Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

July 17, 2023

Contents

theory *CDCL-W-BnB*
imports *CDCL.CDCL-W-Abstract-State*
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. $(M;N;U;k;\top;O) \Rightarrow^{\textit{Propagate}} (ML^{C\vee L};N;U;k;\top;O)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, L *is undefined in* M.

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

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

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

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

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

$(M_1K^{i+1}M_2;N;U;k;D \vee L;O) \Rightarrow^{\textit{Backtrack}} (M_1L^{D\vee L};N;U \cup \{D \vee L\};i;\top;O)$
provided L *is of level* k *and* D *is of level* i.

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

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

$$\mathrm{cost} : \begin{cases} P \rightarrow 3 \\ \neg P \rightarrow 1 \\ Q \rightarrow 1 \\ \neg Q \rightarrow 1 \end{cases}$$

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

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-bnb*$_W$*-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*)›

While it would more be natural to add an sublocale with the extended version clause set, this actually causes a loop in the hierarchy structure (although with different parameters). Therefore, adding theorems (e.g. defining an inductive predicate) causes a loop.

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-bnb*$_W$*-ops* =
conflict-driven-clause-learning-with-adding-init-clause-bnb$_W$*-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

Conversion to CDCL **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*⟩

Overall simplification on states **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*⟩

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*: ‹$S \sim T \Longrightarrow$ *abs-state S* = *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 *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 *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 *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*]:
‹$S \sim T \Longrightarrow$ *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*⟩

CDCL with branch-and-bound

inductive *conflict-opt* :: ‹$'st \Rightarrow 'st \Rightarrow$ *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*›
‹$T \sim$ *update-conflicting* (*Some* (*negate-ann-lits* (*trail* *S*))) *S*›

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

inductive *improvep* :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› **for** *S* :: $'st$ **where**
improve-rule:
‹*improvep* *S* *T*›
if
‹*is-improving* (*trail* *S*) M' *S*› **and**
‹*conflicting* *S* = *None*› **and**
‹$T \sim$ *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 *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' $\Longrightarrow$ *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' $\Longrightarrow$ $ocdcl_W$*-o* S S'› |
bj: ‹*cdcl-bnb-bj* S S' $\Longrightarrow$ $ocdcl_W$*-o* S S'›

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

inductive *cdcl-bnb-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** S :: *'st* **where**
cdcl-bnb-conflict: ‹*conflict* S S' $\Longrightarrow$ *cdcl-bnb-stgy* S S'› |
cdcl-bnb-propagate: ‹*propagate* S S' $\Longrightarrow$ *cdcl-bnb-stgy* S S'› |
cdcl-bnb-improve: ‹*improvep* S S' $\Longrightarrow$ *cdcl-bnb-stgy* S S'› |
cdcl-bnb-conflict-opt: ‹*conflict-opt* S S' $\Longrightarrow$ *cdcl-bnb-stgy* S S'› |
cdcl-bnb-other': ‹$ocdcl_W$*-o* S S' $\Longrightarrow$ *no-confl-prop-impr* S $\Longrightarrow$ *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: $\bigwedge L\ T.$ *conflicting* S = *None* $\Longrightarrow$ *undefined-lit* (*trail* S) L $\Longrightarrow$
      atm-of $L \in$ *atms-of-mm* (*init-clss* S) $\Longrightarrow$
      $T \sim$ *cons-trail* (*Decided* L) S $\Longrightarrow$
      P S T **and**
    skipH: $\bigwedge L\ C'\ M\ E\ T.$
      trail S = *Propagated* L C' # M $\Longrightarrow$
      conflicting S = *Some* E $\Longrightarrow$
      $-L \notin\#\ E \Longrightarrow E \neq \{\#\} \Longrightarrow$
      $T \sim$ *tl-trail* S $\Longrightarrow$
      P S T **and**
    resolveH: $\bigwedge L\ E\ M\ D\ T.$
      trail S = *Propagated* L E # M $\Longrightarrow$
      $L \in\#\ E \Longrightarrow$
      hd-trail S = *Propagated* L E $\Longrightarrow$
      conflicting S = *Some* D $\Longrightarrow$
      $-L \in\#\ D \Longrightarrow$
      get-maximum-level (*trail* S) ((*remove1-mset* ($-L$) D)) = *backtrack-lvl* S $\Longrightarrow$
      $T \sim$ *update-conflicting*
        (*Some* (*resolve-cls* L D E)) (*tl-trail* S) $\Longrightarrow$
      P S T **and**
    backtrackH: $\bigwedge L\ D\ K\ i\ M1\ M2\ T\ D'.$
      conflicting S = *Some* (*add-mset* L D) $\Longrightarrow$
      (*Decided* K # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail* S)) $\Longrightarrow$
      get-level (*trail* S) L = *backtrack-lvl* S $\Longrightarrow$
      get-level (*trail* S) L = *get-maximum-level* (*trail* S) (*add-mset* L D') $\Longrightarrow$
      get-maximum-level (*trail* S) $D' \equiv i \Longrightarrow$
      get-level (*trail* S) K = $i+1$ $\Longrightarrow$
      $D' \subseteq\#\ D \Longrightarrow$
      clauses S + *conflicting-clss* S $\models$*pm* *add-mset* L D' $\Longrightarrow$
      $T \sim$ *cons-trail* (*Propagated* L (*add-mset* L D'))
          (*reduce-trail-to* *M1*

(*add-learned-cls* (*add-mset* L D')
(*update-conflicting None* S))) $\Longrightarrow$
P S T
shows ‹P S T›
⟨*proof*⟩

lemma *obacktrack-backtrackg*: ‹*obacktrack* S T $\Longrightarrow$ *backtrackg* S T›
⟨*proof*⟩

Pluging into normal CDCL

lemma *cdcl-bnb-no-more-init-clss*:
‹*cdcl-bnb* S S' $\Longrightarrow$ *init-clss* S = *init-clss* S'›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-no-more-init-clss*:
‹*cdcl-bnb*** S S' $\Longrightarrow$ *init-clss* S = *init-clss* S'›
⟨*proof*⟩

lemma *conflict-opt-conflict*:
‹*conflict-opt* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.conflict* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *conflict-conflict*:
‹*conflict* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.conflict* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *propagate-propagate*:
‹*propagate* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.propagate* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *decide-decide*:
‹*decide* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.decide* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *skip-skip*:
‹*skip* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.skip* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *resolve-resolve*:
‹*resolve* S T $\Longrightarrow$ $cdcl_W$*-restart-mset.resolve* (*abs-state* S) (*abs-state* T)›
⟨*proof*⟩

lemma *backtrack-backtrack*:
‹*obacktrack* S T $\Longrightarrow$ $cdcl_W$*-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
‹$ocdcl_W$*-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 *$cdcl_W$-o-$cdcl_W$-o*:
‹*$ocdcl_W$-o* $S\ S' \Longrightarrow$ *$cdcl_W$-restart-mset.$cdcl_W$-o* (*abs-state* S) (*abs-state* S')›
⟨*proof*⟩

lemma *cdcl-bnb-stgy-all-struct-inv*:
assumes ‹*cdcl-bnb* $S\ T$› **and** ‹*$cdcl_W$-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 *rtranclp-cdcl-bnb-stgy-all-struct-inv*:
assumes ‹*cdcl-bnb*** $S\ T$› **and** ‹*$cdcl_W$-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 *cdcl-bnb-stgy-$cdcl_W$-or-improve*:
assumes ‹*cdcl-bnb* $S\ T$› **and** ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* S)›
shows ‹($\lambda S\ T.$ *$cdcl_W$-restart-mset.$cdcl_W$* (*abs-state* S) (*abs-state* T) $\vee$ *improvep* $S\ T$) $S\ T$›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-stgy-$cdcl_W$-or-improve*:
assumes ‹*rtranclp cdcl-bnb* $S\ T$› **and** ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* S)›
shows ‹($\lambda S\ T.$ *$cdcl_W$-restart-mset.$cdcl_W$* (*abs-state* S) (*abs-state* T) $\vee$ *improvep* $S\ T$)** $S\ T$›
⟨*proof*⟩

lemma *eq-diff-subset-iff*: ‹$A = B + (A - B) \longleftrightarrow B \subseteq\# A$›
⟨*proof*⟩

lemma *cdcl-bnb-conflicting-clss-mono*:
‹*cdcl-bnb* $S\ T \Longrightarrow$ *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* S) $\Longrightarrow$
conflicting-clss $S \subseteq\#$ *conflicting-clss* T›
⟨*proof*⟩

lemma *cdcl-or-improve-cdclD*:
assumes ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* S)› **and**
‹*cdcl-bnb* $S\ T$›
shows ‹$\exists N.$
$cdcl_W$-restart-mset.$cdcl_W$** (*trail* S, *init-clss* $S + N$, *learned-clss* S, *conflicting* S) (*abs-state* T) $\wedge$
CDCL-W-Abstract-State.init-clss (*abs-state* T) = *init-clss* $S + N$›
⟨*proof*⟩

lemma *rtranclp-cdcl-or-improve-cdclD*:
assumes ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* S)› **and**
‹*cdcl-bnb*** $S\ T$›
shows ‹$\exists N.$
$cdcl_W$-restart-mset.$cdcl_W$** (*trail* S, *init-clss* $S + N$, *learned-clss* S, *conflicting* S) (*abs-state* T) $\wedge$
CDCL-W-Abstract-State.init-clss (*abs-state* T) = *init-clss* $S + N$›
⟨*proof*⟩

definition *cdcl-bnb-struct-invs* :: ‹$'st \Rightarrow$ *bool*› **where**
‹*cdcl-bnb-struct-invs* $S \longleftrightarrow$

atms-of-mm (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)›

lemma *cdcl-bnb-cdcl-bnb-struct-invs*:
‹*cdcl-bnb S T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *cdcl-bnb-struct-invs T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-cdcl-bnb-struct-invs*:
‹*cdcl-bnb*** *S T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *cdcl-bnb-struct-invs T*›
⟨*proof*⟩

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

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

lemma *skip-conflict-is-false-with-level*:
assumes ‹*skip 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 *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 $cdcl_W$*-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* ∼ *S'*› **and**
    TT': ‹*T* ∼ *T'*›
  shows ‹*obacktrack S' T'*›
⟨*proof*⟩

lemma *o*$cdcl_W$*-o-no-smaller-confl-inv*:
  fixes *S S'* :: ‹*'st*›
  assumes
    ‹*o*$cdcl_W$*-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 *o*$cdcl_W$*-o-conflict-is-false-with-level-inv*:
  assumes
    ‹*o*$cdcl_W$*-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* ⟷

$(M' \neq [] \wedge hd\ M' = Decided\ L \wedge MM = tl\ M'\ @\ Decided\ K\ \#\ M) \vee$
$(M' = [] \wedge L = K \wedge MM = M)$›
⟨*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 ∧ *no-smaller-confl* S›

lemma *cdcl-bnb-stgy-invD*:
shows ‹*cdcl-bnb-stgy-inv* $S \longleftrightarrow$ $cdcl_W$*-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 *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*) $\Longrightarrow$ *distinct-cdcl$_W$-state S*›
⟨*proof*⟩

lemma *obacktrack-imp-backtrack*:
‹*obacktrack S T* $\Longrightarrow$ *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* $\Longrightarrow$ *Ex* (*obacktrack S*)›
⟨*proof*⟩

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

lemma *ocdcl$_W$-o-same-weight*: ‹*ocdcl$_W$-o S U* $\Longrightarrow$ *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*: ‹$\bigwedge$*S T*. *improvep S T* $\Longrightarrow$ *init-clss S* = *N* $\Longrightarrow$ (ν (*weight T*), ν (*weight S*)) $\in$ *R*›
and
wf-R: ‹*wf R*›
shows ‹*wf* {(*T*, *S*). *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\wedge$ *cdcl-bnb S T* $\wedge$
init-clss S = *N*}›
(**is** ‹*wf ?A*›)
⟨*proof*⟩

corollary *wf-cdcl-bnb-fixed-iff*:
shows ‹($\forall$*N*. *wf* {(*T*, *S*). *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\wedge$ *cdcl-bnb S T*
$\wedge$ *init-clss S* = *N*}) $\longleftrightarrow$
wf {(*T*, *S*). *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\wedge$ *cdcl-bnb S T*}›
(**is** ‹($\forall$*N*. *wf* (*?A N*)) $\longleftrightarrow$ *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*: ‹$\bigwedge$*S T*. *improvep S T* $\Longrightarrow$ *P S* $\Longrightarrow$ *init-clss S* = *N* $\Longrightarrow$ (ν (*weight T*), ν (*weight S*)) $\in$ *R*› **and**
wf-R: ‹*wf R*› **and**
‹$\bigwedge$*S T*. *cdcl-bnb S T* $\Longrightarrow$ *P S* $\Longrightarrow$ *init-clss S* = *N* $\Longrightarrow$ *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\Longrightarrow$ *P T*›
shows ‹*wf* {(*T*, *S*). *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\wedge$ *cdcl-bnb S T* $\wedge$ *P S* $\wedge$
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 ‹$cdcl\text{-}bnb^{**}$ *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 ‹$cdcl\text{-}bnb\text{-}stgy^{**}$ *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*⟩

lemma *cdcl-bnb-pow2-n-learned-clauses*:
assumes ‹*distinct-mset-mset N*›
‹$cdcl\text{-}bnb^{**}$ (*init-state N*) *T*›
shows ‹*size* (*learned-clss T*) ≤ *2* ^ (*card* (*atms-of-mm N*))›
⟨*proof*⟩
end

end
theory *CDCL-W-Optimal-Model*
imports *CDCL-W-BnB HOL−Library.Extended-Nat*
begin

OCDCL

The following datatype is equivalent to *'a option*. However, 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

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*⟩

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 literal*, *'v literal*, *'b*) *annotated-lits* ⇒ (*'v literal*, *'v literal*, *'b*) *annotated-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*⟩

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 *conflicting-clss-incl-init-clauses*:
‹*atms-of-mm* (*conflicting-clauses N w*) ⊆ *atms-of-mm* (*N*)›
⟨*proof*⟩

lemma *distinct-mset-mset-conflicting-clss2*: ‹*distinct-mset-mset* (*conflicting-clauses N w*)›
⟨*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-int M M′ N w* ⟹

conflicting-clauses N w ⊆# *conflicting-clauses N* (*Some* (*lit-of* ‘# *mset M′*))›
⟨*proof*⟩

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*is-improving-int M M′ N w*›
shows ‹*negate-ann-lits M′* ∈# *conflicting-clauses N* (*Some* (*lit-of* ‘# *mset M′*))›
⟨*proof*⟩

lemma *atms-of-init-clss-conflicting-clauses′*[*simp*]:
‹*atms-of-mm N* ∪ *atms-of-mm* (*conflicting-clauses N S*) = *atms-of-mm N*›
⟨*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* (*Some* (*lit-of* ‘# *mset M′′*))›
⟨*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* :: ‹'*st* ⇒ '*v clause option*› **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*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-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 *atms-of-init-clss-conflicting-clauses*[*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*⟩

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*⟩

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-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 *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[*simp*]: ‹*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-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* (*init-clss S*)› **and**
le: ‹*Found* (ϱ *I*) < ϱ' (*Some* M')› **and**
‹*is-improving M* M'' *S*› **and**
‹*N* = *init-clss S*›
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*:
‹$cdcl\text{-}bnb^{**}$ *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*)› (**is** ‹*?B* ⟹ *?A*›) **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*⟩

end

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

Alternative versions

We instantiate our more general rules with exactly the rule from Christoph's OCDCL with either versions of improve.

Weights

This one is the version of the weight functions used by Christoph Weidenbach. However, we have decided to not instantiate tho calculus with this weight function, because it only a slight restriction.

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

Calculus with simple Improve rule

context *conflict-driven-clause-learning*$_W$*-optimal-weight*
begin

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

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

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

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

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

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

lemma $ocdcl_w$*-stgy-cdcl-bnb-stgy*:
  assumes ‹$ocdcl_w$*-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-*$ocdcl_w$*-stgy-rtranclp-cdcl-bnb-stgy*:
  assumes ‹$ocdcl_w$*-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-*$ocdcl_w$*-no-step-cdcl-bnb*:
  assumes ‹*no-step* $ocdcl_w$ *S*› **and**
    inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
  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* $\Longrightarrow$ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* $\neq$ *None* $\Longrightarrow$ *model-on* (*set-mset* (*the* (*weight T*))) *N* $\wedge$ *set-mset* (*the* (*weight T*)) $\models$*sm N* $\wedge$
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I* $\Longrightarrow$ *consistent-interp* (*set-mset I*) $\Longrightarrow$ *atms-of I* = *atms-of-mm N* $\Longrightarrow$
set-mset I $\models$*sm N* $\Longrightarrow$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight T*)›
⟨*proof*⟩

lemma *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 *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*)
$\wedge$ *ocdcl$_w$-p S T*}›
⟨*proof*⟩

end

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

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 *Neg-in-lits-of-l-definedD*:
‹*Neg A* $\in$ *lits-of-l M* $\Longrightarrow$ *defined-lit M* (*Pos A*)›
⟨*proof*⟩

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
- ΔΣ 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-ops* =
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* = $\sum_{\#}$(*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* $\Delta\Sigma$› **and** ‹$\Delta\Sigma \subseteq$ *atms-of-mm N*›
shows ‹*card* (*atms-of-mm* (*penc N*)) $\leq$ *card* (*atms-of-mm N* $-$ $\Delta\Sigma$) + *2* $*$ *card* $\Delta\Sigma$› (**is** ‹*?A* $\leq$ *?B*›)
⟨*proof*⟩

definition *postp* :: ‹*'v partial-interp* $\Rightarrow$ *'v partial-interp*› **where**
‹*postp I* =
{$A \in I$. *atm-of A* $\notin \Delta\Sigma \wedge$ *atm-of A* $\in \Sigma$} $\cup$ *Pos* ‘ {*A*. $A \in \Delta\Sigma \wedge$ *Pos* (*replacement-pos A*) $\in I$}
$\cup$ *Neg* ‘ {*A*. $A \in \Delta\Sigma \wedge$ *Pos* (*replacement-neg A*) $\in I \wedge$ *Pos* (*replacement-pos A*) $\notin I$}›

lemma *preprocess-clss-model-additional-variables2*:
assumes
‹*atm-of A* $\in \Sigma - \Delta\Sigma$›
shows
‹$A \in$ *postp I* $\longleftrightarrow A \in I$› (**is** *?A*)
⟨*proof*⟩

lemma *encode-clause-iff*:
assumes
‹$\bigwedge A$. $A \in \Delta\Sigma \Longrightarrow$ *Pos A* $\in I \longleftrightarrow$ *Pos* (*replacement-pos A*) $\in I$›
‹$\bigwedge A$. $A \in \Delta\Sigma \Longrightarrow$ *Neg A* $\in I \longleftrightarrow$ *Pos* (*replacement-neg A*) $\in I$›
shows ‹$I \models$ *encode-clause C* $\longleftrightarrow I \models C$›
⟨*proof*⟩

lemma *encode-clauses-iff*:
assumes
‹$\bigwedge A$. $A \in \Delta\Sigma \Longrightarrow$ *Pos A* $\in I \longleftrightarrow$ *Pos* (*replacement-pos A*) $\in I$›
‹$\bigwedge A$. $A \in \Delta\Sigma \Longrightarrow$ *Neg A* $\in I \longleftrightarrow$ *Pos* (*replacement-neg A*) $\in I$›
shows ‹$I \models m$ *encode-clauses C* $\longleftrightarrow I \models m$ *C*›
⟨*proof*⟩

definition Σ_{add} **where**
‹Σ_{add} = *replacement-pos* ‘ $\Delta\Sigma \cup$ *replacement-neg* ‘ $\Delta\Sigma$›

definition *upostp* :: ‹*'v partial-interp* $\Rightarrow$ *'v partial-interp*› **where**
‹*upostp I* =
Neg ‘ {$A \in \Sigma$. $A \notin \Delta\Sigma \wedge$ *Pos A* $\notin I \wedge$ *Neg A* $\notin I$}
$\cup$ {$A \in I$. *atm-of A* $\in \Sigma \wedge$ *atm-of A* $\notin \Delta\Sigma$}
$\cup$ *Pos* ‘ *replacement-pos* ‘ {$A \in \Delta\Sigma$. *Pos A* $\in I$}
$\cup$ *Neg* ‘ *replacement-pos* ‘ {$A \in \Delta\Sigma$. *Pos A* $\notin I$}
$\cup$ *Pos* ‘ *replacement-neg* ‘ {$A \in \Delta\Sigma$. *Neg A* $\in I$}
$\cup$ *Neg* ‘ *replacement-neg* ‘ {$A \in \Delta\Sigma$. *Neg A* $\notin I$}›

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

end

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 +
optimal-encoding-opt-ops Σ $\Delta\Sigma$ *new-vars*
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**
update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*› **and**
Σ $\Delta\Sigma$ :: ‹′*v set*› **and**
ϱ :: ‹′*v clause* ⇒ ′*a* :: {*linorder*}› **and**
new-vars :: ‹′*v* ⇒ ′*v* × ′*v*›
begin

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* ∈ Σ − ΔΣ› |
odecide-replacement-pos: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-pos L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-pos L*))) *S*› **and**
‹*L* ∈ ΔΣ› |
odecide-replacement-neg: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-neg L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-neg L*))) *S*› **and**
‹*L* ∈ ΔΣ›

inductive-cases *odecideE*: ‹*odecide S T*›

definition *no-new-lonely-clause* :: ‹*'v clause* ⇒ *bool*› **where**
‹*no-new-lonely-clause C* ⟷
(∀ *L* ∈ ΔΣ. *L* ∈ *atms-of C* ⟶
Neg (*replacement-pos L*) ∈# *C* ∨ *Neg* (*replacement-neg L*) ∈# *C* ∨ *C* ∈# *additional-constraint L*)›

definition *lonely-weighted-lit-decided* **where**
‹*lonely-weighted-lit-decided S* ⟷
(∀ *L* ∈ ΔΣ. *Decided* (*Pos L*) ∉ *set* (*trail S*) ∧ *Decided* (*Neg L*) ∉ *set* (*trail S*))›

end

locale *optimal-encoding-ops* = *optimal-encoding-opt-ops*
Σ ΔΣ
new-vars +
ocdcl-weight ϱ
for
Σ ΔΣ :: ‹*'v set*› **and**
new-vars :: ‹*'v* ⇒ *'v* × *'v*› **and**
ϱ :: ‹*'v clause* ⇒ *'a* :: {*linorder*}› +
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*:
‹$atm\text{-}of$ ‘ $I \subseteq \Sigma - \Delta\Sigma \Longrightarrow$ $consistent\text{-}interp\ I \longleftrightarrow consistent\text{-}interp\ (postp\ I)$›
⟨*proof*⟩

lemma *new-vars-different-iff*[*simp*]:
‹$A \neq x^{\mapsto 1}$›
‹$A \neq x^{\mapsto 0}$›
‹$x^{\mapsto 1} \neq A$›
‹$x^{\mapsto 0} \neq A$›
‹$A^{\mapsto 0} \neq x^{\mapsto 1}$›
‹$A^{\mapsto 1} \neq x^{\mapsto 0}$›
‹$A^{\mapsto 0} = x^{\mapsto 0} \longleftrightarrow A = x$›
‹$A^{\mapsto 1} = x^{\mapsto 1} \longleftrightarrow A = x$›
‹$(A^{\mapsto 1}) \notin \Sigma$›
‹$(A^{\mapsto 0}) \notin \Sigma$›
‹$(A^{\mapsto 1}) \notin \Delta\Sigma$›
‹$(A^{\mapsto 0}) \notin \Delta\Sigma$›**if** ‹$A \in \Delta\Sigma$› ‹$x \in \Delta\Sigma$› **for** A x
⟨*proof*⟩

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

lemma *atm-of-upostp-subset2*:
‹$atm\text{-}of$ ‘ $I \subseteq \Sigma \Longrightarrow replacement\text{-}pos$ ‘ $\Delta\Sigma\ \cup$
$replacement\text{-}neg$ ‘ $\Delta\Sigma \cup (\Sigma - \Delta\Sigma) \subseteq atm\text{-}of$ ‘ $(upostp\ I)$›
⟨*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: ‹$atms\text{-}of\text{-}mm\ N = \Sigma$› **and**
sat: ‹$I \models sm\ N$› **and**
cons: ‹$consistent\text{-}interp\ I$› **and**
atm: ‹$atm\text{-}of$ ‘ $I \subseteq atms\text{-}of\text{-}mm\ N$›
shows ‹$upostp\ I \models m\ penc\ N$›
⟨*proof*⟩

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

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

⟨*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 *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 ∈# *poss* (*mset-set* ΔΣ). *Pos* (*atm-of* $L^{\mapsto 1}$) ∈# M#} +
{#L ∈# *negs* (*mset-set* ΔΣ). *Pos* (*atm-of* $L^{\mapsto 0}$) ∈# M#}›

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

declare *finite*-Σ[*simp*]

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

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

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

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

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

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

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*⟩

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

lemma Σ*-no-weight*-ϱ_e: ‹*atm-of* $C \in \Sigma - \Delta\Sigma$ ⟹ ϱ_e (*add-mset* C M) = ϱ_e M›
⟨*proof*⟩

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

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

lemma ϱ_e*-mono*: ‹*distinct-mset* $B \Longrightarrow A \subseteq\# B \Longrightarrow \varrho_e\ A \leq \varrho_e\ B$›
⟨*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)) $= \varrho$ (*mset-set* I)› (**is** ‹*?A* = *?B*›)
⟨*proof*⟩

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
update-additional-info
$\Sigma\ \Delta\Sigma$
ϱ
new-vars +
optimal-encoding-ops
$\Sigma\ \Delta\Sigma$
new-vars 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*}› **and**
update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*› **and**
Σ ΔΣ :: ‹′*v set*› **and**
new-vars :: ‹′*v* ⇒ ′*v* × ′*v*›
begin

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*⟩

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* = Σ›
shows
‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* ≠ *None* ⟹ *postp* (*set-mset* (*the* (*weight T*))) ⊨*sm N*›
‹*weight T* ≠ *None* ⟹ *distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹
atms-of I ⊆ *atms-of-mm N* ⟹ *set-mset I* ⊨*sm N* ⟹
ϱ *I* ≥ ϱ (*mset-set* (*postp* (*set-mset* (*the* (*weight T*)))))›
‹*weight T* ≠ *None* ⟹ ϱ_e (*the* (*enc-weight-opt.weight T*)) =
ϱ (*mset-set* (*postp* (*set-mset* (*the* (*enc-weight-opt.weight T*)))))›
⟨*proof*⟩

theorem *full-encoding-OCDCL-complexity*:
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* = Σ›
shows ‹*size* (*learned-clss T*) ≤ *2* ^(*card* (*atms-of-mm N* − ΔΣ)) ∗ *4*^(*card* ΔΣ)›
⟨*proof*⟩

inductive *ocdcl*$_W$*-o-r* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
decide: ‹*odecide S S*′ ⟹ *ocdcl*$_W$*-o-r S S*′› |
bj: ‹*enc-weight-opt.cdcl-bnb-bj S S*′ ⟹ *ocdcl*$_W$*-o-r S S*′›

inductive *cdcl-bnb-r* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
cdcl-conflict: ‹*conflict S S*′ ⟹ *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'* $\Longrightarrow$ *cdcl-bnb-r S S'*› |
cdcl-o': ‹$ocdcl_W$*-o-r S S'* $\Longrightarrow$ *cdcl-bnb-r S S'*›

inductive *cdcl-bnb-r-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
cdcl-bnb-r-conflict: ‹*conflict S S'* $\Longrightarrow$ *cdcl-bnb-r-stgy S S'*› |
cdcl-bnb-r-propagate: ‹*propagate S S'* $\Longrightarrow$ *cdcl-bnb-r-stgy S S'*› |
cdcl-bnb-r-improve: ‹*enc-weight-opt.improvep S S'* $\Longrightarrow$ *cdcl-bnb-r-stgy S S'*› |
cdcl-bnb-r-conflict-opt: ‹*enc-weight-opt.conflict-opt S S'* $\Longrightarrow$ *cdcl-bnb-r-stgy S S'*› |
cdcl-bnb-r-other': ‹$ocdcl_W$*-o-r S S'* $\Longrightarrow$ *no-confl-prop-impr S* $\Longrightarrow$ *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* $\Longrightarrow$ *P T*›
‹*enc-weight-opt.obacktrack S T* $\Longrightarrow$ *P T*›
‹*skip S T* $\Longrightarrow$ *P T*›
‹*resolve S T* $\Longrightarrow$ *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* $\Longrightarrow$ *decide S T*›
⟨*proof*⟩

lemma $ocdcl_W$*-o-r-*$ocdcl_W$*-o*:
‹$ocdcl_W$*-o-r S T* $\Longrightarrow$ *enc-weight-opt.*$ocdcl_W$*-o S T*›
⟨*proof*⟩

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

lemma *cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹*cdcl-bnb-r-stgy S T* $\Longrightarrow$ *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*:
‹$cdcl\text{-}bnb\text{-}r^{**}$ *S T* $\Longrightarrow$ $enc\text{-}weight\text{-}opt.cdcl\text{-}bnb^{**}$ *S T*›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}\ S\ T \Longrightarrow enc\text{-}weight\text{-}opt.cdcl\text{-}bnb\text{-}stgy^{**}\ S\ T$›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-all-struct-inv*:
‹$cdcl\text{-}bnb\text{-}r^{**}\ S\ T \Longrightarrow$
$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (enc\text{-}weight\text{-}opt.abs\text{-}state\ S) \Longrightarrow$
$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (enc\text{-}weight\text{-}opt.abs\text{-}state\ T)$›
⟨*proof*⟩

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

end

lemma *no-step-cdcl-bnb-r-stgy-no-step-cdcl-bnb-stgy*:
assumes
N: ‹$init\text{-}clss\ S = penc\ N$› **and**
Σ: ‹$atms\text{-}of\text{-}mm\ N = \Sigma$› **and**
n-d: ‹$no\text{-}dup\ (trail\ S)$› **and**
tr-alien: ‹$atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (trail\ S) \subseteq \Sigma \cup replacement\text{-}pos\ `\ \Delta\Sigma \cup replacement\text{-}neg\ `\ \Delta\Sigma$›
shows
‹$no\text{-}step\ cdcl\text{-}bnb\text{-}r\text{-}stgy\ S \longleftrightarrow no\text{-}step\ enc\text{-}weight\text{-}opt.cdcl\text{-}bnb\text{-}stgy\ S$› (**is** ‹$?A \longleftrightarrow ?B$›)
⟨*proof*⟩

lemma *cdcl-bnb-r-stgy-init-clss*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy\ S\ T \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$›
⟨*proof*⟩

lemma *rtranclp-cdcl-bnb-r-stgy-init-clss*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}\ S\ T \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$›
⟨*proof*⟩

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

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

lemma *propagation-one-lit-of-same-lvl*:
assumes
‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (abs\text{-}state\ S)$› **and**
‹$no\text{-}smaller\text{-}propa\ S$› **and**
‹$Propagated\ L\ E \in set\ (trail\ S)$› **and**

rea: ‹*reasons-in-clauses S*› **and**
nempty: ‹$E - \{\#L\#\} \neq \{\#\}$›
shows
‹$\exists L' \in\# E - \{\#L\#\}$. *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* = (∑ *C* ∈# (*filter-mset* (λ*C*. *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* $L \in$ *replacement-neg* ‘ $\Delta\Sigma$
then Pos (*replacement-pos* (*SOME K*. $K \in \Delta\Sigma \wedge L =$ *replacement-neg K*))
else Neg L)›

abbreviation *normalize-clause* :: ‹′*v clause* ⇒ ′*v clause*› **where**
‹*normalize-clause C* ≡ *normalize-lit* ‘# *C*›

lemma *normalize-lit*[*simp*]:
‹$L \in \Sigma - \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Pos L*) = (*Pos L*)›
‹$L \in \Sigma - \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Neg L*) = (*Neg L*)›
‹$L \in \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Pos* (*replacement-neg L*)) = *Neg* (*replacement-pos L*)›
‹$L \in \Delta\Sigma \Longrightarrow$ *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* (($\Sigma - \Delta\Sigma$) ∪ *replacement-neg* ‘ $\Delta\Sigma$ ∪ *replacement-pos* ‘ $\Delta\Sigma$))›

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* $L \in$ *replacement-pos* ‘ $\Delta\Sigma$ *then replacement-neg* (*base-atm L*)
else replacement-pos (*base-atm L*))›

lemma *opposite-var-replacement-if*[*simp*]:
‹$L \in$ (*replacement-neg* ‘ $\Delta\Sigma$ ∪ *replacement-pos* ‘ $\Delta\Sigma$) $\Longrightarrow A \in \Delta\Sigma \Longrightarrow$
opposite-var L = *replacement-pos A* ⟷ *L* = *replacement-neg A*›
‹$L \in$ (*replacement-neg* ‘ $\Delta\Sigma$ ∪ *replacement-pos* ‘ $\Delta\Sigma$) $\Longrightarrow A \in \Delta\Sigma \Longrightarrow$
opposite-var L = *replacement-neg A* ⟷ *L* = *replacement-pos A*›
‹$A \in \Delta\Sigma \Longrightarrow$ *opposite-var* (*replacement-pos A*) = *replacement-neg A*›
‹$A \in \Delta\Sigma \Longrightarrow$ *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* (($\Sigma - \Delta\Sigma$) ∪ *replacement-neg* ‘ $\Delta\Sigma$ ∪ *replacement-pos* ‘ $\Delta\Sigma$)›

‹*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-*ΔΣ [] = (λ*L*. *L* ∈ *replacement-pos* ‘ ΔΣ)›
‹*full-unset-literals-in-*ΔΣ′ [] = (λ*L*. *L* ∈ *replacement-neg* ‘ ΔΣ)›
‹*half-unset-literals-in-*ΔΣ [] = (λ*L*. *False*)›
‹*unset-literals-in-*Σ [] = (λ*L*. *L* ∈ Σ − ΔΣ)›
⟨*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*]:
‹$length\ (sorted\text{-}unadded\text{-}literals\ []) = length\ all\text{-}clauses\text{-}literals$›
⟨*proof*⟩

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

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

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

lemma *distinct-sorted-unadded-literals*[*simp*]:
‹$distinct\ (sorted\text{-}unadded\text{-}literals\ M)$›
⟨*proof*⟩

lemma *Collect-req-remove1*:
‹$\{a \in A.\ a \neq b \wedge P\ a\} = (if\ P\ b\ then\ Set.remove\ b\ \{a \in A.\ P\ a\}\ else\ \{a \in A.\ P\ a\})$› **and**
Collect-req-remove2:
‹$\{a \in A.\ b \neq a \wedge P\ a\} = (if\ P\ b\ then\ Set.remove\ b\ \{a \in A.\ P\ a\}\ else\ \{a \in A.\ P\ a\})$›
⟨*proof*⟩

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

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

lemma *no-dup-complete-trail*[*simp*]:
‹$no\text{-}dup\ (complete\text{-}trail\ M) \longleftrightarrow no\text{-}dup\ M$›
⟨*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*) ∈ *lexn less-than n* ⟹ (*b*, *c*) ∈ *lexn less-than n* ∨ *b* = *c* ⟹ (*a*, *c*) ∈ *lexn less-than n*›
‹(*a*, *b*) ∈ *lexn less-than n* ⟹ (*b*, *c*) ∈ *lexn less-than n* ∨ *b* = *c* ⟹ (*a*, *c*) ∈ *lexn less-than n*›
⟨*proof*⟩

lemma *truncate-trail-Prop*[*simp*]:
‹*truncate-trail* (*Propagated L E* # *S*) = *truncate-trail* (*S*)›
⟨*proof*⟩

lemma *ocdcl-score-Prop*[*simp*]:
‹*ocdcl-score* (*Propagated L E* # *S*) = *ocdcl-score* (*S*)›
⟨*proof*⟩

lemma *remove-dup-information-raw2-undefined-*Σ:
‹*distinct xs* ⟹
(⋀*L*. *L* ∈ *set xs* ⟹ *undefined-lit M* (*Pos L*) ⟹ *L* ∈ Σ ⟹ *undefined-lit MM* (*Pos L*)) ⟹
remove-dup-information-raw2 MM
(*map* (*Decided* ∘ *Pos*)
(*filter* (*unset-literals-in-*Σ *M*)
xs)) =
map (*SD-TWO o Decided* ∘ *Pos*)
(*filter* (*unset-literals-in-*Σ *M*)
xs)›
⟨*proof*⟩

lemma *defined-lit-map-Decided-pos*:
‹*defined-lit* (*map* (*Decided* ∘ *Pos*) *M*) *L* ⟷ *atm-of L* ∈ *set M*›
⟨*proof*⟩

lemma *remove-dup-information-raw2-full-undefined-*Σ:
‹*distinct xs* ⟹ *set xs* ⊆ *set all-clauses-literals* ⟹
(⋀*L*. *L* ∈ *set xs* ⟹ *undefined-lit M* (*Pos L*) ⟹ *L* ∉ Σ − ΔΣ ⟹
undefined-lit M (*Pos* (*opposite-var L*)) ⟹ *L* ∈ *replacement-pos* ‘ ΔΣ ⟹
undefined-lit MM (*Pos* (*opposite-var L*))) ⟹
remove-dup-information-raw2 MM
(*map* (*Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-*ΔΣ *M*)
xs)) =
map (*SD-ONE o Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-*ΔΣ *M*)
xs)›
⟨*proof*⟩

lemma *full-unset-literals-in-*ΔΣ*-notin*[*simp*]:
‹*La* ∈ Σ ⟹ *full-unset-literals-in-*ΔΣ *M La* ⟷ *False*›
‹*La* ∈ Σ ⟹ *full-unset-literals-in-*ΔΣ′ *M La* ⟷ *False*›
⟨*proof*⟩

lemma *Decided-in-definedD*: ‹*Decided K* ∈ *set M* ⟹ *defined-lit M K*›
⟨*proof*⟩

lemma *full-unset-literals-in-*ΔΣ′*-full-unset-literals-in-*ΔΣ:
‹*L* ∈ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ ⟹
*full-unset-literals-in-*ΔΣ′ *M* (*opposite-var L*) ⟷ *full-unset-literals-in-*ΔΣ *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-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*⟩

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 ‹$cdcl\text{-}bnb\text{-}r^{**}$ 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 ⟹ $cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}$ 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 ‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}$ 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*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}$ S T ⟹ $cdcl\text{-}bnb\text{-}r^{**}$ 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*:
‹$cdcl\text{-}dpll\text{-}bnb\text{-}r\text{-}stgy\ S\ T \Longrightarrow$
$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (enc\text{-}weight\text{-}opt.abs\text{-}state\ S) \Longrightarrow$
$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (enc\text{-}weight\text{-}opt.abs\text{-}state\ T)$›
⟨*proof*⟩

end

lemma *cdcl-bnb-r-stgy-cdcl-dpll-bnb-r-stgy*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy\ S\ T \Longrightarrow \exists\, T.\ cdcl\text{-}dpll\text{-}bnb\text{-}r\text{-}stgy\ S\ T$›
⟨*proof*⟩

context
fixes $S :: 'st$
assumes *S-Σ*: ‹$atms\text{-}of\text{-}mm\ (init\text{-}clss\ S) = \Sigma - \Delta\Sigma \cup replacement\text{-}pos\ `\ \Delta\Sigma \cup replacement\text{-}neg\ `\ \Delta\Sigma$›
begin

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

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

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

end

lemma *replace-pos-neg-not-both-decided-highest-lvl*:
assumes
struct: ‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (enc\text{-}weight\text{-}opt.abs\text{-}state\ S)$› **and**
smaller-propa: ‹$no\text{-}smaller\text{-}propa\ S$› **and**
smaller-confl: ‹$no\text{-}smaller\text{-}confl\ S$› **and**
dec0: ‹$Pos\ (A^{\mapsto 0}) \in lits\text{-}of\text{-}l\ (trail\ S)$› **and**
dec1: ‹$Pos\ (A^{\mapsto 1}) \in lits\text{-}of\text{-}l\ (trail\ S)$› **and**
add: ‹$additional\text{-}constraints \subseteq\#\ init\text{-}clss\ S$› **and**
[*simp*]: ‹$A \in \Delta\Sigma$›
shows ‹$get\text{-}level\ (trail\ S)\ (Pos\ (A^{\mapsto 0})) = backtrack\text{-}lvl\ S\ \wedge$
$get\text{-}level\ (trail\ S)\ (Pos\ (A^{\mapsto 1})) = backtrack\text{-}lvl\ S$›
⟨*proof*⟩

lemma *cdcl-dpll-bnb-r-stgy-clauses-mono*:
‹$cdcl\text{-}dpll\text{-}bnb\text{-}r\text{-}stgy\ S\ T \Longrightarrow clauses\ S \subseteq\#\ clauses\ T$›
⟨*proof*⟩

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-clauses-mono*:
‹$cdcl\text{-}dpll\text{-}bnb\text{-}r\text{-}stgy^{**}$ *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*:
‹$cdcl\text{-}dpll\text{-}bnb\text{-}r\text{-}stgy^{**}$ *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 :: ‹$'v \Rightarrow$ *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* ⟹ *I'* ⊆# *I* ⟹ *is-dominating* $\mathcal{M}$ *I'*›
⟨*proof*⟩

lemma *is-dominating-add-mset*:
‹*is-dominating* (*add-mset x* $\mathcal{M}$) *I* ⟷
is-dominating $\mathcal{M}$ *I* ∨ (∃ *J*. *I* ⊆# *J* ∧ *model-is-dominated J x*)›

⟨*proof*⟩

definition *is-improving-int*
:: ‹('v, 'v clause) ann-lits ⇒ ('v, 'v clause) ann-lits ⇒ 'v clauses ⇒ 'v cov ⇒ bool›
where
‹*is-improving-int M M' N* $\mathcal{M}$ ⟷
M = *M'* ∧ (∀ *I* ∈# $\mathcal{M}$. ¬*model-is-dominated* (*lit-of* '# *mset M*) *I*) ∧
total-over-m (*lits-of-l M*) (*set-mset N*) ∧
lit-of '# *mset M* ∈ *simple-clss* (*atms-of-mm N*) ∧
lit-of '# *mset M* ∉# $\mathcal{M}$ ∧
M ⊨*asm N* ∧
no-dup M›

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

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

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

lemma *conflicting-clauses-insert*:
assumes ‹*M* ∈ *simple-clss* (*atms-of-mm N*)› **and** ‹*atms-of M* = *atms-of-mm N*›
shows ‹*pNeg M* ∈# *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* ⊨*m N*› **and**
‹*consistent-interp* (*set-mset I*)› **and**
‹¬*tautology I*› **and**
‹*distinct-mset I*›
shows
‹*pNeg I* ∈# *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* ⇒ '*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: '*st* ⇒ ('*v*, '*v clause*) *ann-lits* × '*v clauses* × '*v clauses* × '*v clause option* ×
'*v cov* × '*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* ⇒ *bool*› +
fixes
update-additional-info :: ‹'*v cov* × '*b* ⇒ '*st* ⇒ '*st*›
assumes
update-additional-info:
‹*state S* = (*M*, *N*, *U*, *C*, $\mathcal{M}$) ⟹ *state* (*update-additional-info K' S*) = (*M*, *N*, *U*, *C*, *K'*)› **and**
weight-init-state:
‹⋀*N* :: '*v clauses*. *fst* (*additional-info* (*init-state N*)) = {#}›
begin

definition *update-weight-information* :: ‹('*v*, '*v clause*) *ann-lits* ⇒ '*st* ⇒ '*st*› **where**
‹*update-weight-information M S* =
update-additional-info (*add-mset* (*lit-of* '# *mset M*) (*fst* (*additional-info S*)), *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 *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-bnb*$_W$*-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-bnb*$_W$*-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*:
‹$cdcl\text{-}bnb^{**}$ 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 ⊨*m* *conflicting-clss* S› **and**
‹I ⊨*m* *CDCL-W-Abstract-State.init-clss* (*abs-state* S)›
⟨*proof*⟩

lemma *exists-model-with-true-lit-still-model*:
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**
cdcl: ‹*cdcl-bnb S T*› **and**
no-L-T: ‹¬(∃ *J*∈# *covering T*. *Pos L* ∈# *J*)› **and**
cov: ‹*covering-simple-clss N S*› **and**
NS: ‹*atms-of-mm N* = *atms-of-mm* (*init-clss S*)›
shows ‹*I* ⊨*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* ∈ *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**
cdcl: ‹$cdcl\text{-}bnb^{**}$ *S T*› **and**
cov: ‹*covering-simple-clss N S*› **and**
‹*N* = *init-clss S*›
shows ‹*I* ⊨*m CDCL-W-Abstract-State.init-clss* (*abs-state T*) ∨ (∃ *J*∈# *covering T*. *Pos L* ∈# *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*)) ⊆ *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* ∈ *I* ⟹ *ϱ L* ⟹ *L* ∈ *atms-of-mm N* ⟹ *total-over-m I* (*set-mset N*) ⟹ *consistent-interp I* ⟹ *I* ⊨*m N* ⟹
∃ *J* ∈# *covering T*. *Pos L* ∈# *J*›
⟨*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*′ ⟷
filter-mset (λ*L*. *is-pos L*) *M* ⊆# *filter-mset* (λ*L*. *is-pos L*) *M*′›
⟨*proof*⟩

lemma
‹*cover-all-vars.conflicting-clauses N* 𝓜 =
{# *C* ∈# (*mset-set* (*simple-clss* (*atms-of-mm N*))).
(*pNeg* ‘
{*a*. *a* ∈# *mset-set* (*simple-clss* (*atms-of-mm N*)) ∧
(∃ *M*∈#𝓜. ∃ *J*. *a* ⊆# *J* ∧ *cover-all-vars.model-is-dominated J M*) ∧
atms-of a = *atms-of-mm N*} ∪
set-mset N) ⊨*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* ∈ *I* ⟹ *L* ∈ *atms-of-mm N* ⟹ *total-over-m I* (*set-mset N*) ⟹ *consistent-interp I* ⟹ *I* ⊨*m N* ⟹
∃ *J* ∈# *covering T*. *Pos L* ∈# *J*›
⟨*proof*⟩

end

end
theory *DPLL-W-BnB*
imports
CDCL-W-Optimal-Model
CDCL.DPLL-W
begin

lemma [*simp*]: ‹*backtrack-split M1* = (*M*′, *L* # *M*) ⟹ *is-decided L*›

⟨*proof*⟩

lemma *funpow-tl-append-skip-ge*:
‹$n \geq$ *length* $M' \Longrightarrow$ ((*tl* ^^ *n*) (M' @ M)) = (*tl* ^^ (n − *length* M')) M›
⟨*proof*⟩

The following version is more suited than $\exists\, l{\in}set\ ?M.\ is\text{-}decided\ l \Longrightarrow \exists\, M'\ L'\ M''.\ back\text{-}track\text{-}split\ ?M = (M'',\ L' \#\ M')$ for direct use.

lemma *backtrack-split-some-is-decided-then-snd-has-hd'*:
‹$l{\in}set\ M \Longrightarrow is\text{-}decided\ l \Longrightarrow \exists\, M'\ L'\ M''.\ backtrack\text{-}split\ M = (M'',\ L' \#\ M')$›
⟨*proof*⟩

lemma *total-over-m-entailed-or-conflict*:
shows ‹$total\text{-}over\text{-}m\ M\ N \Longrightarrow M \models s\ N \lor (\exists\, C \in N.\ M \models s\ CNot\ C)$›
⟨*proof*⟩

The locales on DPLL should eventually be moved to the DPLL theory, but currently it is only a discount version (in particular, we cheat and don't use $S \sim T$ in the transition system below, even if it would be cleaner to do as as we de for CDCL).

locale *dpll-ops* =
fixes
trail :: ‹$'st \Rightarrow 'v\ \ dpll_W\text{-}ann\text{-}lits$› **and**
clauses :: ‹$'st \Rightarrow 'v\ clauses$› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
cons-trail :: ‹$'v\ \ dpll_W\text{-}ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$› **and**
state-eq :: ‹$'st \Rightarrow 'st \Rightarrow bool$› (**infix** ‹∼› *50*) **and**
state :: ‹$'st \Rightarrow 'v\ \ dpll_W\text{-}ann\text{-}lits \times 'v\ clauses \times 'b$›
begin

definition *additional-info* :: ‹$'st \Rightarrow 'b$› **where**
‹*additional-info* $S = (\lambda(M,\ N,\ w).\ w)$ (*state* S)›

definition *reduce-trail-to* :: ‹$'v\ \ dpll_W\text{-}ann\text{-}lits \Rightarrow 'st \Rightarrow 'st$› **where**
‹*reduce-trail-to* M S = (*tl-trail* ^^ (*length* (*trail* S) − *length* M)) S›

end

locale *bnb-ops* =
fixes
trail :: ‹$'st \Rightarrow 'v\ \ dpll_W\text{-}ann\text{-}lits$› **and**
clauses :: ‹$'st \Rightarrow 'v\ clauses$› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
cons-trail :: ‹$'v\ \ dpll_W\text{-}ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$› **and**
state-eq :: ‹$'st \Rightarrow 'st \Rightarrow bool$› (**infix** ‹∼› *50*) **and**
state :: ‹$'st \Rightarrow 'v\ \ dpll_W\text{-}ann\text{-}lits \times 'v\ clauses \times 'a \times 'b$› **and**
weight :: ‹$'st \Rightarrow 'a$› **and**
update-weight-information :: ‹$'v\ dpll_W\text{-}ann\text{-}lits \Rightarrow 'st \Rightarrow 'st$› **and**
is-improving-int :: ‹$'v\ \ dpll_W\text{-}ann\text{-}lits \Rightarrow 'v\ \ dpll_W\text{-}ann\text{-}lits \Rightarrow 'v\ clauses \Rightarrow 'a \Rightarrow bool$› **and**
conflicting-clauses :: ‹$'v\ clauses \Rightarrow 'a \Rightarrow 'v\ clauses$›
begin

interpretation *dpll*: *dpll-ops* **where**

trail = *trail* **and**
clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state*
⟨*proof*⟩

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

definition *abs-state* **where**
‹*abs-state S* = (*trail S*, *clauses S* + *conflicting-clss S*)›

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

definition *state'* :: ‹*'st* ⇒ *'v* $dpll_W$*-ann-lits* × *'v clauses* × *'a* × *'v clauses*› **where**
‹*state' S* = (*trail S*, *clauses S*, *weight S*, *conflicting-clss S*)›

definition *additional-info* :: ‹*'st* ⇒ *'b*› **where**
‹*additional-info S* = (λ(*M*, *N*, -, *w*). *w*) (*state S*)›

end

locale $dpll_W$*-state* =
dpll-ops trail clauses
tl-trail cons-trail state-eq state
for
trail :: ‹*'st* ⇒ *'v* $dpll_W$*-ann-lits*› **and**
clauses :: ‹*'st* ⇒ *'v clauses*› **and**
tl-trail :: ‹*'st* ⇒ *'st*› **and**
cons-trail :: ‹*'v* $dpll_W$*-ann-lit* ⇒ *'st* ⇒ *'st*› **and**
state-eq :: ‹*'st* ⇒ *'st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ‹*'st* ⇒ *'v* $dpll_W$*-ann-lits* × *'v clauses* × *'b*› +
assumes
state-eq-ref[*simp*, *intro*]: ‹*S* ∼ *S*› **and**
state-eq-sym: ‹*S* ∼ *T* ⟷ *T* ∼ *S*› **and**
state-eq-trans: ‹*S* ∼ *T* ⟹ *T* ∼ *U'* ⟹ *S* ∼ *U'*› **and**
state-eq-state: ‹*S* ∼ *T* ⟹ *state S* = *state T*› **and**

cons-trail:
⋀*S'*. *state st* = (*M*, *S'*) ⟹
state (*cons-trail L st*) = (*L* # *M*, *S'*) **and**

tl-trail:
‹⋀*S'*. *state st* = (*M*, *S'*) ⟹ *state* (*tl-trail st*) = (*tl M*, *S'*)› **and**
state:
‹*state S* = (*trail S*, *clauses S*, *additional-info S*)›
begin

lemma [*simp*]:
‹*clauses* (*cons-trail uu S*) = *clauses S*›

‹trail (cons-trail uu S) = uu # trail S›
‹trail (tl-trail S) = tl (trail S)›
‹clauses (tl-trail S) = clauses (S)›
‹additional-info (cons-trail L S) = additional-info S›
‹additional-info (tl-trail S) = additional-info S›
⟨proof⟩

lemma *state-simp*[*simp*]:
‹T ∼ S ⟹ trail T = trail S›
‹T ∼ S ⟹ clauses T = clauses S›
⟨proof⟩

lemma *state-tl-trail*: ‹state (tl-trail S) = (tl (trail S), clauses S, additional-info S)›
⟨proof⟩

lemma *state-tl-trail-comp-pow*: ‹state ((tl-trail ^^ n) S) = ((tl ^^ n) (trail S), clauses S, additional-info S)›
⟨proof⟩

lemma *reduce-trail-to-simps*[*simp*]:
‹backtrack-split (trail S) = (M', L # M) ⟹ trail (reduce-trail-to M S) = M›
‹clauses (reduce-trail-to M S) = clauses S›
‹additional-info (reduce-trail-to M S) = additional-info S›
⟨proof⟩

inductive *dpll-backtrack* :: ‹'st ⇒ 'st ⇒ bool› **where**
‹dpll-backtrack S T›
if
‹D ∈# clauses S› **and**
‹trail S ⊨as CNot D› **and**
‹backtrack-split (trail S) = (M', L # M)› **and**
‹T ∼cons-trail (Propagated (−lit-of L) ()) (reduce-trail-to M S)›

inductive *dpll-propagate* :: ‹'st ⇒ 'st ⇒ bool› **where**
‹dpll-propagate S T›
if
‹add-mset L D ∈# clauses S› **and**
‹trail S ⊨as CNot D› **and**
‹undefined-lit (trail S) L›
‹T ∼ cons-trail (Propagated L ()) S›

inductive *dpll-decide* :: ‹'st ⇒ 'st ⇒ bool› **where**
‹dpll-decide S T›
if
‹undefined-lit (trail S) L›
‹T ∼ cons-trail (Decided L) S›
‹atm-of L ∈ atms-of-mm (clauses S)›

inductive *dpll* :: ‹'st ⇒ 'st ⇒ bool› **where**
‹dpll S T› **if** ‹dpll-decide S T› |
‹dpll S T› **if** ‹dpll-propagate S T› |
‹dpll S T› **if** ‹dpll-backtrack S T›

lemma *dpll-is-*$dpll_W$:
‹dpll S T ⟹ $dpll_W$ (trail S, clauses S) (trail T, clauses T)›

⟨*proof*⟩

end

locale *bnb* =
bnb-ops trail clauses
tl-trail cons-trail state-eq state weight update-weight-information is-improving-int conflicting-clauses
for
weight :: ‹$'st \Rightarrow 'a$› **and**
update-weight-information :: ‹$'v$ $dpll_W$*-ann-lits* $\Rightarrow 'st \Rightarrow 'st$› **and**
is-improving-int :: ‹$'v$ $dpll_W$*-ann-lits* $\Rightarrow 'v$ $dpll_W$*-ann-lits* $\Rightarrow 'v$ *clauses* $\Rightarrow 'a \Rightarrow$ *bool*› **and**
trail :: ‹$'st \Rightarrow 'v$ $dpll_W$*-ann-lits*› **and**
clauses :: ‹$'st \Rightarrow 'v$ *clauses*› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
cons-trail :: ‹$'v$ $dpll_W$*-ann-lit* $\Rightarrow 'st \Rightarrow 'st$› **and**
state-eq :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› (**infix** ‹∼› *50*) **and**
conflicting-clauses :: ‹$'v$ *clauses* $\Rightarrow 'a \Rightarrow 'v$ *clauses*›**and**
state :: ‹$'st \Rightarrow 'v$ $dpll_W$*-ann-lits* $\times$ $'v$ *clauses* $\times$ $'a \times 'b$› +
assumes
state-eq-ref[*simp*, *intro*]: ‹$S \sim S$› **and**
state-eq-sym: ‹$S \sim T \longleftrightarrow T \sim S$› **and**
state-eq-trans: ‹$S \sim T \Longrightarrow T \sim U' \Longrightarrow S \sim U'$› **and**
state-eq-state: ‹$S \sim T \Longrightarrow$ *state* S = *state* T› **and**

cons-trail:
$\bigwedge S'$. *state st* = (M, S') $\Longrightarrow$
state (*cons-trail L st*) = $(L \# M, S')$ **and**

tl-trail:
‹$\bigwedge S'$. *state st* = (M, S') $\Longrightarrow$ *state* (*tl-trail st*) = (*tl* M, S')› **and**
update-weight-information:
‹*state* S = (M, N, w, oth) $\Longrightarrow$
$\exists w'$. *state* (*update-weight-information* M' S) = (M, N, w', oth)› **and**

conflicting-clss-update-weight-information-mono:
‹$dpll_W$*-all-inv* (*abs-state* S) $\Longrightarrow$ *is-improving* M M' S $\Longrightarrow$
conflicting-clss S $\subseteq\#$ *conflicting-clss* (*update-weight-information* M' S)› **and**
conflicting-clss-update-weight-information-in:
‹*is-improving* M M' S $\Longrightarrow$ *negate-ann-lits* M' $\in\#$ *conflicting-clss* (*update-weight-information* M'
S)› **and**
atms-of-conflicting-clss:
‹*atms-of-mm* (*conflicting-clss* S) $\subseteq$ *atms-of-mm* (*clauses* S)› **and**
state:
‹*state* S = (*trail* S, *clauses* S, *weight* S, *additional-info* S)›
begin

lemma [*simp*]: ‹*DPLL-W.clauses* (*abs-state* S) = *clauses* S + *conflicting-clss* S›
‹*DPLL-W.trail* (*abs-state* S) = *trail* S›
⟨*proof*⟩

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

lemma [*simp*]:

‹*clauses* (*update-weight-information* M' *S*) = *clauses S*›
‹*weight* (*cons-trail uu S*) = *weight S*›
‹*clauses* (*cons-trail uu S*) = *clauses S*›
‹*conflicting-clss* (*cons-trail uu S*) = *conflicting-clss S*›
‹*trail* (*cons-trail uu S*) = *uu* # *trail S*›
‹*trail* (*tl-trail S*) = *tl* (*trail S*)›
‹*clauses* (*tl-trail S*) = *clauses* (*S*)›
‹*weight* (*tl-trail S*) = *weight* (*S*)›
‹*conflicting-clss* (*tl-trail S*) = *conflicting-clss* (*S*)›
‹*additional-info* (*cons-trail L S*) = *additional-info S*›
‹*additional-info* (*tl-trail S*) = *additional-info S*›
‹*additional-info* (*update-weight-information* M' *S*) = *additional-info S*›
⟨*proof*⟩

lemma *state-simp*[*simp*]:
‹*T* ∼ *S* ⟹ *trail T* = *trail S*›
‹*T* ∼ *S* ⟹ *clauses T* = *clauses S*›
‹*T* ∼ *S* ⟹ *weight T* = *weight S*›
‹*T* ∼ *S* ⟹ *conflicting-clss T* = *conflicting-clss S*›
⟨*proof*⟩

interpretation *dpll*: *dpll-ops trail clauses tl-trail cons-trail state-eq state*
⟨*proof*⟩

interpretation *dpll*: $dpll_W$*-state trail clauses tl-trail cons-trail state-eq state*
⟨*proof*⟩

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

inductive *backtrack-opt* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
backtrack-opt: *backtrack-split* (*trail S*) = (M', *L* # *M*) ⟹ *is-decided L* ⟹ *D* ∈# *conflicting-clss S*
⟹ *trail S* ⊨*as CNot D*
⟹ *T* ∼*cons-trail* (*Propagated* (−*lit-of L*) ()) (*dpll.reduce-trail-to M S*)
⟹ *backtrack-opt S T*

In the definition below the *state′ T* = (*Propagated L* () # *trail S*, *clauses S*, *weight S*, *conflicting-clss S*) are not necessary, but avoids to change the DPLL formalisation with proper locales, as we did for CDCL.

The DPLL calculus looks slightly more general than the CDCL calculus because we can take any conflicting clause from *conflicting-clss S*. However, this does not make a difference for the trail, as we backtrack to the last decision independantly of the conflict.

inductive $dpll_W$*-core* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S T* **where**
propagate: ‹*dpll.dpll-propagate S T* ⟹ $dpll_W$*-core S T*› |
decided: ‹*dpll.dpll-decide S T* ⟹ $dpll_W$*-core S T* › |
backtrack: ‹*dpll.dpll-backtrack S T* ⟹ $dpll_W$*-core S T*› |
backtrack-opt: ‹*backtrack-opt S T* ⟹ $dpll_W$*-core S T*›

inductive-cases $dpll_W$*-coreE*: ‹$dpll_W$*-core S T*›

inductive $dpll_W$*-bound* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
update-info:
‹*is-improving M* M' *S* ⟹ *T* ∼ (*update-weight-information* M' *S*)

$\Longrightarrow$ *dpll*$_W$*-bound S T*›

inductive *dpll*$_W$*-bnb* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
dpll:
‹*dpll*$_W$*-bnb S T*›
if ‹*dpll*$_W$*-core S T*› |
bnb:
‹*dpll*$_W$*-bnb S T*›
if ‹*dpll*$_W$*-bound S T*›

inductive-cases *dpll*$_W$*-bnbE*: ‹*dpll*$_W$*-bnb S T*›

lemma *dpll*$_W$*-core-is-dpll*$_W$:
‹*dpll*$_W$*-core S T* $\Longrightarrow$ *dpll*$_W$ (*abs-state S*) (*abs-state T*)›
⟨*proof*⟩

lemma *dpll*$_W$*-core-abs-state-all-inv*:
‹*dpll*$_W$*-core S T* $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state S*) $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state T*)›
⟨*proof*⟩

lemma *dpll*$_W$*-core-same-weight*:
‹*dpll*$_W$*-core S T* $\Longrightarrow$ *weight S* = *weight T*›
⟨*proof*⟩

lemma *dpll*$_W$*-bound-trail*:
‹*dpll*$_W$*-bound S T* $\Longrightarrow$ *trail S* = *trail T*› **and**
dpll$_W$*-bound-clauses*:
‹*dpll*$_W$*-bound S T* $\Longrightarrow$ *clauses S* = *clauses T*› **and**
dpll$_W$*-bound-conflicting-clss*:
‹*dpll*$_W$*-bound S T* $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state S*) $\Longrightarrow$ *conflicting-clss S* ⊆# *conflicting-clss T*›
⟨*proof*⟩

lemma *dpll*$_W$*-bound-abs-state-all-inv*:
‹*dpll*$_W$*-bound S T* $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state S*) $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state T*)›
⟨*proof*⟩

lemma *dpll*$_W$*-bnb-abs-state-all-inv*:
‹*dpll*$_W$*-bnb S T* $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state S*) $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state T*)›
⟨*proof*⟩

lemma *rtranclp-dpll*$_W$*-bnb-abs-state-all-inv*:
‹*dpll*$_W$*-bnb*** *S T* $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state S*) $\Longrightarrow$ *dpll*$_W$*-all-inv* (*abs-state T*)›
⟨*proof*⟩

lemma *dpll*$_W$*-core-clauses*:
‹*dpll*$_W$*-core S T* $\Longrightarrow$ *clauses S* = *clauses T*›
⟨*proof*⟩

lemma *dpll*$_W$*-bnb-clauses*:
‹*dpll*$_W$*-bnb S T* $\Longrightarrow$ *clauses S* = *clauses T*›
⟨*proof*⟩

lemma *rtranclp-dpll*$_W$*-bnb-clauses*:
‹*dpll*$_W$*-bnb*** *S T* $\Longrightarrow$ *clauses S* = *clauses T*›
⟨*proof*⟩

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

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

lemma [*simp*]:
‹*weight* ((*tl-trail* ^^ *n*) *S*) = *weight S*›
‹*trail* ((*tl-trail* ^^ *n*) *S*) = (*tl* ^^ *n*) (*trail S*)›
‹*clauses* ((*tl-trail* ^^ *n*) *S*) = *clauses S*›
‹*conflicting-clss* ((*tl-trail* ^^ *n*) *S*) = *conflicting-clss S*›
⟨*proof*⟩

lemma *dpll$_W$-core-Ex-propagate*:
‹*add-mset L C* ∈# *clauses S* ⟹ *trail S* ⊨*as CNot C* ⟹ *undefined-lit* (*trail S*) *L* ⟹
Ex (*dpll$_W$-core S*)› **and**
dpll$_W$-core-Ex-decide:
undefined-lit (*trail S*) *L* ⟹ *atm-of L* ∈ *atms-of-mm* (*clauses S*) ⟹
Ex (*dpll$_W$-core S*) **and**
dpll$_W$-core-Ex-backtrack: *backtrack-split* (*trail S*) = (*M'*, *L'* # *M*) ⟹ *is-decided L'* ⟹ *D* ∈# *clauses S* ⟹
trail S ⊨*as CNot D* ⟹ *Ex* (*dpll$_W$-core S*) **and**
dpll$_W$-core-Ex-backtrack-opt: *backtrack-split* (*trail S*) = (*M'*, *L'* # *M*) ⟹ *is-decided L'* ⟹ *D* ∈# *conflicting-clss S*
⟹ *trail S* ⊨*as CNot D* ⟹
Ex (*dpll$_W$-core S*)
⟨*proof*⟩

Unlike the CDCL case, we do not need assumptions on improve. The reason behind it is that we do not need any strategy on propagation and decisions.

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

context
assumes *can-always-improve*:
‹⋀*S*. *trail S* ⊨*asm clauses S* ⟹ (∀ *C* ∈# *conflicting-clss S*. ¬ *trail S* ⊨*as CNot C*) ⟹
dpll$_W$-all-inv (*abs-state S*) ⟹
total-over-m (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*)) ⟹ *Ex* (*dpll$_W$-bound S*)›
begin

lemma *no-step-dpll$_W$-bnb-conflict*:

assumes
ns: ‹*no-step* $dpll_W$*-bnb S*› **and**
invs: ‹$dpll_W$*-all-inv* (*abs-state S*)›
shows ‹∃ *C* ∈# *clauses S* + *conflicting-clss S*. *trail S* ⊨*as CNot C*› (**is** *?A*) **and**
‹*count-decided* (*trail S*) = *0*› **and**
‹*unsatisfiable* (*set-mset* (*clauses S* + *conflicting-clss S*))›
⟨*proof*⟩

end

inductive $dpll_W$*-core-stgy* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **for** *S T* **where**
propagate: ‹*dpll.dpll-propagate S T* ⟹ $dpll_W$*-core-stgy S T*› |
decided: ‹*dpll.dpll-decide S T* ⟹ *no-step dpll.dpll-propagate S* ⟹ $dpll_W$*-core-stgy S T* › |
backtrack: ‹*dpll.dpll-backtrack S T* ⟹ $dpll_W$*-core-stgy S T*› |
backtrack-opt: ‹*backtrack-opt S T* ⟹ $dpll_W$*-core-stgy S T*›

lemma $dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹$dpll_W$*-core-stgy S T* ⟹ $dpll_W$*-core S T*›
⟨*proof*⟩

lemma *rtranclp-*$dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹$dpll_W$*-core-stgy*** *S T* ⟹ $dpll_W$*-core*** *S T*›
⟨*proof*⟩

lemma *no-step-stgy-iff*: ‹*no-step* $dpll_W$*-core-stgy S* ⟷ *no-step* $dpll_W$*-core S*›
⟨*proof*⟩

lemma *full-*$dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹*full* $dpll_W$*-core-stgy S T* ⟹ *full* $dpll_W$*-core S T*›
⟨*proof*⟩

lemma $dpll_W$*-core-stgy-clauses*:
‹$dpll_W$*-core-stgy S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

lemma *rtranclp-*$dpll_W$*-core-stgy-clauses*:
‹$dpll_W$*-core-stgy*** *S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

end

end
theory *DPLL-W-Optimal-Model*
imports
DPLL-W-BnB
begin

locale $dpll_W$*-state-optimal-weight* =
$dpll_W$*-state trail clauses*
tl-trail cons-trail state-eq state +
ocdcl-weight ϱ
for
trail :: ‹'*st* ⇒ '*v* $dpll_W$*-ann-lits*› **and**
clauses :: ‹'*st* ⇒ '*v clauses*› **and**
tl-trail :: ‹'*st* ⇒ '*st*› **and**
cons-trail :: ‹'*v* $dpll_W$*-ann-lit* ⇒ '*st* ⇒ '*st*› **and**
state-eq :: ‹'*st* ⇒ '*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**

state :: ‹'*st* ⇒ '*v* $dpll_W$-*ann-lits* × '*v clauses* × '*v clause option* × '*b*› **and**
ϱ :: ‹'*v clause* ⇒ '*a* :: {*linorder*}› +
fixes
update-additional-info :: ‹'*v clause option* × '*b* ⇒ '*st* ⇒ '*st*›
assumes
update-additional-info:
‹*state S* = (*M*, *N*, *K*) ⟹ *state* (*update-additional-info K' S*) = (*M*, *N*, *K'*)›
begin

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

lemma [*simp*]:
‹*trail* (*update-weight-information M' S*) = *trail S*›
‹*clauses* (*update-weight-information M' S*) = *clauses S*›
‹*clauses* (*update-additional-info c S*) = *clauses S*›
‹*additional-info* (*update-additional-info* (*w*, *oth*) *S*) = (*w*, *oth*)›
⟨*proof*⟩

lemma *state-update-weight-information*: ‹*state S* = (*M*, *N*, *w*, *oth*) ⟹
∃ *w'*. *state* (*update-weight-information M' S*) = (*M*, *N*, *w'*, *oth*)›
⟨*proof*⟩

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

lemma [*simp*]: ‹(*weight* (*update-weight-information M' S*)) = *Some* (*lit-of* '# *mset M'*)›
⟨*proof*⟩

We test here a slightly different decision. In the CDCL version, we renamed *additional-info* from the BNB version to avoid collisions. Here instead of renaming, we add the prefix *bnb*. to every name.

sublocale *bnb*: *bnb-ops* **where**
trail = *trail* **and**
clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state* **and**
weight = *weight* **and**
conflicting-clauses = *conflicting-clauses* **and**
is-improving-int = *is-improving-int* **and**
update-weight-information = *update-weight-information*
⟨*proof*⟩

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

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

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

lemma *state-additional-info'*:
‹*state* S = (*trail* S, *clauses* S, *weight* S, *bnb.additional-info* S)›
⟨*proof*⟩

sublocale *bnb*: *bnb* **where**
trail = *trail* **and**
clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state* **and**
weight = *weight* **and**
conflicting-clauses = *conflicting-clauses* **and**
is-improving-int = *is-improving-int* **and**
update-weight-information = *update-weight-information*
⟨*proof*⟩

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

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

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

lemma *rtranclp-cdcl-bnb-still-model*:

assumes
st: ‹$bnb.dpll_W$*-bnb*** *S T*› **and**
all-struct: ‹$dpll_W$*-all-inv* (*bnb.abs-state S*)› **and**
ent: ‹(*set-mset I* ⊨*sm clauses S* ∧ *set-mset I* ⊨*sm bnb.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* (*clauses S*)›
shows
‹(*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm bnb.conflicting-clss T*) ∨ *Found* (ϱ *I*) ≥ ϱ′ (*weight T*)›
⟨*proof*⟩

lemma *simple-clss-entailed-by-too-heavy-in-conflicting*:
‹*C* ∈# *mset-set* (*simple-clss* (*atms-of-mm* (*clauses S*))) ⟹
too-heavy-clauses (*clauses S*) (*weight S*) ⊨*pm*
(*C*) ⟹ *C* ∈# *bnb.conflicting-clss S*›
⟨*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: ‹(∀ *C* ∈# *bnb.conflicting-clss S*. ¬ *trail S* ⊨*as CNot C*)› **and**
all-struct: ‹$dpll_W$*-all-inv* (*bnb.abs-state S*)›
shows ‹*Ex* ($bnb.dpll_W$*-bound S*)›
⟨*proof*⟩

lemma *no-step-*$dpll_W$*-bnb-conflict*:
assumes
ns: ‹*no-step* $bnb.dpll_W$*-bnb S*› **and**
invs: ‹$dpll_W$*-all-inv* (*bnb.abs-state S*)›
shows ‹∃ *C* ∈# *clauses S* + *bnb.conflicting-clss S*. *trail S* ⊨*as CNot C*› (**is** *?A*) **and**
‹*count-decided* (*trail S*) = *0*› **and**
‹*unsatisfiable* (*set-mset* (*clauses S* + *bnb.conflicting-clss S*))›
⟨*proof*⟩

lemma *full-cdcl-bnb-stgy-larger-or-equal-weight*:
assumes
st: ‹*full* $bnb.dpll_W$*-bnb S T*› **and**
all-struct: ‹$dpll_W$*-all-inv* (*bnb.abs-state S*)› **and**
ent: ‹(*set-mset I* ⊨*sm clauses S* ∧ *set-mset I* ⊨*sm bnb.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* (*clauses S*)›
shows
‹*Found* (ϱ *I*) ≥ ϱ′ (*weight T*)› **and**
‹*unsatisfiable* (*set-mset* (*clauses T* + *bnb.conflicting-clss T*))›
⟨*proof*⟩

end

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

context *optimal-encoding-ops*
begin

We use the following list to generate an upper bound of the derived trails by ODPLL: using lists makes it possible to use recursion. Using *inductive-set* does not work, because it is not possible to recurse on the arguments of a predicate.

The idea is similar to an earlier definition of *simple-clss*, although in that case, we went for recursion over the set of literals directly, via a choice in the recursive call.

definition *list-new-vars* :: ‹*'v list*› **where**
‹*list-new-vars* = (*SOME v. set v* = ΔΣ ∧ *distinct v*)›

lemma
set-list-new-vars: ‹*set list-new-vars* = ΔΣ› **and**
distinct-list-new-vars: ‹*distinct list-new-vars*› **and**
length-list-new-vars: ‹*length list-new-vars* = *card* ΔΣ›
⟨*proof*⟩

fun *all-sound-trails* **where**
‹*all-sound-trails* [] = *simple-clss* (Σ − ΔΣ)› |
‹*all-sound-trails* (*L* # *M*) =
all-sound-trails M ∪ *add-mset* (*Pos* (*replacement-pos L*)) ‘ *all-sound-trails M*
∪ *add-mset* (*Pos* (*replacement-neg L*)) ‘ *all-sound-trails M*›

lemma *all-sound-trails-atms*:
‹*set xs* ⊆ ΔΣ ⟹
C ∈ *all-sound-trails xs* ⟹
atms-of C ⊆ Σ − ΔΣ ∪ *replacement-pos* ‘ *set xs* ∪ *replacement-neg* ‘ *set xs*›
⟨*proof*⟩

lemma *all-sound-trails-distinct-mset*:
‹*set xs* ⊆ ΔΣ ⟹ *distinct xs* ⟹
C ∈ *all-sound-trails xs* ⟹
distinct-mset C›
⟨*proof*⟩

lemma *all-sound-trails-tautology*:
‹*set xs* ⊆ ΔΣ ⟹ *distinct xs* ⟹
C ∈ *all-sound-trails xs* ⟹
¬*tautology C*›
⟨*proof*⟩

lemma *all-sound-trails-simple-clss*:
‹*set xs* ⊆ ΔΣ ⟹ *distinct xs* ⟹
all-sound-trails xs ⊆ *simple-clss* (Σ − ΔΣ ∪ *replacement-pos* ‘ *set xs* ∪ *replacement-neg* ‘ *set xs*)›
⟨*proof*⟩

lemma *in-all-sound-trails-inD*:

‹set xs ⊆ ΔΣ ⟹ distinct xs ⟹ a ∈ ΔΣ ⟹
add-mset (Pos ($a^{\mapsto 0}$)) xa ∈ all-sound-trails xs ⟹ a ∈ set xs›
⟨proof⟩

lemma *in-all-sound-trails-inD′*:
‹set xs ⊆ ΔΣ ⟹ distinct xs ⟹ a ∈ ΔΣ ⟹
add-mset (Pos ($a^{\mapsto 1}$)) xa ∈ all-sound-trails xs ⟹ a ∈ set xs›
⟨proof⟩

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

lemma *all-sound-trails-finite*[*simp*]:
‹finite (all-sound-trails xs)›
⟨proof⟩

lemma *card-all-sound-trails*:
assumes ‹set xs ⊆ ΔΣ› **and** ‹distinct xs›
shows ‹card (all-sound-trails xs) = card (simple-clss (Σ − ΔΣ)) ∗ 3 ^ (length xs)›
⟨proof⟩

end

lemma *simple-clss-all-sound-trails*: ‹simple-clss (Σ − ΔΣ) ⊆ all-sound-trails ys›
⟨proof⟩

lemma *all-sound-trails-decomp-in*:
assumes
‹C ⊆ ΔΣ› ‹C′ ⊆ ΔΣ› ‹C ∩ C′ = {}› ‹C ∪ C′ ⊆ set xs›
‹D ∈ simple-clss (Σ − ΔΣ)›
shows
‹(Pos o replacement-pos) ‘# mset-set C + (Pos o replacement-neg) ‘# mset-set C′ + D ∈ all-sound-trails xs›
⟨proof⟩

lemma (**in** −)*image-union-subset-decomp*:
‹f ‘ (C) ⊆ A ∪ B ⟷ (∃A′ B′. f ‘ A′ ⊆ A ∧ f ‘ B′ ⊆ B ∧ C = A′ ∪ B′ ∧ A′ ∩ B′ = {})›
⟨proof⟩

lemma *in-all-sound-trails*:
assumes
‹⋀L. L ∈ ΔΣ ⟹ Neg (replacement-pos L) ∉# C›
‹⋀L. L ∈ ΔΣ ⟹ Neg (replacement-neg L) ∉# C›
‹⋀L. L ∈ ΔΣ ⟹ Pos (replacement-pos L) ∈# C ⟹ Pos (replacement-neg L) ∉# C›
‹C ∈ simple-clss (Σ − ΔΣ ∪ replacement-pos ‘ set xs ∪ replacement-neg ‘ set xs)› **and**
xs: ‹set xs ⊆ ΔΣ›
shows
‹C ∈ all-sound-trails xs›
⟨proof⟩

end

locale *dpll-optimal-encoding-opt* =
$dpll_W$*-state-optimal-weight trail clauses*

tl-trail cons-trail state-eq state ϱ *update-additional-info* +
optimal-encoding-opt-ops Σ $\Delta\Sigma$ *new-vars*
for
trail :: ‹$'st \Rightarrow 'v$ *dpll*$_W$*-ann-lits*› **and**
clauses :: ‹$'st \Rightarrow 'v$ *clauses*› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
cons-trail :: ‹$'v$ *dpll*$_W$*-ann-lit* $\Rightarrow 'st \Rightarrow 'st$› **and**
state-eq :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› (**infix** ‹∼› *50*) **and**
state :: ‹$'st \Rightarrow 'v$ *dpll*$_W$*-ann-lits* $\times$ $'v$ *clauses* $\times$ $'v$ *clause option* $\times$ $'b$› **and**
update-additional-info :: ‹$'v$ *clause option* $\times$ $'b \Rightarrow 'st \Rightarrow 'st$› **and**
Σ $\Delta\Sigma$ :: ‹$'v$ *set*› **and**
ϱ :: ‹$'v$ *clause* $\Rightarrow 'a$ :: {*linorder*}› **and**
new-vars :: ‹$'v \Rightarrow 'v \times 'v$›
begin

end

locale *dpll-optimal-encoding* =
dpll-optimal-encoding-opt trail clauses
tl-trail cons-trail state-eq state
update-additional-info Σ $\Delta\Sigma$ ϱ *new-vars* +
optimal-encoding-ops
Σ $\Delta\Sigma$
new-vars ϱ
for
trail :: ‹$'st \Rightarrow 'v$ *dpll*$_W$*-ann-lits*› **and**
clauses :: ‹$'st \Rightarrow 'v$ *clauses*› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
cons-trail :: ‹$'v$ *dpll*$_W$*-ann-lit* $\Rightarrow 'st \Rightarrow 'st$› **and**
state-eq :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› (**infix** ‹∼› *50*) **and**
state :: ‹$'st \Rightarrow 'v$ *dpll*$_W$*-ann-lits* $\times$ $'v$ *clauses* $\times$ $'v$ *clause option* $\times$ $'b$› **and**
update-additional-info :: ‹$'v$ *clause option* $\times$ $'b \Rightarrow 'st \Rightarrow 'st$› **and**
Σ $\Delta\Sigma$ :: ‹$'v$ *set*› **and**
ϱ :: ‹$'v$ *clause* $\Rightarrow 'a$ :: {*linorder*}› **and**
new-vars :: ‹$'v \Rightarrow 'v \times 'v$›
begin

inductive *odecide* :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› **where**
odecide-noweight: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) *L*› **and**
‹*atm-of L* ∈ *atms-of-mm* (*clauses S*)› **and**
‹*T* ∼ *cons-trail* (*Decided L*) *S*› **and**
‹*atm-of L* ∈ $\Sigma - \Delta\Sigma$› |
odecide-replacement-pos: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-pos L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-pos L*))) *S*› **and**
‹*L* ∈ $\Delta\Sigma$› |
odecide-replacement-neg: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-neg L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-neg L*))) *S*› **and**
‹*L* ∈ $\Delta\Sigma$›

inductive-cases *odecideE*: ‹*odecide S T*›

inductive *dpll-conflict* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **where**
‹*dpll-conflict S S*›
if ‹*C* ∈# *clauses S*› **and**
‹*trail S* ⊨*as CNot C*›

inductive $odpll_W$*-core-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S T* **where**
propagate: ‹*dpll-propagate S T* ⟹ $odpll_W$*-core-stgy S T*› |
decided: ‹*odecide S T* ⟹ *no-step dpll-propagate S* ⟹ $odpll_W$*-core-stgy S T* › |
backtrack: ‹*dpll-backtrack S T* ⟹ $odpll_W$*-core-stgy S T*› |
backtrack-opt: ‹*bnb.backtrack-opt S T* ⟹ $odpll_W$*-core-stgy S T*›

lemma $odpll_W$*-core-stgy-clauses*:
‹$odpll_W$*-core-stgy S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

lemma *rtranclp-*$odpll_W$*-core-stgy-clauses*:
‹$odpll_W$*-core-stgy*** *S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

inductive $odpll_W$*-bnb-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S T* :: *'st* **where**
dpll:
‹$odpll_W$*-bnb-stgy S T*›
if ‹$odpll_W$*-core-stgy S T*› |
bnb:
‹$odpll_W$*-bnb-stgy S T*›
if ‹*bnb.*$dpll_W$*-bound S T*›

lemma $odpll_W$*-bnb-stgy-clauses*:
‹$odpll_W$*-bnb-stgy S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

lemma *rtranclp-*$odpll_W$*-bnb-stgy-clauses*:
‹$odpll_W$*-bnb-stgy*** *S T* ⟹ *clauses T* = *clauses S*›
⟨*proof*⟩

lemma *odecide-dpll-decide-iff*:
assumes ‹*clauses S* = *penc N*› ‹*atms-of-mm N* = Σ›
shows ‹*odecide S T* ⟹ *dpll-decide S T*›
‹*dpll-decide S T* ⟹ *Ex*(*odecide S*)›
⟨*proof*⟩

lemma
assumes ‹*clauses S* = *penc N*› ‹*atms-of-mm N* = Σ›
shows
$odpll_W$*-core-stgy-*$dpll_W$*-core-stgy*: ‹$odpll_W$*-core-stgy S T* ⟹ *bnb.*$dpll_W$*-core-stgy S T*›
⟨*proof*⟩

lemma
assumes ‹*clauses S* = *penc N*› ‹*atms-of-mm N* = Σ›
shows
$odpll_W$*-bnb-stgy-*$dpll_W$*-bnb-stgy*: ‹$odpll_W$*-bnb-stgy S T* ⟹ *bnb.*$dpll_W$*-bnb S T*›
⟨*proof*⟩

lemma
assumes ‹*clauses* S = *penc* N› **and** [*simp*]: ‹*atms-of-mm* $N = \Sigma$›
shows
rtranclp-odpll$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*: ‹*odpll*$_W$*-bnb-stgy*** S $T \Longrightarrow$ *bnb.dpll*$_W$*-bnb*** S T›
⟨*proof*⟩

lemma *no-step-odpll*$_W$*-core-stgy-no-step-dpll*$_W$*-core-stgy*:
assumes ‹*clauses* S = *penc* N› **and** [*simp*]:‹*atms-of-mm* $N = \Sigma$›
shows
‹*no-step odpll*$_W$*-core-stgy* $S \longleftrightarrow$ *no-step bnb.dpll*$_W$*-core-stgy* S›
⟨*proof*⟩

lemma *no-step-odpll*$_W$*-bnb-stgy-no-step-dpll*$_W$*-bnb*:
assumes ‹*clauses* S = *penc* N› **and** [*simp*]:‹*atms-of-mm* $N = \Sigma$›
shows
‹*no-step odpll*$_W$*-bnb-stgy* $S \longleftrightarrow$ *no-step bnb.dpll*$_W$*-bnb* S›
⟨*proof*⟩

lemma *full-odpll*$_W$*-core-stgy-full-dpll*$_W$*-core-stgy*:
assumes ‹*clauses* S = *penc* N› **and** [*simp*]:‹*atms-of-mm* $N = \Sigma$›
shows
‹*full odpll*$_W$*-bnb-stgy* S $T \Longrightarrow$ *full bnb.dpll*$_W$*-bnb* S T›
⟨*proof*⟩

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

lemma *no-step-dpll-backtrack-iff*:
‹*no-step dpll-backtrack* $S \longleftrightarrow$ (*count-decided* (*trail* S) = *0* $\vee$ ($\forall C \in\#$ *clauses* S. $\neg$*trail* $S \models$*as CNot* C))›
⟨*proof*⟩

lemma *no-step-dpll-conflict*:
‹*no-step dpll-conflict* $S \longleftrightarrow$ ($\forall C \in\#$ *clauses* S. $\neg$*trail* $S \models$*as CNot* C)›
⟨*proof*⟩

definition *no-smaller-propa* :: ‹$'st \Rightarrow$ *bool*› **where**
no-smaller-propa (S ::$'st$) $\longleftrightarrow$
($\forall M\ K\ M'\ D\ L.$ *trail* S = M' @ *Decided* K # $M \longrightarrow$ *add-mset* L $D \in\#$ *clauses* $S \longrightarrow$ *undefined-lit* M $L \longrightarrow \neg M \models$*as CNot* D)

lemma [*simp*]: ‹$T \sim S \Longrightarrow$ *no-smaller-propa* T = *no-smaller-propa* S›
⟨*proof*⟩

lemma *no-smaller-propa-cons-trail*[*simp*]:
‹*no-smaller-propa* (*cons-trail* (*Propagated* L C) S) $\longleftrightarrow$ *no-smaller-propa* S›
‹*no-smaller-propa* (*update-weight-information* M' S) $\longleftrightarrow$ *no-smaller-propa* S›
⟨*proof*⟩

lemma *no-smaller-propa-cons-trail-decided*[*simp*]:
‹*no-smaller-propa* $S \Longrightarrow$ *no-smaller-propa* (*cons-trail* (*Decided* L) S) $\longleftrightarrow$ ($\forall L\ C.$ *add-mset* L $C \in\#$

clauses S ⟶ *undefined-lit* (*trail S*)*L* ⟶ ¬*trail S* ⊨*as CNot C*)›
⟨*proof*⟩

lemma *no-step-dpll-propagate-iff*:
‹*no-step dpll-propagate S* ⟷ (∀ *L C*. *add-mset L C* ∈# *clauses S* ⟶ *undefined-lit* (*trail S*)*L* ⟶ ¬*trail S* ⊨*as CNot C*)›
⟨*proof*⟩

lemma *count-decided-0-no-smaller-propa*: ‹*count-decided* (*trail S*) = *0* ⟹ *no-smaller-propa S*›
⟨*proof*⟩

lemma *no-smaller-propa-backtrack-split*:
‹*no-smaller-propa S* ⟹
backtrack-split (*trail S*) = (*M'*, *L* # *M*) ⟹
no-smaller-propa (*reduce-trail-to M S*)›
⟨*proof*⟩

lemma $odpll_W$*-core-stgy-no-smaller-propa*:
‹$odpll_W$*-core-stgy S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›
⟨*proof*⟩

lemma $odpll_W$*-bound-stgy-no-smaller-propa*: ‹*bnb.*$dpll_W$*-bound S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›
⟨*proof*⟩

lemma $odpll_W$*-bnb-stgy-no-smaller-propa*:
‹$odpll_W$*-bnb-stgy S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›
⟨*proof*⟩

lemma *filter-disjount-union*:
‹(⋀*x*. *x* ∈ *set xs* ⟹ *P x* ⟹ ¬*Q x*) ⟹
length (*filter P xs*) + *length* (*filter Q xs*) =
length (*filter* (λ*x*. *P x* ∨ *Q x*) *xs*)›
⟨*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 *isabelle-should-do-that-automatically*: ‹*Suc* (*a* − *Suc 0*) = *a* ⟷ *a* ≥ *1*›
⟨*proof*⟩
lemma *distinct-count-list-if*: ‹*distinct xs* ⟹ *count-list xs x* = (*if x* ∈ *set xs then 1 else 0*)›
⟨*proof*⟩

abbreviation (*input*) *cut-and-complete-trail* :: ‹*'st* ⇒ -› **where**
‹*cut-and-complete-trail S* ≡ *trail S*›

inductive *odpll$_W$-core-stgy-count* :: ‹'*st* × - ⇒ '*st* × - ⇒ *bool*› **where**
propagate: ‹*dpll-propagate S T* ⟹ *odpll$_W$-core-stgy-count* (*S*, *C*) (*T*, *C*)› |
decided: ‹*odecide S T* ⟹ *no-step dpll-propagate S* ⟹ *odpll$_W$-core-stgy-count* (*S*, *C*) (*T*, *C*) › |
backtrack: ‹*dpll-backtrack S T* ⟹ *odpll$_W$-core-stgy-count* (*S*, *C*) (*T*, *add-mset* (*cut-and-complete-trail S*) *C*)› |
backtrack-opt: ‹*bnb.backtrack-opt S T* ⟹ *odpll$_W$-core-stgy-count* (*S*, *C*) (*T*, *add-mset* (*cut-and-complete-trail S*) *C*)›

inductive *odpll$_W$-bnb-stgy-count* :: ‹'*st* × - ⇒ '*st* × - ⇒ *bool*› **where**
dpll:
‹*odpll$_W$-bnb-stgy-count S T*›
if ‹*odpll$_W$-core-stgy-count S T*› |
bnb:
‹*odpll$_W$-bnb-stgy-count* (*S*, *C*) (*T*, *C*)›
if ‹*bnb.dpll$_W$-bound S T*›

lemma *odpll$_W$-core-stgy-countD*:
‹*odpll$_W$-core-stgy-count S T* ⟹ *odpll$_W$-core-stgy* (*fst S*) (*fst T*)›
‹*odpll$_W$-core-stgy-count S T* ⟹ *snd S* ⊆# *snd T*›
⟨*proof*⟩

lemma *odpll$_W$-bnb-stgy-countD*:
‹*odpll$_W$-bnb-stgy-count S T* ⟹ *odpll$_W$-bnb-stgy* (*fst S*) (*fst T*)›
‹*odpll$_W$-bnb-stgy-count S T* ⟹ *snd S* ⊆# *snd T*›
⟨*proof*⟩

lemma *rtranclp-odpll$_W$-bnb-stgy-countD*:
‹*odpll$_W$-bnb-stgy-count*** *S T* ⟹ *odpll$_W$-bnb-stgy*** (*fst S*) (*fst T*)›
‹*odpll$_W$-bnb-stgy-count*** *S T* ⟹ *snd S* ⊆# *snd T*›
⟨*proof*⟩

lemmas *odpll$_W$-core-stgy-count-induct* = *odpll$_W$-core-stgy-count.induct*[*of* ‹(*S*, *n*)› ‹(*T*, *m*)› **for** *S n T m*, *split-format*(*complete*), *OF dpll-optimal-encoding-axioms*,
consumes 1]

definition *conflict-clauses-are-entailed* :: ‹'*st* × - ⇒ *bool*› **where**
‹*conflict-clauses-are-entailed* =
(λ(*S*, *Cs*). ∀ *C* ∈# *Cs*. (∃ *M' K M M''*. *trail S* = *M'* @ *Propagated K* () # *M* ∧ *C* = *M''* @ *Decided* (−*K*) # *M*))›

definition *conflict-clauses-are-entailed2* :: ‹'*st* × ('*v literal*, '*v literal*, *unit*) *annotated-lits multiset* ⇒ *bool*› **where**
‹*conflict-clauses-are-entailed2* =
(λ(*S*, *Cs*). ∀ *C* ∈# *Cs*. ∀ *C'* ∈# *remove1-mset C Cs*. (∃ *L*. *Decided L* ∈ *set C* ∧ *Propagated* (−*L*) () ∈ *set C'*) ∨
(∃ *L*. *Propagated* (*L*) () ∈ *set C* ∧ *Decided* (−*L*) ∈ *set C'*))›

lemma *propagated-cons-eq-append-propagated-cons*:
‹*Propagated L* () # *M* = *M'* @ *Propagated K* () # *Ma* ⟷
(*M'* = [] ∧ *K* = *L* ∧ *M* = *Ma*) ∨
(*M'* ≠ [] ∧ *hd M'* = *Propagated L* () ∧ *M* = *tl M'* @ *Propagated K* () # *Ma*)›
⟨*proof*⟩

lemma *odpll*$_W$*-core-stgy-count-conflict-clauses-are-entailed*:
assumes
‹*odpll*$_W$*-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*›
shows
‹*conflict-clauses-are-entailed T*›
⟨*proof*⟩

lemma *odpll*$_W$*-bnb-stgy-count-conflict-clauses-are-entailed*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*›
shows
‹*conflict-clauses-are-entailed T*›
⟨*proof*⟩

lemma *odpll*$_W$*-core-stgy-count-no-dup-clss*:
assumes
‹*odpll*$_W$*-core-stgy-count S T*› **and**
‹∀ *C* ∈# *snd S*. *no-dup C*› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹∀ *C* ∈# *snd T*. *no-dup C*›
⟨*proof*⟩

lemma *odpll*$_W$*-bnb-stgy-count-no-dup-clss*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹∀ *C* ∈# *snd S*. *no-dup C*› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹∀ *C* ∈# *snd T*. *no-dup C*›
⟨*proof*⟩

lemma *backtrack-split-conflict-clauses-are-entailed-itself*:
assumes
‹*backtrack-split* (*trail S*) = (*M'*, *L* # *M*)› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state S*)›
shows ‹¬ *conflict-clauses-are-entailed*
(*S*, *add-mset* (*trail S*) *C*)› (**is** ‹¬ *?A*›)
⟨*proof*⟩

lemma *odpll*$_W$*-core-stgy-count-distinct-mset*:
assumes
‹*odpll*$_W$*-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*distinct-mset* (*snd T*)›
⟨*proof*⟩

lemma *odpll$_W$-bnb-stgy-count-distinct-mset*:
assumes
‹*odpll$_W$-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll$_W$-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*distinct-mset* (*snd T*)›
⟨*proof*⟩

lemma *odpll$_W$-core-stgy-count-conflict-clauses-are-entailed2*:
assumes
‹*odpll$_W$-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll$_W$-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*conflict-clauses-are-entailed2 T*›
⟨*proof*⟩

lemma *odpll$_W$-bnb-stgy-count-conflict-clauses-are-entailed2*:
assumes
‹*odpll$_W$-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll$_W$-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*conflict-clauses-are-entailed2 T*›
⟨*proof*⟩

definition *no-complement-set-lit* :: ‹*'v dpll$_W$-ann-lits* ⇒ *bool*› **where**
‹*no-complement-set-lit M* ⟷
(∀ *L* ∈ ΔΣ. *Decided* (*Pos* (*replacement-pos L*)) ∈ *set M* ⟶ *Decided* (*Pos* (*replacement-neg L*)) ∉ *set M*) ∧
(∀ *L* ∈ ΔΣ. *Decided* (*Neg* (*replacement-pos L*)) ∉ *set M*) ∧
(∀ *L* ∈ ΔΣ. *Decided* (*Neg* (*replacement-neg L*)) ∉ *set M*) ∧
atm-of ‘ *lits-of-l M* ⊆ Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›

definition *no-complement-set-lit-st* :: ‹*'st* × *'v dpll$_W$-ann-lits multiset* ⇒ *bool*› **where**
‹*no-complement-set-lit-st* = (λ(*S*, *Cs*). (∀ *C*∈#*Cs*. *no-complement-set-lit C*) ∧ *no-complement-set-lit* (*trail S*))›

lemma *backtrack-no-complement-set-lit*: ‹*no-complement-set-lit* (*trail S*) ⟹
backtrack-split (*trail S*) = (*M'*, *L* # *M*) ⟹
no-complement-set-lit (*Propagated* (− *lit-of L*) () # *M*)›
⟨*proof*⟩

lemma *odpll$_W$-core-stgy-count-no-complement-set-lit-st*:
assumes
‹*odpll$_W$-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**

‹*distinct-mset* (*snd S*)› **and**
invs: ‹$dpll_W$*-all-inv* (*bnb.abs-state* (*fst S*))› **and**
‹*no-complement-set-lit-st S*› **and**
atms: ‹*clauses* (*fst S*) = *penc N*› ‹*atms-of-mm N* = Σ› **and**
‹*no-smaller-propa* (*fst S*)›
shows
‹*no-complement-set-lit-st T*›
⟨*proof*⟩

lemma *odpll*$_W$*-bnb-stgy-count-no-complement-set-lit-st*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹$dpll_W$*-all-inv* (*bnb.abs-state* (*fst S*))› **and**
‹*no-complement-set-lit-st S*› **and**
atms: ‹*clauses* (*fst S*) = *penc N*› ‹*atms-of-mm N* = Σ› **and**
‹*no-smaller-propa* (*fst S*)›
shows
‹*no-complement-set-lit-st T*›
⟨*proof*⟩

definition *stgy-invs* :: ‹'*v clauses* ⇒ '*st* × - ⇒ *bool*› **where**
‹*stgy-invs N S* ⟷
no-smaller-propa (*fst S*) ∧
conflict-clauses-are-entailed S ∧
conflict-clauses-are-entailed2 S ∧
distinct-mset (*snd S*) ∧
(∀ *C* ∈# *snd S*. *no-dup C*) ∧
$dpll_W$*-all-inv* (*bnb.abs-state* (*fst S*)) ∧
no-complement-set-lit-st S ∧
clauses (*fst S*) = *penc N* ∧
atms-of-mm N = Σ
›

lemma *odpll*$_W$*-bnb-stgy-count-stgy-invs*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*stgy-invs N S*›
shows ‹*stgy-invs N T*›
⟨*proof*⟩

lemma *stgy-invs-size-le*:
assumes ‹*stgy-invs N S*›
shows ‹*size* (*snd S*) ≤ *3* ^ (*card* Σ)›
⟨*proof*⟩

lemma *rtranclp-odpll*$_W$*-bnb-stgy-count-stgy-invs*: ‹*odpll*$_W$*-bnb-stgy-count*** *S T* ⟹ *stgy-invs N S* ⟹ *stgy-invs N T*›
⟨*proof*⟩

theorem
assumes ‹*clauses S* = *penc N*› ‹*atms-of-mm N* = Σ› **and**
‹*odpll*$_W$*-bnb-stgy-count*** (*S*, {#}) (*T*, *D*)› **and**
tr: ‹*trail S* = []›

shows ‹*size D* $\leq$ *3* ^ (*card* Σ)›
⟨*proof*⟩

end

end