Approximate Counting in SMT and Value Estimation for Probabilistic Programs

Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar

Max Planck Institute for Software Systems (MPI-SWS), Germany
{dch,rayna,rupak}@mpi-sws.org

Abstract. #SMT, or model counting for logical theories, is a well-known hard problem that generalizes such tasks as counting the number of satisfying assignments to a Boolean formula and computing the volume of a polytope. In the realm of satisfiability modulo theories (SMT) there is a growing need for model counting solvers, coming from several application domains (quantitative information flow, static analysis of probabilistic programs). In this paper, we show a reduction from an approximate version of #SMT to SMT.
We focus on the theories of integer arithmetic and linear real arithmetic. We propose model counting algorithms that provide approximate solutions with formal bounds on the approximation error. They run in polynomial time and make a polynomial number of queries to the SMT solver for the underlying theory, exploiting "for free" the sophisticated heuristics implemented within modern SMT solvers. We have implemented the algorithms and used them to solve a value estimation problem for a model of loop-free probabilistic programs with nondeterminism.

1 Introduction

Satisfiability modulo theories (SMT) is nowadays ubiquitous, and the research landscape is not only enjoying the success of existing SMT solvers, but also generating demand for new features. In particular, there is a growing need for *model counting* solvers; for example, questions in quantitative information flow and in static analysis of probabilistic programs are naturally cast as instances of model counting problems for appropriate logical theories [15,25,28].

We define the #SMT problem that generalizes several model counting questions relative to logical theories, such as computing the number of satisfying assignments to a Boolean formula (#SAT) and computing the volume of a bounded polyhedron in a finite-dimensional real vector space. Specifically, to define model counting modulo a *measured theory*, first suppose every variable in a logical formula comes with a domain which is also a measure space. Assume that, for every logical formula φ in the theory, the set of its models $[\![\varphi]\!]$ is measurable with respect to the product measure; the *model counting (or* #SMT*)* problem then asks, given φ, to compute the measure of $[\![\varphi]\!]$, called the *model count* of φ.

In our work we focus on the model counting problems for theories of bounded integer arithmetic and linear real arithmetic. These problems are complete for the complexity class #**P**, so fast exact algorithms are unlikely to exist.

We extend to the realm of SMT the well-known hashing approach from the world of #SAT, which reduces *approximate* versions of counting to decision problems. From a theoretical perspective, we solve a model counting problem with a resource-bounded algorithm that has access to an oracle for the decision problem. From a practical perspective, we show how to use unmodified existing SMT solvers to obtain approximate solutions to model-counting problems. This reduces an approximate version of #SMT to SMT.

Specifically, for integer arithmetic (not necessarily linear), we give a randomized algorithm that approximates the model count of a given formula φ to within a multiplicative factor $(1+\varepsilon)$ for any given $\varepsilon > 0$. The algorithm makes $O(\frac{1}{\varepsilon}\,|\varphi|)$ SMT queries of size at most $O(\frac{1}{\varepsilon^2}\,|\varphi|^2)$ where $|\varphi|$ is the size of φ.

For linear real arithmetic, we give a randomized algorithm that approximates the model count with an additive error γN, where N is the volume of a box containing all models of the formula, and the coefficient γ is part of the input. The number of steps of the algorithm and the number of SMT queries (modulo the combined theory of integer and linear real arithmetic) are again polynomial.

As an application, we show how to solve the value estimation problem [28] for a model of loop-free probabilistic programs with nondeterminism.

Techniques. Approximation of $\#\mathbf{P}$ functions by randomized algorithms has a rich history in complexity theory [31,34,21,20]. Jerrum, Valiant, and Vazirani [21] described a hashing-based $\mathbf{BPP}^{\mathbf{NP}}$ procedure to approximately compute any $\#\mathbf{P}$ function, and noted that this procedure already appeared implicitly in previous papers by Sipser [30] and Stockmeyer [31]. The procedure works with encoded computations of a Turing machine and is thus unlikely to perform well in practice. Instead, we show a direct reduction from approximate model counting to SMT solving, which allows us to retain the structure of the original formula. An alternate approach could eagerly encode #SMT problems into #SAT, but experience with SMT solvers suggests that a "lazy" approach may be preferable for some problems.

For the theory of linear real arithmetic, we also need an ingredient to handle continuous domains. Dyer and Frieze [11] suggested a discretization that introduces bounded additive error; this placed approximate volume computation for polytopes—or, in logical terms, approximate model counting for quantifier-free linear real arithmetic—in $\#\mathbf{P}$. Motivated by the application in the analysis of probabilistic programs, we extend this technique to handle formulas with existentially quantified variables. To this end, we prove a geometric result that bounds the effect of projections: this gives us an approximate model counting procedure for existentially quantified linear arithmetic formulas. Note that applying quantifier elimination as a preprocessing step may make the resulting formula exponential; instead, our approach works directly on the original formula that contains existentially quantified variables.

We have implemented our algorithm on top of the Z3 SMT solver and applied it to formulas encoding the value estimation problem for probabilistic programs. Our initial experience suggests that simple randomized algorithms using off-the-shelf SMT solvers can be reasonably effective.

Related work. #SMT is a well-known hard problem whose instances have been studied before, e. g., in volume computation [11], in enumeration of lattice points in integer polyhedra [1], and as #SAT [17]. Indeed, very simple subproblems, such as counting the number of satisfying assignments of a Boolean formula or computing the volume of a union of axis-parallel rectangles in $\mathbb{R}^n$ are already #**P**-hard (see Section 2 below).

Existing techniques for #SMT either incorporate model counting primitives into propositional reasoning [26,35] or are based on enumerative combinatorics [23,25,15]. Typically, exact algorithms [23,26,15] are exponential in the worst case, whereas approximate algorithms [25,35] lack provable performance guarantees. In contrast to exact counting techniques, our procedure is easily implementable and uses "for free" the sophisticated heuristics built in off-the-shelf SMT solvers. Although the solutions it produces are not exact, they provably meet user-provided requirements on approximation quality. This is achieved by extending the hashing approach from SAT [16,17,7,13] to the SMT context.

A famous result of Dyer, Frieze, and Kannan [12] states that the volume of a convex polyhedron can be approximated with a multiplicative error in probabilistic polynomial time (without the need for an SMT solver). In our application, analysis of probabilistic programs, we wish to compute the volume of a projection of a Boolean combination of polyhedra; in general, it is, of course, non-convex. Thus, we cannot apply the volume estimation algorithm of [12], so we turn to the "generic" approximation of #**P** using an **NP** oracle instead. Our #SMT procedure for linear real arithmetic allows an additive error in the approximation; it is unknown if the exact volume of a polytope has a small representation [11].

An alternative approach to approximate #SMT is to apply Monte Carlo methods for volume estimation. They can easily handle complicated measures for which there is limited symbolic reasoning available. Like the hashing technique, this approach is also exponential in the worst case [20]: suppose the volume in question, p, is very small and the required precision is a constant multiple of p. In this case, Chernoff bound arguments would suggest the need for $\Omega(\frac{1}{p})$ samples; the hashing approach, in contrast, will perform well. So, while in "regular" settings (when p is non-vanishing) the Monte Carlo approach performs better, "singular" settings (when p is close to zero) are better handled by the hashing approach. The two techniques, therefore, are complementary to each other.

Contributions. We extend, from SAT to SMT, the hashing approach to approximate model counting:

1. We formulate the notion of a measured theory (Section 2) that gives a unified framework for model-counting problems.
2. For the theory of bounded integer arithmetic, we provide a direct reduction (Theorem 1 in Section 2) from approximate counting to SMT.
3. For the theory of bounded linear real arithmetic, we give a technical construction (Lemma 1 in subsection 3.2) that lets us extend the results of Dyer and Frieze to the case when the polytope is given as a projection of a Boolean combination of polytopes; this leads to an approximate model counting procedure for this theory (Theorem 2 in Section 2).

4. As an application, we solve the value estimation problem for small loop-free probabilistic programs with nondeterminism (Section 4).

An extended version of the paper is available as [8].

2 The #SMT Problem

We present a framework for a uniform treatment of model counting both in discrete theories like SAT (where it is literally counting models) and in linear real arithmetic (where it is really volume computation for polyhedra). We then introduce the notion of approximation and show an algorithm for approximate model counting by reduction to SMT.

Preliminaries: Counting Problems and #P. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a *p-relation* if (1) there exists a polynomial $p(n)$ such that if $(x, y) \in R$ then $|y| = p(|x|)$ and (2) the predicate $(x, y) \in R$ can be checked in deterministic polynomial time in the size of x. Intuitively, a p-relation relates inputs x to solutions y. It is easy to see that a decision problem L belongs to **NP** if there is a p-relation R such that $L = \{x \mid \exists y. R(x, y)\}$.

A *counting problem* is a function that maps Σ^* to $\mathbb{N}$. A counting problem $f\colon \Sigma^* \to \mathbb{N}$ belongs to the class #**P** if there exists a p-relation R such that $f(x) = |\{y \mid R(x, y)\}|$, i. e., the class #**P** consists of functions that count the number of solutions to a p-relation [33]. *Completeness* in #**P** is with respect to Turing reductions; the same term is also (ab)used to encompass problems that reduce to a fixed number of queries to a #**P** function (see, e. g., [11]).

#SAT is an example of a #**P**-complete problem: it asks for the number of satisfying assignments to a Boolean formula in CNF [33]. Remarkably, #**P** characterizes the computational complexity not only of "discrete" problems, but also of problems involving real-valued variables: approximate volume computation (with additive error) for bounded rational polyhedra in $\mathbb{R}^k$ is #**P**-complete [11].

Measured Theories and #SMT. Suppose $\mathcal{T}$ is a logical theory. Let $\varphi(x)$ be a formula in this theory with free first-order variables $x = (x_1, \ldots, x_k)$. Assume that $\mathcal{T}$ comes with a fixed interpretation, which specifies domains of the variables, denoted $D_1, \ldots, D_k$, and assigns a meaning to predicates and function symbols in the signature of $\mathcal{T}$. Then a tuple $a = (a_1, \ldots, a_k) \in D_1 \times \ldots \times D_k$ is called a *model* of φ if the sentence $\varphi(a_1, \ldots, a_k)$ holds, i. e., if $a \models_{\mathcal{T}} \varphi(x)$. We denote the set of all models of a formula $\varphi(x)$ by $[\![\varphi]\!]$; the *satisfiability problem* for $\mathcal{T}$ asks, for a formula φ given as input, whether $[\![\varphi]\!] \neq \emptyset$.

Consider the special cases of (propositional) SAT and linear real arithmetic. For SAT, atomic predicates are of the form $x_i = b$, for $b \in \{0, 1\}$, the domain D_i of each x_i is $\{0, 1\}$, and formulas are propositional formulas in conjunctive normal form. For linear real arithmetic, atomic predicates are of the form $c_1 x_1 + \ldots + c_k x_k \leqslant d$, for $c_1, \ldots, c_k, d \in \mathbb{R}$, the domain D_i of each x_i is $\mathbb{R}$, and formulas are conjunctions of atomic predicates. Sets $[\![\varphi]\!]$ in these cases are the set of satisfying assignments and the polyhedron itself, respectively.

Suppose the domains $D_1, \ldots, D_k$ given by the fixed interpretation are measure spaces: each D_i is associated with a σ-algebra $\mathcal{F}_i \subseteq 2^{D_i}$ and a measure

$\mu_i \colon \mathcal{F}_i \to \mathbb{R}$. This means, by definition, that $\mathcal{F}_i$ and μ_i satisfy the following properties: $\mathcal{F}_i$ contains $\emptyset$ and is closed under complement and countable unions, and μ_i is non-negative, assigns 0 to $\emptyset$, and is σ-additive.

In our special cases, these spaces are as follows. For SAT, each $\mathcal{F}_i$ is the set of all subsets of $D_i = \{0, 1\}$, and $\mu_i(A)$ is simply the number of elements in A. For linear real arithmetic, each $\mathcal{F}_i$ is the set of all Borel subsets of $D_i = \mathbb{R}$, and μ_i is the Lebesgue measure.

Assume that each measure μ_i is σ-finite, that is, the domain D_i is a countable union of measurable sets (i. e., of elements of $\mathcal{F}_i$, and so with finite measure associated with them). This condition, which holds for both special cases, implies that the Cartesian product $D_1 \times \ldots \times D_k$ is measurable with respect to a unique *product measure* μ, defined as follows. A set $A \subseteq D_1 \times \ldots \times D_k$ is *measurable* (that is, μ assigns a value to A) if and only if A is an element of the smallest σ-algebra that contains all sets of the form $A_1 \times \ldots \times A_k$, with $A_i \in \mathcal{F}_i$ for all i. For all such sets, it holds that $\mu(A_1 \times \ldots \times A_k) = \mu_1(A_1) \ldots \mu_k(A_k)$.

In our special cases, the product measure $\mu(A)$ of a set A is the number of elements in $A \subseteq \{0, 1\}^k$ and the volume of $A \subseteq \mathbb{R}^k$, respectively.

We say that the theory $\mathcal{T}$ is *measured* if for every formula $\varphi(x)$ in $\mathcal{T}$ with free (first-order) variables $x = (x_1, \ldots, x_k)$ the set $[\![\varphi]\!]$ is measurable. We define the *model count* of a formula φ as $\mathsf{mc}(\varphi) = \mu([\![\varphi]\!])$. Naturally, if the measures in a measured theory can assume non-integer values, the model count of a formula is not necessarily an integer. With every measured theory we associate a *model counting problem*, denoted #SMT[$\mathcal{T}$]: the input is a logical formula $\varphi(x)$ in $\mathcal{T}$, and the goal is to compute the value $\mathsf{mc}(\varphi)$.

The #SAT and volume computation problems are just special cases as intended, since $\mathsf{mc}(\varphi)$ is equal to the number of satisfying assignments of a Boolean formula and to the volume of a polyhedron, respectively.

Approximate Model Counting. We now introduce *approximate* #SMT and show how approximate #SMT reduces to SMT. For our purposes, a *randomized algorithm* is an algorithm that uses internal coin-tossing. We always assume, whenever we use the term, that, for each possible input x to $\mathcal{A}$, the overall probability, over the internal coin tosses r, that $\mathcal{A}$ outputs a wrong answer is at most 1/4. (This error probability 1/4 can be reduced to any smaller $\alpha > 0$, by taking the median across $O(\log \alpha^{-1})$ independent runs of $\mathcal{A}$.)

We say that a randomized algorithm $\mathcal{A}$ *approximates* a real-valued functional problem $\mathcal{C} \colon \Sigma^* \to \mathbb{R}$ *with an additive error* if $\mathcal{A}$ takes as input an $x \in \Sigma^*$ and a rational number $\gamma > 0$ and produces an output $\mathcal{A}(x, \gamma)$ such that

$$\mathsf{Pr}\big[|\mathcal{A}(x, \gamma) - \mathcal{C}(x)| \leqslant \gamma\, \mathcal{U}(x)\big] \geqslant 3/4,$$

where $\mathcal{U} \colon \Sigma^* \to \mathbb{R}$ is some specific and efficiently computable upper bound on the absolute value of $\mathcal{C}(x)$, i. e., $|\mathcal{C}(x)| \leqslant \mathcal{U}(x)$, that comes with the problem $\mathcal{C}$. Similarly, $\mathcal{A}$ *approximates* a (possibly real-valued) functional problem $\mathcal{C} \colon \Sigma^* \to \mathbb{R}$ *with a multiplicative error* if $\mathcal{A}$ takes as input an $x \in \Sigma^*$ and a rational number $\varepsilon > 0$ and produces an output $\mathcal{A}(x, \varepsilon)$ such that

$$\mathsf{Pr}\big[(1 + \varepsilon)^{-1} \mathcal{C}(x) \leqslant \mathcal{A}(x, \varepsilon) \leqslant (1 + \varepsilon)\, \mathcal{C}(x)\big] \geqslant 3/4.$$

The computation time is usually considered relative to $|x|+\gamma^{-1}$ or $|x|+\varepsilon^{-1}$, respectively (note the inverse of the admissible error). Polynomial-time algorithms that achieve approximations with a multiplicative error are also known as fully polynomial-time randomized approximation schemes (FPRAS) [21].

Algorithms can be equipped with *oracles* solving auxiliary problems, with the intuition that an external solver (say, for SAT) is invoked. In theoretical considerations, the definition of the running time of such an algorithm takes into account the preparation of *queries* to the oracle (just as any other computation), but not the answer to a query—it is returned within a single time step. Oracles may be defined as solving some specific problems (say, SAT) as well as any problems from a class (say, from **NP**). The following result is well-known.

Proposition 1 (generic approximate counting [21,31]). *Let $\mathcal{C}\colon \Sigma^* \to \mathbb{N}$ be any member of $\#\mathbf{P}$. There exists a polynomial-time randomized algorithm $\mathcal{A}$ which, using an $\mathbf{NP}$-oracle, approximates $\mathcal{C}$ with a multiplicative error.*

In the rest of this section, we present our results on the complexity of model counting problems, $\#\mathrm{SMT}[\mathcal{T}]$, for measured theories. For these problems, we develop randomized polynomial-time approximation algorithms equipped with oracles, in the flavour of Proposition 1. We describe the proof ideas in Section 3. We relate the theories to value estimation problems of probabilistic programs later in Section 4; our implementation substitutes an appropriate solver for the oracle.

Integer arithmetic. By IA we denote the *bounded* version of integer arithmetic: each free variable x_i of a formula $\varphi(x_1, \ldots, x_k)$ comes with a bounded domain $D_i = [a_i, b_i] \subseteq \mathbb{Z}$, where $a_i, b_i \in \mathbb{Z}$. We use the counting measure $|\cdot|\colon A \subseteq \mathbb{Z} \mapsto |A|$, so the model count $\mathsf{mc}(\varphi)$ of a formula φ is the number of its models. In the formulas, we allow existential (but not universal) quantifiers at the top level. The model counting problem for IA is $\#\mathbf{P}$-complete.

Theorem 1. *The model counting problem for IA can be approximated with a multiplicative error by a polynomial-time randomized algorithm that has oracle access to satisfiability of formulas in IA.*

Linear real arithmetic. By RA we denote the *bounded* version of linear real arithmetic, with possible existential (but not universal) quantifiers at the top level. Each free variable x_i of a formula $\varphi(x_1, \ldots, x_k)$ comes with a bounded domain $D_i = [a_i, b_i] \subseteq \mathbb{R}$, where $a_i, b_i \in \mathbb{R}$. The associated measure is the standard Lebesgue measure, and the model count $\mathsf{mc}(\varphi)$ of a formula φ is the volume of its set of models. (Since we consider linear constraints, any quantifier-free formula defines a finite union of polytopes. It is an easy geometric fact that its projection on a set of variables will again be a finite union of bounded polytopes. Thus, existential quantification involves only finite unions.)

In the model counting problem for RA, the a priori upper bound $\mathcal{U}$ on the solution is $\prod_{i=1}^{k}(b_i - a_i)$; additive approximation of the problem is $\#\mathbf{P}$-complete.

Theorem 2. *The model counting problem for RA can be approximated with an additive error by a polynomial-time randomized algorithm that has oracle access to satisfiability of formulas in $\mathsf{IA} + \mathsf{RA}$ (the combined theory of IA and RA).*

3 Proof Techniques

3.1 Approximate discrete model counting

We now explain the idea behind Theorem 1. Let $\varphi(x)$ be an input formula in IA and $x = (x_1, \ldots, x_k)$ free variables of φ. Suppose M is a big enough integer such that all models of φ have components not exceeding M, i. e., $[\![\varphi]\!] \subseteq [0, M]^k$.

Our approach to approximating $\mathsf{mc}(\varphi) = |[\![\varphi]\!]|$ follows the construction in Jerrum et al. [21], which builds upon the following observation. Suppose our goal is to find a value v such that $v \leqslant \mathsf{mc}(\varphi) \leqslant 2v$, and we have an oracle $\mathcal{E}$, for "Estimate", answering questions of the form $\mathsf{mc}(\varphi) \geqslant^{?} N$. Then it is sufficient to make such queries to $\mathcal{E}$ for $N = N_m = 2^m$, $m = 0, \ldots, k \log(M + 1)$, and the overall algorithm design is reduced to implementing such an oracle efficiently.

It turns out that this can be done with the help of *hashing*. Suppose that a hash function h, taken at random from some family $\mathcal{H}$, maps elements of $[0, M]^k$ to $\{0, 1\}^m$. If the family $\mathcal{H}$ is chosen appropriately, then each potential model w is mapped by h to, say, 0^m with probability 2^{-m}; moreover, one should expect that any set $S \subseteq [0, M]^k$ of size d has roughly $2^{-m} \cdot d$ elements in $h^{-1}(0^m) = \{w \in [0, M]^k \mid h(w) = 0^m\}$. In other words, if $|S| \geqslant 2^m$, then $S \cap h^{-1}(0^m)$ is non-empty with high probability, and if $|S| \ll 2^m$, then $S \cap h^{-1}(0^m)$ is empty with high probability. Distinguishing between empty and non-empty sets is, in its turn, a satisfiability question and, as such, can be entrusted to the IA solver. As a result, we reduced the approximation of the model count of φ to a series of satisfiability questions in IA.

Our algorithm posts these questions as SMT queries of the form

$$\varphi(x) \wedge t(x, x') \wedge (h'(x') = 0^m), \tag{1}$$

where x and x' are tuples of integer variables, each component of x' is either 0 or 1, the formula $t(x, x')$ says that x' is binary encoding of x, and the IA formula $h'(x') = 0^m$ encodes the computation of the hash function h on input x'.

Algorithm 1 is the basis of our implementation. It returns a value v that satisfies the inequalities $(1 + \varepsilon)^{-1}\mathsf{mc}(\varphi) \leqslant v \leqslant (1 + \varepsilon)\,\mathsf{mc}(\varphi)$ with probability at least $1 - \alpha$. Algorithm 1 uses a set of parameters to discharge "small" values by enumeration in the SMT solver (parameters a, p) and to optimally query the solver for larger instances (parameters B, q, r). The procedure $\mathcal{E}$ given as Algorithm 2 asks the SMT solver for IA to produce a satisfying assignments (for a positive integer parameter a) to formulas of the form (1) by calling the procedure SMT. The constant B in the algorithm is defined by $B = ((\sqrt{a+1}+1)/(\sqrt{a+1}-1))^2$. To achieve the required precision with the desired probability, the algorithm makes q copies of the formula, constructing a formula with k' Boolean variables, and does a majority vote over r calls to the procedure $\mathcal{E}$, where

$$q = \left\lceil \frac{1 + \log B}{2 \log(1 + \varepsilon)} \right\rceil \quad \text{and} \quad r = \left\lceil 8 \ln \left(\frac{k' - \lfloor \log B - 2 \log(\sqrt{B} - 1) \rfloor - 3}{\alpha} \right) \right\rceil.$$

Algorithm 1: Approximate model counting for IA

Input: formula $\varphi(x)$ in IA
Output: value $v \in \mathbb{N}$
Parameters: $\varepsilon \in (0,1)$, `/* approximation factor */`
$\alpha \in (0,1)$, `/* error probability */`
$a \in \mathbb{N}$ `/* enumeration limit for SMT solver */`
Pick B, q, p, r based on parameters (see text);
$\psi(x, x') = \varphi(x) \wedge t(x, x')$;
$\psi_q(\mathbf{x}, \mathbf{x}') = \psi(x^1, x'^1) \wedge \psi(x^2, x'^2) \wedge \ldots \wedge \psi(x^q, x'^q)$;
if $(e := \mathrm{SMT}(\psi_q, p+1)) \le p$ **then return** $\sqrt[q]{e}$;
k' := number of bits in $\mathbf{x}'$;
for $m = 1, \ldots, k'+1$ **do**
  $c := 0$; `/* majority vote counter */`
  for $j = 1, \ldots, r$ **do**
    if $\mathcal{E}(\psi_q, k', m, a)$ **then** $c := c + 1$;
  if $c \le r/2$ **then break**;
return $\sqrt[q]{\frac{2^{(m+3/2)}\sqrt{B}}{(\sqrt{B}-1)^2}}$

Algorithm 2: Satisfiability "oracle" $\mathcal{E}$

Input: formula $\psi_q(\mathbf{x}, \mathbf{x}')$ in IA; $k', m, a \in \mathbb{N}$
Output: *true* or *false*
$h' :=$ PICK-HASH(k', m);
$\psi_{h'}(\mathbf{x}, \mathbf{x}') = \psi_q(\mathbf{x}, \mathbf{x}') \wedge (h'(\mathbf{x}') = 0^m)$;
return $(\mathrm{SMT}(\psi_{h'}, a) \ge a)$ `/* check if` $\psi_{h'}$ `has at least` a `assignments */`

For formulas φ with up to $p^{1/q}$ models, where $p = 2\lceil 4/(\sqrt{B}-1)^2 \rceil$, Algorithm 1 returns precisely the model count $\mathsf{mc}(\varphi)$ computed by the procedure SMT which repeatedly calls the solver, counting the number of models up to $p+1$.

The family of hash functions $\mathcal{H}$ used by PICK-HASH needs to satisfy the condition of *pairwise independence*: for any two distinct vectors $x_1, x_2 \in [0, M]^k$ and any two strings $w_1, w_2 \in \{0,1\}^m$, the probability that a random function h from $\mathcal{H}$ satisfies $h(x_1) = w_1$ and $h(x_2) = w_2$ is equal to $1/2^{2m}$. There are several constructions for pairwise independent hash functions; we employ a commonly used family, that of random XOR constraints [34,2,17,6]. Given k' and m, the family contains (in binary encoding) all functions $h' = (h'_1, \ldots, h'_m) \colon \{0,1\}^{k'} \to \{0,1\}^m$ with $h'_i(x_1 \ldots, x_{k'}) = a_{i,0} + \sum_{j=1}^{k'} a_{i,j} x_j$, where $a_{i,j} \in \{0,1\}$ for all i and $+$ is the XOR operator (addition in GF(2)). By randomly choosing the coefficients $a_{i,j}$ we get a random hash function from this family. The size of each query is thus bounded by $O(\frac{1}{\varepsilon^2}|\varphi|^2)$, where $|\varphi|$ is the size of the original formula φ, and there will be at most $O(\frac{1}{\varepsilon}|\varphi|)$ queries in total.

3.2 Approximate continuous model counting

In this subsection we explain the idea behind Theorem 2. Let φ be a formula in RA; using appropriate scaling, we can assume without loss of generality that all the variables share the same domain. Suppose $[\![\varphi]\!] \subseteq [0, M]^k$ and fix some γ, with the prospect of finding a value v that is at most $\varepsilon = \gamma M^k$ away from $\mathsf{mc}(\varphi)$ (we take M^k as the value of the upper bound $\mathcal{U}$ in the definition of additive approximation). We show below how to reduce this task of continuous model counting to additive approximation of a model counting problem for a formula with a discrete set of possible models, which, in turn, will be reduced to that of multiplicative approximation.

We first show how to reduce our continuous problem to a discrete one. Divide the cube $[0, M]^k$ into s^k small cubes with side δ each, $\delta = M/s$. For each $y = (y_1, \ldots, y_k) \in \{0, 1, \ldots, s-1\}^k$, set $\psi'(y) = 1$ if at least one point of the cube $C(y) = \{y_j\delta \leqslant x_j \leqslant (y_j + 1)\,\delta, 1 \leqslant j \leqslant k\}$ satisfies φ; that is, if $C(y) \cap [\![\varphi]\!] \neq \emptyset$.

Imagine that we have a formula ψ such that $\psi(y) = \psi'(y)$ for all $y \in \{0, 1, \ldots, s-1\}^k$, and let ψ be written in a theory with a uniform measure that assigns "weight" M/s to each point $y_j \in \{0, 1, \ldots, s-1\}$; one can think of these weights as coefficients in numerical integration. From the technique of Dyer and Frieze [11, Theorem 2] it follows that for a quantifier-free φ and an appropriate value of s the inequality $|\mathsf{mc}(\psi) - \mathsf{mc}(\varphi)| \leqslant \varepsilon/2$ holds.

Indeed, Dyer and Frieze prove a statement of this form in the context of volume computation of a polyhedron, defined by a system of inequalities $Ax \leqslant b$. However, they actually show a stronger statement: given a collection of m hyperplanes in $\mathbb{R}^k$ and a set $[0, M]^k$, an appropriate setting of s will ensure that out of s^k cubes with side $\delta = M/s$ only a small number J will be *cut*, i. e., intersected by some hyperplane. More precisely, if $s = \lceil mk^2M^k/(\varepsilon/2) \rceil$, then this number J will satisfy the inequality $\delta^k \cdot J \leqslant \varepsilon/2$. Thus, the total volume of cut cubes is at most $\varepsilon/2$, and so, in our terms, we have $|\mathsf{mc}(\psi) - \mathsf{mc}(\varphi)| \leqslant \varepsilon/2$ as desired.

However, in our case the formula φ need not be quantifier-free and may contain existential quantifiers at the top level. If $\varphi(x) = \exists u.\Phi(x, u)$ where Φ is quantifier-free, then the constraints that can "cut" the x-cubes are not necessarily inequalities from Φ. These constraints can rather arise from projections of constraints on variables x and, what makes the problem more difficult, their combinations. However, we are able to prove the following statement:

Lemma 1. *The number $\bar{J}$ of points $y \in \{0, 1, \ldots, s-1\}^k$ for which cubes $C(y)$ are cut satisfies $\bar{\delta}^k \cdot \bar{J} \leqslant \varepsilon/2$ if $\bar{\delta} = M/\bar{s}$, where $\bar{s} = \lceil 2^{\bar{m}+2k}k^2M^k/(\varepsilon/2) \rceil = \lceil 2^{\bar{m}+2k}k^2/(\gamma/2) \rceil$ and $\bar{m}$ is the number of atomic predicates in Φ.*

A consequence of the lemma is that the choice of $\bar{s}$ ensures that the formula $\psi(y) = \exists\, x.(\varphi(x) \wedge x \in C(y))$ written in the combined theory $\mathsf{IA} + \mathsf{RA}$ satisfies the inequality $|\mathsf{mc}(\psi) - \mathsf{mc}(\varphi)| \leqslant \varepsilon/2$. Here we associate the domain of each free variable $y_j \in \{0, 1, \ldots, \bar{s} - 1\}$ with the uniform measure $\mu_j(v) = M/\bar{s}$. Note that the value of $\bar{s}$ chosen in Lemma 1 will still keep the number of steps of our algorithm polynomial in the size of the input, because the number of bits needed to store the integer index along each axis is $\lceil \log(\bar{s} + 1) \rceil$ and not $\bar{s}$ itself.

As a result, it remains to approximate $\mathsf{mc}(\psi)$ with additive error of at most $\varepsilon' = \varepsilon/2 = \gamma M^k/2$, which can be done by invoking the procedure from Theorem 1 that delivers approximation with multiplicative error $\beta = \varepsilon'/M^k = \gamma/2$.

4 Value Estimation for Probabilistic Programs

4.1 The Value Estimation Problem

We now describe an application of approximate model counting to probabilistic programs. Probabilistic programming models extend "usual" nondeterministic programs with the ability to sample values from a distribution and condition the behavior of the program based on observations [18]. Intuitively, probabilistic programs extend an imperative programming language like C with two constructs: a nondeterministic assignment to a variable from a range of values, and a probabilistic assignment that sets a variable to a random value sampled from a distribution. Designed as a modeling framework, probabilistic programs are typically treated as descriptions of probability distributions and not meant to be implemented as usual programs.

We consider a core loop-free imperative language extended with probabilistic statements and with nondeterministic choice:

$$s \ ::= \ x := e \mid x \sim \mathsf{Uniform}(a, b) \mid \mathsf{assume}(p) \mid s; s \mid s [\!] s \mid \mathsf{accept} \mid \mathsf{reject}.$$

The statement $x := e$ models (usual) assignment, $x \sim \mathsf{Uniform}(a, b)$ takes a value uniformly at random from $[a, b]$ and assigns it to x, $\mathsf{assume}(\varphi)$ models observations used to condition a distribution, $[\!]$ models nondeterministic choice between statements, and accept and reject are special accepting and rejecting statements.

Under each given assignment to the probabilistic variables, a program accepts (rejects) if there is an execution path that is compatible with the observations and goes from the initial state to an accepting (resp., rejecting) statement. Consider all possible outcomes of the probabilistic assignments in a program $\mathcal{P}$. Restrict attention to those that result in $\mathcal{P}$ reaching (nondeterministically) at least one of accept or reject statements—such elementary outcomes form the set Term (for "termination"); only these scenarios are compatible with the observations. Similarly, some of these outcomes may result in the program reaching (again, nondeterministically) an accept statement—they form the set Accept. Note that the sets Term and Accept are events in a probability space; define $\mathsf{val}(\mathcal{P})$, the *value* of $\mathcal{P}$, as the conditional probability $\mathsf{Pr}[\mathsf{Accept} \mid \mathsf{Term}]$, which is equal to the ratio $\frac{\mathsf{Pr}[\mathsf{Accept}]}{\mathsf{Pr}[\mathsf{Term}]}$ as $\mathsf{Accept} \subseteq \mathsf{Term}$. We assume that programs are well-formed in that $\mathsf{Pr}[\mathsf{Term}]$ is bounded away from 0.

Now consider a probabilistic program $\mathcal{P}$ *over a measured theory* $\mathcal{T}$, i. e., where the expressions and predicates come from $\mathcal{T}$. Associate a separate variable r with each probabilistic assignment in $\mathcal{P}$ and denote the corresponding distribution by $\mathsf{dist}(r)$. Let R be the set of all such variables r.

Proposition 2. *There exists a polynomial-time algorithm that, given a program $\mathcal{P}$ over $\mathcal{T}$, constructs logical formulas $\varphi_{\mathsf{acc}}(R)$ and $\varphi_{\mathsf{term}}(R)$ over $\mathcal{T}$ such that $\mathsf{Accept} = [\![\varphi_{\mathsf{acc}}]\!]$ and $\mathsf{Term} = [\![\varphi_{\mathsf{term}}]\!]$, where each free variable $r \in R$ is interpreted over its domain with measure $\mathsf{dist}(r)$. Thus, $\mathsf{val}(\mathcal{P}) = \mathsf{mc}(\varphi_{\mathsf{acc}})/\mathsf{mc}(\varphi_{\mathsf{term}})$.*

Proposition 2 reduces the *value estimation* problem—i. e., the problem of estimating $\mathsf{val}(\mathcal{P})$—to model counting. For the theories of integer and linear real arithmetic, we get a $\#\mathbf{P}$ upper bound on the complexity of value estimation. On the other hand, value estimation is $\#\mathbf{P}$-hard, as it easily encodes #SAT. Finally, since the model counting problem can be approximated using a polynomial-time randomized algorithm with a satisfiability oracle, we also get an algorithm for approximate value estimation.

Proposition 3 (complexity of value estimation).

1. *The value estimation problem for loop-free probabilistic programs (over IA and RA) is $\#\mathbf{P}$-complete. The problem is $\#\mathbf{P}$-hard already for programs with Boolean variables only.*
2. *The value estimation problem for loop-free probabilistic programs over IA can be approximated with a multiplicative error by a polynomial-time randomized algorithm that has oracle access to satisfiability of formulas in IA.*
3. *The value estimation problem for loop-free probabilistic programs over RA can be approximated with an additive error by a polynomial-time randomized algorithm that has oracle access to satisfiability of formulas in $\mathsf{IA} + \mathsf{RA}$.*

4.2 Evaluation

We have implemented the algorithm from Subsection 3.1 in C++ on top of the SMT solver Z3 [10]. The SMT solver is used unmodified, with default settings.

Examples. For the evaluation we consider five examples. The first two are probabilistic programs that use nondeterminism. The remaining examples are Bayesian networks encoded in our language.

The Monty Hall problem [29] is a classic problem from probability theory. Imagine a television game show with two characters: the player and the host. The player is facing three doors, numbered 1, 2, and 3; behind one of them is a car, and behind the other two are goats. The player initially picks one of the doors, say door i, but does not open it. The host, who knows the position of the car, then opens another door, say door j with $j \neq i$, and shows a goat behind it. The player then gets to open one of the remaining doors. There are two available strategies: *stay* with the original choice, door i, or *switch* to the remaining alternative, door $k \notin \{i, j\}$. The Monty Hall problem asks, which strategy is better? It is widely known that, in the standard probabilistic setting of the problem, the switching strategy is the better one: it has payoff 2/3, i. e., it chooses the door with the car with probability 2/3; the staying strategy has payoff of only 1/3. We model this problem as a probabilistic program, where the host's actions are modelled using nondeterminism (for details see the extended version [8]).

Example	ε	α	a	k'	m_{acc}	m_{term}	time(s) for φ_{acc}	time(s) for φ_{term}
Monty Hall (1)	0.2	0.01	1	24	13	20	3.37	4.11
Three prisoners (2)	0.2	0.01	1	36	0	20	0.04	19.84
Alarm (3)	0.5	0.1	20	56	36	49	196.54	132.53
Grass model (4)	0.5	0.1	20	48	34	35	85.71	89.37
Sensitivity est. (5)	0.5	0.1	20	66	56	57	295.09	241.55

Table 1. Input and algorithm parameters, and running time. The input parameter ε is the multiplicative approximation factor, α is the desired error probability and a is the number of satisfying assignments the SMT solver checks for; k' is the resulting number of bits and m_{acc} and m_{term} are the maximal hash sizes for φ_{acc} and φ_{term}.

The three prisoners problem. Our second example is a problem that appeared in Martin Gardner's "Mathematical Games" column in the Scientific American in 1959. There, one of three prisoners (1, 2, and 3), who are sentenced to death, is randomly pardoned. The guard gives prisoner 1 the following information: If 2 is pardoned, he gives 1 the name of 3. If 3 is pardoned, he gives him the name of 2. If 1 is pardoned, he flips a coin to decide whether to name 2 or 3. Provided that the guard tells prisoner 1 that prisoner 2 is to be executed, determine what is prisoner 1's chance to be pardoned?

Pearl's burglar alarm and grass model. These two examples are classical Bayesian networks from the literature. Pearl's burglar alarm example is as given in [18, Figure 15]; the grass model is taken from [22, Figure 1].

Kidney disease eGFR sensitivity estimation. The last example is a probabilistic model of a medical diagnostics system with noisy inputs. We considered the program given in [18, Figure 11] using a simplified model of the input distributions. In our setting, we draw the value of the logarithm of the patient's creatinine level uniformly from the set $\{-0.16, -0.09, -0.08, 0, 0.08, 0.09, 0.16, 0.17\}$ (thus approximating the original lognormal distribution), regardless of the patient's gender, and the patient's age from the interval $[30, 80]$. The patient's gender and ethnicity are distributed in the same way as described in [28].

Results. For each program $\mathcal{P}$, we used our tool to estimate the model count of the formulas φ_{acc} and φ_{term}; the value $\mathsf{val}(\mathcal{P})$ of the program is approximated by $v_{\mathsf{acc}}/v_{\mathsf{term}}$, where v_{acc} and v_{term} are the approximate model counts computed by our tool. Table 1 shows input and algorithm parameters for the considered examples, as well as running time (in seconds) for computing v_{acc} and v_{term}. The approximation factor ε, the bound α on the error probability, and the enumeration limit a for the SMT solver are provided by the user. For examples (1) and (2), we choose ε to be 0.2, while for the remaining examples we take 0.5. The chosen value of ε has an impact on the number of copies q of the formula that we construct, an thus on the number k' of Boolean variables in the formula given to the solver. Furthermore, the more satisfying assignments a formula has, the larger dimension m of the hash function is reached during the run; m_{acc} and m_{term} are the maximal values of m reached on the runs on φ_{acc} and φ_{term}.

While our technique can solve these small instances in reasonable time, there remains much room for improvement. Although SAT solvers can scale to large instances, it is well known that even a small number of XOR constraints can quickly exceed the capabilities of state-of-the-art solvers [32]. Since for each m we add m parity constraints to the formula, we run into the SAT bottleneck: computing an approximation of $\mathsf{mc}(\varphi_{\mathsf{acc}})$ for example (4) with $\varepsilon = 0.3$ results in running time of several hours. (At the same time, exact counting by enumerating satisfying assignments is not a feasible alternative either: for the formula φ_{acc} in example (4), which has more than 400 000 of them, performing this task naively with Z3 also took several hours.) The efficiency of our approach can benefit from better handling of XOR constraints in the SMT solver. For example, SAT solvers such as CryptoMiniSat which deal with XOR constraints efficiently can scale to over 1K variables [6,5,17]. This, however, requires incorporating such a SAT solver within Z3.

The scalability needs improvement also in the continuous case, where our discretization procedure introduces a large number of discrete variables. For instance, a more realistic model of example (5) would be one in which the logarithm of the creatinine level is modeled as a continuous random variable. This would result, after discretization, in formulas with hundreds of Boolean variables, which appears to be beyond the limit of Z3's XOR reasoning.

5 Concluding Remarks

Static reasoning questions for probabilistic programs [18,28,19], as well as quantitative and probabilistic analysis of software [3,15,14,24], have received a lot of recent attention. There are two predominant approaches to these questions. The first one is to perform Monte Carlo sampling of the program [28,3,24,4,27]. To improve performance, such methods use sophisticated heuristics and variance reduction techniques, such as stratified sampling in [28,3]. The second approach is based on reduction to model counting [14,15,26,25], either using off-the-shelf #SMT solvers or developing #SMT procedures on top of existing tools. Another recent approach is based on data flow analysis [9]. Our work introduces a new dimension of approximation to this area: we reduce program analysis to #SMT, but carry out a randomized approximation procedure for the count.

By known connections between counting and uniform generation [21,2], our techniques can be adapted to generate (approximately) uniform random samples from the set of models of a formula in IA or RA. Uniform generation from Boolean formulas using hashing techniques was recently implemented and evaluated in the context of constrained random testing of hardware [6,5]. We extend this technique to the SMT setting, which was left as a future direction in [6] (previously known methods for counting integral points of polytopes [1,15] do not generalize to the nonlinear theory IA).

Further directions.

Scalability. An extension of the presented techniques may be desirable to cope with larger instances of #SMT. As argued in subsection 4.2, incorporating XOR-aware reasoning into Z3 can be an important step in this direction.

Theories. Similar techniques apply to theories other than IA and RA. For example, our algorithm can be extended to the combined theory of string constraints and integer arithmetic. While SMT solvers can handle this theory, it would be nontrivial to design a model counting procedure using the previously known approach based on generating functions [25].

Distributions. Although the syntax of our probabilistic programs supports only Uniform, it is easy to simulate other distributions: Bernoulli, uniform with non-constant boundaries, (approximation of) normal. This, however, will not scale well, so future work may incorporate non-uniform distributions as a basic primitive. (An important special case covers weighted model counting in SAT, for which a novel extension of the hashing approach was recently proposed [5].)

Applications. A natural application of the uniform generation technique in the SMT setting would be a procedure that generates program behaviors uniformly at random from the space of possible behaviors. (For the model we studied, program behaviors are trees: the branching comes from nondeterministic choice, and the random variables are sampled from their respective distributions.)

References

1. A. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. In *FOCS 93*. ACM, 1993.
2. M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Inf. Comput.*, 163(2):510–526, 2000.
3. M. Borges, A. Filieri, M. d'Amorim, C. Pasareanu, and W. Visser. Compositional solution space quantification for probabilistic software analysis. In *PLDI*, page 15. ACM, 2014.
4. A. Chaganty, A. Nori, and S. Rajamani. Efficiently sampling probabilistic programs via program analysis. In *AISTATS*, volume 31 of *JMLR Proceedings*, pages 153–160. JMLR.org, 2013.
5. S. Chakraborty, D. Fremont, K. Meel, S. Seshia, and M. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *AAAI'14*, pages 1722–1730, 2014.
6. S. Chakraborty, K. Meel, and M. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *CAV*, volume 8044 of *LNCS*, pages 608–623, 2013.
7. S. Chakraborty, K. Meel, and M. Vardi. A scalable approximate model counter. In *CP: Constraint Programming*, volume 8124 of *LNCS*, pages 200–216, 2013.
8. D. Chistikov, R. Dimitrova, and R. Majumdar. Approximate counting in SMT and value estimation for probabilistic programs. *CoRR*, abs/1411.0659, 2014.
9. G. Claret, S. K. Rajamani, A. V. Nori, A. D. Gordon, and J. Borgström. Bayesian inference using data flow analysis. In *ESEC/FSE'13*, pages 92–102, 2013.
10. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.
11. M. Dyer and A. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.
12. M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

13. S. Ermon, C. Gomes, A. Sabharwal, and B. Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *ICML (2)*, pages 334–342, 2013.
14. A. Filieri, C. Pasareanu, and W. Visser. Reliability analysis in symbolic Pathfinder. In *ICSE*, pages 622–631, 2013.
15. M. Fredrikson and S. Jha. Satisfiability modulo counting: A new approach for analyzing privacy properties. In *LICS*. ACM, 2014.
16. C. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *IJCAI*, pages 2293–2299, 2007.
17. C. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009.
18. A. Gordon, T. Henzinger, A. Nori, S. Rajamani, and S. Samuel. Probabilistic programming. In *FOSE 14*, pages 167–181. ACM, 2014.
19. C.-K. Hur, A. Nori, S. Rajamani, and S. Samuel. Slicing probabilistic programs. In *PLDI*, page 16. ACM, 2014.
20. M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pages 482–520, 1996.
21. M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *TCS*, 43:169–188, 1986.
22. O. Kiselyov and C.-C. Shan. Monolingual probabilistic programming using generalized coroutines. In *UAI*, pages 285–292. AUAI Press, 2009.
23. LattE tool. `https://www.math.ucdavis.edu/~latte`.
24. K. S. Luckow, C. S. Pasareanu, M. B. Dwyer, A. Filieri, and W. Visser. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In *ASE'14*, pages 575–586, 2014.
25. L. Luu, S. Shinde, P. Saxena, and B. Demsky. A model counter for constraints over unbounded strings. In *PLDI*, page 57. ACM, 2014.
26. F. Ma, S. Liu, and J. Zhang. Volume computation for boolean combination of linear arithmetic constraints. In *CADE-22*, LNCS 5663, pages 453–468. Springer, 2009.
27. A. Sampson, P. Panchekha, T. Mytkowicz, K. McKinley, D. Grossman, and L. Ceze. Expressing and verifying probabilistic assertions. In *PLDI*, page 14. ACM, 2014.
28. S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *PLDI*, pages 447–458. ACM, 2013.
29. S. Selvin. A problem in probability. *American Statistician*, 29(1):67, 1975.
30. M. Sipser. A complexity-theoretic approach to randomness. In *STOC*, pages 330–335. ACM, 1983.
31. L. Stockmeyer. On approximation algorithms for $\#P$. *SIAM J. of Computing*, 14:849–861, 1985.
32. A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.
33. L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 9:189–201, 1979.
34. L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.
35. M. Zhou, F. He, X. Song, S. He, G. Chen, and M. Gu. Estimating the volume of solution space for satisfiability modulo linear real arithmetic. *Theory of Computing Systems*, pages 1–25, 2014.