

# Clause Simplifications in Search-Space Decomposition-Based SAT Solvers

Tobias PHILIPP

*International Center for Computational Logic*

*Technische Universität Dresden, Germany*

**Abstract.** Inprocessing is to apply clause simplification techniques during search and is a very attractive idea since it allows to apply computationally expensive techniques. In this paper we present the search space decomposition formalism SSD that models parallel SAT solvers with clause sharing and inprocessing. The main result of this paper is that the sharing model SSD is sound. In the formalism, we combine clause addition and clause elimination techniques, and it covers many SAT solvers such as PaSAT, PaMira, PMSat, MiraXT and ccc. Inprocessing is not used in these solvers, and we therefore propose a novel way how to combine clause sharing, search space decomposition and inprocessing.

**Keywords.** SAT, guiding paths, inprocessing, blocked clause elimination

## 1. Introduction

The satisfiability problem (SAT) is one of the most prominent problems in theoretical computer science and has many applications in software verification [6], planning [25, 36], bioinformatics [27] or scheduling [13]. Modern SAT solvers based on the DPLL algorithm [7] use many advanced techniques like *clause learning* [31], *non-chronological backtracking* [40], *restarts* [12], *clause removal* [3, 10], *preprocessing* [9, 28], *inprocessing* [5, 23, 39], efficient *data structures* [21, 33] and advanced *decision heuristics* [2, 3, 19, 33]. These improvements in the last decades led to a spectacular performance of conflict-driven satisfiability solvers. The *search space decomposition* approach [43] was introduced in 1996, and is a promising approach to solve *hard instances* (see [18, 32, 38] for an overview of parallel SAT solving). In particular, CCC [42] is based on this approach. The *search space*, i.e. the set of interpretations, is decomposed into different spaces by *guiding paths* [43], that are then explored in parallel by modern sequential SAT solvers. This means that the solvers compete in finding a model of the formula, and cooperate in proving its unsatisfiability.

Combining clause sharing with inprocessing in a parallel setting can make a formula unsatisfiable, as the following example demonstrates [29]: Consider the situation in which we have the two solvers *Solver*<sub>1</sub> and *Solver*<sub>2</sub> and the input formula  $F_0$  consisting of the single unit clause ( $x$ ). Then, *Solver*<sub>1</sub> can rewrite  $F_0$  to its negation because this operation preserves satisfiability. Suppose that *Solver*<sub>2</sub> imports the negated formula, then it can terminate with the answer UNSAT because the conjunction of  $F_0$  and its negation

is unsatisfiable, but the input formula  $F_0$  is satisfiable. Therefore, inprocessing must be restricted if we allow clause sharing without any restriction.

To understand the interplay of advanced techniques and to reason about modern SAT solvers, we propose to formalize parallel SAT solvers. Indeed, abstracting the specific solver implementation is important since modern parallel SAT solvers consist of multiple thousand lines of code and are written in programming languages with side effects like C or C++. The formalisms presented in [1, 30, 34] models clause learning sequential SAT solvers, but they are inadequate to express advanced features like restarts, preprocessing, inprocessing and the ability to share clauses. But, *Generic CDCL* [20] can handle the above techniques. A formalization in [29] was introduced, that models portfolio solvers with inprocessing, and it was argued that this can be extended to guiding path solvers.

The contribution of this paper is the formal system SSD that models the computation of search-space decomposition based SAT solvers with clause sharing and inprocessing. It extends the formalism presented in [29] and we formally prove soundness of this system. We combine clause addition techniques and clause elimination techniques.

This paper is structured as follows: We start with basic notions, parallel satisfiability testing and clause simplifications in Sect. 2. Afterwards, we study some properties of guiding paths in Sect. 3, and present the sharing model SSD in Sect. 4. Finally in Sect. 5 we conclude.

## 2. Background

### 2.1. Propositional Logic

We assume a fixed infinite set  $\mathcal{V}$  of Boolean *variables*. A *literal* is a variable  $v$  (*positive literal*) or a negated variable  $\bar{v}$  (*negative literal*). The *complement*  $\bar{x}$  of a positive (negative, resp.) literal  $x$  is the negative (positive, resp.) literal with the same variable as  $x$ . The set of all literals is denoted by  $\mathcal{L}$ . For a set of literals  $J$  the complement of  $J$ , denoted by  $\bar{J}$ , is defined as  $\bar{J} = \{\bar{x} \mid x \in J\}$ , and  $J$  is *consistent* if and only if  $\{x, \bar{x}\} \not\subseteq J$  for every literal  $x \in \mathcal{L}$ . In SAT, we deal with finite clause sets called *formulas*. Each *clause*  $C$  is a finite set of literals. We write a clause  $\{x_1, \dots, x_n\}$  also as disjunction  $(x_1 \vee \dots \vee x_n)$  and a formula  $\{C_1, \dots, C_n\}$  as a conjunction  $(C_1 \wedge \dots \wedge C_n)$ , where the empty clause is denoted by  $\perp$ .

The formula  $F$  after substituting all occurrences of the variable  $v$  with the variable  $w$  is denoted by  $F[v \mapsto w]$ . The set of all variables occurring in a formula  $F$  (in positive or negative literals) is denoted by  $\text{vars}(F)$ . For a formula  $F$  and literal  $x$ , the formula consisting of all clauses in the formula  $F$  that contain the literal  $x$  is denoted by  $F_x$ .

The semantic of formulas is built on interpretations. An *interpretation*  $I$  is a set of literals that contains for all variables  $v$  exactly one of  $v$  or  $\bar{v}$ , and can be understood as a mapping from the set  $\mathcal{V}$  of all Boolean variables to the set  $\{\top, \perp\}$  of truth values. The interpretation  $I$  *satisfies* the literal  $x$ , in symbols,  $I \models x$ , if and only if  $x \in I$ . It *satisfies* the clause  $C$ , in symbols  $I \models C$ , if and only if there is a literal  $x \in C$  such that  $I \models x$ . For a formula  $F$ , the interpretation  $I$  *satisfies* the formula  $F$ , in symbols  $I \models F$ , if and only if the interpretation  $I$  satisfies the clause  $C$  for every clause  $C \in F$ . A *model*  $I$  of a formula  $F$  is an interpretation  $I$  that satisfies the formula  $F$ . If there is such an interpretation  $I$ , the formula  $F$  is *satisfiable*. Otherwise, the formula  $F$  is *unsatisfiable*. The main ques-

tion can be expressed as follows: Given a formula  $F$ , the *Satisfiability Problem (SAT)* asks whether the formula  $F$  is satisfiable. Two formulas  $F$  and  $F'$  are *equisatisfiable*, in symbols  $F \equiv_{\text{sat}} F'$ , if and only if either both are satisfiable or both are unsatisfiable. In this paper, we relate formulas by the *entailment relation*: The formula  $F$  *entails* the formula  $F'$  if and only if every model of the formula  $F$  is a model of the formula  $F'$ . This definition of entailment has two beneficial properties: transitivity and monotonicity. The drawback is that some formula simplification rules must be treated differently. Two formulas  $F$  and  $F'$  are *equivalent*, in symbols  $F \equiv F'$ , if and only if the formula  $F$  entails the formula  $F'$  and vice versa. The formula  $F'$  is an *unsatisfiability-preserving consequence* of a formula  $F$  if and only if  $F \models F'$ , and if the formula  $F$  is unsatisfiable, then the formula  $F'$  is unsatisfiable.

The *reduct of the formula  $F$  w.r.t. the consistent set of literals  $J$* , in symbols  $\text{reduct}(F, J)$  is a formula obtained in two steps: First, we remove every clause from the formula  $F$  that contains a literal from  $J$ , and in the second step, we remove each literal in the remaining clauses such that the complement of the literal is contained in  $J$ .

## 2.2. Formula Simplifications

Simplifying the formula before giving it to the SAT solver has become an important part of the solving chain [28]. The following techniques were proposed among many others.

*Subsumption Elimination* The clause  $C$  *subsumes* the clause  $D$  if and only if  $C \subset D$ . Given an input formula  $(F \wedge C \wedge D)$ , where the clause  $C$  subsumes the clause  $D$ , the subsumption elimination technique produces the formula  $F' = (F \wedge C)$ .

*Hyper Binary Resolution* [4] Given an input formula  $F$  with  $(y \vee x_1 \vee x_2 \vee \dots \vee x_n) \in F$  and  $(\bar{x}_i \vee z) \in F$  for  $i \in \{1, \dots, n\}$ . Then, the clause  $(y \vee z)$  is a *hyper binary resolvent* w.r.t. the formula  $F$ , and the hyper binary resolutions technique adds a hyper binary resolvent, i.e. the result of applying hyper binary resolution is the formula  $F' = F \wedge (y \vee z)$ .

*Variable Elimination* [9, 41] 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  $\bar{v}$ . Formally, the result of applying variable elimination is the formula  $F' = (F \cup (F_v \otimes_v F_{\bar{v}})) \setminus (F_v \cup F_{\bar{v}})$ .

*Equivalence Elimination* [17] Two variables  $v$  and  $w$  are equivalent in the formula  $F$ , if the formula  $F$  entails the equivalence  $v \leftrightarrow w$ . Equivalence elimination produces the formula  $F' = F[v \mapsto w]$ , if the variables  $v$  and  $w$  are equivalent in the formula  $F$ .

*Blocked Clause Elimination* [22] A clause  $C$  is called *blocked* in the formula  $F$ , if it contains a *blocked* literal  $x$  such that for all clauses  $D \in F_{\bar{x}}$  the resolvent of  $C$  and  $D$  upon the variable of  $x$  is a tautology. The result of applying blocked clause elimination in the input formula  $(F \wedge C)$ , where the clause  $C$  is blocked in the formula  $F$ , is the formula  $F' = F$ .

Note that the above techniques produce unsatisfiability-preserving consequences (see [29]). Unfortunately, not all formula simplifications, like the reverse operation of blocked clause elimination [22], generate unsatisfiability-preserving consequences.

### 2.3. Parallel Satisfiability Solvers

*Search space decomposition* is an approach that distributes the search space among multiple SAT solvers that explore the assigned search spaces in parallel. The *search space* of a formula  $F$ , in symbols  $\text{SearchSpace}(F)$ , is the set of all interpretations and the *solution space* of a formula  $F$ , denoted by  $\text{SolutionSpace}(F)$ , is the set of all models of the formula  $F$ . Parallel solvers based on the search space decomposition approach use parallel architecture by searching in these spaces in parallel. This means that if a solver incarnation finds a model of the formula in its assigned search space, the parallel solver immediately terminates the computation and outputs SAT. Therefore, the solvers are competing w.r.t. the SAT answer. On the other hand, the solvers cooperate on finding a witness for the unsatisfiability of the formula. That means, the computation terminates with the output UNSAT, if all solver incarnations proved that no model in the corresponding search spaces exists. The dominant approach to divide the search space are *guiding paths* [43]. Guiding paths divide the search space by a conjunction of literals, and were first implemented in the parallel SAT solver PSATO [43]. A recent solver also applies *look-ahead techniques* to determine the guiding path (e.g. [42]). *Clause sharing* can be implemented in these solvers, as done in the SAT solvers PASAT [24], PAMIRA [37], PMSAT [11]. The SAT solver MIRAXT [26] additionally performs probing-based formula simplifications during search.

### 2.4. State Transition Systems

State transition systems are well-understood, formal descriptions of algorithms in terms of states and binary relations over states. Formally, a *state transition system* is a tuple  $(\Delta, \rightsquigarrow)$  where  $\Delta$  is the set of *states* and  $\rightsquigarrow \subseteq \Delta \times \Delta$  is the *state transition relation*. Given a system  $(\Delta, \rightsquigarrow)$ , we define  $\rightsquigarrow^0 = \{(x, x) \mid x \in \Delta\}$ ,  $\rightsquigarrow^n = \{(x, z) \mid (x, y) \in \rightsquigarrow^{n-1} \text{ and } (y, z) \in \rightsquigarrow\}$  for all  $n \in \mathbb{N}_{>0}$  and  $\rightsquigarrow^* = \bigcup_{i \in \mathbb{N}} \rightsquigarrow^i$ . We write  $x \rightsquigarrow y$  instead of  $(x, y) \in \rightsquigarrow$ .

## 3. Guiding Paths

We start with a formal definition of guiding paths.

**Definition 1** (Guiding Paths). A guiding path  $P$  is a finite set of non-complementary literals. A set  $\mathcal{P}$  of guiding paths is complete if and only if for every interpretation  $I$  we find a guiding path  $P \in \mathcal{P}$  such that  $I \models x$  for every literal  $x \in P$ . The search space of the guiding path  $P$  is  $\text{SearchSpace}(P) = \{I \mid I \models x \text{ for all literals } x \in P\}$ .

The definitions here are slightly different than in [43]: In this paper a guiding path does not include information whether its assigned search space contains a model, and its assigned search space is not part of the search tree of the DPLL procedure [7, 8], but a set of interpretations.

Lemma 1 presents properties of guiding paths and the interplay of the reduct operator, guiding paths and search spaces: A complete set of guiding paths is always non-empty (Lemma 1.1), the trivial set of guiding paths  $\{\emptyset\}$  is complete (Lemma 1.2). Lemma 1.3 relates complete sets of guiding paths and the search space: The union over all guiding paths of a complete set of guiding paths over the search space of  $P$  is equal

to the search space. Lemma 1.4 relates the reduct operator with the search space: The interpretation  $I$  is a model of a formula  $F$  and belongs to the search space that is defined by the guiding path  $P$  if and only if the formula  $\text{reduct}(F, P)$  is satisfiable. Moreover, the formula  $F$  is satisfiable if and only if for some complete set of guiding paths  $\mathcal{P}$  we find a guiding path  $P \in \mathcal{P}$  where  $\text{reduct}(F, P)$  is satisfiable (Lemma 1.5). The reduct operator is *compatible* with the entailment relation (Lemma 1.6).

**Lemma 1.** *Let  $F$  be a formula,  $I$  be an interpretation and  $\mathcal{P}$  a set of guiding paths.*

1. *A complete set of guiding paths is non-empty.*
2.  *$\{\emptyset\}$  is a complete set of guiding paths.*
3.  *$\mathcal{P}$  is complete iff  $\bigcup_{P \in \mathcal{P}} \text{SearchSpace}(P) = \text{SearchSpace}(F)$ .*
4. *There is  $I$  with  $I \models F$  and  $I \in \text{SearchSpace}(P)$  if and only if  $\text{reduct}(F, P)$  is satisfiable.*
5. *Let  $\mathcal{P}$  be complete. Then  $F$  is satisfiable if and only if there is a guiding path  $P \in \mathcal{P}$  such that  $\text{reduct}(F, P)$  is satisfiable.*
6. *If  $F \models F'$ , then  $\text{reduct}(F, J) \models \text{reduct}(F', J)$ .*

*Proof.* See [35]. □

#### 4. The Search Space Decomposition Model SSD

We model the computation of SAT solvers based on the search space decomposition approach by means of a state transition system as follows: A state of computation of a single sequential solver is a formula  $F$  and guiding path  $P$ , represented as the pair  $(F, P)$ . To ease our notation, we represent sequential solver states as  $F$  and say that  $P$  is the *associated guiding path*. A state of computation of a parallel SAT solver based on the search space decomposition approach of  $n$  sequential solver  $Solver_1, \dots, Solver_n$  is then an  $n$ -tuple  $(F_1, \dots, F_n)$  of formulas where  $F_i$  is the state of  $Solver_i$  and  $P_i$  is the guiding path that is assigned to  $Solver_i$ . Then, the *set of states in SSD with multiplicity n* is  $\{(F_1, \dots, F_n) \mid F_i \text{ is a formula}\} \cup \{\text{SAT}, \text{UNSAT}\}$  and we associate a guiding path  $P_i$  to the formula  $F_i$  for every  $i \in \{1, \dots, n\}$ . The *initial state in SSD for the input formula  $F_0$  with multiplicity n and the complete set of guiding paths  $\{P_1, \dots, P_n\}$* , denoted by  $\text{init}_{n, P_1, \dots, P_n}(F_0)$ , is the tuple  $(F_1, \dots, F_n)$  where the guiding path  $P_i$  is associated with the formula  $F_i$  for all  $i \in \{1, \dots, n\}$ . The sharing model SSD is characterized by the following transition relation:

$$\rightsquigarrow_{\text{SSD}} := \rightsquigarrow_{\text{SAT}} \cup \rightsquigarrow_{\text{UNSAT}} \cup \rightsquigarrow_{\text{CM}} \cup \rightsquigarrow_{\text{SHARE}} \cup \rightsquigarrow_{\text{ADD}} \cup \rightsquigarrow_{\text{DEL}} .$$

Corresponding transition rules are given in Figure 1. We have two termination rules: The SAT-rule terminates the computation in the final state SAT if and only if  $\text{reduct}(F_i, P_i)$  is satisfiable for some  $i \in \{1, \dots, n\}$ . By Lemma 1.4 this situation happens if and only if a model of the formula  $F_i$  exists that is contained in the assigned search space of the  $i$ 'th solver. Likewise, the UNSAT-rule terminates the computation in the final state UNSAT if and only if the formula  $\text{reduct}(F_i, P_i)$  is unsatisfiable for every  $i \in \{1, \dots, n\}$ . Again by Lemma 1.4 this situation happens if and only if no model of the formula  $F_i$  exists that is contained in the search space of the  $i$ 'th solver for all  $i \in \{1, \dots, n\}$ . Modern complete solvers are modifying the formula by adding and removing learned clauses. Such clause

- SAT-rule:  $(F_1, \dots, F_n) \rightsquigarrow_{\text{SAT}} \text{SAT}$   
iff  $\text{reduct}(F_i, P_i)$  is satisfiable for some  $i \in \{1, \dots, n\}$ .
- UNSAT-rule:  $(F_1, \dots, F_n) \rightsquigarrow_{\text{UNSAT}} \text{UNSAT}$   
for  $\text{reduct}(F_i, P_i)$  is unsatisfiable for all  $i \in \{1, \dots, n\}$ .
- CM-rule:  $(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\text{CM}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$   
iff  $F_i \equiv F'_i$ .
- SHARE-rule:  $(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\text{SHARE}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$   
iff  $F'_i = F_i \wedge C$ , and  $C \in F_j$  for some  $j \in \{1, \dots, n\} \setminus \{i\}$ .
- ADD-rule:  $(F_1, \dots, F_n) \rightsquigarrow_{\text{ADD}} (F_1 \wedge C, F_2, \dots, F_n)$   
iff  $\text{vars}(C) \subseteq \text{vars}(F_0)$ , and  $F_1 \wedge C \equiv_{\text{sat}} F_1$ .
- DEL-rule:  $(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\text{DEL}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$   
iff  $i > 1$  and  $F'_i$  is an unsatisfiability-preserving consequence of  $F_i$ .

**Figure 1.** Transition relations used to characterize search space decomposition based SAT solvers by means of portfolio systems with input formula  $F_0$  and multiplicity  $n$ . These definitions apply to all formulas  $F_1, \dots, F_n, F'_1, \dots, F'_n$ , clauses  $C$  and  $i \in \{1, \dots, n\}$ , and guiding paths  $P_i$  where  $P_i$  is the guiding path that is assigned to the solver incarnations  $\text{Solver}_i$ .

management is modeled by the CM-rule that rewrites a formula  $F_i$  of a solver incarnation into an equivalent formula  $F'_i$ . Finally, clause sharing is captured by the SHARE-rule that adds a clause  $C$  from the formula  $F_j$  to the formula  $F_i$ , where  $i \neq j$ . The ADD-rule adds a clause  $C$  to the formula  $F_1$ , if  $F_1$  and  $F_1 \wedge C$  are equisatisfiable and  $\text{vars}(C) \subseteq \text{vars}(F_0)$ . On the other hand, clause deletion is captured by the DEL-rule, which replaces a formula  $F_i$  with a formula  $F'_i$ , if the  $F'_i$  is an unsatisfiability-preserving consequence of  $F_i$ , and  $i > 0$ .

We say that the formalism SSD is *sound* if and only if for all formulas  $F_0$  and complete set of guiding paths  $\{P_1, \dots, P_n\}$  we have that  $\text{init}_{n, P_1, \dots, P_n}(F_0) \xrightarrow{*} \text{SAT}$  implies that the formula  $F_0$  is satisfiable and  $\text{init}_{n, P_1, \dots, P_n}(F_0) \xrightarrow{*} \text{UNSAT}$  implies that the formula  $F_0$  is unsatisfiable. Intuitively, soundness means that every answer in the system is correct. Before we prove soundness of SSD, we establish the following invariants:

**Proposition 1.** Let  $n \geq 1$ , let  $F_0, F_1, \dots, F_n$  be formulas,  $P_1, \dots, P_n$  be guiding paths, and let  $m \geq 0$ . Assume  $\text{init}_{n, P_1, \dots, P_n}(F_0) \xrightarrow{m} (F_1, \dots, F_n)$ . Then the following properties hold:

1.  $F_1 \models F_2 \wedge \dots \wedge F_n$ , and
2.  $F_i \equiv_{\text{sat}} F_0$  for all  $i \in \{1, \dots, n\}$ .

*Proof.* See [29] □

We then can prove the following theorem, stating that the computed answers of the sharing model SSD are sound:

**Theorem 1.** The search space decomposition model SSD is sound.

*Proof.* We divide the proof in two parts, first proving that the output SAT is correct, then proving that the output UNSAT is correct. Let  $n > 0$ , and  $\{P_1, \dots, P_n\}$  be a complete set of guiding paths. Suppose  $\text{init}_{n, P_1, \dots, P_n}(F_0) \xrightarrow{*_{\text{SSD}}} (F_1, \dots, F_n) \xrightarrow{\sim_{\text{SSD}}} \text{SAT}(\text{UNSAT}, \text{resp.})$ .

1. SAT: In this case  $\text{reduct}(F_i, P_i)$  is satisfiable for some  $i \in \{1, \dots, n\}$ . Clearly, this observation implies that the formula  $F_i$  is satisfiable. By Proposition 1, we know that the formulas  $F_i$  and  $F_0$  are equisatisfiable. Then the input formula  $F_0$  is satisfiable and consequently the SAT answer is correct.
2. UNSAT: The proof is by contradiction: Suppose that the answer is incorrect, i.e. the input formula  $F_0$  is satisfiable. We know by Proposition 1.2 that the formula  $F_1$  is satisfiable. Since  $\{P_1, \dots, P_n\}$  is complete, we know by Lemma 1.5 that  $\text{reduct}(F_1, P_j)$  is satisfiable for some  $j \in \{1, \dots, n\}$ . Hence, there is an interpretation  $I$  such that  $I \models \text{reduct}(F_1, P_j)$ . By Proposition 1.1 we know that  $F_1 \models F_2 \wedge \dots \wedge F_n$  and conclude by Lemma 1.6 that  $I \models \text{reduct}(F_2 \wedge \dots \wedge F_n, P_j)$ . Since  $\text{reduct}(F_2 \wedge \dots \wedge F_n, P_j) \models \text{reduct}(F_i, P_j)$  for every  $i \in \{2, \dots, n\}$ , we know that  $I \models \text{reduct}(F_i, P_j)$  for every  $i \in \{2, \dots, n\}$ . Hence  $\text{reduct}(F_i, P_j)$  is satisfiable for every  $i \in \{1, \dots, n\}$ . Then  $\text{reduct}(F_j, P_j)$  is satisfiable, but the UNSAT-rule is not applicable, which is a contradiction. Therefore the input formula  $F_0$  is unsatisfiable, and consequently the UNSAT answer is correct.  $\square$

In particular, Theorem 1 states that the particular setting of clause sharing, clause addition and elimination techniques together with search space decomposition with guiding paths is sound.

## 5. Conclusion

Parallel SAT solvers employ many advanced techniques, but not every combination of advanced technique is sound. In particular, the combination of clause sharing and inprocessing can make a formula unsatisfiable.

Consequently, a SAT solver can incorrectly report the unsatisfiability of a formula. Fuzzing is a technique that allows SAT solver engineers to test and empirically verify combinations of techniques. One of the reasons for this methodology is the lack of tools to study such advanced combinations. Moreover, useful combinations that involve complicated constraints might be missed with the purely experimental approach. We therefore propose to formalize modern parallel SAT solvers to understand the interplay of advanced techniques, and to reason about them. In this paper, we developed a formal model in terms of state transition systems.

Search space partitioning is an early approach to use parallel hardware architectures. Guiding paths are a simple method to divide the search space among the solver incarnations. We studied guiding paths and their relation to search spaces, and afterwards developed the sharing model SSD that models parallel SAT solvers based on the guiding paths with clause sharing and inprocessing. The presented formalism covers many SAT solvers such as PASAT [24], PAMIRA [37], PMSAT [11], MIRAXT [26]. We have shown that the formalism SSD, that combines clause addition and clause elimination techniques as in [29], is sound. Therefore, the solvers can be improved by inprocessing techniques, which is a new result to the best of our knowledge.

Moreover, we indicate possibilities to improve modern parallel SAT solvers by new combinations of inprocessing and clause sharing. It is our conviction that the formalism SSD is one part of the tools desired for developing parallel SAT solvers with new combinations of advanced techniques.

As future work, we identify two interesting challenges: Biere combines restricted forms of blocked clause elimination and addition in the upcoming version of PLINGELING, and has shown empirically that it is sound. *How can we use clause addition and elimination techniques in all solver incarnations?* Moreover, the construction of unsatisfiability proofs, DRAT proofs [14–16], becomes an important discipline to gain confidence in UNSAT answers. *How can we create DRAT refutations for search-space decomposition based SAT solvers?*

### Acknowledgement

I would like to thank Steffen Hölldobler, Norbert Manthey and Christoph Wernhard for many fruitful discussions.

## References

- [1] Arnold, H.: A linearized DPLL calculus with clause learning. Tech. rep., Universität Potsdam. Institut für Informatik (2009)
- [2] Audemard, G., Lagniez, J.M., Mazure, B., Sais, L.: On freezing and reactivating learnt clauses. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 188–200. Springer (2011)
- [3] 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, California, USA (2009)
- [4] Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 341–355. Springer (May 2003)
- [5] Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Report Series Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
- [6] Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Irwin, M.J. (ed.) DAC 1999. pp. 317–320. ACM (1999)
- [7] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (Jul 1962)
- [8] Davis, M., Putnam, H.: A computing procedure for quantification theory. JACM 7(3), 201–215 (Jul 1960)
- [9] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer (2005)
- [10] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer (2004)
- [11] Gil, L., Flores, P., Silveira, L.M.: PMSat: a parallel version of MiniSAT. JSAT 6, 71–98 (2008)
- [12] Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning 24(1–2), 67–100 (Feb 2000)
- [13] 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 (2012)
- [14] Heule, M., Jr., W.A.H., Wetzler, N.: Trimming while checking clausal proofs. In: FMCAD 2013. pp. 181–188. IEEE (2013)
- [15] Heule, M., Jr., W.A.H., Wetzler, N.: Verifying refutations with extended resolution. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 345–359. Springer (2013)
- [16] Heule, M., Manthey, N., Philipp, T.: Validating unsatisfiability results of clause sharing parallel SAT solvers. In: Pragmatics of SAT 2014 (2014, accepted)
- [17] Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Büning, H.K., Zhao, X. (eds.) SAT 2011. LNCS, vol. 6695, pp. 201–215. Springer (2011)

- [18] Hölldobler, S., Manthey, N., Nguyen, V., Stecklina, J., Steinke, P.: A short overview on modern parallel SAT-solvers. In: ICACSS 2011. pp. 201–206. IEEE (2011)
- [19] Huang, J.: The effect of restarts on the efficiency of clause learning. In: Veloso, M. (ed.) IJCAI 2007. pp. 2318–2323. IJCAI’07, Morgan Kaufmann Publishers Inc., Hyderabad, India (January 2007)
- [20] Hölldobler, S., Manthey, N., Philipp, T., Steinke, P.: Generic CDCL – a formalization of modern propositional satisfiability solvers. In: Hölldobler, S., Malikov, A., Wernhard, C. (eds.) Proc. of the Young Scientists’ International Workshop on Trends in Information Processing. vol. 1145, pp. 25–34. CEUR-WS.ORG (2014)
- [21] Hölldobler, S., Manthey, N., Saptawijaya, A.: Improving resource-unaware SAT solvers. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 17. LNCS, vol. 6397, pp. 519–534. Springer (2010)
- [22] Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer (2010)
- [23] Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer (2012)
- [24] Jurkowiak, B., Li, C.M., Utard, G.: Parallelizing satz using dynamic workload balancing. Electronic Notes in Discrete Mathematics 9, 174–189 (2001)
- [25] Kautz, H.A., Selman, B.: Planning as satisfiability. In: Neumann, B. (ed.) ECAI 1992. pp. 359–363 (1992)
- [26] Lewis, M., Schubert, T., Becker, B.: Multithreaded SAT solving. In: ASP-DAC 2007. pp. 926–931. IEEE Computer Society, Washington, DC, USA (2007)
- [27] Lynce, I., Marques-Silva, J.: SAT in bioinformatics: Making the case with haplotype inference. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 136–141. Springer (2006)
- [28] Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer Berlin Heidelberg (2012)
- [29] 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 (2013)
- [30] Marić, F.: Formalization and implementation of modern SAT solvers. J. Autom. Reason. 43(1), 81–119 (2009)
- [31] Marques Silva, J.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5), 506–521 (1999)
- [32] Martins, R., Manquinho, V., Lynce, I.: An overview of parallel SAT solving. Constraints 17(3), 304–347 (2012)
- [33] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: DAC 2001. pp. 530–535. Association for Computing Machinery (2001)
- [34] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). JACM 53(6), 937–977 (Nov 2006)
- [35] Philipp, T.: Expressive Models for Parallel Satisfiability Solvers. Master thesis, Technische Universität Dresden (2013)
- [36] Rintanen, J.: Engineering efficient planners with SAT. In: Raedt, L.D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F. (eds.) ECAI 2012. Frontiers in Artificial Intelligence and Applications, vol. 242. IOS Press (2012)
- [37] Schubert, T., Lewis, M., Becker, B.: PaMira - a parallel SAT solver with knowledge sharing. In: Abadir, M.S., Wang, L.C. (eds.) MTV 2005. pp. 29–36 (2005)
- [38] Singer, D.: Parallel Resolution of the Satisfiability Problem: A Survey, chap. 5, pp. 123–148. Wiley Interscience (2006)
- [39] Soos, M.: CryptoMiniSat 2.5.0. In: SAT Race Competitive Event Booklet (July 2010)
- [40] Stallman, R.M., Sussman, G.J.: Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. Artificial Intelligence 9(2), 135–196 (1977)
- [41] Subbarayan, S., Pradhan, D.K.: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 276–291. Springer (2005)
- [42] van der Tak, P., Heule, M., Biere, A.: Concurrent cube-and-conquer. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 475–476. Springer (2012)
- [43] Zhang, H., Bonacina, M.P., Hsiang, J.: PSATO: a distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation 21(4), 543–560 (1996)