Frontiers of Computer Science CN 10-1014/TP
https://doi.org/10.1007/s11704-024-40500-z ISSN 2095-2228
RESEARCH ARTICLE

SharpSMT: A Scalable Toolkit for Measuring Solution Spaces of SMT(LA) Formulas

Cunjing GE[1,2]

1 National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
2 School of Artificial Intelligence, Nanjing University, Nanjing 210023, China

Front. Comput. Sci., **Just Accepted Manuscript** • 10.1007/s11704-024-40500-z
https://journal.hep.com.cn on August 12, 2024

© Higher Education Press 2024

Just Accepted

This is a "Just Accepted" manuscript, which has been examined by the peer-review process and has been accepted for publication. A "Just Accepted" manuscript is published online shortly after its acceptance, which is prior to technical editing and formatting and author proofing. Higher Education Press (HEP) provides "Just Accepted" as an optional and free service which allows authors to make their results available to the research community as soon as possible after acceptance. After a manuscript has been technically edited and formatted, it will be removed from the "Just Accepted" Web site and published as an Online First article. Please note that technical editing may introduce minor changes to the manuscript text and/or graphics which may affect the content, and all legal disclaimers that apply to the journal pertain. In no event shall HEP be held responsible for errors or consequences arising from the use of any information contained in these "Just Accepted" manuscripts. To cite this manuscript please use its Digital Object Identifier (DOI®), which is identical for all formats of publication."

Front. Comput. Sci.
https://doi.org/10.1007/s11704-024-40500-z

RESEARCH ARTICLE

SharpSMT: A Scalable Toolkit for Measuring Solution Spaces of SMT(LA) Formulas

Cunjing GE(✉)[1,2]

1 National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
2 School of Artificial Intelligence, Nanjing University, Nanjing 210023, China

© Higher Education Press 2024

Abstract In this paper, we present SharpSMT, a toolkit for measuring solution spaces of SMT(LA) formulas which are Boolean combinations of linear arithmetic constraints, i.e., #SMT(LA) problems. It integrates SMT satisfiability solving algorithm with various polytope subroutines: volume computation, volume estimation, lattice counting, and approximate lattice counting. We propose a series of new polytope preprocessing techniques which have been implemented in SharpSMT. Experimental results show that the new polytope preprocessing techniques are very effective, especially on application instances. We believe that SharpSMT will be useful in a number of areas.

Keywords #SMT(LA) Problems, DPLL(T) Algorithm, Polytope Preprocessing Techniques, Volume Computation, Lattice Counting

1 Introduction

The satisfiability (SAT) problem in the propositional logic is a fundamental problem in computer science. But in practice, many problems cannot be expressed by propositional formulas directly or naturally. In recent years, there have been a lot of works on solving the Satisfiability Modulo Theories (SMT) problem, which try to decide the satisfiability of logical formulas with respect to combinations of background theories (like reals, integers, arrays, bit-vectors). SMT can be regarded as an extension to SAT, as well as a kind of constraint satisfaction problem (CSP). Quite efficient SMT solvers have been developed, such as CVC5 [1, 2], MathSAT5 [3], Yices [4] and Z3 [5].

Restricted to linear arithmetic (LA) theory, they are SMT(LA) formulas. They have been widely used and thoroughly studied in many areas. In this paper, we focus on the counting version of the SMT(LA) problems, i.e., the #SMT(LA) problems. They have many applications, such as probabilistic inference [6, 7], counting-based search [8, 9], simple temporal planning [10], probabilistic program analysis [11, 12], etc.

Intuitively, the solution space of an SMT(LA) formula can be viewed as the union of many polytopes, since a set of linear constraints corresponds to a polytope. Ma and Ge et al. [13, 14] proposed an algorithm for #SMT(LA) problems. It first enumerates polytopes, and then handling polytopes with subroutines. Finally, they would sum up the volume, or integer point counts of polytopes. They also implemented a prototype tool called VolCE. The performance of lattice counting tool LattE limited the capability of integer solution counting of VolCE. In practice, it often has difficulties when the number of variables is greater than 10 (preventing many applications). Since then, some new methods for approximating integer solution counts were proposed by Ge et al. [15–17], which could solve problems with dozens of dimensions. Naturally, we would like to integrate those new methods with VolCE together.

Received month dd, yyyy; accepted month dd, yyyy

E-mail: gecunjing@nju.edu.cn

In this paper, we present the tool SharpSMT [1)] for #SMT(LA) problems. It supports inputs in SMT-LIB (v2.0) format which is a common language to describe SMT formulas. SharpSMT is an integration of SMT solving and polytope subroutines, such as, volume computation, volume estimation, lattice counting and approximate lattice counting for convex polytopes. To reduce the computations of polytope subroutines, we propose some preprocessing techniques for polytopes, such as, factorization techniques, variable elimination, cache strategy, etc. Experimental results show that the new polytope preprocessing techniques are very effective, especially on application instances. We also find that SharpSMT significantly outperforms the hashing-based counter SMTApproxMC [18] by further comparison experiments.

The rest of the paper is organized as follows: We first present background in Section 2. Next, we present the architecture of SharpSMT with new techniques in Section 3. Then, we discuss the experimental results in Section 4. Finally, we conclude in Section 5.

2 Background

2.1 Preliminaries and Notations

A Linear Constraint (LC) h can be written in the form $h \equiv \sum_{i=1}^{n} a_i x_i \ \mathsf{op}\ b = \vec{a}\vec{x} \ \mathsf{op}\ b$, where x_is are numeric variables, a_is and b are real coefficients, and $\mathsf{op} \in \{<, \leq, >, \geq, =\}$. Note that $\vec{a}\vec{x} = b$ can be represented by $\vec{a}\vec{x} \leq b \wedge \vec{a}\vec{x} \geq b$. In addition, $\vec{a}\vec{x} \geq b$ and $\vec{a}\vec{x} > b$ can be represented by $-\vec{a}\vec{x} \leq -b$ and $-\vec{a}\vec{x} < -b$ respectively. So it is sufficient to use $\mathsf{op} \in \{<, \leq\}$ to represent a LC.

An SMT(LA) formula ϕ with l Boolean variables and n numeric variables can be formally represented by PS_ϕ and H_ϕ, where $\mathrm{PS}_\phi(b_1, \ldots, b_{m+l})$ is a Boolean formula, $H_\phi = \{h_1, \ldots, h_m\}$ is a set of LCs. The Boolean formula PS_ϕ is also called the **propositional skeleton** of ϕ. The propositional skeleton contains logical operators, like AND, OR, NOT. A simple example of an SMT(LA) formula ϕ is

$$\phi \equiv (x + y < 1 \text{ OR } x \geq y) \text{ AND } (x + y < 1 \text{ OR } x < y \text{ OR } b).$$

Let the Boolean variables b_1 and b_2 represent the linear inequalities $h_1 \equiv x + y < 1$ and $h_2 \equiv x < y$ respectively. Then we obtain the propositional skeleton

$$PS_\phi \equiv (b_1 \text{ OR } (\text{NOT } b_2)) \text{ AND } (b_1 \text{ OR } b_2 \text{ OR } b).$$

[1)] Our tool SharpSMT and experimental data including benchmarks can be found at http://www.github.com/bearben/sharpsmt

Definition 1. Let b_i correspond to h_i, for each $1 \leq i \leq m$. Let $\mathtt{bool}(\vec{\alpha})$ represent $(b_{m+1}, \ldots, b_{m+l})$ which are pure Boolean variables of ϕ.

Definition 2. A (partial) assignment $\vec{\alpha}$ of PS_ϕ is a vector $(\alpha_1, \ldots, \alpha_{m+l}) \in \mathbb{B}^{m+l}$, where α_i is either 1 or 0 (or not assigned). It corresponds to $H_{\vec{\alpha}} = \bigcup_{1 \leq i \leq m} H_{\vec{\alpha},i}$, where

$$H_{\vec{\alpha},i} = \begin{cases} \{h_i\} & \text{if } \alpha_i = 1, \\ \{\neg h_i\} & \text{if } \alpha_i = 0, \\ \emptyset & \text{if } \alpha_i \text{ is not assigned.} \end{cases}$$

Definition 3. An assignment $\vec{m}$ of ϕ is in the form $(\vec{x}, \mathtt{bool}(\vec{\alpha})) \in \mathbb{R}^n \times \mathbb{B}^l$ or $\mathbb{Z}^n \times \mathbb{B}^l$, where $\vec{\alpha}$ is an assignment of PS_ϕ and $\vec{x}$ is a point in $\mathbb{R}^n$ or $\mathbb{Z}^n$.

The solution space of ϕ is then the union of sets, formally:

$$\mathcal{M}(\phi) = \bigcup_{\vec{\alpha} \in \mathcal{M}(\mathrm{PS}_\phi)} \mathcal{M}(H_{\vec{\alpha}}) \times \mathtt{bool}(\vec{\alpha}). \quad (1)$$

Note that $H_{\vec{\alpha}}$ can be regarded as a convex polytope. Intuitively, the solution space of an SMT(LA) formula can be viewed as a union of many polytopes.

Definition 4. Given a formula F and a polytope P,

- Let $vol(F)$ and $vol(P)$ denote the volume of solution space of F and the volume of P respectively.
- Let $\#F$ and $\#P$ denote the number of integer solutions of F and lattice points in P respectively.

2.2 DPLL(T)-based Enumeration Scheme

A well-known framework for finding models of SMT(LA) formulas is called DPLL(T) algorithm. Based on it, we present a scheme to find a set of (partial) assignments $\mathcal{A} \subset \mathcal{M}(\mathrm{PS}_\phi)$, s.t.,

$$\mathcal{M}(\phi) = \bigcup_{\vec{\alpha} \in \mathcal{A}} \mathcal{M}(H_{\vec{\alpha}}) \times \mathtt{bool}(\vec{\alpha})$$
$$\text{and } \mathcal{M}(H_{\vec{\alpha}}) \times \mathtt{bool}(\vec{\alpha}) \cap \mathcal{M}(H_{\vec{\beta}}) \times \mathtt{bool}(\vec{\beta}) = \emptyset,$$
$$\forall \vec{\alpha}, \vec{\beta} \in \mathcal{A}, \vec{\alpha} \neq \vec{\beta}. \quad (2)$$

- Step 1. Find a model $\vec{m}$ of ϕ by DPLL(T) algorithm.
- Step 2. From Equation 1, there exists a (partial) assignment $\vec{\alpha} \in \mathcal{M}(\mathrm{PS}_\phi)$, s.t., $\vec{m} \in \mathcal{M}(H_{\vec{\alpha}}) \times \mathtt{bool}(\vec{\alpha})$.
- Step 3. Construct a new formula $\phi' \equiv \phi \wedge NC(\vec{\alpha})$ which is the conjunction of ϕ and the negation clause $NC(\vec{\alpha})$ of $\vec{\alpha}$, The negation clause would prevent the DPLL(T)

	Vinci [19]	PolyVest [14,20]	LattE [21]	Barvinok [22]	ApproxLatCount [17]	Vol2Lat [15]
Input	P	P, ϵ, δ	P	P	P, ϵ, δ	P
Output	$vol(P)$	$\widetilde{vol(P)}$	$\#P$	$\#P$	$\widetilde{\#P}$	$\widetilde{\#P}, lb, ub$
Guarantee	Exact	(ϵ, δ)-bound	Exact	Exact	(ϵ, δ)-bound	$lb \le \widetilde{\#P} \le ub$

Table 1 A summary of polytope subroutines. Note that $\widetilde{vol(P)}$ and $\widetilde{\#P}$ are approximate results of $vol(P)$ and $\#P$ respectively. The (ϵ, δ)-bound of $\widetilde{vol(P)}$ guarantees the output lie in the interval $[(1+\epsilon)^{-1}vol(P), (1+\epsilon)vol(P)]$ with probability at least $1-\delta$. The (ϵ, δ)-bound of $\widetilde{\#P}$ is similar.

algorithm finding models in $\mathcal{M}(H_{\vec{\alpha}}) \times \texttt{bool}(\vec{\alpha})$ again. In detail, $NC(\vec{\alpha}) = \bigvee l_i$, where l_is are literals that $l_i \equiv b_i$ if $\alpha_i = 0$, and $l_i \equiv \neg b_i$ if $\alpha_i = 1$.

- Step 4. Find the next model $\vec{m}' \in \mathcal{M}(\phi')$ and a (partial) assignment $\vec{\alpha}'$, s.t., $\vec{m}' \in \mathcal{M}(H_{\vec{\alpha}'}) \times \texttt{bool}(\vec{\alpha})$. Repeat above steps until $\mathcal{M}(\phi') = \emptyset$, i.e., unsatisfiable.

In this way, we obtain a set $\mathcal{A} = \{\vec{\alpha}, \vec{\alpha}', \dots\}$ satisfying Equation 2. Note that such assignment $\vec{\alpha} \in \mathcal{A}$ can be partial assignments.

2.3 Polytope Subroutines

From Equation 2, $\{\mathcal{M}(H_{\vec{\alpha}}) \times \texttt{bool}(\vec{\alpha}), \forall \vec{\alpha} \in \mathcal{A}\}$ is mutually non-overlapping. Then the size of solution space is:

$$|\mathcal{M}(\phi)| = \sum_{\vec{\alpha}\in\mathcal{A}} |\mathcal{M}(H_{\vec{\alpha}}) \times \texttt{bool}(\vec{\alpha})| = \sum_{\vec{\alpha}\in\mathcal{A}} |\mathcal{M}(H_{\vec{\alpha}})| \cdot 2^{d_{\vec{\alpha}}}, \quad (3)$$

where $d_{\vec{\alpha}}$ is the number of α_is which are not assigned, $m+1 \le i \le l$. For real cases $\vec{x} \in \mathbb{R}^n$, $|\mathcal{M}(H_{\vec{\alpha}})|$ is the volume of a polytope $P_{\vec{\alpha}} = \{\vec{x} \in \mathbb{R}^n : H_{\vec{\alpha}}(\vec{x})\}$, i.e,

$$|\mathcal{M}(\phi)| = \sum_{\vec{\alpha}\in\mathcal{A}} vol(P_{\vec{\alpha}}) \cdot 2^{d_{\vec{\alpha}}}. \quad (4)$$

For integer cases $\vec{x} \in \mathbb{Z}^n$, $|\mathcal{M}(H_{\vec{\alpha}})| = P_{\vec{\alpha}} \cap \mathbb{Z}^n = \#P_{\vec{\alpha}}$ is the number of lattice points that lie in $P_{\vec{\alpha}}$, i.e.,

$$|\mathcal{M}(\phi)| = \sum_{\vec{\alpha}\in\mathcal{A}} \#P_{\vec{\alpha}} \cdot 2^{d_{\vec{\alpha}}}. \quad (5)$$

Therefore, we employ the various tools for computing integer solution points or volume with respect to a polytope, which are also called **polytope subroutines**. Table 1 gives the summary of all subroutines invoked by SharpSMT.

Vinci [19] and PolyVest [14, 20] are tools for computing or estimating the volume of a polytope, which will be invoked by SharpSMT for volume computation or estimation problems of SMT(LA) formulas. Since the algorithm of volume estimation is polynomial-time, thus PolyVest is able to solve larger cases (more variables) than Vinci. In practice, Vinci would solve a case in a few seconds, otherwise, it usually runs out of memory or eventually timeout. Naturally, it is wise to call Vinci first, if Vinci is not able to solve in seconds or runs out of memory, then SharpSMT calls PolyVest.

LattE [21], Barvinok [22], ApproxLatCount (ALC) [17] and Vol2Lat (V2L) [15] are tools for computing or estimating the count of lattices in a polytope. They will be employed for the integer solution counting of SMT(LA) formulas. LattE and Barvinok are both the implementations of Barvinok's algorithm. The tool Barvinok was released after LattE, which has an in general better performance. ALC and V2L are both approximate lattice counters, and are able to solve larger cases (more variables) than Barvinok. But their approximation methods are essentially different. ALC returns the approximation with an (ϵ, δ)-bound, which is similar with the volume estimation method PolyVest. V2L gives an exact bound of $vol(P)$ and $\#P$, if the bound is tight, then $vol(P) \approx \#P$, which means $\#P$ can be approximated by $vol(P)$. Note that it usually takes less than 0.1s to compute the bound by V2L, which is very efficient. Therefore, we have the following strategy: SharpSMT calls Barvinok first. If Barvinok is not able to solve in seconds, then it calls V2L to obtain a bound of $vol(P)$ and $\#P$. If the bound is tight, SharpSMT calls volume computation or estimation methods to approximate the integer solution count, otherwise, it calls ALC.

3 Architecture

Based on Equation 4 and 5, the brief idea of SharpSMT is enumerating feasible assignments by solving the SMT(LA) formula and then accumulating the volumes or lattice counts of polytopes corresponding to these assignments. A schematic overview of the architecture is presented in Fig. 1. A regular run of SharpSMT has four stages: parsing and rewriting, feasible assignments enumeration, polytope preprocessing and polytope subroutines.

3.1 Parsing and Rewritting

SharpSMT reads the input and recursively builds an abstract syntax DAG (directed acyclic graph). Like most SMT solvers, basic rewriting rules are applied to simplify the DAG during the parsing process.

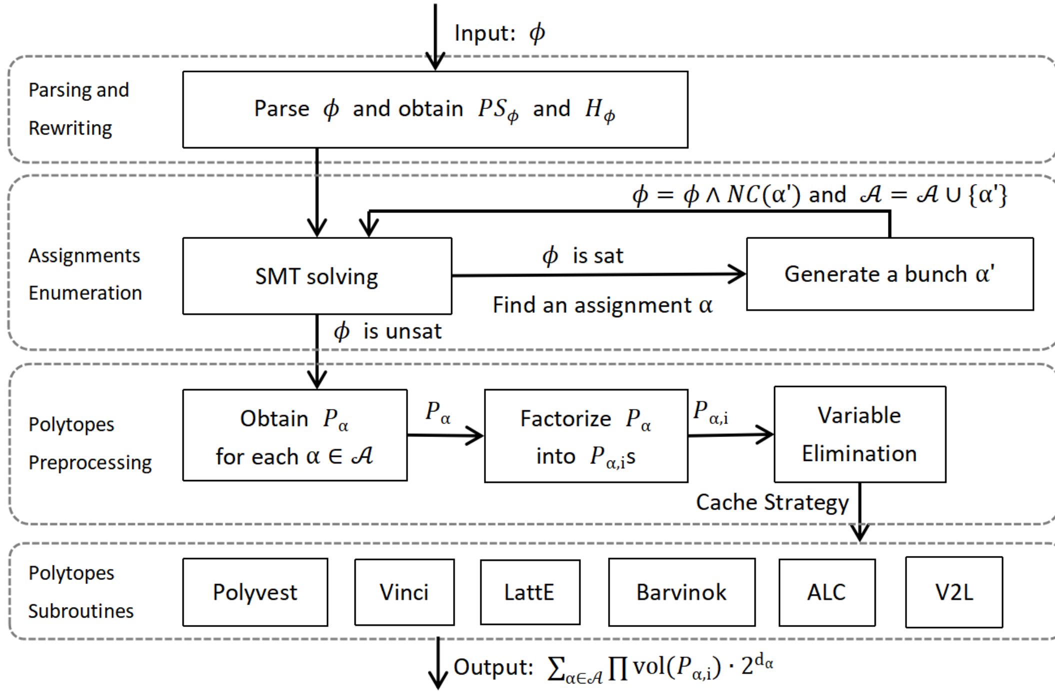

Fig. 1 Schematic overview of the architecture of SharpSMT.

• Inequality Extraction.

Recall that polytope subroutines only take linear inequalities as inputs, so SharpSMT has to extract inequalities from SMT(LA) formulas. The inequality extractor is a component to rewrite the SMT(LA) expression into a combination of Boolean skeletons and normalized linear inequalities. Parser calls extractor whenever it has parsed a comparison operator ($=, <, \leq, >, \geq$, distinct) which is the root of a DAG of inequalities. Note that such a DAG may consist of more than one inequality if it contains `if-then-else` (`ite`) operators. For a DAG with `ite` operators, the extractor recursively traverses the DAG, records the conditions of each `ite` and then reconstructs a syntax tree with Boolean variables which represents those extracted inequalities. Each time an inequality is extracted, SharpSMT creates a Boolean variable to substitute this inequality in the DAG and adds a constraint to link the variable with the inequality.

Fig. 2 is an example of the inequality extraction. Red nodes represent the roots of DAGs. The left-hand-side is a DAG of an input expression in SMT-LIB language: `(= (+ (ite a (ite b x y) (+ x y)) z) (- x))`. The right-hand-side are three syntax trees reconstructed by our inequality extractor. Note that `ieq1`, `ieq2` and `ieq3` represent the inequalities $2x + z = 0$, $x + y + z = 0$ and $2x + y + z = 0$ respectively. Then such syntax trees are Boolean skeletons. Moreover, the conjunction of these three trees is equivalent to the original expression.

• Inequality Normalization.

The inequality extraction also normalizes inequalities. With Boolean operations, one can represent $<, \leq, >$ and $\geq$ simply by $\leq$, e.g., $x + y < 0 \Leftrightarrow not\ (x + y \geq 0) \Leftrightarrow not\ (-x - y \leq 0)$. Thus a normalized inequality is of the form $a_0 + a_1x_1 + \cdots + a_nx_n = (\leq)\ 0$, where a_is are constants, x_is are variables, and $a_0 \geq 0$. Note that one can obtain a unique normalized inequality by keeping first non-zero multiplier to be one. SharpSMT does not normalize inequalities by this way, because it may cause rounding errors when SharpSMT calls polytope subroutines.

Example 1. Consider the following SMT(LRA) formula:

$$((p \oplus q \oplus x - y < 1) \wedge z \leq 0) \vee (x - y \geq 1 \wedge z > 1),$$

where p, q are Boolean variables, and $x, y, z \in [-2, 2]$ are real variables. With the inequality extraction and normalization, SharpSMT transforms this formula into a Boolean skeleton $((p \oplus q \oplus \neg b_1) \wedge b_2) \vee (b_1 \wedge \neg b_3))$ with inequalities $b_1 \equiv 1 - x + y \leq 0$, $b_2 \equiv z \leq 0$, and $b_3 \equiv -1 + z \leq 0$.

3.2 Feasible Assignments Enumeration

In the second stage, SharpSMT tries to find the set of feasible assignments $\mathcal{A}$ in Equation 4 and 5. It employs the enumera-

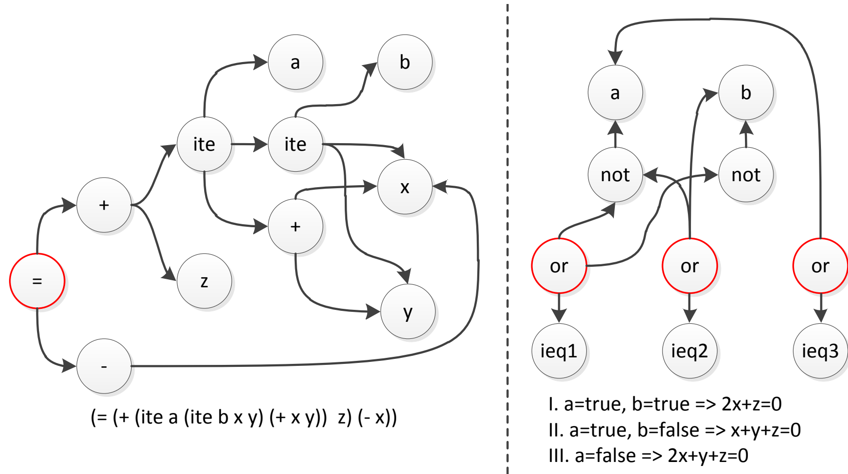

Fig. 2 An example of extracting linear inequalities from a formula which contains `ite` operators.

tion scheme which is shown in Section 2.2. It calls the SMT solver to obtain a feasible assignment $\vec{\alpha}$, then adds a negation clause $NC(\vec{\alpha})$ to rule out the assignment just found, and so forth. Currently, SharpSMT uses Z3, as the SMT(LA) solver, through APIs. Note that it is easy to employ other modern SMT solvers since SharpSMT sees SMT solving as a black box.

- Merge partial assignments into "bunches"

To reduce $|\mathcal{A}|$, SharpSMT employs a technique for merging partial assignments called Bunch strategy [13].

- Step 1. Find k s.t., α_k in $\vec{\alpha}$ is assigned and $\vec{\alpha}' \models \mathrm{PS}_\phi$, where $\vec{\alpha}' = (\alpha_1, \ldots, \neg\alpha_k, \ldots, \alpha_{m+l})$.
- Step 2. If such k exists, then $\vec{\beta} = (\beta_1, \ldots, \beta_{m+l})$, where $\beta_i = \alpha_i, \forall i \neq k$ and β_k is not assigned. Repeat above steps on the new partial assignment $\vec{\beta}$.
- Step 3. Otherwise, the method returns a bunch $\vec{\alpha}$.

Note that $\mathcal{M}(H_{\vec{\beta}}) = \mathcal{M}(H_{\vec{\alpha}}) \cup \mathcal{M}(H_{\vec{\alpha}'})$. It means $\vec{\beta}$ can be viewed as a merged result of $\vec{\alpha}$ and $\vec{\alpha}'$. Then a **bunch** is a partial assignment which cannot be further merged.

Example 2. The SMT formula in Example 1 has 7 feasible assignments as the following:

$$\{p, q, b_1, \neg b_2, \neg b_3\}, \{p, q, \neg b_1, b_2, b_3\}, \{p, \neg q, b_1, b_2, b_3\},$$
$$\{p, \neg q, b_1, \neg b_2, \neg b_3\}, \{\neg p, q, b_1, b_2, b_3\},$$
$$\{\neg p, q, b_1, \neg b_2, \neg b_3\}, \{\neg p, \neg q, \neg b_1, b_2, b_3\}.$$

Since $\{\neg p, \neg q, b_1, \neg b_2, \neg b_3\}$ satisfies the Boolean skeleton and the corresponding set of inequalities is not consistent, then partial assignment $\{b_1, \neg b_2, \neg b_3\}$ is also feasible. In addition, we have $vol(\{b_1, \neg b_2, \neg b_3\}) = vol(\{p, q, b_1, \neg b_2, \neg b_3\}) + vol(\{p, \neg q, b_1, \neg b_2, \neg b_3\}) + vol(\{\neg p, q, b_1, \neg b_2, \neg b_3\}) + vol(\{\neg p, \neg q, b_1, \neg b_2, \neg b_3\})$. Therefore, the bunch strategy reduces three feasible assignments $\{p, q, b_1, \neg b_2, \neg b_3\}$, $\{p, \neg q, b_1, \neg b_2, \neg b_3\}$ and $\{\neg p, q, b_1, \neg b_2, \neg e3\}$ into $\{b_1, \neg b_2, \neg b_3\}$. Note that $\neg z \leq 1$ implies $\neg z \leq 0$ and $z \leq 0$ implies $z \leq 1$. So `SharpSMT` will obtain 5 bunches at last:

$$\{b_1, \neg b_3\}, \{\neg p, q, b_1, b_2\}, \{p, \neg q, b_1, b_2\},$$
$$\{\neg p, \neg q, \neg b_1, b_2\}, \{p, q, \neg b_1, b_2\}.$$

3.3 Polytope Preprocessing

In the third stage, SharpSMT employs some preprocessing techniques on polytopes to improve the efficiency of polytope subroutines in the fourth stage. Given a partial assignment $\vec{\alpha} \in \mathcal{A}$, it corresponds to a polytope $P_{\vec{\alpha}}$. First, SharpSMT factorizes $P_{\vec{\alpha}}$ into small polytopes $P_{\vec{\alpha},1}, \ldots P_{\vec{\alpha},k}$. Then $vol(P_{\vec{\alpha}}) = \prod_{i=1}^{k} vol(P_{\vec{\alpha},i})$. After that, for each $P_{\vec{\alpha},i}$, SharpSMT reduces variables and obtains $P'_{\vec{\alpha},i}$. Then it searches in cache to determine whether $vol(P'_{\vec{\alpha},i})$ has been computed. If $P'_{\vec{\alpha},i}$ is a new polytope for subroutines, it calls subroutines and adds the result into cache. Finally, SharpSMT returns the final result $vol(\phi) = \sum_{\vec{\alpha} \in \mathcal{A}} \prod vol(P'_{\vec{\alpha},i}) \cdot 2^{d_{\vec{\alpha}}}$ by Equation 4. It is similar for computing $\#\phi$, according to Equation 5. In the rest of this section, we give the details of those preprocessing techniques.

- Factorization

The number of variables is a key factor about the difficulty of polytope subroutines. For example, the state-of-the-art exact integer counter or volume computation algorithms often have difficulties when the number of variables is greater than 15. For the state-of-the-art volume approximations, it takes minutes to solve problems with around 100 dimensions, which are still expensive. Therefore, reducing the number of variables will be beneficial to SharpSMT. Our tool employs polytope preprocessing techniques for dimension reduction.

It first factorizes the set of linear inequalities, which corresponds to a given bunch, into mutually independent groups of inequalities, then applies the polytope subroutine to obtain the size of solution space of each group of inequalities, and finally returns the product. SharpSMT also tries to check and reduce irrelevant variables.

Example 3. Consider a set of linear constraints

$$\begin{cases} x_1 + x_3 \le 10, \\ x_2 + x_4 + x_5 \ge 0, \\ 3x_4 - 2x_5 \le 10, \\ -8 \le x_i \le 7, i \in [1, 5]. \end{cases}$$

It can be factorized into two sets of variables $\{x_1, x_3\}$ and $\{x_2, x_4, x_5\}$ and two sets of constraints

$$\begin{cases} x_1 + x_3 \le 10, \\ -8 \le x_i \le 7, i \in \{1, 3\}, \end{cases} \quad \begin{cases} x_2 + x_4 + x_5 \ge 0, \\ 3x_4 - 2x_5 \le 10, \\ -8 \le x_i \le 7, i \in \{2, 4, 5\}. \end{cases}$$

Then the 5-dimensional polytope is thus factorized into two polytopes with at most 3 dimensions, which will save the computation cost of polytope subroutines.

Ge and Biere [16] proposed more sophisticated factorization techniques, but they are not integrated in SharpSMT so far. It will be our future work.

- Variable Elimination

Given a set of inequalities $H_{\vec{\alpha}}$, which corresponds to a feasible assignment $\vec{\alpha}$, it may contains equalities, i.e., $\vec{a}\vec{x} \le b \in H_{\vec{\alpha}}$ and $\vec{a}\vec{x} \ge b \in H_{\vec{\alpha}}$. In such cases, the corresponding polytope $P_{\vec{\alpha}}$ is degenerate and $vol(P_{\vec{\alpha}}) = 0$. To count the integer solutions $\#P_{\vec{\alpha}}$, we can naturally reduce one variable for an equality constraint by Gauss elimination. For example, assume a_1 is non-zero, we can eliminate x_1 by equation $x_1 = (b - a_2x_2 - \cdots - a_nx_n)/a_1$. Obviously, Gauss elimination will not change the count of integer solutions.

Specifically, the subroutine V2L, which approximates integer solution counts via polytopes' volume, relies on Gauss elimination. Since $vol(P_{\vec{\alpha}}) = 0$, the volume cannot be used to approximate the integer count when $\#P_{\vec{\alpha}} > 0$. However, it is probably that $vol(P') \approx \#P' = \#P_{\vec{\alpha}}$, where P' is the polytope generated by Gauss elimination.

- Reusing with Cache

SharpSMT stores the results of calls of polytope subroutines in a cache, so that it can retrieve the result of a polytope which has already been calculated from the cache. For example, $\{\neg p, \neg q, \neg b_1, b_2\}$ and $\{p, q, \neg b_1, b_2\}$ are two different bunches in Example 2, but both correspond to the polytope $\{\neg b_1 \wedge b_2, x, y, z \in [-2, 2]\}$. Recall that SharpSMT has extracted and stored all inequalities, the polytope passed to a subroutine is essentially a feasible assignment, which corresponds to the polytope. So we set the feasible assignment to be the key and bind it to the results returned by the polytope subroutine.

Besides, the factorization technique on polytopes may create same sub-polytopes when factorizing different polytopes. For example, let us consider two bunches $\{b_1, \neg b_3\}$ and $\{p, \neg q, b_1, b_2\}$ in Example 2, which correspond to $P_1 = \{b_1 \wedge \neg b_3, x, y, z \in [-2, 2]\}$ and $P_2 = \{b_1 \wedge b_2, x, y, z \in [-2, 2]\}$ respectively. P_1 can be factorized into $P_{11} = \{b_1, x, y \in [-2, 2]\}$ and $P_{12} = \{\neg b_3, z \in [-2, 2]\}$, and $vol(P_1) = vol(P_{11}) \times vol(P_{12})$. Similarly, P_2 can be factorized into $P_{21} = \{b_1, x, y \in [-2, 2]\}$ and $P_{22} = \{b_2, z \in [-2, 2]\}$. Note that $P_{11} = P_{21}$, so the result of P_{11} can be reused when handling P_2.

- Interval Computation

One-dimensional problems are trivial, but all polytope subroutines do not have special treatments for this. So SharpSMT will handle such cases directly. The one-dimensional polytope is an interval, so we only have to calculate the upper and lower bounds by checking the corresponding inequalities.

- Two-round Strategy [14]

For PolyVest and ALC, as the number of samplings increases, the accuracy of estimation improves, and the estimation process also takes more time. It is important to balance the accuracy and the running time since the subroutines are usually called many times. Therefore, SharpSMT employs a **two-round strategy** [14] that can dynamically determine a proper weight for sampling. At the first round of estimation, each feasible assignment is generated with a fixed small weight to get a quick and rough estimation. Since the volumes (or lattice counts) of polytopes may vary a lot. Intuitively, a feasible assignment with relatively larger volume should be estimated with higher accuracy. Hence in the second round, the weight for each assignment is determined according to its estimated volume from the first round.

Example 4. Recall the bunches listed in Example 2: $\{b_1, \neg b_3\}$, $\{\neg p, q, b_1, b_2\}$, $\{p, \neg q, b_1, b_2\}$, $\{\neg p, \neg q, \neg b_1, b_2\}$, $\{p, q, \neg b_1, b_2\}$. Consider the volume of the first bunch $\{b_1, \neg b_3\}$. SharpSMT factorizes the corresponding polytope $P_1 = \{b_1 \wedge \neg b_3, x, y, z \in [-2, 2]\}$ into $P_{11} = \{b_1, x, y \in [-2, 2]\}$ and $P_{12} = \{\neg b_3, z \in [-2, 2]\}$, where $vol(P_1) = vol(P_{11}) \times vol(P_{12})$. Then it calls

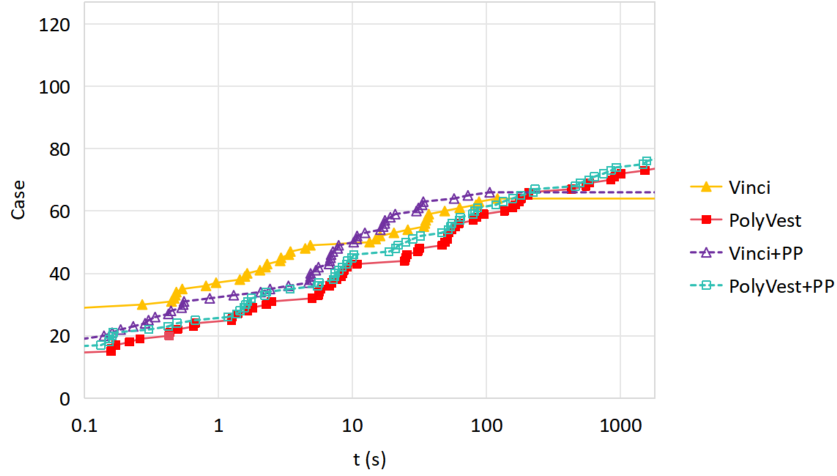

Fig. 3 Performance comparisons among subroutines of SharpSMT on random instances (real variables).

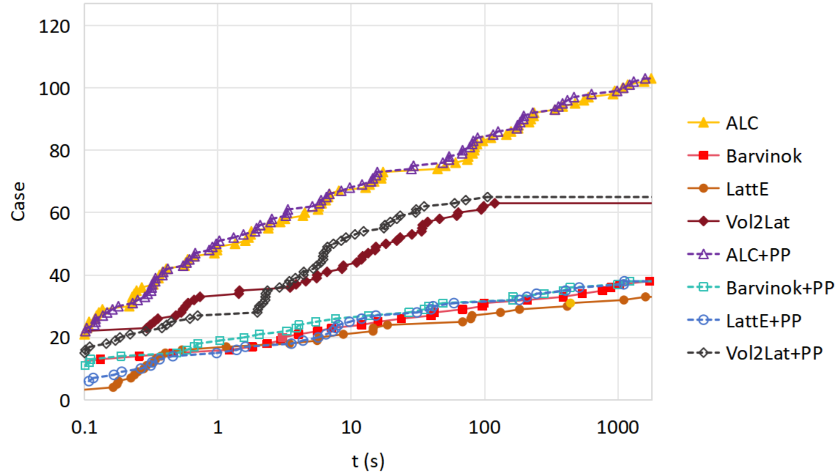

Fig. 4 Performance comparisons among subroutines of SharpSMT on random instances (integer variables).

polytope subroutines on P_{11} and computes the interval of P_{12} directly. Finally, SharpSMT obtains the volume $vol(P_1) = 4.5$, and $vol(\{b_1, \neg b_3\}) = 4 \times vol(P_1) = 18$ as there are 4 possible assignments for pair (a, b). Now let us consider the second bunch $\{\neg p, q, b_1, b_2\}$. The corresponding polytope $P_2 = \{b_1 \wedge b_2, x, y, z \in [-2, 2]\}$ can also be factorized into two polytopes $P_{21} = \{b_1, x, y \in [-2, 2]\}$ and $P_{22} = \{b_2, z \in [-2, 2]\}$. Since $P_{21} = P_{11}$, SharpSMT reuses the result of $vol(P_{11})$. At last, SharpSMT obtains the result $vol(\{\neg p, q, b_1, b_2\}) = 4.5 \times vol(P_{22}) = 9$. Similarly, SharpSMT computes the results of the rest bunches $\{p, \neg q, b_1, b_2\}$, $\{\neg p, \neg q, \neg b_1, b_2\}$ and $\{p, q, \neg b_1, b_2\}$, and sums up. So the total volume of this formula is $18 + 9 + 9 + 23 + 23 = 82$.

4 Evaluation

In this section, we report experimental results about the performance of SharpSMT. SharpSMT provides a command-line parameter `-w` to quickly set bounds to numeric variables. Specifically, with such word-length parameter w, SharpSMT add the bound $x_i \in [-2^{w-1}, 2^{w-1} - 1]$ for each variable x_i. We chose $w = 8$, i.e., $x_i \in [-128, 127]$, for all unbounded SMT(LA) formulas. For (ϵ, δ)-counters, PolyVest and ALC, we chose $\epsilon = 0.2$ and $\delta = 0.1$. Experiments were conducted on Intel(R) Xeon(R) Gold 6248 @ 2.50GHz CPUs with a time limit of 1800 seconds and memory limit of 4 GB per benchmark. The benchmark set consists of:

- Random instances: We generated 126 SMT(LRA) and SMT(LIA) formulas by randomly choosing coefficients

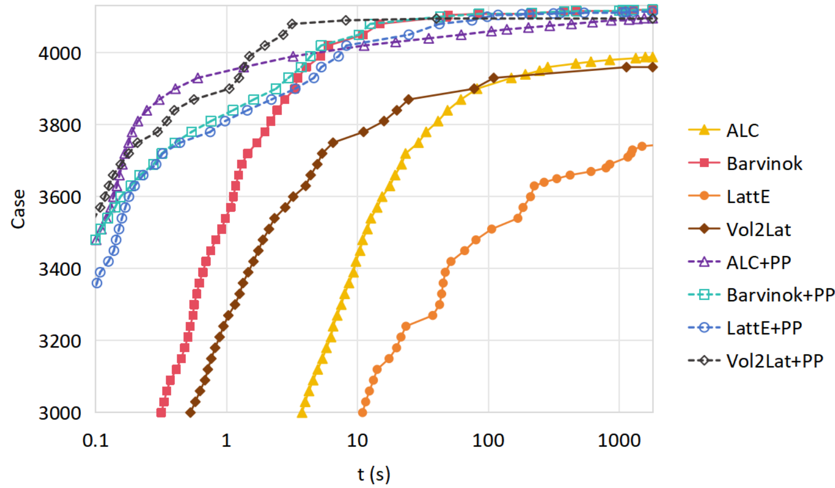

Fig. 5 Performance comparisons among subroutines of SharpSMT on application instances.

of inequalities and literals of clauses. They are named as `lra_b_n_l` or `lia_b_n_l`, which have *b* Boolean variables, *n* real variables and *l* inequalities.

- Application instances: We adopted 4131 benchmarks [16, 17] from program analysis and simple temporal planning (STN).
- Bit-vector formulas: We translated some small-scale random and application instances from SMT(LIA) formulas into SMT(BV) formulas for comparing SharpSMT with hashing-based #SMT(BV) counters.

We conducted experimental on random and application instances with different polytopes subroutines. We also compared the performance of SharpSMT with and without polytope preprocessing techniques. Fig. 3 and 4 present the results on random instances. Fig. 5 presents the results on application instances. In these figures, the x-axis means the running time and presented in logarithmic. Note that "X+PP" are the results of SharpSMT calls the single subroutine "X" with polytope preprocessing (PP) techniques.

From Fig. 3 and 4, we observe that the polytope preprocessing techniques could improve the performance of polytope subroutines a bit. The exact volume computation tool Vinci works better on relative small instances than volume approximation tool PolyVest, on the other hand, PolyVest can solve more larger instances. By comparing lattice counter ALC, Barvinok, LattE and V2L in Fig. 4, we find that ALC is outperforms all other tools, and V2L is the second, and Barvinok is slightly better than LattE. It is reasonable as ALC and V2L are approximate counters.

From Fig. 5, it is obvious that the polytope preprocessing (PP) techniques are very effective. With PP techniques, all subroutines could handle more than 3800 instances (4131 in total) in one second. Moreover, all subroutines with PP could more cases in 1800s timeout. On these application benchmarks, we find that Barvinok is overall the best and then V2L and ALC.

- Comparing with Hashing-based Counters

We compared SharpSMT with SMTApproxMC [18] which is a hashing-based approximate counter for SMT(BV) formulas. For comparison, we translated SMT(LIA) formulas into SMT(BV) formulas semi-manually by replacing integer variables with fixed-length variables, bit-vector constants and bit operations. We experimented SMTApproxMC with parameters $\epsilon = 0.8$ and $\delta = 0.2$. It guarantees the output lying in interval $[1.8^{-1}\#F, 1.8\#F]$ with probability at least 80%, where $\#F$ is the exact solution count of a given formula F. Table 2 presents the result of comparing the performance of SharpSMT with SMTApproxMC on a subset of our benchmarks. In these experiments, SharpSMT calls LattE for integer solution counting inside a polytope, so our tool returns the exact counts instead of estimations. Table 2 shows that our approach significantly outperforms SMTApproxMC. We observe that the running time of SMTApproxMC is closely related to the number of the solutions rather than the number of variables, i.e., the larger number of solutions, the more difficult for SMTApproxMC to handle. Note that Ge [17] also compared Barvinok and ALC with more hashing-based model counters, ApproxMC4 [23], Cachet [24], and Ganak [25]. The experimental results show that Barvinok and ALC are more suitable for counting integer solutions on linear con-

			SharpSMT(ALC)		SharpSMT(Barvinok)		SMTApproxMC	
Instance	#var	#ineq	result $\widetilde{\#\phi}$	t (s)	result $\#\phi$	t (s)	result $\widetilde{\#\phi}$	t (s)
getopPath_1	1	10	242	0.01	242	0.01	242	2.57
getopPath_2	3	20	8085	0.01	8085	0.01	8.38E+03	14.96
findmiddle_4	3	6	5.76e+06	0.02	5.53e+06	0.02	——	——
findmiddle_6	3	6	1.31e+05	0.03	1.31e+05	0.02	1.33e+05	151.5
findmiddle_8	3	6	6.63e+04	0.01	6.53e+04	0.02	6.21e+04	207.9
Space_manage_38	7	15	6.31e+14	0.02	5.41e+14	0.02	——	——
Space_manage_49	13	20	1.94e+27	0.05	2.51e+27	0.05	——	——
tcas_1201	7	18	1.10e+12	0.01	1.10e+12	0.01	——	——
tcas_1214	7	18	4.28e+09	0.01	4.28e+09	0.01	——	——
lia_10_10_5	10	5	1.20e+23	0.03	1.19e+23	22.16	——	——
lia_7_7_3	7	3	2.77e+16	0.02	2.48e+16	2.25	——	——
lia_6_6_3	6	3	9.23e+13	0.02	9.79e+13	8.08	——	——
lia_5_5_2	5	2	9.82e+10	0.02	9.73e+10	0.04	——	——
lia_8_4_4	4	4	1.19e+09	0.08	1.14e+09	1.04	——	——
lia_6_3_3	3	3	1.33e+06	0.01	1.28e+06	0.02	1.12e+06	878.01
lia_3_3_3	3	3	2.97e+06	0.03	2.97e+06	0.02	2.91e+06	909.15
lia_2_2_2	2	2	9.52e+04	0.02	9.43e+04	0.02	8.74e+04	21.86
lia_2_2_1	2	1	6.70e+04	0.02	6.55e+04	0.02	6.50e+04	22.41

Table 2 Comparison results of SharpSMT and `SMTApproxMC` on small-scale instances.

straints. And hashing-based counters only gains upper hand on cases with very small domains, such as, $x_i \in \{-1, 0, 1\}$.

5 Conclusion and Future Work

In this paper, we introduced our tool SharpSMT for computing the volume of the solution space (or counting the number of integer solutions), given an SMT(LA) formula which is a Boolean combination of linear arithmetic inequalities. SharpSMT employs the DPLL(T) algorithm for feasible assignment enumeration, then it calls various polytope subroutines for different tasks. We also proposed a series of new polytope preprocessing techniques and implemented them in SharpSMT. Experimental results show that the new polytope preprocessing techniques are very effective, especially on application instances. We believe that the tool will be useful in a number of domains, such as program analysis and probabilistic verification.

Ge and Biere [16] proposed more sophisticated factorization techniques, but we only implement a direct factorization method in SharpSMT so far. It will be our future work. The two-round strategy [14] has been shown effective with volume approximation method PolyVest. Intuitively, this technique may also work with approximate lattice counting method, such as, ALC and V2L. But SharpSMT only employs two-round strategy with PolyVest so far. We will consider this in the future.

Acknowledgements Cunjing Ge is supported by the National Natural Science Foundation of China (62202218), and is sponsored by CCF-Huawei Populus Grove Fund (CCF-HuaweiFM202309).

References

1. Barrett C, Conway C L, Deters M, Hadarean L, Jovanovic D, King T, Reynolds A, Tinelli C. CVC4. In: Proceedings of Computer Aided Verification - 23rd International Conference (CAV). 2011, 171–177
2. 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, Proceedings of Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference (TACAS). 2022, 415–442
3. Cimatti A, Griggio A, Schaafsma B J, Sebastiani R. The mathsat5 SMT solver. In: Piterman N, Smolka S A, eds, Proceedings of Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference (TACAS). 2013, 93–107
4. Dutertre B. Yices 2.2. In: Proceedings of Computer Aided Verification - 26th International Conference (CAV). 2014, 737–744
5. Moura d L M, Bjørner N. Z3: an efficient SMT solver. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems - 14th International Conference (TACAS). 2008, 337–340
6. Chavira M, Darwiche A. On probabilistic inference by weighted model counting. Artificial Intelligence, 2008, 172(6-7): 772–799
7. Belle V, Passerini A, Broeck d G V. Probabilistic inference in hybrid domains by weighted model integration. In: Yang Q, Wooldridge M J, eds, Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI). 2015, 2770–2776

8. Zanarini A, Pesant G. Solution counting algorithms for constraint-centered search heuristics. In: Proceedings of Principles and Practice of Constraint Programming (CP). 2007, 743–757
9. Pesant G. Counting-based search for constraint optimization problems. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. 2016, 3441–3448
10. Huang A, Lloyd L, Omar M, Boerkoel J C. New perspectives on flexibility in simple temporal planning. In: Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS). 2018, 123–131
11. Geldenhuys J, Dwyer M B, Visser W. Probabilistic symbolic execution. In: Proceedings of International Symposium on Software Testing and Analysis (ISSTA). 2012, 166–176
12. Luckow K S, Pasareanu C S, Dwyer M B, Filieri A, Visser W. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In: Proceedings of ACM/IEEE International Conference on Automated Software Engineering (ASE). 2014, 575–586
13. Ma F, Liu S, Zhang J. Volume computation for boolean combination of linear arithmetic constraints. In: Proceedings of the 22nd International Conference on Automated Deduction (CADE). 2009, 453–468
14. Ge C, Ma F, Zhang P, Zhang J. Computing and estimating the volume of the solution space of SMT(LA) constraints. Theoretical Computer Science, 2018, 743: 110–129
15. Ge C, Ma F, Ma X, Zhang F, Huang P, Zhang J. Approximating integer solution counting via space quantification for linear constraints. In: Kraus S, ed, Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI). 2019, 1697–1703
16. Ge C, Biere A. Decomposition strategies to count integer solutions over linear constraints. In: Zhou Z, ed, Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI). 2021, 1389–1395
17. Ge C. Approximate integer solution counts over linear arithmetic constraints. In: Wooldridge M J, Dy J G, Natarajan S, eds, Proceedings of Thirty-Eighth AAAI Conference on Artificial Intelligence. 2024, 8022–8029
18. Chakraborty S, Meel K S, Mistry R, Vardi M Y. Approximate probabilistic inference via word-level counting. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. 2016, 3218–3224
19. Büeler B, Enge A, Fukuda K. In: Kalai G, Ziegler G M, eds. Exact Volume Computation for Polytopes: A Practical Study. Birkhäuser Basel, 2000, 131–154
20. Ge C, Ma F. A fast and practical method to estimate volumes of convex polytopes. In: Proceedings of Frontiers in Algorithmics - 9th International Workshop (FAW). 2015, 52–65
21. Loera J A D, Hemmecke R, Tauzer J, Yoshida R. Effective lattice point counting in rational convex polytopes. Journal of Symbolic Computation, 2004, 38(4): 1273–1302
22. Verdoolaege S, Seghir R, Beyls K, Loechner V, Bruynooghe M. Counting integer points in parametric polytopes using barvinok's rational functions. Algorithmica, 2007, 48(1): 37–66
23. Soos M, Meel K S. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence. 2019, 1592–1599
24. Sang T, Bacchus F, Beame P, Kautz H A, Pitassi T. Combining component caching and clause learning for effective model counting. In: Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT). 2004
25. Sharma S, Roy S, Soos M, Meel K S. GANAK: A scalable probabilistic exact model counter. In: Kraus S, ed, Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI). 2019, 1169–1176

Cunjing Ge is a PostDoc in School of Artificial Intelligence, Nanjing University, China. He received his Ph.D. degree in Computer Software and Theory from Institute of Software, Chinese Academy of Sciences. His research interests including constraint satisfaction problem, model counting, and abductive learning.