A Verified SAT Solver Framework including Optimization and Partial Valuations

Mathias Fleury
Johannes Kepler University Linz
Linz, Austria
mathias.fleury@jku.at
Max-Planck-Institut für Informatik
Saarbrücken, Germany
mathias.fleury@mpi-inf.mpg.de
Graduate School of Computer Science
Saarbrücken, Germany
s8maflu@stud.uni-saarland.de

Christoph Weidenbach
Max-Planck-Institut für Informatik
Saarbrücken, Germany
weidenb@mpi-inf.mpg.de

Abstract

Based on the formal framework for CDCL (conflict-driven clause learning) by Blanchette et al. verified using the proof assistant Isabelle/HOL, we verify an optimizing extension of CDCL based on branch and bound, called OCDCL, by developing a framework for CDCL with branch and bounds, called CDCL_BnB. OCDCL computes models of minimal cost with respect to total valuations. Through the dual rail encoding, we reduce the search for cost-optimal models with respect to partial valuations to searching for total cost-optimal models, as derived by OCDCL. OCDCL can also be used to solve further optimization tasks such as MAX-SAT and CDCLBnB to find a minimal set of covering models. A large part of the 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 an optimizing CDCL calculus and the first solution that computes cost-optimal models with respect to partial valuations.

Keywords Isabelle/HOL, conflict-driven clause learning, CDCL, optimization, MaxSAT

1 Introduction

Researchers in automated reasoning spend a significant portion of their work time specifying calculi and proving meta-theorems 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 of an effort, called IsaFoL (Isabelle Formalization of Logic) [1] that aims at developing libraries and methodologies to formalize these calculi using the Isabelle/HOL proof assistant [18]. This does not only include developing formal companions to paper proof but also includes simplifying the verification of variants of earlier devised calculi.

We extend the previous formalized propositional satisfiability (SAT) by Blanchette et al. [4] that is 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 a cost-optimal model for a set of clauses. In Weidenbach’s upcoming textbook tentatively called Automated Reasoning—Some Basics, he developed an optimizing CDCL for OPT-SAT called OCDCL. It is based on his CDCL calculus [4] 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 [12], 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 formalizing OCDCL on total valuations, as well as for finding an encoding such that OCDCL is also correct with respect to partial valuations.

Compared with most of the existing literature on finding cost-optimal models, we define our cost function on literals, not on atoms. This arises in applications where a propositional variable encodes membership in one class and its negation membership in a different class. For example, in a
product configuration scenario, the truth of a propositional
may encode the containment of a component in a product
generating some cost. However, the absence of the compo-
nent typically also generates some (often lower) cost.

Our contributions are firstly a rigorous proof of the correct-
ness of OCDCL with respect to total valuations and secondly
a rigorous proof that OCDCL is also correct with respect
to partial valuations using a dual rail encoding [7, 19]. For-
malization 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 essentially solve the problem of finding optimal models
with respect to the two types of valuations using the very
same framework. The OCDCL formalization amounts to
around 3300 lines of proof. This is small compared to the
2600 lines of shared libraries, plus 6600 lines for the formal-
ization of CDCL [4], and 4500 lines for Nieuwenhuis et al.’s
account for CDCL [16]. Thus, thirdly, we show that one
of the standard arguments supporting formal verification,
the reuse of already proved results, works out in the case of
CDCL and OCDCL. The overall formalization took around
1.5 person-month of work and was easy to do. The exten-
sion to partial valuations amounts to 1300 lines. We further
demonstrate the potential of reuse, by applying our frame-
work to two additional problems: MAX-SAT and covering
models. Again the results from OCDCL and the framework,
respectively could be reused. The overall effort for the two
extensions was 800 and 400 lines, respectively.

After some preliminaries on Isabelle and the previous for-
malization of CDCL (Section 2), the optimizing OCDCL cal-
culus is introduced (Section 3). It is described as an abstract
non-deterministic transition system and has the conflict anal-
ysis for the first unique implication point built into its CDCL
part.

We develop an abstract calculus, CDCL_{8bit}, that imple-
ments branch and bound. This calculus implements an ab-
stract and bound and it can be instantiated with weights
(Section 7). In order to overcome the limitation of our cal-
culus to total models. We show an encoding that reduces
finding a cost-optimal model based on partial valuations to
finding a total cost-optimal model (Section 5), which was
also formalized in Isabelle (Section 6). We then apply the
OCDCL calculus to solve MAX-SAT and to find minimal
sets of covering models (Section 7). The paper ends with
a discussion of the obtained results, and future and related
work (Section 8).

All theorems that have been proved in Isabelle show in
the theorem name the Isabelle theorem name with the no-
tation "Isa:fact from the formalization with a link to the
documentation of our formalization.¹ Throughout this paper,
we distinguish "Paper Lemmas" that follow the presentation

¹Note to the reviewers: We assume that the supplementary material and
this PDF are in the same folder.

on paper from Isabelle Lemmas that follows the Isabelle for-
malization. However, we link the "Paper Lemmas" to their
formalized counterpart even if the formalization path is dif-
ferent. This presentation is based on the Ph.D. thesis of one
of the authors.

2 Formalization of CDCL in Isabelle

Isabelle Isabelle [17, 18] is a generic proof assistant that
supports many object logics. The metalogic is an intuition-
istic fragment of higher-order logic (HOL) [8]. The types
are built from type variables ‘a, ’b, . . . and n-ary type con-
structors, normally written in postfix notation (e.g, ‘a list).
The infix type constructor ‘a ⇒ ‘b is interpreted as the (to-
tal) function space from ‘a to ‘b. Function applications are
written in a curried style (e.g., f x y). Anonymous functions
x ⇒ y x are written λ x. y x. The judgment t ⇔ τ indicates
that term t has type τ.

Isabelle adheres to the tradition initiated in the 1970s by
the LCF system [10]: 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 recur-
sive functions. Internally, the system synthesizes appropriate
low-level definitions.

Isabelle projects 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 structur-
large proofs. A locale fixes types, constants, and assump-
tions 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, explicit-
ly fixes a constant c whose type τ a may depend on ‘a,
and states an assumption A::a,c 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::a,t. After the proof, all the lemmas proved
in X become available, with $\tau_a$ and $\tau_c :: \tau_{\tau_0}$ instantiated with $v$ and $t :: \tau_0$.

**CDCL** A framework to formalize the conflict-driven-clause-learning procedure has previously been developed in Isabelle by Blanchette et al. [5]. Clauses are defined as (finite) multisets. For readability, we will write $\bot$ and $A \lor B$ instead of their Isabelle counterparts ($\#$) and $A + B$. Given a literal $L$ and a set of literals $I$, we define entailment by $I \models L$ if $L \in I$.

We can lift it to clauses (multiset of literals) by $I \models C$ if there is a literal $L \in C$ such that $I \models L$. We can also lift it to clause sets $I \models N$ iff $\forall C \in N \models I$. $I \models C$.

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 ($L^C$, where $C$ justifies the propagation). 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 on tuples $(M, N, U, D)$ where $M$ is the current partial model assumption, $N$ are the initial clauses, $U$ are learned clauses, and $D$ is either a conflicting clause that is currently analyzed or $\tau$. For example, the Decide rule extends $M$ by an arbitrary choice $L$:

$$\text{Decide} \quad (M; N; U; \tau) \Rightarrow (M^L; N; U; \tau)$$

if $L$ is undefined in $M$ and $\text{atom}(L) \in N$.

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:

$$\text{inductive decide :: 'st} \Rightarrow 'st \Rightarrow \text{bool where}$$

$\text{undefined_lit (trail S)} L \Rightarrow$

$\text{L \in \text{atom (clauses S) \Rightarrow}$

$'st' \sim \text{append_trail (L')} S \Rightarrow$

$\text{decide S' }$

where the functions `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. Since states are not equivalent if the selectors return the same components. We don’t use the usual equality on `$st'$, and instead use $S \sim T$ that holds if all components are equal. This also gives us more freedom for the implementation of the state. The state and its selectors are defined in a locale specifying their behavior.

To simplify the notation we will use 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 $\text{cost}$ on the set of all literals $\text{Lit}(\Sigma)$ over $\Sigma$ into the positive rationals, $\text{cost} : \text{Lit}(\Sigma) \rightarrow \mathbb{Q}^+$, where our results do not depend specifically on the positive rationals (including 0), e.g., they also hold for the naturals or positive reals. In Isabelle, we take all values of an arbitrary type instead of the set $\text{Lit}(\Sigma)$. The cost function can be extended to a pair of a literal and a partial valuation $\mathcal{A}$ by $\text{cost}(L, \mathcal{A}) := \text{cost}(L)$ if $\mathcal{A} \models L$ and $\text{cost}(L, \mathcal{A}) := 0$ if $L$ is not defined. The function can be extended to (partial) valuations by $\text{cost}(\mathcal{A}) = \sum_{L \in \text{Lit}(\Sigma)} \text{cost}(\mathcal{A}, L)$. We identify partial valuations with consistent sequences $M = [L_1 \ldots L_n]$ of literals. Trails are always consistent. A valuation $I$ is total over clauses $N$ when all atoms of $N$ are defined in $I$.

The intuition behind the extension is to have an incremental CDCL with the usual rules as basis. The calculus is extended in two ways: First, models are identified as such and stored in a new component of the state. Second, CDCL runs are pruned by adding conflict clause on the fly, based on the best model found so far. Another way of seeing it, consists in considering the calculus as a special case of branch-and-bound algorithm: CDCL finds a model that is stored (the branching part), while the stored model is used to limit the depth of the search (the bounding part) by adding conflict clauses that are subsequently analyzed by the usual CDCL rules.

The optimizing conflict-driven clause learning calculus (OCDCCL) solves the weighted SAT problem on total valuations. Compared with a normal CDCL state, a component $O$ is added resulting in a five tuple $(M; N; U; D; O)$. $O$ either stores the best model so far or $\tau$. We extend the cost function to $\tau$ by defining cost $(\tau) = \infty$ (i.e., $\tau$ is the worst possible outcome). OCDCCL is a strict extension of the CDCL rules with additional rules to take the cost of models into account. The additional component $O$ is ignored by the original CDCL rules.

The start state for some clause set $N$ is $(\varepsilon; N; \emptyset; \tau; \tau)$. The calculus searches for models in the usual CDCL-style. Once a model is found, it is ruled out by generating a conflict clause resulting from its negation which is then processed by the standard CDCL conflict analysis (rule Improve, defined below). If a partial model $M$ already exceeds the current cost bound, a conflict clause is generated (rule ConflOpt, defined below). The OCDCCL calculus always terminates in deriving the empty clause $\bot$. If in this case $O = \tau$, then $N$ was unsatisfiable. Otherwise, $O$ contains a cost-optimal total model for $N$.

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** $(M; N; U; \tau; O) \Rightarrow_{\text{OCDCCL}} (M; N; U; \tau; M)$ provided $M \models N$, $M$ is total over $N$ and $\text{cost}(M) < \text{cost}(O)$.

**ConflOpt** $(M; N; U; \tau; O) \Rightarrow_{\text{OCDCCL}} (M; N; U; \neg M; O)$ provided $O \neq \tau$ and $\text{cost}(M) \geq \text{cost}(O)$.

**Prune** $(M; N; U; \tau; O) \Rightarrow_{\text{OCDCCL}} (M; N; U; \neg M; O)$
provided for all total trail extensions \(MM'\) of \(M\), it holds that \(\text{cost}(MM') \geq \text{cost}(O)\).

The Prune rule 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 maintained for efficiency.

The other rules are unchanged imports from the CDCL calculus. They simply ignore the additional component \(O\).

The rules Propagate and Decide extend the trail searching for a model. The rule ConflSat detects a conflict. All three rules implement the classical CDCL-style model search until conflict or success.

**Propagate** \((M; N; U; \top; \tau; O) \Longrightarrow_{\text{ODCL}} (L^{CVL} M; N; U; \top; O)\)

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

**Decide** \((M; N; U; \top; \tau; O) \Longrightarrow_{\text{ODCL}} (L^1 M; N; U; \top; O)\)

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

**ConflSat** \((M; N; U; \top; O) \Longrightarrow_{\text{ODCL}} (M; N; U; D; O)\)

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

Once a conflict has been found, it is analyzed to derive a new clause that is then a first unique implication point \([3]\).

**Skip** \((L^{CVL} M; N; U; D; O) \Longrightarrow_{\text{ODCL}} (M; N; U; D; O)\)

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

**Resolve** \((L^{CVL} M; N; U; D \lor \neg L; O) \Longrightarrow_{\text{ODCL}} (M; N; U; D \lor C; O)\)

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

**Backtrack** \((M_k K^I M_1; N; U; D \lor L; O) \Longrightarrow_{\text{ODCL}} (L^{DVL} M_1; N; U \cup \{D \lor L\}; \top; O)\)

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 \(N = \{P \lor Q\}\) and cost function cost \((P) = 3\), 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 \([P^I]\) 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 \([Q^{P \lor Q}, \neg P^{P \lor Q}, \neg P^I]\) 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 cannot conclude the clause \(\neg P\) to eliminate all further models containing \(P\), because \(P\) could be undefined.

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

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

**Paper Proof.** If the derivation started from \((\epsilon, N, \emptyset, \top, \tau, \text{top})\), the following function is a measure for OCDCL:

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

It is decreasing for the lexicographic order. The hardest part of the proof is the decrease when the rule Backjump is applied: \(3^n - 1 - |U \cup \{D \lor L\}| < 3^n - 1 - |U|\) is decreasing since no clause is relearned. The proof is similar to the one for CDCL, which is discussed in \([4]\).

**Theorem 3** (OCDCL Correctness, Isa:\text{full}_{ocdcl}\_w\_stgy_no_\_conflicting_clause_from_\_init_\_state\_✓). An OCDCL run with a reasonable strategy starting from state \((\epsilon; N; \emptyset; 0; \top; \tau; \epsilon)\) terminates in a state \((M; N; U; \bot; O; \top)\). If \(O = \epsilon\) then \(N\) is unsatisfiable. If \(O \neq \emptyset\) 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\(^+\)** \((M; N; U; \top; O) \Longrightarrow_{\text{ODCL}} (M; N; U; \top; MM')\)

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

**Paper Lemma 4** (OCDCL with Improve\(^+\), Isa:\text{full}_{ocdcl}\_w\_p\_stgy_no_\_conflicting_clause_from_\_init_\_state\_✓). The rule Improve can be replaced by Improve\(^+\): All previously established OCDCL properties are preserved.

The rules ConflOpt can produce very long conflict clauses. Moreover, without conflict minimization, the subsequent backtrack is only chronological, i.e., only the last decision literal is removed from the trail. Even with conflict minimization, they will contain the negation of all decision literals from the trail. It can be advantageous to generate the conflict composed of only the literals with a non-zero weight, i.e., \(\neg (L \in M \mid \text{cost } L > 0)\) instead of \(\neg M\). In this case a more general Skip is required, such that the eventual conflict before application of Backtrack contains one literal of highest level. As said, this is not always beneficial, e.g., the rule used
in Lingeling [2] switches between the two options by taking the shortest clause.

The rules Restart and Forget can also be added to OCDCL with the same well-known implications from CDCL. For example, completeness is only preserved if Restart is applied after longer and longer intervals.

4 Formalization of OCDCL

The formalization takes a different path in order to reuse the original CDCL proofs without copy-paste. The basic idea is to reuse as much as possible from the CDCL formalization, including definitions and invariants, to avoid defining the same concepts anew and reprove the same lemmas, like the consistency of the trail. The second point is that we want to formalize a more abstract approach, namely CDCL with branch and bound to be more general.

A first observation is that an OCDCL run can be seen as a series of CDCL runs, separated by applications of the rules Improve or ConflOpt. This is conceptually similar to the incremental version of the calculus developed in the CDCL framework from Blanchette et al. [4], except that we do no wait for a complete CDCL run before adding clauses and conflicts. We can go further with the analogy: ConflOpt is a conflict rule, except that the clause does not appear in \( N \) or \( U \), but only in a richer set. Therefore, we consider a CDCL run extended with the clauses that ConflOpt can use, namely \( \{ \neg C. \text{cost} C \geq \text{cost} O \} \). With this idea, the Prune rule is now another special case of ConflOpt. Only Improve is different: It adds new clauses because more models are now excluded. Therefore, Improve has no counterpart. Figure 1 shows how CDCL is used twice in the formalization, whereas Figure 2 shows the correspondence on a run.

CDCL\(_{\text{BnB}}\) can be seen as a generalized CDCL interleaved with adding new clauses (Section 4.2). However, since it might not always be possible to calculate explicitly \( T_N(O) \), the calculus does not eagerly find conflict clauses for clauses in \( T_N(O) \).

To generalize over the optimizing CDCL, we abstract over the additional set of clauses, by calling it \( T_N(O) \), and use a predicate is_improving \( M \) \( M' \) \( O \) to indicate that Improve can be applied. That way we obtain an abstract branch-and-bound calculus CDCL\(_{\text{BnB}}\) (Section 4.1).

The advantage of this approach is two fold: First, we formalize a more general calculus that can be instantiated with a different set of clauses to obtain a different branch-and-bound CDCL. Second, and more importantly, the resulting calculus can reuse many proofs already done on CDCL, because it is a CDCL interleaved with adding new clauses. In particular, if applying Improve is terminating, then CDCL\(_{\text{BnB}}\) is also terminating, because CDCL is also terminating.

Finally, we can instantiate CDCL\(_{\text{BnB}}\) to get a generalized version OCDCL\(_g\). The set of clauses \( T_N(O) \) is instantiated by the set of clauses \( \{ D \mid \{ \neg C. \text{cost} C \geq \text{cost} O \} \models D \} \) (Section 4.3). This additional set of clauses is too weak to be able to express the rule Prune in the set of clauses. Therefore, we actually use 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 \( \{ \neg C \mid \text{cost} C \geq \text{cost} O \} \models D \). At last, we specialize OCDCL\(_g\) to get exactly the OCDCL calculus from Section 3 (Section 4.4).

4.1 Branch-and-Bound Calculus, CDCL\(_{\text{BnB}}\)

We use a similar approach to our CDCL formalization with an abstract state and selectors. Compared to CDCL, we add an additional component representing information on the optimizing branch-and-bound part of the calculus. At this abstract level, we do not yet specify the type of this additional component. We parameterize the calculus by a set of clauses \( T_N \) that contains the conflicting clauses that can be used by the rules ConflOpt and Prune, and a predicate is_improving \( M \) \( M' \) \( O \) to indicate that Improve can be applied. For weights, the predicate is_improving \( M \) \( M' \) \( O \) means that the current trail \( M \) is a model, \( M' \) is the information that will be stored, and \( O \) is the currently stored information. \( T \) represents all the clauses that are entailed. We separate \( M \) and \( M' \) because we might want to store a different information than the current trail (e.g., to remove some literals that are not required to satisfy the formulas).
Corresponding CDCL Run
\((\epsilon, N, 0, \top, \top)\)  
\(\Rightarrow^\star\) Decide \((P^I Q^I, N, 0, \top, \top)\)  
\(\Rightarrow\) Improve \((P^I Q^I, N, 0, \top, P Q)\)  
\(\Rightarrow\) ConflOpt \((P^I Q^I, N, 0, \top, Q, P Q)\)  
\(\Rightarrow\) Backtrack \((P^I(\neg Q)^\top \lor \neg Q, N, \{\neg P \lor \neg Q\}, \top, P Q)\)  
\(\Rightarrow\) Improve \((P^I(\neg Q)^\top \lor \neg Q, N, \{\neg P \lor \neg Q\}, \top, P Q)\)  
\(\Rightarrow\) ConflOpt \((P^I(\neg Q)^\top, N, 0, P \lor \neg Q, P Q)\)  
\(\Rightarrow\) Backtrack \((-P^I, N, \{\neg P \lor \neg Q\}, P \lor \neg Q)\)  
\(\Rightarrow\) Propagate \((-P^I Q^I, N, \{\neg P \lor \neg Q\}, P \lor \neg Q)\)  
\(\Rightarrow^\star\) ConflOpt+Resolve \((\epsilon, N, \{\neg P \lor \neg Q\}, \bot, \neg P Q)\)

Corresponding CDCL Run
\((\epsilon, N \cup N_0, 0, \top, \top)\)  
where \(N_0 = 0\) and \(U_0 = \emptyset\)  
\(\Rightarrow^\star\) Decide \((P^I Q^I, N \cup N_0, U_0, \top)\)  
\(\Rightarrow\) Improve \((P^I Q^I, N \cup N_1, U_0, \top)\)  
where \(N_1 = N_0 \cup \{\neg P \lor \neg Q\}\)  
\(\Rightarrow\) Conflict \((P^I Q^I, N \cup N_1, U_1, \neg P \lor \neg Q)\)  
where \(U_1 = \{\neg P \lor \neg Q\}\)  
\(\Rightarrow\) Backtrack \((P^I(\neg Q)^\top \lor \neg Q, N \cup N_1, U_1, \top)\)  
\(\Rightarrow\) Improve \((P^I(\neg Q)^\top \lor \neg Q, N \cup N_1, U_1, \top)\)  
where \(N_2 = N_1 \cup \{P \lor \neg Q, \neg P \lor Q\}\)  
\(\Rightarrow\) Conflict \((P^I(\neg Q)^\top, N \cup N_2, U_1, \neg P \lor \neg Q)\)  
\(\Rightarrow^\star\) Backtrack \((-P^I, N \cup N_2, U_2, \top)\)  
where \(U_2 = U_1 \cup \{\neg P\}\)  
\(\Rightarrow\) Propagate \((P^I Q^I, N \cup N_2, U_2, \top)\)  
\(\Rightarrow^\star\) Conflict+Resolve \((\epsilon, N \cup N_2, U_2, \bot)\)

Figure 2. OCDCL transitions and corresponding CDCL transitions for \(N = \{P \lor Q\}\) with cost \(P = \text{cost } Q = 1\) and cost \(\neg P = \text{cost } \neg Q = 0\) where the horizontal lines separate two successive CDCL runs, separated by adding new clauses.

Precondition 5 (Preconditions on the clauses \(T_N(O)\) and on is Improving \(M M’ O\)). We require that:

- the atoms of \(T_N(O)\) are included in the atoms of \(N\); that is, no new variables are introduced.
- the clauses of \(T_N(O)\) do not contain duplicate literals, because duplicates are incompatible with the conflict analysis.
- if is improving \(M M’ O\), then \(T_N(O) \subseteq T_N(M’)\); i.e., extending the set of clauses is monotone.
- if is improving \(M M’ O\), then \(\neg M \in T_N(M’)\); i.e., the clause \(\neg M\) is entailed by \(T_N(M’)\) and can be used as conflict clause.

The rules ConflOpt_{BnB}, Improve_{BnB}, and Backtrack_{BnB} are defined as follows:

\begin{align*}
\text{ConflOpt_{BnB}} & \quad (M; N; U; T; O) \Rightarrow \text{OCDCL} \quad (M; N; U; \neg M; O) \\
\text{provided } \neg M & \in T_N(O) \\
\text{Improve_{BnB}}^\star & \quad (M; N; U; T; O) \Rightarrow \text{OCDCL} \quad (M; N; U; \neg M; M’) \\
\text{provided is improving } M M’ O & \text{ holds} \\
\text{Backtrack_{BnB}} & \quad (M; K M_1; N; U; D \lor L; O) \Rightarrow \text{OCDCL} \\
& \quad (L^D \lor L M_1; N; U \cup \{D’ \lor L\}; T; O) \\
& \quad \text{provided } L \text{ is of maximum level, } D’ \subseteq D, N + U + T_N(O) \models D’ \lor L, D’ \text{ and } K \text{ are of same level } i, \text{ and } i \text{ strictly less than the maximum level}
\end{align*}

We can simply embed into our CDCL formalization the states with the weights and reuse the previous definitions, properties, and invariants by mapping OCDCL states \((M, N, U, D, O)\) to CDCL states \((M, N, U, D)\). For example, we can reuse the Decide rule and the proofs on it.

Compared with 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 weaker version is sufficient to express conflict minimization as implemented in most SAT solvers [22], but the former is stronger and makes it possible for example to remove decision literals without cost from \(D\).

We use the Improve^\star rule instead of the Improve rule, because the latter is a special case of the former. The strategy favors Conflict and Propagate over all other rules. We do not need to favor ConflOpt over the other rules for correctness, although doing so helps in an implementation.
4.2 Embedding into CDCL

In order to reuse the proofs, we did previously about CDCL, CDCL_{BnB} is seen as a special instance of CDCL by mapping the states \((M; N; U; D; O)\) to the CDCL state \((M; N \cup T_N(O); U; D)\). This is not the mapping described in the previous mapping which mapped to states \((M; N; U; D)\). To distinguish between the two CDCLs, we will refer to the CDCL with the enriched set of clauses as CDCL_e.

In Isabelle, the most direct solution would be to instantiate the CDCL calculus with a selector returning \(N + T_N(O)\) instead of just \(N\). For technical reasons, we cannot do so: This confuses Isabelle, because it leads to duplicated 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 of the latter.

Except for the Improve rule, every OCDCL rule can be mapped to an CDCL_e rule: The ConflictOpt_{BnB} rule corresponds to the Conflict rule (because it can pick a clause from \(T_N(O)\)) and the extended Backtrack rule is mapped to CDCL_e’s Backtrack. On the other hand, the Improve rule has no counterpart and requires some new proofs, but adding clauses is compatible with the CDCL_e invariants.

In the CDCL formalization by Blanchette et al. [4], structural properties are distinguished from the strategy-specific properties. The latter properties ensure that the calculus does not get stuck in a state where we cannot conclude on the satisfiability of the clauses. Whereas the structural properties hold for CDCL_e, the strategy-specific properties do not necessarily hold for CDCL_e. The clause \(\perp\) might be in \(T_N(O)\) without being picked by the ConflictOpt_{BnB} rule. However, we can easily prove that they hold for CDCL_{BnB} and we can reuse the CDCL proof we have already done for most transitions. To reuse some proofs on CDCL’s Backtrack, we generalized some proofs by removing the assumption \(N + U \models D \lor L\) when not required. This is the only change that was required on the formalization of CDCL and it was only done to avoid duplication of proofs.

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

**Theorem 6 (CDCL_{BnB} Termination, Isa:wfc_cdlc_bnb ☐).** If Improve is well founded, then CDCL_{BnB} is also well founded.

**Theorem 7 (CDCL_{BnB} Termination, Isa:full_cdlc_bnb_stgy_no_conflicting_cls_unsat ☐ and Isa:no_step_cdlc_bnb_stgy_empty_conflict ☐).** If Improve can be applied when all literals are set in the trail \(M\) and \(M \models N + U\), then a run terminates in a state \((M', N, U, \perp, O)\) and the resulting set of clauses is unsatisfiable.

4.3 Instantiation with weights, OCDCL_g

Now we have developed an abstract CDCL_{BnB} framework, we can instantiate it 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:

- **local cost =**
  - \(\text{fixes cost} :: 'v literal multiset \Rightarrow 'c\)
  - assumes \(V\). consistent_interp \(B\)(
    - distinct_mset \(B \land A \subseteq B \rightarrow\)
    - \(\text{cost} \leq \text{cost} B\)

We assume that the function cost is monotone with respect to inclusion for consistent duplicate-free models. This is natural for trails, which 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

\[ T_N(O) = \{ \text{C. atom}(C) \subseteq \text{atom}(N) \land C \text{ is not a tautology nor contains duplicates} \land \{ \neg D. \text{cost}(D) \geq \text{cost}(O) \} \models C \} \]

\(\text{is}\_\text{improving} M M' O \leftrightarrow M'\) is a total extension of \(M, M \models N\), any total extensions of \(M\) has the same cost, and \(\text{cost} M < \text{cost} O\)

and then discharge the assumptions given in Precondition 5. OCDCL_g inherit from the invariants from CDCL_{BnB}. For termination, we only have to prove that Improve^+ terminates, which makes it possible to use Theorem 6 to prove termination. The key property of OCDCL_g is the following:

**Lemma 8 (Cheaper Models remains Models Isa:entails_too_heavy_clauses_too_heavy_clauses ☐).** If \(I\) is a total consistent model of \(N\), then either cost \(I\) \(\geq\) cost \(O\) or \(I\) is a total model of \(N \cup T_N(O)\).

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

By transitivity, since \(I\) is total, \(I\) is also a model of \(T_N(O)\) and therefore of \(N \cup T_N(O)\).

This is the proof that breaks if partial models are allowed: Literals of the clause \(D\) might not be defined. Some additional proofs are required to specify the content of the component \(O\). First, the sequence of literals \(O\) is always a total consistent model. This property cannot be inherited from the
correctness of CDCL, because it does not express any property about the component \( O \). The fact that the proof does not work for partial models does not mean the theorem is wrong, but it gave us the idea of the counterexample described in Section 3.

### 4.4 OCDCL

Finally, we can restrict 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\(^+\) 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\(_{\text{bnd}}\). The Prune rule is also seen as a special case of ConflictOpt\(_{\text{bnd}}\). Therefore, every transition is also a transition of OCDDL\(_{\gamma}\). Moreover, since final states of both calculi are the same, a completed run of OCDDL is also a completed run of OCDDL\(_{\gamma}\). Ultimately, the correctness theorem can be inherited.

Overall, the full formalization was easy to do, once we got the idea how to see OCDDL as a special case of a sequence CDCL runs without strategy. Formalizing a changing encoding is different than an already fixed version calculus, as done by Blanchette et al. [4]: we had to change our formalization several times to take into account additional rules: The Prune rule requires to use \( \{ D \mid (\neg C \cdot \text{cost}(C) \geq \text{cost}(O)) \} \vdash D \), while the set of clauses \( \{ C \cdot \text{cost}(C) \geq \text{cost}(O) \} \) is sufficient for Improve\(^+\).

### 5 Optimal Partial Valuations

To reduce the search from optimal partial valuations to optimal total valuations, we use the dual rail encoding [7, 19]. For every proposition variable \( P \), it creates two variables \( P^1 \) and \( P^0 \) indicating that \( P \) is defined positively or negatively. Adding the clause \( \neg P^1 \lor \neg P^0 \) ensures that \( P \) is not defined positively and negatively at the same time. The resulting 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\((L_1 \lor \cdots \lor L_n) := \text{penc}(L_1) \lor \cdots \lor \text{penc}(L_n)\), and penc\((C_1 \land \cdots \land C_m) := \text{penc}(C_1) \land \cdots \land \text{penc}(C_m)\). 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 the encoding penc\((N)\) of \( N \) the cost function is extended to a valuation \( \mathcal{A}' \) on \( \Sigma \cup \Sigma' \) by

\[
\text{cost}'(\mathcal{A}') = \text{cost}'(L) + \left( \{ L \mid L^1 \in \mathcal{A}' \} \cup \{ L \mid L^0 \in \mathcal{A}' \} \right).
\]

Let penc\((\mathcal{A}) : P \mapsto \begin{cases} 1 & \text{if } \mathcal{A}(P^1) = 1 \\ 0 & \text{if } \mathcal{A}(P^0) = 1 \end{cases} \) be a function that transforms a total model of penc \((N)\) into a model of \( N \) and penc\(^-\)(\( \mathcal{A} \)) does the opposite transformation, with penc\(^-\)(\( \mathcal{A} \))(\( P^1 \)) = 1 if \( \mathcal{A}(P) = 1 \), penc\(^-\)(\( \mathcal{A} \))(\( P^0 \)) = 0 if \( \mathcal{A}(P) = 0 \), penc\(^-\)(\( \mathcal{A} \))(\( P^0 \)) = 1 if \( \mathcal{A}(P) = 0 \), penc\(^-\)(\( \mathcal{A} \))(\( P^1 \)) = 0 if \( \mathcal{A}(P) = 1 \), unset otherwise.

**Paper Lemma 9** (Partial and Total Valuations Coincide Module penc, Isa:penc_ent_postp \( \checkmark \) and Isa:penc_ent_upostp \( \checkmark \). Let \( N \) be a clause set.

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

**Paper Lemma 10** (The Encoding penc Preserves Cost Optimal Models, Isa:full_encoding_OCDCL_correctness \( \checkmark \). Let \( N \) be a clause set and cost a cost function over literals from \( N \). If \( \mathcal{A}' \) is a cost-optimal total model for penc\((N)\) over cost, resulting in cost\((\mathcal{A}') = m \), then the partial model penc\((\mathcal{A}') \) is cost-optimal for \( N \) and cost\((\text{penc}(\mathcal{A}')) = m \).

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

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

**Non-Machine-Checked Lemma 11** (OCDDL on the Encoding). Consider a reasonable CDCL run on penc\((N)\). If rule \text{Decide} is restricted to deciding either \( P^1 \) or \( P^0 \) for any propositional variable, and Conflict Opt only considers the decision literals out of \( M \) as a conflict clause, then OCDDL performs at most \(3^n\) Backtrack steps.

**Paper Proof:** Using the strategy on \( P^1 \) or \( P^0 \), there are exactly three combinations that can occur on a trail: a decision \( P^1 \) and \( \neg P^0 \) by propagation, or the other way round, or \( \neg P^1 \) and \( P^0 \).

In summary, for each propositional variable, a run considers at most \(3^n\) cases, overall \(3^n\) cases for \( n \) different variables in \( N \).

### 6 Formalization of the Partial Encoding

In Isabelle, total valuations are defined by Herbrand interpretations, i.e., a set of all true atoms (all others being implicitly false) [20], but we work on partial models for CDCL, because they are similar to a trail. To reason on total models, Blanchette et al. [4] have defined a predicate to indicate whether a model is total or not. We distinguish between literals that can have a nonzero weight \( \Sigma' \) from the others \( \Sigma \setminus \Sigma' \) that can be left unchanged by the encoding.

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

\[
\text{interpretation OCDDL where cost = cost'}
\]
We only have to prove the proof obligation that the cost function cost' is monotone.

Finally, we can prove the correctness Theorem 10. The formalization is 800 lines long for the encoding, and 500 additional lines to restrict Decide.

We have not yet formalized the complexity bound of $3^n$ of Lemma 11, due to time constraints. So far, we have only verified the correctness of the variant of ConflOpt. It can be seen as a special case of conflict analysis and backtrack thanks to conflict minimization:

\[
(M_1K^\dagger M_2, N, U, \neg(M_1K^\dagger M_2)) \implies^* \text{Resolve} (M_1K^\dagger, N, U, \neg(M_1K^\dagger)) \implies^* \text{Backtrack} (M_1\neg K^D, N, U \cup \{D\}, \top)
\]

where $D'$ is the negation of the decisions of $M_1K^\dagger$ and $M_2$ does not contain any decision. If there are no decision in the trail, we set the conflict to $\bot$.

7 Solving Further Optimization Problems

In this section, we show two applications of OCDCL. First, it can be used to solve MaxSAT (Section 7.1) by adding new variables to the problem. Instead of considering the weight of literals, in MaxSAT the weight of clauses is considered.

Second, we apply our CDCL_{BnB} framework to another problem, model coverage (Section 7.2). Both extensions are verified using Isabelle.

7.1 MaxSAT

The maximum satisfiability problem (MaxSAT) is another well-known optimization problem [13]. It consists of two clause sets $N_H$ (hard constraints, mandatory to satisfy) and $N_S$ (soft constraints, optional to satisfy). The set $N_H$ comes with a cost function for clauses that are not satisfied. The aim is to find a total model with minimal cost.

Theorem 12 (Reduction of MaxSAT to OCDCL, Isa/partial_max_sat_is_weight_sat). Let $(N_H, N_S, \text{cost})$ be a MaxSAT problem and let active : $N_S \rightarrow \Sigma^*$ be an injective and surjective mapping for a set $\Sigma'$ of fresh propositional variables that assigns to each soft constraint an activation variable.

Let $I$ be the solution to the OPT-SAT problem $N = N_H \cup \{\text{active}(C) \lor C \mid C \in N_S\}$ with the cost function cost'$(L) = \text{cost}(C)$ if active$(C) = L$ for some $C \in N_S$ and cost'$(L) = 0$ otherwise.

If there is no model $I$ of $N$, the MaxSAT problem has no solution. Otherwise, $I$ without the additional atoms from $\Sigma'$ is an optimal solution to the MaxSAT problem.

Proof: • $N_H$ is satisfiable if and only if MaxSAT has a solution. Therefore, if there is no model $I$ of $N$, then $N_H$ is unsatisfiable.

• Let $I' = \{L \mid L \in I \land \text{atom}(L) \notin \Sigma'\}$. Let $J$ be any other model of $(N_H \cup N_S)$ and $J'$ its total extension to $\Sigma'$.

\[
J' = J \cup \{\text{active}(C) \mid C \in N_S \land J \not\models C\} \cup \{\neg \text{active}(C) \mid C \in N_S \land J \models C\} \text{ to } N.
\]

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

The problem of this MaxSAT encoding is that, in practice, it reduces the number of unit propagations that can be done, in particular if there are many soft constraints.

7.2 A Second Instantiation of CDCL_{BnB}: Model Covering

Our second example demonstrates that our framework can be applied beyond OCDCL. We consider the calculation of covering models. Again this is motivated by a product configuration scenario where a propositional variable encodes the containment of a certain component in a product. For product testing, finding a bucket of products is typically required such that every component occurs at least once in the bucket. Translated into propositional logic: given a set $N$ of clauses, we search for a set of models $M$ such that for each propositional variable $P$ occurring in $N$, $M \models P$ for at least one $M \in M$ or there is no model of $N$ such that $P$ holds.

In order to solve the model covering problem, we define a domination relation: A model is dominated if there is another model that contains the same true variables and some others. More formally, if $I$ and $J$ are total models for $N$, then $I$ is dominated by $J$ if $\{P \mid I \models P\} \subseteq \{Q \mid J \models Q\}$. If a total model is dominated by a model already contained in $M$, then it is not required in the set of covering models. The extension to CDCL are the two additional rules:

\[
\text{Conflic} \quad (M; N; U; \tau; M) \implies_{\text{MCCDCL}} (M; N; U; \neg M; M)
\]

provided for all total extensions $MM'$ with $MM' \models N$, there is an $I \in M$ which dominates $MM'$.

\[
\text{Add} \quad (M; N; U; \tau; M) \implies_{\text{MCCDCL}} (M; N; U; \tau; M \cup \{M\})
\]

provided $M \models N$, all literals from $N$ are defined in $M$ and $M$ is not dominated by a model in $M$.

The full calculus called MCCDCL calculus does not necessarily compute a minimal set of covering models. Minimization is a classical NP-complete problem [11] and can then be done in a second step. The minimal model covering can be computed by creating another CDCL extension, where the set $M$ is explicitly added as a component to a state and used for a branch-and-bound optimization approach, similar to OCDCL [15]. This calculus is another instance of CDCL_{BnB}. In the formalization, we instantiate CDCL_{BnB} with:
\[ \mathcal{T}_N(M) = \{ \mathcal{C} \in \text{atom}(C) \subseteq \text{atom}(N) \wedge \mathcal{C} \text{ is not a tautology nor contains duplicates} \wedge \{ -D \text{ is dominating } M, \text{ total} \cup N \not\models \mathcal{C} \} \text{ is improving } M \to M' \leftrightarrow \]
m\text{ and } M \text{ is not dominated by } M' \] and } M \text{ is consistent, total, duplicate free}.

Compared with OCDCL, \( \mathcal{T}_N(\emptyset) \) is never empty, because it contains at least the clause set \( N \). This is another reason why the CDCL with the \( \mathcal{T}_N(M) \) cannot run with the reasonable strategy: If \( N \) is unsatisfiable, we would have to pick the conflict clause \( \bot \) immediately, which is exactly the problem we are solving with CDCL.

**Theorem 13** (Correctness Isa-cdclcm_correctness). If the clauses in \( N \) do not contain duplicated literals, then a MCCDCL starting from \( (\epsilon, N, \emptyset, \top, \emptyset) \) and in a state \( (\epsilon, N, U, \bot, M) \), and for every variable \( P \) in \( N \), there is a model \( M \) of \( N, M \in M, \) where \( M \models P \), or there is no model satisfying both \( P \) and \( N \).

The proof involves a lemma similar to Lemma 8: Every model is dominated by a model in \( M \) or is still a model of \( N \cup \mathcal{T}_N(M) \).

## 8 Related Work and Conclusion

There are several formalizations of CDCL beyond ours, as discussed in the article [5, Section 6], but we are not aware of any formalization of an optimizing CDCL calculus, or more abstract, 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 [12]. 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 rule Improve*. Moreover, the first unique implication point is built into our calculus. The pruning rule can be simulated by 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. [21]. It assumes that negative literals do not cost anything: This means that the opposite of \( L \) is \( \neg L \) (because \( \neg L \) and \( L \) undefined have the same weight).

Liberatore has developed a variant of DPLL to solve this problem [14]. 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 [9] to solve MAX-SAT. 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.

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 formalization fits nicely into the framework that has previously developed and the used abstraction in Isabelle to simplify reuse and studying variants and extensions.

We started our encoding for cost-minimal models with respect to partial valuations by introducing three extra variables for each variable, where, compared to the dual rail encoding, the third extra variable explicitly modeled whether a variable is defined or undefined [24]. We performed the content of Section 5 and Section 6 with this encoding and only afterwards were pointed by a reviewer to the dual rail encoding. It took us half a day to redo the overall formalization. For us this is another example that the reuse of formalizations can work. This is further demonstrated by the application of the OCDCL results to MAX-SAT and the reuse of the formalization framework to verify the model covering calculus MCCDCL, Section 7. Minimization of the model covering set computed by MCCDCL can also be solved by an afterwards application of a CDCL calculus with branch-and-bound [15], and would probably fit in our framework.

On an abstract level, OCDCL is close to an incremental version of CDCL(T), the calculus used in several modern SMT solvers. The main difference is that the conflicts generated are not the negation of the trail, but implied by the theory. The theory of linear arithmetic (LA) has already been verified in Isabelle/HOL by Thiemann [6, 23], so proving correctness of CDCL(LA) does not need a from-scratch new effort.

**Acknowledgments**

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

**References**


