ClauseSMT: Clause Level NLSAT for Nonlinear Real Arithmetic

Zhonghan Wang[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
wangzh@ios.ac.cn

Abstract. Model-constructing satisfiability calculus (MCSAT) framework has been applied to SMT problems on different arithmetic theories. NLSAT, an implementation using cylindrical algebraic decomposition for explanation, is especially competitive among nonlinear real arithmetic constraints. However, current Conflict-Driven Clause Learning (CDCL)-style algorithms only consider literal information for decision, and thus ignore clause-level influence on arithmetic variables. As a consequence, NLSAT encounters unnecessary conflicts caused by improper literal decisions. We analyze the literal decision caused conflicts, and introduce clause-level information with a direct effect on arithmetic variables. Two main algorithm improvements are presented: clause-level feasible set based look-ahead mechanism and arithmetic propagation based branching heuristic. We implement our solver named clauseSMT on our dynamic variable ordering framework. Experiments show that clauseSMT is competitive on nonlinear real arithmetic theory against existing SMT solvers (CVC5, Z3, YICES2), and outperforms all these solvers on satisfiable instances of SMT(QF_NRA) in SMT-LIB. The effectiveness of our proposed methods are also studied.

Keywords: NLSAT · Nonlinear Real Arithmetic · SMT.

1 Introduction

Satisfiability Modulo Theories (SMT) is a kind of problem which aims to detect the satisfiability of formulas in first order logic. SMT problems are usually involved in theories like linear and nonlinear arithmetic, uninterpretted functions, strings and arrays [8]. As a fundamental problem in software engineering, formal method and programming languages, it is widely used in various applications, such as symbolic execution [10,24], program verification [55,9], program synthesis [52], automata learning [50,54] and neural network verification [4,31,48,47].

Nonlinear real arithmetic (NRA) is one kind of arithmetic theories. It contains atoms which are represented as inequalities of polynomials, thus sometimes it is called theories about polynomial constraints. Variables can take boolean values or real numbers according to their types. Instances in SMT(NRA) are usually

generated from academical and industrial applications. It is commonly used in cyber physical systems [5,49,15], ranking function generation [35,26] and nonlinear hybrid automata analysis [18]. Instances from these applications are collected in the SMT-LIB benchmarks [7]. All these applications gain improvement thanks to the high performance of SMT solvers over nonlinear arithmetic.

Decision procedures about nonlinear arithmetic solving are usually based on cylindrical algebraic decomposition (CAD) [13], a useful real quantifier elimination tool. CAD is used to generate the current unsat cell during the search procedure, and is employed in modern algorithms. Among them, NLSAT [30] is the mainstream algorithm which uses CAD for lemma generation. Its basic idea is to directly assign the value to arithmetic variables, rather than on literal levels as CDCL(T) does.

Although NLSAT designs a novel view of directly assigning arithmetic variables, it still maintains literal decisions when processing arithmetic clauses with several unevaluated literals. Decided literals are then used for conflict analysis in such a CDCL-style framework. However, improper literal decisions sometimes cause conflicts and thus decrease the speed of the whole search procedure. Thus, a heuristic for literal decisions is required.

We present three central problems and give our solutions to them during the algorithm improvement.

- What causes conflicts during NLSAT algorithm? Can we avoid some of them?
- Is there any possible to directly assign the arithmetic variable, regardless of literal-level decision information in such a CDCL-style framework?
- Can we do propagation on arithmetic variables, just like unit propagation in SAT solving? Is the new propagation method helpful for new assignment and conflict detection?

1.1 Contributions

To answer the questions above, this paper for the first time proposes a new algorithm considering clause-level information. First, we analyze the conflict problems caused in NLSAT, and divide them into two categories. As described in [36], each clause narrows the feasible set[3] of an arithmetic variable. This technique has been used to enlarge the operation choices in local search algorithm. However, it has not been considered when designing complete methods like NLSAT. Arithmetic varaibles are sometimes narrowed to an empty search space and thus conflict happens. We describe this kind of problem in the view of interval arithmetic, and gives the solution thanks to the computation of feasible intervals. The clause-level feasible set idea extends the spirit of NLSAT that directly makes progress on arithmetic variables. Second, we introduce the incremental computation of clause-level feasible set, followed by the definition of clause-level propagation. In SAT solvers, unit propagation is an effective tool to deduce the assignment and detect conflict clauses. Similarly, we adopt the clause-level propagation to fix a possible witness for the arithmetic variable, or detect

[3] Also called satisfying domain in [36].

the empty feasible set cases as quickly as possible. Finally, we also present the overall structure and implementation details of our solver clauseSMT. Although dynamic variable ordering has been discussed in [45], SMT-RAT [20] solves less instances due to the lack of effective data structures. We present our techniques and data structures borrowed from SAT solving. We implement the above ideas in the NLSAT module of Z3 solver [41]. Some mathematical methods like root isolation, polynomial operations and algebraic number representation are relied on the existing libraries in Z3. We perform the experiments on the SMT-LIB benchmark, and studies the effect of proposed techniques, including look-ahead mechanism and clause-level propagation. The experiment results show that our algorithm solved the most satisfiable instances and is overall very competitive against other SMT solvers.

1.2 Related works

Methods for SMT solving are divided into two main kinds: complete methods and incomplete methods. Incomplete methods are effective due to their high speed and carefully designed techniques. Among them, interval constraint propagation (ICP) [32,51] is a common method to quickly detect unsatisfiable instances, as implemented in the dReal solver [23]. Recently, local search [27] has also been introduced from SAT problem to arithmetic theories, including integer arithmetic [11,12], linear and multilinear real arithmetic [36], and nonlinear real arithmetic [39,53]. Other incomplete methods like incremental linearization [16,17] and subtropical method [22,43] have also been explored to solve nonlinear constraints more efficiently. Complete methods are the main part of existing SMT solvers because of their high performance on both satisfiable and unsatisfiable instances. Both DPLL(T) [46] and NLSAT framework [30] use CAD for their theory solver or explanation module respectively [33]. By using other light methods like virtual substitution [19], NLSAT extends its idea into its successor MCSAT [42]. Until now this innovative framework is still the main complete method used for arithmetic theories and bitvectors [25]. Some works trying to improve the effectiveness of NLSAT have also been studied. For example, some works try to decrease the complexity of CAD by learning better projection orders of variables [38,14,28], designing innovative projection operators [37], and generating larger cells which are literal-invariant [1,44]. Other works talks about the dynamic branching heuristic for MCSAT [45]. The proof complexity of MCSAT has also been studied theorectically [34]. Besides, some recent works like hybridSMT [56] have studied the feasibility of incorporating local search into CDCL(T).

1.3 Structure of the paper

The structure of the paper is organized as follows: we first give the definition of SMT problems on nonlinear real arithmetic, and also the traditional complete method NLSAT in Section 2. Analysis of conflicts in NLSAT algorithm is studied in Section 3. A feasible set based look-ahead mechanism is presented in Section

4. Followed by the idea of clause-level information, we present the algorithm about clause-level propagation and a new branching heuristic in Section 5. In Section 6, we discuss the implementation details. We compare our solver with other SMT solvers and perform ablation study in Section 7. Finally, we conclude our work and propose potential research directions in Section 8.

2 Preliminaries

In this secion, we introduce the basic definition of SMT problems over nonlinear real arithmetic theory, followed by NLSAT algorithms. Besides, we also introduce the computation of clause-level feasible set.

2.1 Syntax of SMT(QF_NRA)

The syntax of SMT constraints on nonlinear real arithmetic is defined as follows:

$$\begin{aligned}
\text{variables:} \quad & v := x \mid b \\
\text{polynomials:} \quad & p := x \mid c \mid p + p \mid p \cdot p \\
\text{atoms:} \quad & a := b \mid p \leq 0 \mid p \geq 0 \mid p = 0 \\
\text{literals:} \quad & l := a \mid \neg a \\
\text{formulas:} \quad & c := l \mid l \vee l \mid l \wedge l
\end{aligned}$$

Here we denote x for an arithmetic variable which takes real numbers, b for a boolean variable which takes true or false. In nonlinear real arithmetic, an atom is either an boolean atom, defined by a boolean variable, or an arithmetic atom, defined by a strict or non-strict inequality of a polynomial. A literal is considered to be an atom or its negation. A clause is defined by the disjunction combination of literals. In our method, input instances are all transformed into conjunctive normal form (CNF), where the whole constraint is represented by the set of clauses. We denote $\mathbb{B}$ as set of boolean variables, $\mathbb{V}$ as set of arithmetic variables, SMT(NRA) as formulas on the theory of nonlinear real arithmetic.

For the semantics, we define an assignment α as a mapping from variables to values. Specifically, a boolean assignment maps boolean variables to boolean values, written as $\alpha(bool) : b \to \{\top, \bot\}$. An arithmetic assignment maps arithmetic variables to real values, written as $\alpha(real) : v \to \mathbb{R}$. In complete methods like MCSAT, a complete assignment maps all boolean and arithmetic variables to values, while an incomplete assignment only maps part of them. Under a given assignment, an atom can be evaluated to be true if the assignment satisfies it, false if the assignment breaks it, undefined if the atom contains a variable which is not mapped by the assignment. A complete assignment satisfying all clauses is also called a solution to the given instances, and thus proves it is satisfiable. The SMT problem on nonlinear real arithmetic is to look for a solution for the given instance, or prove that it can not be satisfied by any assignment.

Example 1. Let $B = \{b_1, b_2\}$ and $V = \{v_1, v_2, v_3\}$, representing the set of boolean variables and real-valued arithmetic variables respectively. An example of SMT(NRA) formula is shown as follows:

$$(v_1 v_2 >= 12 \vee b_1) \wedge (v_2 + 4v_3^2 < -4 \vee \neg b_2) \wedge (v_1 - v_2^2 + v_3^3 \geq 0)$$

A solution to the formula is shown as follow:

$$\{b_1 \to \top, b_2 \to \bot, v_1 \to 1, v_2 \to 2, v_3 \to 3\}$$

2.2 Feasible Set

For nonlinear real arithmetic constraints, a key technique used for determining the possible values of arithmetic variables is called root isolation. Given an arithmetic atom $p \{\leq, \geq, =, >, <\} 0$, if there is only one variable v not assigned by the assignment, we can calculate the set of its possible values satisfying the atom, called feasible set. In high order polynomial constraints, the computation of feasible set is usually based on root isolation, which uses mathematical methods to solve the roots of the polynomial. Intervals on the real space separated by the roots[4] maintain the evaluation value for the atom, and thus their combination represents the overall (in)feasible set. For literals in a negation form, the feasible set of the literal is actually the complement of the corresponding feasible set of the atom. By forcing the arithmetic variable any value in the feasible set, the atom will be satisfied[5].

Besides the literal level, feasible set can be extended to the clause level, which represents the set of values to make the clause satisfied. We also consider the circumstances when there is only one arithmetic variable being unassigned in the clause. We call that kind of clause a univariate clause. The formal definition of feasible set is shown as follow:

Definition 1. *Given a literal l, an arithmetic variable x, an assignment assigning all variables appearing in the literal l but x, the* feasible set *(resp.* infeasible set*) is the combination of intervals, in which the literal l is satisfied (resp. unsatisfied) when x is mapped.*

Similarly, given a clause c, an arithmetic variable x, an assignment assigning all variables appearing in the clause c but x, the feasible set *(resp.* infeasible set*) means that the clause c is satisfied (resp. unsatisfied) when x is mapped to any value in it. The computation of* feasible set *(resp.* infeasible set*) is done by just taking the union (resp. intersection) of computed feasible set (resp. infeasible set) respect to all literals in that clause. An example of feasible set is shown in Example 2.*

[4] Intervals are also called cells in the perspective of cylindrical algebraic decomposition.

[5] Sometimes the feasible set can be an empty set or a full set, which means the atom is always satisfied or unsatisfied by choosing any value for the arithmetic variable respectively.

Example 2. Given an assignment $\alpha := \{b \to \bot, x \to 4\}$, the feasible set of the clause below

$$b \vee y + x > 0 \vee y^2 > 2$$

should be $\{(-4, -\sqrt{2}) \cup (\sqrt{2}, \infty)\}$.

2.3 NLSAT Algorithm

NLSAT is the main part algorithm over nonlinear real arithmetic theory inside Z3 solver [41]. NLSAT and its successor MCSAT [42] have the ability to handle both boolean and arithmetic variables directly. Some detailed definitions are given below.

- **Boolean Decision:** decide a literal (or a boolean variable), same with sat solving.
- **Semantic Decision:** pick a value from a feasible set.
- **Unit Propagation:** propagate a literal's value when the clause has only one undefined literal, same with sat solving.
- **Real propagation:** propagate a literal's value based on variable's current feasible set.
- **Level:** current boolean decision times.
- **Stage:** current semantic decision times (or number of current assigned arithmetic variables).
- **Evaluation:** evaluate the literal's value when all its variables are assigned.
- **Trail:** a data structure used to record previous updates.

To select the right value for an arithmetic variable, NLSAT updates the feasible set of them in the search process. Assume the current feasible set is $curr\_set$, literal $lit's\ feasible\_set$ is lit_set, real-propagation takes effect when one of the circumstances below happens.

- **lit_set is empty**: the literal is propagated false, because it can't be satisfied by any value.
- **lit_set is full**: the literal is propagated true, because it can be satisfied whatever value we choose.
- **curr_set is a subset of lit_set**: the literal is propagated true, since current feasible set has already satisfied it.
- **curr_set has no intersection with lit_set**: the literal is propagated false, since choosing values from current feasible set must make the literal unsatisfied.

When processing a clause with boolean and arithmetic variables, NLSAT first uses propagation methods and evaluation to detect evaluated literals. If one of the them is true, the clause is skipped; otherwise NLSAT decides the first literal and updates the feasible set. Algorithm 1 shows the processing process section in NLSAT. For the conflict analysis, NLSAT uses cylindrical algebraic decomposition (CAD) as a tool for explanation. Model-based projection generates the conflict cell and learns a lemma to avoid entering it in the future. Algorithm 2 shows a complete NLSAT method.

Algorithm 1: Process Clauses in NLSAT

```
Input  : A set of clauses F
Output: Conflict clause conf_cls
for clause c in clauses F do
    for literal l in clause c do
        feasible_set ← compute feasible set of literal;
        literal_value ← real propogate literal;
        if literal_value = ⊤ then
            break;
        if literal_value = ⊥ then
            continue;
    if only one literal undefined then
        unit propagate;
    else if two or more literals undefined then
        decide the first literal;
    else
        return c;
return No Conflict;
```

Algorithm 2: NLSAT

```
Input  : A formula F
Output: SAT or UNSAT
while true do
    conf_cls ← process clauses;
    if conf_cls is empty then
        // Consistent
        if next variable is boolean then
            Boolean decide;
        else if next variable is arithmetic then
            Semantics decide;
        else
            // No unassigned variables
            return SAT;
    else
        // Conflict
        new_lemma ← conflict analysis;
        if new_lemma is empty then
            return UNSAT;
        else
            backtrack;
```

3 Conflicts During NLSAT algorithm

In this section, we analyze the reasons that cause conflicts in NLSAT algorithms. Generally they can be divided into two categories, semantics decision caused conflicts and literal decision caused conflicts.

3.1 Conflicts Caused by Semantics Decision

In SAT solving, the conflicts are always caused by literal decisions (or boolean variable decisions). For a satisfiable instances, SAT solvers can avoid any conflicts when using a perfect phase selection method. For unsatisfiable instances, conflicts are encountered whatever phase the solver selects. In both cases, when CDCL detects a conflict by wrong decision values, conflict analysis is invoked to generate a new lemma which forces the previous assignment to change.

Similarly, in NLSAT algorithm for SMT solving, conflicts are caused by wrong semantics decision, i.e. choosing a value from a given interval. As discussed in [39], the search space of nonlinear arithmetic is composed by sign-invariant cells. However, in systematic solving like NLSAT, we can not forecast the cell we are currently searching, thus conflicts may happen. For an unsatisfiable instance, each cell in the search space is inconsistent with all polynomial constraints, thus conflicts can not be avoided. We give the demo in Example 3.

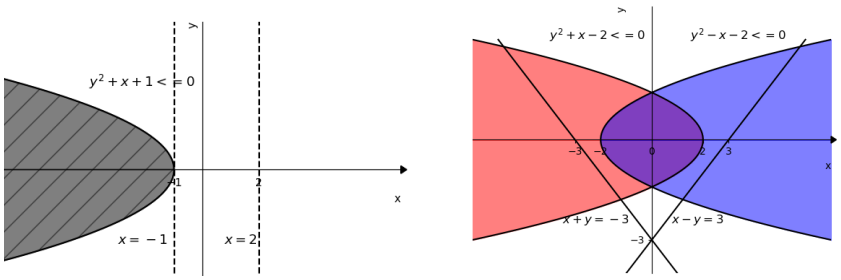

Fig. 1: Left: semantics decision caused conflict. Right: Literal decision caused conflict.

Example 3. Consider the formula is $\{y^2+x+1 \leq 0\}$, the variable order is $\{x, y\}$. As depicted in Figure 1 left, when we decide $\{x \to 2\}$, the satisfying area (shaded area) has no intersection with the dashed line $x = 2$, and thus a conflict happens. This kind of conflicts is caused by wrong semantics decision value for previous variable x, and can be avoided if we choose a right value, for example $\{x \to -2\}$.

3.2 Conflicts Caused by Literal Decision

A key technique used in NLSAT is processing clauses univariate to the current arithmetic variable. In CDCL-style systematic search, literals are required to be

assigned by unit propagation for unit clauses (only one literal unassigned), or decisions for clauses with multiple unassigned literals. However, there has been less attention paid on the literal decision mechanism of NLSAT. Improper literal decisions may cause extra conflicts. We explain this circumstance in Example 4.

Example 4. Suppose we have three clauses shown below:

$$c_1 : y^2 + x - 2 \leq 0 \vee y^2 - x - 2 \leq 0$$
$$c_2 : x + y = -3 \quad c_3 : x - y = 3$$

As depicted in Figure 1 right, the purple area satisfies both polynomials in clause c_1, while the red area and blue one only satisfy $y^2 + x - 2 \leq 0$ and $y^2 - x - 2 \leq 0$ respectively. Two straight lines represent two equality constraints respectively. Suppose the formula for the SMT problem is $\{c_1, c_2\}$, the intersection is located only in the red area. If the formula is $\{c_1, c_3\}$, the intersection is then located only in the blue area. However, when NLSAT algorithm processes a clause with multiple unassigned literals like c_1, it will decide one literal and branch the search space into either red+purple area, or blue+purple area. In this case, there is 50 percent possibility to miss the straight line and causes conflict. However, it is noticed that the feasible set of clause c_1 (i.e. the red+blue+purple area) have an unempty intersection with both straight lines, which means the formula $\{c_1, c_2\}$ and $\{c_1, c_3\}$ are both satisfiable.

Here comes a new problem for this circumstance. Is it possible to avoid conflicts by a better literal decision heuristic? If so, how can we do this? The key idea is to view the literal decision problem as a satisfiability about intervals. We give another Example 5 to help explain.

Example 5. Suppose current assignment is $\alpha ::= \{x \to 0\}$. The clauses univariate to y is shown as follows:

$$c_1 : \quad (y+2)(y+4) \leq x \vee (y-2)(y-4) \leq x$$
$$c_2 : \quad (y+5)(y+6) \leq x \vee (y-1)(y-5) \leq x$$

By calculating the feasible set of each literal, the interval view of the clauses is shown below.

$$c_1 : \quad [-4, -2] \vee [2, 4]$$
$$c_2 : \quad [-6, -5] \vee [1, 5]$$

Then the problem can be demonstrated like this: is there a value that is included in at least one interval of all clauses?

4 Feasible set Based Look-Ahead Mechanism

In this section, we enroll the clause-level feasible set into NLSAT algorithm and design a look-ahead mechanism.

4.1 Look-Ahead Before Processing Clauses

We first extend the feasible set definition to a clause set.

Definition 2. *Given a clause set cs, an arithmetic variable x, an assignment assigning all variables appearing in any clause in cs but x, the* feasible set *(resp.* infeasible set*) means that each clause c is satisfied (resp. unsatisfied) when x is mapped to any value in it. Specifically, the feasible set can be computed by taking the intersection of each clause's feasible set.*

$$feasible\_set(cs) = \bigcap_{c \in cs} feasible\_set(c)$$

By using feasible set of a clause set, it's much easier to directly conclude the search space of an arithmetic variable. Concretely, when the feasible set is not empty, by assigning any value in the set to the variable (semantics decision), the search will make progress and jump to the next stage. On the contrary, when the set is empty, there is no way to avoid inconsistence by choosing a value for the variable. Now we answer the question proposed in Section 3 and give the formal definition below.

Definition 3. *When the feasible set of the clause set is not empty, theoretically we can avoid conflicts by deciding proper literals, we call this case a* path case*. In contrast, when the feasible set is empty, the conflict can not be avoided by literal decisions (is caused by semantics decisions), we call this case a* block case*. Example 6 and 7 show a path case and a block case respectively.*

Example 6. This example shows a path case, since the clauses can be satisfied by deciding literals in the green boxes.

$$\left.\begin{array}{ll} c_1: & [-4,-2] \vee \boxed{[2,\ 4]} \rightarrow \{[-4,2] \cup [2,4]\} \\ c_2: & [-6,-5] \vee \boxed{[1,\ 5]} \rightarrow \{[-6,-5] \cup [1,5]\} \end{array}\right\} \bigwedge \rightarrow \{[2,4]\}$$

Example 7. This example shows a block case, the clauses can not be satisfied whatever literals we decide.

$$\left.\begin{array}{ll} c_1: & \boxed{[\text{-}4,\ \text{-}2]} \vee \boxed{[2,\ 4]} \rightarrow \{[-4,2] \cup [2,4]\} \\ c_2: & \boxed{[\text{-}6,\ \text{-}5]} \vee \boxed{[5,\ 6]} \rightarrow \{[-6,5] \cup [5,6]\} \end{array}\right\} \bigwedge \rightarrow \varnothing$$

The feasible set computation gives us a view of current consistent search space. However, in a CDCL-style algorithm, literals must be assigned for future conflict analysis. Here comes another question? How do we decide literals if we already know it's a path case (i.e. find green boxes in Example 6)? In our work, we use a look-ahead mechanism that allows us appoint a value before processing clauses. Then we utilize the pre-appointed value to look for a consistent decision path. We give the detail in Algorithm 3.

The updated algorithms adds a condition that the feasible set of current literal should include the pre-appointed value. By doing so, feasible sets of all

Algorithm 3: Deciding Literals using pre-appointed value

```
   Input  : A set of clauses F, arithmetic variable v, pre-appointed value val
   Output: Decided literals lits
 1 for clause c in clauses F do
 2     for literal l in clause c do
 3         feasible_set ← compute feasible set of literal;
 4         literal_value ← real propogate literal;
 5         if literal_value = ⊤ then
 6             break;
 7         if literal_value = ⊥ then
 8             continue;
 9         // decide satisfiable literals under val
10         if feasible_set contains val then
11             path_literal ← l;
12     if only one literal undefined then
13         unit propagate;
14     else
15         lits ← lits ∪ {path_literal};
16 return lits;
```

decided literals during the processing procedure will intersect with each other into a clause-set-level feasible set, which allows the arithmetic variable to be assigned by that appointed value. For the block case, the process algorithm is the same with NLSAT, which will later enter resolve procedure and force previous arithmetic assignments to change.

4.2 Look-Ahead After Conflict Analysis

Look-ahead algorithm after conflict analysis is similar to the main search part, as shown in Algorithm 4. The main difference is the incremental computation of decision cases by considering only the feasible set of the new lemma. For the previously processed clauses, we use a vector of intervals to cache their feasible sets. By union the feasible set of the new lemma, we can quickly return the decision case of the arithmetic variable. For the path case, we must reprocess the clauses and try to find a new decision path, because the new generated lemma adds a new constraint on the arithmetic variable, and sometimes forcing the current decision path blocked, as shown in Example 8.

Example 8. Before learning a new lemma, left shows a possible path in green boxes. However, the updated feasible set is not consistent with any literal in the lemma, which causes a conflict. Right shows a new possible case after the

Algorithm 4: Process clauses after a new lemma

Input: A new lemma *lemma*, arithmetic variable v

```
1 lemma_feasible_set ← compute feasible set of clause(lemma);
2 feasible_set[v] = feasible_set[v] ∩ lemma_feasible_set;
3 if feasible_set[v] is empty then
      // block case
      // same with nlsat, call Algorithm  1
4     process clauses;
5 else
      // path case
6     val ← value_selection(feasible_set[v]);
      // call Algorithm 3
7     deciding literals using pre-appointed value (val);
```

incremental lemma.

$$c_1 : \boxed{\text{[-7, -2]}} \vee [2, 8] \qquad c_1 : [-7, -2] \vee \boxed{\text{[2, 8]}}$$

$$c_2 : [-11, -10] \vee \boxed{\text{[-6, 5]}} \qquad c_2 : [-11, -10] \vee \boxed{\text{[-6, 5]}}$$

$$learned : \boxed{\text{[3, 4]}} \vee \boxed{\text{[7, 8]}} \qquad learned : \boxed{\text{[3, 4]}} \vee [7, 8]$$

5 Clause-Level Propagation

Following the idea of using clause-level feasible set, this section introduces a new kind of propagation called clause-level propagation. The idea is inspired from unit propagation (or literal propagation) in SAT solving. For a boolean satisfaction problem, boolean variables can be unit propagated to assign a boolean value. By propagating boolean variables, sat solvers can assign an unassigned variable, or detect conflicts as soon as possible. However, in smt solvers like NLSAT, propagation on arithmetic variables has not been studies yet. We give the formal definition of clause-level propagation in Definition 4.

Definition 4. *Given a clause c, an arithmetic variable x, an assignment α assigning all variables appearing in the clause but x. A clause-level propagation on x is to compute the feasible set of current clause c, which narrows the feasible set of variable x.*

The difference between arithmetic and boolean problems is about the states of search spaces. Unit propagation always propagate the boolean variables to be true or false. In other words, unit propagation always prune the search space of the variable a half, and thus guides the search proceed (cause there are only two possible values for boolean variables). However, in arithmetic view, clauses sometimes only prune the search space by part of the real space, and thus only make the search space contract. In this section, we introduce the algorithm of

Algorithm 5: Clause-Level Propagation

```
Input: Clause set F, a new assigned variable v
for clause cls that contains v do
    if cls is univariate to an arithmetic variable v' then
        cls_feasible_set ← compute_feasible_set of clause (cls);
        feasible_set[v'] = feasible_set[v'] ∩ cls_feasible_set;
        if feasible_set[v'] is empty then
            blocked_vars = blocked_vars ∪ {v'};
        if feasible_set[v'] is single value then
            fixed_vars = fixed_vars ∪ {v'};
```

clause-level propagation, and then use the propagation information to guide our search by changing the next branching variable.

5.1 Clause-Level Propagation Method

In SAT solving, unit propagation happens after assignment. In our algorithm, we incrementally calculate the feasible set of a clause, when it's univariate to an arithmetic variable[6]. The new generated univariate clause will add an additional constraint to the arithmetic variable, and thus prune the search space by taking the intersection. Unlike the boolean search space, we divide the propagation cases into three kinds as shown in Example 9:

- **Block case:** The clause-level feasible set is empty.
- **Fixed case:** The clause-level feasible set contains only one real number, for example $[2, 2]$.
- **Other case:** The clause-level feasible set is narrowed, but not empty or only include one real value.

These information are later used for better branching heuristic. We show the details in Algorithm 5.

Example 9. An example is shown as Figure 2. When variable x is assigned to 0, three clauses become univariate to other variables $\{x, y, k\}$. These three clauses add three new constraints on arithmetic variables, calculated as $\{(-\infty, -2] \cup [2, 6]\}, \{[2, 2]\}, \emptyset$. These three feasible sets just indicate that variables are feasible, fixed or blocked.

5.2 Propagation Based Branching Heuristic

After unit propagation in SAT solving, an unassigned variable is propagated to a value, or a conflict clause is detected effectively. When we get the feasible set for

[6] This does not only happen after the assignment of an arithmetic variable, but also after a boolean variable is assigned. Whatever, when the clause is arithmetically univariate, the feasible set is updated.

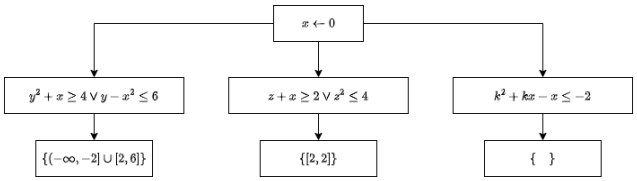

Fig. 2: Demo of clause-level propagation for y (normal case), z (fixed case) and k (block case).

Algorithm 6: Branching Heuristic of Variables

Output: A variable v

```
// branch blocked variables first (conflict)
if blocked_vars ≠ ∅ then
    v ← select_from(blocked_vars);
// branch fixed variables next (propagate value)
else if fixed_vars ≠ ∅ then
    v ← select_from(fixed_vars);
else
    v ← vsids_select;
return v
```

arithmetic variables, propagation and conflict cases still exist, which respectively match the fixed and blocked case talked above. However, current clause-level conflict can not return a conflict clause directly in NLSAT, which is actually the responsibility of processing clauses algorithm. Thus, we use the recorded fixed and blocked information, and force the branching heuristic to aim at solving those variables as soon as possible.

For the normal case, we use Variable State Independent Decaying Sum (VSIDS) [40] as the main branching heuristic, as suggested in [29] and [45]. We give the branching heuristic in Algorithm 6. Details about our implementation of dynamic variable ordering is talked in Section 6.

6 Implementation

All the above algorithms are incorporated into our new solver called clauseSMT.[7]. We describe the details of the solver, which include a detailed version of conflict analysis, and also implementations of dynamic variable ordering framework. The overall structure of clauseSMT is depicted in Figure 3. We introduce our implementation details in Appendix B.

[7] Experimental results and binary files for Linux system are available at `https://zenodo.org/records/13830686`.

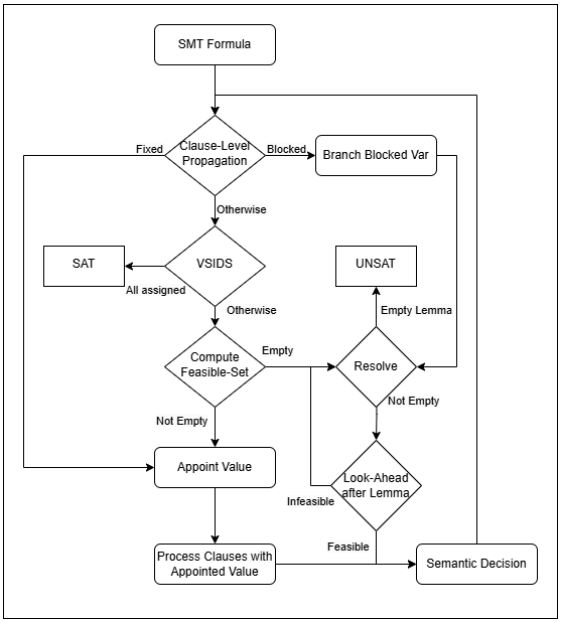

Fig. 3: Overall Structure of clauseSMT.

6.1 Resolve

Although look-ahead based processing algorithm is given in previous section, cases are more complex for the implementation. As discussed, conflicts in NLSAT can be divided into two kinds. For the literal decisions case [8], we maintain the resolve algorithm same with original algorithm. The search engine should backtrack to the maximum decision level of the new lemma, and try to unit propagate the negation of previously decided literals. In other words, this case happens just in pure literal levels. For the second case, we detect literal decision cases with consideration of the incremental clause (learned lemma). Note that since we need to recalculate the decision path, literals that has been decided at that stage need to be reset.

In order to better management the backtrack, we introduce several new kinds of trail normally used in NLSAT algorithm:

- **path_finder**: whenever we check the feasible set of current stage is non-empty, a path exists and this trail is saved.
- **block_finder**: when we find the current stage is blocked, we save this trail.
- **clause_feasible_updated**: whenever we update a clause feasible set of an arithmetic variable, the trail is saved.

Detailed version of resolve is shown in Algorithm 9:

[8] For look-ahead mechanism, this happens only for block cases.

Algorithm 7: Resolve

Input: Conflict clause $conf$, assignment ass

```
new_lemma ← conflict analysis ;
if conflict caused by literal decision then
    max_level ← max_level(new_lemma);
    backtrack to new level(max_level);
    unit_propagate(new_lemma);
else
    // new_lemma must contains previous arithmetic variables
    last_assigned_var ← get_last_assigned_var(new_lemma);
    // previous stage must be path cases.
    backtrack to path finder(last_assigned_var);
    // call Algorithm 4
    process clauses after a new lemma(new_lemma);
```

6.2 Shortcut for UNSAT Instances

A shortcut mechanism is designed for the block case described in Section 4. When the blocked clauses only contain the current variable, we directly conclude that the instance must be unsatisfiable.

7 Evaluation

In this section, we compare our algorithm with other existing solvers, such as Z3 (version 4.13.1) [41], CVC5 (version 1.0.2) [6] and YICES2 (version 2.6.2) [21]. Ablation study of several improvements are also described.

7.1 Experiment Preliminaries

The normally used benchmark for evaluating SMT solvers is SMT-LIB. The whole benchmark of QF_NRA theory consists of 12134 instances, from various applications ranging from nonlinear hybrid automata, ranking functions generating for program analysis, and also mathematical problems. Most instances have been labeled as satisfiable or unsatisfiable, but still some are labeled as unknown. One should note that instances from SMT-LIB always show different forms in clause numbers, literal numbers and even polynomial degrees. Our experiments are evaluated on a server with Intel Xeon Platinum 8153 processor at 2.00 GHz. We limit the running time of each instance 1200 seconds, just the same in the SMT-COMP.

7.2 Overall Result

We compare our algorithm with other SMT solvers in Table 1. Note that when evaluating our algorithm in Z3 solver, we disable all other tactics like DPLL(T)

Category	#inst		Z3	YICES2	CVC5	NLSAT	Ours
20161105-Sturm-MBO	405	SAT	0	0	0	0	0
		UNSAT	124	**285**	**285**	44	39
		SOLVED	124	**285**	**285**	44	39
20161105-Sturm-MGC	9	SAT	**2**	0	0	**2**	**2**
		UNSAT	**7**	0	0	**7**	6
		SOLVED	**9**	0	0	**9**	8
20170501-Heizmann	69	SAT	**2**	0	1	1	**2**
		UNSAT	1	12	9	10	**19**
		SOLVED	3	12	10	11	**21**
20180501-Economics-Mulligan	135	SAT	**93**	91	89	**93**	92
		UNSAT	39	39	35	**41**	**41**
		SOLVED	132	130	124	**134**	131
2019-ezsmt	63	SAT	56	52	50	**58**	36
		UNSAT	**2**	**2**	**2**	**2**	**2**
		SOLVED	58	54	52	**60**	38
20200911-Pine	245	SAT	234	**235**	199	**235**	234
		UNSAT	6	**8**	5	7	5
		SOLVED	240	**243**	204	242	239
20211101-Geogebra	112	SAT	**110**	99	91	**110**	98
		UNSAT	0	0	0	0	0
		SOLVED	**110**	99	91	**110**	98
20220314-Uncu	225	SAT	69	**70**	62	68	**70**
		UNSAT	**155**	153	148	**155**	152
		SOLVED	**224**	223	210	223	222
hong	20	SAT	0	0	0	0	0
		UNSAT	8	**20**	**20**	12	14
		SOLVED	8	**20**	**20**	12	14
hycomp	2752	SAT	**307**	227	225	244	291
		UNSAT	**2242**	2201	2212	2088	2181
		SOLVED	**2549**	2428	2437	2332	2472
kissing	45	SAT	**33**	10	17	12	14
		UNSAT	0	0	0	0	0
		SOLVED	**33**	10	17	12	14
LassoRanker	821	SAT	167	122	**305**	220	302
		UNSAT	151	260	**470**	174	311
		SOLVED	318	382	**775**	394	613
meti-tarski	7006	SAT	**4391**	4369	4343	**4391**	4372
		UNSAT	2605	2588	2581	**2611**	2588
		SOLVED	6996	6957	6924	**7002**	6960
UltimateAutomizer	61	SAT	35	39	35	**45**	39
		UNSAT	11	12	10	**13**	12
		SOLVED	46	51	45	**58**	51
zankl	166	SAT	**70**	58	58	62	56
		UNSAT	28	**32**	**32**	27	30
		SOLVED	**98**	90	90	89	86
Total	12134	SAT	5569	5372	5475	5541	**5608**
		UNSAT	5379	5612	**5809**	5191	5397
		SOLVED	10948	10984	**11284**	10732	11005

Table 1: Summary of results for all instances in SMT-LIB (QF_NRA).

and incomplete algorithms. Solvers including Z3, CVC5 and YICES2 are all tested without any modification, carrying a portfolio of different algorithms. We also test original NLSAT solver by disabling other tactics.

Our algorithm is competitive with current state-of-the-art solvers, like Z3 and CVC5. Specifically, we solve the most satisfiable instances and the second most unsatisfiable instances. Scatter plots about solution times are shown in Figure 4.

Comparison with CVC5 CVC5 solves the most instances in our experiment. However, it is especially effective in unsat instances thanks to incomplete algorithms like interval constraint propagation and incremental linearization. A traditional category called MBO [3], contains only one clause with a very high degree, which makes it difficult to be solved by CAD based algorithms.

Comparison with Original NLSAT Actually, our solver decreases a little in most instances. These instances always contain high degree polynomials and

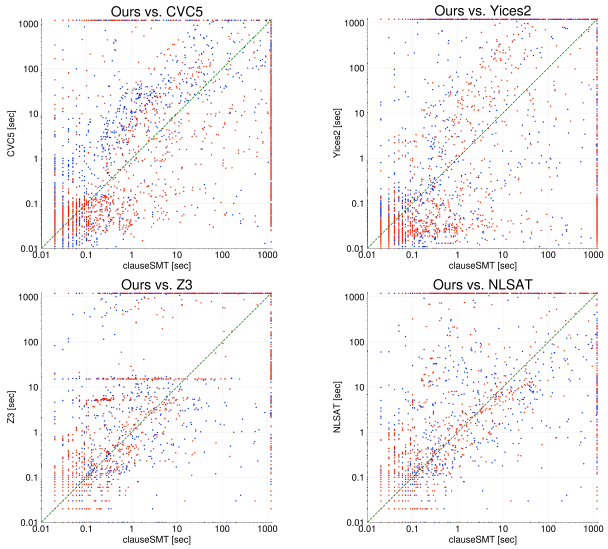

Fig. 4: Run time comparison against CVC5, Z3, YICES2 and nlsat (blue points: satisfiable instances, red points: unsatisfiable instances).

thus make our feasible set computation relatively heavy. However, our solver shows a strong advancement in LassoRanker and hycomp. These categories contains thousands of instances and usually have a complex feasible set relationship. Our solver increases a half in LassoRanker, which comes from termination analysis [26,35]. Our solver has improved NLSAT almost three hundred instances for the overall result.

7.3 Effectiveness of Look-Ahead Mechanism

In order to show the effect of look-ahead mechanism, several versions are implemented, as denoted below. The experimental results are shown in Table 2.

- **Look-Ahead**: Implement feasible set based look-ahead mechanism on original NLSAT solver (static variable order).
- **Lower Degree**: Decide literals with the lowest degree. This is the default heuristic in NLSAT.
- **Random Decide:** Randomly decide literals when processing clauses.

Although the difference of solved problems in most categories is not large, our algorithm solves about 50 more instances in the hycomp [18] category, which contains a huge amount of nonlinear equalities. In this case, hycomp instances usually exist literal path cases, which is exactly the advantage of look-ahead mechanism.

Theoretically, look-ahead mechanism can detect block cases and avoid conflicts for path cases. We record the conflict times in both algorithms. The scatter

Category	#inst	Decide Lower Degree	Random Decide	Look-Ahead
20161105-Sturm-MBO	405	44	45	44
20161105-Sturm-MGC	9	9	9	9
20170501-Heizmann	69	11	5	7
20180501-Economics-Mulligan	135	134	134	134
2019-ezsmt	63	60	59	58
20200911-Pine	245	242	242	243
20211101-Geogebra	112	110	109	110
20220314-Uncu	225	223	224	224
hong	20	12	12	12
hycomp	2752	2332	2272	2388
kissing	45	12	14	15
LassoRanker	821	394	393	389
meti-tarski	7006	7002	7001	7002
UltimateAutomizer	61	58	44	57
zankl	166	89	89	87
Total	12134	10732	10652	10778

Table 2: Comparison of solved instances for different literal decision mechanisms.

plot is shown in Figure 5 left and middle. The red line represents solving this instance brings the same conflict number for both implementation. As the scatter plot shows, most instances can be solved using less times of conflict. Although this property does not bring a huge gap for a 1200 seconds test, it will help boost the systematic search for clauses with multiple literals.

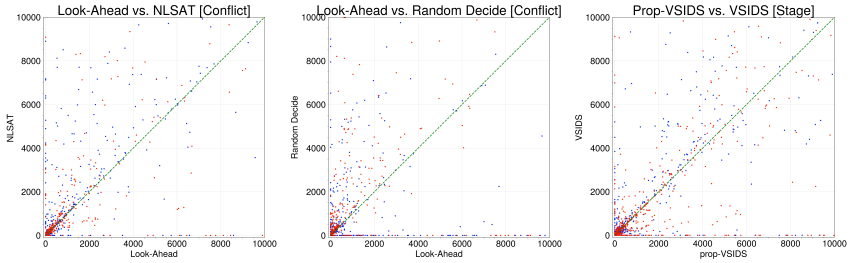

Fig. 5: (Left) Conflict times of look-ahead NLSAT against original NLSAT. (Middle) Conflict times of look-ahead NLSAT against random NLSAT. (Right) Stage comparison of prop-VSIDS against VSIDS (blue points: satisfiable instances, red points: unsatisfiable instances)

7.4 Effectiveness of Clause-Level Propagation

Clause-Level Propagation can only be evaluated on dynamic variable ordering framework. Thus, we implement three versions for comparison: original NLSAT

Category	#inst	Static	VSIDS	prop-VSIDS
20161105-Sturm-MBO	405	44	38	39
20161105-Sturm-MGC	9	9	6	8
20170501-Heizmann	69	7	20	21
20180501-Economics-Mulligan	135	134	133	133
2019-ezsmt	63	58	30	38
20200911-Pine	245	243	239	239
20211101-Geogebra	112	110	101	98
20220314-Uncu	225	224	222	222
hong	20	12	11	11
hycomp	2752	2388	2426	2472
kissing	45	15	14	14
LassoRanker	821	389	571	613
meti-tarski	7006	7002	6974	6960
UltimateAutomizer	61	57	52	51
zankl	166	87	83	86
Total	12134	10778	10920	11005

Table 3: Comparison of solved instances for different branching heuristics.

solver with static order based on variable's degree (static), dynamic NLSAT solver (VSIDS), and clause-level propagation dynamic NLSAT algorithm (prop-VSIDS). Results are shown in Table 3.

The results indicate that VSIDS is effective in MCSAT framework. Besides that, clause-level propagation indeed speeds up the detection of conflicts, and increases the solved instances in most categories. More concretely, prop-VSIDS is especially competitive in three categories: hycomp [18], LassoRanker [26] and meti-tarski [2]. All these three categories contain a huge number of arithmetic clauses, and thus easily makes the arithmetic variable fall into a block case. Stages (semantic decision times) of prop-VSIDS against traditional VSIDS is shown in Figure 5 right. By detecting inconsistent branching choices quickly, the overall stages required for a problem has been decreased a lot.

8 Conclusion and Future Work

In this paper, we present a novel NLSAT-based clause-level algorithm for SMT problem over nonlinear real arithmetic. We category the conflicts situations in NLSAT and discuss about the literal decision problem. Several improvements including feasible set based look-ahead mechanism and clause-level propagation based branching have been presented.

In the future, we wish to find an easier way to tackle with the block case as mentioned in section 4. We hope to design a lighter method compared with CAD to change the previous assignments and create a consistent decision path.

References

1. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Log. Algebraic Methods Program. **119**, 100633 (2021). https://doi.org/10.1016/j.jlamp.2020.100633, https://doi.org/10.1016/j.jlamp.2020.100633
2. Akbarpour, B., Paulson, L.C.: Metitarski: An automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010). https://doi.org/10.1007/s10817-009-9149-2, https://doi.org/10.1007/s10817-009-9149-2
3. Akutsu, T., Hayashida, M., Tamura, T.: Algorithms for inference, analysis and control of boolean networks. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) Algebraic Biology, Third International Conference, AB 2008, Castle of Hagenberg, Austria, July 31-August 2, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5147, pp. 1–15. Springer (2008). https://doi.org/10.1007/978-3-540-85101-1_1, https://doi.org/10.1007/978-3-540-85101-1_1
4. Amir, G., Wu, H., Barrett, C., Katz, G.: An smt-based approach for verifying binarized neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II. p. 203–222. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-72013-1_11, https://doi.org/10.1007/978-3-030-72013-1_11
5. Bae, K., Gao, S.: Modular smt-based analysis of nonlinear hybrid systems. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 180–187 (2017). https://doi.org/10.23919/FMCAD.2017.8102258
6. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24, https://doi.org/10.1007/978-3-030-99524-9_24
7. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
8. Barrett, C.W., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_11, https://doi.org/10.1007/978-3-319-10575-8_11
9. Beyer, D., Dangl, M., Wendler, P.: A unifying view on smt-based software verification. J. Autom. Reason. **60**(3), 299–335 (mar 2018). https://doi.org/10.1007/s10817-017-9432-6, https://doi.org/10.1007/s10817-017-9432-6

10. Cadar, C., Dunbar, D., Engler, D.: Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. p. 209–224. OSDI'08, USENIX Association, USA (2008)
11. Cai, S., Li, B., Zhang, X.: Local search for SMT on linear integer arithmetic. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 227–248. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_12, https://doi.org/10.1007/978-3-031-13188-2_12
12. Cai, S., Li, B., Zhang, X.: Local search for satisfiability modulo integer arithmetic theories. ACM Trans. Comput. Logic **24**(4) (jul 2023). https://doi.org/10.1145/3597495, https://doi.org/10.1145/3597495
13. Caviness, B.F., Johnson, J.R.: Quantifier elimination and cylindrical algebraic decomposition. In: Texts and Monographs in Symbolic Computation (2004)
14. Chen, C., Zhu, Z., Chi, H.: Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In: Mathematical Software – ICMS 2020: 7th International Conference, Braunschweig, Germany, July 13–16, 2020, Proceedings. p. 281–291. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-52200-1_28, https://doi.org/10.1007/978-3-030-52200-1_28
15. Cimatti, A.: Application of smt solvers to hybrid system verification. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 4–4 (2012)
16. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Experimenting on solving nonlinear integer arithmetic with incremental linearization. In: Theory and Applications of Satisfiability Testing – SAT 2018: 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings. p. 383–398. Springer-Verlag, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-319-94144-8_23, https://doi.org/10.1007/978-3-319-94144-8_23
17. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Log. **19**(3), 19:1–19:52 (2018). https://doi.org/10.1145/3230639, https://doi.org/10.1145/3230639
18. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: Cabodi, G., Singh, S. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012. pp. 187–195. IEEE (2012), https://ieeexplore.ieee.org/document/6462573/
19. Corzilius, F., Ábrahám, E.: Virtual substitution for smt-solving. In: Proceedings of the 18th International Conference on Fundamentals of Computation Theory. p. 360–371. FCT'11, Springer-Verlag, Berlin, Heidelberg (2011)
20. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: Smt-rat: An open source c++ toolbox for strategic and parallel smt solving. In: Heule, M., Weaver, S. (eds.) Theory and Applications of Satisfiability Testing – SAT 2015. pp. 360–368. Springer International Publishing, Cham (2015)
21. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_49, https://doi.org/10.1007/978-3-319-08867-9_49

22. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.: Subtropical satisfiability. In: Dixon, C., Finger, M. (eds.) Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10483, pp. 189–206. Springer (2017). https://doi.org/10.1007/978-3-319-66167-4_11, https://doi.org/10.1007/978-3-319-66167-4_11
23. Gao, S., Kong, S., Clarke, E.M.: dreal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_14, https://doi.org/10.1007/978-3-642-38574-2_14
24. Godefroid, P., Klarlund, N., Sen, K.: Dart: directed automated random testing. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 213–223. PLDI '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1065010.1065036, https://doi.org/10.1145/1065010.1065036
25. Graham-Lengrand, S., Jovanović, D., Dutertre, B.: Solving bitvectors with mcsat: Explanations from bits and pieces. In: Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I. p. 103–121. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-51074-9_7, https://doi.org/10.1007/978-3-030-51074-9_7
26. Heizmann, M., Hoenicke, J., Leike, J., Podelski, A.: Linear ranking for linear lasso programs. In: Hung, D.V., Ogawa, M. (eds.) Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8172, pp. 365–380. Springer (2013). https://doi.org/10.1007/978-3-319-02444-8_26, https://doi.org/10.1007/978-3-319-02444-8_26
27. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier / Morgan Kaufmann (2004)
28. Jia, F., Dong, Y., Liu, M., Huang, P., Ma, F., Zhang, J.: Suggesting variable order for cylindrical algebraic decomposition via reinforcement learning. In: Thirty-seventh Conference on Neural Information Processing Systems (2023), https://openreview.net/forum?id=vNsdFwjPtL
29. Jovanovic, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: 2013 Formal Methods in Computer-Aided Design. pp. 173–180 (2013). https://doi.org/10.1109/FMCAD.2013.7027033
30. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7364, pp. 339–354. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_27, https://doi.org/10.1007/978-3-642-31365-3_27
31. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 97–117. Springer International Publishing, Cham (2017)

32. Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. In: Jeannet, B. (ed.) Third Workshop on Tools for Automatic Program Analysis, TAPAS 2012, Deauville, France, September 14, 2012. Electronic Notes in Theoretical Computer Science, vol. 289, pp. 27–40. Elsevier (2012). https://doi.org/10.1016/j.entcs.2012.11.004, https://doi.org/10.1016/j.entcs.2012.11.004
33. Kremer, G.: Cylindrical algebraic decomposition for nonlinear arithmetic problems. Ph.D. thesis, RWTH Aachen University, Germany (2020), https://publications.rwth-aachen.de/record/792185
34. Kremer, G., Abraham, E., Ganesh, V.: On the proof complexity of mcsat (2021)
35. Leike, J., Heizmann, M.: Ranking templates for linear loops. Log. Methods Comput. Sci. **11**(1) (2015). https://doi.org/10.2168/LMCS-11(1:16)2015, https://doi.org/10.2168/LMCS-11(1:16)2015
36. Li, B., Cai, S.: Local search for smt on linear and multi-linear real arithmetic. In: 2023 Formal Methods in Computer-Aided Design (FMCAD). pp. 1–10 (2023). https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_25
37. Li, H., Xia, B.: Solving satisfiability of polynomial formulas by sample-cell projection (2020)
38. Li, H., Xia, B., Zhang, H., Zheng, T.: Choosing better variable orderings for cylindrical algebraic decomposition via exploiting chordal structure. J. Symb. Comput. **116**, 324–344 (2023). https://doi.org/10.1016/j.jsc.2022.10.009, https://doi.org/10.1016/j.jsc.2022.10.009
39. Li, H., Xia, B., Zhao, T.: Local search for solving satisfiability of polynomial formulas. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 87–109. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_5, https://doi.org/10.1007/978-3-031-37703-7_5
40. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232). pp. 530–535 (2001). https://doi.org/10.1145/378239.379017
41. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) 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. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
42. de Moura, L.M., Jovanovic, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7737, pp. 1–12. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_1, https://doi.org/10.1007/978-3-642-35873-9_1
43. Nalbach, J., Ábrahám, E.: Subtropical satisfiability for SMT solving. In: Rozier, K.Y., Chaudhuri, S. (eds.) NASA Formal Methods - 15th International Symposium, NFM 2023, Houston, TX, USA, May 16-18, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13903, pp. 430–446. Springer

(2023). https://doi.org/10.1007/978-3-031-33170-1_26, https://doi.org/10.1007/978-3-031-33170-1_26
44. Nalbach, J., Ábrahám, E., Specht, P., Brown, C.W., Davenport, J.H., England, M.: Levelwise construction of a single cylindrical algebraic cell. J. Symb. Comput. **123**(C) (may 2024). https://doi.org/10.1016/j.jsc.2023.102288, https://doi.org/10.1016/j.jsc.2023.102288
45. Nalbach, J., Kremer, G., Ábrahám, E.: On variable orderings in mcsat for non-linear real arithmetic. In: SC-square@SIAM AG (2019), https://api.semanticscholar.org/CorpusID:204767299
46. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006). https://doi.org/10.1145/1217856.1217859, https://doi.org/10.1145/1217856.1217859
47. Paulsen, B., Wang, C.: Example guided synthesis of linear approximations for neural network verification. In: Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I. p. 149–170. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-13185-1_8, https://doi.org/10.1007/978-3-031-13185-1_8
48. Paulsen, B., Wang, C.: Linsyn: Synthesizing tight linear bounds for arbitrary neural network activation functions. In: Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I. p. 357–376. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-030-99524-9_19, https://doi.org/10.1007/978-3-030-99524-9_19
49. Shoukry, Y., Chong, M., Wakaiki, M., Nuzzo, P., Sangiovanni-Vincentelli, A., Seshia, S.A., Hespanha, J.A.P., Tabuada, P.: Smt-based observer design for cyber-physical systems under sensor attacks. ACM Trans. Cyber-Phys. Syst. **2**(1) (jan 2018). https://doi.org/10.1145/3078621, https://doi.org/10.1145/3078621
50. Tappler, M., Aichernig, B.K., Lorber, F.: Timed automata learning via smt solving. In: NASA Formal Methods: 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24–27, 2022, Proceedings. p. 489–507. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-06773-0_26, https://doi.org/10.1007/978-3-031-06773-0_26
51. Tung, V.X., Khanh, T.V., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. Formal Methods Syst. Des. **51**(3), 462–499 (2017). https://doi.org/10.1007/s10703-017-0284-9, https://doi.org/10.1007/s10703-017-0284-9
52. Wang, J., Wang, C.: Learning to synthesize relational invariants. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. ASE '22, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3551349.3556942, https://doi.org/10.1145/3551349.3556942
53. Wang, Z., Zhan, B., Li, B., Cai, S.: Efficient local search for nonlinear real arithmetic. In: Verification, Model Checking, and Abstract Interpretation: 25th International Conference, VMCAI 2024, London, United Kingdom, January 15–16, 2024, Proceedings, Part I. p. 326–349. Springer-Verlag, Berlin, Heidel-

berg (2024). https://doi.org/10.1007/978-3-031-50524-9_15, https://doi.org/10.1007/978-3-031-50524-9_15
54. Xu, R., An, J., Zhan, B.: Active learning of one-clock timed automata using constraint solving. In: Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings. p. 249–265. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-19992-9_16, https://doi.org/10.1007/978-3-031-19992-9_16
55. Yao, P., Shi, Q., Huang, H., Zhang, C.: Program analysis via efficient symbolic abstraction. Proc. ACM Program. Lang. **5**(OOPSLA) (oct 2021). https://doi.org/10.1145/3485495, https://doi.org/10.1145/3485495
56. Zhang, X., Li, B., Cai, S.: Deep combination of cdcl(t) and local search for satisfiability modulo non-linear integer arithmetic theory. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. ICSE '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3597503.3639105, https://doi.org/10.1145/3597503.3639105

A Complexity Analysis of Look-Ahead Mechanism

A.1 Complexity of Computation Time

Suppose currently we are processing the clause set of variable v, shown as below:

$$\begin{aligned} c_1: &\quad l_{11} \vee l_{12} \vee ... \vee l_{1n_1} \\ c_2: &\quad l_{21} \vee l_{22} \vee ... \vee l_{2n_2} \\ &\quad ... \\ c_m: &\quad l_{m1} \vee l_{m2} \vee ... \vee l_{mn_m} \end{aligned}$$

Union and Intersection of Interval Sets An interval is defined by a tuple

$$< low\_inf, upp\_inf, low\_open, upp\_open, low\_val, upp\_val >$$

where low_inf and upp_inf represents whether the interval contains negative infinity and positive infinity respectively, low_open and upp_open represents whether the interval contains the left value and right value respectively, low_val and upp_val represents the lowest and largest value of the endpoints of the interval.

In nlsat algorithm, an interval set is usually represented by a sorted vector of intervals. Suppose we have two following interval sets below:

$$\begin{aligned} s1: &\quad interval_{11}, interval_{12}, ..., interval_{1m} \\ s2: &\quad interval_{21}, interval_{22}, ..., interval_{2n} \end{aligned}$$

The union calculation of these two interval sets are executed by two pointers starting at the leftmost elements. By taking the union of each two interval elements, the algorithm will eventually terminates if all elements are looped. For the intersection computation, the procedure is similar. Thus, the complexity of union and intersection operation is

$$O(mn)$$

where m and n stand for the length of each interval set respectively.

Original Literal Decision In original NLSAT algorithm, the processing method is done by checking each clause in the set one by one. The algorithm must loop each literal in the clause, to try to assign the literal value (true, false or unknown) and detect unit clauses. Thus, the complexity of looping all literals should be

$$O(mn)$$

where m stands for the number of clauses, n stands for the length (literal number) of each clause.

When looping each literal, original procedure still requires to calculate the feasible-set of that atom and sometimes assign its value by real propagation.

Look-Ahead Mechanism In Look-ahead mechanism, the first calculation is about the total feasible-set of the clause set. In this case, the algorithm still requires to calculate each feasible-set of each literal, and then compute the overall set by taking the intersection or union. Thus, the feasible-set computation method still costs

$$O(mn)$$

where m stands for the number of clauses, n stands for the length (literal number) of each clause, which is the same with the original procedure.

After the computation, the algorithm needs to reprocess all clauses using the clause-level information, as our paper suggested. Thus, the overall time complexity should be the same order with the original algorithm. However, as the look-ahead mechanism avoids literal decision causes conflicts, this cost is worthy.

A.2 Complexity of Memory Usage

Look-ahead mechanism brings more memory usage for the overall algorithm. The main cost is used for caching the feasible-set of each clause, and the overall feasible-set of the clause set of each variable, denoted by

$$O(m + v)$$

where m and v stand for the number of clauses and variables respectively.

B Implementation Details about dynamic variable ordering framework

Watched Variables To detect univariate clauses and lemmas, we implement two watched variables inspired by two watched literals. Each clause or lemma is watched by two variables (boolean or arithmetic) appearing in it. Watchers are changed when one of them is being assigned. Cases are split below:

- there exists a third variable unassigned: we just replace the assigned watcher by this one.
- there is no other variable unassigned: this clause must be univariate to the unassigned watcher.
- both watchers are assigned: we do nothing for this case.

Whenever we detect a univariate clause, feasible set is updated in an eager way, and thus helps the search engine gather more clause-level information.

Projection Order As discussed in [45], in most cases the variable outside the polynomial needs to be assigned at last. Thus, in our implementation we use the projection order exactly the reverse order of the previous assignment.

Branching Heuristic VSIDS is a particularly effective branching heuristic. We increase activities of variables each time a conflict is detected. All variables involved in the conflict analysis will be considered. We design several branching heuristics as suggested by [45]. As suggested in [45], we use VSIDS for branching heuristic. Each time when a conflict is detected, we increase activities of variables involved in the conflict analysis. The first branching variable should be the most active one, whatever type the variable is.

Lemma Management We record the activity of learned lemmas in conflict analysis, and periodically delete half of them. Compared with sat solvers, lemmas are not usually all useful due to the existence of root atoms. Deleting lemmas actually decreases memory allocation spent on useless lemmas.

Parameter Settings Based on our empirical analysis, we design alues of tunable parameters as depicted in Table 4.

Symbol	Description	Value
arith_decay	Decay_factor for arithmetic variables in VSIDS	0.95
bool_decay	Decay_factor for boolean variables in VSIDS	0.95
arith_bump	Incremental amount of arithmetic activity	1
bool_bump	Incremental amount of boolean activity	1
lemma_conf	Initial conflict times for deleting lemmas	100
lemma_conf_inc	Incremental factor of conflict for lemmas	1.5

Table 4: Tunable parameters