A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality

Jasmin Christian Blanchette
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
j.c.blanchette@vu.nl

Mathias Fleury and **Christoph Weidenbach**
Max-Planck-Institut für Informatik
Saarland Informatics Campus
Saarbrücken, Germany
{mfleury,weidenbach}@mpii.de

Abstract

We developed a formal framework for SAT solving using the Isabelle/HOL proof assistant. Through a chain of refinements, an abstract CDCL (conflict-driven clause learning) calculus is connected to a SAT solver that always terminates with correct answers. The framework offers a convenient way to prove theorems about the SAT solver and experiment with variants of the calculus. Compared with earlier verifications, the main novelties are the inclusion of the CDCL rules for forget, restart, and incremental solving and the use of refinement.

1 Introduction

Researchers in automated reasoning spend a significant portion of their work time specifying logical calculi and proving metatheorems about them. These proofs are typically carried out with pen and paper, which is error-prone and can be tedious. As proof assistants are becoming easier to use, it makes sense to employ them.

In this spirit, we started an effort, called IsaFoL (Isabelle Formalization of Logic), that aims at developing libraries and methodology for formalizing modern research in the field, using the Isabelle/HOL proof assistant.[1] Our initial emphasis is on established results about propositional and first-order logic. In particular, we are formalizing large parts of Weidenbach's forthcoming textbook, tentatively called *Automated Reasoning—The Art of Generic Problem Solving*.

The objective of formalization work is not to eliminate paper proofs, but to complement them with rich formal companions. Formalizations help catch mistakes, whether superficial or deep, in specifications and theorems; they make it easy to experiment with changes or variants of concepts; and they help clarify concepts left vague on paper.

This paper presents our formalization of the CDCL (conflict-driven clause learning) calculus [Bayardo Jr. and Schrag, 1996; Marques-Silva and Sakallah, 1996] as described in *Automated Reasoning*. CDCL is a core algorithm implemented in modern SAT solvers. We start with a family of abstract transition systems, following Nieuwenhuis, Oliveras, and Tinelli's [2006] account of CDCL (Section 3). Some of the calculi include rules for learning and forgetting clauses and for restarting the search. All calculi are proved sound and complete, as well as conditionally terminating.

The abstract CDCL calculus is refined into the more concrete calculus presented in *Automated Reasoning* and recently published [Weidenbach, 2015] (Section 4). The latter specifies a criterion for learning clauses representing first unique implication points [Biere *et al.*, 2009], with the guarantee that learned clauses are derived at most once. The calculus also supports incremental solving. This concrete calculus is then refined further, as a certified functional program (Section 5).

Any formalization effort is a case study in the use of a proof assistant. We relied on the following features of Isabelle:

- *Isar* [Wenzel, 2007] is a textual input format that makes it possible to write structured, readable proofs—a requisite for any formalization that aims at clarifying an abstract pen-and-paper proof.
- *Locales* [Ballarin, 2014] parameterize theories over operations and assumptions, encouraging a modular style of development. They are useful to express hierarchies of related concepts and to reduce the number of parameters and assumptions that must be passed around.
- *Sledgehammer* [Blanchette *et al.*, 2013] integrates automatic theorem provers in Isabelle to discharge proof obligations. Many of the provers build on a SAT solver, resulting in a situation where SAT solvers are employed to prove their own metatheory.

Our work is related to other verifications of SAT solvers, typically with the aim of increasing their trustworthiness [Marić, 2010; Lescuyer, 2011; Shankar and Vaucher, 2011; Oe *et al.*, 2012]. This goal has lost some of its significance with the emergence of formats for certificates that are easy to generate, even in highly optimized solvers, and that can be processed efficiently by verified checkers [Cruz-Filipe *et al.*, 2017; Lammich, 2017]. In contrast, our focus is on formalizing the metatheory of CDCL, to study and connect the various members of the family. The main novelties of our framework are the inclusion of rules for forget, restart, and incremental solving and the application of stepwise refinement to transfer results. The original version of this paper was presented at the eighth edition of the International Joint Conference on Automated Reasoning (IJCAR 2016) in Coimbra, Portugal [Blanchette *et al.*, 2016].

[1] `https://bitbucket.org/isafol/isafol`

2 Isabelle

Isabelle [Nipkow *et al.*, 2002; Nipkow and Klein, 2014] is a generic proof assistant that supports many object logics. The metalogic is an intuitionistic fragment of higher-order logic (HOL). 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 *bool* and the familiar logical symbols. Its syntax is similar to that of typed functional programming languages such as Haskell, OCaml, and Standard ML.

Isabelle adheres to the tradition initiated in the 1970s by the LCF system [Gordon *et al.*, 1979]: 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 (as in Prolog) and recursive functions (as in Haskell). Internally, the system synthesizes suitable low-level definitions.

Isabelle developments are organized as collections of text files that build on one another. Each file consists of definitions, lemmas, and proofs expressed in Isar, Isabelle's input language. Proofs are specified in a declarative format.

Isabelle locales are a convenient mechanism for structuring large proofs. A locale fixes types, constants, functions, and assumptions within a specified scope, which are instantiated when using the locale. Locales support inheritance, union, and embedding. For example, the semigroup locale is parameterized over an operator $*$ that is assumed to be associative. The group locale inherits from semigroup and adds a 0 constant and a $^{-1}$ operator, with additional assumptions about them. All lemmas and definitions provided in the context of semigroup are available in group as well.

3 Abstract CDCL

The abstract CDCL calculus by Nieuwenhuis *et al.* [2006] is the starting point of our refinement chain. Properties such as correctness and termination are inherited by subsequent links.

We represent raw and annotated literals by freely generated datatypes parameterized by the types $'v$ (propositional variable, or atom) and $'c$ (clause):

```
datatype 'v lit =        datatype ('v, 'c) ann_lit =
  Pos 'v                   Decided ('v lit)
| Neg 'v                 | Propagated ('v lit) 'c
```

Informally, we write A, $\neg A$, and $L^\dagger$ for positive, negative, and decided literals, and $-L$ for the negation of a literal, with $-(\neg A) = A$. The simpler calculi do not use $'c$.

A $'v$ *clause* is a (finite) multiset over $'v$ *lit*. To ease reading, we write clauses using logical symbols. Given a set I of literals, $I \vDash C$ is true if and only if C and I share a literal. This is lifted to (multi)sets of clauses: $I \vDash N \leftrightarrow \forall C \in N.\ I \vDash C$. A set is satisfiable if there exists a consistent set of literals I such that $I \vDash N$. Finally, $N \vDash N' \leftrightarrow \forall I.\ I \vDash N \longrightarrow I \vDash N'$.

3.1 DPLL with Backjumping

Nieuwenhuis *et al.* present CDCL as a set of transition rules on states. A state is a pair (M, N), where M is the *trail* and N is the set of clauses to satisfy. The trail is a list of annotated literals that represents the partial model under construction. As such, it never contains both a literal L and its negation $-L$ at the same time. The trail grows on the left: Adding a literal L to M results in the new trail LM. We build lists from elements and other lists by simple juxtaposition.

The core of the CDCL calculus is defined as a transition relation DPLL+BJ, a generalization of DPLL (Davis–Putnam–Logemann–Loveland) [Davis *et al.*, 1962] with nonchronological backtracking, or *backjumping*. We write $S \Longrightarrow_{\mathsf{DPLL+BJ}} S'$ for DPLL+BJ S S'. The DPLL+BJ calculus consists of three rules, starting from an initial state (ϵ, N):

Propagate $(M, N) \Longrightarrow_{\mathsf{DPLL+BJ}} (LM, N)$
if N contains a clause $C \vee L$ such that $M \vDash \neg C$ and L is undefined in M (i.e., neither $M \vDash L$ nor $M \vDash -L$)

Backjump $(M'L^\dagger M, N) \Longrightarrow_{\mathsf{DPLL+BJ}} (L'M, N)$
if N contains a conflicting clause C (i.e., $M'L^\dagger M \vDash \neg C$) and there exist C', L' such that $N \vDash C' \vee L'$, $M \vDash \neg C'$, and L' is undefined in M but occurs in N or in $M'L^\dagger$

Decide $(M, N) \Longrightarrow_{\mathsf{DPLL+BJ}} (L^\dagger M, N)$
if the atom of L belongs to N and is undefined in M

The Backjump rule encompasses both DPLL, where $C' \vee L'$ is instantiated to $\neg M \vee -L$, and CDCL backjumping, where $C' \vee L'$ is a new clause derived from N.

Theorem 1 (Termination)**.** DPLL+BJ *is well founded.*

When formalizing the termination proof, we found some inaccuracies in the proof by Nieuwenhuis *et al.* [Blanchette *et al.*, 2016, Section 3.2]. With this exception, we found their proofs clear and easy to follow.

A *final* state is a state from which no transitions are possible. Given a relation $\Longrightarrow$, we write $\Longrightarrow^!$ for the right-restriction of its reflexive transitive closure to final states.

Theorem 2 (Correctness)**.** *If* $(\epsilon, N) \Longrightarrow^!_{\mathsf{DPLL+BJ}} (M, N)$*, then* N *is satisfiable if and only if* $M \vDash N$*.*

3.2 The CDCL Calculus

The abstract CDCL calculus extends DPLL+BJ with a pair of rules for learning new lemmas and forgetting old ones:

Learn $(M, N) \Longrightarrow_{\mathsf{CDCL\_NOT}} (M, N \uplus \{C\})$
if $N \vDash C$ and each atom of C is in N or M

Forget $(M, N \uplus \{C\}) \Longrightarrow_{\mathsf{CDCL\_NOT}} (M, N)$ if $N \vDash C$

The Learn rule is normally applied to clauses built exclusively from atoms in M, because the learned clause is false in M. This property eventually guarantees that the learned clause is not redundant. We call this calculus CDCL_NOT after Nieuwenhuis, Oliveras, and Tinelli. In general, CDCL_NOT does not terminate, because it is possible to learn and forget the same clause infinitely often. But for some parameter instantiations with suitable restrictions on Learn and Forget, the calculus always terminates. In particular, DPLL+BJ always terminates.

Theorem 3 (Termination)**.** *Let* C *be an instance of the* CDCL_NOT *calculus (i.e.,* C $\subseteq$ CDCL_NOT*). If* C *admits no infinite chains consisting exclusively of* Learn *and* Forget *transitions, then* C *is well founded.*

In many SAT solvers, the only clauses that are ever learned are the ones used for backtracking. If we restrict the learning so that it is always done immediately before backjumping, some progress will be made between a Learn and the next Learn or Forget. This idea is captured by the combined rule

Learn+Backjump
$(M'L^{\dagger}M, N) \Longrightarrow_{\mathsf{CDCL\_NOT\_merge}} (L'M, N \uplus \{C' \vee L'\})$
if C', $L^{\dagger}$, L', M, M', N satisfy Backjump's side conditions

The calculus variant that performs this rule instead of Learn or Backjump is called CDCL_NOT_merge.

3.3 Restarts

Modern SAT solvers periodically restart the proof search to apply the effects of a changed dynamic decision literal heuristic. Upon a restart, some learned clauses may be removed, and the trail is reset to ϵ. Since our calculus has a Forget rule, Restart needs only to clear the trail. Adding Restart to CDCL_NOT yields CDCL_NOT+restart. However, this calculus does not terminate, because Restart can always be applied.

A working strategy is to gradually increase the number of transitions between successive restarts. This is formalized via a locale parameterized by a base calculus C and an unbounded function $f : \mathbb{N} \to \mathbb{N}$. The extended calculus C+restartT is defined by the two rules

Restart $(S, n) \Longrightarrow_{C\mathsf{+restartT}} ((\epsilon, N'), n+1)$
if $S \Longrightarrow_{C}^{m} (M', N')$ and $m \geq f\, n$

Finish $(S, n) \Longrightarrow_{C\mathsf{+restartT}} (S',\ n+1)$ if $S \Longrightarrow_{C}^{!} S'$

In practice, the Luby sequence $(1,1,2,1,1,2,4,\ldots)$ [Luby *et al.*, 1993] is often used for f. We also formalized an alternative strategy based on the number of conflicts or learned clauses.

4 A Refined CDCL towards an Implementation

The CDCL_NOT calculus captures the essence of modern SAT solvers without imposing a policy on when to apply specific rules. In particular, the Backjump rule depends on a clause $C' \vee L'$ to justify the propagation of a literal, but does not specify a procedure for coming up with this clause. Weidenbach [2015] developed a calculus that is more specific in this respect, and closer to existing implementations, while keeping many aspects unspecified. This calculus, CDCL_W, is also formalized in Isabelle and connected to CDCL_NOT.

4.1 The New CDCL Calculus

The CDCL_W calculus operates on states (M, N, U, k, D), where M is the trail; N and U are the sets of initial and learned clauses, respectively; k is the decision level (i.e., the number of decision literals in M); D is a conflict clause, or the distinguished clause $\top$ if no conflict has been detected. In M, each decision literal is annotated with a level (Decided $L\ k$ or L^k), and each propagated literal is annotated with the clause that caused its propagation (Propagated $L\ C$ or L^C). The level of a propagated literal L is the level of the closest decision literal that follows it in the trail, or 0 if no such literal exists. The level of a clause is the highest level of any of its literals.

The calculus starts in a state $(\epsilon, N, \emptyset, 0, \top)$. The following rules apply as long as no conflict has been detected:

Propagate $(M, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W}} (L^{C \vee L}M, N, U, k, \top)$
if $C \vee L \in N \uplus U$, $M \vDash \neg C$, and L is undefined in M

Conflict $(M, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W}} (M, N, U, k, D)$
if $D \in N \uplus U$ and $M \vDash \neg D$

Decide $(M, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W}} (L^{k+1}M, N, U, k+1, \top)$
if L is undefined in M and occurs in N

Restart $(M, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W}} (\epsilon, N, U, 0, \top)$ if $M \nvDash N$

Forget $(M, N, U \uplus \{C\}, k, \top) \Longrightarrow_{\mathsf{CDCL\_W}} (M, N, U, k, \top)$
if $M \nvDash N$ and M contains no literal L^C

Once a conflict clause is detected and stored in the state, the following rules collaborate to reduce it and backtrack, exploring a first unique implication point [Biere *et al.*, 2009]:

Skip $(L^C M, N, U, k, D) \Longrightarrow_{\mathsf{CDCL\_W}} (M, N, U, k, D)$
if $D \notin \{\bot, \top\}$ and $-L$ does not occur in D

Resolve $(L^{C \vee L}M, N, U, k, D \vee -L) \Longrightarrow_{\mathsf{CDCL\_W}}$
$(M, N, U, k, C \cup D)$ if D is of level k

Jump $(M'K^{i+1}M, N, U, k, D \vee L) \Longrightarrow_{\mathsf{CDCL\_W}}$
$(L^{D \vee L}M, N, U \uplus \{D \vee L\}, i, \top)$
if L is of level k and D is of level i

In combination, these three rules can be simulated by the combined learning and nonchronological backjump rule Learn+Backjump from CDCL_NOT_merge.

CDCL_W has a notion of conclusive state. A state (M, N, U, k, D) is *conclusive* if $D = \top$ and $M \vDash N$ or if $D = \bot$ and N is unsatisfiable. The calculus always terminates but, without suitable strategy, it can stop in an inconclusive state.

4.2 A Reasonable Strategy

To prove correctness, we follow Weidenbach [2015] and assume a *reasonable* strategy: Propagate and Conflict are preferred over Decide; Restart and Forget are not applied. The resulting calculus, CDCL_W+stgy, refines CDCL_W with the assumption that derivations are produced by a reasonable strategy. This assumption is enough to ensure that the calculus can backjump after detecting a conflict clause other than $\bot$. The crucial invariant is the existence of a literal with the highest level in any conflict, so that Resolve can be applied.

Theorem 4 (Correctness). *If* $(\epsilon, N, \emptyset, 0, \top) \Longrightarrow_{\mathsf{CDCL\_W+stgy}}^{!} S'$, *then* S' *is conclusive.*

Once a conflict clause has been stored in the state, the clause is first reduced by a chain of Skip and Resolve transitions. Then, two scenarios are possible: (1) the conflict is solved by a Jump, at which point the calculus may resume propagating and deciding literals; (2) the reduced conflict is $\bot$, meaning that N is unsatisfiable—i.e., for unsatisfiable clause sets, U contains a resolution proof.

The CDCL_W+stgy calculus is designed so that the same clause cannot be learned twice:

Theorem 5 (Relearning). *Let* $(\epsilon, N, \emptyset, 0, \top) \Longrightarrow_{\mathsf{CDCL\_W+stgy}}^{*} (M, N, U, k, D)$. *No* Jump *transition is possible from the latter state causing the addition of a clause from* $N \uplus U$ *to* U.

4.3 Connection with Abstract CDCL

In *Automated Reasoning*, Theorem 5 is used to establish the termination of CDCL_W+stgy. However, the argument for the termination of CDCL_NOT also applies to CDCL_W regardless of the strategy, a stronger result. To lift CDCL_NOT_merge's termination proof, we show that CDCL_W refines CDCL_NOT_merge. The states are easy to connect: We interpret a CDCL_W tuple (M, N, U, k, C) as a CDCL_NOT pair (M, N).

The main difficulty is to relate the low-level conflict-related CDCL_W rules to their high-level counterparts. Our solution is to introduce an intermediate calculus, called CDCL_W_merge, that combines consecutive low-level transitions into a single transition of a new rule called Reduce+Maybe_Jump. This calculus refines both CDCL_W and CDCL_NOT_merge and is sufficiently similar to CDCL_W so that we can transfer termination and other properties from CDCL_NOT_merge through it. Whenever the CDCL_W calculus performs a low-level sequence of transitions of the form $\mathsf{Conflict}\,(\mathsf{Skip} \mid \mathsf{Resolve})^*\,\mathsf{Jump}^?$, the CDCL_W_merge calculus performs a single transition of Reduce+Maybe_Jump.

Since CDCL_W_merge is mostly a rephrasing of CDCL_W, it makes sense to restrict CDCL_W_merge to a *reasonable* strategy that prefers Propagate and Reduce+Maybe_Jump over Decide, yielding CDCL_W_merge+stgy. The two strategy-restricted calculi have the same end-to-end behavior:

$$S \Longrightarrow^{!}_{\mathsf{CDCL\_W\_merge+stgy}} S' \leftrightarrow S \Longrightarrow^{!}_{\mathsf{CDCL\_W+stgy}} S'$$

4.4 Incremental Solving

SMT (satisfiability modulo theories) solvers [Nieuwenhuis *et al.*, 2006] combine a SAT solver with theory solvers. The main loop runs the SAT solver on a set of clauses. If the SAT solver answers "unsatisfiable," the SMT solver is done; otherwise, the main loop asks the theory solvers to provide theory-motivated clauses to exclude the current candidate model. This design relies on incremental SAT solving: the possibility of adding new clauses to the clause set C of a conclusive satisfiable state and of continuing from there. As a step towards formalizing SMT, we designed a calculus CDCL_W+stgy+incr that performs incremental solving on top of CDCL_W+stgy:

$\mathsf{Add\_Strengthening}_C$ $\quad (M, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W+stgy+incr}} S'$
 if $M \not\models \neg C$ and $(M, N \uplus \{C\}, U, k, \top) \Longrightarrow^{!}_{\mathsf{CDCL\_W+stgy}} S'$

$\mathsf{Add\_Conflict}_C$ $\quad (M'LM, N, U, k, \top) \Longrightarrow_{\mathsf{CDCL\_W+stgy+incr}} S'$
 if $LM \models \neg C$, $-L \in C$, M' contains no C literal, L has level i in LM, and $(LM, N \uplus \{C\}, U, i, C) \Longrightarrow^{!}_{\mathsf{CDCL\_W+stgy}} S'$

Theorem 6 (Correctness)**.** *If S is a conclusive state and $S \Longrightarrow_{\mathsf{CDCL\_W+stgy+incr}} S'$, then S' is conclusive.*

5 An Implementation of CDCL

The final link in our refinement chain is a deterministic SAT solver that implements CDCL_W+stgy, expressed as a functional program in Isabelle. We choose to represent states by tuples (M, N, U, k, D), where propositional variables are coded as natural numbers and multisets are represented by lists. Each transition rule in CDCL_W+stgy is implemented by a corresponding function.

The main function invokes the functions for the rules, looking for conflicts before propagating literals. Termination is proved by showing that the call graph is included in CDCL_W+stgy. The correctness and termination theorems can then be lifted, meaning that the SAT solver is a decision procedure for propositional logic. The program is not efficient, but it satisfies the need for a proof of concept.

In subsequent, not-yet-published work, we perform further refinement steps to derive an imperative SAT solver implementation featuring efficient data structures, including the well-known two-watched-literal optimization. Although our solver is not competitive, it offers acceptable performance for some applications, and custom heuristics can easily be added to improve it. Going further and formalizing a competitive SAT solver would be a huge effort. If trustworthiness is desired, it seems preferable to rely on certificates, which can be checked by verified tools.

6 Discussion

At the time when our IJCAR 2016 paper was written, the formalization consisted of about 28 000 lines of Isabelle text. It had been developed over a period of 10 months almost entirely by Fleury, who also taught himself Isabelle during that time. It covered nearly all of the metatheoretical material of Sections 2.6 to 2.11 of *Automated Reasoning* and Section 2 of Nieuwenhuis *et al.*, including normal form transformations and ground unordered resolution.

It is difficult to quantify the cost of formalization as opposed to paper proofs. For a sketchy paper proof, formalization may take an arbitrarily long time; indeed, Weidenbach's nine-line proof of Theorem 5 initially took 700 lines of Isabelle. In contrast, given a very detailed paper proof, one can obtain a formalization in less time than it took to write the paper proof [Woodcock and Banach, 2007]. A hurdle to formalization is often the lack of suitable libraries. For CDCL, we spent considerable time adding definitions, lemmas, and automation hints to Isabelle's multiset library but otherwise did not need any special libraries. We also found that organizing the proof at a high level—especially locale engineering—is more challenging than discharging proof obligations.

The advantages of computer-checked metatheory are well known from programming language research, where papers are often accompanied by formalizations and proof assistants are used in the classroom. This paper reported on some steps we have taken to apply these methods to automated reasoning. We presented a formal framework for SAT solving in Isabelle/HOL, covering the ground between an abstract calculus and a certified SAT solver. We intend to keep following *Automated Reasoning*, thereby demonstrating that interactive theorem proving is mature enough to be of use to practitioners in automated reasoning.

Acknowledgments

We thank everyone who helped us with the conference version of this paper. This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

References

[Ballarin, 2014] Clemens Ballarin. Locales: A module system for mathematical theories. *J. Autom. Reasoning*, 52(2):123–153, 2014.

[Bayardo Jr. and Schrag, 1996] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. In Eugene C. Freuder, editor, *CP96*, volume 1118 of *LNCS*, pages 46–60. Springer, 1996.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Blanchette *et al.*, 2013] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *J. Autom. Reasoning*, 51(1):109–128, 2013.

[Blanchette *et al.*, 2016] Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. In Nicola Olivetti and Ashish Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 25–44. Springer, 2016.

[Cruz-Filipe *et al.*, 2017] Luís Cruz-Filipe, Marijn Heule, Warren Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *CADE-26*, *LNCS*. Springer, 2017.

[Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[Gordon *et al.*, 1979] Michael J. C. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *LNCS*. Springer, 1979.

[Lammich, 2017] Peter Lammich. Efficient verified (UN)SAT certificate checking. In Leonardo de Moura, editor, *CADE-26*, *LNCS*. Springer, 2017.

[Lescuyer, 2011] Stephane Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq*. PhD thesis, 2011.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, 1993.

[Marić, 2010] Filip Marić. Formal verification of a modern SAT solver by shallow embedding into Isabelle/HOL. *Theor. Comput. Sci.*, 411(50):4333–4356, 2010.

[Marques-Silva and Sakallah, 1996] João P. Marques-Silva and Karem A. Sakallah. GRASP—A new search algorithm for satisfiability. In *ICCAD '96*, pages 220–227. IEEE Computer Society Press, 1996.

[Nieuwenhuis *et al.*, 2006] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

[Nipkow and Klein, 2014] Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer, 2014.

[Nipkow *et al.*, 2002] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Oe *et al.*, 2012] Duckki Oe, Aaron Stump, Corey Oliver, and Kevin Clancy. `versat`: A verified modern SAT solver. In Viktor Kuncak and Andrey Rybalchenko, editors, *VMCAI 2012*, volume 7148 of *LNCS*, pages 363–378. Springer, 2012.

[Shankar and Vaucher, 2011] Natarajan Shankar and Marc Vaucher. The mechanical verification of a DPLL-based satisfiability solver. *Electr. Notes Theor. Comput. Sci.*, 269:3–17, 2011.

[Weidenbach, 2015] Christoph Weidenbach. Automated reasoning building blocks. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, volume 9360 of *LNCS*, pages 172–188. Springer, 2015.

[Wenzel, 2007] Makarius Wenzel. Isabelle/Isar—A generic framework for human-readable proof documents. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*. University of Białystok, 2007.

[Woodcock and Banach, 2007] Jim Woodcock and Richard Banach. The verification grand challenge. *J. Univers. Comput. Sci.*, 13(5):661–668, 2007.