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

\textbf{Abstract.} Based on our formal framework for CDCL (conflict-driven clause learning) verified using the proof assistant Isabelle/HOL, we formalize an optimizing calculus based on branch and bound, OCDCL. It finds optimal models of minimum cost with respect to total valuations. Through an encoding, we also show that a variant finds cost optimal models of minimum cost with respect to partial valuations. 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.

\section{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{isafol} 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{weidenbach2020}, 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 based on his CDCL calculus by copy-pasting the proofs. As a case study, we wanted to formalize this variant. 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 of the semantics to optimal models based on total valuations. The developed OCDCL calculus is similar to Larrosa et al.’s DPLL_{BB} calculus [9], 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 an encoding and a rigorous proof that OCDCL is also correct with respect to partial valuations on the encoding. To the best of our knowledge, the encoding is also the first correct solution to computing optimal-cost models with respect to partial valuations. Our results support the often-stated motivation for formalizations in that we could find and eventually solve the issue with respect to the two types of valuations this way. Furthermore, the often-mentioned reuse of formal results also applies to our extension from CDCL to OCDCL. 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 [3], and 4500 lines for Nieuwenhuis et al.’s account [12]. The formalization took around 1.5 month work. The extension to partial valuations amounts to 1300 lines. The Isabelle formalization can be found online with a correspondence between the theorems and their Isabelle counterpart.\footnote{https://bitbucket.org/isafol/isafol/src/master/Weidenbach_Book/}

Now the paper is organized as follows. After some preliminaries on the formalization of CDCL (Section 2), the optimizing calculus, called OCDCL, 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. It is well suited for a formalization as its core rules are exactly the rule of the calculus we have formalized earlier (Section 4). 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. 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). This was also formalized in Isabelle (Section 6).

2 Formalization of CDCL in Isabelle

Isabelle Isabelle [13, 14] is a generic proof assistant that supports many object logics. The metalogic is an intuitionistic fragment of higher-order logic (HOL) [5].
The types are built from type variables \( \alpha, \beta, \ldots \) and \( n \)-ary type constructors, normally written in postfix notation (e.g., \( \alpha \text{ list} \)). The infix type constructor \( \alpha \Rightarrow \beta \) is interpreted as the (total) function space from \( \alpha \) to \( \beta \). Function applications are written in a curried style (e.g., \( f x y \)). Anonymous functions \( x \mapsto y \) are written \( \lambda x.y \). The judgment \( t :: \tau \) indicates that term \( t \) has type \( \tau \).

Propositions are simply terms of type \( \text{prop} \). Symbols belonging to the signature are uniformly called \( \text{constants} \), even if they are functions or predicates. The metalogical operators include universal quantification \( \forall :: (\alpha \Rightarrow \text{prop}) \Rightarrow \text{prop} \) and implication \( \Rightarrow :: \text{prop} \Rightarrow \text{prop} \Rightarrow \text{prop} \). The notation \( \forall x. p_x \) is syntactic sugar for \( \forall (\lambda x. p_x) \) and similarly for other binder notations.

Isabelle/HOL is the instantiation of Isabelle with HOL, an object logic for classical HOL extended with rank-1 (top-level) polymorphism and Haskell-style type classes. It axiomatizes a type \( \text{bool} \) of Booleans as well as its own set of logical symbols (\( \forall, \exists, \text{False}, \text{True}, \neg, \land, \lor, \rightarrow, \leftrightarrow, = \)). The object logic is embedded in the metalogic via a constant \( \text{Trueprop} :: \text{bool} \Rightarrow \text{prop} \), which is normally not printed. The distinction between the two logical levels is important operationally but not semantically.

Isabelle adheres to the tradition initiated in the 1970s by the LCF system [8]; 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

\[
\text{locale} \ X = \text{fixes} \ c :: \tau_a \\text{ assumes} \ A_{a,c}
\]

The definition of locale \( X \) implicitly fixes a type \( \alpha \), explicitly fixes a constant \( c \) whose type \( \tau_a \) may depend on \( \alpha \), and states an assumption \( A_{a,c} :: \text{prop} \) over \( \alpha \) and \( c \). Definitions made within the locale may depend on \( \alpha \) 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 \( \alpha \), 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 \texttt{interpretation} \( X \) where \( c = c' \) emits the proof obligation \( A_{v,t} \), where \( v \) and \( t :: \tau_v \) may depend on types and constants from \( Y \). After the proof, all the lemmas proved in \( X \) become available in \( Y \),
with \( \alpha \) and \( c :: \tau_\alpha \) instantiated with \( \nu \) and \( t :: \tau_\nu \). Similarly, a locale can be interpreted by specifying the constants and the types parametrizing the locale. The assumptions must be discharged. For example, the following interprets the locale \( X \) by setting the parameter \( c' \) for parameter \( c \).

CDCL We have previously developed a framework to formalize the conflict-driven-clause-learning procedure [4] in Isabelle. Clauses are defined as (finite) multisets. For readability, we will write \( \bot \) and \( A \lor B \) instead of their Isabelle counterpart \( \{ \# \} \) and \( A + B \). We define entailment by \( I \models L \) iff \( L \in I \). We can lift it to clauses by \( I \models C \) iff there is a literal \( L \in C \) such that \( I \models L \). We can also lift it to clause sets \( I \models N \) iff \( \forall C \in N. I \models C \).

The conflict-driven clause learning is a procedure that builds a candidate model, called the trail or \( M \). 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. 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:

**Decide** \((M; N; U; \top) \Rightarrow_{\text{CDCL}} (ML; N; U; \top)\)

provided \( L \) is undefined in \( M \), atom \( L \in N \) and \( k \) is the number of decisions in \( M \).

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:

\[
\text{inductive decide :: } 'st \Rightarrow 'st \Rightarrow \text{bool where}
\text{undefined_lit } (\text{trail } S) L \Rightarrow
L \in \text{atom } (\text{clauses } S) \Rightarrow
S' \sim \text{append_trail } (\text{Decided } L) S \Rightarrow
\text{decide } S S'
\]

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.

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 with more complicated data structures further to be able to simply add two-watched literals by instantiating the locales. Due to the handling this indirection between the real concrete representation and the fields that can be accessed, we use a different equality: \( S \sim T \) if all components are equal.

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

A partial \( \Sigma \)-valuation is a partial mapping \( A \): \( \Sigma \rightarrow \{0,1\} \) from the set of propositional variables \( \Sigma \) into \( \{0,1\} \). For any atom \( P \in \Sigma \), we write \( A(P) \downarrow \) if \( A \)
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: \( \mathcal{A}(\neg P) := 1 - \mathcal{A}(P) \) if \( \mathcal{A}(P) \downarrow \) and undefined 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.

Note that in case a partial \( \Sigma \)-valuation is total, the above definition coincides with the classical definition of a valuation. So a partial valuation is a generalization of the classical total valuation. Like there are \( 2^n \) total valuations for \( | \Sigma | = n \), there are \( 3^n \) partial valuations.

We assume a total cost function \( \text{cost} \) on the set of all literals \( \text{Lit}(\Sigma) \) over \( \Sigma \) into \( \mathbb{K}^+ \), \( \text{cost} : \text{Lit}(\Sigma) \to \mathbb{K}^+ \). \( \mathbb{K}^+ \) 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 \( \text{cost}(L, \mathcal{A}) := \text{cost}(L) \) 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 (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 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 \( \text{cost}(\top) = \infty \).

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 [20], 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. The first few rules use the trail:

**Propagate** \((M; N; U; \top; O) \rightarrow_{\text{OCDCL}} (ML_{\perp}; N; U; \top; O)\)

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

**Decide** \((M; N; U; \top; O) \rightarrow_{\text{OCDCL}} (ML_{\bot}; N; U; \top; O)\)

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

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

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** \((ML_{\perp}; N; U; D; O) \rightarrow_{\text{OCDCL}} (M; N; U; D; O)\)

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

**Resolve** \((ML_{\perp}; N; U; D \lor \text{comp}(L); O) \rightarrow_{\text{OCDCL}} (M; N; U; D \lor C; O)\)

provided \( D \) is of level \( k \), where \( k \) is the number of decisions in \( M \)
Backtrack \((M_1 K^\dagger M_2; N; U; D \lor L; O) \Rightarrow_{\text{OCDCL}} (M_1 L^{D\lor L}; N; U \cup \{D \lor L\}; \top; O)\)

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

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

ConfOpt \((M; N; U; \top; O) \Rightarrow_{\text{OCDCL}} (M; N; U; \neg M; O)\)

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

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

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

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

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, but can increase performance. In practice, Prune is 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.

The idea behind OCDCL is to enumerate models. However, 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 \(\text{cost}(P) = 3\), \(\text{cost}(\neg P) = \text{cost}(Q) = \text{cost}(\neg Q) = 1\). 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 current best bound on an optimal-cost model. After backtracking, it can propagate \(Q\) with trail \([\neg P, P \lor Q]\) 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. The problem is that 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). 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:

\[
\mu((M; N; U; D; O)) = \begin{cases} 
(3^n - 1 - |U|, 1, n - |M|, \text{cost}(O)), & D = \top \\
(3^n - |U|, 0, |M|, \text{cost}(O)), & \text{else} 
\end{cases}
\]
The measure is decreasing since no clause is relearned.

\[\square\]

**Theorem 3 (OCDCL Correctness).** 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 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**\(^+\) \( (M; N; U; \top; O) \Rightarrow \text{OCDCL} \ (M; N; U; \top; MM') \)

provided \( M \models N \), \( MM' \) is any 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**\(^+\)). In OCDCL, the rule Improve can be replaced by rule Improve\(^+\): 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 [2]: 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

We want to formalize OCDCL to make sure that it is correct. The proof is done in four steps: (1) We define a more abstract branch-and-bound calculus, CDCL\(\text{BnB}\). This calculus relies on an additional unspecified set of clauses. (2) Except for the Improve rule that adds clauses to this unspecified set of clauses, CDCL\(\text{BnB}\) is seen as a special case of CDCL, where the additional clauses are part of the initial set of clauses. This makes it possible to inherit proofs and some correctness arguments. (3) We instantiate the branch-and-bound calculus with the weight function to get a generalized version OCDCL\(_g\). (4) Finally, we specialize OCDCL\(_g\) to get OCDCL from Section 3.

#### 4.1 Assumptions

We create a locale with several assumptions that are implicit in the previous section:
locale OCDCL =  
fixes Σ :: 'v set and  
ΔΣ :: 'v set and  
assumes  
finite (ΔΣ) and  
ΔΣ ⊆ Σ and  
inj_on (λA.A^1) ΔΣ and  
inj_on (λA.A^0) ΔΣ and  
inj_on (λA.A') ΔΣ and  
(λA.A^0) ΔΣ ∩ (λA.A^1) ΔΣ = ∅ and  
(λA.A^0) ΔΣ ∩ (λA.A') ΔΣ = ∅ and  
(λA.A^1) ΔΣ ∩ (λA.A') ΔΣ = ∅ and  
∀ C. atom C ∈ Σ − ΔΣ. cost (CM) = cost (C).

We assume that the set ΔΣ is finite: This is in practice always the case, since we consider a finite set of formulas. Then we assume the additional variables are fresh (i.e., not in Σ) and distinct: The functions (λA.A^0), (λA.A^1), and (λA.A') are injective on ΔΣ (predicate inj_on) and they generate different literals. This assumption is implicit in the notation, but must be made explicit in Isabelle.

### 4.2 Branch-and-Bound Calculus, CDCL\_BnB

We use a similar approach to our previous formalization with an abstract state and selectors, but 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 assume the existence of two additional information: a predicate is\_improving \( M M' O \) and a separate set of clauses \( T \). For weights, the predicate is\_improving \( M M' O \) means that trail \( M \) is a model, \( M' \) the information that will be stored and \( O \) the current stored information.

We assume the existence of a separate set of clauses (depending on the state), \( T \). This set of clauses represents all the clauses that are entailed for an external reason. We require that:

- the atoms of \( T_N (O) \) are included in the atoms of \( N \);
- the clauses of \( T_N (O) \) do not contain duplicate literals;
- if is\_improving \( M M' O \), then \( T_N (O) \subseteq T_N (M') \)
- if is\_improving \( M M' O \), then \( ¬M \in T_N (M') \)

Instead of writing properties directly on costs, we use \( T \). For example, the rules ConfOpt\_BnB, Improve\_BnB, and BackTrack\_BnB are defined as follows:

- **ConfOpt\_BnB** \( (M; N; U; k; \top; O) \implies_{OCDCL} (M; N; U; k; ¬M; O) \) provided \( ¬M \notin T_N (O) \)
- **Improve\_BnB** \( ^+ (M; N; U; k; \top; O) \implies_{OCDCL} (M; N; U; k; ¬M; M') \) provided is\_improving \( M M' O \)
- **Backtrack\_BnB** \( (M_1 K^{i+1} M_2; N; U; D \lor L; O) \implies_{OCDCL} (M_1 L^{D \lor L}; N; U \cup \{D' \lor L\}; \top; O) \)
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.

There is no Prune rule yet. CDCL’s Backtrack rule is not reused for OCDCL. Unlike the version from Section 3, we make it possible to have conflict-clause minimization (as we have done for CDCL’s backtrack earlier [6]): Instead of $D \lor L$, a clause $D' \lor L$ is learned such that $D' \subseteq D$ and $N + U + T_N (O) \models D' \lor L$.

In contrast, if we had reused Backtrack from CDCL, only the weaker entailment $N + U \models D' \lor L$ would be used. This is not required for conflict minimization as implemented in most SAT solvers [18], but makes it possible 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 conflictOpt over the other rules for correctness, although preferring confiOpt over Decide helps in an implementation.

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 $O$.

### 4.3 Embedding into CDCL

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

For technical reasons, we cannot instantiate the CDCL locale with the selector for init clauses returning $N + T_N (O)$. This actually confuses Isabelle. Instead we instantiate the CDCL with tuples and add a conversion function to map the states:

```plaintext
interpretation CDCL where
  trail = (λ(M, N, U, D, O). M) and
  init_class = (λ(M, N, U, D, O). N + T_N (O)) and
  learned_class = (λ(M, N, U, D, O). U) and
  conflicting = (λ(M, N, U, D, O). D)
```

Except for the Improve rule, every OCDCL rule can be mapped to a CDCL rule: The ConflictOpt$_{BnB}$ rule corresponds to the Conflict rule (because it can also take all clauses, including from $T_N (O)$) 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 (as long as the new clauses do not contain duplicate literals).

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 $T_N(O)$ without being picked by the ConflictOpt$_{BnB}$ rule. However, we can easily prove that they hold for CDCL$_{BnB}$: We reuse the proof we have already done for most transitions.

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.

Not all transitions of CDCL can be taken by OCDCL: Propagations of clauses in $T_N(O)$ are not possible in OCDCL (but can be emulated by decisions and conflict analysis).

With the additional assumptions that Improve can always be applied when the trail is a total model, we show that the final set of clauses is unsatisfiable, in which case the conflict clause is $\bot$.

### 4.4 Instantiation with weights, OCDCL$_g$

Finally, we can instantiate $T$ 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:

```isabelle
locale cost = 
  fixes cost :: 'v literal multiset ⇒ 'c
  assumes 
    ∀C. consistent_interp B ∧ distinct_mset B ∧ A ⊆ B → cost A ≤ cost B
```

We also assume that the type 'c has a linear order. We only 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 than the cost is a sum over the literals. We take

$$T_N(O) = \{C. \text{atom}(C) ⊆ \text{atom}(N)$$
$$\land C \text{ is not a tautology nor contains duplicates}$$
$$\land \{\neg C. \text{cost}(C) ≥ \text{cost}(O)\} \models C\}$$

is_improving $M M' O = \{M' \text{ is a total extension of } M', M \models N,$
$$\text{any total extensions of } M, \text{ has the same cost and}$$
$$\text{cost } M < \text{cost } O\}.$$

discharge the assumptions over it.

OCDCL$_g$ inherit from invariants. For termination, we only to prove that Improve$^+$ terminates to reuse the proof we already made on CDCL$_{BnB}$. We then prove that $O$ contains a model. An important property is the following theorem:

**Isabelle Lemma 5** If $I$ is a total consistent model of $N$, then either cost $(I) ≥$ 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 a model of \( T_N,O\).

Some additional proofs are required to specify the content of the component \( O \). First, \( O \) always contains a total consistent model. Second, we prove that \( O \) contains an optimal model at the end of an OCDCL run.

4.5 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 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. The Prune rule is also seen a special case of ConflictOpt. Therefore, every transition is also a transition of CDCLg. Moreover as final states of both calculi are the same too, as any full run of OCDCL is also a full run of CDCLg. 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 (like the Improve rule).

5 Optimal Partial Valuations

The idea is to simulate the partial valuation semantics by the total valuation semantics through an encoding penc\( (N) \) of the clause set \( N \). For every propositional variable \( P \), penc adds an additional fresh variable \( P' \) to indicate whether \( P \) is defined in a partial valuation. The function penc is defined such that if \( A \models N \) for partial \( A \) then there is a total \( A' \models \text{penc}(N) \), and, \( A(P)\downarrow \) if \( A'(P') = 1 \), and if \( A(P)\downarrow \) then \( A(P) = A'(P) \). Furthermore, if \( A' \models \text{penc}(N) \) for total \( A' \), then there exists a partial \( A \models N \) such that (i) \( A(P)\downarrow \) if \( A'(P') = 1 \), and, (ii) if \( A(P)\downarrow \) then \( A(P) = A'(P) \).

The encoding penc is defined on literals by \( \text{penc}(P) := (P \land P') \), \( \text{penc}(\neg P) := (\neg P \land P') \), and lifts to clauses and clause sets by \( \text{penc}(L_1 \lor \cdots \lor L_n) := \text{penc}(L_1) \lor \cdots \lor \text{penc}(L_n) \), and, \( \text{penc}(C_1 \land \cdots \land C_m) := \text{penc}(C_1) \land \cdots \land \text{penc}(C_m) \). Note that penc\( (N) \) is no longer in CNF.

Given an encoding penc\( (N) \) of a clause set \( N \) the cost function is extended to a valuation \( A' \) on \( \Sigma \cup \Sigma' \) by \( \text{cost}'(A') := \sum_{L \in L(\Sigma)} \text{cost}'(A,L) \) where \( \text{cost}'(L,A) := \text{cost}(L) \) if \( A' \models L \land \text{atom}(L)' \) and \( \text{cost}'(L,A) := 0 \) otherwise. Furthermore, \( \text{cost}(A,L) := 0 \) for \( L \in \Delta(\Sigma') \).

Let \( \text{pdec}(A) : P \rightarrow \begin{cases} A(P) & \text{if } A(P') = 1 \\ \text{unset} & \text{otherwise} \end{cases} \) the function that process a total model of penc\( N \) into a model of \( N \).
Lemma 6 (Partial and Total Valuations Coincide Modulo penc). Let \( N \) be a clause set.

1. If \( A \models N \) for partial \( A \) then there is a total \( A' \models penc(N) \) where (i) \( A(P) \downarrow \) iff \( A'(P') = 1 \); and (ii) if \( A(P) \downarrow \) then \( A(P) = A'(P) \).

2. If \( A' \models penc(N) \) for a total \( A' \), then \( pdec(A') \models N \).

Proof. 1. Define \( A'(P') := 1 \) if \( A(P) \downarrow \) and \( A'(P') := 0 \) otherwise. Define \( \overline{A'}(P) := A(P) \downarrow \) and \( \overline{A'}(P) := 0 \) otherwise. For \( A' \models penc(N) \) it is sufficient to show that \( \overline{A}(L) = 1 \) iff \( A'(\overline{penc}(L)) = 1 \). If \( \overline{A}(-P) = 1 \) then \( A'(-P) = 1 \) and \( A'(P') = 1 \), hence \( A'(\overline{penc}(-P)) = 1 \). The other properties hold obviously by construction.

2. For \( pdec(A) \models N \) it is sufficient to show that \( pdec(A)(L) = 1 \) iff \( A'(\overline{penc}(L)) = 1 \). The proof is almost identical to case 1. The other properties hold obviously by construction. \( \square \)

Note that for the total valuation construction of Lemma 6.1 we could also have chosen \( A'(P) := 1 \) in case \( A(P) \) is not defined.

Lemma 7 (penc Preserves Cost Optimal Models). 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 \( \text{cost}'(A') = m \), then a cost-optimal partial model \( A \) for \( N \) has also \( \text{cost}(A) = m \).

Proof. By contradiction. Assume there is a partial model \( A \) for \( N \) with \( \text{cost}(A) = k \), \( k < m \). Now define a total model \( A' \) for \( penc(N) \) by \( A'(P) := A(P) \) and \( A'(P') := 1 \) if \( A(P) \downarrow \) and \( \overline{A'}(P) := 0 \) and \( \overline{A'}(P') := 0 \) otherwise. It is easy to see that \( A' \models penc(N) \) and that \( \text{cost}'(A') = k \), a contradiction. \( \square \)

If \( N \) is a set of clauses then \( penc(N) \) is no a longer in CNF, but a set of disjunctions of conjunctions of the form \( L \land P' \) where \( P = \text{atom}(L) \). A straightforward naive CNF transformation results in a worst case exponential blowup in the number of clauses. Instead for every variable \( P \) we introduce two further variables \( P^1 \) and \( P^0 \). Where \( P^1 \) means “\( P \) is defined” and the new variable \( P^1 \) stands for “\( P \) is defined and true” and \( P^0 \) stands for “\( P \) is defined and false”.

The two new variables take the role of renamings [15], so \( P^1 \leftrightarrow (P \land P') \) and \( P^0 \leftrightarrow (\neg P \land P') \). Then \( penc(N) \) is satisfiable iff \( penc\left(N[\neg P/P^0, P/P^1]\right) \land (P^1 \leftrightarrow (P \land P')) \land (P^0 \leftrightarrow (\neg P \land P')) \) is satisfiable. Eventually, we obtain a CNF that can be obtained from \( N \) be replacing every literal \( \neg P \) with \( P^0 \), every literal \( P \) with \( P^1 \), and add the equivalences for \( P^0 \) and \( P^1 \). The defining equivalences for \( P^1 \) and \( P^0 \) result in six clauses after a CNF transformation.

\[
\begin{align*}
\neg P^0 \lor \neg P & \quad \neg P^1 \lor P \\
\neg P^0 \lor P' & \quad \neg P^1 \lor P' \\
P \lor \neg P' \lor P^0 & \quad \neg P \lor \neg P' \lor P^1
\end{align*}
\]

In order to ease the proof of the Lemma 9 below, we add a further clause expressing that for each variable \( P \) it must be true, false, or undefined: \( P^0 \lor
$P_1 \lor \neg P'$. It is a logical consequence out of the other six clauses, but requires a decision to derive that information. In summary, if there are $n$ variables in $N$ then the final clause set called $\text{ren}^+ (\text{penc}(N))$ after the encoding and replacement of the conjunctions and clausification has the size $|N| + 7n$. Recall that for $n$ propositional variables there are $2^n$ total valuations and $3^n$ partial valuations. The clauses in $\text{ren}^+ (\text{penc}(N))$ that are the result renamings of clauses in $N$ solely consists of positive literals $P_1, P_0$.

**Lemma 8 (OCDCL on the Encoding).** Consider a reasonable OCDCL run on $\text{ren}^+ (\text{penc}(N))$. If rule Decide is restricted to deciding either $P_1$ or $P_0$ or $\neg P'$ 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 partial valuation satisfying $N$.

**Proof.** First note that if there is a decision $P_0$ via propagation $\neg P, P'$, and $\neg P_1$ are derived. Analogously, a decision of $P_1$ results in $P, P'$, and $\neg P_0$ by propagation. A decision $\neg P'$ results in both $\neg P_1$ and $\neg P_0$ by propagation but leaves $P, \neg P$ undefined. Thus if there is no conflict then it may be that for some propositional variables neither $P$ nor $\neg P$ are on the trail. But in this case their value can be arbitrarily chosen because $\neg P'$ is on the trail and together with $\neg P_1$ and $\neg P_0$ this satisfies the seven clauses introduced by $\text{ren}^+$.

**Non-Machine-Checked Lemma 9 (OCDCL on the Encoding)** Consider a reasonable CDCL run on $\text{ren}^+ (\text{penc}(N))$. If rule Decide is restricted to deciding either $P_1$ or $P_0$ or $\neg P'$ 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 classical DPLL run without learning on the clause set $N' = \text{ren}^+ (\text{penc}(N))$ resulting from $N$ after encoding and clausification. 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 DPLL will flip the most recent decision on some $P_1, P_0, \text{or } \neg P'$ generating the complement. Now out of the clause $P_0 \lor P_1 \lor \neg P'$, one literal becomes false, and two undefined. The strategy decides then one of the two remaining literals. Again, if this result in a conflict, the clause is propagating afterwards. In summary, for each propositional variable, a DPLL run considers at most 3 cases, overall $3^n$ cases for $n$ different variables in $N$.

OCDCL’s backtracking is no longer chronological and learned clauses are learned. However, based on the above argument on DPLL, it is sufficient to show that OCDCL only learns clauses consisting of literals with atoms $P_1, P_0$ or $P'$ but never $P$. This is by construction true for the rule ConflOpt, because it only considers decision literals. So it remains to show that an application of Backtrack after ConflSat learns a clause without a $P$ literal. A literal with atom $P$ only occurs in the clauses $\neg P_0 \lor \neg P$, or a $\neg P_1 \lor P$ or a $P \lor \neg P' \lor P_0$ or a $\neg P \lor \neg P' \lor P_1$ out of $N'$. In all cases, if it eventually occurs in a conflict clause, it is resolved away before Backtracking. Note that the set $N'$ does not contain
any occurrences of \( P \) literals and we never decide a \( P \) literal. Therefore, also Backtrack after ConflOpt only learns clauses build over literals with atoms \( P_1 \), \( P_0 \) or \( P' \).

\[ \square \]

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 developed for the development of Herbrand interpretations [16]. This, however, is not really adapted to the verification of CDCL. Therefore, we are already using partial models, similar to a trail, except that the order does not matter. We then use predicates to require that a model is consistent (consistent_interp), total, and does not contain duplicate literals (distinct_mset).

Because we want to be able to use the OCDCL calculus that works only on formulas in CNF, we define the transformation with the introduction of variables, instead of working on general formulas.

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

\[
\text{interpretation OCDCL where}
\]

\[
\text{cost} = \text{cost}'
\]

We have to prove that cost' is monotone. We write \( S \rightarrow t_{CDCL_{BnB,\text{stgy}}} T \) to indicate that CDCL_{BnB,\text{stgy}} has run from \( S \) to \( T \) and no further transition is possible. Finally, we can prove the following correctness theorem:

Isabelle Theorem 10 (Formalized version of Lemma 7)

assumes

\[
(\text{init\_state}(penc N)) \rightarrow t_{CDCL_{BnB,\text{stgy}}} T \text{ and all clauses of } N \text{ are distinct and atom } N = \Sigma \text{ and weight } T \neq \text{None}
\]

shows distinct_mset I \(
\rightarrow\) consistent_interp I \(
\rightarrow\) atom I \(\subseteq\) \(\Sigma\) \(
\rightarrow\) I \(=\) N \(\rightarrow\) cost I \(\geq\) cost (pdec (weight T))

Unlike Lemma 7, the source of the optimal model is hard-coded inside the theorem. Otherwise, the proof is similar. The formalization is 800 lines long for the encoding, and 500 additional lines for the restriction of decide. It is developed as a variant of CDCL_{BnB} that is connected to CDCL_{BnB} by showing that the latter includes the transition of the former and that they have the same final states.

We have not yet formalized Lemma 9, but plan to do so. The classical DPLL backtrack can be seen as a special case of conflict analysis and backjumping thanks to conflict minimization: \((M_1 K^\dagger M2, N, U, \neg(M_1 K^\dagger M2)) \rightarrow t_{\text{Resolve}} (M_1 K^\dagger, N, U, \neg(M_1 K^\dagger)) \rightarrow t_{\text{Backtrack}} (M_1 \neg K^\dagger(M_1 K^\dagger), N, U \cup \{D'\}, \top), \) where \( D' \) is the negation of the decisions of \( M_1 K^\dagger \). The main issue is to transfer the global property (3^n models) to a local property stating that after a Backtrack, some measure is decreasing.
7 Related Work and Conclusion

There are several formalization of CDCL beyond ours, as we discuss in our previous article [4, 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 [9]. 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\(^+\). Moreover, the first unique implication point is built in our calculus. The Pruning rule can be simulated by applying their Learn rule: \(\neg M \lor c \geq \text{cost} \leq O \) is entailed by the clauses.

A related problem to finding the minimum partial model is called Minimum-Weight Propositional Satisfiability by Sebastiani et al. [17]. It assumes that negative literals do not cost anything: This means that the opposite \(L\) is \(\neg L\) (as \(\neg L\) 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 methods builds total models. Liberatore has developed a variant of DPLL to solve this problem [11]. 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 [7] to solve MAX-SAT. The later problem consists in finding a model that satisfies all mandatory constraints and as many soft clauses as possible. Unlike the problem presented here, the clauses are weighted, not the literals.

We have presented here a variant of CDCL to find optimal models and an encoding on reducing 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 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 implemented in Isabelle/HOL by Thiemann [19], so formally proving correctness of CDCL\((T)\) does not need a from scratch new effort. Furthermore, OCDCL can be used as a basis for formalizing and proving correctness of MAX-SAT [10].

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 [4]. We thank Armin Biere and Roberto Sebastiani for a number of helpful discussions.
References


