A Verified SAT Solver Framework including Optimization and Partial Valuations

Mathias Fleury\textsuperscript{1,2}, Christoph Weidenbach\textsuperscript{1}, and Dominic Zimmer\textsuperscript{1,3}

\textsuperscript{1} Max-Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
\{mathias.fleury,weidenb\}@mpi-inf.mpg.de

\textsuperscript{2} Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
s8mafleu@stud.uni-saarland.de

\textsuperscript{3} Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
s8dozimm@stud.uni-saarland.de

Abstract. Based on our formal framework for CDCL (conflict-driven clause learning) verified using the proof assistant Isabelle/HOL, we verify an optimizing calculus based on branch and bound, OCDCL. It finds optimal models of minimum cost with respect to total valuations. Through the dual rail encoding, we reduce the search for partial optimal models to total optimal models, as derived by OCDCL. OCDCL can also be used to solve MAXSAT. The developed CDCL with branch and bound is kept abstract allowing us to express the search for model covering. A large part of our original CDCL framework could be reused without changes to reduce the complexity of the new formalization. To the best of our knowledge, this is the first rigorous formalization of a CDCL solver computing cost optimal models and the first solution to compute cost-optimal models with respect to partial valuations.

1 Introduction

Researchers in automated reasoning spend a significant portion of their work time specifying calculi and proving metatheorems about them. These proofs are mostly carried out with pen and paper, which is error-prone and can be tedious. Especially when working on variants of previously devised calculi, it is easy to miss subtle but important differences. We are part an effort, called IsaFoL (Isabelle Formalization of Logic) \cite{isafo1} that aims at developing libraries and methodologies to formalize these calculi using the Isabelle/HOL proof assistant \cite{isabelle}. This does not only include developing formal companions to paper proof but also includes simplifying the verification of variants of earlier devised calculi.

We have previously formalized propositional satisfiability (SAT) \cite{fleury2018}, based on Weidenbach’s account of conflict-driven clause learning (CDCL). An important extension to CDCL is optimizing propositional satisfiability (OPT-SAT): Given a cost function on literals, OPT-SAT aims at finding an optimal-cost model for a set of clauses. In Weidenbach’s upcoming textbook tentatively called \textit{Automated
Reasoning–The Art of Generic Problem Solving, he developed an optimizing CDCL for OPT-SAT called OCDCL. It is based on his CDCL calculus and the correctness is proved by copy-pasting the CDCL proofs. As a case study, we tried to formalize it. When doing so, we realized that the calculus did not find optimal models if partial valuations are considered: The typical CDCL-learned-clause mechanism in the context of searching for (optimal) models does not apply with respect to partial valuations. The only way we found to fix this problem was a restriction to optimal models based on total valuations. The developed OCDCL calculus is similar to Larrosa et al.'s DPLL\_BB calculus [11], although it is not clear whether the authors are aware that their calculus is only correct for models with respect to total valuations. We took this as a further motivation for both formalizing OCDCL on total valuations, as well as for finding an encoding such that OCDCL is also correct with respect to partial valuations.

Our contributions are first a rigorous proof of the correctness of OCDCL with respect to total valuations and second a rigorous proof that OCDCL is also correct with respect to partial valuations on the dual rail encoding [6,18]. Formalization and verifications in proof assistants are often justified by the idea that they can be reused and extended. This is exactly what we are doing here. In the formalization, we could find and eventually solve the problem of finding optimal models with respect to the two types of valuations. The OCDCL formalization amounts to around 3300 lines of proof. This is small compared to the 2600 lines of shared libraries, 6000 lines for the formalization of Weidenbach's account of CDCL [4], and 4500 lines for Nieuwenhuis et al.'s account [15]. The formalization took around 1.5 month work. The extension to partial valuations amounts to 1300 lines.

Now the paper is organized as follows. After some preliminaries on Isabelle and the previous formalization of CDCL (Section 2), the optimizing calculus is introduced (Section 3). It is described as an abstract non-deterministic transition system and has the conflict analysis for the first unique implication point built-in. Unlike other variants that have been considered, it is possible to have different nonzero weights for a literal and its negation (similarly to how in real life the opposite of “taking the GPS in a car” can be “a cache is required for the cable”, both options having a nonzero cost).

OCDCL is well suited for a formalization as its core rules are exactly the rule of the calculus we have formalized earlier. Our formalization tries to reuse as much from our previous formalization as possible and especially tries to avoid copy-paste, thanks to the abstractions developed earlier. We develop an abstract calculus with branch and bounds, CDCL\_BnB, that we instantiate with weights. Finally, to overcome the limitation of our calculus, we show an encoding that reduces finding a partial optimal model into a total optimal model (Section 5), which was also formalized in Isabelle (Section 6). MAX-SAT can be solved by OCDCL (Section 7). Finally, we use instantiate the abstract CDCL\_BnB differently to find a set of covering models (Section 8).

In the paper we will refer to theorem names with the notation “\texttt{fact 2}” from the formalization: It is a link to the documentation of our formalization.
2 Formalization of CDCL in Isabelle

Isabelle [16, 17] is a generic proof assistant that supports many object logics. The metalogic is an intuitionistic fragment of higher-order logic (HOL) [7]. The types are built from type variables ’a, ’b, . . . and n-ary type constructors, normally written in postfix notation (e.g., ’a list). The infix type constructor ’a ⇒ ’b is interpreted as the (total) function space from ’a to ’b. Function applications are written in a curried style (e.g., f x y). Anonymous functions x ↦ y are written λx. y. The judgment t :: τ indicates that term t has type τ.

Isabelle adheres to the tradition initiated in the 1970s by the LCF system [9]: All inferences are derived by a small trusted kernel; types and functions are defined rather than axiomatized to guard against inconsistencies. High-level specification mechanisms let us define important classes of types and functions, notably inductive predicates and recursive functions. Internally, the system synthesizes appropriate low-level definitions.

Isabelle developments are organized as collections of theory files, or modules, that build on one another. Each file consists of definitions, lemmas, and proofs expressed in Isar, Isabelle’s input language. Proofs are specified either as a sequence of tactics that manipulate the proof state directly or in a declarative, natural deduction format. Our formalization almost exclusively employs the more readable declarative style.

Isabelle locales are a convenient mechanism for structuring large proofs. A locale fixes types, constants, and assumptions within a specified scope. For example, the following declares a locale:

```
locale X = fixes c :: τ_a assumes A_a, c
```

The definition of locale X implicitly fixes a type ’a, explicitly fixes a constant c whose type τ_a may depend on ’a, and states an assumption A_a, c :: prop over ’a and c. Definitions made within the locale may depend on ’a and c, and lemmas proved within the locale may additionally depend on A_a, c. A single locale can introduce several types, constants, and assumptions. Seen from the outside, the lemmas proved in X are polymorphic in type variable ’a, universally quantified over c, and conditional on A_a, c.

Locales support inheritance, union, and instantiations. To instantiate X, we must provide definitions of the types and constants of X together with proofs of X’s assumptions. The command `interpretation X where c = c'` emits the proof obligation A_v,t. After the proof, all the lemmas proved in X become available in Y, with ’a and c :: τ_a instantiated with v and t :: τ_v.

CDCL We have previously developed a framework to formalize the conflict-driven-clause-learning procedure [5] in Isabelle. Clauses are defined as (finite) multisets. For readability, we will write ⊥ and A ∨ B instead of their Isabelle counterpart { # } and A + B. Given a literal L and I is a set of literals, we define entailment by I ⊨ L iff L ∈ I. We can lift it to clauses (multiset of literals) by I ⊨ C iff there is a literal L ∈ C such that I ⊨ L. We can also lift it to clause sets I ⊨ N iff ∀C ∈ N. I ⊨ C.
The conflict-driven clause learning is a procedure that builds a candidate model, called the trail or $M$. It contains literals that have either been decided ($L^\dagger$) or propagated ($L^C$). Each time a clause is not satisfied by the trail, CDCL analyzes the clauses to adapt the trail and learns a new clause to avoid running into the same dead end again. CDCL is presented as a non-deterministic transition system. It operates over a tuple $(M, N, U, D)$ where $N$ are the initial clauses, $U$ are learned clauses, and $D$ is either a conflicting clause that is currently analyzed or $\top$. For example, one of the rules is the Decide rule that extends $M$ by an arbitrary choice $L$:

**Decide** $(M; N; U; \top) \Rightarrow (ML^\dagger; N; U; \top)$

if $L$ is undefined in $M$ and atom $L \in N$.

Instead of a tuple, the formalization uses abstract states of type 'st associated with selectors to access the different components of the states. For example, the Decide rule is actually defined in the following way:

```plaintext
inductive decide :: 'st ⇒ 'st ⇒ bool where
  undefined_lit (trail S) L ⇒ L ∈ atom (clauses S) ⇒
  S' ∼ append_trail (Decided L) S ⇒
  decide S S'
```

where trail $S$ and clauses $S$ selects $M$ and $N + U$, and append_trail $L$ $S$ appends the annotated literal $L$ to the trail without changing the other components. Due to the handling this indirection between the real concrete representation and the fields that can be accessed, we use a different equality: $S ∼ T$ holds if all components are equals. The state and its selectors are defined in a locale specifying their behavior. The initial motivation was to abstract over the current representation in order to be able to refine the state further with more complicated data structures like watch lists.

To simplify the notations, we will use the notation with tuples $(M, N, U, D)$ instead of referring to each component with the selectors.

### 3 Optimizing Conflict-Driven Clause Learning

We assume a total cost function cost on the set of all literals $\text{Lit}(\Sigma)$ over $\Sigma$ into $\mathbb{A}$, $\text{cost} : \text{Lit}(\Sigma) \rightarrow \mathbb{A}$. $\mathbb{A}$ is composed of positive values (e.g. natural numbers, or positive rational or reals). It can be extended to a pair of a literal and a partial valuation by $\text{cost}(L, A) := \text{cost}(L)$ if $A \models L$ and $\text{cost}(L, A) := 0$ if $L$ is not defined. The function can be extended to (partial) valuations by $\text{cost}(A) = \sum_{L \in \text{Lit}(\Sigma)} \text{cost}(A, L)$. We identify partial valuations with consistent sequences (like trails) $M = [L_1 \ldots L_n]$ of literals. A valuation $I$ is total over clauses $N$ when all atoms of $N$ is defined in $I$.

The optimizing conflict-driven clause learning calculus (OCDCL) solves the weighted SAT problem on total valuations. Compared to a normal CDCL state, a component $O$ is added. It either stores the best model so far or $\top$. We extend the cost function to $\top$ by defining $\text{cost}(\top) = \infty$ (i.e., $\top$ is the worst possible).
OCDCL is composed of a CDCL backbone and additional rules to take the weight into account. It employs the same rules as Weidenbach’s account of CDCL [23], except for the additional component \( O \) that is ignored by the CDCL specific rules. The level of a literal is the number of decisions left of its atom in the trail \( M \). We lift the definition to clauses, by defining the level of a clause as the maximum of the levels of its literals or 0 if it is empty.

First, there are three rules involving the last component \( O \) that implement a branch-and-bound approach on the models:

**Improve** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (M;N;U;\top;O) \)

provided \( M \models N, M \) is total over \( N \) and \( \text{cost}(M) < \text{cost}(O) \)

**ConflOpt** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (M;N;U;\neg M;O) \)

provided \( O \neq \top \) and \( \text{cost}(M) \geq \text{cost}(O) \)

**Prune** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (M;N;U;\neg M;O) \)

provided for all total trail extensions \( MM' \) of \( M \), \( \text{cost}(MM') \geq \text{cost}(O) \)

The Prune is not necessary for the correctness and completeness. In practice, Prune would be an integral part of any optimizing solver where a lower-bound on the cost of all extensions of \( M \) is kept to provide an efficient implementation.

After that, the other rules are the standard CDCL rules. They simply ignore the the additional component \( O \). The first few rules use either expand the trail or identify conflicts between the trail and the set of clauses:

**Propagate** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (ML^{\text{CvL}};N;U;\top;O) \)

provided \( C \cup L \in N \cup U, M \models \neg C, L \) is undefined in \( M \)

**Decide** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (ML^1;N;U;\top;O) \)

provided \( L \) is undefined in \( M \), contained in \( N \)

**ConflSat** \( (M;N;U;\top;O) \implies_{\text{OCDCL}} (M;N;U;D;O) \)

provided \( D \in N \cup U \) and \( M \models \neg D \)

Once a conflict is found, it is analyzed to derive a new clause that is the first unique implication point. These two rules do not change either compared to their CDCL counterpart:

**Skip** \( (ML^{\text{CvL}};N;U;D;O) \implies_{\text{OCDCL}} (M;N;U;D;O) \)

provided \( D \notin \{\top, \bot\} \) and \( \neg L \) does not occur in \( D \)

**Resolve** \( (ML^{\text{CvL}};N;U;D \vee \text{comp}(L);O) \implies_{\text{OCDCL}} (M;N;U;D \vee C;O) \)

provided \( D \) is of level \( k \), where \( k \) is the number of decisions in \( M \)

**Backtrack** \( (M_1K^1M_2;N;U;D \vee L;O) \implies_{\text{OCDCL}} (M_1L^{\text{DvL}};N;U \cup \{D \vee L\};\top;O) \)

provided \( L \) is of level \( k \) and \( D \) and \( K \) are of level \( i < k \)

The typical CDCL-learned-clause mechanism in the context of searching for (optimal) models does not apply with respect to partial valuations. Consider the clause set \( N = \{P \vee Q\} \) and cost function \( \text{cost}(P) = 3, \text{cost}(\neg P) = \text{cost}(Q) = \text{cost}(\neg Q) = 1. \) An optimal-cost model based on total valuations is \([\neg P, Q]\) at overall cost 2, whereas an optimal-cost model based on partial valuations is
just \([Q]\) at cost 1. The cost of undefined variables is always considered to be 0. Now the run of an optimizing branch-and-bound CDCL framework may start by deciding \([P^*]\) and detect that this is already a model for \(N\). Hence, it learns \(\neg P\) and establishes 3 as the best current bound on an optimal-cost model. After backtracking, it can propagate \(Q\) with trail \([\neg P \, \neg P \, P \lor Q]\) resulting in a model of cost 2 learning the clause \(P \lor \neg Q\). The resulting clause set \([P \lor Q, \neg P, P \lor \neg Q]\) is unsatisfiable and hence 2 is considered to be the cost-optimal result. The issue is that although this CDCL run already stopped as soon as a partial valuation (trail) is a model for the clause set, it does not compute the optimal result with respect to partial valuations. From the existence of a model with respect to a partial valuation \([P]\) we must not conclude the clause \(\neg P\), because \(P\) could be undefined.

**Definition 1 (Reasonable OCDCL Strategy).** A OCDCL strategy is reasonable if ConflSat is preferred over ConflOpt is preferred over Improve is preferred over Propagate which is preferred over the remaining rules.

**Lemma 2 (OCDCL Termination, \(\text{wf}_{\text{ocdcl}}\).** OCDCL with a reasonable strategy terminates in a state \((M; N; U; \bot; O)\).

**Proof.** Assuming the state is well-formed, the following function is a measure for OCDCL:

\[
\mu((M; N; U; D; O)) = \begin{cases} 
3^n - 1 - |U|, 1, n - |M|, \text{cost}(O), D = \top \\
3^n - 1 - |U|, 0, |M|, \text{cost}(O), \text{otherwise}
\end{cases}
\]

It is decreasing for the lexicographic order. The hardest part of the proof is the decrease when Backjump: \(3^n - 1 - |U| < 3^n - 1 - |U\) is decreasing since no clause is relearned. The proof is similar to the one for CDCL. \(\square\)

**Theorem 3 (OCDCL Correctness, \(\text{full}_{\text{ocdcl}}\) \(\text{stgy}_{\text{no conflicting clause from init state}}\).** OCDCL with a reasonable strategy starting from a state \((\epsilon; N; \emptyset; 0; \top; \epsilon)\) terminates in a state \((M; N; U; 0; \bot; O)\). If \(O = \epsilon\) then \(N\) is unsatisfiable. If \(O \neq \epsilon\) then \(O \models N\) and for any other total model \(M'\) with \(M' \models N\) it holds \(\text{cost}(M') \geq \text{cost}(O)\).

The rule Improve can actually be generalized to situations where \(M\) is not total, but all literals with weights have been set.

**Improve**

\[
\text{Improve}^+ (M; N; U; \top; O) \implies_{\text{OCDCL}} (M; N; U; \top; MM')
\]

provided \(M \models N\), \(MM'\) is a total extension, \(\text{cost}(M) < \text{cost}(O)\), and for any total extension \(MM''\) of the trail, it holds \(\text{cost}(M) = \text{cost}(MM'')\).

**Lemma 4 (Improve**

\[
\text{Improve}^+ \text{, full}_{\text{ocdcl}} \text{, stgy}_{\text{no conflicting clause from init state}} \text{ and } \text{wf}_{\text{ocdcl}}\). In OCDCL, the rule Improve can be replaced by rule Improve**

\[
\text{Improve}^+ : \text{All previously established OCDCL properties are preserved.}
\]

The rules ConflOpt, Improve, and Improve can produce very long conflict clauses. Even with conflict minimization, they will contain the negation of all
decisions on the trail. It can be better to generate the conflict composed of only the literals with a weight, i.e., \( \neg \{ L \in M. \text{cost } L > 0 \} \) instead of \( \neg M \), although a more general Skip is required, such that the conflict contains one literal of highest level. This might not always be beneficial, because this is the opposite of the DECO optimization (DECision Only) used in Lingeling [3]: When the conflict is much longer than the clause only composed of decisions, then the latter is used.

It would also be possible to add the rules Restart and Forget, to change the search direction and remove some clauses, similarly to CDCL: Restart is applied after longer and longer intervals.

4 Formalization of OCDCL

If we ignore the Improve rule, the remaining OCDCL transitions are very similar to a generalized CDCL where all clauses that can be used by rules ConflOpt and Prune are included in the set of clauses. These are the clauses that are entailed by the clauses of weight larger than the optimal model found so far, i.e. the clauses \( D \) such that \( \{ \neg C. \text{cost } C \geq \text{cost } O \} \models D \). Hence, we can see OCDCL transitions as CDCL transitions and reuse proof on the latter. The rule Improve of OCDCL simply adds new clauses to this set. In the formalization, we abstract over this clause set by using instead a set \( T_N(O) \) and assume that it is increasing when the Improve rule is applied. yields a more abstract branch-and-bound calculus CDCL\textit{BnB} (Section 4.1). CDCL\textit{BnB} is seen as a special case of CDCL, where the additional clauses \( T_N(O) \) are part of the initial set of clauses. This reduces many proofs to reusing their CDCL counterpart: confiOpt becomes equivalent to confiSAT, since it picks a clause of \( T_N(O) \). We do not specify the type of \( O \) (Section 4.2). This reduces the burden to develop the new variant and makes it possible to reuse many proofs and especially all the invariants about the states: We neither have to redefine nor reprove most of them.

We instantiate CDCL\textit{BnB} to get a generalized version OCDCL\textit{g}: The set of clauses \( T_N(O) \) is instantiated by \( \{ D \mid \{ \neg C. \text{cost } C \geq \text{cost } O \} \models D \} \) (Section 4.3). Finally, we specialize OCDCL\textit{g} to get OCDCL from Section 3 (Section 4.4).

4.1 Branch-and-Bound Calculus, CDCL\textit{BnB}

We use a similar approach to our CDCL formalization with an abstract state and selectors, except that we add an additional component representing information on the branch-and-bound part of the calculus. We do not yet specify the type of this additional component. We do not only assume the existence of a separate set of clauses \( T_N(O) \), but also a predicate is\textit{improving} \( M M' O \) to indicate that Improve can be applied. For weights, the predicate is\textit{improving} \( M M' O \) means that the current trail \( M \) is a model, \( M' \) the information that will be stored, and \( O \) the current stored information. \( T \) represents all the clauses that are entailed. We require that:

- the atoms of \( T_N(O) \) are included in the atoms of \( N \). We don’t introduce new variables.
– the clauses of $T_N(O)$ do not contain duplicate literals. Duplicates are incompatible with the conflict analysis.
– if is improving $M M', O$, then $T_N(O) \subseteq T_N(M')$.
– if is improving $M M', O$, then $\neg M \in T_N(M')$.

The rules ConflOptBnB, ImproveBnB, and BacktrackBnB are defined as follows:

**ConflOptBnB** \( (M; N; U; k; \top; O) \implies_{OCDCL} (M; N; U; k; \neg M; O) \)
provided $\neg M \in T_N(O)$

**ImproveBnB** \( (M; N; U; k; \top; O) \implies_{OCDCL} (M; N; U; k; \neg M; M') \)
provided is improving $M M'$ holds

**BacktrackBnB** \( (M_1 K^{i+1} M_2; N; U; D \lor L; O) \implies_{OCDCL} (M_1 L', D' \lor L; N; U \cup \{D' \lor L\}; \top; O) \)
provided $L$ is of maximum level, $D' \subseteq D$, $N + U + T_N(O) \models D' \lor L$ and $D'$ is of level $i$ strictly less than the maximum level.

We can simply embed into our CDCL formalization the states with the weights and reuse the previous definitions, properties, and invariants. For example, we can reuse the Decide and some of the proofs on it. Moreover, we can reuse the invariants we have defined for CDCL. At this level, we have no information on what is stored in $O$.

Compared to the rule from Section 3, we make it possible to express conflict-clause minimization: Instead of $D \lor L$, a clause $D' \lor L$ is learned such that $D' \subseteq D$ and $N + U + T_N(O) \models D' \lor L$. While all other CDCL rules are reused, the Backtrack rule is not reused for OCDCL: If we had reused Backtrack from CDCL, only the weaker entailment $N + U \models D' \lor L$ would be used. The latter version is sufficient to express conflict minimization as implemented in most SAT solvers [21], but the former is stronger and makes it possible for example to remove decision literals without cost from $D$.

We use the Improve+ rule instead of the Improve rule, because the latter is a special case of the former. The strategy consists simply as favoring Conflict and Propagate over all other rules. We do not need to favor conflOpt over the other rules for correctness, although preferring conflOpt over Decide helps in an implementation.

### 4.2 Embedding into CDCL

To reuse the proof we did previously about CDCL, CDCLBnB is seen as a special instance of CDCL: We map the states $(M; N; U; D; O)$ to the CDCL state $(M; N \cup T_N(O); U; D)$.

The most direct solution would be instantiate the CDCL calculus with a the selector returning $N + T_N(O)$ instead of just $N$. For technical reasons, we cannot do so: This confuses Isabelle, because it leads to duplicate theorems and notations. Instead, we add an explicit conversion from $(M; N; U; D; O)$ to $(M; N + T_N(O); U; D)$ and consider CDCL on tuples on the latter.

Except for the Improve rule, every OCDCL rule can be mapped to a CDCL rule: The ConflictOptBnB rule corresponds to the Conflict rule (because it can
pick a clause including from $\mathcal{T}_N(O)$ and the extended Backtrack rule is mapped to CDCL’s Backtrack with the additional component. On the other hand, the Improve rule has no counterpart and requires some additional proofs. But adding clauses is compatible with the invariants.

In our formalization, we distinguish the structural from the strategy-specific properties. The strategy-specific properties ensure that the calculus does not get stuck in a state where we cannot conclude on the satisfiability of the clauses. The strategy-specific properties do not necessarily hold: The clause $\bot$ might be in $\mathcal{T}_N(O)$ without being picked by the ConflictOptBnB rule. However, we can easily prove that they hold for CDCLBnB and we can reuse the proof we have already done for most transitions. To reuse some proofs on CDCL’s Backtrack, we generalized some proofs by removing the assumption $N + U \models D' \lor L'$ to share more theorems. This is the only generalization we did on CDCL.

Not all transitions of CDCL can be taken by OCDCL: Propagating of clauses in $\mathcal{T}_N(O)$ is not possible. The structural properties are sufficient to prove that OCDCL is terminating, if Improve$^+$ can be applied only finitely often, because the CDCL calculus is terminating. At this level, Improve$^+$ is too abstract to prove that it terminates. With the additional assumptions that Improve always be applied when the trail is a total model satisfying the clauses (if one exits), we show that the final set of clauses is unsatisfiable.

4.3 Instantiation with weights, OCDCL$_g$

Finally, we instantiate $\mathcal{T}_N(O)$ with weights and save the best current found model in $O$. We assume the existence of a cost function that is monotone with respect to inclusion:

```
locale cost =  
  fixes cost :: 'v literal multiset => 'c  
  assumes \forall C. consistent_interp B \land distinct_mset B \land A \subseteq B \rightarrow \  
    cost A \leq cost B
```

We assume that cost is function is monotone with respect to inclusion for consistent duplicate-free models. This is natural for trails, who by construction do not contain duplicates. The monotonicity is less restrictive than the condition from Section 3, which mandates that the cost is a sum over the literals. We take

$$\mathcal{T}_N(O) = \{ C. \ atom(C) \subseteq atom(N) \land C \text{ is not a tautology nor contains duplicates} \land \{ \neg D. \ cost(D) \geq cost(O) \} \vdash C \}$$

is improving $M \leftrightarrow M'$ $O \leftrightarrow M'$ is a total extension of $M$, $M \models N$,

any total extensions of $M$ has the same cost, and

$cost M < cost O$

and then discharge the assumptions over it.
OCDCL\textsubscript{g} inherit from the invariants from CDCL\textsubscript{BnB}. For termination, we only to prove that Improve\textsuperscript{+} terminates to reuse the proof we already made on CDCL\textsubscript{BnB}. The key property of our construction is the following:

**Lemma 5 (entails too heavy clauses too heavy clauses \(\emptyset\))** If \(I\) is a total consistent model of \(N\), then either \(\text{cost}(I) \geq \text{cost}(O)\) or \(I\) is a total model of \(T_{N}(O)\).

**Proof.** Assume \(\text{cost}(I) < \text{cost}(O)\). First, we can show that \(I \vDash \{ \neg C. \text{cost}(C) \geq \text{cost}(O) \}\). Let \(D\) be a clause of \(\{ \neg C. \text{cost}(C) \geq \text{cost}(O) \}\). \(C\) is not a subset of \(I\) (by monotonicity of cost, \(\text{cost}(I) \geq \text{cost}(C)\)). Therefore, there is at least a literal \(L\) in \(C\) such that \(\neg L\) in \(I\). Hence \(I \vDash C\).

By transitivity, since \(I\) is total, \(I\) is also a model of \(T_{N}(O)\). \(\Box\)

This is the proofs that breaks if partial models are allowed. Some additional proofs are required to specify the content of the component \(O\). First, \(O\) always contains a total consistent model. This property cannot be inherited from the correctness of CDCL, because CDCL does not know about the component \(O\).

**4.4 OCDCL**

Finally, we can refine the calculus to precisely the rules expressed in Section 3. We define two calculi: one with only the rule Improve, and the other with both Improve\textsuperscript{+} and Prune. In both cases, the rule ConflictOpt is only applied when \(\text{cost}(M) > \text{cost}(O)\) and is therefore a special case of ConflictOpt\textsubscript{BnB}. The Prune rule is also seen a special case of ConflictOpt\textsubscript{BnB}. Therefore, every transition is also a transition of OCDCL\textsubscript{g}. Moreover, since final states of both calculi are the same, a full run of OCDCL is also a full run of OCDCL\textsubscript{g}. Therefore, the correctness theorem can be inherited.

Overall, the full formalization was easy to do, once we got the idea how to see OCDCL as a special case of CDCL. Formalizing a changing target is different than an already fixed version calculus: We had to change our formalization several times to take into account additional rules: The Prune rule requires to use \(\{ D \mid \{ \neg C. \text{cost}(C) \geq \text{cost}(O) \} \vdash D \}\), while the set of clauses \(\{ D \mid \{ \neg C. \text{cost}(C) \geq \text{cost}(O) \} \vdash D \}\) is sufficient for Improve\textsuperscript{+}.

**5 Optimal Partial Valuations**

A partial \(\Sigma\)-valuation is a partial mapping \(\mathcal{A} : \Sigma \rightarrow \{0, 1\}\) from the set of propositional variables \(\Sigma\) into \(\{0, 1\}\). For any atom \(P \in \Sigma\), we write \(\mathcal{A}(P) \downarrow\) if \(\mathcal{A}\) is defined on \(P\). If \(\mathcal{A}(P) \downarrow\) and \(\mathcal{A}(P) = 1\) we write \(\mathcal{A} \models P\). The valuation \(\mathcal{A}\) can be extended to literals, clauses and clause set as follows: \(\mathcal{A}(-P) := 1 - \mathcal{A}(P)\) if \(\mathcal{A}(P) \downarrow\) and unset otherwise. \(\mathcal{A}(L_1 \lor \cdots \lor L_n) := 1\) if there is some \(L_i\) with \(\mathcal{A}(L_i) \downarrow\) and \(\mathcal{A}(L_i) = 1\). \(\mathcal{A}(C_1 \land \cdots \land C_n) := 1\) if \(\mathcal{A}(C_i) \downarrow\) and \(\mathcal{A}(C_i) = 1\) for all \(i\). If \(\mathcal{A}\) is defined and evaluates a literal, clause, clause set to 1 we write \(\mathcal{A} \models L\), \(\mathcal{A} \models L_1 \lor \cdots \lor L_n\), and \(\mathcal{A} \models C_1 \land \cdots \land C_n\), respectively. As usual we identify clause sets and conjunctions of clauses.
To reduce the search from optimal partial valuations to optimal total valuations, we use the dual rail encoding \([6, 18]\). For every proposition variable \(P\), we create two variables \(P^1\) and \(P^0\) indicating that \(P\) is defined positively or negatively. We also add the clause \(\neg P^1 \lor \neg P^0\) to ensure that \(P\) is not defined positively and negatively at the same time. The resulting clause set is called penc \(N\).

More precisely, the encoding penc is defined on literals by penc(\(P\)) := (\(P^1\)), penc(\(\neg P\)) := (\(P^0\)), and lifted to clauses and clause sets by penc(\(L_1 \lor \cdots \lor L_n\)) := penc(\(L_1\)) \lor \cdots \lor penc(\(L_n\)), and, penc(\(C_1 \land \cdots \land C_m\)) := penc(\(C_1\)) \land \cdots \land penc(\(C_m\)). We call \(\Sigma'\) the set of all newly introduced atoms.

The important property of this encoding is that \(\neg P^1\) does not entail \(P^0\): If \(P\) is not positive, it does not have to be negative either.

Given an encoding penc(\(N\)) of a clause set \(N\) the cost function is extended to a valuation \(A'\) on \(\Sigma \cup \Sigma'\) by cost'\((A')\) = cost \((\{L \mid L^1 \in A'\} \cup \{-L \mid L^0 \in A'\})\).

Let pdec \((A)\) := \[
\begin{cases} 
1 & \text{if } A(P^1) = 1 \\
0 & \text{if } A(P^0) = 1 \\
\text{unset} & \text{otherwise}
\end{cases}
\]

to transform a total model of penc \(N\) into a model of \(N\) and pdec\(^-\) \((A)\) does the opposite transformation, with pdec\(^-\) \((A)(P^1)\) = 1 if \(A(P) = 1\), pdec\(^-\) \((A)(P^1)\) = 0 if \(A(P) = 0\), pdec\(^-\) \((A)(P^0)\) = 1 if \(A(P) = 0\), pdec\(^-\) \((A)(P^0)\) = 0 if \(A(P) = 1\), unset otherwise.

**Lemma 6** (Partial and Total Valuations Coincide Modulo penc, penc\_ent\_postp \(\emptyset\) and penc\_ent\_upostp \(\emptyset\)). Let \(N\) be a clause set.

1. If \(A \models N\) for a partial model \(A\) then pdec\(^-\) \((A) \models \text{penc}(N)\);
2. If \(A' \models \text{penc}(N)\) for a total model \(A'\), then pdec \((A') \models N\).

**Lemma 7** (penc Preserves Cost Optimal Models, full_encoding\_OCDCL\_correctness \(\emptyset\)). Let \(N\) be a clause set and cost a cost function over literals from \(N\). If \(A'\) is a cost-optimal total model for penc \((N)\) over cost', resulting in cost'\((A')\) = \(m\), then the partial model pdec \((A')\) is cost-optimal for \(N\) and cost(pdec \((A')\)) = \(m\).

**Proof.** Assume there is a partial model \(A\) for \(N\) with cost \((A)\) = \(k\). The model pdec\(^-\) \(A\) is another model of \(N\). As \(A'\) is cost optimal, cost'\((\text{pdec}^- (A))\) \(\geq\) cost'\((A')\). Moreover, cost'\((\text{pdec} (A'))\) = cost \((A')\) and cost(pdec \((A)\)) = cost \((A)\). Ultimately, \(A\) is not better than \(A'\) and \(A'\) has cost \(m\). \(\square\)

penc \((N)\) contains \(|N| + |\Sigma|\) clauses. Recall that for \(n\) propositional variables there are \(2^n\) total valuations and \(3^n\) partial valuations.

**Lemma 8** (OCDCL on the Encoding). Consider a reasonable OCDCL run on penc \((N)\). If rule Decide is restricted to deciding either \(P^1\) or \(P^0\) for any propositional variable, then in any state where all the previously mentioned literals have been decided, Propagate was exhaustively applied and Conflict is not applicable, the trail represents a total valuation satisfying \(N\).
Non-Machine-Checked Lemma 9 (OCDCL on the Encoding) Consider a reasonable CDCL run on penc($N$). If rule Decide is restricted to deciding either $P^1$ or $P^0$ for any propositional variable, and ConflOpt only considers the decision literals out of $M$ as a conflict clause, then OCDCL performs at most $3^n$ Backtrack steps.

Proof. Consider a run on the clause set $N' = penc(N)$ resulting from $N$ after encoding. For rule Decide we use the above strategy. By Lemma 8, if all decisions are done, the respective trail is either a model for $N'$, or there is a conflict. Assume there is a conflict. Then OCDCL will flip the most recent decision on some $P^1$ or $P^0$ generating the complement. The strategy decides then remaining literals. In summary, for each propositional variable, a run considers at most 3 cases, overall $3^n$ cases for $n$ different variables in $N$. \qed

6 Formalization of the Partial Encoding

In Isabelle, there are total valuations defined by giving a set of all true atoms (all others being false), mostly used for of Herbrand interpretations [19]. There, however, are not really adapted to the verification of CDCL. Therefore, we are already using partial models, similar to a trail and have a predicate to indicate that a model is total. We distinguish between literals that can have a weight $\Delta \Sigma$ from the others ($\Sigma \setminus \Delta \Sigma$) that can be left unchanged by the encoding.

The proofs are very similar to the proofs described in Section 5. We instantiate the OCDCL calculus with the cost' function:

\begin{verbatim}
interpretation OCDCL where cost = cost'
\end{verbatim}

We have to prove the proof obligation that cost' is monotone, which can be easily discharged.

Finally, we can prove the correctness Theorem 7. The proof is the same. The formalization is 800 lines long for the encoding, and 500 additional lines for the restriction of decide.

We have not yet formalized the complexity bound of $3^n$ of Lemma 9, but plan to do so. So far, we have only verified the correctness of the variant with the classical DPLL backtrack. It can be seen as a special case of conflict analysis and backtrack thanks to conflict minimization: $(M_1 K^\uparrow M_2, N, U, \neg (M_1 K^\downarrow M_2)) \Rightarrow^* \text{Resolve} (M_1 K^\uparrow, N, U, \neg (M_1 K^\downarrow)) \Rightarrow^* \text{Backtrack} (M_1 \neg K^{D'}, N, U \cup \{D\}, \top)$, where $D'$ is the negation of the decisions of $M_1 K^\uparrow$ and $M_2$ does not contain any decision. If there are no decision in the trail, we set the conflict to $\bot$.

7 Solving Partial MAX-SAT with OCDCL

Partial maximum satisfiability problem (MAX-SAT) is a famous problem [12]. Two sets of clauses $N_H$ (hard constraints, mandatory to satisfy) and $N_S$ (soft constraints, optional to satisfy) with a cost when they are satisfied. The aim is to find a total model with maximal weight. If the weights are equal, the MAX-SAT tries to satisfy as many clauses as possible.
Theorem 10 (partial_max_sat_is_weight_sat $\emptyset$). Given a MAX-SAT problem $(N_H, N_S, \text{cost})$ and a mapping $L$ from $N_S$ to an additional distinct positive literal.

Let $I$ be the solution to the OPT-SAT problem $N = N_H \cup \{L(C) \lor C. C \in N_S\}$ with the cost function:

$$\text{cost}'(M) = \sum_{C \in N_S \atop L(C) \in M} \text{count}(N_S, C) \times \text{cost} C$$

where count$(N_S, C)$ is the number of times the clause $C$ appears in $N_S$.

If there is no such model, the problem has no solution. Otherwise, the model without the additional atoms \{L.L \in I \land \text{atom}(I) \in \text{atom}(N_H + N_S)\} is optimal.

Proof. – $N_H$ is satisfiable iff MAX-SAT has a solution. Therefore, if there is no model, then $N_H$ is unsatisfiable.

– Let $I'$ be \{L. L \in I \land \text{atom}(L) \in \text{atom}(N_H + N_S)\}. Let $J$ be any other model and $J'$ the total extension $J \cup \{L(C). C \in N_S \land J \not\models C\} \cup \{-L(C). C \in N_S \land J \models C\}$ to $N$.

$J'$ satisfies $N_H$ and is a total consistent model of $N$. Therefore, cost'$(J') \geq$ cost$(I)$, because $J$ is the optimal model. By definition, cost'$(I) = \text{cost}(I')$ and cost'$(J) = \text{cost}(J')$. Therefore, $I$ is an optimal MAX-SAT model.

If we consider $N = \{C_1, \ldots, C_n\}$, then we can distinguish clauses from each other. Then, it is sufficient to consider the problem $N = N_H \cup \{L_i \lor C_i. i \in [1, n]\}$ with the cost function cost'$(M) = \sum_{L_i \in M} \text{cost} C_i$.

8 Model Covering, Another Instance of CDCL with branch-and-bounds

We use the framework described in Section 5 to express a different problem, the calculation of covering models. Given a function $\rho: 'v \Rightarrow \text{bool}$, the aim is to find a set of models $M$ such that if $\rho(P)$ then there is a model where $P$ is true. Given constraints on features, it makes it possible to detect if some of them are dead \cite{2} and cannot be taken. Once the set $M$ is computed, it is possible to minimize it. This is the classical NP-complete Set Cover Problem \cite{10}.

To solve the model-covering problem, we define the domination relation: A model is dominated if there another that contains more true literals. More formally, $I$ is dominated by $J$ if \{L \in I \mid \rho(L)\} $\subseteq$ \{L \in J \mid \rho(L)\}. If a total model is dominated by a model of $M$, then it is not required in the covering. The model covering can be computed by creating another CDCL variant, where the set $M$ is explicitly built in the last component of a state. The differences are the additional rules:

ConflCM \quad (M; N; U; \top; M) \implies_{\text{CDCLcm}} (M; N; U; \neg M; M)$

provided for all total extensions $MM'$ with $MM' \models N$, there is an $I \in M$ which dominates $MM'$.
Add \((M; N; U; k; \top; \mathcal{M}) \implies_{\text{CDCLcm}} (M; N; U; k; \top; \mathcal{M} \cup \{M\})\)

provided \(M \models N\), all literals from \(N\) are defined in \(M\) and \(M\) is not dominated by a model in \(\mathcal{M}\).

This calculus is another instance of CDCL\textsubscript{BnB}. In the formalization, we instantiate CDCL\textsubscript{BnB} with:

\[
T_N(M) = \{C. \ \text{atom}(C) \subseteq \text{atom}(N) \\
\quad \land C \text{ is not a tautology nor contains duplicates} \\
\quad \land \{-D|_p. \ \text{is dominating} \ \mathcal{M} \ D, \ \text{total}\} \cup N \models C\}
\]

is improving \(M M' \mathcal{M} = \{M = M', M \models N, \ \quad M \text{ is not dominated by } \mathcal{M}\)
\(M \text{ is consistent, total, duplicate free}\}\)

where \(D|_p\) is the restriction to the atoms making \(\rho\) is true. Compared to OCDCL, \(T_N(\emptyset)\) is never empty, because it contains at least \(N\).

**Theorem 11 (CDCLcm Correctness, cdclcm_correctness \(\emptyset\)).** If the clauses in \(N\) contains no duplicated literal and \((\epsilon, N, \emptyset, \top, \emptyset) \implies_{\text{CDCLcm}} T\), then for every literal \(L\) such that \(\rho(L)\), either there is no model of \(N\) where \(L\) holds, or one appears in \(T\).

The proof involves a lemma similar to Lemma 5: Every model is either dominated by a model in \(\mathcal{M}\), or is still a model of \(T_N(\mathcal{M})\). Since the calculus ends in a state where the clauses are unsatisfiable, we can conclude.

## 9 Related Work and Conclusion

There are several formalization of CDCL beyond ours, as we discuss in our previous article [5, Section 6], but we are not aware of any formalization of an optimizing CDCL, or more generally a formalization used as a starting point to formalize variants of CDCL.

There are several variants of optimizing SAT. Larossa et al. have developed a similar approach to ours [11]. They define cost optimality with respect to partial models, but their Improve rule only considers total models. Our calculus is slightly more general due to the inclusion of the rules Improve\textsuperscript{+}. Moreover, the first unique implication point is built in our calculus. The Pruning rule can be simulated by applying their Learn rule: \(\neg M \lor c \geq \text{cost } O\) is entailed by the clauses.

A related problem to finding the minimum partial model is called Minimum-Weight Propositional Satisfiability by Sebastiani et al. [20]. It assumes that negative literals do not cost anything: This means that the opposite \(L\) is \(-L\) (as \(-L\) and \(L\) undefined have the same weight). Although, Liberatore’s method can return partial models, it is an Herbrand model: It is entirely given by the
set of all true atoms. Therefore, the method builds total models. Liberatore has
developed a variant of DPLL to solve this problem [13]. Each time a variable is
decided, it is first set to true, then set to false. Moreover, if the current model is
larger than a given bound, then the search stops exploring the current branch.
When a new better model is found, the search is restarted with the new lower
bound. A version lifted to CDCL has been implemented in zChaff [8] to solve
MAX-SAT.

The model-covering problem is related to the minimal model covering, in the
sense that first a model covering must be found before minimizing it. The latter
can also be solved by CDCL with branch and bounds [14], and would probably
fit in our framework.

We have presented here a variant of CDCL to find optimal models and used
the dual rail encoding to reduce the search of optimal models with respect to
partial valuations to the search of optimal models with respect to total valuations.
Both have been formalized using the proof assistant Isabelle/HOL. This formal-
ization was simple thanks to the framework we have previously developed and
the abstraction we have used in Isabelle to simplify reuse and studying variants
and extensions.

On an abstract level, OCDCL is close to an incremental version of CDCL(T),
the calculus used in SMT solvers. The main difference is that the conflicts
generated are not the negation of the trail, but a smaller conflict. The theory of
linear arithmetic has already been verified in Isabelle/HOL by Thiemann [22],
so proving correctness of CDCL(T) does not need a from-scratch new effort.

Acknowledgement. Jasmin Blanchette gave us his permission to reuse, in a
slightly adapted form, the presentation of Isabelle he cowrote for the formalization
of CDCL [5]. We thank Armin Biere and Roberto Sebastiani for a number of
helpful discussions. We also thank the reviewer who pointed out the dual rail
encoding that is more elegant than our initial solution.

The second author has received support by the DFG as part of CRC 248 (see
perspicuous-computing.science).

References

1. Becker, H., Bentkamp, A., Blanchette, J.C., Fleury, M., From, A.H., Jensen, A.B.,
Lammich, P., Larsen, J.B., Michaelis, J., Nipkow, T., Peltier, N., Popescu, A.,
Robillard, S., Schlichtkrull, A., Tourret, S., Traytel, D., Villadsen, J., Petar, V.:
IsaFoL: Isabelle Formalization of Logic, https://bitbucket.org/isafol/isafol/
Competition 2016—Solver and Benchmark Descriptions. Department of Computer
(2016)


