Deep Combination of CDCL(T) and Local Search for Satisfiability Modulo Non-Linear Integer Arithmetic Theory

Xindi Zhang
Bohan Li
Shaowei Cai*
{zhangxd,libh,caisw}@ios.ac.cn
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology, University of Chinese Academy of Sciences
Beijing, China

ABSTRACT

Satisfiability Modulo Theory (SMT) generalizes the propositional satisfiability problem (SAT) by extending support for various first-order background theories. In this paper, we focus on the SMT problems in Non-Linear Integer Arithmetic (NIA) theory, referred to as SMT(NIA), which has wide applications in software engineering. The dominant paradigm for SMT(NIA) is the CDCL(T) framework, while recently stochastic local search (SLS) has also shown its effectiveness. However, the cooperation between the two methods has not been studied yet. Motivated by the great success of the deep cooperation of CDCL and SLS for SAT, we propose a two-layer hybrid approach for SMT(NIA). The outer-layer interleaves between the inner-layer and an independent SLS solver. In the inner-layer, we take CDCL(T) as the main body, and design DCL(T)-guided SLS solver, which is invoked at branches corresponding to skeleton solutions and returns useful information to improve the branching heuristics of CDCL(T). We implement our ideas on top of the CDCL(T) tactic of Z3 with an SLS solver called LocalSMT, resulting in a hybrid solver dubbed HybridSMT. Extensive experiments are carried out on the standard SMT(NIA) benchmarks from SMT-LIB, where most of the instances are from real-world software engineering applications of termination and non-termination analysis. Experiment results show that HybridSMT significantly improves the CDCL(T) solver in Z3. Moreover, our solver can solve 10.36% more instances than the currently best SMT(NIA) solver, and is more efficient for software verification instances.

KEYWORDS

SMT(NIA), CDCL(T), Local Search, Hybrid Method

ACM Reference Format:
Xindi Zhang, Bohan Li, and Shaowei Cai. 2024. Deep Combination of CDCL(T) and Local Search for Satisfiability Modulo Non-Linear Integer Arithmetic Theory. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639105

*Corresponding author.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3639105

1 INTRODUCTION

Satisfiability modulo theories (SMT) is the problem that asks the satisfiability of a given first-order logic formula with respect to some background theories, and the formula is usually quantifier-free. Typical SMT theories include integers, real numbers, arrays, bit-vectors, and strings. In this paper, we focus on the SMT problem under the theory of *Non-Linear Integer Arithmetic* (NIA), which is referred to as SMT(NIA). SMT(NIA) has a wide range of applications, such as neural network verification [40], model checking [39], game theory [5], and program synthesis [64].

SMT(NIA) solvers serve as an important reasoning engine in many software engineering applications, such as symbolic execution [54] and hybrid system analysis [56, 58]. Specifically, in terms of state reachability checking [38], invariant generation [26], and (non-)termination prove problem [27, 48], a popular approach is to assume a linear template for the invariant or ranking function, and then the Farkas lemma is adopted to eliminate the universal quantifiers. Such approaches are denoted as Invariant synthesis and ranking function synthesis [15], where the Farkas lemma could introduce non-linear terms in the form of $x \times y$ [26].

With the support of certain background theories, SMT can be seen as the generalization of the propositional satisfiability (SAT) problem. Motivated by the success of the conflict-driven clause learning (CDCL) framework of SAT solving, SMT solvers attempt to utilize or generalize the advanced techniques of SAT solvers from propositional logic to the fragment of first-order logic.

Generally, the SMT(NIA) problem is undecidable. Indeed, Matiyasevich's theorem [28] shows that Hilbert's 10th problem is impossible by proving the satisfiability of NIA is theoretically undecidable. Therefore, all algorithms for SMT(NIA) are incomplete. Nevertheless, from the viewpoint of practical solving, many SMT(NIA) problems arising from applications can be solved by carefully designed solvers. Considerable efforts have been made and many effective approaches have been developed for solving SMT(NIA), which can produce sound results for many practical SMT(NIA) problems.

CDCL(T) [59] is a representative framework for solving SMT on many theories, including the NIA theory. It abstracts the given SMT formula to a Boolean skeleton and uses a CDCL-based SAT solver to deal with the skeleton. Meanwhile, a theory-related decision procedure, which is also called the *theory solver*, is used to

refine the Boolean skeleton and guide the SAT solver (by learning new atoms or clauses). It is also known as the *lazy* method or the DPLL(T) framework. Almost all state-of-the-art SMT solvers have implemented the CDCL(T) approach, including Z3 [29], CVC5 [2], Yices2 [32], MathSAT5 [25], and SMTInterpol [22].

Stochastic Local Search (SLS) is another powerful paradigm for SAT, and recently has witnessed success in solving SMT(NIA). In 2022, Cai et. al. developed the first SLS solver for the integer arithmetic theories [18]. The SLS solver directly operates on the variables rather than on the extracted Boolean skeleton. It begins with a complete assignment and modifies the values of some variables iteratively until a model is found or the time limit is reached. The SLS solver outperforms existing methods on some benchmarks of satisfiable instances encoded from software verification problems.

Rather than CDCL(T) and SLS, various approaches have been introduced to solve SMT(NIA), such as the eager approach (also called bit-blasting) [16, 35], the model-construction satisfiability (MCSat) calculus (or NLSAT algorithm [30, 43–45]), and the linearization based methods [13, 14, 24].

It seems that the latest remarkable progress in SAT solving is the development of hybrid solvers by deep cooperation of CDCL and SLS [20, 21]. In a word, the hybrid method is mainly a CDCL solver, and invokes an SLS solver according to contextual information during the CDCL process. An important feature of the hybrid method is that CDCL and SLS communicate information between each other. Almost all state-of-the-art SAT solvers (KISSAT [33], CaDiCaL [7], Relaxed [65, 66],CryptoMiniSat [61]) are hybrid solvers of this type.

Yet, we are unaware of any hybrid SMT(NIA) method combining CDCL(T) and SLS, although portfolios simply call a CDCL(T) solver and an SLS solver sequentially have shown some benefits [17, 18]. Motivated by the great success of hybrid SAT solvers, this work is devoted to designing an efficient hybrid SMT(NIA) solver by the deep cooperation of CDCL(T) and SLS.

We concentrate on three central questions and offer responses to them during algorithm development.

Q1: How to schedule SLS and CDCL(T) reasonably?

R: A hybrid framework for SMT(NIA) (Sect. 3): We design a two-layer hybrid method. In the outer-layer, our method interleaves between the inner-layer solver and an independent SLS. The inner-layer takes CDCL(T) as the main body, and a CDCL(T)-guided SLS solver is invoked at branches corresponding to skeleton solutions; the CDCL(T) solver and the guided SLS communicate information between each other.

Q2: How to use CDCL(T) to guide SLS?

R: CDCL(T)-guided SLS at promising branches (Sect. 4): When CDCL(T) reaches an assignment that satisfies the Boolean skeleton of the input formula, we extract a sub-formula w.r.t. the assignment and invoke SLS to work at the extracted formula. If SLS finds a model, the original formula is satisfiable. If the SLS fails to find a model within the time limit, the CDCL(T) procedure continues from the breakpoint where the SLS enters. We filter out the SLS calls using a heuristically *difference score* to prevent two neighboring SLS from searching in the same search space.

Q3: How to exploit information from SLS to enhance CDCL(T)?

R: Improving CDCL(T) branching heuristics with SLS assignment and conflict frequency (Sect. 5): Branching is one of the most crucial components of CDCL(T), which is concerned with variable ordering and phase selection. We design a phase resetting method using the best assignment in recent CDCL(T)-guided SLS calls, i.e., the one with the fewest falsified clauses. We also employ the conflict frequency of atoms to enhance the variable ordering heuristic of CDCL(T).

A hybrid solver named HybridSMT is implemented on top of the CDCL(T) tactic of Z3 [29] with the cooperation of a SLS solver called LocalSMT [20]. HybridSMT is compared with state-of-the-art SMT(NIA) solvers on benchmarks from SMTLIB. The majority of the benchmarks are focused on the termination/non-termination verification in software engineering. Extensive experiments show that a significant advantage in the number of solved instances and the run time. Specifically, compared to Z3, which is known as the currently best solver for SMT(NIA), HybridSMT solves more instances, and is overall 3.79 times faster on the instances solved by both solvers. A further improved solver named Hybrid+Z3 has been developed by integrating HybridSMT into Z3, allowing Z3 to solve 10.36% more instances with a 52.76% run time reduction.

The rest of this paper is organized as follows: Section 2 gives the preliminaries and backgrounds. Sections 3–5 introduces the framework of our method and the main techniques. Section 6 shows the effectiveness of our methods and performs detailed assessments. Section 7 supplements more details for the related works. Finally, section 8 concludes this paper.

2 PRELIMINARIES

This section introduces some preliminary knowledge, including basic definitions and symbols of SMT(NIA), a representative CDCL(T) implementation in Z3 [29], and a typical stochastic local search framework shown in LocalSMT [18].

2.1 Preliminaries of SMT(NIA)

2.1.1 The Propositional Satisfiability Problem (SAT). Given a set of propositional (Boolean) variables $P = \{p_1, p_2, ..., p_n\}$, a propositional *literal* is either a variable p_i or its negation $\neg p_i$. A *clause* $C = l_1 \vee l_2 \vee ... \vee l_q$ is the disjunction of a set of literals. The Conjunctive Normal Form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ is the conjunction of a set of clauses.

The *assignment* for a CNF is a mapping $\alpha : P \rightarrow \{False, True\}$. SAT is the problem of deciding whether there is an assignment, under which a given CNF formula evaluates to true.

2.1.2 Non-Linear Integers Arithmetic (NIA). The expression of the form $x_1^{d_1} x_2^{d_2} ... x_n^{d_n}$ is *monomial*, where x_i are distinct (arithmetic) variables, d_i are the (positive integer-valued) degree of x_i for all $i \in \{1, ..., n\}$. A monomial is linear if $n = 1$ and $d_1 = 1$; otherwise, it is non-linear. The expression of the form $\Sigma_i a_i m_i$, where each m_i is a monomial, is *polynomial*. A polynomial is linear if the monomials that form the polynomial are all linear; otherwise, it is non-linear.

A *polynomial inequality*, is an expression of the form $\sigma : \Sigma_i a_i m_i \leq k$, where a_i and k are constant coefficients. σ is an NIA inequality if all the arithmetic variables in σ are integer-valued arithmetic variables; σ is a Linear Integer Arithmetic (LIA) inequality if σ is a special NIA inequality that the polynomial is linear.

With the Boolean operators ($\neg, \vee, \wedge$), the other inequalities ($<, >, \geq, \neq$) and equality ($=$) can be expressed. In detail, $\Sigma_i a_i m_i < k$

can be expressed as $\Sigma_i a_i m_i \leq k-1$; $\Sigma_i a_i m_i > k$ as $\neg(\Sigma_i a_i m_i \leq k)$; $\Sigma_i a_i m_i \geq k$ as $\neg(\Sigma_i a_i m_i \leq k-1)$; $\Sigma_i a_i m_i = k$ as $(\Sigma_i a_i m_i \leq k) \wedge \neg(\Sigma_i a_i m_i \leq k-1)$; $\Sigma_i a_i m_i \neq k$ as $(\Sigma_i a_i m_i \leq k-1) \vee \neg(\Sigma_i a_i m_i \leq k)$.

2.1.3 Satisfiability Modulo theories (SMT). SMT extends SAT with certain background theories. We focus on the SMT problem with quantifier-free NIA background theory in this paper. The *atom* of an SMT(NIA) formula is either a propositional variable (Boolean atom) or a polynomial inequality (NIA atom). SMT(NIA) extends the propositional variables P with integer arithmetic variables $X = \{x_1, x_2, ..., x_m\}$; extends the propositional literal with arithmetic literals, i.e., NIA atoms or the negation of NIA atoms.

The *assignment* of SMT(NIA) is a mapping $\alpha : P \rightarrow \{False, True\}$, $X \rightarrow \mathbb{Z}$, and α is a *complete* assignment if it maps all variables to a value; otherwise, α is a *partial* assignment. We use $\alpha(x)$ to denote the value of a variable x under α. A literal is a true (false) literal if it is evaluated to be true (false) under a given assignment; otherwise it is an undecided literal. A clause is *satisfied* if it has at least one true literal, *falsified* if all the literals are false; otherwise, it is an *unresolved* clause. A *level* consists of a heuristically decided variable along with the deduced variables.

The *(Boolean) skeleton* of an SMT formula F is to replace each atom σ in F by a unique Boolean variable p_σ, which is called the *(Boolean) encoder* of σ.

Example 2.1. Given a set of integer arithmetic variables $X = \{x_1, x_2, x_3, x_4\}$ and a Boolean variables set $P = \{p_1, p_2\}$, $F_{SMT(NIA)}$ is a typical CNF formula of SMT(NIA). S_F is the Boolean skeleton of $F_{SMT(NIA)}$ by introducing three new Boolean variables (p_{σ_1}, p_{σ_2}, and p_{σ_3}) for the NIA atoms ($3x_1x_2 \leq 2$, $-x_2 - 3x_4 \leq 0$, and $2x_3^3x_4 + 4x_2 + x_1 \leq 8$) respectively.

$$\begin{aligned} F_{SMT(NIA)} &= (p_1 \vee \neg p_2) \\ &\wedge (\neg(3x_1x_2 \leq 2) \vee (-x_2 - 3x_4 \leq 0)) \\ &\wedge (p_2 \vee (2x_3^3x_4 + 4x_2 + x_1 \leq 8)) \end{aligned} \tag{1}$$

$$S_F = (p_1 \vee \neg p_2) \wedge (\neg p_{\sigma_1} \vee p_{\sigma_2}) \wedge (p_2 \vee p_{\sigma_3}) \tag{2}$$

2.2 Typical CDCL(T) Solvers

In the CDCL(T) framework, a SAT solver in the conflict-driven clause learning (CDCL) framework is used to reason about the Boolean structure and solve the Boolean formula, while a theory solver receives assignments from the SAT solver and solves the conjunctions of NIA atoms, including consistency checking of the assignments and the theory-based deduction.

Algorithm 1 describes a typical CDCL(T) framework, which is implemented in Z3 [29]. After loading the formula F, the Boolean skeleton S_F of F is extracted, and the assignment α_S for S_F is maintained throughout the whole process of CDCL(T).

In line 2, $propagate()$ deduces the assignment of unassigned variables of S according to two reasoning techniques: *Boolean constraint propagation* deduces directly from current α_S; *theory propagation* first constructs a T-theory formula according to currently assigned encoders and then performs reasoning with theory solvers of T-theory. Note that new atoms and their encoders may be implied and S_F is updated accordingly.

Algorithm 1: A Typical CDCL(T) Framework

```
INPUT  : An SMT formula F
OUTPUT : 'SAT' or 'UNSAT'
1 while True do
2     c ← propagate();
3     if c ≠ ∅ then
4         l ← resolve_conflict(c);
5         if l < 0 then return 'UNSAT';
6         backtrack(l);
7     else
8         if !decide() then return 'SAT';
```

Once a conflict is found in the propagation process, a new learnt clause is derived and added to the clause database (line 4). At the same time, the backtracking level l is deduced and the assignment of some conflict-related variables is canceled (lines 4–6). If the conflict is derived without heuristically assigned variables, that is $l < 0$, it returns 'UNSAT' (line 5). Note that the number of conflicts encountered is usually used as a metric to estimate the runtime.

On the other hand, if no further implications can be made in the propagation process and there are no conflicts under current α_S, an unassigned variable will be picked according to *branching* heuristics and assigned according to *phase selection* heuristics. Once a conflict-free complete assignment is found, it returns 'SAT' (line 8). More details of CDCL(T), such as clause learning and theory solvers, can be found in [10, 47].

Variable State Independent Decaying Sum (VSIDS) [51]. Here we introduce the variant in Z3 [29], which is the most popular version for CDCL and CDCL(T). It associates each Boolean variable with an *activity* score. Once a variable is involved in a conflict, its activity is *increased* with a float number act_{inc}. As the number of conflicts increases, act_{inc} will exponentially increase. It picks the unassigned variable with the maximum activity when selecting a branching variable.

Phase Selection Heuristics. When a Boolean variable is selected, the assignment should be decided. The *phase saving* technique [55] assigns the variable to the assignment that it has just assigned, and the *uncached* variables, which are the variables that have not been assigned before, will be assigned to a default value or assigned randomly. The *rephase* technique [21] periodically sets the phases according to local search candidate solutions [20] or target phases [33]. When the input file is an NIA instance, Z3 [29] switches between two phase selection strategies. For every 800 conflicts, it turns on the phase saving technique [55] and assigns $False$ for the uncached variables for the first 700 conflicts; and it always assigns $True$ for any heuristically chosen variables for the last 100 conflicts.

Restarts. Restart is a commonly used method in SAT solvers to avoid being stuck in a bad branching direction and migrate the heavy-tailed phenomenon of runtime [36]. Similarly, the CDCL(T) algorithm of Z3 uses the geometric inner/outer strategy [9].

Algorithm 2: A Typical SLS Framework

```
   INPUT  : An SMT formula F
   OUTPUT : A Model α
1  α ← initialize();
2  while not meet terminate condition do
3      if F is satisfied under α then return α;
4      op ← choose_operation();
5      perfom(op, α);
6      collect(); /* For hybrid algorithm          */
```

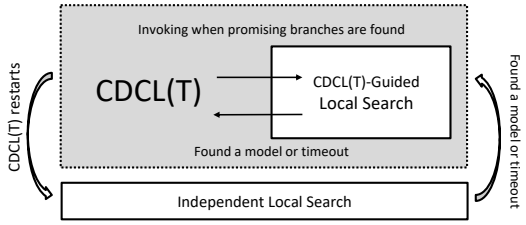


Figure 1: The CDCL(T) and SLS hybrid framework

2.3 Typical Local Search Solvers

Given a formula F, the set of all possible complete assignments forms the *search space*. A complete assignment α is a *candidate solution*, and its *cost*, denoted as $cost(\alpha)$, is the number of falsified clauses under α. α is a *solution* (or *model*) if and only if $cost(\alpha) = 0$.

A typical Stochastic Local Search (SLS) solver maintains a candidate solution α, which is iteratively modified until it becomes a solution or a pre-set termination condition is met.

A representative SMT(NIA) SLS solver is implemented in LocalSMT [18], whose framework is shown in Algorithm 2. LocalSMT constructs the initial candidate solution α by mapping all the Boolean variables to *False* and the arithmetic variables to zero (line 2). In the main loop, an *operation*, a variable should be assigned with which value, is chosen and performed, trying to decrease the cost of the current α (lines 4–5).

In this paper, we modify the local search to collect the information after each step (line 6). The termination of a local search call is determined by the number of non-improved steps (*nImpr*), which is controlled by a hyper-parameter θ. The local search procedure terminates when $nImpr > \theta$, unless it finds a model before that.

3 A HYBRID FRAMEWORK FOR SMT(NIA)

We design a hybrid framework for SMT(NIA), which not only exploits the complementary ability of CDCL(T) and SLS solvers, but also improves both solvers by communicating information. In this framework, we have one CDCL(T) solver and two SLS solvers. The two SLS solvers have different roles, and we call one of them the *Independent SLS* solver, and the other the *CDCL(T)-guided SLS* solver.

Overall, our method works in two layers, as shown in Figure 1. In the outer-layer, our solver can be seen as a sequential portfolio solver, which alternates between the hybrid solver described in the inner-layer and the Independent SLS solver. In the inner-layer, the CDCL(T) solver is the main body, and it interacts with the CDCL(T)-guided SLS solver, communicating information with each other. Note that the outer-layer is simply dedicated to making use of the complementary performance between CDCL(T) and SLS, while the inner-layer, corresponding to a hybrid solver that deeply combines CDCL(T) and SLS, is the main contribution of this work.

In the following, we describe the outer-layer of our method. The details of the inner-layer, including the CDCL(T)-guided SLS and the details about improving CDCL(T) using SLS information, will be introduced in Sections 4 and 5.

In the outer-layer, the Independent SLS is invoked only right after a restart of CDCL(T). Specifically, the condition to invoke Independent SLS follows a rule that SLS is called less and less often. The 1^{st} call of Independent SLS happens at the first restart of CDCL(T), the 2^{nd} happens after q more restarts since the 1^{st} call, the 3^{rd} happens after q^2 more restarts since the 2^{nd} SLS call. Generally, the i^{th} Independent SLS call is invoked when q^{i-1} restarts have occurred after the $(i-1)^{th}$ call. In this paper, q is set to 2.

In this work, we adopt the LocalSMT solver as the Independent SLS solver. Each time the Independent SLS is activated, it works on the full formula without any guidance from CDCL(T), aiming to find a model within the limit on the search steps, which is controlled by a parameter θ. We set θ to $3 \times 10E4 + (i-1) \times 10E4$, with a upper bound of $3 \times 10E5$.

If Independent SLS returns 'SAT', then the given formula is satisfiable; otherwise, when the step limit of each Independent SLS call is reached, the algorithm switches to the CDCL(T) process in the inner layer. The unsatisfiability of a formula can only be proven by the CDCL(T) process.

Intuitions on the settings of outer-layer. At the end of this section, we try to provide intuitions to the invoking and termination policies of Independent SLS. (1) In order to solve the instances for which one of the two solvers (CDCL(T) and LocalSMT) excels, the time slice for each solver should begin small and increase as progresses. Therefore, the step limit for Independent SLS gradually increases. (2) SLS exhibits the cycling phenomenon, i.e., it visits the space it has previously searched, as it does not record the previous history, whereas the learned lemmas assist CDCL(T) in pruning the search space. Thus, as the solving procedure progresses, it is reasonable to gradually reduce the proportion of time slots for the independent SLS and allocate more time for the CDCL(T) part (indeed the hybrid solver of the inner-layer).

4 CDCL(T)-GUIDED LOCAL SEARCH

The inner-layer is a hybrid solver that cooperates a CDCL(T) solver and an SLS solver in both directions: CDCL(T) guides SLS, while SLS gives information back to CDCL(T). This section introduces how CDCL(T) guides local search.

4.1 Extracting Sub-formula from Skeleton Solution

Recall that a CDCL(T) solver consists of a CDCL-based SAT solver and a theory solver. The SAT solver deals with the Boolean skeleton, where each NIA atom is encoded into a Boolean variable which is called an *encoder*. A *solution* to the skeleton, which we refer to as *skeleton solution*, is a (maybe partial) assignment to the Boolean variables, including both Boolean encoders and original Boolean

Algorithm 3: The Hybrid Framework

```
INPUT  : An SMT formula F
OUTPUT : 'SAT' or 'UNSAT'
/* Codes related to promising branches (Sect. 4)
   are colored brown; codes to SLS information
   (Sect. 5) are colored red.                    */
1  r ← 0, p ← 0;
2  ds ← 0, g_ds ← ϵ;
3  while True do
4      c ← propagate();
5      if c ≠ ∅ then
6          l ← resolve_conflict(c);
7          if l < 0 then return 'UNSAT';
8          backtrack(l);
9          if ∃ unknown core clauses then
10             ds ← ds + 1
11     else
           /* Promising branches in Section 4      */
12         if reach a skeleton solution and ds > g_ds then
13             CDCL(T)_guided_SLS();
               /* Update phase and order in Sect.5 */
14             SLS_update_phase_order();
15             ds ← 0, g_ds ← g_ds + 1;
16         if !decide() then return 'SAT';
17         if meet restart condition then
18             restart();
19             SLS_rephase(); /* Related to Sect. 5 */
20             if (r++) ≥ (1 − q^p)/(1 − q) then
21                 independent_SLS();
22                 SLS_update_order(); /* CF in Sect.5 */
23                 p ← p + 1;
24             ds ← 0, g_ds ← ϵ;
```

variables, that satisfies the Boolean skeleton of the initial formula, not including those added encoders and clauses learned during CDCL(T). In Z3, the original clauses (may be simplified before solving) are also called *core clauses*.

The CDCL(T)-guided SLS solver is invoked when a solution to the skeleton is identified by the CDCL(T) solver, along with a filtering condition that will be introduced in the next subsection.

When the CDCL(T) solver meets the condition to call the SLS solver, a sub-formula is extracted from the original formula, by only keeping the true literals. It is obvious that if the resulting sub-formula is satisfiable, then the original SMT(NIA) formula is satisfiable as well. This extracted sub-formula is fed to the SLS solver. In this sense, we say the SLS is guided by the CDCL(T) solver, as it works on the sub-formula induced from the skeleton solution found by CDCL(T).

Example 4.1. Given an input SMT(NIA) formula $F_{SMT(NIA)} = (p_1 \vee \neg p_2) \wedge (\neg(3x_1x_2 \leq 2) \vee (-x_2 - 3x_4 \leq 0)) \wedge (p_2 \vee (2x_3^3x_4 + 4x_2 + x_1 \leq 8))$, and its Boolean skeleton is $S_F = (p_1 \vee \neg p_2) \wedge (\neg p_{\sigma_1} \vee p_{\sigma_2}) \wedge (p_2 \vee p_{\sigma_3})$. A partial assignment $\alpha = \{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F, p_2 \rightarrow F, p_{\sigma_3} \rightarrow T\}$ is a solution to the Boolean skeleton as all the clauses in S_F are satisfied. From this branch, a sub-formula of the skeleton $S'_F = (p_1 \vee \neg p_2) \wedge (\neg p_{\sigma_1}) \wedge (p_{\sigma_3})$ can be extracted by removing the false literals and the undecided literal. By replacing the encoders with the related NIA atoms, we get an SMT(NIA) sub-formula $F_{SLS} = \{(p_1 \vee \neg p_2) \wedge \neg(3x_1x_2 \leq 2) \wedge (2x_3^3x_4 + 4x_2 + x_1 \leq 8)\}$.

Note that when extracting a sub-formula to feed to local search, we only focus on the original formula. We do not send the learnt clauses to the SLS solver, as this would bring a heavy burden to SLS — the run time performance of LocalSMT is closely related to the number of clauses. Experiments on sending good-quality clauses with Literal Block Distance [1] no larger than 6, as typical SAT solvers [20], show similar performance.

4.2 Pruning SLS Calls with Difference Score

Now we know that the SLS solver will be invoked if a solution to the Boolean skeleton is reached. However, if we call SLS at every solution to the skeleton, there would be too many SLS calls, many of which work on similar or even the same sub-formulas.

Example 4.2. Given an SMT(NIA) formula $F_{SMT(NIA)} = (p_1 \vee \neg p_2) \wedge (\neg(3x_1x_2 \leq 2) \vee (-x_2 - 3x_4 \leq 0)) \wedge (p_2 \vee (2x_3^3x_4 + 4x_2 + x_1 \leq 8))$, where the first two clauses are origin clauses and the last clause is a learned lemma, and the skeleton of $F_{SMT(NIA)}$ is $S_F = (p_1 \vee \neg p_2) \wedge (\neg p_{\sigma_1} \vee p_{\sigma_2}) \wedge (p_2 \vee p_{\sigma_3})$. Here are three consecutive solutions, i.e., α_1, α_2, and α_3, for the Boolean skeleton S_F: $\alpha_1 = \{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F\}$, $\alpha_2 = \{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F, p_{\sigma_3} \rightarrow F\}$, $\alpha_3 = \{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F, p_{\sigma_3} \rightarrow F\}$. From the three solutions, we can extract the same sub-formula $F_{sub} = \{p_1 \wedge \neg(3x_1x_2 \leq 2)\}$ and send it to SLS solver. On the other hand, from the solution $\alpha_4 = \{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F, p_2 \rightarrow F\}$, a different sub-formula $F'_{sub} = \{(p_1 \vee \neg p_2) \wedge \neg(3x_1x_2 \leq 2)\}$ can be extracted.

As shown in Example 4.2, in the CDCL(T) process, there can be a cluster of skeleton solutions that have similar sub-formulas or even share the same sub-formula. Our intention is to call SLS with different sub-formulas, as the algorithmic behavior and the collected information tend to be similar when SLS works on similar sub-formulas. Therefore, we must make sure that the sub-formulas of two adjacent SLS calls have considerable differences.

As the number of Boolean variables of the skeleton is almost the same as the number of the arithmetic literals of the CNF for most of the instances, directly counting the number of skeleton variables with different assignments needs to traverse the CNF. Thus, we design a method to heuristically measure the differences by monitoring the backtracking behaviors of CDCL(T).

When backtracking, the value of at least one skeleton variable will be flipped. Therefore, search areas separated by several restarts would not be too close. We use the *difference score*, denoted as *ds*, to measure the number of backtracks that occurred since the last call of the CDCL(T)-guided SLS solver. Nevertheless, due to the newly learned encoders during the CDCL(T) procedure, backtracking may not lead to the flipping of a skeleton variable from the initial skeleton. Thus, we only count the number of backtracks after which the assignment is not a skeleton solution.

If CDCL(T) reaches a skeleton solution, and at that time, ds is greater than threshold g_{ds}, we invoke the CDCL(T)-guided SLS solver. As shown in Algorithm 3, g_{ds} is initialized to ϵ, which is increased by 1 as the number of CDCL(T)-guided SLS calls increases. $\epsilon = 3$ in this paper.

The CDCL(T)-guided SLS, as with the Independent SLS, also uses the local search engine LocalSMT. Compared to the Independent SLS, the formula fed to the CDCL(T)-guided SLS is usually much smaller, and the frequency of being called is higher. Thus, the step limit for CDCL(T)-guided SLS is set to be smaller than that for Independent SLS. Moreover, if the previous CDCL(T)-guided SLS cannot find a solution, it is likely that the following SLS calls before the next restart of CDCL(T) also fail. Based on these considerations, the step limit θ for the CDCL(T)-guided SLS is set as follows: for each CDCL(T) restart, θ is initialized to $3 \times 10E4$ (the minimum number of the Independent SLS), then it is decreased by $1 \times 10E4$ after each call, with a lower bound of $1 \times 10E4$.

5 ENHANCING CDCL(T) WITH SLS INFORMATION

In this section, we introduce how to improve CDCL(T) by exploiting SLS information. The performance of CDCL(T) is significantly influenced by the branching heuristics, including the variable ordering heuristic and the phase selection heuristic. In the CDCL(T) tactic of Z3, the variable ordering heuristic is VSIDS [51], and the phase selection heuristic relies on the phase saving technique [55]. This section introduces how to improve these two heuristics of CDCL(T) by using information from SLS.

Recent works on hybrid SAT solving have shown that CDCL solvers can gain significant improvements by using SLS information including variable assignments and conflict frequency of variables to enhance the branching heuristics [21]. However, such techniques work on a Boolean level and cannot be directly applied in the context of SMT. This section lifts such techniques from SAT (Boolean level) to word level of SMT.

5.1 Phase Selection with SLS Assignments

CDCL(T) solvers take CDCL as the reasoning engine to deal with the abstracted Boolean skeleton. Thus, the heuristics of the CDCL engine have a significant impact on the performance of the CDCL(T) solver. An important component for modern CDCL SAT solvers and CDCL(T) SMT solvers is phase selection.

Recently, effective phase selection heuristics have attracted much attention in the SAT-solving direction. Biere et al. proposed a phase resetting method that periodically picked the phase based on the target phases [8]; Cai et al. proposed a phase resetting method that periodically resets the phase based on local search assignment and won the best paper of the SAT conference [20].

We propose a phase resetting method based on the assignments obtained by the CDCL(T)-guided SLS. The SLS solver LocalSMT maintains an assignment to the variables (both original Boolean variables and integer variables) rather than to the Boolean variables abstracted from the skeleton. Therefore, we need to translate the SLS assignments into the assignments for the abstracted Boolean skeleton, so that they can be used in the CDCL(T) engine.

Phase Type	α_{sls}	α_T	α_F	α_{rnd}	α_{rev}	No Change
Probability	50%	10%	10%	10%	10%	10%

Table 1: Probability Distribution For Each Phase. Half for intensification (α_{SLS}) and half for diversification (the rest).

SLS solvers work in the space of complete assignments. For any complete word-level assignment α, we can easily figure out the truth value of all atoms ($True$ or $False$) in the input formula F. Thus, we can obtain a translated Boolean-level assignment for the skeleton S_F by assigning the Boolean encoders to the value of the corresponding atom. We trivially extend the translated assignment to cover the encoders that are newly learned during CDCL(T) and not sent to SLS by marking them as "unknown".

As described in Algorithm 3, we use the translated assignments from CDCL(T)-guided SLS for phase resetting. To do so, we record the assignment with the minimum *cost* during the latest call of the SLS solver, and its corresponding translated assignment is also recorded, denoted as α_{sls}, to emphasize this Boolean-level assignment is obtained from SLS. When using α_{sls} to reset the abstracted Boolean variables' phases, the phases are assigned randomly for the "unknown" variables in the translated assignment. The objective of phase resetting with α_{sls} is to focus on the most promising branches under the guide of SLS. To prevent searching similar search space, we introduce the diversity phases as follows.

- Always $True$/$False$ (α_T/α_F). Rewrite the saved phases for each variable to $True$/$False$.
- Randomly (α_{rnd}). Rewrite the saved phases for each variable to a random value from $\{True, False\}$.
- Reverse α_{rev}. Reverse the saved phases for each assigned variable from $True$ (resp., $False$) to $False$ (resp., $True$).
- No Change. Skip the current phase resetting process.

When phase resetting after each restart, we choose the phase following the probability distribution as Table 1. We tuned the parameters manually with a minimum interval of 5%. The performance deteriorates significantly (solving more than 100 fewer instances) when the probability for α_{SLS} is small. The settings for the other diversification phases have robustness, as pre-experiments show that different settings have little impact on the performance (within 30 instances).

5.2 Variable Ordering with SLS Conflict Frequency

Another important heuristic in CDCL/CDCL(T) is the variable ordering heuristic, which is used to pick a variable to branch. Although proposed two decades ago, VSIDS [51] is still one of the most powerful ordering methods. The underlying idea is to pick a variable that is likely to lead to a conflict quickly, and thus produce lemmas to prune the search space.

We propose to enhance the VSIDS of the CDCL(T) tactic in Z3 by utilizing the concept of *conflict frequency*.

Definition 1. *In an SMT SLS process with n steps, the conflict frequency of an atom a, denoted as $cf(a)$, is the number of steps in which it appears in at least one falsified clause divided by n.*

The current scoring function of the branching heuristics (e.g., VSIDS) of CDCL(T) is to estimate the possibility of meeting a conflict after picking each variable. The higher the score for a variable, the greater the probability that it will appear in a conflict. The conflict frequency is a method for calculating the proportion of occurrences of conflicts for a given variable, which can be regarded as a supplement to the scoring function of the branching heuristics of CDCL(T) to some extent.

As shown in the Algorithm 3, after each Independent SLS and every δ CDCL(T)-guided SLS, for each variable a in the skeleton of the input formula, the activity of a is increased by $\zeta \times cf(a)$, where ζ is a parameter whose goal is to balance between the influence of $cf(a)$ collected from SLS and the conflict information collected by CDCL(T). In this paper, $\delta = 20$ and $\zeta = 200$. Note that we do not set δ to 1, because the number of the CDCL(T)-guided SLS calls is significantly greater than the number of the Independent SLS calls, and if we take the collected conflict frequency from the two SLS solvers as the same, the influence will be dominated by the conflict frequency provided by the CDCL(T)-guided SLS calls.

6 EXPERIMENTS

We perform extensive experiments to evaluate HybridSMT, which aims to answer the following research questions (RQ):

- **RQ1**: Does HybridSMT outperform other state-of-the-art (SOTA) solvers on the SMTLIB-NIA benchmark?
- **RQ2**: Are all the proposed methods effective?
- **RQ3**: What role did SLS and CDCL(T) play in the process of cooperation?

6.1 Experiments Setups

6.1.1 Environment. In this paper, all experiments are carried out on a cluster with two AMD EPYC 7763 CPU @ 2.45Ghz of 128 physical cores in total and 1T RAM under the operating system Ubuntu 20.04 LTS (64-bit). Each solver has only one chance to solve each instance within 1200 seconds.

6.1.2 Benchmarks. We use the standard benchmark for SMT(NIA) provided by SMT-LIB (SMTLIB-NIA), which can be downloaded from http://smtlib.cs.uiowa.edu/. This authoritative benchmark is usually used to evaluate the performance of SMT solvers in the annual SMT competitions [3]. This benchmark is rich in industrial instances, and most of the instances have real-world applications in the domain of software engineering. We introduce some of the categories with a relatively high proportion.

- The 'ITS', 'SAT14', and 'CInteger' families are generated by VeryMax [12] with termination proving problem [13].
- The 'AProVE' family is generated by an automated program termination verification tool [4].
- The 'LassoRanker' and 'UltimateLassoRanker' families use a tool that analyzes termination and non-termination of lasso-shaped programs [48, 49].
- The 'UltimateAutomizer' and 'UltimateAutomizer2023' family is generated by applying an automatic software verification tool to benchmarks from the SV-COMP [6].[1]
- The 'Leipzig' family provided by J. Waldmann is coming from termination analysis with matrix interpretation.
- The 'calypto' family is produced by Calypto Design Systems, Inc. in the context of sequential equivalence checking.
- The 'Math' family is generated from the magic square, diophantine equation, and taxicab number.
- The 'Dartagnan' family is from concurrency Safety analysis and checking state reachability under weak memory models [57].
- The 'MCM' family is from solving the Multiple Constant Multiplication (MCM) problem using Pseudo-Boolean Satisfiability [50].

6.1.3 Implementation. Our solvers, HybridSMT and Hybrid+Z3, are implemented in C++ and compiled by g++ with '-O3' option. The parameters for HybridSMT are manually tuned, while the competitors have been tuned on the SMT-COMP benchmark by their respective authors. Preliminary experiments indicate that our methods are not sensitive to the parameters.

Source code and detailed experimental statistics can be found in https://github.com/hybridsmt/hybridsmt.

6.1.4 Metrics and Plots. Here are the indicators for assistance:

- #Ins denotes the number of instances in a dataset.
- #S (#U) is the number of solved satisfiable (unsatisfiable) instances, and #A = #S+#U.
- #B denotes the number of instances that can be uniquely solved by HybridSMT, compared to its base solvers.
- #ByI (resp. #ByG) is the number of instances that are solved by the **I**ndependent (resp. CDCL(T)-**G**uided) SLS.
- #Ct(I) (resp. #Ct(G)) is the average number of calls of the Independent (resp. CDCL(T)-Guided) SLS.
- T(I) (resp. T(G)) is the average percentage of the runtime of the Independent (resp. CDCL(T)-Guided) SLS.

The *cumulative distribution functions* (CDF) plots show the number of solved instances depending on the time, and the higher the curve, the better the solver.

This paper uses *scatter plots* to compare the runtime (in seconds) of HybridSMT with other solvers. Axes are shown in logarithmic scale. The x-axis (resp. y-axis) solver is more appropriate for instances whose node is located above (resp. below) the diagonal line. The nodes marked with an orange cross (resp., blue circle) denote satisfiable (resp., unsatisfiable) instances. Note that the continuous points drawn horizontally or vertically in the plots are because of the switching of strategies or engines.

6.2 Comparison to SOTA Solvers (RQ1)

6.2.1 Competitors. We choose four SOTA solvers from recent SMT Competitions (SMT-COMP) [3]. We consider the following tools:

- Z3 (version 4.8.17) [29]. It is a portfolio of the lazy CDCL(T), eager bit-blasting, and the NLSAT algorithm [45].
- MathSAT5 (version 5.6.8) [25]. It uses the incremental linearization method [23] for SMT(NIA) instances.
- Yices2 (version 2.6.4) [32]. It uses purely the MCSat/NLSAT algorithm for SMT(NIA) problems.
- CVC5 (version 1.0.2) [2]. It integrates bit-blasting, NLSAT, and incremental linearization for SMT(NIA) solving.

[1] To save space, 'UltimateLassoRanker', 'UltimateAutomizer', and 'UltimateAutomizer2023' are noted as 'ULasso', 'UAuto', and 'UAuto23', respectively.

Benchmark	#Ins	VBS_1			**HybridSMT**			Z3			MathSAT5			Yices2			CVC5			ninc-cores		
		#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**
AProVE	2409	1663	733	2396	1652	696	2348	1658	697	**2355**	1647	565	2212	1593	**711**	2304	1373	622	1995	**1663**	245	1908
Calypto	177	80	97	177	78	**97**	175	**80**	95	175	79	90	169	79	96	175	79	94	173	**80**	**97**	**177**
CInteger	1818	863	707	1570	**857**	**686**	**1543**	771	510	1281	717	458	1175	520	463	983	323	378	701	849	140	989
Dartagnan	374	13	341	354	**13**	**341**	**354**	**13**	**341**	**354**	**13**	326	339	9	319	328	**13**	324	337	0	148	148
ezsmt	8	8	0	8	**8**	0	**8**	**8**	0	**8**	**8**	0	**8**	**8**	0	**8**	**8**	0	**8**	0	0	0
ITS	17046	9686	4712	14398	**9594**	**4613**	**14207**	8540	3977	12517	7784	3794	11578	6816	3557	10373	5502	3172	8674	9428	2314	11742
LassoRanker	106	4	102	106	4	**102**	**106**	4	**102**	**106**	4	101	105	4	85	89	4	93	97	4	89	93
LCTES	2	0	2	2	0	**2**	**2**	0	**2**	**2**	0	1	1	0	0	0	0	1	1	0	0	0
SAT14	1926	1853	73	1926	**1853**	**73**	**1926**	1852	68	1920	1801	67	1868	1840	66	1906	1802	72	1874	**1853**	63	1916
sqrtmodinv	27	0	11	11	0	**10**	**10**	0	**10**	**10**	0	0	0	0	0	0	0	1	1	0	0	0
ULasso	32	6	26	32	6	25	31	6	**26**	**32**	6	**26**	**32**	6	**26**	**32**	6	**26**	**32**	6	**26**	**32**
UAuto	7	0	7	7	0	**7**	**7**	0	**7**	**7**	0	**7**	**7**	0	**7**	**7**	0	**7**	**7**	0	1	1
UAuto23	58	8	13	21	7	3	10	2	2	4	7	**10**	**17**	5	0	5	**8**	6	14	5	0	5
Leipzig	167	160	4	164	157	2	159	**159**	1	**160**	128	2	130	101	1	102	90	2	92	155	**4**	159
Math	1100	687	7	694	100	**7**	107	**659**	**7**	**666**	203	**7**	210	112	**7**	119	227	**7**	234	550	0	550
MCM	186	84	5	89	**78**	0	**78**	15	1	16	13	0	13	11	0	11	16	**4**	20	19	0	19
ALL	25443	15115	6840	21955	14407	**6664**	**21071**	13767	5846	19613	12410	5454	17864	11104	5338	16442	9451	4809	14260	**14612**	3127	17739

Table 2: Compared to four state-of-the-art solvers shown in recent SMT-COMP and the best solver in [13]. VBS_1 is the virtual best solver of all the solvers presented in this paper.

We provide a virtual best solver (VBS), VBS_1, which uses a hypothetical algorithm that selects the best solver from a given portfolio of all the solvers in this paper. The VBS_2 denotes the VBS solver of pure CDCL(T) and pure SLS.

We also choose maxsmt, omt, ninc, cores, and ninc-cores [13], which are variants of barcelogic [11] based on linearization [2]. They are proposed to handle the termination proving instances belonging to the 'ITS', 'SAT14', and 'CInteger' families, taking up 81.7% of the SMTLIB-NIA. According to our experiments, ninc-cores has the best performance among the five solvers that appeared in [13].

6.2.2 Comparison to state-of-the-art solvers. Table 2 reports the results of the comparison between HybridSMT and six state-of-the-art SMT(NIA) solvers, and we have the following observations:

- Overall, HybridSMT significantly outperforms its competitors. In detail, HybridSMT solves 1458(7.43%),3207(17.95%), 4629(28.15%), 6811(47.76%), and 3332(18.78%) more instances than Z3, MathSAT5, Yices2, CVC5, and ninc-cores, respectively. From the results for each category, we learn that HybridSMT is the optimal choice for the termination and non-termination verification problems.
- HybridSMT has the best performance on #U, which solves 1749.2 (42.5%) more instances than its five competitors on average. The results confirm again the effectiveness.
- HybridSMT ranked second on #S, which is slightly worse than ninc-cores. This is mainly due to the worse performance of HybridSMT on the 'Math' family, which is more suitable for using the bit-blasting method [3]. HybridSMT has the best satisfiable performance if we neglect the 'Math' family and nearly all the remaining instances are from software engineering applications.

6.2.3 Runtime Comparison with Scatter Plots. Runtime is another important metric to consider. The five sub-figures (a)–(e) of Figure 3 show the runtime comparison between HybridSMT and its SOTA competitors. The scatter plots illustrate the run time for the instances solved by both solvers in comparison.

Two conclusions can be drawn. First, HybridSMT is much faster on the satisfiable instances than the four competitors from SMT-COMP; However, HybridSMT is slightly slower on unsatisfiable instances, but it can solve a greater number of them. Overall, the average runtime is a bit slower than the base CDCL(T) solver due to the introduction of SLS. Second, ninc-cores can quickly solve certain instances in one second. This motivated us to cooperate with the ninc-cores style solvers by utilizing them during the initial solving stage when designing a portfolio.

6.2.4 Improve Z3 with Our Hybrid Method. Besides the popular CDCL(T) algorithm, Z3 [29] has also implemented other powerful methods such as bit-blasting [16] and NLSAT [45]. Previous sections have shown the power of our hybrid method, and we are interested to know whether our method can collaborate with other methods and gain further improvement through simple portfolios.

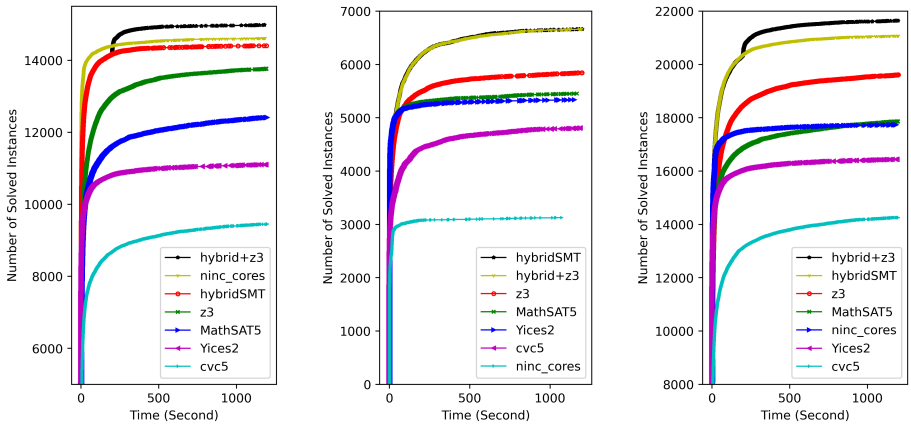
Table 3 [4] shows the results of Hybrid+Z3[5], which replaces the CDCL(T) tactic by HybridSMT. Hybrid+Z3 solves 575 (resp. 573) more #S (resp. #A) instances than HybridSMT, and has similar performance on #U, where the improvement mainly comes from the 'Math' and 'MCM' families. The reason that Z3 outperforms our solvers on the 'AProVE' family is that the instances have many Boolean literals. LocalSMT is not strong at dealing Boolean literals [19], while some methods in Z3, like CDCL(T) and bit-blasting, are strong in Boolean reasoning. Figure 3h summarizes the run time for instances solved by both HybridSMT and Hybrid+Z3. The two solvers have similar performance in these instances, confirming that HybridSMT cooperates well with other methods.

[2]We noticed that there are 13 instances that a solver in [13] report 'UNSAT' while all the other solvers report 'SAT'. Therefore, the instances are considered unsolved for the solvers shown in [13].

[3]Replacing the CDCL(T) tactic of Z3 with HybridSMT improve the performance of the 'Math' family with the help of the bit-blasting tactic in Z3.

[4]Note that in order to save space, any families with less than 200 instances are not individually reported in the tables. Instead, they are summarized in the "Other" row.

[5]Z3 is a sequential portfolio of the CDCL(T) tactic and many other NIA algorithms. Hybrid+Z3 integrates our strategies on the top of the CDCL(T) tactic, while HybridSMT disables all the NIA algorithms in Hybrid+Z3 other than our hybrid method.



(a) SAT **(b) UNSAT** **(c) ALL**

Figure 2: CDFs for SAT, UNSAT, ALL instances for comparing HybridSMT and Hybrid+Z3 with SOTA solvers.

Benchmark	**Hybrid+Z3**			HybridSMT			Z3		
	#S	#U	**#A**	#S	#U	**#A**	#S	#U	**#A**
AProVE	1655	**697**	2352	1652	696	2348	**1658**	**697**	**2355**
CInteger	854	**691**	**1545**	**857**	686	1543	771	510	1281
Dartagnan	13	341	354	13	341	354	13	341	354
ITS	9589	4606	14195	**9594**	**4613**	**14207**	8540	3977	12517
SAT14	**1853**	**73**	**1926**	**1853**	**73**	**1926**	1852	68	1920
Math	**677**	7	**684**	100	7	107	659	7	666
MCM	**82**	0	**82**	78	0	78	15	**1**	16
Other	259	247	506	**260**	**248**	**508**	259	245	504
ALL	**14982**	6662	**21644**	14407	**6664**	21071	13767	5846	19613

Table 3: Hybrid+Z3: Improve Z3 by HybridSMT.

Benchmark	**HybridSMT**			CDCL(T)			LocalSMT
	#S	#U	**#A**	#S	#U	**#A**	#S/ #A
AProVE	**1652**	**696**	**2348**	1643	695	2338	1620
CInteger	**857**	**686**	**1543**	717	681	1398	790
Dartagnan	**13**	**341**	**354**	**13**	**341**	**354**	0
ITS	9594	4613	**14207**	8273	**4638**	12911	**9627**
SAT14	**1853**	**73**	**1926**	**1853**	**73**	**1926**	1478
Math	**100**	**7**	**107**	**100**	**7**	**107**	0
MCM	78	0	78	15	**1**	16	**86**
Other	**260**	248	**508**	228	**249**	477	245
ALL	**14407**	6664	**21071**	12842	**6685**	19527	13846

Table 4: Effectiveness of the hybrid methods

Benchmark	**HybridSMT**	V1	V2	V3	V4	V5
AProVE	2348	2344	2350	2335	2341	**2351**
CInteger	**1543**	1518	1538	1534	**1543**	1538
Dartagnan	354	354	354	354	354	354
ITS	**14207**	14081	13894	14047	13748	14109
SAT14	**1926**	**1926**	**1926**	**1926**	1925	**1926**
Math	107	107	107	107	107	107
MCM	78	**93**	14	82	83	82
Other	508	494	**509**	507	505	506
ALL	**21071**	20917	20692	20892	20606	20973

Table 5: Effectiveness of each component.

Runtime analysis with CDF. To better understand the trend of the number of solved instances, we draw the CDFs comparing HybridSMT and Hybrid+Z3 with five SOTA SMT(NIA) solvers. From the three CDFs in Figure 2, we learn that our solver solves the most #S, #U, and #A. It shows that the hybrid methods improve not only #S, but also #U. A reasonable explanation is that better branching (order and phase) heuristics help CDCL(T) to find more high-quality clauses and meet more conflicts for faster pruning [21, 53].

It is worth mentioning that SMTLIB marks 5212 SMT(NIA) instances with the label 'unknown', which are highly challenging. HybridSMT (resp. Hybrid+Z3) solves 260 (resp. 260) satisfiable and 1810 (resp. 1811) unsatisfiable instances out of the 'unknown' instances. It further confirms the effectiveness of our methods.

6.3 Effectiveness of Proposed Strategies (RQ2)

We analyze the effectiveness of the proposed techniques, by comparing HybridSMT to its base solvers as well as ablation analyses.

6.3.1 Comparison with Its Base Solvers. We compared HybridSMT to Z3's CDCL(T) tactic and LocalSMT, the results are given in Table 4, with the runtime comparisons shown in Figures 3f and 3g.

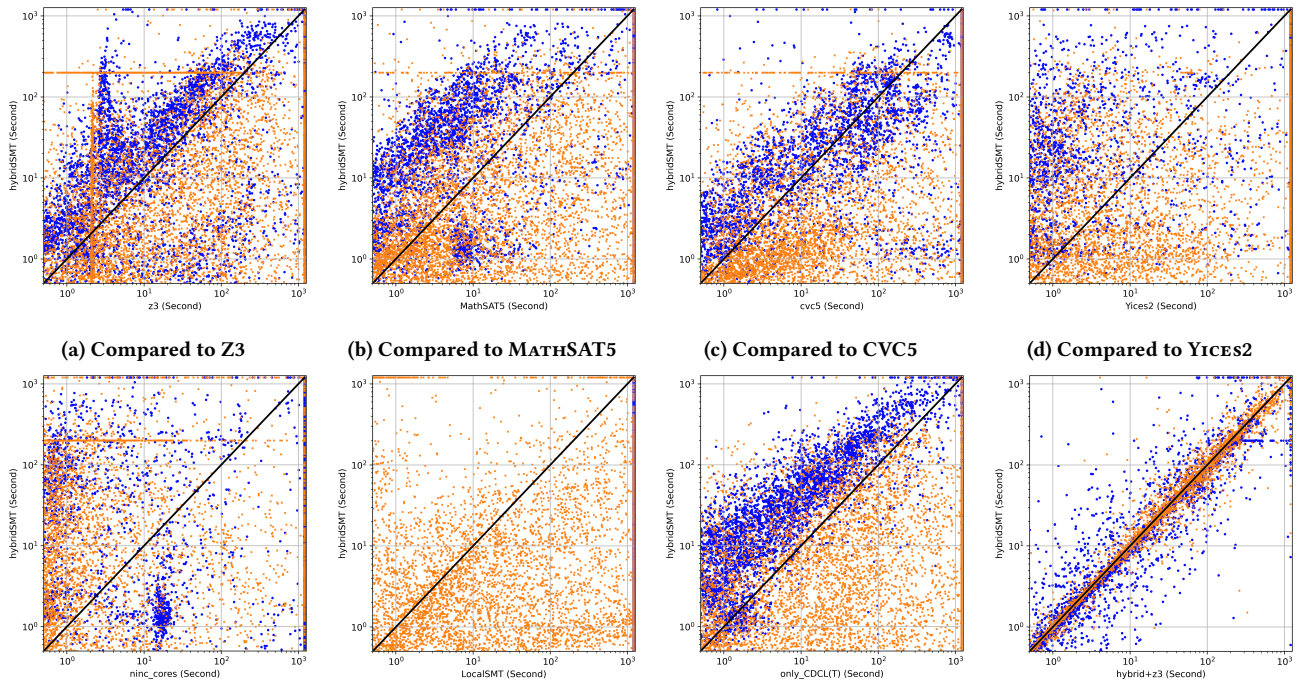
On the instance that both can solve, HybridSMT is 3.4× faster on average compared to Z3's CDCL(T) tactic. The CDCL(T) performance on satisfiable instances can be significantly improved by 12.2%. The runtime and the #U number on unsatisfiable instances have been affected by the allocation for SLS, as it is incapable of handling such instances. Thanks to the successful SLS scheduling methods and the SLS information, the negative influence on unsatisfiable instances brought by invoking SLS can be neglected.

Compared to LocalSMT, HybridSMT solves 561 more satisfiable instances, and the average runtime for the solvable instances is about 3.12× faster.

6.3.2 Assessment for Each Component. To analyze the effectiveness of each method in HybridSMT, we modify HybridSMT to obtain five alternative versions as follows.

- V1: Removing the CDCL(T)-guided SLS.
- V2: Removing the Independent SLS.
- V3: Using the original phase resetting method in Z3.
- V4: Disabling the components related to phases.
- V5: Disabling the conflict frequency component.

Comparisons in Table 5 show the significant contributions of each component. Specifically, the five variants solve 154, 379, 179,



(a) Compared to Z3 **(b) Compared to MathSAT5** **(c) Compared to CVC5** **(d) Compared to Yices2**

(e) Compared to ninc _cores **(f) Compared to LocalSMT** **(g) Compared to CDCL(T)** **(h) Compared to Hybrid+Z3**

Figure 3: Scatter plots that compare HybridSMT with six SOTA SMT(NIA) solvers, Z3's CDCL(T) tactic, and Hybrid+Z3. The nodes marked with an orange cross (resp., blue circle) denote satisfiable (resp., unsatisfiable) instances.

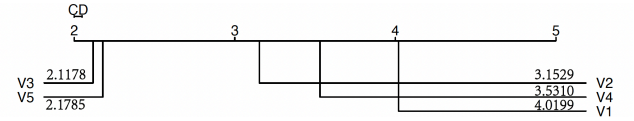


Figure 4: Critical difference diagram about 5 variants.

465, and 98 fewer instances than HybridSMT, respectively. And our methods contribute mainly to 3 termination verification families named 'CInteger', 'ITS', and 'SAT14'.

We use an R package (https://github.com/b0rxa/scmamp) to draw the critical difference (CD) diagram [31]. It evaluates the statistical difference between the 5 variants with the Friedman test [34] under the null hypothesis that algorithms have the same performance. Figure 4 shows that the 5 variants have critical differences with a significant level of 0.01, and gives the averaged performance ranks. The lower the rank, the higher the contributions. There is a considerable difference between the two solvers if their distance from each other exceeds the top-left CD bar. We can learn that the independent SLS and phase resetting contribute the most to the number of solved instances, while the CDCL(T)-guided SLS contributes the most to the rank (considering runtime).

6.4 Analysis for the Cooperation (RQ3)

For a better understanding of the roles of SLS, we carried out additional experiments, and the results are summarized in Table 6.[6].

[6]We conducted an extra experiment for statistical collection, which has similar performance to the one without information collection.

We observe that there are 164 instances that can be uniquely solved by HybridSMT, and can not be solved by a simple portfolio of SLS and CDCL(T). Even a virtual best solver that picks the solver with the best result for each instance would fail for these instances.

There are about 66% satisfiable instances are solved by the internal SLS calls. During each successful run, approximately 1.8 Independent SLS calls and 88.1 CDCL(T)-guided SLS calls are made, accounting for 39% and 16% of the total runtime, respectively. When solving unsatisfiable instances, SLS calls take up more runtime, but the price for each SLS call is much smaller than the one when solving satisfiable instances. The average runtime for each independent (resp. CDCLT(T)-guided) SLS call on unsatisfiable instances is about 3.6 (resp. 7.0) times faster than the one on the satisfiable instances.

6.5 Threats to Validity

There are three potential threats to the validity of our evaluations:

Correctness of Implementation.

The development of deep integration of SLS into CDCL(T) needs a significant amount of work. To ensure correctness, we tested our solver on the scrambled benchmarks via the official SMT-COMP scrambler, and compared the results of our solvers to the other competitors. The source codes are reviewed by three co-workers, and we use validators from z3 and SMT Competition to check the model for satisfiable instances.

Random Characteristics of Portfolio.

Using execution time to switch between portfolios introduces non-determinism into the solvers result. Firstly, the SLS invoking

Benchmark	For SAT instances									For UNSAT instances						
	#S	VBS_2(S)	#B(S)	#ByI	#ByG	#Ct(I)	#Ct(G)	T(I)%	T(G)%	#U	VBS_2(U)	#B(U)	#Ct(I)	#Ct(G)	T(I)%	T(G)%
AproVE	1654	1662	1	893	34	0.59	0.51	5.97	1.72	696	695	4	1.3	21.16	13.5	19.02
CInteger	857	835	25	618	203	1.79	33.74	49.89	9.0	688	681	10	4.02	293.0	43.88	32.36
Dartagnan	13	13	0	0	0	0.0	0.0	0.0	0.0	341	341	0	0.06	0.03	3.75	0.28
ITS	9590	10216	67	5726	1247	1.09	13.25	46.05	5.16	4612	4638	41	3.96	318.47	38.42	33.41
SAT14	1853	1853	0	411	144	1.11	1.98	42.15	37.9	73	73	0	1.62	46.66	29.62	53.27
Math	100	100	0	1	0	1.02	1.01	22.88	22.37	7	7	0	1.29	2.0	12.65	13.98
MCM	81	86	9	81	0	2.19	42.16	68.54	10.36	0	1	-	-	-	-	-
Other	260	257	5	122	50	1.18	13.6	24.18	9.34	247	249	2	0.77	1.48	36.36	6.96
ALL	14408	15022	107	7852	1678	1.09	11.63	40.71	9.42	6664	6685	57	3.34	253.44	34.4	29.32

Table 6: Analysis for the role of internal local search solver in HybridSMT. VBS_2 denotes the virtual best solver of Z3 and LocalSMT.

conditions of our hybrid method are based on deterministic heuristics. Meanwhile, to learn the impact of time parameters, we repeated the experiment for Hybrid+Z3 and Z3, which have time-based portfolio parameters, and experiments show similar results with differences of no more than 10 instances between different tests.

Benchmark Categories

Benchmarks are dominated by termination-proving instances, which may bias the results. We can evaluate the performance of a solver by the number of categories in which it excels, and our solvers win most of the categories. Besides the termination-proving instances, our solvers perform well on other families, such as the 'Dartagnan' family (the concurrency safety analysis, and reachability checking), and the 'MCM' family (multiple constant multiplication problems).

7 RELATED WORKS

Besides CDCL(T) and the stochastic local search (SLS), various approaches have been introduced to solve SMT(NIA). The *eager* approach, also called *bit-blasting*, reduces the formula to an equisatisfiable Boolean formula and feeds it to a SAT solver [16, 35, 42]. The dependencies between variables are encoded into Boolean constraints by limiting the ranges of arithmetic variables [35, 63], or deriving all possible transitivity constraints [62], sometimes using both methods [60]. There are studies on preprocessing methods [37, 52] before bit-blasting. The *model-construction satisfiability* (MCSat) calculus [30, 43, 44], or NLSAT algorithm [45], extends the CDCL(T) framework, by assigning concrete values to arithmetic variables instead of assigning Boolean values to the skeleton. Other approaches include *incremental linearization* [13, 14, 23, 24], branch-and-bound on the context of cylindrical algebraic decomposition (CAD), and virtual substitution (VS) [46]. Reinforcement Learning is used to improve the variable order selection heuristics in CAD [41]. State-of-the-art SMT(NIA) solvers such as Z3 [29] and CVC5 [2] are portfolios, where CDCL(T) is a main method.

The most related works are hybrid SAT solvers based on the cooperation between CDCL and SLS [20, 21]. It takes CDCL as the main body, invokes SLS when the percentage of the assigned variable reaches a threshold, and uses the assignment and conflict frequency of variables from SLS to improve the branching heuristics. Our method can be seen as a generalized version of such SAT hybrid method. The differences are as follows. Firstly, we use a two-layer framework, and SLS is called in two different flavors with different roles in our framework. Secondly, we propose a novel CDCL(T)-guided SLS method. Thirdly, we lift the techniques of using local search assignments and conflict frequencies from SAT to SMT by many adaptions and different strategies.

We are not aware of any work that combines CDCL(T) and SLS with information communication in the SMT area.

8 CONCLUSION

We proposed a two-layer hybrid method for the SMT(NIA) problem We take CDCL(T) as the main body, an independent SLS is periodically invoked, and a CDCL(T)-guided SLS is invoked when skeleton solutions are found. To enhance the branching heuristics of CDCL(T), the conflict frequency of variables and the best candidate solution from recent SLS calls are utilized. Our experiments demonstrate the superiority of our hybrid solver in software engineering applications. The proposed method also can be applied to solve other arithmetic theories.

In the future, we plan to deeply study the clause interaction strategies between CDCL(T) and SLS, and investigate the hybrid methods of other effective algorithms, e.g., the deep combination of NLSAT and SLS, and the hybrid methods on other background theories.

9 ACKNOWLEDGMENTS

This work is supported by NSFC Grant 62122078.

REFERENCES

[1] Gilles Audemard and Laurent Simon. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Twenty-first international joint conference on artificial intelligence*. Citeseer.

[2] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. 2022. cvc5: A versatile and industrial-strength SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 415–442.

[3] Clark Barrett, Leonardo De Moura, and Aaron Stump. 2005. SMT-COMP: Satisfiability modulo theories competition. In *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings 17*. Springer, 20–23.

[4] Karsten Behrmann, Andrej Dyck, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Patrick Kabasci, Peter Schneider-Kamp, and René Thiemann. 2014. Bit-blasting for SMT-NIA with AProVE. *Proc. SMT-COMP* 14 (2014).

[5] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2014. A constraint-based approach to solving games on infinite graphs. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 221–233.

[6] Dirk Beyer. 2015. Software Verification and Verifiable Witnesses: (Report on SV-COMP 2015). In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European*

Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21. Springer, 401–416.
[7] Armin Biere. 2017. Cadical, lingeling, plingeling, treengeling and yalsat entering the sat competition 2018. *Proceedings of SAT Competition* 14 (2017), 316–336.
[8] Armin Biere and Mathias Fleury. 2020. Chasing target phases. In *Workshop on the Pragmatics of SAT*.
[9] Armin Biere and Andreas Fröhlich. 2015. Evaluating CDCL restart schemes. *Proceedings of Pragmatics of SAT* (2015), 1–17.
[10] Armin Biere, Marijn Heule, and Hans van Maaren. 2009. *Handbook of satisfiability*. Vol. 185. IOS press.
[11] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2008. The Barcelogic SMT Solver: Tool Paper. In *International Conference on Computer Aided Verification*. Springer, 294–298.
[12] Cristina Borralleras, Daniel Larraz, Albert Oliveras, José Miguel Rivero, Enric Rodríguez-Carbonell, and Albert Rubio. 2016. VeryMax: tool description for termCOMP 2016. In *15th International Workshop on Termination*. 18.
[13] Cristina Borralleras, Daniel Larraz, Enric Rodríguez-Carbonell, Albert Oliveras, and Albert Rubio. 2019. Incomplete SMT techniques for solving non-linear formulas over the integers. *ACM Transactions on Computational Logic (TOCL)* 20, 4 (2019), 1–36.
[14] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2012. SAT modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning* 48, 1 (2012), 107–131.
[15] Aaron R Bradley, Zohar Manna, and Henny B Sipma. 2005. Linear ranking with reachability. In *International Conference on Computer Aided Verification*. Springer, 491–504.
[16] Randal E Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A Seshia, Ofer Strichman, and Bryan Brady. 2007. Deciding bit-vector arithmetic with abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems: 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24-April 1, 2007. Proceedings 13*. Springer, 358–372.
[17] Shaowei Cai, Bohan Li, Jinkun Lin, Zhonghan Wang, Bohua Zhan, Xindi Zhang, and Mengyu Zhao. [n. d.]. Z3++ at SMT-COMP 2022. ([n. d.]).
[18] Shaowei Cai, Bohan Li, and Xindi Zhang. 2022. Local Search For Satisfiability Modulo Integer Arithmetic Theories. *ACM Transactions on Computational Logic* (2022).
[19] Shaowei Cai, Bohan Li, and Xindi Zhang. 2022. Local search for smt on linear integer arithmetic. In *International Conference on Computer Aided Verification*. Springer, 227–248.
[20] Shaowei Cai and Xindi Zhang. 2021. Deep cooperation of cdcl and local search for sat. In *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*. Springer, 64–81.
[21] Shaowei Cai, Xindi Zhang, Mathias Fleury, and Armin Biere. 2022. Better decision heuristics in CDCL through local search and target phases. *Journal of Artificial Intelligence Research* 74 (2022), 1515–1563.
[22] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. 2012. SMTInterpol: An interpolating SMT solver. In *International SPIN Workshop on Model Checking of Software*. Springer, 248–254.
[23] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. 2018. Experimenting on solving nonlinear integer arithmetic with incremental linearization. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 383–398.
[24] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. 2018. Incremental linearization: A practical approach to satisfiability modulo nonlinear arithmetic and transcendental functions. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 19–26.
[25] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013. The mathsat5 smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 93–107.
[26] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma. 2003. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification: 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003. Proceedings 15*. Springer, 420–432.
[27] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006. Termination proofs for systems code. *ACM Sigplan Notices* 41, 6 (2006), 415–426.
[28] Martin Davis. 1973. Hilbert's tenth problem is unsolvable. *The American Mathematical Monthly* 80, 3 (1973), 233–269.
[29] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*. Springer, 337–340.
[30] Leonardo De Moura and Dejan Jovanović. 2013. A model-constructing satisfiability calculus. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 1–12.
[31] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.
[32] Bruno Dutertre. 2014. Yices 2.2. In *International Conference on Computer Aided Verification*. Springer, 737–744.
[33] ABKFM Fleury and Maximilian Heisinger. 2020. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. *SAT COMPETITION 2020* (2020), 50.
[34] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association* 32, 200 (1937), 675–701.
[35] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. 2007. SAT solving for termination analysis with polynomial interpretations. In *Theory and Applications of Satisfiability Testing–SAT 2007: 10th International Conference, Lisbon, Portugal, May 28-31, 2007. Proceedings 10*. Springer, 340–354.
[36] Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning* 24, 1-2 (2000), 67–100.
[37] Liana Hadarean, Kshitij Bansal, Dejan Jovanović, Clark Barrett, and Cesare Tinelli. 2014. A tale of two solvers: Eager and lazy approaches to bit-vectors. In *International Conference on Computer Aided Verification*. Springer, 680–695.
[38] Matthias Heizmann, Daniel Dietsch, Jan Leike, Betim Musa, and Andreas Podelski. 2015. Ultimate Automizer with Array Interpolation: (Competition Contribution). In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*. Springer, 455–457.
[39] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2013. Software model checking for people who love automata. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 36–52.
[40] Pei Huang, Haoze Wu, Yuting Yang, Ieva Daukantas, Min Wu, Yedi Zhang, and Clark Barrett. 2023. Towards Efficient Verification of Quantized Neural Networks. arXiv:2312.12679 [cs.LG]
[41] Fuqi Jia, Yuhang Dong, Minghao Liu, Pei Huang, Feifei Ma, and Jian Zhang. 2023. Suggesting Variable Order for Cylindrical Algebraic Decomposition via Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.
[42] Fuqi Jia, Rui Han, Pei Huang, Minghao Liu, Feifei Ma, and Jian Zhang. 2023. Improving Bit-Blasting for Nonlinear Integer Constraints. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 14–25.
[43] Dejan Jovanović. 2017. Solving nonlinear integer arithmetic with MCSAT. In *Verification, Model Checking, and Abstract Interpretation: 18th International Conference, VMCAI 2017, Paris, France, January 15–17, 2017, Proceedings 18*. Springer, 330–346.
[44] Dejan Jovanovic, Clark Barrett, and Leonardo De Moura. 2013. The design and implementation of the model constructing satisfiability calculus. In *2013 Formal Methods in Computer-Aided Design*. IEEE, 173–180.
[45] Dejan Jovanović and Leonardo De Moura. 2013. Solving non-linear arithmetic. *ACM Communications in Computer Algebra* 46, 3/4 (2013), 104–105.
[46] Gereon Kremer, Florian Corzilius, and Erika Ábrahám. 2016. A generalised branch-and-bound approach and its application in SAT modulo nonlinear integer arithmetic. In *Computer Algebra in Scientific Computing: 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings 18*. Springer, 315–335.
[47] Daniel Kroening and Ofer Strichman. 2016. *Decision procedures*. Springer.
[48] Jan Leike and Matthias Heizmann. 2014. Geometric series as nontermination arguments for linear lasso programs. *arXiv preprint arXiv:1405.4413* (2014).
[49] Jan Leike and Matthias Heizmann. 2018. Geometric nontermination arguments. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 266–283.
[50] Nuno P Lopes, Levent Aksoy, Vasco Manquinho, and José Monteiro. 2010. Optimally solving the mcm problem using pseudo-boolean satisfiability. *arXiv preprint arXiv:1011.2685* (2010).
[51] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*. 530–535.
[52] Aina Niemetz and Mathias Preiner. 2020. Bitwuzla at the SMT-COMP 2020. *arXiv preprint arXiv:2006.01621* (2020).
[53] Chanseok Oh. 2015. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 307–323.
[54] Hristina Palikareva and Cristian Cadar. 2013. Multi-solver support in symbolic execution. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 53–68.

[55] Knot Pipatsrisawat and Adnan Darwiche. 2007. A lightweight component caching scheme for satisfiability solvers. In *Theory and Applications of Satisfiability Testing–SAT 2007: 10th International Conference, Lisbon, Portugal, May 28-31, 2007. Proceedings 10*. Springer, 294–299.

[56] André Platzer, Jan-David Quesel, and Philipp Rümmer. 2009. Real world verification. In *Automated Deduction–CADE-22: 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings 22*. Springer, 485–501.

[57] Hernán Ponce-de León, Thomas Haas, and Roland Meyer. 2022. Dartagnan: Smt-based violation witness validation (competition contribution). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 418–423.

[58] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. 2004. Constructing invariants for hybrid systems. In *Hybrid Systems: Computation and Control: 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004. Proceedings 7*. Springer, 539–554.

[59] Roberto Sebastiani. 2007. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation* 3, 3-4 (2007), 141–224.

[60] Sanjit A Seshia, Shuvendu K Lahiri, and Randal E Bryant. 2003. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proceedings of the 40th annual Design Automation Conference*. 425–430.

[61] Mate Soos. 2016. The CryptoMiniSat 5 set of solvers at SAT Competition 2016. *Proceedings of SAT Competition* (2016), 28.

[62] Ofer Strichman, Sanjit A Seshia, and Randal E Bryant. 2002. Deciding separation formulas with SAT. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer, 209–222.

[63] Muralidhar Talupur, Nishant Sinha, Ofer Strichman, and Amir Pnueli. 2004. Range allocation for separation logic. In *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*. Springer, 148–161.

[64] Ashish Tiwari, Adria Gascón, and Bruno Dutertre. 2015. Program synthesis using dual interpretation. In *International Conference on Automated Deduction*. Springer, 482–497.

[65] Xindi Zhang and Shaowei Cai. 2020. Relaxed backtracking with rephasing. *Proceedings of SAT competition* (2020), 15–15.

[66] Xindi Zhang, Shaowei Cai, and Zhihan Chen. 2021. Improving cdcl via local search. *SAT COMPETITION 2021* (2021), 42.