Generic CDCL – A Formalization of Modern Propositional Satisfiability Solvers

Steffen Hölldobler, Norbert Manthey, Tobias Philipp and Peter Steinke

International Center for Computational Logic
Technische Universität Dresden

Abstract

Modern propositional satisfiability (or SAT) solvers are very powerful due to recent developments on the underlying data structures, the used heuristics to guide the search, the deduction techniques to infer knowledge, and the formula simplification techniques that are used during pre- and inprocessing. However, when all these techniques are put together, the soundness of the combined algorithm is not guaranteed any more, and understanding the complex dependencies becomes non-trivial. In this paper we present a small set of rules that allows to model modern SAT solvers in terms of a state transition system. With these rules all techniques which are applied in modern SAT solvers can be modeled adequately. Furthermore, we show that this set of rules results is sound, complete and confluent. Finally, we compare the proposed transition system to related systems, and show how widely used solving techniques can be modeled.

1 Introduction

Many practical problems of computer science are in the complexity class NP. Constraint satisfaction [7], answer set programming [36], and satisfiability checking [6] are well studied formalisms that can handle problems of this class. We study the *propositional satisfiability problem* (SAT) that consists of a propositional formula and asks whether there exists a satisfying assignment for the formula. From a complexity theory point of view SAT is NP-complete [8] and, thus, intractable. Still, there exists many industrial and academic applications that can be solved nicely with modern SAT solvers. In many areas in computer science combinatorial problems arise which can be translated into SAT problems and can be solved by finding a model for the SAT problem. For instance a SAT-based railway scheduling software outperformed the native version [14]. Likewise, haplotype matching [23] can be solved nicely with modern SAT solvers. These are only two examples out of the huge range of successfully solved problems [6].

The success of the SAT approach lies in the strength of today's SAT solvers. These solvers construct an assignment by successively interleaving two processes, viz., guessing and propagating the assignment of literals. The main inference rule is *unit propagation*, an efficient form of *resolution*. Combined with a *decision* rule unit propagation is the core of the basic algorithm known as the *DPLL algorithm* [9]. In the case that a contradiction is found in the formula with respect to the current variable assignment, SAT solvers backtrack and learn a conflict clause which prevents the current and similar conflicts. With the addition of so-called *learned clauses* the basic algorithm is known as *CDCL algorithm* [28].

Modern systematic SAT solvers are highly tuned and complex proof procedures employing many advanced techniques like *clause learning* [28], *non-chronological backtracking*, *restarts* [13], *clause removal* [3, 4, 11], *decision heuristics* [4, 29], and formula simplification techniques [20]. Specialized, cache-conscious data structures [17] further contribute to their performance. This way, today's solvers like `Riss`, `MiniSAT` or `Lingeling` can handle formulas with millions of variables and millions of clauses.

However, the success of modern solvers carries a price tag: *increased code complexity.* Successful SAT solvers like the above mentioned ones consist of multiple thousand lines of code and are written in programming languages with side effects like C or C++. Due to this code complexity, the behavior of SAT solvers is hard to understand and state-of-the-art SAT solver internals are hard to teach. Moreover, finding additional techniques and integrating them into a SAT solver is getting more complex, as we have to consider the interplay with all the remaining techniques. Consequently, abstracting from specific algorithms, data structures, and heuristics is extremely important in order to discover and prove properties of a modern SAT solver as well as to understand the principles of SAT solving.

The problem of the increased code complexity was tackled already by different formalizations, notably *Linearized DPLL* [1], *Rule-based SAT Solver Descriptions* [27], and *Abstract DPLL* [31]. However, these systems do not appropriately model modern SAT solvers anymore. In particular, preprocessing and applying preprocessing techniques interleaved with search, known as *inprocessing*, became a crucial part in SAT solving. Applying formula simplification techniques also during search is an attractive idea since it allows to use valuable formula simplifications while taking learned clauses into account. For example, the SAT solver `Lingeling` benefits considerably from this approach. Järvisalo et al. [20] developed a formal system that describes the interplay of formula simplifications and the learned clause database.

The contribution of this paper is a new formalism *Generic CDCL* that allows to model the computation of modern SAT solvers. Equipped with a small set of simple state transition rules, we can model all well-established techniques like preprocessing, inprocessing, restarts, clause sharing, inference techniques stronger than unit propagation, as well as clause learning and forgetting. Generic CDCL allows us to reason about the behavior of SAT solvers independently of the specific implementation. Additionally, the framework is a first step to explain the internals of a modern SAT solvers in a compact and easy way. We neither model the learned clause database explicitly, nor the clause learning procedure nor the model construction algorithms for formula simplification techniques. Instead, we design the rules as general as possible to be able the model techniques such as on-the-fly clause improvement [15] and lazy hyper binary resolution [5]. By being able to simulate each step of a SAT solving technique with a set of rules, a first understanding of the properties of search steps can be given. Hence, the transition system is also a first step to track the single steps of a SAT solver, such that afterwards conclusions might be drawn or to make first conjectures why current systems work well. Still, this paper focuses on the presentation of Generic CDCL, with the main result being the proof that Generic CDCL and, consequently, all its instances are sound. Additionally, the proofs of completeness and confluence of Generic CDCL are presented.

The paper is structured as follows: In Section 2 we describe basic concepts of satisfiability testing. We present Generic CDCL in Section 3, where we also prove that Generic CDCL correctly solves the satisfiability problems, and prove further properties. In Section 4 we give an example how an execution of a SAT solver is modelled by the presented formalism. Afterwards, we compare Generic CDCL with related formalism in Section 5 and we conclude the paper in Section 6.

2 Preliminaries

2.1 The Satisfiability Problem

We assume a fixed infinite set $\mathcal{V}$ of Boolean *variables.* A *literal* is a variable v (*positive literal*) or a negated variable $\overline{v}$ (*negative literal*). The *complement* $\overline{x}$ of a positive (negative, resp.)

literal x is the negative (positive, resp.) literal with the same variable as x. The complement of a set S of literals, denoted with $\overline{S}$, is defined as $\overline{S} = \{\overline{x} \mid x \in S\}$. Finite *multisets* of clauses are called *formulas*, where a *clause* is a finite set of literals. A *multiset* is a set in which elements are allowed to appear more than once. Set operators applied to multisets denote their multiset analogs. In particular, the expression $F \setminus \{C\}$ then denotes the multiset that is obtained from the multiset F by deleting one occurrence of C. Sometimes, we write a clause $\{x_1, \ldots, x_n\}$ also as the disjunction $(x_1 \vee \ldots \vee x_n)$ and a formula $\{C_1, \ldots, C_n\}$ as the conjunction $(C_1 \wedge \ldots \wedge C_n)$. The empty clause is denoted by $\perp$, the empty formula by $\top$. The set of all variables occurring in a formula F (in positive or negative literals) is denoted by $\mathsf{vars}(F)$; the set of all literals occurring in F as elements in the set representation is denoted by $\mathsf{lits}(F)$. For instance, if $x, y \in \mathcal{V}$, then $F = \{\{\overline{x}, y\}, \{y\}\}$ is a formula, its alternative representation using logical connectives is $(\overline{x} \vee y) \wedge y$, $\mathsf{vars}(F) = \{x, y\}$, and $\mathsf{lits}(F) = \{\overline{x}, y\}$.

The semantics of formulas is based on the notion of an interpretation I, represented by a set of literals which does not contain a complementary pair x, $\overline{x}$ of literals. An interpretation I is *total* iff for each $v \in \mathcal{V}$ either $v \in I$ or $\overline{v} \in I$. The satisfaction relation $\models$ is defined as follows: Let I be an interpretation, then $I \models \top$, $I \not\models \perp$, $I \models (x_1 \vee \ldots \vee x_n)$ iff $I \models x_i$ for some $i \in \{1, \ldots, n\}$, and $I \models (C_1 \wedge \ldots \wedge C_n)$ iff $I \models C_i$ for all $i \in \{1, \ldots, n\}$. The interpretation I is a *model* for the formula F iff $I \models F$. In the case that a formula F has a model, the formula F is *satisfiable*, otherwise the formula F is *unsatisfiable*. We modify interpretations, i.e. the expression $I[S]$, where S is a set of literals that does not contain a complementary pair of literals, denotes the interpretation defined as follows: $I[S] \models x$ for every $x \in S$ and $I[S] \models y$ if $\{y, \overline{y}\} \not\subseteq S$ and $I \models y$.

We relate formulas by three relations: the *entailment*, the *equivalence* and the *equisatisfiability* relation: A formula F *entails* a formula F' iff every total model of F is a model of F'. Two formulas F and F' are *equivalent*, in symbols $F \equiv F'$, iff F entails F' and F' entails F. Two formulas F and F' are *equisatisfiable*, in symbols $F \equiv_{\mathsf{sat}} F'$, iff either both formulas are satisfiable or both formulas are unsatisfiable.

Let x be a literal, C be a clause in which x occurs, and D be a clause in which $\overline{x}$ occurs. Then the clause $(C \cup D) \setminus \{x, \overline{x}\}$ is the *resolvent of the clauses C and D upon the literal x*. A *linear resolution derivation from the clause C to the clause D in the formula F* is a finite sequence of clauses $(C_i \mid 1 \leq i \leq n)$ such that $C_1 = C$, $C_n = D$ and C_i is a resolvent of the clause C_{i-1} and some clause in the formula F for all $i \in \{2, \ldots, n-1\}$; the reader should observe that F entails D and that the addition of entailed clauses to a formula preserves equivalence.

2.2 Variable Assignments and the Reduct Operator

Let J be a finite sequence of literals. In J each literal may be marked as a *decision literal* by placing a dot on top like in $\dot{x}$; if a literal x is not marked, then this literal is a *propagation literal*. Let J be a sequence of literals of length n. We say that literal $x \in J$ iff there is a $k \in \{1, \ldots, n\}$ such that $x = x_k$. Let $J_1 = (x_1, \ldots, x_n)$ and $J_2 = (y_1, \ldots, y_m)$ be two sequences of literals; their concatenation $J_1 J_2$ is the sequence $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. If a finite sequence J of literals does not contain a complementary pair of literals, then J represents an interpretation. The empty sequence of literals is denoted by ε. As this condition is always met in this paper, we identify sequences of literals with interpretations whenever appropriate.

The *reduct of a formula F w.r.t. an interpretation J*, in symbols $F|_J$, is defined as

$$F|_J := \{C|_J \mid C \in F \text{ and for every literal } x \in C \text{ we find that } x \notin J\},$$

where $C|_J = C \setminus \{\overline{x} \mid x \in J\}$. Intuitively, the reduct operator expresses the state of a SAT solver, where the formula F is the *working formula* and J is the *working interpretation*. For instance, let $F = \{\{\overline{x}, y\}, \{z\}\}$, then $F|_x = \{\{y\}, \{z\}\}$, $F|_{\overline{z}} = \{\{\overline{x}, y\}, \bot\}$ and $F|_{y\,z} = \top$, where the interpretations are written as sequences of literals. The reader should observe that the reduct operator does not distinguish between propagation and decision literals. The properties of the reduct operator are summarized in Lemma 1:

Lemma 1 (Reduct Operator)**.** *Let F, F' be formulas and x a literal.*

1. *If $J \subseteq I$, then $I \models F$ if and only if $I \models F|_J$.*
2. *$I \models F|_J$ if and only if $I[J] \models F$.*
3. *$I \models F|_J$ if and only if there exists I' such that $I'(x) = I(x)$ for every $x \notin J$ and $I' \models F \wedge \bigwedge_{x \in J} x$.*
4. *If $F \equiv F'$, then $F|_J \equiv F'|_J$ for every interpretation J.*
5. *F is satisfiable iff there exists a J such that $F|_J = \top$.*

Proof. For details see [34, pp.10–12].

1. follows straightforward from the definition of the reduct.

2. We show both directions:

$\Rightarrow$ Suppose that the claim is incorrect, i.e. $I[J] \not\models F$. Then there is a clause $C \in F$ such that $I[J] \not\models C$. Then there is no literal $x \in C$ such that 1) $x \in J$ or 2) $x \in I$ and $\overline{x} \notin J$. We now distinguish between two cases, and show that there is a literal x that contradicts 1) or 2).

- $C|_J \notin F|_J$: Then there is a literal $x' \in C$ such that $x' \in J$ and therefore we have a contradiction to 1).
- $C|_J \in F|_J$: Since $I \models F|_J$, we know that $I \models C|_J$. Then there is a literal $x' \in C|_J$ such that $x' \in I$. Since $x' \in C$, we know that $\overline{x'} \notin J$ by the definition of the reduct operator. Therefore, we have a contradiction to 2).

$\Leftarrow$ Let $I[J] \models F$. Since $J \subseteq I[J]$ we know that $I[J] \models F|_J$ by Lemma 1.1. Moreover, we know that $\mathsf{vars}(F|_J) \cap (J \cup \overline{J}) = \emptyset$, i.e. the literals in J do not occur in $F|_J$. Consequently, $I \models F|_J$.

Since in both cases, we have a contradiction to the assumption that $I[J] \not\models F$, we conclude that $I[J] \models F$.

3. follows straightforward from Lemma 1.2.

4. To show the claim, notice that Lemma 1.3 characterizes the reduct operator in a *semantic* way. Consequently, the reduct operator cannot distinguish equivalent formulas.

5. We show both directions:

$\Rightarrow$ Let $J = \emptyset$. If the formula F is satisfiable, we know that the formula $F|_J = F$ is satisfiable.

$\Leftarrow$ Let the formula $F|_J$ be satisfiable. By Lemma 1.3 we know that there exists an interpretation I' such that $I' \models F \wedge \bigwedge_{x \in J} x$. It immediately follows that $I' \models F$ and hence, the formula F is satisfiable. □

SAT-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{SAT}} \mathsf{SAT}$ iff $F|_J = \top$.

UNSAT-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$ iff
$\bot \in F|_J$ and J contains only propagation literals.

DEC-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{DEC}} F \mathbin{/\!\!/} J\,\dot{x}$ iff
$x \in \mathsf{vars}(F) \cup \overline{\mathsf{vars}(F)}$ and $\{x, \overline{x}\} \cap J = \emptyset$.

INF-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{INF}} F \mathbin{/\!\!/} J\,x$ iff
$F|_J \equiv_{\mathsf{sat}} F|_{J\,x}$, $x \in \mathsf{vars}(F) \cup \overline{\mathsf{vars}(F)}$ and $\{x, \overline{x}\} \cap J = \emptyset$.

LEARN-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{LEARN}} F \cup \{C\} \mathbin{/\!\!/} J$ iff $F \models C$.

REMOVE-rule: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{REMOVE}} F \setminus \{C\} \mathbin{/\!\!/} J$ iff $F \setminus \{C\} \models C$.

BACK-rule: $F \mathbin{/\!\!/} J\,J' \rightsquigarrow_{\mathsf{BACK}} F \mathbin{/\!\!/} J$.

INP-rule: $F \mathbin{/\!\!/} \varepsilon \rightsquigarrow_{\mathsf{INP}} F' \mathbin{/\!\!/} \varepsilon$ iff $F \equiv_{\mathsf{sat}} F'$.

Figure 1: Transition relations of Generic CDCL. These relations apply to all formulas F and F', clauses C, literals x and sequences of literals J and J'.

3 Generic CDCL

Modern SAT solvers are rooted in the linearized DPLL [9] algorithm and consist of the following components: *termination criteria*, a *decision component* that selects the branching literals, an *inference component* that adds propagation literals to the working interpretation, a *backtracking component* that rolls back wrong decisions and a *formula management component* that simplifies the working formula. We maintain two data structures during modelling a modern SAT solver: the *working formula* F, that is the multiset of clauses that is currently present in the SAT solver, and the *working interpretation* J, which corresponds to the partial interpretation that is build by the SAT solver. Together, these two components define the *state*, which is the pair $F \mathbin{/\!\!/} J$. The components of the solver are modelled as a transition relation over the set of states; the union of the rules depicted in Fig. 1 is then the transition relation of Generic CDCL. Formally, we model the computation of modern SAT solvers by means of state transition systems as follows:

Definition 1 (Generic CDCL)**.** Generic CDCL *is a state transition system whose sets of states is*

$$\{F \mathbin{/\!\!/} J \mid F \textit{ is a formula and } J \textit{ is a sequence of literals}\} \cup \{\mathsf{SAT}, \mathsf{UNSAT}\},$$

whose initial state for the input formula F *is* $\mathsf{init}(F) = F \mathbin{/\!\!/} \varepsilon$*, whose set of terminal states is* $\{\mathsf{SAT}, \mathsf{UNSAT}\}$*, and whose transition relation* $\rightsquigarrow$ *is defined as:*

$$\rightsquigarrow \;:=\; \{\rightsquigarrow_{\mathsf{SAT}}, \rightsquigarrow_{\mathsf{UNSAT}}, \rightsquigarrow_{\mathsf{DEC}}, \rightsquigarrow_{\mathsf{INF}}, \rightsquigarrow_{\mathsf{LEARN}}, \rightsquigarrow_{\mathsf{REMOVE}}, \rightsquigarrow_{\mathsf{BACK}}, \rightsquigarrow_{\mathsf{INP}}\}.$$

The SAT-rule terminates the computation with the output SAT, if the reduct of the working formula w.r.t. the working interpretation is the empty formula. This condition can be decided in linear time w.r.t. the size of the working formula F. By Lemma 1.5 the working formula is then satisfiable.

The UNSAT-rule terminates the computation with the output UNSAT, if no model of the working formula exists. The lack of a model is the case when a conflict occurs in the top level, i.e. $\bot \in F|_J$ and the interpretation J contains only propagation literals. These conditions can be decided in polynomial time, since the interpretation as well as each clause have to be checked at most once.

The DEC-rule extends the working interpretation by an unassigned literal $\dot{x}$ as a decision literal. The variable of the decision literal must occur in the working formula.

The INF-rule extends the working interpretation by a propagation literal x, if the reducts of the working formula w.r.t. the working interpretation and its extension are equisatisfiable. Essentially, this rule covers unit propagation. However, adding other implied literals, as for example finding *backbone literals* [33], performing *look-ahead* on an intermediate search node [16] or adding unit clauses based on *probing* [24] and *extended unit propagation* [21] is also covered by the given condition.

The BACK-rule models backtracking, as well as *backjumping*, *assignment stack shrinking* [30] and *restarts* [13], by deleting outermost right literals in the working interpretation.

The LEARN-rule adds a clause C to the working formula, if this clause is entailed by the working formula F. Deciding whether $F \models C$ holds, is coNP-complete. Similarly to the INF-rule, SAT solvers avoid this check by using techniques for creating the clause C that ensure this property, as for example resolution. To the best of our knowledge, all learning techniques, ranging from learning *UIP clauses* [29], bi-asserting clauses [18,35] to *clause minimization* [38] use only resolution, such that all these techniques are covered by this rule.

The REMOVE-rule deletes one occurrence of the clause C of the working formula F, if $F \setminus \{C\} \models C$. The question whether $F \setminus \{C\} \models C$ holds is coNP-complete. Typically, we use tractable algorithms to identify redundant clauses. For instance, clauses that were introduced by the LEARN-rule but have turned out to be useless and did not participate in the elimination of other clauses in the formula can be removed. For more details on the deletion of clauses see [20].

The INP-rule models formula simplifications that are used in pre- and inprocessing. The rule replaces the working formula with an equisatisfiable formula when the working interpretation is empty. To the best of our knowledge, this way all used formula simplification techniques that are used during preprocessing and inprocessing can be modelled [20].

Let $\overset{*}{\rightsquigarrow}$ be the reflexive and transitive closure of $\rightsquigarrow$. We define $x \overset{0}{\rightsquigarrow} x$ for all states x, and $x \overset{n}{\rightsquigarrow} z$ for all natural numbers $n \in \mathbb{N}$ if and only if there exists a state y such that $x \overset{n-1}{\rightsquigarrow} y \rightsquigarrow z$. In the next subsection we investigate the question whether Generic CDCL correctly solves the SAT problem.

3.1 Generic CDCL is Sound

Formally, we define Generic CDCL to be *sound* iff for all formulas F_0 we have that $\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} \mathsf{SAT}$ implies that F_0 is satisfiable and $\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} \mathsf{UNSAT}$ implies that F_0 is unsatisfiable.

Before proceeding to the soundness proof of Generic CDCL, two invariants of Generic CDCL are studied. These invariants are presented in the proposition below: *Invariant 1* states that the rules of Generic CDCL do not change the satisfiability of the working formula, and *invariant 2* states that whenever the working interpretation is of the form $J_1\,x\,J_2$, where x is a propagation literal, then the reducts of the working formula w.r.t. J_1 and $J_1\,x$ are equisatisfiable.

Proposition 1 (Invariants)**.** *Let F_0, F be formulas, J be a sequence of literals, and $n \in \mathbb{N}$. If $\mathsf{init}(F_0) \overset{n}{\rightsquigarrow} F \mathbin{/\!/} J$, then*

1. *$F_0 \equiv_{\mathsf{sat}} F$, and*
2. *$F|_{J_1} \equiv_{\mathsf{sat}} F|_{J_1\,x}$, for all sequences of literals J_1, J_2 and propagation literals x with $J = J_1\,x\,J_2$.*

Proof. The claims are proven by induction on the number of steps n. For the base case $n = 0$, *1.* follows from $F_0 = F$ and *2.* holds since the J is empty. For the induction step, assume that the claim holds for the state $F \mathbin{/\!/} J$ and suppose that $F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{R}} F' \mathbin{/\!/} J'$, where

$$\mathsf{R} \in \{\mathsf{DEC}, \mathsf{INF}, \mathsf{LEARN}, \mathsf{REMOVE}, \mathsf{BACK}, \mathsf{INP}\}.$$

- DEC-rule: In this case, $F' = F$ and $J' = J\,\dot{x}$ for some decision literal $\dot{x}$ with $\{x, \overline{x}\} \cap J = \emptyset$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ holds by induction. *2.* holds because the appended literal is a decision literal. Formally, let J_1', J_2' be literal sequences, y be a propagation literal such that $J' = J_1'\,y\,J_2'\dot{x}$. By induction, we conclude that $F|_{J_1'} \equiv_{\mathsf{sat}} F|_{J_1'\,y}$. Hence, $F'|_{J_1'} \equiv_{\mathsf{sat}} F'|_{J_1\,y}$.
- INF-rule: In this case, $F' = F$ and $J' = J\,x$ for some propagation literal x with $\{x, \overline{x}\} \cap J = \emptyset$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ holds by induction. *2.* follows from the definition of the INF-rule: Consider the literal sequences J_1', J_2' and a propagation literal y such that $J' = J_1'\,y\,J_2'$. In the case that $y = x$, we know that J_2' is the empty sequence of literals and consequently $F|_{J_1'} \equiv_{\mathsf{sat}} F|_{J_1'\,y}$ holds by the definition of the INF-rule. In the case of $y \neq x$, we can conclude the claim by induction.
- BACK-rule: In this case, $F' = F$ and $J = J'\,J''$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ by induction. *2.* holds because the literal sequence J' is a prefix of J. Formally, let J_1', J_2' be literal sequences and y be a propagation literal such that $J' = J_1'\,y\,J_2'$. By induction, we conclude that $F|_{J_1'} \equiv_{\mathsf{sat}} F|_{J_1'\,y}$, and consequently we know that $F'|_{J_1'} \equiv_{\mathsf{sat}} F'|_{J_1'\,y}$.
- LEARN-rule: In this case, $F' = F \cup \{C\}$ where $F \models C$ and $J' = J$. *1.* follows because the addition of the entailed clause C preserves the equivalence between F and $F \cup \{C\}$. *2.* follows from the reduct operator being a semantic operator by Lemma 1.4 and therefore $F'|_{J_1'} \equiv_{\mathsf{sat}} F'|_{J_1'\,y}$ holds by induction for every literal sequences J_1', J_2' and propagation literals y with $J' = J_1'\,y\,J_2'$.
- REMOVE-rule: This case can be treated as in the LEARN-rule.
- INP-rule: In this case, $F' \equiv_{\mathsf{sat}} F$ and J' is the empty sequence. Consequently, *1.* holds by the definition of the INP-rule, and *2.* is satisfied as $J'' = \varepsilon$. □

We can now show the first main theorem in this paper.

Theorem 1 (Soundness)**.** *Generic CDCL is sound.*

Proof. We divide the proof in two parts, first proving that the output SAT is correct, and then proving that the output UNSAT is correct. Let F_0, F be formulas, J be a sequence of literals and suppose that

$$\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} F \mathbin{/\!/} J \rightsquigarrow \mathsf{SAT}(\mathsf{UNSAT}, \text{resp.}).$$

SAT. By the definition of the SAT-rule, we know that $F|_J = \top$. By Lemma 1.5, we know that the formula F is satisfiable. From the result that the formula F is satisfiable together with the property that the formulas F_0 and F are equisatisfiable as shown in Prop. 1(*1.*), we conclude that the input formula F_0 is satisfiable.

UNSAT. By the definition of the UNSAT-rule, we know that $\bot \in F|_J$ and the working interpretation $J = (x_1 \ldots x_n)$ contains only propagation literals. Since a conflict occurs, $F|_J$ is unsatisfiable. From the result that the formula $F|_J$ is unsatisfiable and the working interpretation J contains only propagation literals we can repeatably apply Prop. 1(*2.*) and obtain that the formula F is unsatisfiable. Since the formula F is unsatisfiable and the formulas F and F_0 are equisatisfiable by Prop. 1(*1.*), we conclude that F_0 is unsatisfiable. □

3.2 Generic CDCL is Complete

We continue to show completeness of Generic CDCL: If the input formula F_0 is satisfiable, then $\mathsf{init}(F_0) \overset{*}{\leadsto} \mathsf{SAT}$ and if the formula F is unsatisfiable, then $\mathsf{init}(F_0) \overset{*}{\leadsto} \mathsf{UNSAT}$.

Theorem 2. *Generic CDCL is complete.*

Proof. Suppose that the input formula F_0 is unsatisfiable. Then, $F_0 \models \bot$ and consequently $\mathsf{init}(F_0) = F_0 \mathbin{/\!\!/} \varepsilon \leadsto_{\mathsf{LEARN}} F \wedge \bot \mathbin{/\!\!/} \varepsilon \leadsto_{\mathsf{UNSAT}} \mathsf{UNSAT}$. Consequently, $\mathsf{init}(F_0) \overset{*}{\leadsto} \mathsf{UNSAT}$. Now, suppose that the input formula F_0 is satisfiable. Then, there is a model $J = (x_1\, x_2 \ldots x_n)$ for the formula F_0. Then, $\mathsf{init}(F_0) = F_0 \mathbin{/\!\!/} \varepsilon \leadsto_{\mathsf{DEC}} F \mathbin{/\!\!/} (\dot{x_1}) \ldots \leadsto_{\mathsf{DEC}} F \mathbin{/\!\!/} (\dot{x_1}\, \dot{x_2} \ldots \dot{x_n}) \leadsto_{\mathsf{SAT}} \mathsf{SAT}$. Consequently, $\mathsf{init}(F_0) \overset{*}{\leadsto} \mathsf{SAT}$. Hence, Generic CDCL is complete. □

The result is mainly of theoretical interest, as it guarantees the existence of complete instantiations of Generic CDCL. In deed, not every instance of Generic CDCL is complete. In practice, we want to have the chance to reach a final state from *every* reachable state, and not just from the initial state. This property follows from *confluence* and completeness.

3.3 Generic CDCL is Confluent

Generic CDCL is *confluent* iff for every formula F it holds that if $\mathsf{init}(F_0) \overset{*}{\leadsto} T_1$ and $\mathsf{init}(F_0) \overset{*}{\leadsto} T_2$, then there is a state T such that $T_1 \overset{*}{\leadsto} T$ and $T_2 \overset{*}{\leadsto} T$. In particular, if Generic CDCL is confluent and complete, then we can reach a final state in every reachable state. Note that our definition is slightly different to the standard notion of confluence as it restrict to reachable states.

Theorem 3. *Generic CDCL is confluent.*

Proof. Roughly speaking, confluence of Generic CDCL follows from its completeness and the ability to perform restarts by the BACK-rule. Suppose that $\mathsf{init}(F_0) \overset{*}{\leadsto} T_1$ and $\mathsf{init}(F_0) \overset{*}{\leadsto} T_2$. We have to show that $T_1 \overset{*}{\leadsto} T$ and $T_2 \overset{*}{\leadsto} T$ for some state T. Consider the case that F_0 is satisfiable. Then we know that $T_1 \neq \mathsf{UNSAT}$ and $T_2 \neq \mathsf{UNSAT}$ because Generic CDCL is sound by Theorem 1. We distinguish between the following cases:

- $T_1 = F_1 \mathbin{/\!\!/} J_1$, $T_2 = F_2 \mathbin{/\!\!/} J_2$. By Prop 1 (1) it holds that $F_0 \equiv_{\mathsf{sat}} F_1$ and $F_0 \equiv_{\mathsf{sat}} F_2$. Hence, F_1 and F_2 are satisfiable. Since Generic CDCL is complete by Theorem. 2, we know that $F_1 \mathbin{/\!\!/} J_1 \leadsto_{\mathsf{BACK}} F_1 \mathbin{/\!\!/} \varepsilon \overset{*}{\leadsto} \mathsf{SAT}$ and $F_2 \mathbin{/\!\!/} J_2 \leadsto_{\mathsf{BACK}} F_2 \mathbin{/\!\!/} \varepsilon \overset{*}{\leadsto} \mathsf{SAT}$, i.e. $T = \mathsf{SAT}$.

- $T_1 = \mathsf{SAT}$, $T_2 = F_2 \mathbin{/\!/} J_2$. By Prop. 1 (1) it holds that $F_0 \equiv_{\mathsf{sat}} F_2$. Hence, F_2 is satisfiable. Since Generic CDCL is complete by Theorem 2, we know that $F_2 \mathbin{/\!/} J_2 \leadsto_{\mathsf{BACK}} F_2 \mathbin{/\!/} \varepsilon \overset{*}{\leadsto} \mathsf{SAT}$. Then, $T = \mathsf{SAT}$.
- $T_1 = F_1 \mathbin{/\!/} J_1$, $T_2 = SAT$. This case can be shown as above.
- $T_1 = \mathsf{SAT}$, $T_2 = \mathsf{SAT}$. In this case, $T = T_1 = T_2$.

The case that F_0 is unsatisfiable can be proven in a similar way. □

3.4 Generic CDCL Subsumes Important SAT Solving Techniques

We now describe a few selected SAT solving techniques, and demonstrate that Generic CDCL can adequately model these techniques.

Subsumption. For a formula F, the clause $C \in F$ *subsumes* the clause $D \in F$ iff $C \subseteq D$. In this case, D can be deleted if $C \neq D$ because $F \setminus \{D\} \models D$. Consequently, $F \mathbin{/\!/} J \leadsto_{\mathsf{REMOVE}} F \setminus \{D\} \mathbin{/\!/} J$ holds for every literal sequence J. Removing subsumed clauses is done as a preprocessing step in SAT solvers and during clause learning, for example when a learned clause is a unit clause.

Tautologies. A clause C is a *tautology* if it contains a complementary pair of literals. Every formula F entails a tautology and the REMOVE-rule in Generic CDCL subsumes this techniques. Tautologies are eliminated during preprocessing.

Conflict-Directed Backtracking and Learning [37]. This technique is an improvement of naive backtracking that takes the reason of the conflict into account. Consider the state $F \mathbin{/\!/} J\,\dot{x}\,J'$ and a clause $C \in F$ where $C|_{J\,\dot{x}\,J'} = \bot$. The clause C is the *conflict clause.* If there is a *linear resolution derivation* from the conflict clause C to a clause D in the formula F such that $D|_J$ is the unit clause y, the technique rewrites the state $F \mathbin{/\!/} J\,\dot{x}\,J'$ into the state $F \cup \{C\} \mathbin{/\!/} J\,y$. Conflict-directed backtracking and learning can be simulated by the following transition steps: $F \mathbin{/\!/} J\,\dot{x}\,J' \leadsto_{\mathsf{BACK}} F \mathbin{/\!/} J \leadsto_{\mathsf{LEARN}} F \cup \{D\} \mathbin{/\!/} J \leadsto_{\mathsf{INF}} F \cup \{D\} \mathbin{/\!/} J\,y$.

Unit Propagation. A clause that contains a single literal is a *unit clause.* Unit propagation adds the propagation literal x to the literal sequence J, whenever the reduct of the working formula w.r.t. J contains the unit clause (x). Since $F|_J \models x$, we know that $F|_J \equiv_{\mathsf{sat}} F|_{J\,x}$ and consequently the INF-rule subsumes unit propagation.

Pure Literal. A literal x is *pure* in the formula F, if $x \in \mathsf{lits}(F)$ and $\overline{x} \notin \mathsf{lits}(F)$. For pure literals, it holds that $F \equiv_{\mathsf{sat}} F|_x$ and, therefore, whenever a literal x is pure in the formula $F|_J$ for some literal sequence J, Generic CDCL can add the pure literal to the working interpretation: $F \mathbin{/\!/} J \leadsto_{\mathsf{INF}} F \mathbin{/\!/} J\,x$.

On-the-fly Clause Improvement [15]. Given a formula $(F \wedge C \wedge D)$ such that $C \otimes D$ subsumes the clause C, self-subsuming resolution produces the formula $(F \wedge (C \otimes D) \wedge D)$. It is straight-forward to see that the result of self-subsuming resolution can obtained by the REMOVE- and LEARN-rule.

Variable Elimination [10, 39]. For two formulas F_1, F_2 and a variable v, the set of all non-tautological resolvents of a clause in the formula F_1 with a clause in the formula F_2 upon the variable v is denoted by $F_1 \otimes_v F_2$. Given an input formula F and a variable $v \in \mathcal{V}$, variable elimination adds all resolvents of the input formula F upon the variable v, and afterwards deletes all clauses containing the literals v or $\overline{v}$. Variable Elimination can be applied when the working interpretation is empty: The addition of resolvents preserves the equivalence of formulas and the LEARN-rule is used. Afterwards, the clauses containing the literals v or $\overline{v}$ are eliminated, which is done with the INP-rule.

Blocked Clause Elimination [19]. A clause C is *blocked* in the formula F if it contains a literal x such that all resolvents of the clause C and clauses $D \in F$ with $\overline{x} \in D$ upon x are tautological. Blocked clauses are removed from a formula during pre- and inprocessing. If C is blocked in F, then $F \equiv_{\mathsf{sat}} F \setminus \{C\}$ and, therefore, the INP-rule subsumes the blocked clause elimination technique.

Extended Resolution [2]. A variable v is fresh in a certain context, if it does not occur in a formula. Extended resolution adds a definition of a fresh variable v to the formula, i.e. given an input formula F, two literals $x, y \in \mathsf{lits}(F)$ and the fresh variable v, the technique produces the formula $F' = F \wedge (\overline{v} \vee x \vee y) \wedge (\overline{x} \vee v) \wedge (\overline{y} \vee v)$. We can model extended resolution as follows: First, we perform a restart such that the INP-rule is applicable. Afterwards, we add the clauses from extended resolution and construct the original working interpretation by applying the INF- and DEC-rule. Note that these steps are possible, since extended resolution uses *fresh* variables, and no clauses are removed, such that all propagated literals can be added to J again after the restart.

Bounded Variable Addition [25]. This technique adds a partial definition of a fresh variable v to the formula: First, a fresh variable v is introduced like in extended resolution, resulting in the intermediate formula $G = F \wedge (v \vee \overline{x} \vee \overline{y}) \wedge (\overline{v} \vee x) \wedge (\overline{v} \vee y)$, where $x, y \in \mathsf{lits}(F)$. Next, all clauses $C, D \in G \setminus \{(\overline{v} \vee x), (\overline{v} \vee y)\}$ which have a common subclause E such that $C = (x \vee E)$ and $D = (y \vee E)$ are replaced by the new clause $(v \vee E)$, $G := (G \setminus \{(x \vee E), (y \vee E)\}) \cup \{(v \vee E)\}$. Finally, the formula F', the result of applying bounded variable addition, is obtained from the formula H by removing the clause $(v \vee \overline{x} \vee \overline{y})$. Bounded Variable Addition is applicable when the working interpretation is empty and can be simulated in terms of the INP-rule.

4 A Realistic SAT Trace as Generic CDCL

We will illustrate the interplay of these components in form of a trace of a modern SAT solver and use the following notation: the *double underlined literals* are literals that are mapped to $\perp$ by the working interpretation, and *underlined clauses* are satisfied by the working interpretation. Let $\{a, b, c, \ldots, g\}$ be a set of variables and consider the formula F and the empty literal sequence ε, that form an initial state:

$$\begin{array}{rcl} I & = & \varepsilon \\ F & = & (a \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (\neg c \vee e) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg d \vee \neg c) \end{array}$$

As a preprocessing step, we apply *blocked clause elimination* and find that the clause $(a \vee c)$

has the blocking literal a. Hence, we apply the INP-rule and obtain:

$$\begin{aligned} I &= \varepsilon \\ F &= (\neg a \lor \neg c \lor d) \land (\neg c \lor e) \land (b \lor \neg c) \land (\neg b \lor \neg d \lor \neg c) \end{aligned}$$

Now, suppose the decision component is initialized such that it decides the literals in alphabetically ordering and always selects the positive literal. First, the literal a is selected and we obtain the following state after applying the DEC-rule:

$$\begin{aligned} I &= (\dot{a}) \\ F &= (\underline{\underline{\neg a}} \lor \neg c \lor d) \land (\neg c \lor e) \land (b \lor \neg c) \land (\neg b \lor \neg d \lor \neg c) \end{aligned}$$

We then select the literal b and obtain the following state after applying the DEC-rule:

$$\begin{aligned} I &= (\dot{a}\,\dot{b}) \\ F &= (\underline{\underline{\neg a}} \lor \neg c \lor d) \land (\neg c \lor e) \land \underline{(b \lor \neg c)} \land (\underline{\underline{\neg b}} \lor \neg d \lor \neg c) \end{aligned}$$

Then, we select the literal c and obtain the following state after applying the DEC-rule:

$$\begin{aligned} I &= (\dot{a}\,\dot{b}\,\dot{c}) \\ F &= (\underline{\underline{\neg a}} \lor \underline{\underline{\neg c}} \lor d) \land (\underline{\underline{\neg c}} \lor e) \land \underline{(b \lor \neg c)} \land (\underline{\underline{\neg b}} \lor \neg d \lor \underline{\underline{\neg c}}) \end{aligned}$$

Now, we can infer the literals d and e by unit propagation. Unit propagation is subsumed by the INF-rule, i.e. we apply the rule twice, resulting in the following state:

$$\begin{aligned} I &= (\dot{a}\,\dot{b}\,\dot{c}\,d\,e) \\ F &= \underline{(\neg a \lor \neg c \lor d)} \land \underline{(\neg c \lor e)} \land \underline{(b \lor \neg c)} \land (\underline{\underline{\neg b}} \lor \underline{\underline{\neg d}} \lor \underline{\underline{\neg c}}) \end{aligned}$$

The clause $(\neg b \lor \neg d \lor \neg c)$ cannot be satisfied any more, i.e. the formula together with the made decisions is unsatisfiable. Consequently, we apply conflict-directed backtracking and learning as follows: First, we compute a linear resolution derivation in the *reason clauses* from the *conflict clause*: $(\neg b \lor \neg d \lor \neg c) \otimes (\neg a \lor \neg c \lor d) = (\neg a \lor \neg b \lor \neg c)$. We minimize the resulting clause by *self-subsuming resolution*, i.e. $(\neg a \lor \neg b \lor \neg c) \otimes (b \lor \neg c) = (\neg a \lor \neg c) =: C$. Then, the clause C is added to the formula with the LEARN-rule. Finally, we apply the BACK-rule such that the clause C becomes a unit. Moreover, we eliminate the first clause in F since C subsumes the clause, i.e. we apply the REMOVE-rule.

$$\begin{aligned} I &= (\dot{a}\,\neg c) \\ F &= \underline{(\neg c \lor e)} \land \underline{(b \lor \neg c)} \land \underline{(\neg b \lor \neg d \lor \neg c)} \land \underline{(\neg a \lor \neg c)} \end{aligned}$$

Now, the reduct of the working formula w.r.t. the working interpretation is empty since each clause is satisfied by the working interpretation. Hence, the SAT-rule is applicable and we terminate the computation with the answer SAT.

5 Related Work

Several attempts have been made to formalize sequential SAT solvers in terms of transition systems or proofs calculi: Abstract DPLL [31], Linearized DPLL [1], and Rule-based SAT solver description [27]. For very similar problem types and their solvers, *SAT modulo theory* solvers, the formalism *DPLL(T)* has been proposed [32]. In general, these formalizations are based on a notion of state like our formalism Generic CDCL.

However, these formalizations cannot adequately model recent SAT solvers: For instance, Linearized DPLL does not model the SAT solver `MiniSAT`, because Linearized DPLL restricts decision literals to occur in the working formula, but the solver `MiniSAT` can also select the complements of such literals. Additionally, Linearized DPLL does not model formula simplification techniques such as blocked clause elimination, or probing-based inference techniques. Similarly, Abstract DPLL and the Rule-based SAT solver description [27] do not model formula simplifications that changes the semantics of formulas, i.e. the set of models, like blocked clause elimination. Marić highlighted the implementation of clause learning techniques in his Rule-based SAT solver description [27], but it does not include recent developments such as *clause strengthening*. All these formalizations consider DPLL-based SAT solvers, but the ancient *pure literal rule* is not subsumed in these systems. While DPLL(T) models the pure literal rule, this formalism does not cover formula simplifications, or inference techniques that are more powerful than unit propagation. In contrast, to the best of our knowledge, Generic CDCL subsumes all recent SAT techniques.

In [20] Jarvisalo et al. developed a formal system to model clause learning, forgetting and formula simplification techniques to understand the side-effects of the combination of different rules. In particular, they construct the model for an initial formula by characterizing the state of SAT solvers in terms of a set of redundant clauses, a set of irredundant clauses and a stack of clauses.

The interplay of clause sharing and formula simplification techniques in *parallel* SAT solvers was analyzed in [26], where the state of a sequential SAT solver was modelled just as the working formula. We believe that Generic CDCL is an important fragment to understand sequential SAT solvers with inprocessing and their cooperation in the parallel-portfolio setting with clause sharing.

6 Conclusion

The propositional satisfiability problem is of great practical interest and can be efficiently answered by modern SAT solvers like `Riss`, `Lingeling` or `MiniSAT`. Today, modern SAT solvers are highly tuned proof procedures with many advanced techniques. Therefore, it is desirable to investigate SAT solving techniques in combination with each other and to abstract from implementations.

In this paper, we developed Generic CDCL, a formalism that models the computation of modern SAT solvers in terms of a state transition system, where each transition rule abstracts a component in a SAT solver. In particular, the transition rules INF and INP model formula simplification techniques like blocked clause elimination and inference techniques such as the pure literal rule. We have examined invariants in Generic CDCL and have shown that Generic CDCL is sound, complete and confluent. In contrast to previous work on formalizations of SAT solvers, we can model all recent techniques. The findings add to our understanding of the interplay of inprocessing techniques with the other components of SAT solvers.

The presented formalism provides a small set of rules that might prove to be particularly useful for teaching SAT solving and we believe Generic CDCL deepens the understanding more than presenting pseudo code and algorithm implementations only. As future work, we plan to investigate termination strategies and the question how we can bridge the gap between the implementation and the abstract rules further. In particular, we are interested in describing the clause learning and inference techniques closer to the implementation.

References

[1] Holger Arnold. A linearized DPLL calculus with clause learning. Technical report, Universität Potsdam. Institut für Informatik, 2009.

[2] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning SAT solvers. In Maria Fox and David Poole, editors, *AAAI 2010*, pages 15–20, Menlo Park, California, 2010. AAAI Press.

[3] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. On freezing and reactivating learnt clauses. In Karem A. Sakallah and Laurent Simon, editors, *SAT 2011*, volume 6695 of *LNCS*, pages 188–200, Heidelberg, 2011. Springer.

[4] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009*, pages 399–404, Pasadena, 2009. Morgan Kaufmann Publishers Inc.

[5] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In Giunchiglia and Tacchella [12], pages 341–355.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, Amsterdam, 2009.

[7] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[8] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *STOC 1991*, pages 151–158, Shaker Heights, Ohio, 1971. ACM.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[10] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT 2005*, volume 3569 of *LNCS*, pages 61–75, Heidelberg, 2005. Springer.

[11] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Giunchiglia and Tacchella [12], pages 502–518.

[12] Enrico Giunchiglia and Armando Tacchella, editors. *SAT 2003*, volume 2919 of *LNCS*, Heidelberg, 2004. Springer.

[13] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100, 2000.

[14] P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke. Solving periodic event scheduling problems with SAT. In H. Jiang, W. Ding, M. Ali, and X. Wu, editors, *IEA / AIE 2012*, volume 7345 of *LNCS*, pages 166–175, Heidelberg, 2012. Springer.

[15] Hyojung Han and Fabio Somenzi. On-the-fly clause improvement. In Kullmann [22], pages 209–222.

[16] Marijn Heule and Hans van Maaren. Look-ahead based SAT solvers. In Biere et al. [6], pages 155–184.

[17] S. Hölldobler, N. Manthey, and A. Saptawijaya. Improving resource-unaware SAT solvers. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR 2010*, volume 6397 of *LNCS*, pages 519–534, Heidelberg, 2010. Springer.

[18] Saïd Jabbour, Jerry Lonlac, and Lakhdar Saïs. Adding new bi-asserting clauses for faster search in modern SAT solvers. In Alan M. Frisch and Peter Gregory, editors, *SARA 2013*, pages 66–72, Menlo Park, California, 2013. AAAI.

[19] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *TACAS 2010*, volume 6015 of *LNCS*, pages 129–144, Heidelberg, 2010. Springer.

[20] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 355–370, Heidelberg, 2012. Springer.

[21] Michael Kaufmann and Stephan Kottler. Beyond unit propagation in SAT solving. In Panos M. Pardalos and Steffen Rebennack, editors, *SEA 2011*, volume 6630 of *LNCS*, pages 267–279. Springer, 2011.

[22] Oliver Kullmann, editor. *SAT 2009*, Heidelberg, 2009. Springer.

[23] Inês Lynce and João Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *AAAI 2006*, pages 104–109, Menlo Park, California, 2006. AAAI Press.

[24] Inês Lynce and João P. Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *ICTAI 2003*, pages 105–110, Sacramento, California, USA, 2003. IEEE Computer Society.

[25] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In Armin Biere, Amir Nahir, and Tanja Vos, editors, *HVC 2012*, volume 7857 of *LNCS*, pages 102–117, Heidelberg, 2013. Springer.

[26] Norbert Manthey, Tobias Philipp, and Christoph Wernhard. Soundness of inprocessing in clause sharing SAT solvers. In Matti Järvisalo and Allen Van Gelder, editors, *SAT 2013*, volume 7962 of *LNCS*, pages 22–39, Heidelberg, 2013. Springer.

[27] Filip Marić. Formalization and implementation of modern SAT solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.

[28] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[29] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535, New York, 2001. ACM.

[30] Alexander Nadel and Vadim Ryvchin. Assignment stack shrinking. In Ofer Strichman and Stefan Szeider, editors, *SAT 2010*, volume 6175 of *LNCS*, pages 375–381, Heidelberg, 2010. Springer.

[31] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL modulo theories. In Franz Baader and Andrei Voronkov, editors, *LPAR 2004*, volume 3452 of *LNCS*, pages 36–50, Heidelberg, 2005. Springer.

[32] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to DPLL(T). *JACM*, 53(6):937–977, 2006.

[33] Andrew J. Parkes. Clustering at the phase transition. In Benjamin Kuipers and Bonnie L. Webber, editors, *AAAI 1997, IAAI 1997*, pages 340–345, Menlo Park, California, 1997. AAAI Press / The MIT Press.

[34] Tobias Philipp. Expressive models for parallel satisfiability solvers. Master thesis, Technische Universität Dresden, Informatik Fakultät, 2013.

[35] Knot Pipatsrisawat and Adnan Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In *AI 2008*, pages 1481–1484, Menlo Park, California, 2008. AAAI Press.

[36] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming.* Elsevier Science Inc., New York, 2006.

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

[38] Niklas Sörensson and Armin Biere. Minimizing learned clauses. In Kullmann [22], pages 237–243.

[39] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In Holger H. Hoos and David G. Mitchell, editors, *SAT 2004*, volume 3542 of *LNCS*, pages 276–291, Heidelberg, 2005. Springer.