A Verified SAT Solver Framework including Optimization and Partial Valuations

Mathias Fleury[1,2], Christoph Weidenbach[1], and Dominic Zimmer[1,3]

[1] Max-Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
{mathias.fleury,weidenb}@mpi-inf.mpg.de
[2] Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
s8mafleu@stud.uni-saarland.de
[3] Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
s8dozimm@stud.uni-saarland.de

Abstract. Based on our formal framework for CDCL (conflict-driven clause learning) verified using the proof assistant Isabelle/HOL, we verify an optimizing 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 CDCL_{BnB} to find a minimal set of covering models. 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 an optimizing CDCL calculus and the first solution that computes cost-optimal models with respect to partial valuations.

1 Introduction

Researchers in automated reasoning spend a significant portion of their work time specifying calculi and proving metatheorems about them. These proofs are mostly carried out with pen and paper, which is error-prone and can be tedious. Especially when working on variants of previously devised calculi, it is easy to miss subtle but important differences. We are part 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 [16]. 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) [4], 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 $\mathrm{DPLL}_{\mathrm{BB}}$ calculus [10], 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 component typically also generates some (often lower) cost.

Our contributions are firstly a rigorous proof of the correctness 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 [6,17]. Formalization and verifications in proof assistants are often justified by the idea that they can be reused and extended. This is exactly what we are doing here. In the formalization, we eventually 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 6000 lines for the formalization of CDCL [4], and 4500 lines for Nieuwenhuis *et al.*'s account for CDCL [14]. 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 $1.5\,\mathrm{person}-\mathrm{month}$ of work and was easy to do. The extension to partial valuations amounts to 1300 lines. We further demonstrate the potential of reuse, by applying our framework 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 formalization of CDCL (Section 2), the optimizing OCDCL calculus is introduced (Section 3). It is described as an abstract non-deterministic transition system and has the conflict analysis for the first unique implication point built-into its CDCL part.

We develop an abstract calculus implementing branch and bound, CDCL_{BnB}, that we instantiate with weights (Section 7). In order to overcome the limitation of our calculus 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 notation "Isa:fact ☑" from the formalization with a link to the documentation of our formalization.

2 Formalization of CDCL in Isabelle

Isabelle Isabelle [15, 16] is a generic proof assistant that supports many object logics. The metalogic is an intuitionistic fragment of higher-order logic (HOL) [7]. The types are built from type variables $'a, 'b, \ldots$ and n-ary type constructors, normally written in postfix notation (e.g, $'a$ *list*). The infix type constructor $'a \Rightarrow 'b$ is interpreted as the (total) function space from $'a$ to $'b$. Function applications are written in a curried style (e.g., f x y). Anonymous functions $x \mapsto y_x$ are written $\lambda x.\ y_x$. The judgment $t :: \tau$ indicates that term t has type τ.

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

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

$$\texttt{locale}\ \mathrm{X} = \texttt{fixes}\ \mathrm{c} :: \tau_{'a}\ \texttt{assumes}\ A_{'a,\mathrm{c}}$$

The definition of locale X implicitly fixes a type $'a$, explicitly fixes a constant c whose type $\tau_{'a}$ may depend on $'a$, and states an assumption $A_{'a,\mathrm{c}}$ 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 $A_{'a,\mathrm{c}}$. 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 $A_{'a,\mathrm{c}}$.

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 $A_{\upsilon,t}$. After the proof, all the lemmas proved in X become available, with $'a$ and c :: $\tau_{'a}$ instantiated with υ and $t :: \tau_{\upsilon}$.

CDCL We have previously developed a framework to formalize the conflict-driven-clause-learning procedure [5] in Isabelle. Clauses are defined as (finite) multisets. For readability, we will write $\bot$ and $A \vee B$ instead of their Isabelle counterparts $\{\#\}$ and $A + B$. Given a literal L and a set of literals I, we define entailment by $I \vDash L$ iff $L \in I$. We can lift it to clauses (multiset of literals) by $I \vDash C$ iff there is a literal $L \in C$ such that $I \vDash L$. We can also lift it to clause sets $I \vDash N$ iff $\forall C \in N.\ I \vDash C$.

The conflict-driven clause learning is a procedure that builds a candidate model, called the *trail* or M. It contains literals that have either been decided ($L^\dagger$) or propagated (L^C 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 $\top$. For example, the Decide rule extends M by an arbitrary choice L:

Decide $\quad (M; N; U; \top) \implies (ML^\dagger; N; U; \top)$

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:

`inductive` decide :: $'st \Rightarrow 'st \Rightarrow bool$ `where`
undefined_lit (trail S) $L \Longrightarrow L \in$ atom (clauses S) $\Longrightarrow$
$S' \sim$ append_trail ($L^\dagger$) $S \Longrightarrow$
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. Since states are 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 cost on the set of all literals $Lit(\Sigma)$ over Σ into the positive rationals, cost : $Lit(\Sigma) \to \mathbb{Q}^+$, where our results do not depend specifically on the positive rationals (including 0), e.g., they also hold for the naturals or positive reals. The cost function can be extended to a pair of a literal and a partial valuation $\mathcal{A}$ by $\operatorname{cost}(L, \mathcal{A}) := \operatorname{cost}(L)$ if $\mathcal{A} \models L$ and $\operatorname{cost}(L, \mathcal{A}) := 0$ if L is not defined. The function can be extended to (partial) valuations by $\operatorname{cost}(\mathcal{A}) = \sum_{L \in Lit(\Sigma)} \operatorname{cost}(\mathcal{A}, L)$. We identify partial valuations with consistent sequences $M = [L_1 \dots L_n]$ of literals. Trails are always be consistent. A valuation I is total over clauses N when all atoms of N are defined in I.

The optimizing conflict-driven clause learning calculus (OCDCL) 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 $\top$. We extend the cost function to $\top$ by defining $\operatorname{cost}(\top) = \infty$ (i.e., $\top$ is the worst possible outcome). OCDCL 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 $(\epsilon; N; \emptyset; \top; \top)$. 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 OCDCL calculus always terminates in deriving the empty clause $\perp$. If in this case $O = \top$, 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 $(M; N; U; \top; O) \implies_{\text{OCDCL}} (M; N; U; \top; M)$

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

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

provided $O \neq \top$ and $\operatorname{cost}(M) \geq \operatorname{cost}(O)$.

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

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

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 $(M; N; U; \top; O) \implies_{\text{OCDCL}} (ML^{C \vee L}; N; U; \top; O)$

provided $C \vee L \in N \cup U$, $M \models \neg C$, L is undefined in M.

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

provided L is undefined in M, contained in N.

ConflSat $(M; N; U; \top; O) \Longrightarrow_{\text{OCDCL}} (M; N; U; D; O)$

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 $(ML^{C\vee L}; N; U; D; O) \Longrightarrow_{\text{OCDCL}} (M; N; U; D; O)$

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

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

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

Backtrack $(M_1K^{\dagger}M_2; N; U; D \vee L; O) \Longrightarrow_{\text{OCDCL}} (M_1L^{D\vee L}; N; U \cup \{D \vee L\}; \top; O)$

provided L is of level k and D and K are of level $i < k$.

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

Lemma 2 (OCDCL Termination, Isa:wf_ocdcl_W ☑**).** *OCDCL with a reasonable strategy terminates in a state* $(M; N; U; \bot; O)$.

Proof. If the derivation started from $(\epsilon, N, \emptyset, \top, top)$, the following function is a measure for OCDCL:

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

It is decreasing for the lexicographic order. The hardest part of the proof is the decrease when Backjump: $3^n - 1 - |U \cup \{D \vee L\}| < 3^n - 1 - |U|$ is decreasing since no clause is relearned. The proof is similar to the one for CDCL. $\square$

Theorem 3 (Correctness of OCDCL, Isa:full_ocdcl$_w$_stgy_no_conflicting_-clause_from_init_state ☑**).** *OCDCL with a reasonable strategy starting from state* $(\epsilon; N; \emptyset; 0; \top; \epsilon)$ *terminates in a state* $(M; N; U; 0; \bot; O)$*. If* $O = \epsilon$ *then* N *is unsatisfiable. If* $O \neq \epsilon$ *then* $O \models N$ *and for any other total model* M' *with* $M' \models N$ *it holds* $\text{cost}(M') \geq \text{cost}(O)$*.*

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

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

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 $\text{cost}(M) = \text{cost}(MM'')$.

Lemma 4 (Improve$^+$, Isa:wf_ocdcl$_W$_p ☑ **and Isa:**full_ocdcl$_W$_p_stgy_no_conflicting_clause_from_init_state ☑**).**

The rule Improve can be replaced by rule Improve$^+$: All previously established OCDCL properties are preserved.

The rules ConflOpt can produce very long conflict clauses. 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

If we ignore the Improve rule, the remaining OCDCL transitions are very similar to a generalized CDCL where all conflicting clauses that can be used by the rules ConflOpt and Prune are included in the set of clauses. These are the clauses that are entailed by the clauses of weight larger than the optimal model found so far, i.e. the clauses D such that $\{\neg C \mid \text{cost}\, C \geq \text{cost}\, O\} \vDash D$. Hence, we can see OCDCL transitions as CDCL transitions and reuse proof on the latter. The rule Improve of OCDCL simply adds new clauses to this set.

In the formalization, we abstract over this clause set by using instead a set $\mathcal{T}_N(O)$ and use a predicate is_improving $M\ M'\ O$ to indicate that Improve can be applied. This yields a more abstract branch-and-bound calculus CDCL_{BnB} (Section 4.1). CDCL_{BnB} is seen as a special case of CDCL, where the additional clauses $\mathcal{T}_N(O)$ are part of the initial set of clauses. This reduces many proofs to reusing their CDCL counterpart: ConflOpt becomes equivalent to ConflSAT, since it picks a clause of $\mathcal{T}_N(O)$. We do not specify the type of O (Section 4.2).

This reduces the burden to develop the new variant and makes it possible to reuse many proofs and especially all the invariants about the states: We neither have to redefine nor reprove most of them.

We instantiate CDCL_{BnB} to get a generalized version OCDCL_g: The set of clauses $\mathcal{T}_N(O)$ is instantiated by $\{D \mid \{\neg C.\, \text{cost}\, C \geq \text{cost}\, O\} \vDash D\}$ (Section 4.3). Finally, we specialize OCDCL_g to get OCDCL from Section 3 (Section 4.4).

4.1 Branch-and-Bound Calculus, CDCL_{BnB}

We use a similar approach to our CDCL formalization with an abstract state and selectors, except that we add an additional component representing information on the optimizing branch-and-bound part of the calculus. We do not yet specify the type of this additional component. We parametrize the calculus by a set of clauses $\mathcal{T}_N$ 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. $\mathcal{T}$ represents all the clauses that are entailed. We require that:

- the atoms of $\mathcal{T}_N\,(O)$ are included in the atoms of N. We do not introduce new variables.
- the clauses of $\mathcal{T}_N\,(O)$ do not contain duplicate literals. Duplicates are incompatible with the conflict analysis.
- if is_improving $M\ M'\ O$, then $\mathcal{T}_N\,(O) \subseteq \mathcal{T}_N\,(M')$.
- if is_improving $M\ M'\ O$, then $\neg M \in \mathcal{T}_N\,(M')$.

The rules $\text{ConflOpt}_{\text{BnB}}$, $\text{Improve}_{\text{BnB}}$, and $\text{Backtrack}_{\text{BnB}}$ are defined as follows:

$\text{ConflOpt}_{\text{BnB}}$ $(M; N; U; k; \top; O) \Longrightarrow_{\text{OCDCL}} (M; N; U; k; \neg M; O)$

provided $\neg M \in \mathcal{T}_N\,(O)$

$\text{Improve}_{\text{BnB}}{}^{+}$ $(M; N; U; k; \top; O) \Longrightarrow_{\text{OCDCL}} (M; N; U; k; \neg M; M')$

provided is_improving $M\ M'\ O$ holds

$\text{Backtrack}_{\text{BnB}}$ $(M_1 K^{\dagger} M_2; N; U; D \vee L; O) \Longrightarrow_{\text{OCDCL}} (M_1 L^{D' \vee L}; N; U \cup \{D' \vee L\}; \top; O)$

provided L is of maximum level, $D' \subseteq D$, $N + U + \mathcal{T}_N\,(O) \vDash D' \vee L$, D' and K are of same level i, and i strictly less than the maximum level

We can simply embed into our CDCL formalization the states with the weights and reuse the previous definitions, properties, and invariants 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. At this level, anything can be stored in O.

Compared with the rule from Section 3, we make it possible to express conflict-clause minimization: Instead of $D \vee L$, a clause $D' \vee L$ is learned such that $D' \subseteq D$ and $N + U + \mathcal{T}_N\,(O) \vDash D' \vee L$. 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 $N + U \vDash D' \vee L$ would be used. The latter version is sufficient to express conflict minimization as implemented in most SAT solvers [20], but the former is stronger and makes it possible for example to remove decision literals without cost from D.

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

4.2 Embedding into CDCL

In order to reuse the proof 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 $(M; N \cup \mathcal{T}_N\,(O); U; D)$.

In Isabelle, the most direct solution would be to instantiate the CDCL calculus with a selector returning $N + \mathcal{T}_N\,(O)$ 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 $(M; N + \mathcal{T}_N\,(O); U; D)$ and consider CDCL on tuples of the latter.

Except for the Improve rule, every OCDCL rule can be mapped to a CDCL rule: The $\text{ConflictOpt}_{\text{BnB}}$ rule corresponds to the Conflict rule (because it can pick a clause from $\mathcal{T}_N\,(O)$) and the extended Backtrack rule is mapped to CDCL'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 invariants.

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

Not all transitions of CDCL can be taken by OCDCL: Propagating of clauses in $\mathcal{T}_N\,(O)$ is not possible. The structural properties are sufficient to prove that OCDCL is terminating as long as Improve$^+$ can be applied only finitely often, because the CDCL calculus is terminating. At this level, Improve$^+$ is too abstract to prove that it terminates. With the additional assumptions that Improve 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.

4.3 Instantiation with weights, OCDCL_g

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

$\texttt{locale}$ cost $=$
  $\texttt{fixes}$ cost :: *'v literal multiset* $\Rightarrow$ *'c*
  $\texttt{assumes}$ $\forall C.$ consistent_interp $B \wedge$ distinct_mset $B \wedge A \subseteq B \longrightarrow$
    cost $A \leq$ cost B

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

$$\begin{aligned}
\mathcal{T}_N(O) = \{C.\ & \mathrm{atom}(C) \subseteq \mathrm{atom}(N) \\
& \wedge C \text{ is not a tautology nor contains duplicates} \\
& \wedge \{-D.\ \mathrm{cost}\,(D) \geq \mathrm{cost}\,(O)\} \vDash C\}\} \\
\text{is_improving } M\ M'\ O \leftrightarrow\ & M' \text{ is a total extension of } M,\ M \vDash N, \\
& \text{any total extensions of } M \text{ has the same cost, and} \\
& \mathrm{cost}\, M < \mathrm{cost}\, O
\end{aligned}$$

and then discharge the assumptions over it.

OCDCL_g inherit from the invariants from $\mathrm{CDCL_{BnB}}$. For termination, we only have to prove that $\mathrm{Improve}^+$ terminates to reuse the proof we already made on $\mathrm{CDCL_{BnB}}$. The key property of OCDCL_g is the following:

Lemma 5 (Isa:entails_too_heavy_clauses_too_heavy_clauses ☑) *If I is a total consistent model of N, then either* $\mathrm{cost}\,(I) \geq \mathrm{cost}\,(O)$ *or I is a total model of $\mathcal{T}_N\,(O)$.*

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

By transitivity, since I is total, I is also a model of $\mathcal{T}_N\,(O)$. □

This is the proof that breaks if partial models are allowed. 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 know about O.

4.4 OCDCL

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

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

5 Optimal Partial Valuations

A partial Σ-valuation is a partial mapping $\mathcal{A}\colon \Sigma \nrightarrow \{0,1\}$ from the set of propositional variables Σ into $\{0,1\}$. For any atom $P \in \Sigma$, we write $\mathcal{A}(P)\!\downarrow$ if $\mathcal{A}$ is defined on P. If $\mathcal{A}(P)\!\downarrow$ and $\mathcal{A}(P) = 1$, we write $\mathcal{A} \models P$. The valuation $\mathcal{A}$ can be extended to literals, clauses, and clause sets as follows: $\mathcal{A}(\neg P) := 1 - \mathcal{A}(P)$ if $\mathcal{A}(P)\!\downarrow$ and unset otherwise. $\mathcal{A}(L_1 \vee \cdots \vee L_n) := 1$ if there is some L_i with $\mathcal{A} \vDash \mathcal{A}(L_i)$. $\mathcal{A}(C_1 \wedge \cdots \wedge C_n) := 1$ 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 $\mathcal{A} \models L$, $\mathcal{A} \models L_1 \vee \cdots \vee L_n$, and $\mathcal{A} \models C_1 \wedge \cdots \wedge C_n$, respectively. As usual, we identify clause sets and conjunctions of clauses.

To reduce the search from optimal partial valuations to optimal total valuations, we use the dual rail encoding [6, 17]. For every proposition variable P, we create two variables P^1 and P^0 indicating that P is defined positively or negatively. We also add the clause $\neg P^1 \vee \neg P^0$ to ensure that P is not defined positively and negatively at the same time. The resulting set is called $\mathrm{penc}\,(N)$.

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

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

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

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

Lemma 6 (Partial and Total Valuations Coincide Modulo penc**, Isa:**penc_ent_postp ☑ **and Isa:**penc_ent_upostp ☑**).** *Let N be a clause set.*

1. *If $\mathcal{A} \models N$ for a partial model $\mathcal{A}$ then* $\mathrm{pdec}^-\,(\mathcal{A}) \models \mathrm{penc}(N)$*;*

2. If $\mathcal{A}' \models \mathrm{penc}(N)$ *for a total model* $\mathcal{A}'$*, then* $\mathrm{pdec}\,(\mathcal{A}') \models N$*.*

Lemma 7 (Encoding penc **Preserves Cost Optimal Models, Isa:**full_encoding_OCDCL_correctness ☑**).** *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* $\mathrm{penc}(N)$ *over* cost'*, resulting in* $\mathrm{cost}'(\mathcal{A}') = m$*, then the partial model* $\mathrm{pdec}\,(\mathcal{A}')$ *is cost-optimal for* N *and* $\mathrm{cost}(\mathrm{pdec}\,(\mathcal{A}')) = m$*.*

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

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

Non-Machine-Checked Lemma 8 (OCDCL on penc**)** *Consider a reasonable CDCL run on* $\mathrm{penc}(N)$*. If rule Decide is restricted to deciding either* P^1 *or* P^0 *for any propositional variable, and ConflOpt only considers the decision literals out of* M *as a conflict clause, then OCDCL performs at most* 3^n *Backtrack steps.*

Proof. 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 $\neg P^0$. In summary, for each propositional variable, a run considers at most 3 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) [18], but we use partial models for CDCL, similar to a trail by adding a predicate to indicate whether a model is total. We distinguish between literals that can have a weight $\Delta\Sigma$ from the others ($\Sigma \setminus \Delta\Sigma$) that can be left unchanged by the encoding.

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

`interpretation` OCDCL `where` cost $= \mathrm{cost}'$

We have to prove the proof obligation that cost' is monotone.

Finally, we can prove the correctness Theorem 7. 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 8. 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: $(M_1 K^\dagger M_2, N, U, \neg(M_1 K^\dagger M_2)) \Longrightarrow^\star_{\mathrm{Resolve}} (M_1 K^\dagger, N, U, \neg(M_1 K^\dagger)) \Longrightarrow^\star_{\mathrm{Backtrack}} (M_1 \neg K^{D'}, N, U \cup \{D'\}, \top)$, where D' is the negation of the decisions of $M_1 K^\dagger$ 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 how OCDCL can be used to solve MAX-SAT and can be extended to solve minimal model coverage. Both extensions are verified using Isabelle.

Maximum satisfiability (MAX-SAT) is a well-known optimization problem [11]. It consists of two clause sets N_H (hard constraints, mandatory to satisfy) and N_S (soft constraints, optional to satisfy). The set N_S comes with a cost function for clauses that are not satisfied. The aim is to find a total model with minimal cost.

Theorem 9 (Isa:partial_max_sat_is_weight_sat ☑**).** *Let* $(N_H, N_S, \mathrm{cost})$ *be an instance of a MAX-SAT problem and let* $active : N_S \to \Sigma'$ *be an injective and surjective mapping for a set* Σ' *of fresh propositional variables that assigns to each soft constraint an activation variable.*

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

If there is no model I *of* N*, the NAX-SAT problem has no solution. Otherwise,* I *without the additional atoms from* Σ' *is an optimal solution to the MAX-SAT problem.*

Proof.
- N_H is satisfiable iff MAX-SAT has a solution. Therefore, if there is no model I of N, then N_H is unsatisfiable.
- Let $I' = \{L \mid L \in I \wedge \mathrm{atom}(L) \notin \Sigma'\}$. Let J be any other model of $(N_H \cup N_S)$ and J' its total extension to Σ': $J' = J \cup \{\mathrm{active}(C) \mid C \in N_S \wedge J \not\models C\} \cup \{\neg\mathrm{active}(C) \mid C \in N_S \wedge J \models C\}$ to N.
  J' satisfies N_H and is a total consistent model of N. Therefore, $\mathrm{cost}'(J') \geq \mathrm{cost}(I)$, because I is the optimal model of N. By definition, $\mathrm{cost}'(I) = \mathrm{cost}(I')$ and $\mathrm{cost}'(J) = \mathrm{cost}(J')$. Therefore, I is an optimal MAX-SAT model.

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 to propositional logic: given a set N of clauses we search for a set of models $\mathcal{M}$ such that for each propositional variable P occurring in N, $M \models P$ for at least one $M \in \mathcal{M}$ 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 more true propositional variables. 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 $\mathcal{M}$, then it is not required in a minimal solution. The model covering can be computed by creating another CDCL extension, where the set $\mathcal{M}$ is explicitly added as a component to a state and used for a branch-and-bound optimization approach, similar to OCDCL. The extension to CDCL are the two additional rules:

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

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

Add $\quad (M; N; U; k; \top; \mathcal{M}) \implies_{\text{CDCLcm}} (M; N; U; k; \top; \mathcal{M} \cup \{M\})$

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

The CDCLcm calculus does not necessarily compute a minimal set of covering models. Minimization can then be done in a second step. This calculus is another instance of CDCL_{BnB}. In the formalization, we instantiate CDCL_{BnB} with:

$$\begin{aligned}
\mathcal{T}_N(\mathcal{M}) = \{C.\ & \text{atom}(C) \subseteq \text{atom}(N) \\
& \wedge C \text{ is not a tautology nor contains duplicates} \\
& \wedge \{-D.\ \text{is_dominating } \mathcal{M}\ D,\ \text{total}\} \cup N \vDash C\}
\end{aligned}$$

$$\begin{aligned}
\text{is_improving } M\ M'\ \mathcal{M} \leftrightarrow\ & M = M' \text{ and } M \vDash N \\
& \text{and } M \text{ is not dominated by } \mathcal{M} \\
& \text{and } M \text{ is consistent, total, duplicate free}
\end{aligned}$$

Compared with OCDCL, $\mathcal{T}_N(\emptyset)$ is never empty, because it contains at least N.

Theorem 10 (CDCLcm Correctness, Isa:cdclcm_correctness ☑**).** *If there is no clause of N that contains duplicated literals and $(\epsilon, N, \emptyset, \top, \emptyset) \implies^{!}_{CDCLcm} (\epsilon, N, U, \bot, \mathcal{M})$, then for every variable P in N, there is a model M of N, $M \in \mathcal{M}$, where $M \vDash P$, or there is no model satisfying both P and N.*

The proof involves a lemma similar to Lemma 5: Every model is dominated by a model in $\mathcal{M}$ or is still a model of $\mathcal{T}_N(\mathcal{M})$.

8 Related Work and Conclusion

There are several formalizations of CDCL beyond ours, as we discuss in our previous article [5, Section 6], but we are not aware of any formalization of an optimizing CDCL 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 [10]. 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: $\neg M \vee c \geq \text{cost}(O)$ is entailed by the clauses.

A related problem to finding the minimum partial model is called minimum-weight propositional satisfiability by Sebastiani *et al.* [19]. 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 [12]. Each time a variable is decided, it is first set to true, then set to false. Moreover, if the current model is larger than a given bound, then the search stops exploring the current branch. When a new better model is found, the search is restarted with the new lower bound. A version lifted to CDCL has been implemented in zChaff [8] to solve MAX-SAT. 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 we have previously developed and the abstraction we have used 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. 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 CDCLcm, Section 7. Minimization of the model covering set computed by CDCLcm can also be solved by an afterwards application of a CDCL calculus with branch-and-bound [13], 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 [21], so proving correctness of CDCL(LA) does not need a from-scratch new effort.

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

References

1. Becker, H., Bentkamp, A., Blanchette, J.C., Fleury, M., From, A.H., Jensen, A.B., Lammich, P., Larsen, J.B., Michaelis, J., Nipkow, T., Peltier, N., Popescu, A., Robillard, S., Schlichtkrull, A., Tourret, S., Traytel, D., Villadsen, J., Petar, V.: IsaFoL: Isabelle Formalization of Logic, `https://bitbucket.org/isafol/isafol/`
2. Biere, A.: Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016. In: Balyo, T., Heule, M., Järvisalo, M. (eds.) Proc. of SAT Competition 2016—Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2016-1, pp. 44–45. University of Helsinki (2016)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
4. Blanchette, J.C., Fleury, M., Lammich, P., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. J. Autom. Reasoning **61**(1-4), 333–365 (2018)
5. Blanchette, J.C., Fleury, M., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. In: Olivetti, N., Tiwari, A. (eds.) Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings. LNCS, vol. 9706, pp. 25–44. Springer (2016)
6. Bryant, R.E., Beatty, D.L., Brace, K.S., Cho, K., Sheffler, T.J.: COSMOS: A compiled simulator for MOS circuits. In: DAC. pp. 9–16. IEEE Computer Society Press / ACM (1987)
7. Church, A.: A formulation of the simple theory of types. J. Symb. Log. **5**(2), 56–68 (1940)
8. Giunchiglia, E., Maratea, M.: Solving optimization problems with DLL. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 – September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 377–381. IOS Press (2006)
9. Gordon, M.J.C., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, LNCS, vol. 78. Springer (1979)
10. Larrosa, J., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A framework for certified boolean branch-and-bound optimization. J. Autom. Reasoning **46**(1), 81–102 (2011)
11. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 613–631. IOS Press (2009)
12. Liberatore, P.: Algorithms and experiments on finding minimal models. Tech. Rep. 09-99, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza" (1999)
13. Manquinho, V.M., Silva, J.P.M.: Satisfiability-based algorithms for pseudo-boolean optimization using gomory cuts and search restarts. In: ICTAI. pp. 150–155. IEEE Computer Society (2005)
14. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006)
15. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)

16. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
17. Palopoli, L., Pirri, F., Pizzuti, C.: Algorithms for selective enumeration of prime implicants. Artif. Intell. **111**(1-2), 41–72 (1999)
18. Schlichtkrull, A.: Formalization of the resolution calculus for first-order logic. In: ITP. LNCS, vol. 9807, pp. 341–357. Springer (2016)
19. Sebastiani, R., Giorgini, P., Mylopoulos, J.: Simple and minimum-cost satisfiability for goal models. In: Persson, A., Stirna, J. (eds.) Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings. LNCS, vol. 3084, pp. 20–35. Springer (2004)
20. Sörensson, N., Biere, A.: Minimizing learned clauses. In: SAT. LNCS, vol. 5584, pp. 237–243. Springer (2009)
21. Thiemann, R.: Extending a verified simplex algorithm. In: Barthe, G., Korovin, K., Schulz, S., Suda, M., Sutcliffe, G., Veanes, M. (eds.) LPAR-22 Workshop and Short Paper Proceedings. Kalpa Publications in Computing, vol. 9, pp. 37–48. EasyChair (2018)