPEC(λ-. True);
    if ¬finished then
      RETURN (finished, T)
    else
      SPEC (λ(b, U). b ⟶ conclusive-CDCL-run CS T U)
  }›

definition  SAT0-bounded :: ‹nat clause-l list ⇒ (bool × nat twl-st) nres› where
  ‹SAT0-bounded CS = do{
    let (S :: nat twl-st-init) = init-state0;
    T ←  SPEC (λT. init-dt-spec0 CS (to-init-state0 S) T);
    finished ← SPEC(λ-. True);
    if ¬finished then do {
      RETURN (False, fst init-state0)
    } else do {
      let T = fst T;
      if get-conflict T ≠ None
      then RETURN (True, T)
      else if CS = [] then RETURN (True, fst init-state0)
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT (clauses-to-update T = {#});
        ASSERT(clause `# (get-clauses T) + unit-clss T + subsumed-clauses T = mset  `# mset CS);
        ASSERT(get-learned-clss T = {#});
        cdcl-twl-stgy-restart-prog-bounded T
      }
    }
 }›

lemma SAT0-bounded-SAT-bounded:
  assumes ‹Multiset.Ball (mset `# mset CS) distinct-mset›
  shows ‹SAT0-bounded CS ≤ ⇓ ({((b, S), (b', T)). b = b' ∧ (b ⟶ T = state_W-of S)}) (SAT-bounded
(mset `# mset CS))›
    (is ‹- ≤ ⇓?A -›)
⟨proof⟩

definition  SAT-l-bounded :: ‹nat clause-l list ⇒ (bool × nat twl-st-l) nres› where
  ‹SAT-l-bounded CS = do{
      let S = init-state-l;
      T ← init-dt CS (to-init-state-l S);
      finished ← SPEC (λ- :: bool. True);
      if ¬finished then do {
        RETURN (False, fst init-state-l)
      } else do {
        let T = fst T;
        if get-conflict-l T ≠ None
        then RETURN (True, T)
        else if CS = [] then RETURN (True, 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 + get-subsumed-clauses-l
T= mset ‘# mset CS);
           ASSERT(learned-clss-l (get-clauses-l T) = {#});
           cdcl-twl-stgy-restart-prog-bounded-l T
        }

    }
 }⟩
```

lemma *SAT-l-bounded-SAT0-bounded*:
  assumes *dist*: ⟨*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*⟩
  shows ⟨*SAT-l-bounded CS* ≤ ⇓ {((*b*, *T*),(*b*′, *T*′)). *b*=*b*′ ∧ (*b* ⟶ (*T*, *T*′) ∈ *twl-st-l None*)} (*SAT0-bounded CS*)⟩
⟨*proof*⟩

definition *SAT-wl-bounded* :: ⟨*nat clause-l list* ⇒ (*bool* × *nat twl-st-wl*) *nres*⟩ **where**

```
  ⟨SAT-wl-bounded CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    ASSERT(distinct-mset-set (mset ‘ set CS));
    let 𝒜ᵢₙ′ = extract-atms-clss CS {};
    let S = init-state-wl;
    T ← init-dt-wl′ CS (to-init-state S);
    let T = from-init-state T;
    finished ← SPEC (λ- :: bool. True);
    if ¬finished then do {
        RETURN(finished, T)
    } else do {
      if get-conflict-wl T ≠ None
      then RETURN (True, T)
     else if CS = [] then RETURN (True, ([], fmempty, None, {#}, {#}, {#}, {#}, {#}, λ-. undefined))
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜ᵢₙ′));
        ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T + get-subsumed-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-bounded-wl T
      }
    }
  }⟩
```

lemma *SAT-l-bounded-alt-def*:

```
  ⟨SAT-l-bounded CS = do{
    𝒜 ← RETURN (); ~~atoms~~
    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 {
      RETURN(failed, fst init-state-l)
    } else do {
      let T = T;
      if get-conflict-l-init T ≠ None
      then RETURN (True, fst T)
      else if CS = [] then RETURN (True, 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) + get-subsumed-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-bounded-l T
      }
    }
  }›
  ⟨proof⟩
```

lemma *SAT-wl-bounded-SAT-l-bounded*:
  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-bounded CS* ≤ ⇓ {((*b*, *T*),(*b'*, *T'*)). *b* =*b'* ∧ (*b* ⟶ (*T*, *T'*) ∈ *state-wl-l None*)} (*SAT-l-bounded CS*)›
⟨*proof*⟩

definition *SAT-bounded'* :: ‹*nat clauses* ⇒ (*bool* × *nat literal list option*) *nres*› **where**

```
  ‹SAT-bounded' CS = do {
      (b, T) ← SAT-bounded CS;
      RETURN(b, if conflicting T = None then Some (map lit-of (trail T)) else None)
  }
›
```

definition *model-if-satisfiable-bounded* :: ‹*nat clauses* ⇒ (*bool* × *nat literal list option*) *nres*› **where**

```
  ‹model-if-satisfiable-bounded CS = SPEC (λ(b, M). b ⟶
           (if satisfiable (set-mset CS) then M ≠ None ∧ set (the M) ⊨sm CS else M = None))›
```

lemma *SAT-bounded-model-if-satisfiable*:
  ‹(*SAT-bounded'*, *model-if-satisfiable-bounded*) ∈ [λ*CS*. (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id*→
      ⟨{((*b*, *S*), (*b'*, *T*)). *b* = *b'* ∧ (*b* ⟶ *S* = *T*)}⟩*nres-rel*›
    (**is** ‹- ∈[λ*CS*. *?P CS*]$_f$ *Id* → -›)
⟨*proof*⟩

lemma *SAT-bounded-model-if-satisfiable'*:
  ‹(*uncurry* (λ-. *SAT-bounded'*), *uncurry* (λ-. *model-if-satisfiable-bounded*)) ∈
     [λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id* $\times_r$ *Id*→ ⟨{((*b*, *S*), (*b'*, *T*)). *b* = *b'* ∧ (*b* ⟶ *S* = *T*)}⟩*nres-rel*›
  ⟨*proof*⟩

```
definition SAT-l-bounded′ where
  ‹SAT-l-bounded′ CS = do{
    (b, S) ← SAT-l-bounded CS;
    RETURN (b, if b ∧ get-conflict-l S = None then Some (map lit-of (get-trail-l S)) else None)
  }›
```

```
definition SAT0-bounded′ where
  ‹SAT0-bounded′ CS = do{
    (b, S) ← SAT0-bounded CS;
    RETURN (b, if b ∧ get-conflict S = None then Some (map lit-of (get-trail S)) else None)
  }›
```

```
lemma SAT-l-bounded′-SAT0-bounded′:
  assumes ‹Multiset.Ball (mset ‘# mset CS) distinct-mset›
  shows ‹SAT-l-bounded′ CS ≤ ⇓ {((b, S), (b′, T)). b = b′ ∧ (b ⟶ S = T)} (SAT0-bounded′ CS)›
  ⟨proof⟩
```

```
lemma SAT0-bounded′-SAT-bounded′:
  assumes ‹Multiset.Ball (mset ‘# mset CS) distinct-mset›
  shows ‹SAT0-bounded′ CS ≤ ⇓ {((b, S), (b′, T)). b = b′ ∧ (b ⟶ S = T)} (SAT-bounded′ (mset ‘#
mset CS))›
  ⟨proof⟩
```

```
definition IsaSAT-bounded :: ‹nat clause-l list ⇒ (bool × nat literal list option) nres› where
  ‹IsaSAT-bounded CS = do{
    (b, S) ← SAT-wl-bounded CS;
    RETURN (b, if b ∧ get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
  }›
```

```
lemma IsaSAT-bounded-alt-def:
  ‹IsaSAT-bounded CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    ASSERT(distinct-mset-set (mset ‘ set CS));
    let 𝒜ᵢₙ′ = extract-atms-clss CS {};
    S ← RETURN init-state-wl;
    T ← init-dt-wl′ CS (to-init-state S);
    failed ← SPEC (λ- :: bool. True);
    if ¬failed then do {
        RETURN (False, extract-stats init-state-wl)
    } else do {
      let T = from-init-state T;
      if get-conflict-wl T ≠ None
      then RETURN (True, extract-stats T)
      else if CS = [] then RETURN (True, Some [])
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜ᵢₙ′));
        ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T + get-subsumed-clauses-wl
T = mset ‘# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        T ← rewatch-st T;
        T ← RETURN (finalise-init T);
        (b, S) ← cdcl-twl-stgy-restart-prog-bounded-wl T;
        RETURN (b, if b ∧ get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
      }
    }
  }› (is ‹?A = ?B›) for CS opts
⟨proof⟩
```

definition *IsaSAT-bounded-heur* :: ‹*opts* ⇒ *nat clause-l list* ⇒ (*bool* × (*bool* × *nat literal list* × *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 ≤ uint32-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 isasat-fast-init T ∧ ¬is-failed-heur-init T
    then do {
      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 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 b ∧ 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)
  }›
```

definition *empty-conflict-code'* :: ‹(*bool* × - *list* × *stats*) *nres*› **where**

```
  ‹empty-conflict-code' = do{
     let M0 = [];
     RETURN (False, M0, (0, 0, 0, 0, 0, 0, 0, ema-fast-init))}›
```

lemma *IsaSAT-bounded-heur-alt-def*:

```
  ‹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 ≤ uint32-max);
    let $\mathcal{A}_{in}'$ = mset-set (extract-atms-clss CS {});
    ASSERT(isasat-input-bounded $\mathcal{A}_{in}'$);
    ASSERT(distinct-mset $\mathcal{A}_{in}'$);
    S ← init-state-wl-heur $\mathcal{A}_{in}'$;
    (T::twl-st-wl-heur-init) ← init-dt-wl-heur False CS S;
    failed ← RETURN ((isasat-fast-init T ∧ ¬is-failed-heur-init T));
    if ¬failed
    then  do {
       RETURN (False, empty-init-code)
    } else do {
      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 do {
         ASSERT($\mathcal{A}_{in}'$ ≠ {#});
         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
≠ 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 b ∧ get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
           else extract-state-stat U)
      }
    }
  }›
  ⟨proof⟩
```

lemma *IsaSAT-heur-bounded-IsaSAT-bounded*:
‹*IsaSAT-bounded-heur b CS* ≤ ⇓(*bool-rel* $\times_f$ *model-stat-rel*) (*IsaSAT-bounded CS*)›
⟨*proof*⟩

lemma *ISASAT-bounded-SAT-l-bounded'*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*› **and**
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-bounded CS* ≤ ⇓ {((*b*, *S*), (*b'*, *S'*)). *b* = *b'* ∧ (*b* ⟶ *S* = *S'*)} (*SAT-l-bounded' CS*)›
⟨*proof*⟩

lemma *IsaSAT-bounded-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-bounded-heur opts CS* ≤ ⇓ {((*b*, *m*), (*b'*, *m'*)). *b*=*b'* ∧ (*b* ⟶ (*m*,*m'*) ∈ *model-stat-rel*)}
(*model-if-satisfiable-bounded* (*mset* ‘# *mset CS*))›
⟨*proof*⟩

lemma *IsaSAT-bounded-heur-model-if-sat'*:
‹(*uncurry IsaSAT-bounded-heur*, *uncurry* (λ-. *model-if-satisfiable-bounded*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*) ∧
(∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*)]$_f$

Id $\times_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨{((*b*, *m*), (*b*′, *m*′)). *b*=*b*′ ∧ (*b* ⟶ (*m*,*m*′) ∈ *model-stat-rel*)}⟩*nres-rel*›
⟨*proof*⟩

end
theory *IsaSAT-LLVM*
imports *Version IsaSAT-CDCL-LLVM*
IsaSAT-Initialisation-LLVM Version IsaSAT
IsaSAT-Restart-LLVM
begin

Chapter 22

Code of Full IsaSAT

abbreviation *model-stat-assn* **where**
⟨*model-stat-assn* ≡ *bool1-assn* $\times_a$ (*arl64-assn unat-lit-assn*) $\times_a$ *stats-assn*⟩

abbreviation *model-stat-assn*$_0$ ::
  bool ×
   nat literal list ×
   64 word ×
   64 word × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *ema*
   ⇒ *1 word* ×
    (*64 word* × *64 word* × *32 word ptr*) ×
    64 word ×
    64 word × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *ema*
    ⇒ *llvm-amemory* ⇒ *bool*
where
⟨*model-stat-assn*$_0$ ≡ *bool1-assn* $\times_a$ (*al-assn unat-lit-assn*) $\times_a$ *stats-assn*⟩

abbreviation *lits-with-max-assn* :: ⟨*nat multiset*
  ⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*⟩ **where**
⟨*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* $\times_a$ *uint32-nat-assn*) *lits-with-max-rel*⟩

abbreviation *lits-with-max-assn*$_0$ :: ⟨*nat multiset*
  ⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*⟩ **where**
⟨*lits-with-max-assn*$_0$ ≡ *hr-comp* (*al-assn atom-assn* $\times_a$ *unat32-assn*) *lits-with-max-rel*⟩

lemma *lits-with-max-assn-alt-def*: ⟨*lits-with-max-assn* = *hr-comp* (*arl64-assn atom-assn* $\times_a$ *uint32-nat-assn*)
    (*lits-with-max-rel O* ⟨*nat-rel*⟩*IsaSAT-Initialisation.mset-rel*)⟩
⟨*proof*⟩

lemma *init-state-wl-D′-code-isasat*: ⟨(*hr-comp isasat-init-assn*
 (*Id* $\times_f$
  (*Id* $\times_f$
   (*Id* $\times_f$
    (*nat-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*⟩
⟨*proof*⟩

definition *model-assn* **where**
⟨*model-assn* = *hr-comp model-stat-assn model-stat-rel*⟩

lemma *extract-model-of-state-stat-alt-def*:

‹RETURN o extract-model-of-state-stat = (λ((M, M′), N′, D′, j, W′, vm, clvls, cach, lbd,
outl, stats,
heur, vdom, avdom, lcount, opts, old-arena).
do {mop-free M′; mop-free N′; mop-free D′; mop-free j; mop-free W′; mop-free vm;
mop-free clvls;
mop-free cach; mop-free lbd; mop-free outl; mop-free heur;
mop-free vdom; mop-free avdom; mop-free opts;
mop-free old-arena;
RETURN (False, M, stats)
})›
⟨proof⟩

schematic-goal mk-free-lookup-clause-rel-assn[sepref-frame-free-rules]: ‹MK-FREE lookup-clause-rel-assn ?fr›
⟨proof⟩

schematic-goal mk-free-trail-pol-fast-assn[sepref-frame-free-rules]: ‹MK-FREE conflict-option-rel-assn ?fr›
⟨proof⟩

schematic-goal mk-free-vmtf-remove-assn[sepref-frame-free-rules]: ‹MK-FREE vmtf-remove-assn ?fr›
⟨proof⟩

schematic-goal mk-free-cach-refinement-l-assn[sepref-frame-free-rules]: ‹MK-FREE cach-refinement-l-assn ?fr›
⟨proof⟩

schematic-goal mk-free-lbd-assn[sepref-frame-free-rules]: ‹MK-FREE lbd-assn ?fr›
⟨proof⟩

schematic-goal mk-free-opts-assn[sepref-frame-free-rules]: ‹MK-FREE opts-assn ?fr›
⟨proof⟩

schematic-goal mk-free-heuristic-assn[sepref-frame-free-rules]: ‹MK-FREE heuristic-assn ?fr›
⟨proof⟩

context
fixes l-dummy :: ‹′l::len2 itself›
fixes ll-dummy :: ‹′ll::len2 itself›
fixes L LL AA
defines [simp]: ‹L ≡ (LENGTH (′l))›
defines [simp]: ‹LL ≡ (LENGTH (′ll))›
defines [simp]: ‹AA ≡ raw-aal-assn TYPE(′l::len2) TYPE(′ll::len2)›
begin
private lemma n-unf: ‹hr-comp AA (⟨⟨the-pure A⟩list-rel⟩list-rel) = aal-assn A› ⟨proof⟩

context
notes [fcomp-norm-unfold] = n-unf
begin

lemma aal-assn-free[sepref-frame-free-rules]: ‹MK-FREE AA aal-free›
⟨proof⟩
sepref-decl-op list-list-free: ‹λ-::- list list. ()› :: ‹⟨⟨A⟩list-rel⟩list-rel → unit-rel› ⟨proof⟩

lemma *hn-aal-free-raw*: ‹(*aal-free*,*RETURN o op-list-list-free*) ∈ $AA^d \rightarrow_a$ *unit-assn*›
  ⟨*proof*⟩

  sepref-decl-impl *aal-free*: *hn-aal-free-raw*
    ⟨*proof*⟩

  lemmas *array-mk-free*[*sepref-frame-free-rules*] = *hn-MK-FREEI*[*OF aal-free-hnr*]
end
end

schematic-goal *mk-free-isasat-init-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE isasat-init-assn ?fr*›
  ⟨*proof*⟩

sepref-def *extract-model-of-state-stat*
  is ‹*RETURN o extract-model-of-state-stat*›
  :: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a$ *model-stat-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *extract-model-of-state-stat.refine*

lemma *extract-state-stat-alt-def*:
  ‹*RETURN o extract-state-stat* = (λ(*M*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
      heur,
      vdom, *avdom*, *lcount*, *opts*, *old-arena*).
    do {*mop-free M*; *mop-free N'*; *mop-free D'*; *mop-free j*; *mop-free W'*; *mop-free vm*;
      mop-free clvls;
      mop-free cach; *mop-free lbd*; *mop-free outl*; *mop-free heur*;
      mop-free vdom; *mop-free avdom*; *mop-free opts*;
      mop-free old-arena;
     RETURN (*True*, [], *stats*)})›
  ⟨*proof*⟩

sepref-def *extract-state-stat*
  is ‹*RETURN o extract-state-stat*›
  :: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a$ *model-stat-assn*›
  ⟨*proof*⟩

lemma *convert-state-hnr*:
  ‹(*uncurry* (*return oo* (λ- *S*. *S*)), *uncurry* (*RETURN oo convert-state*))
  ∈ $ghost\text{-}assn^k *_a (isasat\text{-}init\text{-}assn)^d \rightarrow_a$
    isasat-init-assn›
  ⟨*proof*⟩

sepref-def *IsaSAT-use-fast-mode-impl*
  is ‹*uncurry0* (*RETURN IsaSAT-use-fast-mode*)›
  :: ‹$unit\text{-}assn^k \rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *IsaSAT-use-fast-mode-impl.refine extract-state-stat.refine*

sepref-def *empty-conflict-code'*
  is ‹*uncurry0* (*empty-conflict-code*)›
  :: ‹$unit\text{-}assn^k \rightarrow_a$ *model-stat-assn*›
  ⟨*proof*⟩

declare *empty-conflict-code'.refine*[*sepref-fr-rules*]

sepref-def *empty-init-code'*
  is ‹*uncurry0* (*RETURN empty-init-code*)›
  :: ‹$unit\text{-}assn^k \rightarrow_a$ *model-stat-assn*›
  ⟨*proof*⟩

declare *empty-init-code'.refine*[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-full*

sepref-register *to-init-state from-init-state get-conflict-wl-is-None-init extract-stats init-dt-wl-heur*

definition *isasat-fast-bound* :: ‹*nat*› **where**
‹*isasat-fast-bound* = *sint64-max* − (*uint32-max div 2* + *MAX-HEADER-SIZE*+*1*)›

lemma *isasat-fast-bound-alt-def*: ‹*isasat-fast-bound* = *9223372034707292156*›
  ⟨*proof*⟩

sepref-def *isasat-fast-bound-impl*
  is ‹*uncurry0* (*RETURN isasat-fast-bound*)›
  :: ‹$unit\text{-}assn^k \rightarrow_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *isasat-fast-bound-impl.refine*

lemma *isasat-fast-init-alt-def*:
  ‹*RETURN o isasat-fast-init* = (λ(*M*, *N*, -). *RETURN* (*length N* ≤ *isasat-fast-bound*))›
  ⟨*proof*⟩

sepref-def *isasat-fast-init-code*
  is ‹*RETURN o isasat-fast-init*›
  :: ‹$isasat\text{-}init\text{-}assn^k \rightarrow_a$ *bool1-assn*›
  ⟨*proof*⟩

declare *isasat-fast-init-code.refine*[*sepref-fr-rules*]

declare *convert-state-hnr*[*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*]

thm *init-state-wl-D'-code.refine*[*FCOMP init-state-wl-D'*[*unfolded convert-fref*],
  unfolded lits-with-max-assn-alt-def[*symmetric*]]

lemma [*sepref-fr-rules*]: ‹(*init-state-wl-D'-code*, *init-state-wl-heur-fast*)
∈ [λ*x*. *distinct-mset x* ∧
      (∀ *L*∈#$\mathcal{L}_{all}$ *x*.
          nat-of-lit L
          ≤ *uint32-max*)]$_a$ $lits\text{-}with\text{-}max\text{-}assn^k$ → *isasat-init-assn*›
  ⟨*proof*⟩

lemma *is-failed-heur-init-alt-def*:
‹*is-failed-heur-init* = (λ(-, -, -, -, -, -, -, -, -, -, -, *failed*). *failed*)›
⟨*proof*⟩

sepref-def *is-failed-heur-init-impl*
is ‹*RETURN o is-failed-heur-init*›
:: ‹*isasat-init-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *is-failed-heur-init-impl.refine*

definition *ghost-assn* **where** ‹*ghost-assn* = *hr-comp unit-assn virtual-copy-rel*›

lemma [*sepref-fr-rules*]: ‹(*return o* (λ-. ()), *RETURN o virtual-copy*) ∈ *lits-with-max-assn*^{k} $\rightarrow_a$ *ghost-assn*›
⟨*proof*⟩

sepref-register *virtual-copy empty-conflict-code empty-init-code*
isasat-fast-init is-failed-heur-init
extract-model-of-state-stat extract-state-stat
isasat-information-banner
finalise-init-code
IsaSAT-Initialisation.rewatch-heur-st-fast
get-conflict-wl-is-None-heur
cdcl-twl-stgy-prog-bounded-wl-heur
get-conflict-wl-is-None-heur-init
convert-state

lemma *isasat-information-banner-alt-def*:
‹*isasat-information-banner S* =
RETURN (())›
⟨*proof*⟩

schematic-goal *mk-free-ghost-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE ghost-assn ?fr*›
⟨*proof*⟩

sepref-def *IsaSAT-code*
is ‹*uncurry IsaSAT-bounded-heur*›
:: ‹*opts-assn*^{d} $*_a$ (*clauses-ll-assn*)^k $\rightarrow_a$ *bool1-assn* $\times_a$ *model-stat-assn*›
⟨*proof*⟩

definition *default-opts* **where**
‹*default-opts* = (*True*, *True*, *True*)›

sepref-def *default-opts-impl*
is ‹*uncurry0* (*RETURN default-opts*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *opts-assn*›
⟨*proof*⟩

definition *IsaSAT-bounded-heur-wrapper* :: ‹- ⇒ (*nat*) *nres*›**where**
‹*IsaSAT-bounded-heur-wrapper C* = *do* {
(*b*, (*b'*, -)) ← *IsaSAT-bounded-heur default-opts C*;
RETURN ((*if b then 2 else 0*) + (*if b' then 1 else 0*))
}›

The calling convention of LLVM and clang is not the same, so returning the model is currently unsupported. We return only the flags (as ints, not as bools) and the statistics.

```
sepref-register IsaSAT-bounded-heur default-opts
sepref-def IsaSAT-code-wrapped
  is ‹IsaSAT-bounded-heur-wrapper›
  :: ‹(clauses-ll-assn)^k →a sint64-nat-assn›
  ⟨proof⟩
```

The setup to transmit the version is a bit complicated, because it LLVM does not support direct export of string literals. Therefore, we actually convert the version to an array chars (more precisely, of machine words – ended with 0) that can be read and printed by the C layer. Note the conversion must be automatic, because the version depends on the underlying git repository.

```
function array-of-version where
  ‹array-of-version i str arr =
    (if i ≥ length str then arr
    else array-of-version (i+1) str (arr[i := str ! i]))›
⟨proof⟩
termination
  ⟨proof⟩

sepref-definition llvm-version
  is ‹uncurry0 (RETURN (
        let str = map (nat-of-integer o (of-char :: - ⇒ integer)) (String.explode Version.version) @ [0] in
         array-of-version 0 str (replicate (length str) 0)))›
  :: ‹unit-assn^k →a array-assn sint32-nat-assn›
  ⟨proof⟩

experiment
begin
  lemmas [llvm-code] = llvm-version-def

  lemmas [llvm-inline] =
    unit-propagation-inner-loop-body-wl-fast-heur-code-def
    NORMAL-PHASE-def DEFAULT-INIT-PHASE-def QUIET-PHASE-def
    find-unwatched-wl-st-heur-fast-code-def
    update-clause-wl-fast-code-def

  export-llvm
    IsaSAT-code-wrapped is ‹int64-t IsaSAT-code-wrapped(CLAUSES)›
    llvm-version is ‹STRING-VERSION llvm-version›
    default-opts-impl
    IsaSAT-code
    opts-restart-impl
    count-decided-pol-impl is ‹uint32-t count-decided-st-heur-pol-fast(TRAIL)›
    arena-lit-impl is ‹uint32-t arena-lit-impl(ARENA, int64-t)›
  defines ‹
     typedef struct {int64-t size; struct {int64-t used; uint32-t *clause;};} CLAUSE;
     typedef struct {int64-t num-clauses; CLAUSE *clauses;} CLAUSES;

     typedef struct {int64-t size; struct {int64-t capacity; int32-t *data;};} ARENA;
     typedef int32-t* STRING-VERSION;

     typedef struct {int64-t size; struct {int64-t capacity; uint32-t *data;};} RAW-TRAIL;
     typedef struct {int64-t size; int8-t *polarity;} POLARITY;
```

typedef struct {*int64-t size*; *int32-t* ∗*level*;} *LEVEL*;
typedef struct {*int64-t size*; *int64-t* ∗*reasons*;} *REASONS*;
typedef struct {*int64-t size*; *struct* {*int64-t capacity*; *int32-t* ∗*data*;};} *CONTROL-STACK*;
typedef struct {*RAW-TRAIL raw-trail*;
struct {*POLARITY pol*;
struct {*LEVEL lev*;
struct {*REASONS resasons*;
struct {*int32-t dec-lev*;
CONTROL-STACK cs;};};};};} *TRAIL*;
›
file ‹*code/isasat-restart.ll*›

end

definition *model-bounded-assn* **where**
‹*model-bounded-assn* =
hr-comp (*bool1-assn* $\times_a$ *model-stat-assn*$_0$)
{((*b*, *m*), (*b*′, *m*′)). *b*=*b*′ ∧ (*b* ⟶ (*m*,*m*′) ∈ *model-stat-rel*)}›

definition *clauses-l-assn* **where**
‹*clauses-l-assn* = *hr-comp* (*IICF-Array-of-Array-List.aal-assn unat-lit-assn*)
(*list-mset-rel O* ⟨*list-mset-rel*⟩*IsaSAT-Initialisation.mset-rel*)›

theorem *IsaSAT-full-correctness*:
‹(*uncurry IsaSAT-code*, *uncurry* (λ-. *model-if-satisfiable-bounded*))
∈ [λ(-, *a*). *Multiset.Ball a distinct-mset* ∧
(∀ *C*∈#*a*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*)]$_a$ *opts-assn*^{d} $*_a$ *clauses-l-assn*^{k} → *model-bounded-assn*›
⟨*proof*⟩

end