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. 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 adequately modeled. Finally, we compare Generic CDCL with related systems.

1 Introduction

Many practical problems of computer science are in the complexity class NP. There are many well studied formalisms that can handle problems of this class, among them are constraint satisfaction [20], answer set programming [7], or satisfiability checking [3]. Although the two former formalisms admit a richer language, the latter approach is still very competitive even if the expressivity of its language is comparatively low.

The propositional satisfiability problem (SAT) consists of a propositional formula and asks whether there is a satisfying assignment for the Boolean variables occurring in the formula. From a complexity theory point of view SAT is NP-complete [4] and, thus, intractable. Still, there are many industrial and academic applications that can be solved nicely with modern SAT solvers. For instance a SAT-based railway scheduling software outperformed the native version [9]. Likewise, haplotype matching [13] can be solved nicely with modern SAT solvers.

The success of the SAT approach lies in the strength of today's SAT solvers. SAT solvers do not operate on testing all possible variable assignments, but on constructing 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 it is the core of the basic algorithm known as the DPLL algorithm [5]. In the case that a contradiction is found in the formula with respect to the current variable assignment, advanced 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 [16].

Modern systematic SAT solvers are highly tuned and complex proof procedures employing many advanced techniques like clause learning [16], non-chronological backtracking, restarts [8], clause removal [2,6], decision heuristics [2,17], and formula simplification techniques [12]. Specialized, cache-conscious data structures [10] further improve the 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 the 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.

This problem was tackled by different formalizations, notably LINEARIZED DPLL [1], RULE-BASED SAT SOLVER DESCRIPTIONS [15], and ABSTRACT DPLL [18]. 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.

The contribution of this paper is the formalism Generic CDCL that models 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, as well as clause learning and forgetting. This formalism allows us to reason about the behavior of SAT solvers independently of the specific implementation. Additionally, the framework is a first step to explain how modern SAT solvers are working in a compact and easy way. Besides the presentation of Generic CDCL, the main result of this paper is the proof that Generic CDCL and, consequently, all its instances are sound.

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. Afterwards, we compare Generic CDCL with related formalism in Section 4 and we conclude the paper in Section 5.

¹ The SAT solver Riss is freely available at tools.computational-logic.org.

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 sets of clauses are called formulas, where a clause is a finite set of literals. 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 \bot , the empty formula by \top . The formula obtained from F by replacing all occurrences of the variable v by the variable v is denoted by v. The set of all variables occurring in a formula v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v, then v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive or negative literals) is denoted by v (in positive literals) is denot

The semantics of formulas is based on the notion of an interpretation. An interpretation I is 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 $I \models \top$, $I \not\models \bot$, $I \models (x_1 \lor \ldots \lor x_n)$ iff $I \models x_i$ for some $i \in \{1, \ldots, n\}$, and $I \models (C_1 \land \ldots \land C_n)$ iff $I \models C_i$ for all $i \in \{1, \ldots, n\}$. Interpretation I is a model for the formula F iff $I \models F$. In the case that a formula F has a model, then F is satisfiable, otherwise it is unsatisfiable.

We relate formulas by three relations: the entailment, the equivalence and the equisatisfiability relation: Formula F entails 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 are satisfiable or both are unsatisfiable.

For instance, the interpretation $I = \{x, \neg z\}$ is a model of the formula $F_1 = (x \lor y) \land (\overline{x} \lor \overline{z})$ and, therefore, F is satisfiable. The formula $F_2 = x \land \overline{x}$ has no model and, therefore, is unsatisfiable. The formula $F_3 = x \land \overline{z}$ is satisfiable and, therefore, the formulas F_1 and F_3 are equisatisfiable, but the formulas F_1 and F_2 are not equisatisfiable. In fact, $F_3 \models F_1$ since every total model I of the formula F_3 must contain x and \overline{z} and, hence, the two clauses of F_1 are satisfied by I. Finally, we find for all clauses C and formulas F that $C \lor T \equiv T \lor C \equiv T$, $C \lor \bot \equiv \bot \lor C \equiv C$, $F \land T \equiv T \land F \equiv F$, and $F \land \bot \equiv \bot \land F \equiv \bot$.

Let x be a literal, $C = (x \vee C')$ and $D = (\overline{x} \vee D')$ be two clauses. Then the clause $(C' \vee D')$ 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 $i \in \{2, \ldots, n-1\}$; one 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 it 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_1J_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. 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 $F|_J := \{C|_J \mid C \in F \text{ and for every literal } x \in C \text{ we find that } x \not\in J\}$, 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 assignment. For instance, let $F = \{\{\overline{x},y\},\{z\}\},$ then $F|_x = \{\{y\},\{z\}\},$ $F|_{\overline{z}} = \{\{\overline{x},y\},\bot\}$ and $F|_{yz} = \top$, where the interpretations are written as sequences of literals. One should observe that the reduct operator does not distinguish between propagation and decision literals.

Lemma 1 below summarizes the properties of the reduct operator: (i) The reduct is monotone. (ii) A formula F is satisfiable iff there exists an interpretation J such that the reduct of a formula w.r.t. J is the empty formula. (iii) If the reduct of a formula F w.r.t. the interpretation $\{x\}$ is unsatisfiable, and the formula F entails the literal x, then the formula F is unsatisfiable. (iv) The reduct operator is a *semantic operator* in the sense that it cannot distinguish equivalent formulas.

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

- (i) $F|_{J} \subseteq (F \cup F')|_{J}$ for every interpretation J.
- (ii) F is satisfiable iff there exists a J such that $F|_{J} = \top$.
- (iii) If $F|_x$ is unsatisfiable and $F \models x$, then F is unsatisfiable.
- (iv) If $F \equiv F'$, then $F|_{J} \equiv F'|_{J}$ for every interpretation J.

Proof. See [19, pp.10–12].

3 Generic CDCL

Modern SAT solvers are based on the linearized DPLL [5] algorithm and consists of the following components: termination criteria, a decision component that picks the branching literals, an inference component that adds propagation literals to the working sequence of literals, a backtracking component that rolls back wrong decisions and a formula management component that simplifies the working formula. We maintain two data structures during the execution of modern SAT solvers: the working formula, and the working set of literals. Together they define the state. The components are modelled as a transition relation over the set of states; the union of the rules in Fig. 1 is then the transition relation

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

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

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

INF-rule: $F \ /\!\!/ J \sim_{\mathsf{INF}} F \ /\!\!/ J x$ iff $F|_J \equiv_{\mathsf{sat}} F|_{Jx}$ and $\{x, \overline{x}\} \cap J = \emptyset$.

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

FORGET-rule: $F \ /\!\!/ J \sim_{\mathsf{FORGET}} F \setminus \{C\} \ /\!\!/ J$ iff $F \setminus \{C\} \models C$.

BACK-rule: $F \ /\!\!/ J J' \sim_{\mathsf{BACK}} F \ /\!\!/ J$.

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

Fig. 1: Transition relations of Generic CDCL. These relations apply to all formulas F and F', clauses C, literals x and lists of literals J and J'. ε denotes the empty sequence of literals.

of Generic CDCL: Formally, we model the computation of modern SAT solvers by means of state transition systems as follows:

Definition 2 (Generic CDCL). Generic CDCL is a state transition system whose sets of states is

```
\{F \ /\!\!/ \ J \ | \ F \ is \ a \ formula \ and \ J \ is \ a \ sequence \ of \ literals \} \cup \{SAT, UNSAT\},
```

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

```
\sim := {\sim}_{SAT}, \sim_{UNSAT}, \sim_{DEC}, \sim_{INF}, \sim_{LEARN}, \sim_{FORGET}, \sim_{BACK}, \sim_{INP} }.
```

The SAT-rule terminates the computation with the output SAT, if the reduct of the working formula w.r.t. the working set of literals 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 (ii) the working formula is then satisfiable.

The UNSAT-rule terminates the computation with the output UNSAT, if no model of the working formula exists. This is the case when a conflict occurs in the top level, i.e. $\bot \in F|_J$ and the sequence J of literals contains only propagation literals. These conditions can be decided in polynomial time.

The DEC-rule extends the working sequence of literals by an unassigned literal \dot{x} as a decision literal.

The INF-rule extends the working list of literals by a propagation literal x, if the reducts of the working formula w.r.t. the working sequence of literals and its extension are equisatisfiable.

The BACK-rule models backtracking, as well as backjumping and restarts, by deleting outermost right literals in the working sequence of literals.

The LEARN-rule adds a clause C to the working formula, if it 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.

The FORGET-rule deletes a 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 [12].

The INP-rule models formula simplifications that are used in pre- and inprocessing. It replaces the working formula with an equisatisfiable formula when the working sequence of literals is empty.

Let $\overset{*}{\sim}$ be the reflexive and transitive closure of \leadsto . Let $x\overset{0}{\leadsto} x$ for all states x, and $x\overset{n}{\leadsto} z$ for all natural numbers $n\in\mathbb{N}$ if and only if there exists a state y such that $x\overset{n-1}{\leadsto} y\leadsto z$. In the next subsection we investigate the question whether Generic CDCL correctly solves the SAT problem. Formally, we define Generic CDCL to be *sound* iff for all formulas F_0 we have that $\inf(F_0)\overset{*}{\leadsto} \mathsf{SAT}$ implies that F_0 is satisfiable and $\inf(F_0)\overset{*}{\leadsto} \mathsf{UNSAT}$ implies that F_0 is unsatisfiable.

3.1 Generic CDCL is Sound

Before proceeding to the soundness proof of Generic CDCL, it will be necessary to study two invariants of Generic CDCL that are presented in the proposition below: (i) states that the rules of Generic CDCL do not change the satisfiability of the working formula, and (ii) states that whenever the working sequence of literals is of the form $J_1 x J_2$, where x is a propagation literal, the reducts of the working formula w.r.t. J_1 and $J_1 x$ are equisatisfiable.

Proposition 3 (Invariants). Let F_0 , F be formulas, J be a sequence of literals, and $n \in \mathbb{N}$. If $\operatorname{init}(F_0) \stackrel{n}{\leadsto} F /\!\!/ J$, 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 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 $F/\!\!/J$ and suppose that $F/\!\!/J \sim_{\mathsf{R}} F'/\!\!/J'$ where $\mathsf{R} \in \{\mathsf{DEC}, \mathsf{INF}, \mathsf{LEARN}, \mathsf{FORGET}, \mathsf{BACK}, \mathsf{INP}\}$:

- 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, $F'|_{J'_1} \equiv_{\mathsf{sat}} F'|_{J_1 y}$.

 BACK-rule: In this case, F' = F and J = J' J''. 1. follows since $F_0 \equiv_{\mathsf{sat}} F$ by
- BACK-rule: In this case, F' = F and J = J' J". 1. follows since F₀ ≡_{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} ≡_{sat} F|_{J'₁y}, and consequently we know that F'|_{J'₁} ≡_{sat} F'|_{J'₁y}.
 LEARN-rule: In this case, F' = F ∪ {C} where F ⊨ C and J' = J. 1. follows
- LEARN-rule: In this case, $F' = F \cup \{C\}$ where $F \models C$ and J' = J. 1. follows since $F \equiv F'$ and the addition of the entailed clause C preserves equivalence of the working formula. 2. follows from the reduct operator being a semantic operator by Lemma 1.iv 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 $J' = J'_1 y J'_2$.
- FORGET-rule: This case can be treated as in the LEARN-rule.
- INF-rule: In this case, F' = F and J' = Jx 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.
- INP-rule: In this case, $F' \equiv_{\sf sat} F$ 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 main theorem in this paper.

Theorem 4 (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

$$\operatorname{init}(F_0) \stackrel{*}{\sim} F /\!\!/ J \sim \operatorname{SAT}(\operatorname{UNSAT}, \operatorname{resp.}).$$

SAT By the definition of the SAT-rule, we know that $F|_J = \top$. By Lemma 1.ii, 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 given in Prop. 3(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 sequence of literals $J = (x_1 \dots x_n)$ contains only propagation literals. Since a conflict occurs, $F|_J$ is unsatisfiable. From the result that the formula $F|_J$ is unsatisfiable and the literal sequence J contains only propagation literals we can repeatably apply Prop. 3(2.) and we obtain that the formula F is unsatisfiable. Since the formula F is unsatisfiable and the formulas F and F_0 are equisatisfiable by Prop. 3(1.), we conclude that F_0 is unsatisfiable.

3.2 Generic CDCL Subsumes Important SAT Solving Techniques

We now describe some important 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 $C \subset D$. In this case, D can be deleted because $F \setminus \{D\} \models D$. Consequently, $F /\!\!/ J \sim_{\mathsf{FORGET}} F \setminus \{D\} /\!\!/ J$ holds for every literal sequence J. Removing subsumed clauses is done as a preprocessing step in SAT solvers and during clause learning.

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

Conflict-Directed Backtracking and Learning [21] is an improvement of naive backtracking that takes the reason of the conflict into account. Consider the state $F \not\parallel J\dot{x}J'$ 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 $F \not\parallel J\dot{x}J'$ into the state $F \cup \{C\} \not\parallel Jy$. Conflict-directed backtracking and learning can be simulated by the following transition steps: $F \not\parallel J\dot{x}J' \leadsto_{\mathsf{BACK}} F \not\parallel J \leadsto_{\mathsf{LEARN}} F \cup \{D\} \not\parallel J \leadsto_{\mathsf{INF}} F \cup \{D\} \not\parallel Jy$.

Blocked Clause Elimination [11] 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 $D \in F$ upon x are tautological. Blocked clauses are removed from a formula during preand 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.

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 resolution.

Pure Literal A literal x is pure in the formula F, if $\overline{x} \notin \text{lits}(F)$. 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 literal sequence: $F \not\parallel J \sim_{\mathsf{INF}} F \not\parallel J x$.

4 Related Work

Several attempts have been made to formalize sequential SAT solvers in terms of transition systems or proofs calculi: Abstract DPLL [18], Linearized DPLL [1], and Rule-based SAT solver description [15]. In general, these formalization are based on a notion of state like 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 [15] do not model formula simplifications that changes the semantics of formulas like blocked clause elimination. Maric highlighted the implementation of clause learning techniques in his Rule-based SAT solver description [15], 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. In contrast, Generic CDCL subsumes all recent SAT techniques to the best of our knowledge.

In [12] 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. They drew our attention to the interplay of the learned clause database with inprocessing techniques. The interplay of clause sharing and formula simplification techniques in *parallel* SAT solvers was analyzed in [14], 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.

5 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 advaned techniques. Therefore, it is desireable 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. 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.

A limitation of Generic CDCL is its lack of details in the learning and inference component. As future work, we plan to investigate properties such as completeness, confluence and termination.

References

1. Arnold, H.: A linearized DPLL calculus with clause learning. Tech. rep., Universität Potsdam. Institut für Informatik (2009)

 Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009. pp. 399–404. Morgan Kaufmann Publishers Inc., Pasadena (2009)

- 3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press, Amsterdam (2009)
- 4. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) STOC 1991. pp. 151–158. ACM (1971)
- Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
- 6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
- Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) ICLP 1988. pp. 1070–1080. MIT Press (1988)
- 8. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. 24(1–2), 67–100 (2000)
- 9. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving periodic event scheduling problems with SAT. In: Jiang, H., Ding, W., Ali, M., Wu, X. (eds.) IEA / AIE 2012. LNCS, vol. 7345, pp. 166–175. Springer, Heidelberg (2012)
- 10. Hölldobler, S., Manthey, N., Saptawijaya, A.: Improving resource-unaware SAT solvers. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 519–534. Springer, Heidelberg (2010)
- 11. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
- Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
- 13. Lynce, I., Marques-Silva, J.: Efficient haplotype inference with Boolean satisfiability. In: AAAI 2006. pp. 104–109. AAAI Press (2006)
- 14. Manthey, N., Philipp, T., Wernhard, C.: Soundness of inprocessing in clause sharing SAT solvers. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 22–39. Springer, Heidelberg (2013)
- 15. Marić, F.: Formalization and implementation of modern SAT solvers. J. Autom. Reason. 43(1), 81-119 (2009)
- 16. Marques Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5), 506–521 (1999)
- 17. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC 2001. pp. 530–535. ACM, New York (2001)
- 18. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract DPLL and abstract DPLL modulo theories. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS, vol. 3452, pp. 36–50. Springer, Heidelberg (2005)
- 19. Philipp, T.: Expressive Models for Parallel Satisfiability Solvers. Master thesis, Technische Universität Dresden (2013)
- 20. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming. Elsevier Science Inc., New York (2006)
- Silva, J.P.M., Sakallah, K.A.: GRASP a new search algorithm for satisfiability.
 In: ICCAD 1996. pp. 220–227. IEEE Computer Society, Washington (1996)