Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

theory *CDCL-W-Optimal-Model*
imports *CDCL.CDCL-W-Abstract-State HOL−Library.Extended-Nat Weidenbach-Book-Base.Explorer*
begin

0.1 CDCL Extensions

A counter-example for the original version from the book has been found (see below). There is no simple fix, except taking complete models.

Based on Dominik Zimmer's thesis, we later reduced the problem of finding partial models to finding total models. We later switched to the more elegant dual rail encoding (thanks to the reviewer).

0.1.1 Optimisations

notation *image-mset* (**infixr** *'# 90*)

The initial version was supposed to work on partial models directly. I found a counterexample while writing the proof:

Nitpicking 0.1.

Christoph's book draft 0.1. $(M; N; U; k; \top; O) \Rightarrow^{Propagate} (ML^{C \vee L}; N; U; k; \top; O)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, L *is undefined in* M.

$(M; N; U; k; \top; O) \Rightarrow^{Decide} (ML^{k+1}; N; U; k+1; \top; O)$
provided L *is undefined in* M, *contained in* N.

$(M; N; U; k; \top; O) \Rightarrow^{ConflSat} (M; N; U; k; D; O)$
provided $D \in (N \cup U)$ *and* $M \models \neg D$.

$(M; N; U; k; \top; O) \Rightarrow^{ConflOpt} (M; N; U; k; \neg M; O)$
provided $O \neq \epsilon$ *and* $\text{cost}(M) \geq \text{cost}(O)$.

$(ML^{C \vee L}; N; U; k; D; O) \Rightarrow^{Skip} (M; N; U; k; D; O)$
provided $D \notin \{\top, \bot\}$ *and* $\neg L$ *does not occur in* D.

$(ML^{C \vee L}; N; U; k; D \vee -(L); O) \Rightarrow^{Resolve} (M; N; U; k; D \vee C; O)$
provided D *is of level* k.

$(M_1 K^{i+1} M_2; N; U; k; D \vee L; O) \Rightarrow^{Backtrack} (M_1 L^{D \vee L}; N; U \cup \{D \vee L\}; i; \top; O)$
provided L *is of level* k *and* D *is of level* i.

$(M; N; U; k; \top; O) \Rightarrow^{Improve} (M; N; U; k; \top; M)$
provided $M \models N$ *and* $O = \epsilon$ *or* $\text{cost}(M) < \text{cost}(O)$.

This calculus does not always find the model with minimum cost. Take for example the following cost function:

$$\text{cost} : \begin{cases} P \to 3 \\ \neg P \to 1 \\ Q \to 1 \\ \neg Q \to 1 \end{cases}$$

and the clauses $N = \{P \vee Q\}$. *We can then do the following transitions:*
$(\epsilon, N, \varnothing, \top, \infty)$
$\Rightarrow^{Decide} (P^1, N, \varnothing, \top, \infty)$
$\Rightarrow^{Improve} (P^1, N, \varnothing, \top, (P, 3))$
$\Rightarrow^{conflictOpt} (P^1, N, \varnothing, \neg P, (P, 3))$
$\Rightarrow^{backtrack} (\neg P^{\neg P}, N, \{\neg P\}, \top, (P, 3))$
$\Rightarrow^{propagate} (\neg P^{\neg P} Q^{P \vee Q}, N, \{\neg P\}, \top, (P, 3))$
$\Rightarrow^{improve} (\neg P^{\neg P} Q^{P \vee Q}, N, \{\neg P\}, \top, (\neg P\, Q, 2))$
$\Rightarrow^{conflictOpt} (\neg P^{\neg P} Q^{P \vee Q}, N, \{\neg P\}, P \vee \neg Q, (\neg P\, Q, 2))$
$\Rightarrow^{resolve} (\neg P^{\neg P}, N, \{\neg P\}, P, (\neg P\, Q, 2))$
$\Rightarrow^{resolve} (\epsilon, N, \{\neg P\}, \bot, (\neg P\, Q, 3))$
However, the optimal model is Q.

The idea of the proof (explained of the example of the optimising CDCL) is the following:

1. We start with a calculus OCDCL on (M, N, U, D, Op).

2. This extended to a state (M, N + *all-models-of-higher-cost*, U, D, Op).

3. Each transition step of OCDCL is mapped to a step in CDCL over the abstract state. The abstract set of clauses might be unsatisfiable, but we only use it to prove the invariants on the state. Only adding clause cannot be mapped to a transition over the abstract state, but adding clauses does not break the invariants (as long as the additional clauses do not contain duplicate literals).

4. The last proofs are done over CDCLopt.

We abstract about how the optimisation is done in the locale below: We define a calculus *cdcl-bnb* (for branch-and-bounds). It is parametrised by how the conflicting clauses are generated and the improvement criterion.

We later instantiate it with the optimisation calculus from Weidenbach's book.

Helper libraries

lemma (**in** −) *Neg-atm-of-itself-uminus-iff*: ‹*Neg* (*atm-of xa*) ≠ − *xa* ⟷ *is-neg xa*›
⟨*proof*⟩

lemma (**in** −) *Pos-atm-of-itself-uminus-iff*: ‹*Pos* (*atm-of xa*) ≠ − *xa* ⟷ *is-pos xa*›
⟨*proof*⟩

definition *model-on* :: ‹'*v partial-interp* ⇒ '*v clauses* ⇒ *bool*› **where**
‹*model-on I N* ⟷ *consistent-interp I* ∧ *atm-of* ‘ *I* ⊆ *atms-of-mm N*›

CDCL BNB

locale *conflict-driven-clause-learning-with-adding-init-clause-cost*$_W$*-no-state* =
state$_W$*-no-state*
state-eq state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
for
state-eq :: '*st* ⇒ '*st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: '*st* ⇒ ('*v*, '*v clause*) *ann-lits* × '*v clauses* × '*v clauses* × '*v clause option* ×
'*a* × '*b* **and**
trail :: '*st* ⇒ ('*v*, '*v clause*) *ann-lits* **and**
init-clss :: '*st* ⇒ '*v clauses* **and**
learned-clss :: '*st* ⇒ '*v clauses* **and**
conflicting :: '*st* ⇒ '*v clause option* **and**

cons-trail :: ('*v*, '*v clause*) *ann-lit* ⇒ '*st* ⇒ '*st* **and**
tl-trail :: '*st* ⇒ '*st* **and**
add-learned-cls :: '*v clause* ⇒ '*st* ⇒ '*st* **and**
remove-cls :: '*v clause* ⇒ '*st* ⇒ '*st* **and**
update-conflicting :: '*v clause option* ⇒ '*st* ⇒ '*st* **and**

init-state :: *'v clauses* ⇒ *'st* +
fixes
update-weight-information :: (*'v*, *'v clause*) *ann-lits* ⇒ *'st* ⇒ *'st* **and**
is-improving-int :: (*'v*, *'v clause*) *ann-lits* ⇒ (*'v*, *'v clause*) *ann-lits* ⇒ *'v clauses* ⇒ *'a* ⇒ *bool* **and**
conflicting-clauses :: *'v clauses* ⇒ *'a* ⇒ *'v clauses* **and**
weight :: ‹*'st* ⇒ *'a*›
begin

abbreviation *is-improving* **where**
‹*is-improving M M' S* ≡ *is-improving-int M M' (init-clss S) (weight S)*›

definition *additional-info'* :: *'st* ⇒ *'b* **where**
additional-info' S = (λ(-, -, -, -, -, *D*). *D*) (*state S*)

definition *conflicting-clss* :: ‹*'st* ⇒ *'v literal multiset multiset*› **where**
‹*conflicting-clss S* = *conflicting-clauses* (*init-clss S*) (*weight S*)›

definition *abs-state*
:: *'st* ⇒ (*'v*, *'v clause*) *ann-lit list* × *'v clauses* × *'v clauses* × *'v clause option*
where
‹*abs-state S* = (*trail S*, *init-clss S* + *conflicting-clss S*, *learned-clss S*,
conflicting S)›

end

locale *conflict-driven-clause-learning-with-adding-init-clause-cost$_W$-ops* =
conflict-driven-clause-learning-with-adding-init-clause-cost$_W$-no-state
state-eq state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
— Adding a clause:
update-weight-information is-improving-int conflicting-clauses weight
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'a × *'b* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

cons-trail :: (*'v*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
update-weight-information :: (*'v*, *'v clause*) *ann-lits* ⇒ *'st* ⇒ *'st* **and**

is-improving-int :: ('*v*, '*v clause*) *ann-lits* ⇒ ('*v*, '*v clause*) *ann-lits* ⇒ '*v clauses* ⇒ '*a* ⇒ *bool* **and**
conflicting-clauses :: '*v clauses* ⇒ '*a* ⇒ '*v clauses* **and**
weight :: ‹'*st* ⇒ '*a*› +
assumes
state-prop':
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *weight S*, *additional-info' S*)›
and
update-weight-information:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w'*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w'*, *other*)› **and**
atms-of-conflicting-clss:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)› **and**
distinct-mset-mset-conflicting-clss:
‹*distinct-mset-mset* (*conflicting-clss S*)› **and**
conflicting-clss-update-weight-information-mono:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ⟹ *is-improving M M' S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M' S*)›
and
conflicting-clss-update-weight-information-in:
‹*is-improving M M' S* ⟹ *negate-ann-lits M'* ∈# *conflicting-clss* (*update-weight-information M' S*)›
begin

sublocale *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

declare *reduce-trail-to-skip-beginning*[*simp*]

lemma *state-eq-weight*[*state-simp*, *simp*]: ‹*S* ∼ *T* ⟹ *weight S* = *weight T*›
⟨*proof*⟩

lemma *conflicting-clause-state-eq*[*state-simp*, *simp*]:
‹*S* ∼ *T* ⟹ *conflicting-clss S* = *conflicting-clss T*›
⟨*proof*⟩

lemma
weight-cons-trail[*simp*]:
‹*weight* (*cons-trail L S*) = *weight S*› **and**
weight-update-conflicting[*simp*]:
‹*weight* (*update-conflicting C S*) = *weight S*› **and**
weight-tl-trail[*simp*]:
‹*weight* (*tl-trail S*) = *weight S*› **and**

weight-add-learned-cls[*simp*]:
‹*weight* (*add-learned-cls D S*) = *weight S*›
⟨*proof*⟩

lemma *update-weight-information-simp*[*simp*]:
‹*trail* (*update-weight-information C S*) = *trail S*›
‹*init-clss* (*update-weight-information C S*) = *init-clss S*›
‹*learned-clss* (*update-weight-information C S*) = *learned-clss S*›
‹*clauses* (*update-weight-information C S*) = *clauses S*›
‹*backtrack-lvl* (*update-weight-information C S*) = *backtrack-lvl S*›
‹*conflicting* (*update-weight-information C S*) = *conflicting S*›
⟨*proof*⟩

lemma
conflicting-clss-cons-trail[*simp*]: ‹*conflicting-clss* (*cons-trail K S*) = *conflicting-clss S*› **and**
conflicting-clss-tl-trail[*simp*]: ‹*conflicting-clss* (*tl-trail S*) = *conflicting-clss S*› **and**
conflicting-clss-add-learned-cls[*simp*]:
‹*conflicting-clss* (*add-learned-cls D S*) = *conflicting-clss S*› **and**
conflicting-clss-update-conflicting[*simp*]:
‹*conflicting-clss* (*update-conflicting E S*) = *conflicting-clss S*›
⟨*proof*⟩

inductive *conflict-opt* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S T* :: *'st* **where**
conflict-opt-rule:
‹*conflict-opt S T*›
if
‹*negate-ann-lits* (*trail S*) ∈# *conflicting-clss S*›
‹*conflicting S* = *None*›
‹*T* ∼ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›

inductive-cases *conflict-optE*: ‹*conflict-opt S T*›

inductive *improvep* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
improve-rule:
‹*improvep S T*›
if
‹*is-improving* (*trail S*) *M' S*› **and**
‹*conflicting S* = *None*› **and**
‹*T* ∼ *update-weight-information M' S*›

inductive-cases *improveE*: ‹*improvep S T*›

lemma *invs-update-weight-information*[*simp*]:
‹*no-strange-atm* (*update-weight-information C S*) = (*no-strange-atm S*)›
‹*cdcl*$_W$*-M-level-inv* (*update-weight-information C S*) = *cdcl*$_W$*-M-level-inv S*›
‹*distinct-cdcl*$_W$*-state* (*update-weight-information C S*) = *distinct-cdcl*$_W$*-state S*›
‹*cdcl*$_W$*-conflicting* (*update-weight-information C S*) = *cdcl*$_W$*-conflicting S*›
‹*cdcl*$_W$*-learned-clause* (*update-weight-information C S*) = *cdcl*$_W$*-learned-clause S*›
⟨*proof*⟩

lemma *conflict-opt-cdcl*$_W$*-all-struct-inv*:
assumes ‹*conflict-opt S T*› **and**
inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state T*)›
⟨*proof*⟩

lemma *reduce-trail-to-update-weight-information*[*simp*]:
‹*trail* (*reduce-trail-to M* (*update-weight-information M′ S*)) = *trail* (*reduce-trail-to M S*)›
⟨*proof*⟩

lemma *additional-info-weight-additional-info′*: ‹*additional-info S* = (*weight S*, *additional-info′ S*)›
⟨*proof*⟩

lemma
weight-reduce-trail-to[*simp*]: ‹*weight* (*reduce-trail-to M S*) = *weight S*› **and**
additional-info′-reduce-trail-to[*simp*]: ‹*additional-info′* (*reduce-trail-to M S*) = *additional-info′ S*›
⟨*proof*⟩

lemma *conflicting-clss-reduce-trail-to*[*simp*]: ‹*conflicting-clss* (*reduce-trail-to M S*) = *conflicting-clss S*›
⟨*proof*⟩

lemma *improve-cdcl$_W$-all-struct-inv*:
assumes ‹*improvep S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state T*)›
⟨*proof*⟩

cdcl$_W$-restart-mset.cdcl$_W$-stgy-invariant is too restrictive: *cdcl$_W$-restart-mset.no-smaller-confl* is needed but does not hold(at least, if cannot ensure that conflicts are found as soon as possible).

lemma *improve-no-smaller-conflict*:
assumes ‹*improvep S T*› **and**
‹*no-smaller-confl S*›
shows ‹*no-smaller-confl T*› **and** ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

lemma *conflict-opt-no-smaller-conflict*:
assumes ‹*conflict-opt S T*› **and**
‹*no-smaller-confl S*›
shows ‹*no-smaller-confl T*› **and** ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

fun *no-confl-prop-impr* **where**
‹*no-confl-prop-impr S* ⟷
no-step propagate S ∧ *no-step conflict S*›

We use a slighlty generalised form of backtrack to make conflict clause minimisation possible.

inductive *obacktrack* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
obacktrack-rule: ‹
conflicting S = *Some* (*add-mset L D*) ⟹
(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) ⟹
get-maximum-level (*trail S*) *D′* ≡ *i* ⟹
get-level (*trail S*) *K* = *i* + *1* ⟹
D′ ⊆# *D* ⟹
clauses S + *conflicting-clss S* ⊨*pm add-mset L D′* ⟹
T ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) ⟹
obacktrack S T›

inductive-cases *obacktrackE*: ‹*obacktrack S T*›

inductive *cdcl-bnb-bj* :: *'st* ⇒ *'st* ⇒ *bool* **where**
skip: *skip S S'* ⟹ *cdcl-bnb-bj S S'* |
resolve: *resolve S S'* ⟹ *cdcl-bnb-bj S S'* |
backtrack: *obacktrack S S'* ⟹ *cdcl-bnb-bj S S'*

inductive-cases *cdcl-bnb-bjE*: *cdcl-bnb-bj S T*

inductive $ocdcl_W$*-o* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
decide: *decide S S'* ⟹ $ocdcl_W$*-o S S'* |
bj: *cdcl-bnb-bj S S'* ⟹ $ocdcl_W$*-o S S'*

inductive *cdcl-bnb* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
cdcl-conflict: *conflict S S'* ⟹ *cdcl-bnb S S'* |
cdcl-propagate: *propagate S S'* ⟹ *cdcl-bnb S S'* |
cdcl-improve: *improvep S S'* ⟹ *cdcl-bnb S S'* |
cdcl-conflict-opt: *conflict-opt S S'* ⟹ *cdcl-bnb S S'* |
cdcl-other': $ocdcl_W$*-o S S'* ⟹ *cdcl-bnb S S'*

inductive *cdcl-bnb-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
cdcl-bnb-conflict: *conflict S S'* ⟹ *cdcl-bnb-stgy S S'* |
cdcl-bnb-propagate: *propagate S S'* ⟹ *cdcl-bnb-stgy S S'* |
cdcl-bnb-improve: *improvep S S'* ⟹ *cdcl-bnb-stgy S S'* |
cdcl-bnb-conflict-opt: *conflict-opt S S'* ⟹ *cdcl-bnb-stgy S S'* |
cdcl-bnb-other': $ocdcl_W$*-o S S'* ⟹ *no-confl-prop-impr S* ⟹ *cdcl-bnb-stgy S S'*

lemma $ocdcl_W$*-o-induct*[*consumes 1*, *case-names decide skip resolve backtrack*]:
fixes *S* :: *'st*
assumes $cdcl_W$*-restart*: $ocdcl_W$*-o S T* **and**
decideH: ⋀*L T*. *conflicting S* = *None* ⟹ *undefined-lit* (*trail S*) *L* ⟹
atm-of L ∈ *atms-of-mm* (*init-clss S*) ⟹
T ∼ *cons-trail* (*Decided L*) *S* ⟹
P S T **and**
skipH: ⋀*L C' M E T*.
trail S = *Propagated L C'* # *M* ⟹
conflicting S = *Some E* ⟹
−*L* ∉# *E* ⟹ *E* ≠ {#} ⟹
T ∼ *tl-trail S* ⟹
P S T **and**
resolveH: ⋀*L E M D T*.
trail S = *Propagated L E* # *M* ⟹
L ∈# *E* ⟹
hd-trail S = *Propagated L E* ⟹
conflicting S = *Some D* ⟹
−*L* ∈# *D* ⟹
get-maximum-level (*trail S*) ((*remove1-mset* (−*L*) *D*)) = *backtrack-lvl S* ⟹
T ∼ *update-conflicting*
(*Some* (*resolve-cls L D E*)) (*tl-trail S*) ⟹
P S T **and**
backtrackH: ⋀*L D K i M1 M2 T D'*.
conflicting S = *Some* (*add-mset L D*) ⟹
(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) ⟹

get-maximum-level (*trail S*) *D′* ≡ *i* ⟹
get-level (*trail S*) *K* = *i*+*1* ⟹
D′ ⊆# *D* ⟹
clauses S + *conflicting-clss S* ⊨*pm add-mset L D′* ⟹
T ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) ⟹
P S T
shows *P S T*
⟨*proof*⟩

lemma *obacktrack-backtrackg*: ‹*obacktrack S T* ⟹ *backtrackg S T*›
⟨*proof*⟩

Pluging into normal CDCL

lemma *cdcl-bnb-no-more-init-clss*:
‹*cdcl-bnb S S′* ⟹ *init-clss S* = *init-clss S′*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-no-more-init-clss*:
‹*cdcl-bnb*** *S S′* ⟹ *init-clss S* = *init-clss S′*›
⟨*proof*⟩

lemma *conflict-opt-conflict*:
‹*conflict-opt S T* ⟹ *cdcl*$_W$*-restart-mset.conflict* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *conflict-conflict*:
‹*conflict S T* ⟹ *cdcl*$_W$*-restart-mset.conflict* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *propagate-propagate*:
‹*propagate S T* ⟹ *cdcl*$_W$*-restart-mset.propagate* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *decide-decide*:
‹*decide S T* ⟹ *cdcl*$_W$*-restart-mset.decide* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *skip-skip*:
‹*skip S T* ⟹ *cdcl*$_W$*-restart-mset.skip* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *resolve-resolve*:
‹*resolve S T* ⟹ *cdcl*$_W$*-restart-mset.resolve* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *backtrack-backtrack*:
‹*obacktrack S T* ⟹ *cdcl*$_W$*-restart-mset.backtrack* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *ocdcl*$_W$*-o-all-rules-induct*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *′st*

assumes
$ocdcl_W\text{-}o\ S\ T$ **and**
$\bigwedge T.\ decide\ S\ T \Longrightarrow P\ S\ T$ **and**
$\bigwedge T.\ obacktrack\ S\ T \Longrightarrow P\ S\ T$ **and**
$\bigwedge T.\ skip\ S\ T \Longrightarrow P\ S\ T$ **and**
$\bigwedge T.\ resolve\ S\ T \Longrightarrow P\ S\ T$
shows $P\ S\ T$
⟨*proof*⟩

lemma $cdcl_W\text{-}o\text{-}cdcl_W\text{-}o$:
$\langle ocdcl_W\text{-}o\ S\ S' \Longrightarrow cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}o\ (abs\text{-}state\ S)\ (abs\text{-}state\ S')\rangle$
⟨*proof*⟩

lemma *cdcl-bnb-stgy-all-struct-inv*:
assumes ⟨*cdcl-bnb S T*⟩ **and** ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)$⟩
shows ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ T)$⟩
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-all-struct-inv*:
assumes ⟨$cdcl\text{-}bnb^{**}\ S\ T$⟩ **and** ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)$⟩
shows ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ T)$⟩
⟨*proof*⟩

definition *cdcl-bnb-struct-invs* :: ⟨$'st \Rightarrow bool$⟩ **where**
⟨*cdcl-bnb-struct-invs S* $\longleftrightarrow$
$atms\text{-}of\text{-}mm\ (conflicting\text{-}clss\ S) \subseteq atms\text{-}of\text{-}mm\ (init\text{-}clss\ S)$⟩

lemma *cdcl-bnb-cdcl-bnb-struct-invs*:
⟨*cdcl-bnb S T* $\Longrightarrow$ *cdcl-bnb-struct-invs S* $\Longrightarrow$ *cdcl-bnb-struct-invs T*⟩
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-cdcl-bnb-struct-invs*:
⟨$cdcl\text{-}bnb^{**}\ S\ T \Longrightarrow cdcl\text{-}bnb\text{-}struct\text{-}invs\ S \Longrightarrow cdcl\text{-}bnb\text{-}struct\text{-}invs\ T$⟩
⟨*proof*⟩

lemma *cdcl-bnb-stgy-cdcl-bnb*: ⟨*cdcl-bnb-stgy S T* $\Longrightarrow$ *cdcl-bnb S T*⟩
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-cdcl-bnb*: ⟨$cdcl\text{-}bnb\text{-}stgy^{**}\ S\ T \Longrightarrow cdcl\text{-}bnb^{**}\ S\ T$⟩
⟨*proof*⟩

The following does *not* hold, because we cannot guarantee the absence of conflict of smaller level after *improve* and *conflict-opt*.

lemma *cdcl-bnb-all-stgy-inv*:
assumes ⟨*cdcl-bnb S T*⟩ **and** ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)$⟩ **and**
⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy\text{-}invariant\ (abs\text{-}state\ S)$⟩
shows ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy\text{-}invariant\ (abs\text{-}state\ T)$⟩
⟨*proof*⟩

lemma *skip-conflict-is-false-with-level*:
assumes ⟨*skip S T*⟩ **and**
struct-inv: ⟨$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)$⟩ **and**
confl-inv:⟨*conflict-is-false-with-level S*⟩
shows ⟨*conflict-is-false-with-level T*⟩
⟨*proof*⟩

lemma *propagate-conflict-is-false-with-level*:
assumes ‹*propagate S T*› **and**
struct-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
confl-inv:‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

lemma *cdcl*$_W$*-o-conflict-is-false-with-level*:
assumes ‹*cdcl*$_W$*-o S T*› **and**
struct-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
confl-inv: ‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

lemma *cdcl*$_W$*-o-no-smaller-confl*:
assumes ‹*cdcl*$_W$*-o S T*› **and**
struct-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
confl-inv: ‹*no-smaller-confl S*› **and**
lev: ‹*conflict-is-false-with-level S*› **and**
n-s: ‹*no-confl-prop-impr S*›
shows ‹*no-smaller-confl T*›
⟨*proof*⟩

declare *cdcl*$_W$*-restart-mset.conflict-is-false-with-level-def* [*simp del*]

lemma *improve-conflict-is-false-with-level*:
assumes ‹*improvep S T*› **and** ‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

declare *conflict-is-false-with-level-def*[*simp del*]

lemma *trail-trail* [*simp*]:
‹*CDCL-W-Abstract-State.trail* (*abs-state S*) = *trail S*›
⟨*proof*⟩

lemma [*simp*]:
‹*CDCL-W-Abstract-State.trail* (*cdcl*$_W$*-restart-mset.reduce-trail-to M* (*abs-state S*)) =
trail (*reduce-trail-to M S*)›
⟨*proof*⟩

lemma [*simp*]:
‹*CDCL-W-Abstract-State.trail* (*cdcl*$_W$*-restart-mset.reduce-trail-to M* (*abs-state S*)) =
trail (*reduce-trail-to M S*)›
⟨*proof*⟩

lemma *cdcl*$_W$*-M-level-inv-cdcl*$_W$*-M-level-inv*[*iff*]:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*abs-state S*) = *cdcl*$_W$*-M-level-inv S*›
⟨*proof*⟩

lemma *obacktrack-state-eq-compatible*:
assumes
bt: *obacktrack S T* **and**
SS': $S \sim S'$ **and**
TT': $T \sim T'$
shows *obacktrack S' T'*

⟨*proof*⟩

lemma *ocdcl$_W$-o-no-smaller-confl-inv*:
fixes $S\ S' :: 'st$
assumes
ocdcl$_W$-o $S\ S'$ **and**
n-s: *no-step conflict* S **and**
lev: *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state* S) **and**
max-lev: *conflict-is-false-with-level* S **and**
smaller: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *cdcl-bnb-stgy-no-smaller-confl*:
assumes ‹*cdcl-bnb-stgy* $S\ T$› **and**
‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state* S)› **and**
‹*no-smaller-confl* S› **and**
‹*conflict-is-false-with-level* S›
shows ‹*no-smaller-confl* T›
⟨*proof*⟩

lemma *ocdcl$_W$-o-conflict-is-false-with-level-inv*:
assumes
ocdcl$_W$-o $S\ S'$ **and**
lev: *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state* S) **and**
confl-inv: *conflict-is-false-with-level* S
shows *conflict-is-false-with-level* S'
⟨*proof*⟩

lemma *cdcl-bnb-stgy-conflict-is-false-with-level*:
assumes ‹*cdcl-bnb-stgy* $S\ T$› **and**
‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state* S)› **and**
‹*no-smaller-confl* S› **and**
‹*conflict-is-false-with-level* S›
shows ‹*conflict-is-false-with-level* T›
⟨*proof*⟩

lemma *decided-cons-eq-append-decide-cons*: ‹*Decided* L # $MM = M'$ @ *Decided* K # $M \longleftrightarrow$
$(M' \neq [] \wedge hd\ M' = Decided\ L \wedge MM = tl\ M'$ @ *Decided* K # $M) \vee$
$(M' = [] \wedge L = K \wedge MM = M)$›
⟨*proof*⟩

lemma *either-all-false-or-earliest-decomposition*:
shows ‹$(\forall\, K\ K'.\ L = K'$ @ $K \longrightarrow \neg P\ K) \vee$
$(\exists\, L'\ L''.\ L = L''$ @ $L' \wedge P\ L' \wedge (\forall\, K\ K'.\ L' = K'$ @ $K \longrightarrow K' \neq [] \longrightarrow \neg P\ K))$›
⟨*proof*⟩

lemma *trail-is-improving-Ex-improve*:
assumes *confl*: ‹*conflicting* S = *None*› **and**
imp: ‹*is-improving* (*trail* S) $M'\ S$›
shows ‹*Ex* (*improvep* S)›
⟨*proof*⟩

definition *cdcl-bnb-stgy-inv* :: $'st \Rightarrow$ *bool* **where**
‹*cdcl-bnb-stgy-inv* $S \longleftrightarrow$ *conflict-is-false-with-level* $S \wedge$ *no-smaller-confl* S›

lemma *cdcl-bnb-stgy-invD*:
shows ‹*cdcl-bnb-stgy-inv S* ⟷ *cdcl$_W$-stgy-invariant S*›
⟨*proof*⟩

lemma *cdcl-bnb-stgy-stgy-inv*:
‹*cdcl-bnb-stgy S T* ⟹ *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) ⟹
cdcl-bnb-stgy-inv S ⟹ *cdcl-bnb-stgy-inv T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-stgy-inv*:
‹*cdcl-bnb-stgy*** *S T* ⟹ *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) ⟹
cdcl-bnb-stgy-inv S ⟹ *cdcl-bnb-stgy-inv T*›
⟨*proof*⟩

lemma *learned-clss-learned-clss*[*simp*]:
‹*CDCL-W-Abstract-State.learned-clss* (*abs-state S*) = *learned-clss S*›
⟨*proof*⟩

lemma *state-eq-init-clss-abs-state*[*state-simp*, *simp*]:
‹*S* ∼ *T* ⟹ *CDCL-W-Abstract-State.init-clss* (*abs-state S*) = *CDCL-W-Abstract-State.init-clss* (*abs-state T*)›
⟨*proof*⟩

lemma
init-clss-abs-state-update-conflicting[*simp*]:
‹*CDCL-W-Abstract-State.init-clss* (*abs-state* (*update-conflicting* (*Some D*) *S*)) =
CDCL-W-Abstract-State.init-clss (*abs-state S*)› **and**
init-clss-abs-state-cons-trail[*simp*]:
‹*CDCL-W-Abstract-State.init-clss* (*abs-state* (*cons-trail K S*)) =
CDCL-W-Abstract-State.init-clss (*abs-state S*)›
⟨*proof*⟩

lemma *cdcl-bnb-cdcl$_W$-learned-clauses-entailed-by-init*:
assumes
‹*cdcl-bnb S T*› **and**
entailed: ‹*cdcl$_W$-restart-mset.cdcl$_W$-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl$_W$-restart-mset.cdcl$_W$-learned-clauses-entailed-by-init* (*abs-state T*)›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-cdcl$_W$-learned-clauses-entailed-by-init*:
assumes
‹*cdcl-bnb*** *S T*› **and**
entailed: ‹*cdcl$_W$-restart-mset.cdcl$_W$-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl$_W$-restart-mset.cdcl$_W$-learned-clauses-entailed-by-init* (*abs-state T*)›
⟨*proof*⟩

lemma *atms-of-init-clss-conflicting-clss2*[*simp*]:
‹*atms-of-mm* (*init-clss S*) ∪ *atms-of-mm* (*conflicting-clss S*) = *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

lemma *no-strange-atm-no-strange-atm*[*simp*]:
‹*cdcl$_W$-restart-mset.no-strange-atm* (*abs-state S*) = *no-strange-atm S*›
⟨*proof*⟩

lemma *cdcl$_W$-conflicting-cdcl$_W$-conflicting*[*simp*]:
‹*cdcl$_W$-restart-mset.cdcl$_W$-conflicting* (*abs-state S*) = *cdcl$_W$-conflicting S*›
⟨*proof*⟩

lemma *distinct-cdcl$_W$-state-distinct-cdcl$_W$-state*:
‹*cdcl$_W$-restart-mset.distinct-cdcl$_W$-state* (*abs-state S*) ⟹ *distinct-cdcl$_W$-state S*›
⟨*proof*⟩

lemma *conflicting-abs-state-conflicting*[*simp*]:
‹*CDCL-W-Abstract-State.conflicting* (*abs-state S*) = *conflicting S*› **and**
clauses-abs-state[*simp*]:
‹*cdcl$_W$-restart-mset.clauses* (*abs-state S*) = *clauses S* + *conflicting-clss S*› **and**
abs-state-tl-trail[*simp*]:
‹*abs-state* (*tl-trail S*) = *CDCL-W-Abstract-State.tl-trail* (*abs-state S*)› **and**
abs-state-add-learned-cls[*simp*]:
‹*abs-state* (*add-learned-cls C S*) = *CDCL-W-Abstract-State.add-learned-cls C* (*abs-state S*)› **and**
abs-state-update-conflicting[*simp*]:
‹*abs-state* (*update-conflicting D S*) = *CDCL-W-Abstract-State.update-conflicting D* (*abs-state S*)›
⟨*proof*⟩

lemma *sim-abs-state-simp*: ‹*S* ∼ *T* ⟹ *abs-state S* = *abs-state T*›
⟨*proof*⟩

lemma *abs-state-cons-trail*[*simp*]:
‹*abs-state* (*cons-trail K S*) = *CDCL-W-Abstract-State.cons-trail K* (*abs-state S*)› **and**
abs-state-reduce-trail-to[*simp*]:
‹*abs-state* (*reduce-trail-to M S*) = *cdcl$_W$-restart-mset.reduce-trail-to M* (*abs-state S*)›
⟨*proof*⟩

lemma *obacktrack-imp-backtrack*:
‹*obacktrack S T* ⟹ *cdcl$_W$-restart-mset.backtrack* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *backtrack-imp-obacktrack*:
‹*cdcl$_W$-restart-mset.backtrack* (*abs-state S*) *T* ⟹ *Ex* (*obacktrack S*)›
⟨*proof*⟩

lemma *cdcl$_W$-same-weight*: ‹*cdcl$_W$ S U* ⟹ *weight S* = *weight U*›
⟨*proof*⟩

lemma *ocdcl$_W$-o-same-weight*: ‹*ocdcl$_W$-o S U* ⟹ *weight S* = *weight U*›
⟨*proof*⟩

This is a proof artefact: it is easier to reason on *improvep* when the set of initial clauses is fixed (here by N). The next theorem shows that the conclusion is equivalent to not fixing the set of clauses.

lemma *wf-cdcl-bnb*:
assumes *improve*: ‹⋀*S T*. *improvep S T* ⟹ *init-clss S* = *N* ⟹ (*ν* (*weight T*), *ν* (*weight S*)) ∈ *R*›
and
wf-R: ‹*wf R*›
shows ‹*wf* {(*T*, *S*). *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T* ∧
init-clss S = *N*}›
(**is** ‹*wf ?A*›)
⟨*proof*⟩

corollary *wf-cdcl-bnb-fixed-iff*:
shows ‹(∀ *N*. *wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*
∧ *init-clss S* = *N*}) ⟷
wf {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*}›
(**is** ‹(∀ *N*. *wf* (*?A N*)) ⟷ *wf ?B*›)
⟨*proof*⟩

The following is a slightly more restricted version of the theorem, because it makes it possible to add some specific invariant, which can be useful when the proof of the decreasing is complicated.

lemma *wf-cdcl-bnb-with-additional-inv*:
assumes *improve*: ‹⋀*S T*. *improvep S T* ⟹ *P S* ⟹ *init-clss S* = *N* ⟹ (*ν* (*weight T*), *ν* (*weight S*)) ∈ *R*› **and**
wf-R: ‹*wf R*› **and**
‹⋀*S T*. *cdcl-bnb S T* ⟹ *P S* ⟹ *init-clss S* = *N* ⟹ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ⟹ *P T*›
shows ‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T* ∧ *P S* ∧
init-clss S = *N*}›
(**is** ‹*wf ?A*›)
⟨*proof*⟩

lemma *conflict-is-false-with-level-abs-iff*:
‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*abs-state S*) ⟷
conflict-is-false-with-level S›
⟨*proof*⟩

lemma *decide-abs-state-decide*:
‹*cdcl*$_W$*-restart-mset.decide* (*abs-state S*) *T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *Ex*(*decide S*)›
⟨*proof*⟩

lemma *cdcl-bnb-no-conflicting-clss-cdcl*$_W$:
assumes ‹*cdcl-bnb S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$ (*abs-state S*) (*abs-state T*) ∧ *conflicting-clss S* = {#}›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-no-conflicting-clss-cdcl*$_W$:
assumes ‹*cdcl-bnb*** *S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹*cdcl*$_W$*-restart-mset.cdcl*$_W{}^{**}$ (*abs-state S*) (*abs-state T*) ∧ *conflicting-clss S* = {#}›
⟨*proof*⟩

lemma *conflict-abs-ex-conflict-no-conflicting*:
assumes ‹*cdcl*$_W$*-restart-mset.conflict* (*abs-state S*) *T*› **and** ‹*conflicting-clss S* = {#}›
shows ‹∃ *T*. *conflict S T*›
⟨*proof*⟩

lemma *propagate-abs-ex-propagate-no-conflicting*:
assumes ‹*cdcl*$_W$*-restart-mset.propagate* (*abs-state S*) *T*› **and** ‹*conflicting-clss S* = {#}›
shows ‹∃ *T*. *propagate S T*›
⟨*proof*⟩

lemma *cdcl-bnb-stgy-no-conflicting-clss-cdcl*$_W$*-stgy*:
assumes ‹*cdcl-bnb-stgy S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy* (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-no-conflicting-clss-cdcl$_W$-stgy*:
assumes ‹*cdcl-bnb-stgy*** *S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹*cdcl$_W$-restart-mset.cdcl$_W$-stgy*** (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

context
assumes *can-always-improve*:
‹⋀*S*. *trail S* ⊨*asm clauses S* ⟹ *no-step conflict-opt S* ⟹
conflicting S = *None* ⟹
cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv (*abs-state S*) ⟹
total-over-m (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*)) ⟹ *Ex* (*improvep S*)›
begin

The following theorems states a non-obvious (and slightly subtle) property: The fact that there is no conflicting cannot be shown without additional assumption. However, the assumption that every model leads to an improvements implies that we end up with a conflict.

lemma *no-step-cdcl-bnb-cdcl$_W$*:
assumes
ns: ‹*no-step cdcl-bnb S*› **and**
struct-invs: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl$_W$-restart-mset.cdcl$_W$* (*abs-state S*)›
⟨*proof*⟩

lemma *no-step-cdcl-bnb-stgy*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *None* ∨ *conflicting S* = *Some* {#}›
⟨*proof*⟩

lemma *no-step-cdcl-bnb-stgy-empty-conflict*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *Some* {#}›
⟨*proof*⟩

lemma *full-cdcl-bnb-stgy-no-conflicting-clss-unsat*:
assumes
full: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*› **and**
ent-init: ‹*cdcl$_W$-restart-mset.cdcl$_W$-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
[*simp*]: ‹*conflicting-clss T* = {#}›
shows ‹*unsatisfiable* (*set-mset* (*init-clss S*))›
⟨*proof*⟩

lemma *ocdcl$_W$-o-no-smaller-propa*:
assumes ‹*ocdcl$_W$-o S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**

smaller-propa: ‹*no-smaller-propa S*› **and**
n-s: ‹*no-confl-prop-impr S*›
shows ‹*no-smaller-propa T*›
⟨*proof*⟩

lemma *ocdcl$_W$-no-smaller-propa*:
assumes ‹*cdcl-bnb-stgy S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
n-s: ‹*no-confl-prop-impr S*›
shows ‹*no-smaller-propa T*›
⟨*proof*⟩

Unfortunately, we cannot reuse the proof we have alrealy done.

lemma *ocdcl$_W$-no-relearning*:
assumes ‹*cdcl-bnb-stgy S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
n-s: ‹*no-confl-prop-impr S*› **and**
dist: ‹*distinct-mset* (*clauses S*)›
shows ‹*distinct-mset* (*clauses T*)›
⟨*proof*⟩

lemma *full-cdcl-bnb-stgy-unsat*:
assumes
st: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
⟨*proof*⟩

end

lemma *cdcl-bnb-reasons-in-clauses*:
‹*cdcl-bnb S T* ⟹ *reasons-in-clauses S* ⟹ *reasons-in-clauses T*›
⟨*proof*⟩

end

OCDCL

This locales includes only the assumption we make on the weight function.

locale *ocdcl-weight* =
fixes
ϱ :: ‹'*v clause* ⇒ '*a* :: {*linorder*}›
assumes
ϱ-mono: ‹*distinct-mset B* ⟹ *A* ⊆# *B* ⟹ *ϱ A* ≤ *ϱ B*›
begin

lemma *ϱ-empty-simp*[*simp*]:
assumes ‹*consistent-interp* (*set-mset A*)› ‹*distinct-mset A*›
shows ‹*ϱ A* ≥ *ϱ* {#}› ‹¬*ϱ A* < *ϱ* {#}› ‹*ϱ A* ≤ *ϱ* {#} ⟷ *ϱ A* = *ϱ* {#}›

⟨*proof*⟩

end

This is one of the version of the weight functions used by Christoph Weidenbach.

locale *ocdcl-weight-WB* =
  fixes
    ν :: ⟨*'v literal* ⇒ *nat*⟩
begin

definition ϱ :: ⟨*'v clause* ⇒ *nat*⟩ **where**
  ⟨ϱ *M* = ($\sum$ *A* ∈# *M*. ν *A*)⟩

sublocale *ocdcl-weight* ϱ
  ⟨*proof*⟩

end

The following datatype is equivalent to *'a option*. Howover, it has the opposite ordering. Therefore, I decided to use a different type instead of have a second order which conflicts with `~~/src/HOL/Library/Option_ord.thy`.

datatype *'a optimal-model* = *Not-Found* | *is-found*: *Found* (*the-optimal*: *'a*)

instantiation *optimal-model* :: (*ord*) *ord*
begin
  fun *less-optimal-model* :: ⟨*'a* :: *ord optimal-model* ⇒ *'a optimal-model* ⇒ *bool*⟩ **where**
  ⟨*less-optimal-model Not-Found - = False*⟩
| ⟨*less-optimal-model* (*Found* -) *Not-Found* ⟷ *True*⟩
| ⟨*less-optimal-model* (*Found a*) (*Found b*) ⟷ *a* < *b*⟩

fun *less-eq-optimal-model* :: ⟨*'a* :: *ord optimal-model* ⇒ *'a optimal-model* ⇒ *bool*⟩ **where**
  ⟨*less-eq-optimal-model Not-Found Not-Found* = *True*⟩
| ⟨*less-eq-optimal-model Not-Found* (*Found* -) = *False*⟩
| ⟨*less-eq-optimal-model* (*Found* -) *Not-Found* ⟷ *True*⟩
| ⟨*less-eq-optimal-model* (*Found a*) (*Found b*) ⟷ *a* ≤ *b*⟩

instance
  ⟨*proof*⟩

end

instance *optimal-model* :: (*preorder*) *preorder*
  ⟨*proof*⟩

instance *optimal-model* :: (*order*) *order*
  ⟨*proof*⟩

instance *optimal-model* :: (*linorder*) *linorder*
  ⟨*proof*⟩

instantiation *optimal-model* :: (*wellorder*) *wellorder*
begin

lemma *wf-less-optimal-model*: *wf* {(*M* :: *'a optimal-model*, *N*). *M* < *N*}
⟨*proof*⟩

instance ⟨*proof*⟩

end

locale *conflict-driven-clause-learning*$_W$*-optimal-weight* =
conflict-driven-clause-learning$_W$
state-eq
state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting
— get state:
init-state +
ocdcl-weight ϱ
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'v clause option × *'b* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

cons-trail :: (*'v*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**
init-state :: *'v clauses* ⇒ *'st* **and**
ϱ :: ‹*'v clause* ⇒ *'a* :: {*linorder*}› +
fixes
update-additional-info :: ‹*'v clause option* × *'b* ⇒ *'st* ⇒ *'st*›
assumes
update-additional-info:
‹*state S* = (*M*, *N*, *U*, *C*, *K*) ⟹ *state* (*update-additional-info K' S*) = (*M*, *N*, *U*, *C*, *K'*)› **and**
weight-init-state:
‹⋀*N* :: *'v clauses*. *fst* (*additional-info* (*init-state N*)) = *None*›
begin

definition *update-weight-information* :: ‹(*'v*, *'v clause*) *ann-lits* ⇒ *'st* ⇒ *'st*› **where**
‹*update-weight-information M S* =
update-additional-info (*Some* (*lit-of* ‘# *mset M*), *snd* (*additional-info S*)) *S*›

lemma
trail-update-additional-info[*simp*]: ‹*trail* (*update-additional-info w S*) = *trail S*› **and**
init-clss-update-additional-info[*simp*]:
‹*init-clss* (*update-additional-info w S*) = *init-clss S*› **and**
learned-clss-update-additional-info[*simp*]:
‹*learned-clss* (*update-additional-info w S*) = *learned-clss S*› **and**
backtrack-lvl-update-additional-info[*simp*]:
‹*backtrack-lvl* (*update-additional-info w S*) = *backtrack-lvl S*› **and**
conflicting-update-additional-info[*simp*]:

‹*conflicting* (*update-additional-info w S*) = *conflicting S*› **and**
clauses-update-additional-info[*simp*]:
‹*clauses* (*update-additional-info w S*) = *clauses S*›
⟨*proof*⟩

lemma
trail-update-weight-information[*simp*]:
‹*trail* (*update-weight-information w S*) = *trail S*› **and**
init-clss-update-weight-information[*simp*]:
‹*init-clss* (*update-weight-information w S*) = *init-clss S*› **and**
learned-clss-update-weight-information[*simp*]:
‹*learned-clss* (*update-weight-information w S*) = *learned-clss S*› **and**
backtrack-lvl-update-weight-information[*simp*]:
‹*backtrack-lvl* (*update-weight-information w S*) = *backtrack-lvl S*› **and**
conflicting-update-weight-information[*simp*]:
‹*conflicting* (*update-weight-information w S*) = *conflicting S*› **and**
clauses-update-weight-information[*simp*]:
‹*clauses* (*update-weight-information w S*) = *clauses S*›
⟨*proof*⟩

definition *weight* **where**
‹*weight S* = *fst* (*additional-info S*)›

lemma
additional-info-update-additional-info[*simp*]:
additional-info (*update-additional-info w S*) = *w*
⟨*proof*⟩

lemma
weight-cons-trail2[*simp*]: ‹*weight* (*cons-trail L S*) = *weight S*› **and**
clss-tl-trail2[*simp*]: *weight* (*tl-trail S*) = *weight S* **and**
weight-add-learned-cls-unfolded:
weight (*add-learned-cls U S*) = *weight S*
and
weight-update-conflicting2[*simp*]: *weight* (*update-conflicting D S*) = *weight S* **and**
weight-remove-cls2[*simp*]:
weight (*remove-cls C S*) = *weight S* **and**
weight-add-learned-cls2[*simp*]:
weight (*add-learned-cls C S*) = *weight S* **and**
weight-update-weight-information2[*simp*]:
weight (*update-weight-information M S*) = *Some* (*lit-of* ‘# *mset M*)
⟨*proof*⟩

abbreviation *ϱ′* :: ‹*'v clause option* ⇒ *'a optimal-model*› **where**
‹*ϱ′ w* ≡ (*case w of None* ⇒ *Not-Found* | *Some w* ⇒ *Found* (*ϱ w*))›

definition *is-improving-int*
:: (*'v*, *'v clause*) *ann-lits* ⇒ (*'v*, *'v clause*) *ann-lits* ⇒ *'v clauses* ⇒
'v clause option ⇒ *bool*
where
‹*is-improving-int M M′ N w* ⟷ *Found* (*ϱ* (*lit-of* ‘# *mset M′*)) < *ϱ′ w* ∧
M′ ⊨*asm N* ∧ *no-dup M′* ∧
lit-of ‘# *mset M′* ∈ *simple-clss* (*atms-of-mm N*) ∧
total-over-m (*lits-of-l M′*) (*set-mset N*) ∧
(∀ *M′*. *total-over-m* (*lits-of-l M′*) (*set-mset N*) ⟶ *mset M* ⊆# *mset M′* ⟶
lit-of ‘# *mset M′* ∈ *simple-clss* (*atms-of-mm N*) ⟶

ϱ (*lit-of* ‘# *mset M'*) = ϱ (*lit-of* ‘# *mset M*))›

definition *too-heavy-clauses*
:: ‹'*v clauses* ⇒ '*v clause option* ⇒ '*v clauses*›
where
‹*too-heavy-clauses M w* =
{#*pNeg C* | *C* ∈# *mset-set* (*simple-clss* (*atms-of-mm M*)). ϱ' *w* ≤ *Found* (ϱ *C*)#}›

definition *conflicting-clauses*
:: ‹'*v clauses* ⇒ '*v clause option* ⇒ '*v clauses*›
where
‹*conflicting-clauses N w* =
{#*C* ∈# *mset-set* (*simple-clss* (*atms-of-mm N*)). *too-heavy-clauses N w* ⊨*pm C*#}›

lemma *too-heavy-clauses-conflicting-clauses*:
‹*C* ∈# *too-heavy-clauses M w* ⟹ *C* ∈# *conflicting-clauses M w*›
⟨*proof*⟩

lemma *too-heavy-clauses-contains-itself*:
‹*M* ∈ *simple-clss* (*atms-of-mm N*) ⟹ *pNeg M* ∈# *too-heavy-clauses N* (*Some M*)›
⟨*proof*⟩

lemma *too-heavy-clause-None*[*simp*]: ‹*too-heavy-clauses M None* = {#}›
⟨*proof*⟩

lemma *atms-of-mm-too-heavy-clauses-le*:
‹*atms-of-mm* (*too-heavy-clauses M I*) ⊆ *atms-of-mm M*›
⟨*proof*⟩

lemma
atms-too-heavy-clauses-None:
‹*atms-of-mm* (*too-heavy-clauses M None*) = {}› **and**
atms-too-heavy-clauses-Some:
‹*atms-of w* ⊆ *atms-of-mm M* ⟹ *distinct-mset w* ⟹ ¬*tautology w* ⟹
atms-of-mm (*too-heavy-clauses M* (*Some w*)) = *atms-of-mm M*›
⟨*proof*⟩

lemma *entails-too-heavy-clauses-too-heavy-clauses*:
assumes
‹*consistent-interp I*› **and**
tot: ‹*total-over-m I* (*set-mset* (*too-heavy-clauses M w*))› **and**
‹*I* ⊨*m too-heavy-clauses M w*› **and**
w: ‹*w* ≠ *None* ⟹ *atms-of* (*the w*) ⊆ *atms-of-mm M*›
‹*w* ≠ *None* ⟹ ¬*tautology* (*the w*)›
‹*w* ≠ *None* ⟹ *distinct-mset* (*the w*)›
shows ‹*I* ⊨*m conflicting-clauses M w*›
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$
where
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-cost*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *weight* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
⟨*proof*⟩

lemma *state-additional-info′*:
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *weight S*, *additional-info′ S*)›
⟨*proof*⟩

lemma *state-update-weight-information*:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w′*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w′*, *other*)›
⟨*proof*⟩

lemma *conflicting-clss-incl-init-clss*:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

lemma *distinct-mset-mset-conflicting-clss2*: ‹*distinct-mset-mset* (*conflicting-clss S*)›
⟨*proof*⟩

lemma *too-heavy-clauses-mono*:
‹*ϱ a* > *ϱ* (*lit-of* ‘# *mset M*) ⟹ *too-heavy-clauses N* (*Some a*) ⊆#
too-heavy-clauses N (*Some* (*lit-of* ‘# *mset M*))›
⟨*proof*⟩

lemma *is-improving-conflicting-clss-update-weight-information*: ‹*is-improving M M′ S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M′ S*)›
⟨*proof*⟩

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*is-improving M M′ S*›
shows ‹*negate-ann-lits M′* ∈# *conflicting-clss* (*update-weight-information M′ S*)›
⟨*proof*⟩

lemma *atms-of-init-clss-conflicting-clss*[*simp*]:
‹*atms-of-mm* (*init-clss S*) ∪ *atms-of-mm* (*conflicting-clss S*) = *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

lemma *lit-of-trail-in-simple-clss*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ⟹
lit-of ‘# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
⟨*proof*⟩

lemma *pNeg-lit-of-trail-in-simple-clss*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ⟹
pNeg (*lit-of* ‘# *mset* (*trail S*)) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
⟨*proof*⟩

lemma *conflict-clss-update-weight-no-alien*:
‹*atms-of-mm* (*conflicting-clss* (*update-weight-information M S*))
⊆ *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

sublocale *state*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *state*$_W$*-no-state*
where
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$
where
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**

conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-cost*$_W$*-ops*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *weight* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
⟨*proof*⟩

lemma *wf-cdcl-bnb-fixed*:
‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*
∧ *init-clss S* = *N*}›
⟨*proof*⟩

lemma *wf-cdcl-bnb2*:
‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)
∧ *cdcl-bnb S T*}›
⟨*proof*⟩

lemma *not-entailed-too-heavy-clauses-ge*:
‹*C* ∈ *simple-clss* (*atms-of-mm N*) ⟹ ¬ *too-heavy-clauses N w* ⊨*pm pNeg C* ⟹ ¬*Found* (*ϱ C*) ≥ *ϱ*′ *w*›
⟨*proof*⟩

lemma *pNeg-simple-clss-iff*[*simp*]:
‹*pNeg C* ∈ *simple-clss N* ⟷ *C* ∈ *simple-clss N*›
⟨*proof*⟩

lemma *can-always-improve*:
assumes
ent: ‹*trail S* ⊨*asm clauses S*› **and**
total: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))› **and**
n-s: ‹*no-step conflict-opt S*› **and**
confl: ‹*conflicting S* = *None*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*Ex* (*improvep S*)›
⟨*proof*⟩

lemma *no-step-cdcl-bnb-stgy-empty-conflict2*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *Some* {#}›
⟨*proof*⟩

lemma *cdcl-bnb-larger-still-larger*:
assumes
‹*cdcl-bnb S T*›
shows ‹*ϱ'* (*weight S*) ≥ *ϱ'* (*weight T*)›
⟨*proof*⟩

lemma *obacktrack-model-still-model*:
assumes
‹*obacktrack S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹*set-mset I* ⊨*sm clauses S*› ‹*set-mset I* ⊨*sm conflicting-clss S*› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
le: ‹*Found* (*ϱ I*) < *ϱ'* (*weight T*)›
shows
‹*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm conflicting-clss T*›
⟨*proof*⟩

lemma *entails-too-heavy-clauses-if-le*:
assumes
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm N*› **and**
le: ‹*Found* (*ϱ I*) < *ϱ'* (*Some M'*)›
shows
‹*set-mset I* ⊨*m too-heavy-clauses N* (*Some M'*)›
⟨*proof*⟩

lemma *entails-conflicting-clauses-if-le*:
fixes *M''*
defines ‹*M'* ≡ *lit-of* ‘# *mset M''*›
assumes
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm N*› **and**
le: ‹*Found* (*ϱ I*) < *ϱ'* (*Some M'*)› **and**
‹*is-improving-int M M'' N w*›
shows
‹*set-mset I* ⊨*m conflicting-clauses N* (*weight* (*update-weight-information M'' S*))›
⟨*proof*⟩

lemma *improve-model-still-model*:

assumes
‹*improvep S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹*set-mset I* ⊨*sm clauses S*› ‹*set-mset I* ⊨*sm conflicting-clss S*› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
le: ‹*Found* (*ϱ I*) < *ϱ*′ (*weight T*)›
shows
‹*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm conflicting-clss T*›
⟨*proof*⟩

lemma *cdcl-bnb-still-model*:
assumes
‹*cdcl-bnb S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹*set-mset I* ⊨*sm clauses S*› ‹*set-mset I* ⊨*sm conflicting-clss S*› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*›
shows
‹(*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm conflicting-clss T*) ∨ *Found* (*ϱ I*) ≥ *ϱ*′ (*weight T*)›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-still-model*:
assumes
st: ‹*cdcl-bnb*** *S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹(*set-mset I* ⊨*sm clauses S* ∧ *set-mset I* ⊨*sm conflicting-clss S*) ∨ *Found* (*ϱ I*) ≥ *ϱ*′ (*weight S*)› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*›
shows
‹(*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm conflicting-clss T*) ∨ *Found* (*ϱ I*) ≥ *ϱ*′ (*weight T*)›
⟨*proof*⟩

lemma *full-cdcl-bnb-stgy-larger-or-equal-weight*:
assumes
st: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹(*set-mset I* ⊨*sm clauses S* ∧ *set-mset I* ⊨*sm conflicting-clss S*) ∨ *Found* (*ϱ I*) ≥ *ϱ*′ (*weight S*)› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*Found* (*ϱ I*) ≥ *ϱ*′ (*weight T*)› **and**
‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
⟨*proof*⟩

lemma *full-cdcl-bnb-stgy-unsat2*:
  assumes
    st: ‹*full cdcl-bnb-stgy S T*› **and**
    all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
    opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
    stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
  shows
    ‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
⟨*proof*⟩

lemma *weight-init-state2*[*simp*]: ‹*weight* (*init-state S*) = *None*› **and**
  conflicting-clss-init-state[*simp*]:
    ‹*conflicting-clss* (*init-state N*) = {#}›
  ⟨*proof*⟩

First part of Theorem 2.15.6 of Weidenbach's book

lemma *full-cdcl-bnb-stgy-no-conflicting-clause-unsat*:
  assumes
    st: ‹*full cdcl-bnb-stgy S T*› **and**
    all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
    opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
    stgy-inv: ‹*cdcl-bnb-stgy-inv S*› **and**
    [*simp*]: ‹*weight T* = *None*› **and**
    ent: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
  shows ‹*unsatisfiable* (*set-mset* (*init-clss S*))›
⟨*proof*⟩

definition *annotation-is-model* **where**
  ‹*annotation-is-model S* ⟷
    (*weight S* ≠ *None* ⟶ (*set-mset* (*the* (*weight S*)) ⊨*sm init-clss S* ∧
      consistent-interp (*set-mset* (*the* (*weight S*))) ∧
      atms-of (*the* (*weight S*)) ⊆ *atms-of-mm* (*init-clss S*) ∧
      total-over-m (*set-mset* (*the* (*weight S*))) (*set-mset* (*init-clss S*)) ∧
      distinct-mset (*the* (*weight S*))))›

lemma *cdcl-bnb-annotation-is-model*:
  assumes
    ‹*cdcl-bnb S T*› **and**
    ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
    ‹*annotation-is-model S*›
  shows ‹*annotation-is-model T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-annotation-is-model*:
  ‹*cdcl-bnb*** *S T* ⟹ *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) ⟹
    annotation-is-model S ⟹ *annotation-is-model T*›
  ⟨*proof*⟩

Theorem 2.15.6 of Weidenbach's book

theorem *full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*:
  assumes
    st: ‹*full cdcl-bnb-stgy* (*init-state N*) *T*› **and**
    dist: ‹*distinct-mset-mset N*›
  shows
    ‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› **and**

⟨*weight T* ≠ *None* ⟹ *consistent-interp* (*set-mset* (*the* (*weight T*))) ∧
atms-of (*the* (*weight T*)) ⊆ *atms-of-mm N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N* ∧
total-over-m (*set-mset* (*the* (*weight T*))) (*set-mset N*) ∧
distinct-mset (*the* (*weight T*))⟩ **and**
⟨*distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹ *atms-of I* = *atms-of-mm N* ⟹
set-mset I ⊨*sm N* ⟹ *Found* (*ϱ I*) ≥ *ϱ′* (*weight T*)⟩
⟨*proof*⟩

lemma *pruned-clause-in-conflicting-clss*:
assumes
ge: ⟨⋀*M′*. *total-over-m* (*set-mset* (*mset* (*M* @ *M′*))) (*set-mset* (*init-clss S*)) ⟹
distinct-mset (*atm-of* ‘# *mset* (*M* @ *M′*)) ⟹
consistent-interp (*set-mset* (*mset* (*M* @ *M′*))) ⟹
Found (*ϱ* (*mset* (*M* @ *M′*))) ≥ *ϱ′* (*weight S*)⟩ **and**
atm: ⟨*atms-of* (*mset M*) ⊆ *atms-of-mm* (*init-clss S*)⟩ **and**
dist: ⟨*distinct M*⟩ **and**
cons: ⟨*consistent-interp* (*set M*)⟩
shows ⟨*pNeg* (*mset M*) ∈# *conflicting-clss S*⟩
⟨*proof*⟩

Alternative versions

Calculus with simple Improve rule

To make sure that the paper version of the correct, we restrict the previous calculus to exactly the rules that are on paper.

inductive *pruning* :: ⟨*′st* ⇒ *′st* ⇒ *bool*⟩ **where**
pruning-rule:
⟨*pruning S T*⟩
if
⟨⋀*M′*. *total-over-m* (*set-mset* (*mset* (*map lit-of* (*trail S*) @ *M′*))) (*set-mset* (*init-clss S*)) ⟹
distinct-mset (*atm-of* ‘# *mset* (*map lit-of* (*trail S*) @ *M′*)) ⟹
consistent-interp (*set-mset* (*mset* (*map lit-of* (*trail S*) @ *M′*))) ⟹
ϱ′ (*weight S*) ≤ *Found* (*ϱ* (*mset* (*map lit-of* (*trail S*) @ *M′*)))⟩
⟨*conflicting S* = *None*⟩
⟨*T* ∼ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*⟩

inductive *oconflict-opt* :: *′st* ⇒ *′st* ⇒ *bool* **for** *S T* :: *′st* **where**
oconflict-opt-rule:
⟨*oconflict-opt S T*⟩
if
⟨*Found* (*ϱ* (*lit-of* ‘# *mset* (*trail S*))) ≥ *ϱ′* (*weight S*)⟩
⟨*conflicting S* = *None*⟩
⟨*T* ∼ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*⟩

inductive *improve* :: *′st* ⇒ *′st* ⇒ *bool* **for** *S T* :: *′st* **where**
improve-rule:
⟨*improve S T*⟩
if
⟨*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*init-clss S*))⟩
⟨*Found* (*ϱ* (*lit-of* ‘# *mset* (*trail S*))) < *ϱ′* (*weight S*)⟩
⟨*trail S* ⊨*asm init-clss S*⟩
⟨*conflicting S* = *None*⟩
⟨*T* ∼ *update-weight-information* (*trail S*) *S*⟩

This is the basic version of the calculus:

inductive $ocdcl_w :: \langle 'st \Rightarrow 'st \Rightarrow bool\rangle$ **for** $S :: 'st$ **where**
ocdcl-conflict: $conflict\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$ |
ocdcl-propagate: $propagate\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$ |
ocdcl-improve: $improve\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$ |
ocdcl-conflict-opt: $oconflict\text{-}opt\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$ |
ocdcl-other': $ocdcl_W\text{-}o\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$ |
ocdcl-pruning: $pruning\ S\ S' \Longrightarrow ocdcl_w\ S\ S'$

inductive $ocdcl_w\text{-}stgy :: \langle 'st \Rightarrow 'st \Rightarrow bool\rangle$ **for** $S :: 'st$ **where**
$ocdcl_w$*-conflict*: $conflict\ S\ S' \Longrightarrow ocdcl_w\text{-}stgy\ S\ S'$ |
$ocdcl_w$*-propagate*: $propagate\ S\ S' \Longrightarrow ocdcl_w\text{-}stgy\ S\ S'$ |
$ocdcl_w$*-improve*: $improve\ S\ S' \Longrightarrow ocdcl_w\text{-}stgy\ S\ S'$ |
$ocdcl_w$*-conflict-opt*: $conflict\text{-}opt\ S\ S' \Longrightarrow ocdcl_w\text{-}stgy\ S\ S'$ |
$ocdcl_w$*-other'*: $ocdcl_W\text{-}o\ S\ S' \Longrightarrow no\text{-}confl\text{-}prop\text{-}impr\ S \Longrightarrow ocdcl_w\text{-}stgy\ S\ S'$

lemma *pruning-conflict-opt*:
assumes *ocdcl-pruning*: $\langle pruning\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle conflict\text{-}opt\ S\ T\rangle$
⟨*proof*⟩

lemma *ocdcl-conflict-opt-conflict-opt*:
assumes *ocdcl-pruning*: $\langle oconflict\text{-}opt\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle conflict\text{-}opt\ S\ T\rangle$
⟨*proof*⟩

lemma *improve-improvep*:
assumes *imp*: $\langle improve\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle improvep\ S\ T\rangle$
⟨*proof*⟩

lemma $ocdcl_w$*-cdcl-bnb*:
assumes $\langle ocdcl_w\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle cdcl\text{-}bnb\ S\ T\rangle$
⟨*proof*⟩

lemma $ocdcl_w$*-stgy-cdcl-bnb-stgy*:
assumes $\langle ocdcl_w\text{-}stgy\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle cdcl\text{-}bnb\text{-}stgy\ S\ T\rangle$
⟨*proof*⟩

lemma *rtranclp-ocdcl*$_w$*-stgy-rtranclp-cdcl-bnb-stgy*:
assumes $\langle ocdcl_w\text{-}stgy^{**}\ S\ T\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$
shows $\langle cdcl\text{-}bnb\text{-}stgy^{**}\ S\ T\rangle$
⟨*proof*⟩

lemma *no-step-ocdcl*$_w$*-no-step-cdcl-bnb*:
assumes $\langle no\text{-}step\ ocdcl_w\ S\rangle$ **and**
inv: $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)\rangle$

shows ‹*no-step cdcl-bnb S*›
⟨*proof*⟩

lemma *all-struct-init-state-distinct-iff*:
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state* (*init-state N*)) ⟷
distinct-mset-mset N›
⟨*proof*⟩

lemma *no-step-*$ocdcl_w$*-stgy-no-step-cdcl-bnb-stgy*:
assumes ‹*no-step* $ocdcl_w$*-stgy S*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb-stgy S*›
⟨*proof*⟩

lemma *full-*$ocdcl_w$*-stgy-full-cdcl-bnb-stgy*:
assumes ‹*full* $ocdcl_w$*-stgy S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*full cdcl-bnb-stgy S T*›
⟨*proof*⟩

corollary *full-*$ocdcl_w$*-stgy-no-conflicting-clause-from-init-state*:
assumes
st: ‹*full* $ocdcl_w$*-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* ≠ *None* ⟹ *model-on* (*set-mset* (*the* (*weight T*))) *N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N*
∧
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹ *atms-of I* = *atms-of-mm N* ⟹
set-mset I ⊨*sm N* ⟹ *Found* (*ϱ I*) ≥ *ϱ'* (*weight T*)›
⟨*proof*⟩

lemma *wf-*$ocdcl_w$:
‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)
∧ $ocdcl_w$ *S T*}›
⟨*proof*⟩

Calculus with generalised Improve rule

Now a version with the more general improve rule:

inductive $ocdcl_w$*-p* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
ocdcl-conflict: *conflict S S'* ⟹ $ocdcl_w$*-p S S'* |
ocdcl-propagate: *propagate S S'* ⟹ $ocdcl_w$*-p S S'* |
ocdcl-improve: *improvep S S'* ⟹ $ocdcl_w$*-p S S'* |
ocdcl-conflict-opt: *oconflict-opt S S'* ⟹ $ocdcl_w$*-p S S'* |
ocdcl-other': $ocdcl_W$*-o S S'* ⟹ $ocdcl_w$*-p S S'* |
ocdcl-pruning: *pruning S S'* ⟹ $ocdcl_w$*-p S S'*

inductive $ocdcl_w$*-p-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
$ocdcl_w$*-p-conflict*: *conflict S S'* ⟹ $ocdcl_w$*-p-stgy S S'* |
$ocdcl_w$*-p-propagate*: *propagate S S'* ⟹ $ocdcl_w$*-p-stgy S S'* |
$ocdcl_w$*-p-improve*: *improvep S S'* ⟹ $ocdcl_w$*-p-stgy S S'* |
$ocdcl_w$*-p-conflict-opt*: *conflict-opt S S'* ⟹ $ocdcl_w$*-p-stgy S S'*|

$ocdcl_w$-p-pruning: *pruning S S′ ⟹ ocdcl$_w$-p-stgy S S′ |*
$ocdcl_w$-p-other′: *ocdcl$_W$-o S S′ ⟹ no-confl-prop-impr S ⟹ ocdcl$_w$-p-stgy S S′*

lemma *ocdcl$_w$-p-cdcl-bnb*:
assumes ‹*ocdcl$_w$-p S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb S T*›
⟨*proof*⟩

lemma *ocdcl$_w$-p-stgy-cdcl-bnb-stgy*:
assumes ‹*ocdcl$_w$-p-stgy S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb-stgy S T*›
⟨*proof*⟩

lemma *rtranclp-ocdcl$_w$-p-stgy-rtranclp-cdcl-bnb-stgy*:
assumes ‹*ocdcl$_w$-p-stgy** S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb-stgy** S T*›
⟨*proof*⟩

lemma *no-step-ocdcl$_w$-p-no-step-cdcl-bnb*:
assumes ‹*no-step ocdcl$_w$-p S*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb S*›
⟨*proof*⟩

lemma *no-step-ocdcl$_w$-p-stgy-no-step-cdcl-bnb-stgy*:
assumes ‹*no-step ocdcl$_w$-p-stgy S*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb-stgy S*›
⟨*proof*⟩

lemma *full-ocdcl$_w$-p-stgy-full-cdcl-bnb-stgy*:
assumes ‹*full ocdcl$_w$-p-stgy S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*full cdcl-bnb-stgy S T*›
⟨*proof*⟩

corollary *full-ocdcl$_w$-p-stgy-no-conflicting-clause-from-init-state*:
assumes
st: ‹*full ocdcl$_w$-p-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*weight T = None ⟹ unsatisfiable* (*set-mset N*)› **and**
‹*weight T ≠ None ⟹ model-on* (*set-mset* (*the* (*weight T*))) *N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N*
∧
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I ⟹ consistent-interp* (*set-mset I*) ⟹ *atms-of I = atms-of-mm N ⟹*
set-mset I ⊨*sm N ⟹ Found* (*ϱ I*) ≥ *ϱ′* (*weight T*)›
⟨*proof*⟩

lemma *cdcl-bnb-stgy-no-smaller-propa*:
‹*cdcl-bnb-stgy S T ⟹ cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) ⟹

no-smaller-propa S $\Longrightarrow$ *no-smaller-propa T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-no-smaller-propa*:
‹*cdcl-bnb-stgy*** *S T* $\Longrightarrow$ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) $\Longrightarrow$
no-smaller-propa S $\Longrightarrow$ *no-smaller-propa T*›
⟨*proof*⟩

lemma *wf-ocdcl*$_w$*-p*:
‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)
∧ *ocdcl*$_w$*-p S T*}›
⟨*proof*⟩

end

end
theory *CDCL-W-Partial-Encoding*
imports *CDCL-W-Optimal-Model*
begin

0.1.2 Encoding of partial SAT into total SAT

As a way to make sure we don't reuse theorems names:

interpretation *test*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. *0*› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)›
⟨*proof*⟩

We here formalise the encoding from a formula to another formula from which we will use to derive the optimal partial model.

While the proofs are still inspired by Dominic Zimmer's upcoming bachelor thesis, we now use the dual rail encoding, which is more elegant that the solution found by Christoph to solve the problem.

The intended meaning is the following:

- Σ is the set of all variables
- $\Delta\Sigma$ is the set of all variables with a (possibly non-zero) weight: These are the variable that needs to be replaced during encoding, but it does not matter if the weight 0.

locale *optimal-encoding-opt* = *conflict-driven-clause-learning*$_W$*-optimal-weight*
state-eq

state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
ϱ
update-additional-info
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'v clause option × *'b* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

cons-trail :: (*'v*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
ϱ :: ‹*'v clause* ⇒ *'a* :: {*linorder*}› **and**
update-additional-info :: ‹*'v clause option* × *'b* ⇒ *'st* ⇒ *'st*› +
fixes Σ ΔΣ :: ‹*'v set*› **and**
new-vars :: ‹*'v* ⇒ *'v* × *'v*›
begin

abbreviation *replacement-pos* :: ‹*'v* ⇒ *'v*› ((-)$^{\mapsto 1}$ *100*) **where**
‹*replacement-pos A* ≡ *fst* (*new-vars A*)›

abbreviation *replacement-neg* :: ‹*'v* ⇒ *'v*› ((-)$^{\mapsto 0}$ *100*) **where**
‹*replacement-neg A* ≡ *snd* (*new-vars A*)›

fun *encode-lit* **where**
‹*encode-lit* (*Pos A*) = (*if A* ∈ ΔΣ *then Pos* (*replacement-pos A*) *else Pos A*)› |
‹*encode-lit* (*Neg A*) = (*if A* ∈ ΔΣ *then Pos* (*replacement-neg A*) *else Neg A*)›

lemma *encode-lit-alt-def*:
‹*encode-lit A* = (*if atm-of A* ∈ ΔΣ
then Pos (*if is-pos A then replacement-pos* (*atm-of A*) *else replacement-neg* (*atm-of A*))
else A)›
⟨*proof*⟩

definition *encode-clause* :: ‹*'v clause* ⇒ *'v clause*› **where**
‹*encode-clause C* = *encode-lit* ‘# *C*›

lemma *encode-clause-simp*[*simp*]:
‹*encode-clause* {#} = {#}›

⟨*encode-clause* (*add-mset A C*) = *add-mset* (*encode-lit A*) (*encode-clause C*)⟩
⟨*encode-clause* (*C* + *D*) = *encode-clause C* + *encode-clause D*⟩
⟨*proof*⟩

definition *encode-clauses* :: ⟨'*v clauses* ⇒ '*v clauses*⟩ **where**
⟨*encode-clauses C* = *encode-clause* ‘# *C*⟩

lemma *encode-clauses-simp*[*simp*]:
⟨*encode-clauses* {#} = {#}⟩
⟨*encode-clauses* (*add-mset A C*) = *add-mset* (*encode-clause A*) (*encode-clauses C*)⟩
⟨*encode-clauses* (*C* + *D*) = *encode-clauses C* + *encode-clauses D*⟩
⟨*proof*⟩

definition *additional-constraint* :: ⟨'*v* ⇒ '*v clauses*⟩ **where**
⟨*additional-constraint A* =
{#{#*Neg* ($A^{\mapsto 1}$), *Neg* ($A^{\mapsto 0}$)#}#}⟩

definition *additional-constraints* :: ⟨'*v clauses*⟩ **where**
⟨*additional-constraints* = $\bigcup$#(*additional-constraint* ‘# (*mset-set* ΔΣ))⟩

definition *penc* :: ⟨'*v clauses* ⇒ '*v clauses*⟩ **where**
⟨*penc N* = *encode-clauses N* + *additional-constraints*⟩

lemma *size-encode-clauses*[*simp*]: ⟨*size* (*encode-clauses N*) = *size N*⟩
⟨*proof*⟩

lemma *size-penc*:
⟨*size* (*penc N*) = *size N* + *card* ΔΣ⟩
⟨*proof*⟩

lemma *atms-of-mm-additional-constraints*: ⟨*finite* ΔΣ ⟹
atms-of-mm additional-constraints = *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ⟩
⟨*proof*⟩

lemma *atms-of-mm-encode-clause-subset*:
⟨*atms-of-mm* (*encode-clauses N*) ⊆ (*atms-of-mm N* − ΔΣ) ∪ *replacement-pos* ‘ {*A* ∈ ΔΣ. *A* ∈ *atms-of-mm N*}
∪ *replacement-neg* ‘ {*A* ∈ ΔΣ. *A* ∈ *atms-of-mm N*}⟩
⟨*proof*⟩

In every meaningful application of the theorem below, we have ΔΣ ⊆ *atms-of-mm N*.

lemma *atms-of-mm-penc-subset*: ⟨*finite* ΔΣ ⟹
atms-of-mm (*penc N*) ⊆ *atms-of-mm N* ∪ *replacement-pos* ‘ ΔΣ
∪ *replacement-neg* ‘ ΔΣ ∪ ΔΣ⟩
⟨*proof*⟩

lemma *atms-of-mm-encode-clause-subset2*: ⟨*finite* ΔΣ ⟹ ΔΣ ⊆ *atms-of-mm N* ⟹
atms-of-mm N ⊆ *atms-of-mm* (*encode-clauses N*) ∪ ΔΣ⟩
⟨*proof*⟩

lemma *atms-of-mm-penc-subset2*: ⟨*finite* ΔΣ ⟹ ΔΣ ⊆ *atms-of-mm N* ⟹
atms-of-mm (*penc N*) = (*atms-of-mm N* − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ⟩
⟨*proof*⟩

theorem *card-atms-of-mm-penc*:
assumes ⟨*finite* ΔΣ⟩ **and** ⟨ΔΣ ⊆ *atms-of-mm N*⟩

shows ‹*card* (*atms-of-mm* (*penc N*)) ≤ *card* (*atms-of-mm N* − ΔΣ) + *2* ∗ *card* ΔΣ› (**is** ‹*?A* ≤ *?B*›)
⟨*proof*⟩

definition *postp* :: ‹'*v partial-interp* ⇒ '*v partial-interp*› **where**
‹*postp I* =
{*A* ∈ *I*. *atm-of A* ∉ ΔΣ ∧ *atm-of A* ∈ Σ} ∪ *Pos* ‘ {*A*. *A* ∈ ΔΣ ∧ *Pos* (*replacement-pos A*) ∈ *I*}
∪ *Neg* ‘ {*A*. *A* ∈ ΔΣ ∧ *Pos* (*replacement-neg A*) ∈ *I* ∧ *Pos* (*replacement-pos A*) ∉ *I*}›

lemma *preprocess-clss-model-additional-variables2*:
assumes
‹*atm-of A* ∈ Σ − ΔΣ›
shows
‹*A* ∈ *postp I* ⟷ *A* ∈ *I*› (**is** *?A*)
⟨*proof*⟩

lemma *encode-clause-iff*:
assumes
‹⋀*A*. *A* ∈ ΔΣ ⟹ *Pos A* ∈ *I* ⟷ *Pos* (*replacement-pos A*) ∈ *I*›
‹⋀*A*. *A* ∈ ΔΣ ⟹ *Neg A* ∈ *I* ⟷ *Pos* (*replacement-neg A*) ∈ *I*›
shows ‹*I* ⊨ *encode-clause C* ⟷ *I* ⊨ *C*›
⟨*proof*⟩

lemma *encode-clauses-iff*:
assumes
‹⋀*A*. *A* ∈ ΔΣ ⟹ *Pos A* ∈ *I* ⟷ *Pos* (*replacement-pos A*) ∈ *I*›
‹⋀*A*. *A* ∈ ΔΣ ⟹ *Neg A* ∈ *I* ⟷ *Pos* (*replacement-neg A*) ∈ *I*›
shows ‹*I* ⊨*m encode-clauses C* ⟷ *I* ⊨*m C*›
⟨*proof*⟩

definition Σ_{add} **where**
‹Σ_{add} = *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›

definition *upostp* :: ‹'*v partial-interp* ⇒ '*v partial-interp*› **where**
‹*upostp I* =
Neg ‘ {*A* ∈ Σ. *A* ∉ ΔΣ ∧ *Pos A* ∉ *I* ∧ *Neg A* ∉ *I*}
∪ {*A* ∈ *I*. *atm-of A* ∈ Σ ∧ *atm-of A* ∉ ΔΣ}
∪ *Pos* ‘ *replacement-pos* ‘ {*A* ∈ ΔΣ. *Pos A* ∈ *I*}
∪ *Neg* ‘ *replacement-pos* ‘ {*A* ∈ ΔΣ. *Pos A* ∉ *I*}
∪ *Pos* ‘ *replacement-neg* ‘ {*A* ∈ ΔΣ. *Neg A* ∈ *I*}
∪ *Neg* ‘ *replacement-neg* ‘ {*A* ∈ ΔΣ. *Neg A* ∉ *I*}›

lemma *atm-of-upostp-subset*:
‹*atm-of* ‘ (*upostp I*) ⊆
(*atm-of* ‘ *I* − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ ∪
replacement-neg ‘ ΔΣ ∪ Σ›
⟨*proof*⟩

inductive *odecide* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
odecide-noweight: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) *L*› **and**
‹*atm-of L* ∈ *atms-of-mm* (*init-clss S*)› **and**
‹*T* ∼ *cons-trail* (*Decided L*) *S*› **and**

⟨*atm-of* $L \in \Sigma - \Delta\Sigma$⟩ |
odecide-replacement-pos: ⟨*odecide* S T⟩
if
⟨*conflicting* $S = None$⟩ **and**
⟨*undefined-lit* (*trail* S) (*Pos* (*replacement-pos* L))⟩ **and**
⟨$T \sim$ *cons-trail* (*Decided* (*Pos* (*replacement-pos* L))) S⟩ **and**
⟨$L \in \Delta\Sigma$⟩ |
odecide-replacement-neg: ⟨*odecide* S T⟩
if
⟨*conflicting* $S = None$⟩ **and**
⟨*undefined-lit* (*trail* S) (*Pos* (*replacement-neg* L))⟩ **and**
⟨$T \sim$ *cons-trail* (*Decided* (*Pos* (*replacement-neg* L))) S⟩ **and**
⟨$L \in \Delta\Sigma$⟩

inductive-cases *odecideE*: ⟨*odecide* S T⟩

definition *no-new-lonely-clause* :: ⟨*'v clause* ⇒ *bool*⟩ **where**
⟨*no-new-lonely-clause* $C \longleftrightarrow$
($\forall L \in \Delta\Sigma$. $L \in$ *atms-of* $C \longrightarrow$
Neg (*replacement-pos* L) $\in\#$ $C \vee$ *Neg* (*replacement-neg* L) $\in\#$ $C \vee C \in\#$ *additional-constraint* L)⟩

definition *lonely-weighted-lit-decided* **where**
⟨*lonely-weighted-lit-decided* $S \longleftrightarrow$
($\forall L \in \Delta\Sigma$. *Decided* (*Pos* L) $\notin$ *set* (*trail* S) $\wedge$ *Decided* (*Neg* L) $\notin$ *set* (*trail* S))⟩

end

locale *optimal-encoding* = *optimal-encoding-opt*
state-eq
state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
ϱ
update-additional-info
Σ $\Delta\Sigma$
new-vars
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** $\sim$ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'v clause option × *'b* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**
cons-trail :: (*'v*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**

remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
ϱ :: ‹*'v clause* ⇒ *'a* :: {*linorder*}› **and**
update-additional-info :: ‹*'v clause option* × *'b* ⇒ *'st* ⇒ *'st*› **and**
Σ ΔΣ :: ‹*'v set*› **and**
new-vars :: ‹*'v* ⇒ *'v* × *'v*› +
assumes
finite-Σ:
‹*finite* ΔΣ› **and**
ΔΣ-Σ:
‹ΔΣ ⊆ Σ› **and**
new-vars-pos:
‹*A* ∈ ΔΣ ⟹ *replacement-pos A* ∉ Σ› **and**
new-vars-neg:
‹*A* ∈ ΔΣ ⟹ *replacement-neg A* ∉ Σ› **and**
new-vars-dist:
‹*inj-on replacement-pos* ΔΣ›
‹*inj-on replacement-neg* ΔΣ›
‹*replacement-pos* ‘ ΔΣ ∩ *replacement-neg* ‘ ΔΣ = {}› **and**
Σ-*no-weight*:
‹*atm-of C* ∈ Σ − ΔΣ ⟹ *ϱ* (*add-mset C M*) = *ϱ M*›
begin

lemma *new-vars-dist2*:
‹*A* ∈ ΔΣ ⟹ *B* ∈ ΔΣ ⟹ *A* ≠ *B* ⟹ *replacement-pos A* ≠ *replacement-pos B*›
‹*A* ∈ ΔΣ ⟹ *B* ∈ ΔΣ ⟹ *A* ≠ *B* ⟹ *replacement-neg A* ≠ *replacement-neg B*›
‹*A* ∈ ΔΣ ⟹ *B* ∈ ΔΣ ⟹ *replacement-neg A* ≠ *replacement-pos B*›
⟨*proof*⟩

lemma *consistent-interp-postp*:
‹*consistent-interp I* ⟹ *consistent-interp* (*postp I*)›
⟨*proof*⟩

The reverse of the previous theorem does not hold due to the filtering on the variables of ΔΣ. One example of version that holds:

lemma
assumes ‹*A* ∈ ΔΣ›
shows ‹*consistent-interp* (*postp* {*Pos A* , *Neg A*})› **and**
‹¬*consistent-interp* {*Pos A*, *Neg A*}›
⟨*proof*⟩

Some more restricted version of the reverse hold, like:

lemma *consistent-interp-postp-iff*:
‹*atm-of* ‘ *I* ⊆ Σ − ΔΣ ⟹ *consistent-interp I* ⟷ *consistent-interp* (*postp I*)›
⟨*proof*⟩

lemma *new-vars-different-iff*[*simp*]:
‹*A* ≠ $x^{\mapsto 1}$›
‹*A* ≠ $x^{\mapsto 0}$›
‹$x^{\mapsto 1}$ ≠ *A*›
‹$x^{\mapsto 0}$ ≠ *A*›
‹$A^{\mapsto 0}$ ≠ $x^{\mapsto 1}$›

‹$A^{\mapsto 1} \neq x^{\mapsto 0}$›
‹$A^{\mapsto 0} = x^{\mapsto 0} \longleftrightarrow A = x$›
‹$A^{\mapsto 1} = x^{\mapsto 1} \longleftrightarrow A = x$›
‹$(A^{\mapsto 1}) \notin \Sigma$›
‹$(A^{\mapsto 0}) \notin \Sigma$›
‹$(A^{\mapsto 1}) \notin \Delta\Sigma$›
‹$(A^{\mapsto 0}) \notin \Delta\Sigma$› **if** ‹$A \in \Delta\Sigma$› ‹$x \in \Delta\Sigma$› **for** A x
⟨*proof*⟩

lemma *consistent-interp-upostp*:
‹*consistent-interp* $I \Longrightarrow$ *consistent-interp* (*upostp* I)›
⟨*proof*⟩

lemma *atm-of-upostp-subset2*:
‹*atm-of* ‘ $I \subseteq \Sigma \Longrightarrow$ *replacement-pos* ‘ $\Delta\Sigma \cup$
replacement-neg ‘ $\Delta\Sigma \cup (\Sigma - \Delta\Sigma) \subseteq$ *atm-of* ‘ (*upostp* I)›
⟨*proof*⟩

lemma $\Delta\Sigma$-*notin-upost*[*simp*]:
‹$y \in \Delta\Sigma \Longrightarrow$ *Neg* $y \notin$ *upostp* I›
‹$y \in \Delta\Sigma \Longrightarrow$ *Pos* $y \notin$ *upostp* I›
⟨*proof*⟩

lemma *penc-ent-upostp*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$› **and**
sat: ‹$I \models sm$ N› **and**
cons: ‹*consistent-interp* I› **and**
atm: ‹*atm-of* ‘ $I \subseteq$ *atms-of-mm* N›
shows ‹*upostp* $I \models m$ *penc* N›
⟨*proof*⟩

lemma *satisfiable-penc*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$› **and**
sat: ‹*satisfiable* (*set-mset* N)›
shows ‹*satisfiable* (*set-mset* (*penc* N))›
⟨*proof*⟩

lemma *penc-ent-postp*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$› **and**
sat: ‹$I \models sm$ *penc* N› **and**
cons: ‹*consistent-interp* I›
shows ‹*postp* $I \models m$ N›
⟨*proof*⟩

lemma *satisfiable-penc-satisfiable*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$› **and**
sat: ‹*satisfiable* (*set-mset* (*penc* N))›
shows ‹*satisfiable* (*set-mset* N)›
⟨*proof*⟩

lemma *satisfiable-penc-iff*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$›
shows ‹*satisfiable* (*set-mset* (*penc* N)) $\longleftrightarrow$ *satisfiable* (*set-mset* N)›
⟨*proof*⟩

abbreviation ϱ_e*-filter* :: ‹*'v literal multiset* ⇒ *'v literal multiset*› **where**
‹ϱ_e*-filter* $M \equiv \{\#L \in\# \mathit{poss}\ (\mathit{mset\text{-}set}\ \Delta\Sigma).\ \mathit{Pos}\ (\mathit{atm\text{-}of}\ L^{\mapsto 1}) \in\# M\#\} +$
$\{\#L \in\# \mathit{negs}\ (\mathit{mset\text{-}set}\ \Delta\Sigma).\ \mathit{Pos}\ (\mathit{atm\text{-}of}\ L^{\mapsto 0}) \in\# M\#\}$›

definition ϱ_e :: ‹*'v literal multiset* ⇒ *'a* :: {*linorder*}› **where**
‹$\varrho_e\ M = \varrho\ (\varrho_e$*-filter* $M)$›

lemma ϱ_e*-mono*: ‹*distinct-mset* $B \Longrightarrow A \subseteq\# B \Longrightarrow \varrho_e\ A \leq \varrho_e\ B$›
⟨*proof*⟩

interpretation *enc-weight-opt*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
$\varrho = \varrho_e$ **and**
update-additional-info = *update-additional-info*
⟨*proof*⟩

lemma Σ*-no-weight-*ϱ_e: ‹*atm-of* $C \in \Sigma - \Delta\Sigma \Longrightarrow \varrho_e\ (\mathit{add\text{-}mset}\ C\ M) = \varrho_e\ M$›
⟨*proof*⟩

lemma ϱ*-cancel-notin-*$\Delta\Sigma$:
‹$(\bigwedge x.\ x \in\# M \Longrightarrow \mathit{atm\text{-}of}\ x \in \Sigma - \Delta\Sigma) \Longrightarrow \varrho\ (M + M') = \varrho\ M'$›
⟨*proof*⟩

lemma ϱ*-mono2*:
‹*consistent-interp* (*set-mset* M') ⟹ *distinct-mset* M' ⟹
$(\bigwedge A.\ A \in\# M \Longrightarrow \mathit{atm\text{-}of}\ A \in \Sigma) \Longrightarrow (\bigwedge A.\ A \in\# M' \Longrightarrow \mathit{atm\text{-}of}\ A \in \Sigma) \Longrightarrow$
$\{\#A \in\# M.\ \mathit{atm\text{-}of}\ A \in \Delta\Sigma\#\} \subseteq\# \{\#A \in\# M'.\ \mathit{atm\text{-}of}\ A \in \Delta\Sigma\#\} \Longrightarrow \varrho\ M \leq \varrho\ M'$›
⟨*proof*⟩

lemma *finite-upostp*: ‹*finite* I ⟹ *finite* Σ ⟹ *finite* (*upostp* I)›
⟨*proof*⟩

declare *finite-*Σ[*simp*]
lemma *consistent-interp-unionI*:
‹*consistent-interp* $A \Longrightarrow$ *consistent-interp* $B \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow -a \notin B) \Longrightarrow (\bigwedge a.\ a \in B \Longrightarrow -a \notin A) \Longrightarrow$
consistent-interp $(A \cup B)$›
⟨*proof*⟩

lemma *consistent-interp-poss*: ‹*consistent-interp* (*Pos* ‘ A)› **and**
consistent-interp-negs: ‹*consistent-interp* (*Neg* ‘ A)›
⟨*proof*⟩

lemma ϱ_e-*upostp*-ϱ:
  assumes [*simp*]: ‹*finite* Σ› **and**
    ‹*finite* I› **and**
    cons: ‹*consistent-interp* I› **and**
    I-Σ: ‹*atm-of* ‘ $I \subseteq \Sigma$›
  shows ‹ϱ_e (*mset-set* (*upostp* I)) = ϱ (*mset-set* I)› (**is** ‹*?A* = *?B*›)
⟨*proof*⟩

lemma *encode-lit-eq-iff*:
  ‹*atm-of* $x \in \Sigma \Longrightarrow$ *atm-of* $y \in \Sigma \Longrightarrow$ *encode-lit* x = *encode-lit* $y \longleftrightarrow x = y$›
  ⟨*proof*⟩

lemma *distinct-mset-encode-clause-iff*:
  ‹*atms-of* $N \subseteq \Sigma \Longrightarrow$ *distinct-mset* (*encode-clause* N) $\longleftrightarrow$ *distinct-mset* N›
  ⟨*proof*⟩

lemma *distinct-mset-encodes-clause-iff*:
  ‹*atms-of-mm* $N \subseteq \Sigma \Longrightarrow$ *distinct-mset-mset* (*encode-clauses* N) $\longleftrightarrow$ *distinct-mset-mset* N›
  ⟨*proof*⟩

lemma *distinct-additional-constraints*[*simp*]:
  ‹*distinct-mset-mset additional-constraints*›
  ⟨*proof*⟩

lemma *distinct-mset-penc*:
  ‹*atms-of-mm* $N \subseteq \Sigma \Longrightarrow$ *distinct-mset-mset* (*penc* N) $\longleftrightarrow$ *distinct-mset-mset* N›
  ⟨*proof*⟩

lemma *finite-postp*: ‹*finite* $I \Longrightarrow$ *finite* (*postp* I)›
  ⟨*proof*⟩

theorem *full-encoding-OCDCL-correctness*:
  assumes
    st: ‹*full enc-weight-opt.cdcl-bnb-stgy* (*init-state* (*penc* N)) T› **and**
    dist: ‹*distinct-mset-mset* N› **and**
    atms: ‹*atms-of-mm* $N = \Sigma$›
  shows
    ‹*weight* T = *None* $\Longrightarrow$ *unsatisfiable* (*set-mset* N)› **and**
    ‹*weight* $T \neq$ *None* $\Longrightarrow$ *postp* (*set-mset* (*the* (*weight* T))) $\models sm$ N›
    ‹*weight* $T \neq$ *None* $\Longrightarrow$ *distinct-mset* $I \Longrightarrow$ *consistent-interp* (*set-mset* I) $\Longrightarrow$
      atms-of $I \subseteq$ *atms-of-mm* $N \Longrightarrow$ *set-mset* $I \models sm$ $N \Longrightarrow$
      ϱ $I \geq \varrho$ (*mset-set* (*postp* (*set-mset* (*the* (*weight* T)))))›
    ‹*weight* $T \neq$ *None* $\Longrightarrow$ ϱ_e (*the* (*enc-weight-opt.weight* T)) =
      ϱ (*mset-set* (*postp* (*set-mset* (*the* (*enc-weight-opt.weight* T)))))›
⟨*proof*⟩

inductive $ocdcl_W$-*o-r* :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
  decide: *odecide* S S' $\Longrightarrow$ $ocdcl_W$-*o-r* S S' |
  bj: *enc-weight-opt.cdcl-bnb-bj* S S' $\Longrightarrow$ $ocdcl_W$-*o-r* S S'

inductive *cdcl-bnb-r* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** S :: *'st* **where**
  cdcl-conflict: *conflict* S S' $\Longrightarrow$ *cdcl-bnb-r* S S' |
  cdcl-propagate: *propagate* S S' $\Longrightarrow$ *cdcl-bnb-r* S S' |
  cdcl-improve: *enc-weight-opt.improvep* S S' $\Longrightarrow$ *cdcl-bnb-r* S S' |

cdcl-conflict-opt: *enc-weight-opt.conflict-opt S S′ ⟹ cdcl-bnb-r S S′* |
cdcl-o′: *ocdcl$_W$-o-r S S′ ⟹ cdcl-bnb-r S S′*

inductive *cdcl-bnb-r-stgy* :: ‹*'st ⇒ 'st ⇒ bool*› **for** *S* :: *'st* **where**
cdcl-bnb-r-conflict: *conflict S S′ ⟹ cdcl-bnb-r-stgy S S′* |
cdcl-bnb-r-propagate: *propagate S S′ ⟹ cdcl-bnb-r-stgy S S′* |
cdcl-bnb-r-improve: *enc-weight-opt.improvep S S′ ⟹ cdcl-bnb-r-stgy S S′* |
cdcl-bnb-r-conflict-opt: *enc-weight-opt.conflict-opt S S′ ⟹ cdcl-bnb-r-stgy S S′* |
cdcl-bnb-r-other′: *ocdcl$_W$-o-r S S′ ⟹ no-confl-prop-impr S ⟹ cdcl-bnb-r-stgy S S′*

lemma *ocdcl$_W$-o-r-cases*[*consumes 1*, *case-names odecode obacktrack skip resolve*]:
assumes
‹*ocdcl$_W$-o-r S T*›
‹*odecide S T ⟹ P T*›
‹*enc-weight-opt.obacktrack S T ⟹ P T*›
‹*skip S T ⟹ P T*›
‹*resolve S T ⟹ P T*›
shows ‹*P T*›
⟨*proof*⟩

context
fixes *S* :: *'st*
assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = (Σ − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ
∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *odecide-decide*:
‹*odecide S T ⟹ decide S T*›
⟨*proof*⟩

lemma *ocdcl$_W$-o-r-ocdcl$_W$-o*:
‹*ocdcl$_W$-o-r S T ⟹ enc-weight-opt.ocdcl$_W$-o S T*›
⟨*proof*⟩

lemma *cdcl-bnb-r-cdcl-bnb*:
‹*cdcl-bnb-r S T ⟹ enc-weight-opt.cdcl-bnb S T*›
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹*cdcl-bnb-r-stgy S T ⟹ enc-weight-opt.cdcl-bnb-stgy S T*›
⟨*proof*⟩

end

context
fixes *S* :: *'st*
assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = (Σ − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ
∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *rtranclp-cdcl-bnb-r-cdcl-bnb*:
‹*cdcl-bnb-r** S T ⟹ enc-weight-opt.cdcl-bnb** S T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹*cdcl-bnb-r-stgy*** *S T* ⟹ *enc-weight-opt.cdcl-bnb-stgy*** *S T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-all-struct-inv*:
‹*cdcl-bnb-r*** *S T* ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-all-struct-inv*:
‹*cdcl-bnb-r-stgy*** *S T* ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
⟨*proof*⟩

end

lemma *total-entails-iff-no-conflict*:
assumes ‹*atms-of-mm N* ⊆ *atm-of* ‘ *I*› **and** ‹*consistent-interp I*›
shows ‹*I* ⊨*sm N* ⟷ (∀ *C* ∈# *N*. ¬*I* ⊨*s CNot C*)›
⟨*proof*⟩

lemma *no-step-cdcl-bnb-r-stgy-no-step-cdcl-bnb-stgy*:
assumes
N: ‹*init-clss S* = *penc N*› **and**
Σ: ‹*atms-of-mm N* = Σ› **and**
n-d: ‹*no-dup* (*trail S*)› **and**
tr-alien: ‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ Σ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
shows
‹*no-step cdcl-bnb-r-stgy S* ⟷ *no-step enc-weight-opt.cdcl-bnb-stgy S*› (**is** ‹*?A* ⟷ *?B*›)
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-init-clss*:
‹*cdcl-bnb-r-stgy S T* ⟹ *init-clss S* = *init-clss T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-init-clss*:
‹*cdcl-bnb-r-stgy*** *S T* ⟹ *init-clss S* = *init-clss T*›
⟨*proof*⟩

lemma [*simp*]:
‹*enc-weight-opt.abs-state* (*init-state N*) = *abs-state* (*init-state N*)›
⟨*proof*⟩

corollary
assumes
Σ: ‹*atms-of-mm N* = Σ› **and** *dist*: ‹*distinct-mset-mset N*› **and**
‹*full cdcl-bnb-r-stgy* (*init-state* (*penc N*)) *T*›
shows
‹*full enc-weight-opt.cdcl-bnb-stgy* (*init-state* (*penc N*)) *T*›
⟨*proof*⟩

lemma *Neg-in-lits-of-l-definedD*:
‹*Neg A* ∈ *lits-of-l M* ⟹ *defined-lit M* (*Pos A*)›

⟨*proof*⟩

lemma *propagation-one-lit-of-same-lvl*:
assumes
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*no-smaller-propa S*› **and**
‹*Propagated L E* ∈ *set* (*trail S*)› **and**
rea: ‹*reasons-in-clauses S*› **and**
nempty: ‹*E* − {#*L*#} ≠ {#}›
shows
‹∃ *L'* ∈# *E* − {#*L*#}. *get-level* (*trail S*) *L* = *get-level* (*trail S*) *L'*›
⟨*proof*⟩

lemma *simple-backtrack-obacktrack*:
‹*simple-backtrack S T* ⟹ $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
enc-weight-opt.obacktrack S T›
⟨*proof*⟩

end

interpretation *test-real*: *optimal-encoding-opt* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. (*0*::*real*)› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)› **and**
Σ = ‹{*1*..(*100*::*nat*)}› **and**
ΔΣ = ‹{*1*..(*50*::*nat*)}› **and**
new-vars = ‹λ*n*. (*200* + *2*∗*n*, *200* + *2*∗*n*+*1*)›
⟨*proof*⟩

lemma *mult3-inj*:
‹*2* ∗ *A* = *Suc* (*2* ∗ *Aa*) ⟷ *False*› **for** *A Aa*::*nat*
⟨*proof*⟩

interpretation *test-real*: *optimal-encoding* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**

update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. (*0*::*real*)› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)› **and**
Σ = ‹{*1*..(*100*::*nat*)}› **and**
ΔΣ = ‹{*1*..(*50*::*nat*)}› **and**
new-vars = ‹λ*n*. (*200* + *2*∗*n*, *200* + *2*∗*n*+*1*)›
⟨*proof*⟩

interpretation *test-nat*: *optimal-encoding-opt* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. (*0*::*nat*)› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)› **and**
Σ = ‹{*1*..(*100*::*nat*)}› **and**
ΔΣ = ‹{*1*..(*50*::*nat*)}› **and**
new-vars = ‹λ*n*. (*200* + *2*∗*n*, *200* + *2*∗*n*+*1*)›
⟨*proof*⟩

interpretation *test-nat*: *optimal-encoding* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. (*0*::*nat*)› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)› **and**
Σ = ‹{*1*..(*100*::*nat*)}› **and**
ΔΣ = ‹{*1*..(*50*::*nat*)}› **and**
new-vars = ‹λ*n*. (*200* + *2*∗*n*, *200* + *2*∗*n*+*1*)›
⟨*proof*⟩

end
theory *CDCL-W-MaxSAT*
imports *CDCL-W-Optimal-Model*
begin

0.1.3 Partial MAX-SAT

definition *weight-on-clauses* **where**
‹*weight-on-clauses* N_S ϱ I = ($\sum$ C ∈# (*filter-mset* (λC. I ⊨ C) N_S). ϱ C)›

definition *atms-exactly-m* :: ‹'*v partial-interp* ⇒ '*v clauses* ⇒ *bool*› **where**
‹*atms-exactly-m* I N ⟷
total-over-m I (*set-mset* N) ∧
atms-of-s I ⊆ *atms-of-mm* N›

Partial in the name refers to the fact that not all clauses are soft clauses, not to the fact that we consider partial models.

inductive *partial-max-sat* :: ‹'*v clauses* ⇒ '*v clauses* ⇒ ('*v clause* ⇒ *nat*) ⇒
'*v partial-interp option* ⇒ *bool*› **where**
partial-max-sat:
‹*partial-max-sat* N_H N_S ϱ (*Some* I)›
if
‹I ⊨*sm* N_H› **and**
‹*atms-exactly-m* I ((N_H + N_S))› **and**
‹*consistent-interp* I› **and**
‹⋀I'. *consistent-interp* I' ⟹ *atms-exactly-m* I' (N_H + N_S) ⟹ I' ⊨*sm* N_H ⟹
weight-on-clauses N_S ϱ I' ≤ *weight-on-clauses* N_S ϱ I› |
partial-max-unsat:
‹*partial-max-sat* N_H N_S ϱ *None*›
if
‹*unsatisfiable* (*set-mset* N_H)›

inductive *partial-min-sat* :: ‹'*v clauses* ⇒ '*v clauses* ⇒ ('*v clause* ⇒ *nat*) ⇒
'*v partial-interp option* ⇒ *bool*› **where**
partial-min-sat:
‹*partial-min-sat* N_H N_S ϱ (*Some* I)›
if
‹I ⊨*sm* N_H› **and**
‹*atms-exactly-m* I (N_H + N_S)› **and**
‹*consistent-interp* I› **and**
‹⋀I'. *consistent-interp* I' ⟹ *atms-exactly-m* I' (N_H + N_S) ⟹ I' ⊨*sm* N_H ⟹
weight-on-clauses N_S ϱ I' ≥ *weight-on-clauses* N_S ϱ I› |
partial-min-unsat:
‹*partial-min-sat* N_H N_S ϱ *None*›
if
‹*unsatisfiable* (*set-mset* N_H)›

lemma *atms-exactly-m-finite*:
assumes ‹*atms-exactly-m* I N›
shows ‹*finite* I›
⟨*proof*⟩

lemma
fixes N_H :: ‹'*v clauses*›
assumes ‹*satisfiable* (*set-mset* N_H)›
shows *sat-partial-max-sat*: ‹∃ I. *partial-max-sat* N_H N_S ϱ (*Some* I)› **and**
sat-partial-min-sat: ‹∃ I. *partial-min-sat* N_H N_S ϱ (*Some* I)›
⟨*proof*⟩

inductive *weight-sat*

:: ⟨'v clauses ⇒ ('v literal multiset ⇒ 'a :: linorder) ⇒
'v literal multiset option ⇒ bool⟩
where
weight-sat:
⟨*weight-sat N ϱ* (*Some I*)⟩
if
⟨*set-mset I* ⊨*sm N*⟩ **and**
⟨*atms-exactly-m* (*set-mset I*) *N*⟩ **and**
⟨*consistent-interp* (*set-mset I*)⟩ **and**
⟨*distinct-mset I*⟩
⟨⋀*I'*. *consistent-interp* (*set-mset I'*) ⟹ *atms-exactly-m* (*set-mset I'*) *N* ⟹ *distinct-mset I'* ⟹
set-mset I' ⊨*sm N* ⟹ *ϱ I'* ≥ *ϱ I*⟩ |
partial-max-unsat:
⟨*weight-sat N ϱ None*⟩
if
⟨*unsatisfiable* (*set-mset N*)⟩

lemma *partial-max-sat-is-weight-sat*:
fixes *additional-atm* :: ⟨'v *clause* ⇒ 'v⟩ **and**
ϱ :: ⟨'v *clause* ⇒ *nat*⟩ **and**
N_S :: ⟨'v *clauses*⟩
defines
⟨*ϱ'* ≡ (λ*C*. *sum-mset*
((λ*L*. *if L* ∈ *Pos* ‘ *additional-atm* ‘ *set-mset N_S*
then count N_S (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# *N_S*)
∗ *ϱ* (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# *N_S*)
else 0) ‘# *C*))⟩
assumes
add: ⟨⋀*C*. *C* ∈# *N_S* ⟹ *additional-atm C* ∉ *atms-of-mm* (*N_H* + *N_S*)⟩
⟨⋀*C D*. *C* ∈# *N_S* ⟹ *D* ∈# *N_S* ⟹ *additional-atm C* = *additional-atm D* ⟷ *C* = *D*⟩ **and**
w: ⟨*weight-sat* (*N_H* + (λ*C*. *add-mset* (*Pos* (*additional-atm C*)) *C*) ‘# *N_S*) *ϱ'* (*Some I*)⟩
shows
⟨*partial-max-sat N_H N_S ϱ* (*Some* {*L* ∈ *set-mset I*. *atm-of L* ∈ *atms-of-mm* (*N_H* + *N_S*)})⟩
⟨*proof*⟩

lemma *sum-mset-cong*:
⟨(⋀*a*. *a* ∈# *A* ⟹ *f a* = *g a*) ⟹ (∑ *a*∈#*A*. *f a*) = (∑ *a*∈#*A*. *g a*)⟩
⟨*proof*⟩

lemma *partial-max-sat-is-weight-sat-distinct*:
fixes *additional-atm* :: ⟨'v *clause* ⇒ 'v⟩ **and**
ϱ :: ⟨'v *clause* ⇒ *nat*⟩ **and**
N_S :: ⟨'v *clauses*⟩
defines
⟨*ϱ'* ≡ (λ*C*. *sum-mset*
((λ*L*. *if L* ∈ *Pos* ‘ *additional-atm* ‘ *set-mset N_S*
then ϱ (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# *N_S*)
else 0) ‘# *C*))⟩
assumes
⟨*distinct-mset N_S*⟩ **and** — This is implicit on paper
add: ⟨⋀*C*. *C* ∈# *N_S* ⟹ *additional-atm C* ∉ *atms-of-mm* (*N_H* + *N_S*)⟩
⟨⋀*C D*. *C* ∈# *N_S* ⟹ *D* ∈# *N_S* ⟹ *additional-atm C* = *additional-atm D* ⟷ *C* = *D*⟩ **and**
w: ⟨*weight-sat* (*N_H* + (λ*C*. *add-mset* (*Pos* (*additional-atm C*)) *C*) ‘# *N_S*) *ϱ'* (*Some I*)⟩
shows
⟨*partial-max-sat N_H N_S ϱ* (*Some* {*L* ∈ *set-mset I*. *atm-of L* ∈ *atms-of-mm* (*N_H* + *N_S*)})⟩
⟨*proof*⟩

lemma *atms-exactly-m-alt-def*:
‹*atms-exactly-m* (*set-mset y*) *N* ⟷ *atms-of y* ⊆ *atms-of-mm N* ∧
total-over-m (*set-mset y*) (*set-mset N*)›
⟨*proof*⟩

lemma *atms-exactly-m-alt-def2*:
‹*atms-exactly-m* (*set-mset y*) *N* ⟷ *atms-of y* = *atms-of-mm N*›
⟨*proof*⟩

lemma (**in** *conflict-driven-clause-learning*$_W$ *-optimal-weight*) *full-cdcl-bnb-stgy-weight-sat*:
‹*full cdcl-bnb-stgy* (*init-state N*) *T* ⟹ *distinct-mset-mset N* ⟹ *weight-sat N* ϱ (*weight T*)›
⟨*proof*⟩

end
theory *CDCL-W-Partial-Optimal-Model*
imports *CDCL-W-Partial-Encoding*
begin
lemma *isabelle-should-do-that-automatically*: ‹*Suc* (*a* − *Suc 0*) = *a* ⟷ *a* ≥ *1*›
⟨*proof*⟩

lemma (**in** *conflict-driven-clause-learning*$_W$ *-optimal-weight*)
conflict-opt-state-eq-compatible:
‹*conflict-opt S T* ⟹ *S* ∼ *S'* ⟹ *T* ∼ *T'* ⟹ *conflict-opt S' T'*›
⟨*proof*⟩

context *optimal-encoding*
begin

definition *base-atm* :: ‹'*v* ⇒ '*v*› **where**
‹*base-atm L* = (*if L* ∈ Σ − ΔΣ *then L else*
if L ∈ *replacement-neg* ‘ ΔΣ *then* (*SOME K*. (*K* ∈ ΔΣ ∧ *L* = *replacement-neg K*))
else (*SOME K*. (*K* ∈ ΔΣ ∧ *L* = *replacement-pos K*)))›

lemma *normalize-lit-Some-simp*[*simp*]: ‹(*SOME K*. *K* ∈ ΔΣ ∧ (*L*$^{\mapsto 0}$ = *K*$^{\mapsto 0}$)) = *L*› **if** ‹*L* ∈ ΔΣ› **for** *K*
⟨*proof*⟩

lemma *base-atm-simps1*[*simp*]:
‹*L* ∈ Σ ⟹ *L* ∉ ΔΣ ⟹ *base-atm L* = *L*›
⟨*proof*⟩

lemma *base-atm-simps2*[*simp*]:
‹*L* ∈ (Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ ⟹
K ∈ Σ ⟹ *K* ∉ ΔΣ ⟹ *L* ∈ Σ ⟹ *K* = *base-atm L* ⟷ *L* = *K*›
⟨*proof*⟩

lemma *base-atm-simps3*[*simp*]:
‹*L* ∈ Σ − ΔΣ ⟹ *base-atm L* ∈ Σ›
‹*L* ∈ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ ⟹ *base-atm L* ∈ ΔΣ›
⟨*proof*⟩

lemma *base-atm-simps4*[*simp*]:
‹*L* ∈ ΔΣ ⟹ *base-atm* (*replacement-pos L*) = *L*›
‹*L* ∈ ΔΣ ⟹ *base-atm* (*replacement-neg L*) = *L*›

⟨*proof*⟩

fun *normalize-lit* :: ⟨*'v literal* ⇒ *'v literal*⟩ **where**
⟨*normalize-lit* (*Pos L*) =
(*if L* ∈ *replacement-neg* ‘ ΔΣ
then Neg (*replacement-pos* (*SOME K*. (*K* ∈ ΔΣ ∧ *L* = *replacement-neg K*)))
else Pos L)⟩ |
⟨*normalize-lit* (*Neg L*) =
(*if L* ∈ *replacement-neg* ‘ ΔΣ
then Pos (*replacement-pos* (*SOME K*. *K* ∈ ΔΣ ∧ *L* = *replacement-neg K*))
else Neg L)⟩

abbreviation *normalize-clause* :: ⟨*'v clause* ⇒ *'v clause*⟩ **where**
⟨*normalize-clause C* ≡ *normalize-lit* ‘# *C*⟩

lemma *normalize-lit*[*simp*]:
⟨*L* ∈ Σ − ΔΣ ⟹ *normalize-lit* (*Pos L*) = (*Pos L*)⟩
⟨*L* ∈ Σ − ΔΣ ⟹ *normalize-lit* (*Neg L*) = (*Neg L*)⟩
⟨*L* ∈ ΔΣ ⟹ *normalize-lit* (*Pos* (*replacement-neg L*)) = *Neg* (*replacement-pos L*)⟩
⟨*L* ∈ ΔΣ ⟹ *normalize-lit* (*Neg* (*replacement-neg L*)) = *Pos* (*replacement-pos L*)⟩
⟨*proof*⟩

definition *all-clauses-literals* :: ⟨*'v list*⟩ **where**
⟨*all-clauses-literals* =
(*SOME xs*. *mset xs* = *mset-set* ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ))⟩

datatype (**in** −) *'c search-depth* =
sd-is-zero: *SD-ZERO* (*the-search-depth*: *'c*) |
sd-is-one: *SD-ONE* (*the-search-depth*: *'c*) |
sd-is-two: *SD-TWO* (*the-search-depth*: *'c*)

abbreviation (**in** −) *un-hide-sd* :: ⟨*'a search-depth list* ⇒ *'a list*⟩ **where**
⟨*un-hide-sd* ≡ *map the-search-depth*⟩

fun *nat-of-search-deph* :: ⟨*'c search-depth* ⇒ *nat*⟩ **where**
⟨*nat-of-search-deph* (*SD-ZERO* -) = *0*⟩ |
⟨*nat-of-search-deph* (*SD-ONE* -) = *1*⟩ |
⟨*nat-of-search-deph* (*SD-TWO* -) = *2*⟩

definition *opposite-var* **where**
⟨*opposite-var L* = (*if L* ∈ *replacement-pos* ‘ ΔΣ *then replacement-neg* (*base-atm L*)
else replacement-pos (*base-atm L*))⟩

lemma *opposite-var-replacement-if*[*simp*]:
⟨*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *A* ∈ ΔΣ ⟹
opposite-var L = *replacement-pos A* ⟷ *L* = *replacement-neg A*⟩
⟨*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *A* ∈ ΔΣ ⟹
opposite-var L = *replacement-neg A* ⟷ *L* = *replacement-pos A*⟩
⟨*A* ∈ ΔΣ ⟹ *opposite-var* (*replacement-pos A*) = *replacement-neg A*⟩
⟨*A* ∈ ΔΣ ⟹ *opposite-var* (*replacement-neg A*) = *replacement-pos A*⟩

⟨*proof*⟩

context
assumes [*simp*]: ‹*finite* Σ›
begin

lemma *all-clauses-literals*:
‹*mset all-clauses-literals* = *mset-set* ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
‹*distinct all-clauses-literals*›
‹*set all-clauses-literals* = ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
⟨*proof*⟩

definition *unset-literals-in-*Σ **where**
‹*unset-literals-in-*Σ *M L* ⟷ *undefined-lit M* (*Pos L*) ∧ *L* ∈ Σ − ΔΣ›

definition *full-unset-literals-in-*ΔΣ **where**
‹*full-unset-literals-in-*ΔΣ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *undefined-lit M* (*Pos* (*opposite-var L*)) ∧
L ∈ *replacement-pos* ‘ ΔΣ›

definition *full-unset-literals-in-*ΔΣ′ **where**
‹*full-unset-literals-in-*ΔΣ′ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *undefined-lit M* (*Pos* (*opposite-var L*)) ∧
L ∈ *replacement-neg* ‘ ΔΣ›

definition *half-unset-literals-in-*ΔΣ **where**
‹*half-unset-literals-in-*ΔΣ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *defined-lit M* (*Pos* (*opposite-var L*))›

definition *sorted-unadded-literals* :: ‹('*v*, '*v clause*) *ann-lits* ⇒ '*v list*› **where**
‹*sorted-unadded-literals M* =
(*let*
M0 = *filter* (*full-unset-literals-in-*ΔΣ′ *M*) *all-clauses-literals*;
— weight is 0
M1 = *filter* (*unset-literals-in-*Σ *M*) *all-clauses-literals*;
— weight is 2
M2 = *filter* (*full-unset-literals-in-*ΔΣ *M*) *all-clauses-literals*;
— weight is 2
M3 = *filter* (*half-unset-literals-in-*ΔΣ *M*) *all-clauses-literals*
— weight is 1
in
M0 @ *M3* @ *M1* @ *M2*)›

definition *complete-trail* :: ‹('*v*, '*v clause*) *ann-lits* ⇒ ('*v*, '*v clause*) *ann-lits*› **where**
‹*complete-trail M* =
(*map* (*Decided o Pos*) (*sorted-unadded-literals M*) @ *M*)›

lemma *in-sorted-unadded-literals-undefD*:
‹*atm-of* (*lit-of l*) ∈ *set* (*sorted-unadded-literals M*) ⟹ *l* ∉ *set M*›
‹*atm-of* (*l′*) ∈ *set* (*sorted-unadded-literals M*) ⟹ *undefined-lit M l′*›
‹*xa* ∈ *set* (*sorted-unadded-literals M*) ⟹ *lit-of x* = *Neg xa* ⟹ *x* ∉ *set M*› **and**
set-sorted-unadded-literals[*simp*]:
‹*set* (*sorted-unadded-literals M*) =
Set.filter (λ*L*. *undefined-lit M* (*Pos L*)) (*set all-clauses-literals*)›
⟨*proof*⟩

lemma [*simp*]:
‹*full-unset-literals-in-*$\Delta\Sigma$ [] = ($\lambda L.\ L \in$ *replacement-pos* ‘ $\Delta\Sigma$)›
‹*full-unset-literals-in-*$\Delta\Sigma'$ [] = ($\lambda L.\ L \in$ *replacement-neg* ‘ $\Delta\Sigma$)›
‹*half-unset-literals-in-*$\Delta\Sigma$ [] = ($\lambda L.$ *False*)›
‹*unset-literals-in-*Σ [] = ($\lambda L.\ L \in \Sigma - \Delta\Sigma$)›
⟨*proof*⟩

lemma *filter-disjount-union*:
‹($\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *P x* $\Longrightarrow \neg$*Q x*) $\Longrightarrow$
length (*filter P xs*) + *length* (*filter Q xs*) =
length (*filter* ($\lambda x.\ P\ x \vee Q\ x$) *xs*)›
⟨*proof*⟩
lemma *length-sorted-unadded-literals-empty*[*simp*]:
‹*length* (*sorted-unadded-literals* []) = *length all-clauses-literals*›
⟨*proof*⟩

lemma *sorted-unadded-literals-Cons-notin-all-clauses-literals*[*simp*]:
assumes
‹*atm-of* (*lit-of K*) $\notin$ *set all-clauses-literals*›
shows
‹*sorted-unadded-literals* (*K* # *M*) = *sorted-unadded-literals M*›
⟨*proof*⟩

lemma *sorted-unadded-literals-cong*:
assumes ‹$\bigwedge L.\ L \in$ *set all-clauses-literals* $\Longrightarrow$ *defined-lit M* (*Pos L*) = *defined-lit M'* (*Pos L*)›
shows ‹*sorted-unadded-literals M* = *sorted-unadded-literals M'*›
⟨*proof*⟩

lemma *sorted-unadded-literals-Cons-already-set*[*simp*]:
assumes
‹*defined-lit M* (*lit-of K*)›
shows
‹*sorted-unadded-literals* (*K* # *M*) = *sorted-unadded-literals M*›
⟨*proof*⟩

lemma *distinct-sorted-unadded-literals*[*simp*]:
‹*distinct* (*sorted-unadded-literals M*)›
⟨*proof*⟩

lemma *Collect-req-remove1*:
‹{*a* ∈ *A*. *a* ≠ *b* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})› **and**
Collect-req-remove2:
‹{*a* ∈ *A*. *b* ≠ *a* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})›
⟨*proof*⟩

lemma *card-remove*:
‹*card* (*Set.remove a A*) = (*if a* ∈ *A then card A* − *1 else card A*)›
⟨*proof*⟩

lemma *sorted-unadded-literals-cons-in-undef*[*simp*]:
‹*undefined-lit M* (*lit-of K*) $\Longrightarrow$
atm-of (*lit-of K*) ∈ *set all-clauses-literals* $\Longrightarrow$
Suc (*length* (*sorted-unadded-literals* (*K* # *M*))) =
length (*sorted-unadded-literals M*)›

⟨*proof*⟩

lemma *no-dup-complete-trail*[*simp*]:
⟨*no-dup* (*complete-trail M*) ⟷ *no-dup M*⟩
⟨*proof*⟩

lemma *tautology-complete-trail*[*simp*]:
⟨*tautology* (*lit-of* ‘# *mset* (*complete-trail M*)) ⟷ *tautology* (*lit-of* ‘# *mset M*)⟩
⟨*proof*⟩

lemma *atms-of-complete-trail*:
⟨*atms-of* (*lit-of* ‘# *mset* (*complete-trail M*)) =
atms-of (*lit-of* ‘# *mset M*) ∪ (Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ⟩
⟨*proof*⟩

fun *depth-lit-of* :: ⟨('*v*,-) *ann-lit* ⇒ ('*v*, -) *ann-lit search-depth*⟩ **where**
⟨*depth-lit-of* (*Decided L*) = *SD-TWO* (*Decided L*)⟩ |
⟨*depth-lit-of* (*Propagated L C*) = *SD-ZERO* (*Propagated L C*)⟩

fun *depth-lit-of-additional-fst* :: ⟨('*v*,-) *ann-lit* ⇒ ('*v*, -) *ann-lit search-depth*⟩ **where**
⟨*depth-lit-of-additional-fst* (*Decided L*) = *SD-ONE* (*Decided L*)⟩ |
⟨*depth-lit-of-additional-fst* (*Propagated L C*) = *SD-ZERO* (*Propagated L C*)⟩

fun *depth-lit-of-additional-snd* :: ⟨('*v*,-) *ann-lit* ⇒ ('*v*, -) *ann-lit search-depth list*⟩ **where**
⟨*depth-lit-of-additional-snd* (*Decided L*) = [*SD-ONE* (*Decided L*)]⟩ |
⟨*depth-lit-of-additional-snd* (*Propagated L C*) = []⟩

This function is suprisingly complicated to get right. Remember that the last set element is at the beginning of the list

fun *remove-dup-information-raw* :: ⟨('*v*, -) *ann-lits* ⇒ ('*v*, -) *ann-lit search-depth list*⟩ **where**
⟨*remove-dup-information-raw* [] = []⟩ |
⟨*remove-dup-information-raw* (*L* # *M*) =
(*if atm-of* (*lit-of L*) ∈ Σ − ΔΣ *then depth-lit-of L* # *remove-dup-information-raw M*
else if defined-lit (*M*) (*Pos* (*opposite-var* (*atm-of* (*lit-of L*))))
then if Decided (*Pos* (*opposite-var* (*atm-of* (*lit-of L*)))) ∈ *set* (*M*)
then remove-dup-information-raw M
else depth-lit-of-additional-fst L # *remove-dup-information-raw M*
else depth-lit-of-additional-snd L @ *remove-dup-information-raw M*)⟩

definition *remove-dup-information* **where**
⟨*remove-dup-information xs* = *un-hide-sd* (*remove-dup-information-raw xs*)⟩

lemma [*simp*]: ⟨*the-search-depth* (*depth-lit-of L*) = *L*⟩
⟨*proof*⟩

lemma *length-complete-trail*[*simp*]: ⟨*length* (*complete-trail* []) = *length all-clauses-literals*⟩
⟨*proof*⟩

lemma *distinct-count-list-if*: ⟨*distinct xs* ⟹ *count-list xs x* = (*if x* ∈ *set xs then 1 else 0*)⟩
⟨*proof*⟩

lemma *length-complete-trail-Cons*:
⟨*no-dup* (*K* # *M*) ⟹
length (*complete-trail* (*K* # *M*)) =

(*if atm-of (lit-of K) ∈ set all-clauses-literals then 0 else 1*) + *length* (*complete-trail M*)›
⟨*proof*⟩

lemma *length-complete-trail-eq*:
‹*no-dup M* ⟹ *atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
length (*complete-trail M*) = *length all-clauses-literals*›
⟨*proof*⟩

lemma *in-set-all-clauses-literals-simp*[*simp*]:
‹*atm-of L* ∈ Σ − ΔΣ ⟹ *atm-of L* ∈ *set all-clauses-literals*›
‹*K* ∈ ΔΣ ⟹ *replacement-pos K* ∈ *set all-clauses-literals*›
‹*K* ∈ ΔΣ ⟹ *replacement-neg K* ∈ *set all-clauses-literals*›
⟨*proof*⟩

lemma [*simp*]:
‹*remove-dup-information* [] = []›
⟨*proof*⟩

lemma *atm-of-remove-dup-information*:
‹*atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
atm-of ‘ (*lits-of-l* (*remove-dup-information M*)) ⊆ *set all-clauses-literals*›
⟨*proof*⟩

primrec *remove-dup-information-raw2* :: ‹('*v*, -) *ann-lits* ⇒ ('*v*, -) *ann-lits* ⇒
('*v*, -) *ann-lit search-depth list*› **where**
‹*remove-dup-information-raw2 M'* [] = []› |
‹*remove-dup-information-raw2 M'* (*L* # *M*) =
(*if atm-of* (*lit-of L*) ∈ Σ − ΔΣ *then depth-lit-of L* # *remove-dup-information-raw2 M' M*
else if defined-lit (*M* @ *M'*) (*Pos* (*opposite-var* (*atm-of* (*lit-of L*))))
then if Decided (*Pos* (*opposite-var* (*atm-of* (*lit-of L*)))) ∈ *set* (*M* @ *M'*)
then remove-dup-information-raw2 M' M
else depth-lit-of-additional-fst L # *remove-dup-information-raw2 M' M*
else depth-lit-of-additional-snd L @ *remove-dup-information-raw2 M' M*)›

lemma *remove-dup-information-raw2-Nil*[*simp*]:
‹*remove-dup-information-raw2* [] *M* = *remove-dup-information-raw M*›
⟨*proof*⟩

This can be useful as simp, but I am not certain (yet), because the RHS does not look simpler than the LHS.

lemma *remove-dup-information-raw-cons*:
‹*remove-dup-information-raw* (*L* # *M2*) =
remove-dup-information-raw2 M2 [*L*] @
remove-dup-information-raw M2›
⟨*proof*⟩

lemma *remove-dup-information-raw-append*:
‹*remove-dup-information-raw* (*M1* @ *M2*) =
remove-dup-information-raw2 M2 M1 @
remove-dup-information-raw M2›
⟨*proof*⟩

lemma *remove-dup-information-raw-append2*:

‹*remove-dup-information-raw2 M* (*M1* @ *M2*) =
remove-dup-information-raw2 (*M* @ *M2*) *M1* @
remove-dup-information-raw2 M M2›
⟨*proof*⟩

lemma *remove-dup-information-subset*: ‹*mset* (*remove-dup-information M*) ⊆# *mset M*›
⟨*proof*⟩

lemma *no-dup-subsetD*: ‹*no-dup M* ⟹ *mset M′* ⊆# *mset M* ⟹ *no-dup M′*›
⟨*proof*⟩

lemma *no-dup-remove-dup-information*:
‹*no-dup M* ⟹ *no-dup* (*remove-dup-information M*)›
⟨*proof*⟩

lemma *atm-of-complete-trail*:
‹*atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
atm-of ‘ (*lits-of-l* (*complete-trail M*)) = *set all-clauses-literals*›
⟨*proof*⟩

lemmas [*simp del*] =
remove-dup-information-raw.simps
remove-dup-information-raw2.simps

lemmas [*simp*] =
remove-dup-information-raw-append
remove-dup-information-raw-cons
remove-dup-information-raw-append2

definition *truncate-trail* :: ‹('*v*, -) *ann-lits* ⇒ -› **where**
‹*truncate-trail M* ≡
(*snd* (*backtrack-split M*))›

definition *ocdcl-score* :: ‹('*v*, -) *ann-lits* ⇒ -› **where**
‹*ocdcl-score M* =
rev (*map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail* (*truncate-trail M*))))›

interpretation *enc-weight-opt*: *conflict-driven-clause-learning*$_W$ *-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
$\varrho = \varrho_e$ **and**
update-additional-info = *update-additional-info*
⟨*proof*⟩

lemma
‹$(a,\ b) \in \mathit{lexn\ less\text{-}than}\ n \Longrightarrow (b,\ c) \in \mathit{lexn\ less\text{-}than}\ n \vee b = c \Longrightarrow (a,\ c) \in \mathit{lexn\ less\text{-}than}\ n$›
‹$(a,\ b) \in \mathit{lexn\ less\text{-}than}\ n \Longrightarrow (b,\ c) \in \mathit{lexn\ less\text{-}than}\ n \vee b = c \Longrightarrow (a,\ c) \in \mathit{lexn\ less\text{-}than}\ n$›
⟨*proof*⟩

lemma *truncate-trail-Prop*[*simp*]:
‹$\mathit{truncate\text{-}trail}\ (\mathit{Propagated}\ L\ E\ \#\ S) = \mathit{truncate\text{-}trail}\ (S)$›
⟨*proof*⟩

lemma *ocdcl-score-Prop*[*simp*]:
‹$\mathit{ocdcl\text{-}score}\ (\mathit{Propagated}\ L\ E\ \#\ S) = \mathit{ocdcl\text{-}score}\ (S)$›
⟨*proof*⟩

lemma *remove-dup-information-raw2-undefined-*Σ:
‹$\mathit{distinct}\ xs \Longrightarrow$
$(\bigwedge L.\ L \in \mathit{set}\ xs \Longrightarrow \mathit{undefined\text{-}lit}\ M\ (\mathit{Pos}\ L) \Longrightarrow L \in \Sigma \Longrightarrow \mathit{undefined\text{-}lit}\ MM\ (\mathit{Pos}\ L)) \Longrightarrow$
$\mathit{remove\text{-}dup\text{-}information\text{-}raw2}\ MM$
$(\mathit{map}\ (\mathit{Decided} \circ \mathit{Pos})$
$(\mathit{filter}\ (\mathit{unset\text{-}literals\text{-}in\text{-}}\Sigma\ M)$
$xs)) =$
$\mathit{map}\ (\mathit{SD\text{-}TWO\ o\ Decided} \circ \mathit{Pos})$
$(\mathit{filter}\ (\mathit{unset\text{-}literals\text{-}in\text{-}}\Sigma\ M)$
$xs)$›
⟨*proof*⟩

lemma *defined-lit-map-Decided-pos*:
‹$\mathit{defined\text{-}lit}\ (\mathit{map}\ (\mathit{Decided} \circ \mathit{Pos})\ M)\ L \longleftrightarrow \mathit{atm\text{-}of}\ L \in \mathit{set}\ M$›
⟨*proof*⟩

lemma *remove-dup-information-raw2-full-undefined-*Σ:
‹$\mathit{distinct}\ xs \Longrightarrow \mathit{set}\ xs \subseteq \mathit{set}\ \mathit{all\text{-}clauses\text{-}literals} \Longrightarrow$
$(\bigwedge L.\ L \in \mathit{set}\ xs \Longrightarrow \mathit{undefined\text{-}lit}\ M\ (\mathit{Pos}\ L) \Longrightarrow L \notin \Sigma - \Delta\Sigma \Longrightarrow$
$\mathit{undefined\text{-}lit}\ M\ (\mathit{Pos}\ (\mathit{opposite\text{-}var}\ L)) \Longrightarrow L \in \mathit{replacement\text{-}pos}\ `\ \Delta\Sigma \Longrightarrow$
$\mathit{undefined\text{-}lit}\ MM\ (\mathit{Pos}\ (\mathit{opposite\text{-}var}\ L))) \Longrightarrow$
$\mathit{remove\text{-}dup\text{-}information\text{-}raw2}\ MM$
$(\mathit{map}\ (\mathit{Decided} \circ \mathit{Pos})$
$(\mathit{filter}\ (\mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma\ M)$
$xs)) =$
$\mathit{map}\ (\mathit{SD\text{-}ONE\ o\ Decided} \circ \mathit{Pos})$
$(\mathit{filter}\ (\mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma\ M)$
$xs)$›
⟨*proof*⟩

lemma *full-unset-literals-in-*Δ*Σ-notin*[*simp*]:
‹$La \in \Sigma \Longrightarrow \mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma\ M\ La \longleftrightarrow \mathit{False}$›
‹$La \in \Sigma \Longrightarrow \mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma'\ M\ La \longleftrightarrow \mathit{False}$›
⟨*proof*⟩

lemma *Decided-in-definedD*: ‹$\mathit{Decided}\ K \in \mathit{set}\ M \Longrightarrow \mathit{defined\text{-}lit}\ M\ K$›
⟨*proof*⟩

lemma *full-unset-literals-in-*Δ*Σ′-full-unset-literals-in-*Δ*Σ*:
‹$L \in \mathit{replacement\text{-}pos}\ `\ \Delta\Sigma \cup \mathit{replacement\text{-}neg}\ `\ \Delta\Sigma \Longrightarrow$
$\mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma'\ M\ (\mathit{opposite\text{-}var}\ L) \longleftrightarrow \mathit{full\text{-}unset\text{-}literals\text{-}in\text{-}}\Delta\Sigma\ M\ L$›
⟨*proof*⟩

lemma *remove-dup-information-raw2-full-unset-literals-in-ΔΣ′*:
‹(⋀*L*. *L* ∈ *set* (*filter* (*full-unset-literals-in-ΔΣ′ M*) *xs*) ⟹ *Decided* (*Pos* (*opposite-var L*)) ∈ *set M′*)
⟹
set xs ⊆ *set all-clauses-literals* ⟹
(*remove-dup-information-raw2*
M′
(*map* (*Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-ΔΣ′* (*M*))
xs))) = []›
⟨*proof*⟩

lemma
fixes *M* :: ‹('*v*, -) *ann-lits*› **and** *L* :: ‹('*v*, -) *ann-lit*›
defines ‹*n1* ≡ *map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail* (*L* # *M*)))› **and**
‹*n2* ≡ *map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail M*))›
assumes
lits: ‹*atm-of* ‘ (*lits-of-l* (*L* # *M*)) ⊆ *set all-clauses-literals*› **and**
undef: ‹*undefined-lit M* (*lit-of L*)›
shows
‹(*rev n1*, *rev n2*) ∈ *lexn less-than n* ∨ *n1* = *n2*›
⟨*proof*⟩
lemma
defines ‹*n* ≡ *card* Σ›
assumes
‹*init-clss S* = *penc N*› **and**
‹*enc-weight-opt.cdcl-bnb-stgy S T*› **and**
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows ‹(*ocdcl-score* (*trail T*), *ocdcl-score* (*trail S*)) ∈ *lexn less-than n* ∨
ocdcl-score (*trail T*) = *ocdcl-score* (*trail S*)›
⟨*proof*⟩

end

interpretation *enc-weight-opt*: $conflict\text{-}driven\text{-}clause\text{-}learning_W$*-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
ϱ = ϱ_e **and**
update-additional-info = *update-additional-info*
⟨*proof*⟩

inductive *simple-backtrack-conflict-opt* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
‹*simple-backtrack-conflict-opt S T*›
if

⟨*backtrack-split* (*trail S*) = (*M2*, *Decided K # M1*)⟩ **and**
⟨*negate-ann-lits* (*trail S*) ∈# *enc-weight-opt.conflicting-clss S*⟩ **and**
⟨*conflicting S* = *None*⟩ **and**
⟨*T* ∼ *cons-trail* (*Propagated* (−*K*) (*DECO-clause* (*trail S*)))
(*add-learned-cls* (*DECO-clause* (*trail S*)) (*reduce-trail-to M1 S*))⟩

inductive-cases *simple-backtrack-conflict-optE*: ⟨*simple-backtrack-conflict-opt S T*⟩

lemma *simple-backtrack-conflict-opt-conflict-analysis*:
assumes ⟨*simple-backtrack-conflict-opt S U*⟩ **and**
inv: ⟨$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)⟩
shows ⟨∃ *T T′*. *enc-weight-opt.conflict-opt S T* ∧ *resolve*** *T T′*
∧ *enc-weight-opt.obacktrack T′ U*⟩
⟨*proof*⟩

inductive *conflict-opt0* :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ **where**
⟨*conflict-opt0 S T*⟩
if
⟨*count-decided* (*trail S*) = *0*⟩ **and**
⟨*negate-ann-lits* (*trail S*) ∈# *enc-weight-opt.conflicting-clss S*⟩ **and**
⟨*conflicting S* = *None*⟩ **and**
⟨*T* ∼ *update-conflicting* (*Some* {#}) (*reduce-trail-to* ([] :: (*'v*, *'v clause*) *ann-lits*) *S*)⟩

inductive-cases *conflict-opt0E*: ⟨*conflict-opt0 S T*⟩

inductive *cdcl-dpll-bnb-r* :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ **for** *S* :: *'st* **where**
cdcl-conflict: *conflict S S′* ⟹ *cdcl-dpll-bnb-r S S′* |
cdcl-propagate: *propagate S S′* ⟹ *cdcl-dpll-bnb-r S S′* |
cdcl-improve: *enc-weight-opt.improvep S S′* ⟹ *cdcl-dpll-bnb-r S S′* |
cdcl-conflict-opt0: *conflict-opt0 S S′* ⟹ *cdcl-dpll-bnb-r S S′* |
cdcl-simple-backtrack-conflict-opt:
⟨*simple-backtrack-conflict-opt S S′* ⟹ *cdcl-dpll-bnb-r S S′*⟩ |
cdcl-o′: $ocdcl_W$*-o-r S S′* ⟹ *cdcl-dpll-bnb-r S S′*

inductive *cdcl-dpll-bnb-r-stgy* :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ **for** *S* :: *'st* **where**
cdcl-dpll-bnb-r-conflict: *conflict S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′* |
cdcl-dpll-bnb-r-propagate: *propagate S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′* |
cdcl-dpll-bnb-r-improve: *enc-weight-opt.improvep S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′* |
cdcl-dpll-bnb-r-conflict-opt0: *conflict-opt0 S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′* |
cdcl-dpll-bnb-r-simple-backtrack-conflict-opt:
⟨*simple-backtrack-conflict-opt S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*⟩ |
cdcl-dpll-bnb-r-other′: $ocdcl_W$*-o-r S S′* ⟹ *no-confl-prop-impr S* ⟹ *cdcl-dpll-bnb-r-stgy S S′*

lemma *no-dup-dropI*:
⟨*no-dup M* ⟹ *no-dup* (*drop n M*)⟩
⟨*proof*⟩

lemma *tranclp-resolve-state-eq-compatible*:
⟨*resolve*$^{++}$ *S T* ⟹*T* ∼ *T′* ⟹ *resolve*$^{++}$ *S T′*⟩
⟨*proof*⟩

lemma *conflict-opt0-state-eq-compatible*:
⟨*conflict-opt0 S T* ⟹ *S* ∼ *S′* ⟹ *T* ∼ *T′* ⟹ *conflict-opt0 S′ T′*⟩
⟨*proof*⟩

lemma *conflict-opt0-conflict-opt*:
assumes ‹*conflict-opt0 S U*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹∃ *T*. *enc-weight-opt.conflict-opt S T* ∧ *resolve*** *T U*›
⟨*proof*⟩

lemma *backtrack-split-some-is-decided-then-snd-has-hd2*:
‹∃ *l*∈*set M*. *is-decided l* ⟹ ∃ *M' L' M''*. *backtrack-split M* = (*M''*, *Decided L'* # *M'*)›
⟨*proof*⟩

lemma *no-step-conflict-opt0-simple-backtrack-conflict-opt*:
‹*no-step conflict-opt0 S* ⟹ *no-step simple-backtrack-conflict-opt S* ⟹
no-step enc-weight-opt.conflict-opt S›
⟨*proof*⟩

lemma *no-step-cdcl-dpll-bnb-r-cdcl-bnb-r*:
assumes ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows
‹*no-step cdcl-dpll-bnb-r S* ⟷ *no-step cdcl-bnb-r S*› (**is** ‹*?A* ⟷ *?B*›)
⟨*proof*⟩

lemma *cdcl-dpll-bnb-r-cdcl-bnb-r*:
assumes ‹*cdcl-dpll-bnb-r S T*› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*cdcl-bnb-r*** *S T*›
⟨*proof*⟩

lemma *resolve-no-prop-confl*: ‹*resolve S T* ⟹ *no-step propagate S* ∧ *no-step conflict S*›
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-res*:
‹*resolve S T* ⟹ *cdcl-bnb-r-stgy S T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-res*:
‹*resolve*** *S T* ⟹ *cdcl-bnb-r-stgy*** *S T*›
⟨*proof*⟩

lemma *obacktrack-no-prop-confl*: ‹*enc-weight-opt.obacktrack S T* ⟹ *no-step propagate S* ∧ *no-step conflict S*›
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-bt*:
‹*enc-weight-opt.obacktrack S T* ⟹ *cdcl-bnb-r-stgy S T*›
⟨*proof*⟩

lemma *cdcl-dpll-bnb-r-stgy-cdcl-bnb-r-stgy*:
assumes ‹*cdcl-dpll-bnb-r-stgy S T*› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*cdcl-bnb-r-stgy*** *S T*›
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-cdcl-bnb-r*:
‹*cdcl-bnb-r-stgy S T* ⟹ *cdcl-bnb-r S T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r*:
  ‹*cdcl-bnb-r-stgy*** *S T* $\Longrightarrow$ *cdcl-bnb-r*** *S T*›
  ⟨*proof*⟩

context
  fixes *S* :: *'st*
  assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
begin
lemma *cdcl-dpll-bnb-r-stgy-all-struct-inv*:
  ‹*cdcl-dpll-bnb-r-stgy S T* $\Longrightarrow$
    cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) $\Longrightarrow$
    cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
  ⟨*proof*⟩

end

lemma *cdcl-bnb-r-stgy-cdcl-dpll-bnb-r-stgy*:
  ‹*cdcl-bnb-r-stgy S T* $\Longrightarrow$ ∃ *T*. *cdcl-dpll-bnb-r-stgy S T*›
  ⟨*proof*⟩

context
  fixes *S* :: *'st*
  assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-cdcl-bnb-r*:
  assumes ‹*cdcl-dpll-bnb-r-stgy*** *S T*› **and**
    ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
  shows ‹*cdcl-bnb-r-stgy*** *S T*›
  ⟨*proof*⟩

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-all-struct-inv*:
  ‹*cdcl-dpll-bnb-r-stgy*** *S T* $\Longrightarrow$
    cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) $\Longrightarrow$
    cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
  ⟨*proof*⟩

lemma *full-cdcl-dpll-bnb-r-stgy-full-cdcl-bnb-r-stgy*:
  assumes ‹*full cdcl-dpll-bnb-r-stgy S T*› **and**
    ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
  shows ‹*full cdcl-bnb-r-stgy S T*›
  ⟨*proof*⟩

end

lemma *replace-pos-neg-not-both-decided-highest-lvl*:
  assumes
    struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
    smaller-propa: ‹*no-smaller-propa S*› **and**
    smaller-confl: ‹*no-smaller-confl S*› **and**
    dec0: ‹*Pos* ($A^{\mapsto 0}$) ∈ *lits-of-l* (*trail S*)› **and**
    dec1: ‹*Pos* ($A^{\mapsto 1}$) ∈ *lits-of-l* (*trail S*)› **and**
    add: ‹*additional-constraints* ⊆# *init-clss S*› **and**
    [*simp*]: ‹*A* ∈ ΔΣ›

shows ‹*get-level* (*trail S*) (*Pos* ($A^{\mapsto 0}$)) = *backtrack-lvl S* ∧
get-level (*trail S*) (*Pos* ($A^{\mapsto 1}$)) = *backtrack-lvl S*›
⟨*proof*⟩

lemma *cdcl-dpll-bnb-r-stgy-clauses-mono*:
‹*cdcl-dpll-bnb-r-stgy S T* ⟹ *clauses S* ⊆# *clauses T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-clauses-mono*:
‹*cdcl-dpll-bnb-r-stgy*** *S T* ⟹ *clauses S* ⊆# *clauses T*›
⟨*proof*⟩

lemma *cdcl-dpll-bnb-r-stgy-init-clss-eq*:
‹*cdcl-dpll-bnb-r-stgy S T* ⟹ *init-clss S* = *init-clss T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-init-clss-eq*:
‹*cdcl-dpll-bnb-r-stgy*** *S T* ⟹ *init-clss S* = *init-clss T*›
⟨*proof*⟩

context
fixes *S* :: *'st* **and** *N* :: ‹*'v clauses*›
assumes *S*-Σ: ‹*init-clss S* = *penc N*›
begin

lemma *replacement-pos-neg-defined-same-lvl*:
assumes
struct: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
A: ‹*A* ∈ ΔΣ› **and**
lev: ‹*get-level* (*trail S*) (*Pos* (*replacement-pos A*)) < *backtrack-lvl S*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*Pos* (*replacement-pos A*) ∈ *lits-of-l* (*trail S*) ⟹
Neg (*replacement-neg A*) ∈ *lits-of-l* (*trail S*)›
⟨*proof*⟩

lemma *replacement-pos-neg-defined-same-lvl'*:
assumes
struct: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
A: ‹*A* ∈ ΔΣ› **and**
lev: ‹*get-level* (*trail S*) (*Pos* (*replacement-neg A*)) < *backtrack-lvl S*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*Pos* (*replacement-neg A*) ∈ *lits-of-l* (*trail S*) ⟹
Neg (*replacement-pos A*) ∈ *lits-of-l* (*trail S*)›
⟨*proof*⟩

end

definition *all-new-literals* :: ‹*'v list*› **where**

‹*all-new-literals* = (*SOME xs. mset xs* = *mset-set* (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ))›

lemma *set-all-new-literals*[*simp*]:
‹*set all-new-literals* = (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
⟨*proof*⟩

This function is basically resolving the clause with all the additional clauses $\{\#Neg\ (L^{\mapsto 1}),\ Neg\ (L^{\mapsto 0})\#\}$.

fun *resolve-with-all-new-literals* :: ‹'*v clause* ⇒ '*v list* ⇒ '*v clause*› **where**
‹*resolve-with-all-new-literals C* [] = *C*› |
‹*resolve-with-all-new-literals C* (*L* # *Ls*) =
remdups-mset (*resolve-with-all-new-literals* (*if Pos L* ∈# *C then add-mset* (*Neg* (*opposite-var L*)) (*removeAll-mset* (*Pos L*) *C*) *else C*) *Ls*)›

abbreviation *normalize2* **where**
‹*normalize2 C* ≡ *resolve-with-all-new-literals C all-new-literals*›

lemma *Neg-in-normalize2*[*simp*]: ‹*Neg L* ∈# *C* ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs*›
⟨*proof*⟩

lemma *Pos-in-normalize2D*[*dest*]: ‹*Pos L* ∈# *resolve-with-all-new-literals C xs* ⟹ *Pos L* ∈# *C*›
⟨*proof*⟩

lemma *opposite-var-involutive*[*simp*]:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *opposite-var* (*opposite-var L*) = *L*›
⟨*proof*⟩

lemma *Neg-in-resolve-with-all-new-literals-Pos-notin*:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *set xs* ⊆ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹
Pos (*opposite-var L*) ∉# *C* ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs* ⟷ *Neg L* ∈# *C*›
⟨*proof*⟩

lemma *Pos-in-normalize2-Neg-notin*[*simp*]:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹
Pos (*opposite-var L*) ∉# *C* ⟹ *Neg L* ∈# *normalize2 C* ⟷ *Neg L* ∈# *C*›
⟨*proof*⟩

lemma *all-negation-deleted*:
‹*L* ∈ *set all-new-literals* ⟹ *Pos L* ∉# *normalize2 C*›
⟨*proof*⟩

lemma *Pos-in-resolve-with-all-new-literals-iff-already-in-or-negation-in*:
‹*L* ∈ *set all-new-literals* ⟹*set xs* ⊆ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs*⟹
Neg L ∈# *C* ∨ *Pos* (*opposite-var L*) ∈# *C*›
⟨*proof*⟩

lemma *Pos-in-normalize2-iff-already-in-or-negation-in*:
‹*L* ∈ *set all-new-literals* ⟹ *Neg L* ∈# *normalize2 C* ⟹
Neg L ∈# *C* ∨ *Pos* (*opposite-var L*) ∈# *C*›
⟨*proof*⟩

This proof makes it hard to measure progress because I currently do not see a way to distinguish

between *add-mset* $(A^{\mapsto 1})$ C and *add-mset* $(A^{\mapsto 1})$ (*add-mset* $(A^{\mapsto 0})$ C).

lemma
  assumes
    ‹*enc-weight-opt.cdcl-bnb-stgy S T*› **and**
    struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
    dist: ‹*distinct-mset* (*normalize-clause* '# *learned-clss S*)› **and**
    smaller-propa: ‹*no-smaller-propa S*› **and**
    smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
  shows ‹*distinct-mset* (*remdups-mset* (*normalize2* '# *learned-clss T*))›
  ⟨*proof*⟩
 find-theorems *get-level Pos Neg*

end

end
theory *CDCL-W-Covering-Models*
  imports *CDCL-W-Optimal-Model*
begin

0.2 Covering Models

I am only interested in the extension of CDCL to find covering mdoels, not in the required subsequent extraction of the minimal covering models.

type-synonym *'v cov* = ‹*'v literal multiset multiset*›

lemma *true-clss-cls-in-susbsuming*:
  ‹$C' \subseteq\# C \Longrightarrow C' \in N \Longrightarrow N \models p\ C$›
  ⟨*proof*⟩

locale *covering-models* =
  fixes
    ϱ :: ‹*'v* ⇒ *bool*›
begin

definition *model-is-dominated* :: ‹*'v literal multiset* ⇒ *'v literal multiset* ⇒ *bool*› **where**
‹*model-is-dominated M M'* ⟷
  filter-mset (λ*L*. *is-pos L* ∧ ϱ (*atm-of L*)) *M* ⊆# *filter-mset* (λ*L*. *is-pos L* ∧ ϱ (*atm-of L*)) *M'*›

lemma *model-is-dominated-refl*: ‹*model-is-dominated I I*›
  ⟨*proof*⟩

lemma *model-is-dominated-trans*:
  ‹*model-is-dominated I J* ⟹ *model-is-dominated J K* ⟹ *model-is-dominated I K*›
  ⟨*proof*⟩

definition *is-dominating* :: ‹*'v literal multiset multiset* ⇒ *'v literal multiset* ⇒ *bool*› **where**
  ‹*is-dominating* $\mathcal{M}$ *I* ⟷ (∃ *M*∈#$\mathcal{M}$. ∃ *J*. *I* ⊆# *J* ∧ *model-is-dominated J M*)›

lemma
  is-dominating-in:
    ‹*I* ∈# $\mathcal{M}$ ⟹ *is-dominating* $\mathcal{M}$ *I*› **and**
  is-dominating-mono:
    ‹*is-dominating* $\mathcal{M}$ *I* ⟹ *set-mset* $\mathcal{M}$ ⊆ *set-mset* $\mathcal{M}'$ ⟹ *is-dominating* $\mathcal{M}'$ *I*› **and**

is-dominating-mono-model:
‹*is-dominating* $\mathcal{M}$ *I* $\Longrightarrow$ *I'* $\subseteq\#$ *I* $\Longrightarrow$ *is-dominating* $\mathcal{M}$ *I'*›
⟨*proof*⟩

lemma *is-dominating-add-mset*:
‹*is-dominating* (*add-mset x* $\mathcal{M}$) *I* $\longleftrightarrow$
is-dominating $\mathcal{M}$ *I* $\vee$ ($\exists$ *J*. *I* $\subseteq\#$ *J* $\wedge$ *model-is-dominated J x*)›
⟨*proof*⟩

definition *is-improving-int*
:: ‹('*v*, '*v clause*) *ann-lits* $\Rightarrow$ ('*v*, '*v clause*) *ann-lits* $\Rightarrow$ '*v clauses* $\Rightarrow$ '*v cov* $\Rightarrow$ *bool*›
where
‹*is-improving-int M M' N* $\mathcal{M}$ $\longleftrightarrow$
M = *M'* $\wedge$ ($\forall$ *I* $\in\#$ $\mathcal{M}$. $\neg$*model-is-dominated* (*lit-of* '$\#$ *mset M*) *I*) $\wedge$
total-over-m (*lits-of-l M*) (*set-mset N*) $\wedge$
lit-of '$\#$ *mset M* $\in$ *simple-clss* (*atms-of-mm N*) $\wedge$
lit-of '$\#$ *mset M* $\notin\#$ $\mathcal{M}$ $\wedge$
M $\models$*asm N* $\wedge$
no-dup M›

This criteria is a bit more general than Weidenbach's version.

abbreviation *conflicting-clauses-ent* **where**
‹*conflicting-clauses-ent N* $\mathcal{M}$ $\equiv$
{#*pNeg* {#*L* $\in\#$ *x*. ϱ (*atm-of L*)#}.
x $\in\#$ *filter-mset* (λ*x*. *is-dominating* $\mathcal{M}$ *x* $\wedge$ *atms-of x* = *atms-of-mm N*)
(*mset-set* (*simple-clss* (*atms-of-mm N*)))#}+ *N*›

definition *conflicting-clauses*
:: ‹'*v clauses* $\Rightarrow$ '*v cov* $\Rightarrow$ '*v clauses*›
where
‹*conflicting-clauses N* $\mathcal{M}$ =
{#*C* $\in\#$ *mset-set* (*simple-clss* (*atms-of-mm N*)).
conflicting-clauses-ent N $\mathcal{M}$ $\models$*pm C*#}›

lemma *conflicting-clauses-insert*:
assumes ‹*M* $\in$ *simple-clss* (*atms-of-mm N*)› **and** ‹*atms-of M* = *atms-of-mm N*›
shows ‹*pNeg M* $\in\#$ *conflicting-clauses N* (*add-mset M w*)›
⟨*proof*⟩

lemma *is-dominating-in-conflicting-clauses*:
assumes ‹*is-dominating* $\mathcal{M}$ *I*› **and**
atm: ‹*atms-of-s* (*set-mset I*) = *atms-of-mm N*› **and**
‹*set-mset I* $\models$*m N*› **and**
‹*consistent-interp* (*set-mset I*)› **and**
‹$\neg$*tautology I*› **and**
‹*distinct-mset I*›
shows
‹*pNeg I* $\in\#$ *conflicting-clauses N* $\mathcal{M}$›
⟨*proof*⟩

end

locale *conflict-driven-clause-learning*$_W$*-covering-models* =
conflict-driven-clause-learning$_W$
state-eq
state

— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting
— get state:
init-state +
covering-models ϱ
for
state-eq :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
state :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times 'v\ cov \times 'b$ **and**
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
learned-clss :: $'st \Rightarrow 'v\ clauses$ **and**
conflicting :: $'st \Rightarrow 'v\ clause\ option$ **and**

cons-trail :: $('v, 'v\ clause)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-learned-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
remove-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
update-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**
init-state :: $'v\ clauses \Rightarrow 'st$ **and**
ϱ :: $\langle 'v \Rightarrow bool \rangle$ +
fixes
update-additional-info :: $\langle 'v\ cov \times 'b \Rightarrow 'st \Rightarrow 'st \rangle$
assumes
update-additional-info:
$\langle state\ S = (M,\ N,\ U,\ C,\ \mathcal{M}) \Longrightarrow state\ (update\text{-}additional\text{-}info\ K'\ S) = (M,\ N,\ U,\ C,\ K') \rangle$ **and**
weight-init-state:
$\langle \bigwedge N :: 'v\ clauses.\ fst\ (additional\text{-}info\ (init\text{-}state\ N)) = \{\#\} \rangle$
begin

definition *update-weight-information* :: $\langle ('v, 'v\ clause)\ ann\text{-}lits \Rightarrow 'st \Rightarrow 'st \rangle$ **where**
$\langle update\text{-}weight\text{-}information\ M\ S =$
update-additional-info (*add-mset* (*lit-of* ‘# *mset M*) (*fst* (*additional-info S*)), *snd* (*additional-info S*)) *S*$\rangle$

lemma
trail-update-additional-info[*simp*]: $\langle trail\ (update\text{-}additional\text{-}info\ w\ S) = trail\ S \rangle$ **and**
init-clss-update-additional-info[*simp*]:
$\langle init\text{-}clss\ (update\text{-}additional\text{-}info\ w\ S) = init\text{-}clss\ S \rangle$ **and**
learned-clss-update-additional-info[*simp*]:
$\langle learned\text{-}clss\ (update\text{-}additional\text{-}info\ w\ S) = learned\text{-}clss\ S \rangle$ **and**
backtrack-lvl-update-additional-info[*simp*]:
$\langle backtrack\text{-}lvl\ (update\text{-}additional\text{-}info\ w\ S) = backtrack\text{-}lvl\ S \rangle$ **and**
conflicting-update-additional-info[*simp*]:
$\langle conflicting\ (update\text{-}additional\text{-}info\ w\ S) = conflicting\ S \rangle$ **and**
clauses-update-additional-info[*simp*]:
$\langle clauses\ (update\text{-}additional\text{-}info\ w\ S) = clauses\ S \rangle$
⟨*proof*⟩

lemma
trail-update-weight-information[*simp*]:
$\langle trail\ (update\text{-}weight\text{-}information\ w\ S) = trail\ S \rangle$ **and**

init-clss-update-weight-information[*simp*]:
‹*init-clss* (*update-weight-information w S*) = *init-clss S*› **and**
learned-clss-update-weight-information[*simp*]:
‹*learned-clss* (*update-weight-information w S*) = *learned-clss S*› **and**
backtrack-lvl-update-weight-information[*simp*]:
‹*backtrack-lvl* (*update-weight-information w S*) = *backtrack-lvl S*› **and**
conflicting-update-weight-information[*simp*]:
‹*conflicting* (*update-weight-information w S*) = *conflicting S*› **and**
clauses-update-weight-information[*simp*]:
‹*clauses* (*update-weight-information w S*) = *clauses S*›
⟨*proof*⟩

definition *covering* :: ‹'*st* ⇒ '*v cov*› **where**
‹*covering S* = *fst* (*additional-info S*)›

lemma
additional-info-update-additional-info[*simp*]:
additional-info (*update-additional-info w S*) = *w*
⟨*proof*⟩

lemma
covering-cons-trail2[*simp*]: ‹*covering* (*cons-trail L S*) = *covering S*› **and**
clss-tl-trail2[*simp*]: *covering* (*tl-trail S*) = *covering S* **and**
covering-add-learned-cls-unfolded:
covering (*add-learned-cls U S*) = *covering S*
and
covering-update-conflicting2[*simp*]: *covering* (*update-conflicting D S*) = *covering S* **and**
covering-remove-cls2[*simp*]:
covering (*remove-cls C S*) = *covering S* **and**
covering-add-learned-cls2[*simp*]:
covering (*add-learned-cls C S*) = *covering S* **and**
covering-update-covering-information2[*simp*]:
covering (*update-weight-information M S*) = *add-mset* (*lit-of* ‘# *mset M*) (*covering S*)
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-cost*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**

init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *covering* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
⟨*proof*⟩

lemma *state-additional-info2'*:
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *covering S*, *additional-info' S*)›
⟨*proof*⟩

lemma *state-update-weight-information*:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w'*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w'*, *other*)›
⟨*proof*⟩

lemma *conflicting-clss-incl-init-clss*:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

lemma *conflict-clss-update-weight-no-alien*:
‹*atms-of-mm* (*conflicting-clss* (*update-weight-information M S*))
⊆ *atms-of-mm* (*init-clss S*)›
⟨*proof*⟩

lemma *distinct-mset-mset-conflicting-clss2*: ‹*distinct-mset-mset* (*conflicting-clss S*)›
⟨*proof*⟩

lemma *total-over-m-atms-incl*:
assumes ‹*total-over-m M* (*set-mset N*)›
shows
‹*x* ∈ *atms-of-mm N* ⟹ *x* ∈ *atms-of-s M*›
⟨*proof*⟩

lemma *negate-ann-lits-simple-clss-iff*[*iff*]:
‹*negate-ann-lits M* ∈ *simple-clss N* ⟷ *lit-of* `# *mset M* ∈ *simple-clss N*›
⟨*proof*⟩

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*is-improving M M' S*›
shows ‹*negate-ann-lits M'* ∈# *conflicting-clss* (*update-weight-information M' S*)›
⟨*proof*⟩

lemma *is-improving-conflicting-clss-update-weight-information*: ‹*is-improving M M' S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M' S*)›

⟨*proof*⟩

sublocale *state*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *state*$_W$*-no-state* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-cost*$_W$*-ops*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**

add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *covering* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
⟨*proof*⟩

definition *covering-simple-clss* **where**
‹*covering-simple-clss N S* ⟷ (*set-mset* (*covering S*) ⊆ *simple-clss* (*atms-of-mm N*)) ∧
distinct-mset (*covering S*) ∧
(∀ *M* ∈# *covering S*. *total-over-m* (*set-mset M*) (*set-mset N*))›

lemma [*simp*]: ‹*covering* (*init-state N*) = {#}›
⟨*proof*⟩

lemma ‹*covering-simple-clss N* (*init-state N*)›
⟨*proof*⟩

lemma *cdcl-bnb-covering-simple-clss*:
‹*cdcl-bnb S T* ⟹ *init-clss S* = *N* ⟹ *covering-simple-clss N S* ⟹ *covering-simple-clss N T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-covering-simple-clss*:
‹*cdcl-bnb*** *S T* ⟹ *init-clss S* = *N* ⟹ *covering-simple-clss N S* ⟹ *covering-simple-clss N T*›
⟨*proof*⟩

lemma *wf-cdcl-bnb-fixed*:
‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*
∧ *covering-simple-clss N S* ∧ *init-clss S* = *N*}›
⟨*proof*⟩

lemma *can-always-improve*:
assumes
ent: ‹*trail S* ⊨*asm clauses S*› **and**
total: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))› **and**
n-s: ‹*no-step conflict-opt S*› **and**
confl: ‹*conflicting S* = *None*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*Ex* (*improvep S*)›
⟨*proof*⟩

lemma *exists-model-with-true-lit-entails-conflicting*:
assumes
L-I: ‹*Pos L* ∈ *I*› **and**
L: ‹ϱ *L*› **and**
L-in: ‹*L* ∈ *atms-of-mm* (*init-clss S*)› **and**
ent: ‹*I* ⊨*m init-clss S*› **and**
cons: ‹*consistent-interp I*› **and**
total: ‹*total-over-m I* (*set-mset N*)› **and**
no-L: ‹¬(∃ *J*∈# *covering S*. *Pos L* ∈# *J*)› **and**
cov: ‹*covering-simple-clss N S*› **and**
NS: ‹*atms-of-mm N* = *atms-of-mm* (*init-clss S*)›

shows ‹$I \models m$ *conflicting-clss* S› **and**
‹$I \models m$ *CDCL-W-Abstract-State.init-clss* (*abs-state* S)›
⟨*proof*⟩

lemma *exists-model-with-true-lit-still-model*:
assumes
L-I: ‹*Pos* $L \in I$› **and**
L: ‹ϱ L› **and**
L-in: ‹$L \in$ *atms-of-mm* (*init-clss* S)› **and**
ent: ‹$I \models m$ *init-clss* S› **and**
cons: ‹*consistent-interp* I› **and**
total: ‹*total-over-m* I (*set-mset* N)› **and**
cdcl: ‹*cdcl-bnb* S T› **and**
no-L-T: ‹$\neg(\exists J \in\# \textit{covering } T.\ \textit{Pos } L \in\# J)$› **and**
cov: ‹*covering-simple-clss* N S› **and**
NS: ‹*atms-of-mm* N = *atms-of-mm* (*init-clss* S)›
shows ‹$I \models m$ *CDCL-W-Abstract-State.init-clss* (*abs-state* T)›
⟨*proof*⟩

lemma *rtranclp-exists-model-with-true-lit-still-model*:
assumes
L-I: ‹*Pos* $L \in I$› **and**
L: ‹ϱ L› **and**
L-in: ‹$L \in$ *atms-of-mm* (*init-clss* S)› **and**
ent: ‹$I \models m$ *init-clss* S› **and**
cons: ‹*consistent-interp* I› **and**
total: ‹*total-over-m* I (*set-mset* N)› **and**
cdcl: ‹*cdcl-bnb*** S T› **and**
cov: ‹*covering-simple-clss* N S› **and**
‹N = *init-clss* S›
shows ‹$I \models m$ *CDCL-W-Abstract-State.init-clss* (*abs-state* T) $\vee$ $(\exists J \in\# \textit{covering } T.\ \textit{Pos } L \in\# J)$›
⟨*proof*⟩

lemma *is-dominating-nil*[*simp*]: ‹¬*is-dominating* {#} x›
⟨*proof*⟩

lemma *atms-of-conflicting-clss-init-state*:
‹*atms-of-mm* (*conflicting-clss* (*init-state* N)) $\subseteq$ *atms-of-mm* N›
⟨*proof*⟩

lemma *no-step-cdcl-bnb-stgy-empty-conflict2*:
assumes
n-s: ‹*no-step cdcl-bnb* S› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state* S)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv* S›
shows ‹*conflicting* S = *Some* {#}›
⟨*proof*⟩

theorem *cdclcm-correctness*:
assumes
full: ‹*full cdcl-bnb-stgy* (*init-state* N) T› **and**
dist: ‹*distinct-mset-mset* N›
shows
‹*Pos* $L \in I \Longrightarrow \varrho\ L \Longrightarrow L \in$ *atms-of-mm* $N \Longrightarrow$ *total-over-m* I (*set-mset* N) $\Longrightarrow$ *consistent-interp* $I \Longrightarrow I \models m\ N \Longrightarrow$

$\exists J \in\# \mathit{covering}\ T.\ \mathit{Pos}\ L \in\# J\rangle$
⟨*proof*⟩

end

Now we instantiate the previous with λ-. *True*: This means that we aim at making all variables that appears at least ones true.

global-interpretation *cover-all-vars*: *covering-models* ⟨λ-. *True*⟩
⟨*proof*⟩

context *conflict-driven-clause-learning*$_W$*-covering-models*
begin

interpretation *cover-all-vars*: *conflict-driven-clause-learning*$_W$*-covering-models* **where**
ϱ = ⟨λ-::'*v*. *True*⟩ **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

lemma
⟨*cover-all-vars.model-is-dominated M M'* $\longleftrightarrow$
filter-mset (λ*L*. *is-pos L*) *M* $\subseteq\#$ *filter-mset* (λ*L*. *is-pos L*) *M'*⟩
⟨*proof*⟩

lemma
⟨*cover-all-vars.conflicting-clauses N* $\mathcal{M}$ =
{# *C* $\in\#$ (*mset-set* (*simple-clss* (*atms-of-mm N*))).
(*pNeg* ‘
{*a*. *a* $\in\#$ *mset-set* (*simple-clss* (*atms-of-mm N*)) $\wedge$
($\exists$ *M*$\in\#\mathcal{M}$. $\exists$ *J*. *a* $\subseteq\#$ *J* $\wedge$ *cover-all-vars.model-is-dominated J M*) $\wedge$
atms-of a = *atms-of-mm N*} $\cup$
set-mset N) $\models p$ *C*#}⟩
⟨*proof*⟩

theorem *cdclcm-correctness-all-vars*:
assumes
full: ⟨*full cover-all-vars.cdcl-bnb-stgy* (*init-state N*) *T*⟩ **and**
dist: ⟨*distinct-mset-mset N*⟩
shows
⟨*Pos L* $\in$ *I* $\Longrightarrow$ *L* $\in$ *atms-of-mm N* $\Longrightarrow$ *total-over-m I* (*set-mset N*) $\Longrightarrow$ *consistent-interp I* $\Longrightarrow$ *I* $\models m$ *N* $\Longrightarrow$
$\exists J \in\# \mathit{covering}\ T.\ \mathit{Pos}\ L \in\# J\rangle$
⟨*proof*⟩

end

end