Soundness of Inprocessing in Clause Sharing SAT Solvers*

Norbert Manthey, Tobias Philipp and Christoph Wernhard

Knowledge Representation and Reasoning Group
Technische Universität Dresden

Abstract. We present a formalism that models the computation of clause sharing portfolio solvers with inprocessing. The soundness of these solvers is not a straightforward property since shared clauses can make a formula unsatisfiable. Therefore, we develop characterizations of simplification techniques and suggest various settings how clause sharing and inprocessing can be combined. Our formalization models most of the recent implemented portfolio systems and we indicate possibilities to improve these. A particular improvement is a novel way to combine clause addition techniques – like blocked clause addition – with clause deletion techniques – like blocked clause elimination or variable elimination.

1 Introduction

The satisfiability problem (SAT) is one of the most prominent problems in theoretical computer science and has many applications in verification, planning, model checking [7] or scheduling [14]. Modern SAT solvers employ many advanced techniques like *clause learning* [33], *non-chronological backtracking*, *restarts* [13], *clause removal* [11,4] and advanced *decision heuristics* [35,4,3], making SAT very attractive for problem-solving. Among all these additional techniques, clause learning is the most powerful one, both from a theoretical and an empirical point of view [36,5,27]. Formula simplification techniques like *variable elimination* [10,40]and *bounded variable addition* [32] are applied to further improve the efficiency. In recent SAT solvers, (e.g. LINGELING [6]), simplifications are also applied during search as *inprocessing* techniques, on all clauses at hand, including learned clauses [26,31].

The parallel architecture of today's computers is utilized in portfolio solvers like PPFOLIO [37], PFOLIOUZK [42], MANYSAT [18] and PLINGELING [6]. In the portfolio approach, different sequential solvers work on the same formula. For instance, PFOLIOUZK consists of several different conflict-driven systematic solvers, look-ahead SAT solvers and incomplete SAT solvers. Each of them is allowed to apply arbitrary simplification techniques. In contrast to PPFOLIO, MANYSAT's portfolio consists only of systematic CDCL SAT solvers, but exchanges learned clauses. Thus, MANYSAT can solve instances faster than the

* The second author was supported by the International MSc Program in Computational Logic (MCL)

best sequential solver of its portfolio. However, no simplification techniques are applied during search. Plingeling combines the two worlds: The solver applies a restricted form of inprocessing and shares unit clauses. Proving soundness of such clause sharing portfolio solvers where simplification techniques are applied during search is not trivial.

To the best of the authors' knowledge, there does not exists a soundness proof of any parallel clause sharing SAT solver with inprocessing. The authors are not aware of a formalization of parallel SAT solvers, which can be used to verify when clause sharing is sound. The research focus so far has been mainly on clause sharing strategies [17,15,16,1], and on utilizing computing resources [22,23].

In this paper, we bridge the gap between simplification techniques and clause sharing SAT solvers. A formal *sharing model* is introduced, which models the computation of clause sharing portfolio solvers with inprocessing. This model allows to reason about the behavior of portfolio solvers. Additionally, it allows to explain how portfolio solvers are working in a compact and easy way. Abstracting from the specific solver implementation is important since modern systems are complex and are written in programming languages with side effects. Equipped with this sharing model, we can show that, assumed the base solving engine is correct, inprocessing does not harm portfolio solvers on satisfiable instances. In general, inprocessing has to be restricted to obtain a valid system. In particular, we will see the soundness of a combination of restricted inprocessing rules and clause sharing as present in the Plingeling system. We will also see the soundness of specific combinations with further inprocessing techniques, for instance that adding blocked clauses is possible in only a single solver incarnation of the portfolio. In addition to soundness proofs, several examples demonstrate that certain combinations of techniques cannot be applied in the presence of clause sharing. The result presented here can be easily lifted to parallel SAT solvers based on the guiding path approach [43], since these solvers only restrict the way how the search space is traversed but work on the same input formula.

The paper is structured as follows: Followed by a specification of the used formal notation and an outline of the necessary preliminaries of modern SAT solving in Section 2, the new sharing model is introduced in Section 3, variations of it are discussed, and the soundness of several combinations of inprocessing techniques and clause sharing is proven. Section 4 concludes the paper.

2 Preliminaries

We introduce the used notion of *Satisfiability Testing* (SAT), briefly discuss recent solving approaches that exploit parallel computing resources and finally give some insights in modern formula simplification techniques.

2.1 Notation and Basic Concepts

We assume a fixed set $\mathcal{V}$ of Boolean variables, or briefly just *variables.* A *literal* is a variable v (*positive literal*) or a negated variable $\overline{v}$ (*negative literal*). We overload the overbar notation: The *complement* $\overline{x}$ of a positive (negative, resp.) literal x is the negative (positive, resp.) literal with the same variable as x. A *clause* is a finite set of literals, a *formula* is a finite set of clauses. We write a clause $\{x_1, \ldots, x_n\}$ also as disjunction $x_1 \vee \ldots \vee x_n$, and a formula $\{C_1, \ldots, C_n\}$ also as conjunction $C_1 \wedge \ldots \wedge C_n$. In this notation, the empty clause is written as $\bot$ (falsum) and the the empty formula as $\top$ (verum). Occasionally, we write formulas with further standard connectives such as $\leftrightarrow$. This can be understood as meta level notation for equivalent formulas in conjunctive normal form.

An *interpretation* is a mapping from the set $\mathcal{V}$ of all Boolean variables to the set $\{\top, \bot\}$ of truth values. The satisfaction relation $\models$ can then be defined inductively as follows: If x is a positive (negative, resp.) literal with variable v, then $I \models x$ holds if and only if $I(v) = \top$ ($I(v) = \bot$, resp.). If C is a clause, then $I \models C$ if and only if there is a literal $l \in C$ such that $I \models l$. If F is a formula, then $I \models F$ if and only if for all clauses $C \in F$ it holds that $I \models C$. A *general formula* is a literal, a clause or a formula. If, for an interpretation I and general formula F it holds that $I \models F$, we say that I is a *model* of F, or that I *satisfies* F. If there exists an interpretation that satisfies F, then F is called *satisfiable*, otherwise *unsatisfiable*. Two general formulas are *equisatisfiable* if and only if either both of them are satisfiable or both are unsatisfiable. A general formula F *entails* a general formula G, written $F \models G$, if and only if for all interpretations I such that $I \models F$ it holds that $I \models G$. This definition of entailment has two nice properties that simplify the proofs in this paper: transitivity and monotonicity. The drawback is that some formula simplification rules must be treated in an extra step. Two general formulas F and G are *equivalent*, written $F \equiv G$, if and only if $F \models G$ and $G \models F$.

A clause that contains exactly a single literal is called a *unit clause*. A clause that contains a literal and its complement is called a *tautology*. 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 in the clauses in F by $\mathsf{lits}(F)$. A literal x such that $x \in \mathsf{lits}(F)$ and $\overline{x} \notin \mathsf{lits}(F)$ is called *pure* in F. If F is a formula and x is a literal, then the formula consisting of all clauses in F that contain x is denoted by F_x. If v is a variable and $C = v \vee C'$ as well as $D = \overline{v} \vee D'$ are clauses, then the clause $C' \vee D'$ is called the *resolvent of C and D upon v*. For two formulas F, G and variable v, the set of all resolvents of a clause in F with a clause in G upon v is denoted by $F \otimes_v G$. The formula F after substituting all occurrences of the variable v with the variable w is denoted by $F[v \mapsto w]$.

2.2 Parallel SAT Solving

The first parallel SAT solvers have been based on the DPLL procedure and divided the search space among the parallel resources [8]. Later on, parallel architecture has been exploited in different ways. A survey on parallel SAT solving is given in [21,34]. Parallel SAT solvers can be classified into competitive approaches and cooperative approaches. The first class contains the *portfolio* approach, where several solver incarnations try to solve the same formula (e.g. [18,6,1]). The latter class contains the *search space partitioning* methods (e.g. [19]). Here, the search space of the input formula is divided into several sub spaces, represented by several modified formulas. Finally, there exist hybrid approaches, which combine competitive and cooperative strategies (e.g. [22,23]). Guiding path solvers [43] are hard to categorize: on the one hand, this approach partitions the search space. On the other hand, these systems do not alter the input formula, but control the partitioning based on the current interpretation of the specific solvers. From a modeling point of view, guiding path solvers could be added to search space partitioning. Since our sharing model is based on the formula that each solver uses, portfolio and guiding path solvers are quite similar.

Since clause learning is essential for sequential solvers, this technique has been lifted to parallel solvers as well: Clauses are shared among solver incarnations [18,17], even if different subformulas are solved [24,29]. However, most of these solver incarnations do not use clause simplification techniques for *inprocessing*. During inprocessing clause simplification techniques are interleaved with search and are also applied to learned and shared clauses. To the best of the authors' knowledge, all sharing implementations do not check the validity of the received clause, but the implementation assumes that using received clauses is sound. Obvious reasons for not performing such checks are the computational costs of testing properties like $F \models C$ or testing equisatisfiability of F and $F \wedge C$ with respect to a clause C received by a solver whose working formula is F.

In the following, we focus on the portfolio approach, and discuss some systems in detail. A nice property of this approach is that the search can be stopped, as soon as one solver incarnation found the solution for the input formula. This property motivates combining special solvers for formulas from special categories to a single portfolio solver, which then solves a given formula in the time required by the fastest of the sequential component solvers. Portfolio solvers in recent competitions like PPFOLIO [37] and PFOLIOUZK [42] simply execute several powerful SAT solvers in parallel, even scheduled on a sequential machine. This easy approach scales with the number of available solvers. If the number of available parallel computing resources exceeds the number of solvers, parallel solvers will be added to the portfolio.

More sophisticated portfolio approaches use a single solving engine and execute multiple incarnations in parallel (e.g. ManySAT [18] or PeneLoPe [2]). An advantage of single engine portfolio solvers is that learned clauses can be shared among the incarnations. A soundness proof is obvious: since all incarnations are solving the same formula, and the solver always preserve the equivalence of the formula, all learned clauses are a consequence of the input formula. Knowledge sharing enables the portfolio solver to solve a formula even faster than the best sequential incarnation, because the search of this best incarnation is enhanced with more clauses that cut off search space. Another difference to these single engine portfolio solvers has been introduced with the Plingeling system [6]. This solver executes Lingeling as the solving engine, which applies simplification technique during search, and also shared learned clauses.

2.3 Inprocessing

Simplifying the formula before giving it to the SAT solver has become a crucial part of the solving chain. Several techniques, such as bounded variable elimination [40,10], blocked clause elimination [25] and addition [28,26], equivalent literal elimination [12], probing [30], and automated re-encoding by using extended resolution [32,41] have been proposed for simplification. These techniques have been originally suggested with preprocessing, application before search, in mind. However, with minor modifications to treat learned clauses correctly, these techniques can also be used during search as *inprocessing*. Successful SAT solvers that participated in recent competitions, such as PrecoSAT [6], CryptoMiniSAT [38] or Lingeling [6], utilize this. Many simplification techniques result in formulas that have a certain semantic relationship to the input formula, defined as follows:

Definition 1 (Unsat-Preserving Consequence). *A formula F' is called an* unsat-preserving consequence *of a formula F if and only if*

1. *$F \models F'$, and*
2. *If F is unsatisfiable, then F' is unsatisfiable.*

Note that if F' is an unsat-preserving consequence of F, it immediately follows that F and F' are equisatisfiable. If F' is an unsat-preserving consequence of F, then F' is also a consequence of F in the standard sense. That is, any model of F is also a model of F'. A special case of unsat-preserving consequence applies if $F \equiv F'$. As examples with respect to simplifications that involve addition and removal of clauses consider that if C is a clause entailed by F, then $F \wedge C$ is an unsat-preserving consequence of F; and if $F = F' \wedge C$ and F' is equisatisfiable with F, then F' is an unsat-preserving consequence of F.

In [26] a formal framework that can be used to prove the correctness of existing and new inprocessing rules has been presented. From the properties of simplification techniques discussed there we will drop the introduced redundancy property, because this property is related to very specific simplification procedures that go beyond the scope of this paper. The particular simplifications that we will consider can be described as follows, where we adopt the convention that F denotes the input formula and F' the output formula:
Variable Elimination (VE) [40,10] eliminates a variable v by resolution: $F' = (F \cup F_v \otimes_v F_{\overline{v}}) \setminus (F_v \cup F_{\overline{v}})$. Note that $v \notin \mathsf{vars}(F')$. It follows from properties of propositional resolution that F' is an unsat-preserving consequence of F.
Equivalence Elimination (EE) [12] replaces all occurrences of a variable v with the variable w, if the input F entails the equivalence $w \leftrightarrow v$, that is, $F' = F[v \mapsto w]$. Since $F \equiv F \wedge (w \leftrightarrow v) \equiv F[v \mapsto w] \wedge (w \leftrightarrow v) \equiv F' \wedge (w \leftrightarrow v)$, it follows that $F \models F'$. Moreover, this technique preserves satisfiability [20] and thus F' is an unsat-preserving consequence of F.
Blocked Clause Elimination (BCE) [25] removes a *blocked clause* C from the input formula F. Then $F' = F \setminus \{C\}$. A clause C is called *blocked*, if it contains a literal x such that for all clauses $D \in F_{\overline{x}}$ the resolvent of C and D upon the

variable of x is a tautology. Obviously, $F \models F'$ and since equisatisfiability is preserved by this technique, F' is an unsat-preserving consequence of F [25].

Extended Resolution (ER) [41] adds a definition of a fresh variable v to the formula: $F' = F \wedge (\overline{v} \vee x \vee y) \wedge (\overline{x} \vee v) \wedge (\overline{y} \vee v)$, where the two literals x and y occur already in the formula, that is, $x, y \in \mathsf{lits}(F)$. As presented in [41], this addition preserves satisfiability of the formula, that is, F' and F are equisatisfiable. However, $F \not\models F'$, because of the fresh variable, and therefore F' is not an unsat-preserving consequence of F.

Blocked Clause Addition (BCA) [28,26] is the dual technique of BCE: BCA adds a clause C, such that this clause is blocked in the new formula: $F' = F \wedge C$. Note that the formula F' is not an unsat-preserving consequence of F, as can be seen in the following small counterexample: Let $F = (x \vee y)$. Then the clause $(\overline{x} \vee \overline{y})$ is blocked with the blocking literal $\overline{y}$. However, $F \not\models (\overline{x} \vee \overline{y})$.

Bounded Variable Addition (BVA) [32] can be understood as adding a partial definition of a fresh variable v to the formula: First, a fresh variable 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 $x, y \in \mathsf{lits}(F)$. The formulas F and G are equisatisfiable. 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 $(v \vee E)$. The resulting formula H is equivalent to G. Finally, the formula F', the result of BVA, is obtained from H by removing the clause $(v \vee \overline{x} \vee \overline{y})$. As presented in [32], BVA preserves satisfiability, that is, F' and F are equisatisfiable, because the old clauses can be restored by resolution. However, $F \not\models F'$ and thus F' is not an unsat-preserving consequence of F.

3 Sharing Clauses in Portfolio Solvers

We model the computation of clause-sharing solver portfolios by means of state transition systems as follows: A state of computation of a single sequential solver is a formula. A state of computation of a portfolio of n sequential solvers $Solver_1, \dots, Solver_n$ is an n-tuple $(F_1, \dots, F_n)$ of formulas, where F_i is the state of the $Solver_i$. Thus, a state of a portfolio is a snapshot of the states of its member solvers. A *portfolio system with input formula* F_0 *and multiplicity* n is a state transition system whose set of states is $\{(F_1, \dots, F_n) \mid F_1, \dots, F_n \text{ are formulas}\} \cup \{\mathsf{SAT}, \mathsf{UNSAT}\}$, whose initial state $\mathsf{init}(n, F_0)$ is the n-tuple $(F_0, \dots, F_0)$, and whose set of terminal states is $\{\mathsf{SAT}, \mathsf{UNSAT}\}$. For a transition relation $\rightsquigarrow$ of a portfolio system we define $\rightsquigarrow^*$ as the reflexive transitive closure of $\rightsquigarrow$, we define $x \rightsquigarrow^0 x$, and for all natural numbers $n > 0$ we define: $x \rightsquigarrow^n y$ if and only if $x \rightsquigarrow^{n-1} z \rightsquigarrow y$. In the course of the paper, we will investigate the soundness of portfolio systems with different transition relations corresponding to different allowed inprocessing methods. Soundness of a SAT solver, and thus also of a portfolio system, is a combination of two properties: refutational soundness and soundness with respect to satisfiability. More precisely, a portfolio system with input formula F_0, multiplicity n and transition

SAT-rule:	$(F_1, \dots, F_i, \dots, F_n) \rightsquigarrow_{\mathsf{SAT}} \mathsf{SAT}$ iff F_i is satisfiable.
UNSAT-rule:	$(F_1, \dots, F_i, \dots, F_n) \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$ iff F_i is unsatisfiable.
CM-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{CM}} (F_1, \dots F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff $F_i \equiv F'_i$.
CS-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{CS}} (F_1, \dots, F_{i-1}, F_i \wedge C, F_{i+1}, \dots, F_n)$ iff $C \in F_j$ for some $j \in \{1, \dots, i-1, i+1, \dots, n\}$.
UI-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{UI}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff F_i and F'_i are equisatisfiable.
ER-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{ER}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff F'_i is obtained by extended resolution from F_i.
BVA-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{BVA}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff F'_i is obtained by bounded variable addition from F_i.
RI-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{RI}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff F'_i is an unsat-preserving consequence of F_i.
ADD-rule:	$(F_1, \dots, F_n) \rightsquigarrow_{\mathsf{ADD}} (F_1 \wedge C, F_2, \dots, F_n)$ iff $\mathsf{vars}(C) \subseteq \mathsf{vars}(F_0)$, and the formulas $F_1 \wedge C$ and F_1 are equisatisfiable.
DEL-rule:	$(F_1, \dots, F_{i-1}, F_i, F_{i+1}, \dots, F_n) \rightsquigarrow_{\mathsf{DEL}} (F_1, \dots, F_{i-1}, F'_i, F_{i+1}, \dots, F_n)$ iff $i > 1$ and F_i is an unsat-preserving consequence of F'_i.

Fig. 1: Transition relations used to characterize clause sharing models 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 $i \in \{1, \dots, n\}$. Fresh variables introduced by extended resolution (ER-rule) and by bounded variable addition (BVA-rule) have to be *globally* fresh, that is, they are not allowed to occur in $F_1, \dots, F_n$.

relation $\rightsquigarrow$ is called *sound* if and only $\mathsf{init}(n, F_0) \rightsquigarrow^* \mathsf{UNSAT}$ implies that F is unsatisfiable and $\mathsf{init}(n, F_0) \rightsquigarrow^* \mathsf{SAT}$ implies that F is satisfiable.

We investigate four different portfolio systems whose transition relation is composed of the relations presented in Fig. 1, which we also call *rules*. The particular systems that we are going to investigate are SysA, SysB, SysC and SysD, characterized by the following respective transition relations:

$$\begin{aligned}
\rightsquigarrow_{\mathsf{SysA}} &:= \rightsquigarrow_{\mathsf{SAT}} \cup \rightsquigarrow_{\mathsf{UNSAT}} \cup \rightsquigarrow_{\mathsf{CM}} \cup \rightsquigarrow_{\mathsf{CS}}.\\
\rightsquigarrow_{\mathsf{SysB}} &:= \rightsquigarrow_{\mathsf{SysA}} \cup \rightsquigarrow_{\mathsf{UI}}.\\
\rightsquigarrow_{\mathsf{SysC}} &:= \rightsquigarrow_{\mathsf{SysA}} \cup \rightsquigarrow_{\mathsf{RI}} \cup \rightsquigarrow_{\mathsf{ER}} \cup \rightsquigarrow_{\mathsf{BVA}}.\\
\rightsquigarrow_{\mathsf{SysD}} &:= \rightsquigarrow_{\mathsf{SysA}} \cup \rightsquigarrow_{\mathsf{ADD}} \cup \rightsquigarrow_{\mathsf{DEL}} \cup \rightsquigarrow_{\mathsf{ER}} \cup \rightsquigarrow_{\mathsf{BVA}}.
\end{aligned}$$

- ▶ SysA is a basic portfolio system that models clause sharing portfolios where inprocessing rules preserve equivalence: The first termination rule is the SAT-rule that terminates the computation with the answer SAT, if one formula

F_i in the state is satisfiable. Likewise, the second termination rules is the UNSAT-rule that leads to the answer UNSAT, if one formula F_i in the state is unsatisfiable. Modern complete solvers are modifying the formula by adding and removing learned clauses. Such clause management is modelled 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 CS-rule that adds a clause C from the formula F_j to the formula F_i, where $i \neq j$. As we will see later, solvers like ManySAT [18] or PeneLoPe [2] can be modeled as instances of SysA.

- SysB extends the system SysA by the UI-rule that performs inprocessing, just constrained to preserve equisatisfiability: The UI-rule replaces a formula F_i with another formula F_i', if they are equisatisfiable. We will later see that the general UI-rule does not harmonize with clause sharing, leading to refutational unsoundness.
- SysC extends the system SysA by the RI-, ER- and BVA-rule. The RI-rule replaces a formula F_i with a formula F_i', if the F_i' is an unsat-preserving consequence of F_i. Extended resolution and bounded variable addition can be performed on a formula F_i by the ER- and BVA-rule. We will later see that the formalism SysC is sound, and that Plingeling [6] is an instance of this formalism.
- SysD extends the system SysA by the ER- and BVA-rule and two new rules that allow to perform clause addition techniques on a single designated solver whose state is represented without loss of generality by F_1, and only clause deletion techniques on the remaining solvers, represented by $F_2, \ldots, F_n$ to obtain a sound system: The ADD-rule adds a clause C to the formula F_1, if F_1 and $F_1 \wedge C$ are equisatisfiable and $\mathsf{vars}(C) \subseteq \mathsf{vars}(F_0)$. On the other hand, the clause deletion is captured by the DEL-rule, which like the RI-rule, except that the DEL-rule can only modify $F_2, \ldots, F_n$.

3.1 Sharing Model SysA – Where Equivalence is Preserved

Let us start with the system SysA that models clause sharing portfolios where inprocessing preserves equivalence. The rules of SysA guarantee that all formulas $F_i \in (F_1, \ldots, F_n)$ of the current state are equivalent to the input formula, as expressed by the following lemma:

Lemma 2 (Key Invariant of SysA). *Let $n \geq 1$, let $F_0, F_1, \ldots, F_n$ be formulas, and let $m \geq 0$. If $\mathsf{init}(n, F_0) \rightsquigarrow^m_{\mathsf{SysA}} (F_1, \ldots, F_n)$, then for all $i \in \{1, \ldots, n\}$ it holds that $F_i \equiv F_0$.*

Proof. The CM-rule preserves equivalence and therefore the CS-rule only adds clauses that are entailed by the corresponding formula. The claim then holds since equivalence is transitive. □

The following theorem is a straightforward consequence of the above lemma:

Theorem 3 (Soundness of SysA). SysA *is sound.*

Proof. If the input formula F_0 is satisfiable (unsatisfiable, resp.), the UNSAT-rule (SAT-rule, resp.) is never applicable, since by Lemma 2 all formulas in states reached by SysA are equivalent to F_0. $\square$

The portfolio solvers ManySAT and PeneLoPe are built upon MiniSAT [18,2]. PeneLoPe differs from ManySAT in the clause import and export strategy and in the applied heuristics. Since the learned clauses in MiniSAT can be obtained by resolution [39], MiniSAT preserves the equivalence of formulas. Before searching for a model of the input formula F_0, the formula is subjected to a preprocessor in ManySAT and PeneLoPe. The portfolio systems then work on the result of the preprocessor. Consequently, SysA models the behavior of ManySAT and PeneLoPe with respect to the preprocessed formula as input.

3.2 Sharing Model SysB – Inprocessing without Limits

The portfolio transition system SysB models clause sharing portfolios where inprocessing can be applied constrained only by equisatisfiability. Many inprocessing techniques like variable elimination (VE), equivalence elimination (EE) and blocked clause addition (BCA) do preserve equisatisfiability, but not equivalence. Combining clause sharing and the general UI-rule inprocessing is problematic:

Example 4. Consider the two formulas x and $\overline{x}$, where x is a variable. Both formulas x and $\overline{x}$ are satisfiable and equisatisfiable. Since $x \wedge \overline{x}$ is unsatisfiable, the execution $\mathsf{init}(2, x) = (x, x) \rightsquigarrow_{\mathsf{UI}} (x, \overline{x}) \rightsquigarrow_{\mathsf{CS}} (x \wedge \overline{x}, \overline{x}) \rightsquigarrow_{\mathsf{UNSAT}}$ UNSAT of a parallel SAT solver with transition relation $\rightsquigarrow_{\mathsf{SysB}}$ produces an incorrect result.

Thus, the system SysB is not refutationally sound. This small example shows that general inprocessing is incompatible with clause sharing, since the answer UNSAT can be incorrect. The phenomenon that the addition of a clause makes a formula unsatisfiable is the reason why clause sharing is a non-trivial problem. On the other hand, the system SysB is sound with respect to satisfiability. SAT answers obtained with SysB are still correct, as stated in the Theorem 5 below, preceded by a lemma that shows the underlying invariant of SysB transitions:

Lemma 5 (Key Invariant of SysB). *Let $n \geq 1$, let $F_0, F_1, \ldots, F_n$ be formulas, and let $m \geq 0$. If $\mathsf{init}(n, F_0) \rightsquigarrow^m_{\mathsf{SysB}} (F_1, \ldots, F_n)$ and for some $i \in \{1, \ldots, n\}$ it holds that if F_i is satisfiable, then F_0 is satisfiable.*

Proof. By induction on m: For the base case $m = 0$, the statement is trivially true since every component formula in $\mathsf{init}(n, F_0)$ is identical to F_0. For the induction step, assume that the claim holds for the state $(F_1, \ldots, F_n)$ and that $(F_1, \ldots, F_n) \rightsquigarrow_R (F_1, \ldots, F_{i-1}, F'_i, F_{i+1}, \ldots, F_n)$ for some $1 \leq i \leq n$ and some rule R in SysB. The induction step follows from the fact that if F'_i is satisfiable, then F_i is satisfiable, which holds for each non-terminating rule of SysB:

- CM-rule: Since $F'_i \equiv F_i$.
- UI-rule: Since F'_i and F_i are equisatisfiable.

– CS-rule: In this case $F'_i = F_i \wedge C$ for some clause C. Thus, it holds that $F'_i \models F_i$ which implies that if F'_i is satisfiable, then F_i is satisfiable. $\square$

Theorem 6 (Soundness of SysB w.r.t Satisfiability). *Let $n \geq 1$ and let F_0 be a formula. If $\mathsf{init}(n, F_0) \rightsquigarrow^m_{\mathsf{SysB}}$ SAT, then F_0 is satisfiable.*

Proof. Immediate from Lemma 5 and the definition of the SAT-rule. $\square$

3.3 Sharing Model SysC – With Clause Deletion Techniques

System SysC models clause sharing portfolios where clause deletion techniques are used. In contrast to model SysB, the result of a computation in SysC is always sound. The parallel SAT solver Plingeling is an instance of SysC. The portfolio of Plingeling consists of differently configured incarnations of the sequential solver Lingeling. The following inprocessing rules are used in Plingeling [6]: Variable elimination (VE) [10], equivalence elimination (EE) [12], and blocked clause elimination (BCE) [25]. We do not discuss the other used techniques that preserve equivalence like hyper binary resolution since they are modelled by the CM-rule. Since these techniques transform a formula F into unsat-preserving consequence of F, these techniques can be modelled by the RI-rule.

In the following, we need to trace the clauses introduced by extended resolution or bounded variable addition. For a particular sequence of transition steps $S_0 \rightsquigarrow S_1 \rightsquigarrow \ldots \rightsquigarrow S_m$, we let D_m denote the set of all definition clauses that were introduced by the ER- and BVA-rule in the sequence. Note that an implementation does not need to construct the set D_m, but we will use this set in the proofs to show that if $\mathsf{init}(n, F_0) \rightsquigarrow^m (F_1, \ldots, F_n)$, then F_0 and F_i are always equisatisfiable for $i \in \{0, \ldots, n\}$. On the other hand, an implementation has to guarantee that the variables introduced by extended resolution or bounded variable addition are *globally* fresh, that is, fresh throughout all involved sequential solvers. Incorrect UNSAT answers can then not be obtained with SysC.

Theorem 8 below states the soundness of SysC. This theorem is based again on a key invariant stated in Lemma 7 as claim (iii). The lemma shows two further invariants of SysC as claims (i) and (ii), which involve the D_m. In the context of this paper these are just applied to prove invariant (iii).

Lemma 7 (Key Invariants of SysC). *Let $n \geq 1$, let $F_0, F_1, \ldots, F_n$ be formulas, and let $m \geq 0$. Assume $\mathsf{init}(n, F_0) \rightsquigarrow^m_{\mathsf{SysC}} (F_1, \ldots, F_n)$ with a transition sequence that has the formula D_m as the set of clauses introduced by the* ER- *and* BVA-*rule. Then the following properties hold:*

(i) $F_0 \wedge D_m \models F_1 \wedge \ldots \wedge F_n$.
(ii) *F_0 and $F_0 \wedge D_m$ are equisatisfiable.*
(iii) *For all $i \in \{1, \ldots, n\}$ it holds that F_i and F_0 are equisatisfiable.*

Proof. We show the statement by induction on the number m of transition steps. For the base case $m = 0$, the claims (i)–(iii) are easy to see, since $D_0 = \top$ and $\mathsf{init}(n, F_0)$ is the n-tuple $(F_0, \ldots, F_0)$. For the induction step, assume that the claim holds for the state $(F_1, \ldots, F_n)$ and that $(F_1, \ldots, F_n) \rightsquigarrow_{\mathsf{R}}$

$(F_1, \ldots, F_{i-1}, F'_i, F_{i+1}, \ldots, F_n)$ for some rule R in SysC. We distinguish according to the applied rule R:

- CM-rule: Then $F'_i \equiv F_i$ and $D_{m+1} = D_m$. (i) holds since the substitution of a subformula by an equivalent formula preserves equivalence. (ii) holds since $D_{m+1} = D_m$. (iii) holds since F'_i and F_i are equivalent and equisatisfiable.
- RI-rule: Then $F_i \models F'_i$, F_i and F'_i are equisatisfiable and $D_{m+1} = D_m$. (i) holds since the entailment relation is transitive. (ii) holds since $D_{m+1} = D_m$. (iii) holds by the definition of the RI-rule.
- ER-rule: Then $F'_i = F_i \wedge (\overline{v} \vee x \vee y) \wedge (\overline{x} \vee v) \wedge (\overline{y} \vee v)$ and $D_{m+1} = D_m \wedge (\overline{v} \vee x \vee y) \wedge (\overline{x} \vee v \wedge (\overline{y} \vee v)$, where v is a fresh variable and $x, y \in \mathsf{lits}(F_i)$. Then (i) holds since the formula $v \leftrightarrow x \vee y$ is added on the left hand side and the right hand side of the entailment. (ii) holds since v is a fresh variable and (iii) holds since extended resolution preserves satisfiability.
- BVA-rule: Bounded variable addition consists of three steps: First, a fresh variable v is defined by $(v \vee \overline{x} \vee \overline{y}) \wedge (\overline{v} \vee x) \wedge (\overline{v} \vee y)$ like in the ER-rule, then the formula is rewritten into an equivalent formula, as in the CM-rule. Finally, the clause $(v \vee \overline{x} \vee \overline{y})$ is deleted, which can be modeled by the RI-rule.
- CS-rule: (i) is clear since $F_1 \wedge \ldots \wedge F_{i-1} \wedge F_i \wedge F_{i+1} \wedge \ldots \wedge F_n \equiv F_1 \wedge \ldots \wedge F_{i-1} \wedge F_i \wedge C \wedge F_{i+1} \wedge \ldots \wedge F_n$, which follows since $C \in F_j$ for some $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$. (ii) holds since $D_{m+1} = D_m$. (iii) holds since the equisatisfiability of F_0 and $F'_i = F_i \wedge C$ can be proven as follows: In case F_0 is unsatisfiable, by the induction assumption F_i is also unsatisfiable, and thus $F_i \wedge C = F'_i$ is also unsatisfiable. Assume now the other case, that F_0 is satisfiable. By statement (ii) of the induction assumption it follows that $F_0 \wedge D_m$ is satisfiable, and thus by statement (i) of the induction assumption that also $F_1 \wedge \ldots \wedge F_n$ is satisfiable. Since $i, j \in \{1, \ldots, n\}$ and $F_j \models C$ it follows that also $F_i \wedge C = F'_i$ is satisfiable. $\square$

Theorem 8 (Soundness of SysC). SysC *is sound.*

Proof. Let $n \geq 0$ and let $F_0, F_1, \ldots, F_n$ be formulas such that $\mathsf{init}(n, F_0) \rightsquigarrow^*_{\mathsf{SysC}} (F_1, \ldots, F_n) \rightsquigarrow_{\mathsf{SysC}} \mathsf{UNSAT}$ (SAT, resp.). From the definition of the UNSAT-rule (SAT-rule, resp.) it follows that there exists an $i \in \{1, \ldots, n\}$ s.t. the formula F_i is unsatisfiable (satisfiable, resp.). By Lemma 7.iii it follows that F_0 is unsatisfiable (satisfiable, resp.). $\square$

The above theorem shows in particular that combining clause sharing and applying inprocessing rules as done in PLINGELING is sound.

3.4 Sharing Model SysD – With Clause Addition Techniques

Motivated by the blocked clause addition technique, we will now study how to combine clause addition with clause deletion techniques and develop the sharing model SysD. To the best of our knowledge, blocked clause addition is not applied in clause sharing portfolios. The following examples illustrate two main problems that occur: First, if we allow two sequential solver to perform clause addition techniques like blocked clause addition, UNSAT-answers may be incorrect.

Example 9. Consider the satisfiable formula $F_0 = (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (x \vee z) \wedge (y \vee \overline{z})$ from [26]. Note that the clause $(\overline{x} \vee \overline{z})$ is blocked w.r.t. $\overline{z}$ and the clause $(\overline{y} \vee z)$ is blocked w.r.t. z. By using BCA, $\mathsf{init}(2, F_0) \rightsquigarrow_{\mathsf{UI}} (F_0 \wedge (\overline{x} \vee \overline{z}), F_0) \rightsquigarrow_{\mathsf{UI}} (F_0 \wedge (\overline{x} \vee \overline{z}), F_0 \wedge (\overline{y} \vee z)) \rightsquigarrow_{\mathsf{CS}} (F_0 \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{y} \vee z), F_0 \wedge (\overline{y} \vee z)) \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$, but this answer is incorrect.

This incorrect answer is the reason, why we restrict clause addition techniques such that they are permitted only in one designated solver, the first component of the portfolio. The second problem is combining clause deletion techniques and clause addition techniques in one solver: it may lead to incorrect UNSAT-answers:

Example 10. Consider the satisfiable formula $F_0 = F' \wedge (\overline{x} \vee \overline{z})$ from [26] where $F' = (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (x \vee z) \wedge (y \vee \overline{z})$. Then, $(\overline{x} \vee \overline{z})$ is blocked and after removing this clause, the clause $(\overline{y} \vee z)$ can be again added by blocked clause addition. Then, $\mathsf{init}(2, F_0) \rightsquigarrow_{\mathsf{UI}} (F', F' \wedge (\overline{x} \vee \overline{z})) \rightsquigarrow_{\mathsf{UI}} (F' \wedge (\overline{y} \vee z), F' \wedge (\overline{x} \vee \overline{z})) \rightsquigarrow_{\mathsf{CS}} (F' \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z}), F' \wedge (\overline{x} \vee \overline{z})) \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$, but this answer is incorrect.

For this reason, the presented system in general applies clause deletion techniques in all solvers *except* the designated solver in which clause addition is permitted. With these examples in mind, we can show the following property in SysD:

Lemma 11 (Key Invariants of SysD). *Let $n \geq 1$, let $F_0, F_1, \ldots, F_n$ be formulas, and let $m \geq 0$. Assume $\mathsf{init}(n, F_0) \rightsquigarrow^m_{\mathsf{SysD}} (F_1, \ldots, F_n)$ with a transition sequence that has the formula D_m as the set of clauses introduced by the* ER- *and* BVA-*rule. Then the following properties hold:*

(i) *$F_1 \wedge D_m \models F_2 \wedge \ldots \wedge F_n$.*
(ii) *F_1 and $F_1 \wedge D_m$ are equisatisfiable.*
(iii) *for all $i \in \{1, \ldots, n\}$ it holds that F_i and F_0 are equisatisfiable.*

Proof. As before, we show the statement by induction on the number m of transition steps. For the base case $m = 0$, the claims (i)–(iii) are easy to see, since $D_0 = \top$ and $\mathsf{init}(n, F_0)$ is the n-tuple $(F_0, \ldots, F_0)$. For the induction step, assume that the claim holds for the state $(F_1, \ldots, F_n)$ and that $(F_1, \ldots, F_n) \rightsquigarrow_{\mathsf{R}} (F_1, \ldots, F_{i-1}, F'_i, F_{i+1}, \ldots, F_n)$ for some rule R in SysD. We distinguish according to the applied rule R:

- CM-rule: Then $F'_i \equiv F_i$ and $D_m = D_{m+1}$. Claims (i) and (ii) follow since replacement of a subformula with an equivalent formula preserves equivalence, claim (iii) follows since equivalence implies equisatisfiability.
- ADD-rule: Then $F'_1 = F_1 \wedge C$ and F'_1 is equisatisfiable to F_1. Since the entailment relation is monotone, (i) holds. (ii) follows since by the definition of the ADD-rule $\mathsf{vars}(C) \subseteq \mathsf{vars}(F_0)$ and D_m only contains clauses introduced by extended resolution or bounded variable addition. (iii) is an immediate consequence of the definition of the ADD-rule.
- DEL-rule. Then $F_i \models F'_i$ for $i \in \{2, \ldots, n\}$. Consequently, $F_1 \wedge D_m \models F_2 \wedge \ldots F_{i-1} \wedge F_i \wedge F_{i+1} \wedge \ldots \wedge F_n \models F_2 \wedge \ldots F_{i-1} \wedge F'_i \wedge F_{i+1} \wedge \ldots \wedge F_n$ and thus (i) holds. Since $i \neq 1$, (ii) holds and (iii) is a consequence of the requirement that F_i and F'_i are equisatisfiable implied by the unsat-preserving consequence precondition of the rule.

– ER-rule: Then $F_i' = F_i \wedge (\overline{v} \vee x \vee y) \wedge (\overline{x} \vee v) \wedge (\overline{y} \vee v)$, where v is a globally fresh variable and $x, y \in \mathsf{lits}(F_i)$. (iii) holds as extended resolution preserves satisfiability. For showing (i) and (ii) we consider the following cases:
  - $i = 1$: Then, (i) is an immediate consequence of the entailment relation being monotone. (ii) holds since a globally fresh variable was defined.
  - $i > 1$: Then (i) holds as the added clause of F_i is also contained in D_{m+1}. (ii) holds as $i \neq 1$ and a globally fresh variable was defined.

– BVA-rule: Then $F_i' = F_i \wedge C$ and $D_{m+1} = D_m \wedge C'$ where $C = (\overline{v} \vee x) \wedge (\overline{v} \vee y)$ and $C' = C \wedge (\overline{v} \vee x \vee y)$. Similar to the ER-rule, (i) holds. Since $F_i' \wedge D_{m+1} \equiv F_i \wedge D_{m+1}$ and F_i and F_i' are equisatisfiable, (ii) holds. (iii) holds since bounded variable addition preserves equisatisfiability.

– CS-rule: Then $F_i' = F_i \wedge C$ where $C \in F_j$ for some $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$. We consider the following cases:
  - $j = 1$ and $i > 1$: Then F_1 exports a clause. (i) holds as the entailment relation is monotone. (ii) holds since $i > 1$. (iii) is shown as follows: If F_i is unsatisfiable, then $F_i \wedge C$ is unsatisfiable and by the induction assumption, F_0 is unsatisfiable. Otherwise, if F_i is satisfiable, it follows by induction assumption that F_0 is satisfiable and then that F_1 is satisfiable. Then $F_1 \wedge D_m$ is satisfiable (since D_m only contains definitions) and hence by (i) $F_1 \wedge D_m \wedge F_2 \wedge \ldots \wedge F_n$ is satisfiable. We can conclude that $F_i \wedge C$ is satisfiable as $C \in F_1$.
  - $i = 1$ and $j > 1$: Then F_1 imports the clause C from F_j. (i) holds since the entailment relation is monotone. (ii) is proven as follows: Suppose F_1 is satisfiable. Then $F_1 \wedge D_m$ is satisfiable by induction assumption and since (ii) holds we know that $F_1 \wedge D_m \models C$. Consequently $F_1 \wedge C \wedge D_m$ is satisfiable. Otherwise, if F_1 is unsatisfiable, $F_1 \wedge D_m$ is unsatisfiable by induction assumption and consequently $F_1 \wedge C \wedge D_m$ is unsatisfiable. For showing (iii), suppose that F_1 is unsatisfiable, then $F_1 \wedge C$ is unsatisfiable. Otherwise, let F_1 be satisfiable, then $F_1 \wedge D_m$ is satisfiable by induction assumption and consequently $F_1 \wedge D_m \wedge F_2 \wedge \ldots F_n$ is satisfiable by (i). Therefore, $F_1 \wedge C$ is satisfiable since $C \in F_j$.
  - $j \neq 1$ and $i \neq 1$: This can be done similar to the proof of Prop. 7. □

Theorem 12 (Soundness of SysD). SysD *is sound.*

Proof. Analogously to the proof of Theorem 8, but based on Lemma 11.iii. □

Theorem 12 justifies that we can use clause addition procedures in a single solver and clause deletion procedures in the remaining solvers. The authors are not aware of a parallel SAT solver that uses this combination of techniques.

Example 10 shows that mixing deletion and addition procedures in a single sequential solver does not work. Moreover, using clause addition techniques in more than a single solver incarnation is problematic. If solver S_1 adds a clause C by *technique A* and solver S_2 adds clause D by *technique B*, then S_1 might not be able to apply B anymore since applying B might not preserve equisatisfiability. As already pointed out in [26], the application of simplification rules must take the learned clauses into account. Accordingly, the solver incarnations that apply clause addition techniques must take all clauses of the other solvers into account. Hence, if several solvers in a portfolio apply clause addition techniques, the implementation has to guarantee that the learned clause database is synchronized before. Otherwise, learned clauses of other solvers could be received after the clause addition and then the soundness of the overall system can break. We require that the ADD-rule only adds clauses that have variables occurring in the original formula. The reason is illustrated in the following:

Example 13. Consider the satisfiable formula $F_0 = (x \land y)$. Suppose that the ADD-rule does not require that the added clause C only contains variables that occur in F_0. Then $\mathsf{init}(2, F_0) \rightsquigarrow_{\mathsf{BVA}} (F_0, F_0 \land (v \lor \overline{x} \lor \overline{y}) \land (\overline{v} \lor x) \land (\overline{v} \lor y)) \rightsquigarrow_{\mathsf{CS}} (F_0 \land (\overline{v} \lor x), F_0 \land (v \lor \overline{x} \lor \overline{y}) \land (\overline{v} \lor x) \land (\overline{v} \lor y)) \rightsquigarrow_{\mathsf{ADD}} (F_0 \land (\overline{v} \lor x) \land \overline{v}, F_0 \land (v \lor \overline{x} \lor \overline{y}) \land (\overline{v} \lor x) \land (\overline{v} \lor y)) \rightsquigarrow_{\mathsf{CS}} (F_0 \land (\overline{v} \lor x) \land \overline{v}, F_0 \land (v \lor \overline{x} \lor \overline{y}) \land (\overline{v} \lor x) \land (\overline{v} \lor y) \land \overline{v}) \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$, but this answer is incorrect.

Restricting the variables of the added clause is not a big restriction. Basically, instead of adding a clause $(v \lor C)$, where v is defined by as $(x \lor y)$, one can add the clause $(x \lor y \lor C)$ by the ADD-rule. Afterwards, the clause $(x \lor C)$ can be obtained by resolution. This technique only works, if the definition is completely present at the solver. An alternative to this syntactical restriction of the added clause is the requirement that F_1 and $F_1 \land C \land D_m$ are equisatisfiable.

3.5 Guiding Path Solvers

Rooted in the DPLL algorithm [9], a *guiding path* [43] is the current partial interpretation of a (sequential) SAT solver. In the parallel setting, assume the sequence of decision literals D_i of the solver S_i to be $D_i = (l_1, \ldots, l_n)$. Now, if another solver incarnation S_j should be added to the parallel solver, this solver would simply create another sequence $D_j = D_i[l_k \mapsto \overline{l_k}]$, for exactly one $k \leq n$. Thus, the new sequence D_j differs exactly in the polarity of one literal of the sequence D_i. Now, the two solvers proceed with their search until one of the two either finds a solution or backtracks to the sequence $D = (l_1, \ldots, l_{k-1})$. If the latter case is reached, the whole sub search space can be closed, and thus also the other solver can be terminated and given a new sequence. Note, none of the steps in the guiding path approach touched the underlying formula F.

Therefore, our presented sharing model is fully applicable to guiding path solvers, because it models only the formulas but not the search process of the single solver incarnations.

4 Conclusion

In SAT solving, it is desirable to explore advanced techniques in combination with each other. However, soundness of such combinations is not always easy to see. Combinations that are sound and useful, but involve constraints that are too complicated to be easily captured by intuition might be missed with experimental verification approaches. Here we have seen an approach to overcome this situation for the case of parallel SAT solvers that perform clause sharing in combination with a variety of inprocessing techniques. The behavior of portfolio solvers is represented formally as a state transition system. Different sets of transition rules allow to represent different combinations of inprocessing simplifications and ways of clause sharing among the component solvers running in parallel. The three currently most successful clause sharing portfolio solving systems can be modeled as such transition systems, allowing to prove their soundness. Moreover, the soundness of further, not yet implemented, combinations of particular ways of clause sharing with particular inprocessing techniques has been shown.

The considered simplifications include variable elimination, equivalence elimination, blocked clause elimination, extended resolution, blocked clause addition and bounded variable addition. For many of these, the preservation of a semantic relation, which we call *unsat-preserving consequence*, has been identified as the crucial property behind soundness. We have inspected four transition systems, SysA, SysB, SysC, and SysD, characterized by different sets of inprocessing rules. The four models allow unrestricted clause sharing among the component solvers running in parallel. SysA models solvers that employ only equivalence preserving techniques. With SysB, we considered parallel solvers that allow unrestricted satisfiability preserving simplifications. Such solvers would be sound with respect to satisfiability, but not refutationally sound, and thus hardly of practical use. SysC covers parallel solvers that utilize inprocessing to *weaken* formulas, for example by removing clauses, and extended resolution. SysD shows the soundness of clause sharing with combinations of clause addition techniques, and clause deletion techniques. To the best of the authors' knowledge such parallel solvers have not yet been considered in the literature. The following table gives a summary of the results proven in this paper, and the currently implemented systems to which they apply.

Presented sharing models with properties and covered systems

Model	*Covered inprocessing techniques*	*Soundness*	*Systems*
SysA	preserve equivalence	✓	ManySAT, PeneLoPe
SysB	preserve equisatisfiability	no	–
SysC	clause deletion techniques	✓	Plingeling
SysD	clause addition and deletion techniques	✓	open

The presented modeling also applies to parallel SAT solvers that follow the guiding path approach. As future work, we plan to extend it to parallel SAT solvers that rely on iterative partitioning [22,23], where clause sharing has also been introduced [24,29]. With the characteristics of SysD, another open question arises: can modern portfolio solvers be improved by adding a single distinguished solver incarnation that performs clause addition techniques?

Acknowledgments The authors would like to thank Armin Biere for insights into the architecture of Plingeling.

References

1. Audemard, G., Hoessen, B., Jabbour, S., Lagniez, J.M., Piette, C.: Revisiting clause exchange in parallel SAT solving. In: Proc. 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '12). LNCS, vol. 7317, pp. 200–213. Springer (2012)
2. Audemard, G., Hoessen, B., Jabbour, S., Lagniez, J.M., Piette, C.: Penelope, a parallel clause-freezer solver. In: SAT Challenge 2012; Solver and Benchmark Descriptions. pp. 43–44 (2012)
3. Audemard, G., Lagniez, J.M., Mazure, B., Saïs, L.: On freezing and reactivating learnt clauses. In: Proc. 14th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '11). LNCS, vol. 6695, pp. 188–200. Springer (2011)
4. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proc. 21st Int. Joint Conf. on Artifical Intelligence (IJCAI '09). pp. 399–404. Morgan Kaufmann (2009)
5. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligence Research 22(1), 319–351 (Dec 2004)
6. Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Report Series Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
7. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Proc. 36th Annual ACM/IEEE Design Automation Conf. (DAC). pp. 317–320. ACM (1999)
8. Böhm, M., Speckenmeyer, E.: A fast parallel SAT-solver – efficient workload balancing. Annals of Mathematics and Artificial Intelligence 17, 381–400 (1996), (Based on a technical report published already in 1994.)
9. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. CACM 5(7), 394–397 (Jul 1962)
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '05). LNCS, vol. 3569, pp. 61–75. Springer (2005)
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '03). LNCS, vol. 2919, pp. 502–518. Springer (2004)
12. Gelder, A.V.: Toward leaner binary-clause reasoning in a satisfiability solver. Annals of Mathematics and Artificial Intelligence 43(1), 239–253 (2005)
13. 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)
14. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving periodic event scheduling problems with SAT. In: Advanced Research in Applied Artificial Intelligence – 25th Int. Conf. on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2012). LNCS, vol. 7345, pp. 166–175. Springer (2012)
15. Guo, L., Hamadi, Y., Jabbour, S., Sais, L.: Diversification and intensification in parallel SAT solving. In: Principles and Practice of Constraint Programming (CP 2010). LNCS, vol. 6308, pp. 252–265. Springer (2010)
16. Hamadi, Y., Jabbour, S., Piette, C., Sais, L.: Deterministic parallel DPLL. JSAT 7(4), 127–132 (2011)

17. Hamadi, Y., Jabbour, S., Sais, L.: Control-based clause sharing in parallel SAT solving. In: Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 499–504 (2009)
18. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel SAT solver. JSAT 6(4), 245–262 (2009)
19. Heule, M., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In: Hardware and Software: Verification and Testing – 7th Int. Haifa Verification Conf. (HVC 2011). LNCS, vol. 7261, pp. 50–65. Springer (2012)
20. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Proc. 14th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '11). LNCS, vol. 6695, pp. 201–215. Springer (2011)
21. Hölldobler, S., Manthey, N., Nguyen, V., Stecklina, J., Steinke, P.: A short overview on modern parallel SAT-solvers. In: Proc. Int. Conf. on Advanced Computer Science and Information Systems (ICACSIS 2011). pp. 201–206. IEEE (2011)
22. Hyvärinen, A.E.J., Junttila, T., Niemelä, I.: Partitioning SAT instances for distributed solving. In: Proc. 17th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning. pp. 372–386. LPAR'10, Springer (2010)
23. Hyvärinen, A.E.J., Manthey, N.: Designing scalable parallel SAT solvers. In: Proc. 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '12). LNCS, vol. 7317, pp. 214–227. Springer (2012)
24. Hyvärinen, A.E.J., Junttila, T.A., Niemelä, I.: Grid-based SAT solving with iterative partitioning and clause learning. In: Principles and Practice of Constraint Programming (CP 2011). LNCS, vol. 6876, pp. 385–399. Springer (2011)
25. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 6015, pp. 129–144. Springer (2010)
26. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Proc. 6th Int. Joint Conf. on Automated Reasoning (IJCAR 2012). LNCS, vol. 7364, pp. 355–370. Springer (2012)
27. Katebi, H., Sakallah, K.A., Marques-Silva, J.P.: Empirical study of the anatomy of modern SAT solvers. In: Proc. 14th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '11). LNCS, vol. 6695, pp. 343–356. Springer (2011)
28. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96–97(1), 149–176 (Oct 1999)
29. Lanti, D., Manthey, N.: Sharing information in parallel search with search space partitioning. In: Learning and Intelligent Optimization – 7th Int. Conf. (LION 7) (2013), to appear
30. Lynce, I., Marques-Silva, J.P.: Probing-based preprocessing techniques for propositional satisfiability. In: Proc. 15th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI '03). pp. 105–. IEEE (2003)
31. Manthey, N.: Coprocessor 2.0: a flexible CNF simplifier. In: Proc. 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '12). LNCS, vol. 7317, pp. 436–441. Springer (2012)
32. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of Boolean formulas. In: Hardware and Software: Verification and Testing – 8th Int. Haifa Verification Conf. (HVC 2012). LNCS, Springer (2013), to appear
33. Marques-Silva, J.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5), 506–521 (1999)
34. Martins, R., Manquinho, V., Lynce, I.: An overview of parallel SAT solving. Constraints 17(3), 304–347 (2012)

35. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. 38th Annual ACM/IEEE Design Automation Conf. (DAC). pp. 530–535. ACM (2001)
36. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence 175(2), 512–525 (Feb 2011)
37. Roussel, O.: Description of ppfolio 2012. In: Proc. SAT Challenge 2012; Solver and Benchmark Descriptions, p. 46. Univ. of Helsinki (2012), `http://hdl.handle.net/10138/34218`
38. Soos, M.: Cryptominisat 2.5.0. In: SAT Race Competitive Event Booklet (July 2010), `http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_13.pdf`, retrieved 11 February 2013
39. Sörensson, N., Biere, A.: Minimizing learned clauses. In: Proc. 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '09). LNCS, vol. 5584, pp. 237–243. Springer (2009)
40. Subbarayan, S., Pradhan, D.K.: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In: Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '04). LNCS, vol. 3542, pp. 276–291. Springer (2005)
41. Tseitin, G.S.: On the complexity of derivations in the propositional calculus. Studies in Mathematics and Mathematical Logic Part II, 115–125 (1968)
42. Wotzlaw, A., van der Grinten, A., Speckenmeyer, E., Porschen, S.: pfoliouzk: Solver description. In: Proc. SAT Challenge 2012; Solver and Benchmark Descriptions. p. 45. Univ. of Helsinki (2012), `http://hdl.handle.net/10138/34218`
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)