An Approximate Skolem Function Counter*

Arijit Shaw[1,2], **Brendan Juba**[3], **Kuldeep S. Meel**[4]

[1] Chennai Mathematical Institute, India
[2] IAI, TCG-CREST, Kolkata, India
[3] Washington University in St. Louis, USA
[4] University of Toronto, Canada

Abstract

One approach to probabilistic inference involves counting the number of models of a given Boolean formula. Here, we are interested in inferences involving higher-order objects, i.e., functions. We study the following task: Given a Boolean specification between a set of inputs and outputs, count the number of functions of inputs such that the specification is met. Such functions are called Skolem functions.

We are motivated by the recent development of scalable approaches to Boolean function synthesis. This stands in relation to our problem analogously to the relationship between Boolean satisfiability and the model counting problem. Yet, counting Skolem functions poses considerable new challenges. From the complexity-theoretic standpoint, counting Skolem functions is not only $\#P$-hard; it is quite unlikely to have an FPRAS (Fully Polynomial Randomized Approximation Scheme) as the problem of synthesizing a Skolem function remains challenging, even given access to an NP oracle.

The primary contribution of this work is the first algorithm, SkolemFC, that computes an estimate of the number of Skolem functions. SkolemFC relies on technical connections between counting functions and propositional model counting: our algorithm makes a linear number of calls to an approximate model counter and computes an estimate of the number of Skolem functions with theoretical guarantees. Moreover, we show that Skolem function count can be approximated through a polynomial number of calls to a SAT oracle. Our prototype displays impressive scalability, handling benchmarks comparably to state-of-the-art Skolem function synthesis engines, even though counting all such functions ostensibly poses a greater challenge than synthesizing a single function.

1 Introduction

Probabilistic inference problems arise throughout AI and are tackled algorithmically by casting them as problems such as model counting (Gomes, Sabharwal, and Selman 2021; Chakraborty, Meel, and Vardi 2021). In this work, we are interested in approaching inference questions for higher-order objects, specifically *Skolem functions*: that is, we wish to compute the number of possible Skolem functions for a given specification $F(X, Y)$. Counting Skolem functions is the natural analog of #SAT for Skolem functions, yet to our knowledge, it has not been previously studied.

More precisely, recall that given two sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ of variables and a Boolean formula $F(X, Y)$ over $X \cup Y$, the problem of Boolean functional synthesis is to compute a vector $\Psi = \langle \psi_1, \ldots, \psi_m \rangle$ of Boolean functions ψ_i, often called Skolem functions, such that $\exists Y F(X, Y) \equiv F(X, \Psi(X))$. Informally, given a specification between inputs and outputs, the task is to synthesize a function vector Ψ that maps each assignment of the inputs to an assignment of the outputs so that the combined assignment meets the specification (whenever such an assignment exists). Skolem synthesis is a fundamental problem in formal methods and has been investigated by theoreticians and practitioners alike over the past few decades. The past few years have witnessed the development of techniques that showcase the promise of scalability in their ability to handle challenging specifications (Jiang, Lin, and Hung 2009; Tabajara and Vardi 2017; Rabe et al. 2018; Akshay et al. 2019; Golia et al. 2021).

The scalability of today's Skolem synthesis engines is reminiscent of the scalability of SAT solvers in the early 2000s. Motivated by the scalability of SAT solvers (Froleyks et al. 2021), researchers sought algorithmic frameworks for problems beyond satisfiability, such as MaxSAT (Ansótegui, Bonet, and Levy 2013; Li and Manya 2021), model counting (Gomes, Sabharwal, and Selman 2021; Chakraborty, Meel, and Vardi 2021), sampling (Chakraborty, Meel, and Vardi 2014), and the like. The development of scalable techniques for these problems also helped usher in new applications, even though the initial investigation had not envisioned many of them. In a similar vein, motivated in part by this development of scalable techniques for functional synthesis, we investigate the Skolem counting problem. We observe in Section 1.2 that algorithms for such tasks also have potential applications in security and the engineering of specifications. Being a natural problem, we will see that our study also naturally leads to deep technical connections between counting functions and counting propositional models and the development of new techniques, which is of independent interest.

Counting Skolem functions indeed raises new technical challenges. The existing techniques developed in the context of propositional model counting either construct (implicitly or explicitly) a representation of the space of all models (Thur-

*The full version of the paper: https://arxiv.org/abs/2312.12026 ley 2006; Dudek, Phan, and Vardi 2020; Sharma et al. 2019) or at least enumerate a small number of models (Chakraborty, Meel, and Vardi 2013, 2016; Soos and Meel 2019; Yang and Meel 2023), which in practice amounts to a few tens to hundreds of models of formulas constrained by random XORs. Such approaches are unlikely to work in the context of Skolem function counting, where even finding one Skolem function is hard, and there are no techniques that enable the enumeration of Skolem functions.

1.1 Technical Contribution

The primary contribution of this work is the development of a novel algorithmic framework, called SkolemFC, that approximates the Skolem function count with a theoretical guarantee, using only linearly many calls to an approximate model counter and almost-uniform sampler. First, we observe that Skolem function counting can be reduced to an exponential number of model counting calls, serving as a baseline Skolem function counter. The core technical idea of SkolemFC is to reduce the problem of approximate Skolem function counting to only linearly many (in $m = |Y|$) calls to propositional model counters. Of particular note is the observation that SkolemFC can provide an approximation to the number of Skolem functions without enumerating even one Skolem function. As approximate model counting and almost-uniform sampling can be done by logarithmically many calls to SAT oracle, we show that Skolem function counting can also be reduced to polynomially many calls to a SAT oracle.

To measure the impact of the algorithm, we implement SkolemFC and demonstrate its potential over a set of benchmarks arising from prior studies in the context of Skolem function synthesis. Out of 609 instances, SkolemFC could solve 375 instances, while a baseline solver could solve only eight instances. For context, the state-of-the-art Skolem function synthesis tool Manthan2 (Golia et al. 2021) effectively tackled 509 instances from these benchmarks, while its precursor, Manthan (Golia, Roy, and Meel 2020), managed only 356 instances with a timeout of 7200 seconds.

1.2 Applications

This problem arises in several potential application areas.

Specification engineering. The first and primary motivation stems from the observation that specification synthesis (Albarghouthi, Dillig, and Gurfinkel 2016; Prabhu et al. 2021) (i.e., the process of constructing $F(X, Y)$) and function synthesis form part of the iterative process wherein one iteratively modifies specifications based on the functions that are constructed by the underlying engine. In this context, one helpful measure is to determine the number of possible semantically different functions that satisfy the specification, as often a large number of possible Skolem functions indicates the vagueness of specifications and highlights the need for strengthening the specification. Note that the use of the count is qualitative here, and hence an approximate order of magnitude (log count) suffices.

Diversity at the specification level. In system security and reliability, a classic technique is to generate and use a diverse variety of functionally equivalent implementations of components (Baudry and Monperrus 2015). Although classically, this is achieved by transformations of the code that preserve the function computed, we may also be interested in producing a variety of functions that satisfy a common specification. Unlike transformations on the code, it is not immediately clear whether a specification even admits a diverse collection of functions – indeed, the function may be uniquely defined. Thus, counting the number of such functions is necessary to assess the potential value of taking this approach, and again a rough order of magnitude estimate suffices. Approximate counting of the functions may also be a useful primitive for realizing such an approach.

Uninterpreted functions in SMT. A major challenge in the design of counting techniques for SMT (Chistikov, Dimitrova, and Majumdar 2015; Chakraborty et al. 2016) lies in handling uninterpreted functions (Kroening and Strichman 2016). Since Skolem functions capture a restricted but large enough class of uninterpreted functions (namely, the case where a given uninterpreted function is allowed to depend on all X variables), progress in Skolem function counting is needed if we hope to make progress on the general problem of counting of uninterpreted functions in SMT.

Evaluation of a random Skolem function. Although synthesis of Skolem functions remains challenging in general, we note that approximate counting enables a kind of incremental evaluation by using the standard techniques for reducing sampling to counting. More concretely, given a query input, we can estimate the number of functions that produce each output: this is trivial if the range is small (e.g., Boolean), and otherwise, we can introduce random XOR constraints to incrementally specify the output. Once an output is specified for the query point, we may retain these constraints when estimating the number of consistent functions for subsequent queries, thereby obtaining an approximately uniform function conditioned on the answers to the previous queries.

1.3 Organization

The rest of the paper is organized as follows: we discuss related work in Section 2 and present notation and preliminaries in Section 3. We then present the primary technical contribution of our work in Section 4. We present the empirical analysis of the prototype implementation of SkolemFC in Section 5. We finally conclude in Section 6.

2 Related Work

Despite the lack of prior studies focused on the specific problem of counting Skolem functions, significant progress has been made in synthesizing these functions. Numerous lines of research have emerged in the field of Skolem function synthesis. The first, incremental determinization, iteratively pinpoints variables with distinctive Skolem functions, making decisions on any remaining variables by adding provisional clauses that render them deterministic (Rabe 2019; Rabe and Seshia 2016; Rabe et al. 2018). The second line of research involves obtaining Skolem functions by eliminating quantifiers using functional composition and reducing the size of composite functions through the application of Craig interpolation (Jiang, Lin, and Hung 2009; Jiang 2009). The third, CEGAR-style approaches, commence with an initial set of approximate Skolem functions, and proceed to a phase of counter-example guided refinement to improve upon these candidate functions (John et al. 2015; Akshay et al. 2017, 2018). Work on the representation of specification $F(X,Y)$ has led to efficient synthesis using ROBDD representation and functional composition (Balabanov and Jiang 2011), with extensions to factored specifications (Tabajara and Vardi 2017; Chakraborty et al. 2018). Notable advancements include the new negation normal form, SynNNF, amenable to functional synthesis (Akshay et al. 2019). Finally, a data-driven method has arisen (Golia, Roy, and Meel 2020; Golia et al. 2021), relying on constrained sampling to generate satisfying assignments for a formula F.

A related problem in the descriptive complexity of functions definable by counting the Skolem functions for *fixed* formulas have been shown to characterize $\#AC_0$ (Haak and Vollmer 2019). By contrast, we are interested in the problem where the *formula* is the input. Our algorithm also bears similarity to the FPRAS proposed for the descriptive complexity class $\#\Sigma_1$ (Durand et al. 2021), which is obtained by an FPRAS for counting the number of *functions* satisfying a DNF over atomic formulas specifying that the functions must/must not take specific values at specific points. Nevertheless, our problem is fundamentally different in that it is easy to find functions satisfying such DNFs, whereas synthesis of Skolem functions is unlikely to be possible in polynomial time.

The specifications for the functions are often expressed in terms of quantified Boolean formulas (QBFs). Another quantitative question on QBFs is AllQBF (Becker et al. 2012), finding all assignments of the free variables of a given QBF such that the formula evaluates to true. CountingQBF (Shukla et al. 2022) poses a similar query within a quantitative aspect. However, their relevance to counting functions isn't clear.

3 Notation and Preliminaries

We use lowercase letters (with subscripts) to denote propositional variables and uppercase letters to denote a subset of variables. The formula $\exists Y F(X,Y)$ is existentially quantified in Y , where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. By n and m we denote the number of X and Y variables in the formula. Therefore, $n = |X|, m = |Y|$. For simplicity, we write a formula $F(X,Y)$ as F if the X, Y is clear from the context. A *model* is an assignment (true or false) to all the variables in F, such that F evaluates to true. Let $\mathsf{Sol}(F)_{\downarrow S}$ denote the set of models of formula F projected on $S \subseteq X \cup Y$. If $S = X \cup Y$, we write the set as $\mathsf{Sol}(F)$. Let σ be a partial assignment for the variables X of F. Then $\mathsf{Sol}(F \wedge (X = \sigma))$ denotes the models of F where $X = \sigma$.

NP Oracles and SAT oracles. Given a Boolean formula F, an NP oracle determines the satisfiability of the formula. A SAT solver is a practical tool solving the problem of satisfiability. Following definition of Delannoy and Meel (2022), a SAT oracle takes in a formula F and returns a satisfying assignment σ if F is satisfiable and $\perp$ otherwise. The SAT oracle model captures the behavior of the modern SAT solvers.

Propositional Model Counting. Given a formula F and a projection set S, the problem of model counting is to compute $|\mathsf{Sol}(F)_{\downarrow S}|$. An *approximate model counter* takes in a formula F, projection set S, tolerance parameter ε, and confidence parameter δ, and returns c such that $\Pr\left[\frac{|\mathsf{Sol}(F)_{\downarrow S}|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathsf{Sol}(F)_{\downarrow S}|\right] \geq 1-\delta$. It is known that $\log(n)$ calls to a SAT oracle are necessary (Chakraborty et al. 2023) and sufficient (Chakraborty, Meel, and Vardi 2016) to achieve (ε, δ) guarantees for approximately counting the models of a formula with n variables.

Propositional Sampling. Given a Boolean formula F and a projection set S, a *sampler* is a probabilistic algorithm that generates a random element in $\mathsf{Sol}(F)_{\downarrow S}$. An *almost uniform sampler* G takes a tolerance parameter ε along with F and S, and guarantees $\forall y \in \mathsf{Sol}(F)_{\downarrow S}$, $\frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)_{\downarrow S}|} \leq \Pr[G(F,S,\varepsilon) = y] \leq \frac{(1+\varepsilon)}{|\mathsf{Sol}(F)_{\downarrow S}|}$. Delannoy and Meel (2022) showed, $\log(n)$ many calls to a SAT oracle suffices to generate almost uniform samples from a formula with n variables.

Skolem Functions. Given a Boolean specification $F(X,Y)$ between set of inputs $X = \{x_1, \ldots, x_n\}$ and vector of outputs $Y = \langle y_1, \ldots, y_m \rangle$, a function vector $\Psi(X) = \langle \psi_1(X), \psi_2(X), \ldots, \psi_m(X) \rangle$ is a Skolem function vector if $y_i \leftrightarrow \psi_i(X)$ and $\exists Y F(X,Y) \equiv F(X,\Psi)$. We refer to Ψ as the Skolem function vector and Ψ_i as the Skolem function for y_i. We'll use the notation $\mathsf{Skolem}(F,Y)$ to denote the set of possible $\Psi(X)$ satisfying the condition $\exists Y F(X,Y) \equiv F(X, \Psi(X))$. Two Skolem function vectors Ψ_1 and Ψ_2 are different, if there exists an assignment $\sigma \in \mathsf{Sol}(F)_{\downarrow X}$ for which $\Psi_1(\sigma) \neq \Psi_2(\sigma)$.

For a specification $\exists Y F(X,Y)$, the number of Skolem functions itself can be as large as $2^{n \cdot 2^m}$, and the values of n and m are quite large in many practical cases. Beyond being a theoretical possibility, the count of Skolem functions is often quite big, and such values are sometimes difficult to manipulate and store as 64-bit values. Therefore, we are interested in the logarithm of the counts, and define the problem of approximate Skolem function counting as following:

Problem Statement. Given a Boolean specification $F(X,Y)$, tolerance parameter ε, confidence parameter δ, let $\ell = \log(|\mathsf{Skolem}(F,Y)|)$, the task of approximate Skolem function counting is to give an estimate Est, such that $\Pr[(1-\varepsilon)\ell \leq \mathsf{Est} \leq (1+\varepsilon)\ell] \geq 1-\delta$.

In practical scenarios, the input *specification* is often given as a quantified Boolean formula (QBF). The output of the synthesis problem is a function, which is expressed as a Boolean circuit. In our setting, even if two functions have different circuits, if they have identical input-output behavior, we consider them to be the same function. For example, let $f_1(x) = x$ and $f_2 = \neg(\neg x)$. We'll consider f_1 and f_2 to be the same function.

Illustrative Example. Let's examine a formula defined on three sets of variables X, Y_0, Y_1, where each set contains five Boolean variables, interpreted as five-bit integers:

Algorithm 1: Stopping Rule (ε, δ)

```
1: $t \leftarrow 0, x \leftarrow 0, s \leftarrow 4\ln(2/\delta)(1+\varepsilon)/\varepsilon^2$
2: while $x < s$ do
3:     $t \leftarrow t+1$
4:     Generate Random Variable $Z_t$
5:     $x \leftarrow x + Z_t$
6: Est $\leftarrow s/t$
7: return Est
```

$\exists Y_0 Y_1 F(X, Y_0 Y_1)$, where F represents the constraint for factorization, $X = Y_0 \times Y_1, Y_0 \leq Y_1, Y_0 \neq 1$. The number of Skolem functions of F gives the number of distinct ways to implement a factorization function for 5-bit input numbers. There exist multiple X's for which there are multiple factorizations: A Skolem function S_1 may factorize 12 as 4×3 and a function S_2 may factorize 12 as 2×6.

Stopping Rule Algorithm. Let $Z_1, Z_2, \ldots$ denote independently and identically distributed (i.i.d.) random variables taking values in the interval $[0, 1]$ and with mean μ. Intuitively, Z_t is the outcome of experiment t. Then the Stopping Rule algorithm (Algorithm 1) approximates μ as stated by Theorem 3.1 (Dagum et al. 1995; Dagum and Luby 1997).

Theorem 3.1 (Stopping Rule Theorem). For all $0 < \varepsilon \leq 2, \delta > 0$, if the Stopping Rule algorithm returns Est, then $\Pr[\mu(1-\varepsilon) \leq \mathsf{Est} \leq \mu(1+\varepsilon)] > (1-\delta)$.

FPRAS. A Fully Polynomial Randomized Approximation Scheme (FPRAS) is a randomized algorithm that, for any fixed $\varepsilon > 0$ and any fixed probability $\delta > 0$, produces an answer that is within a factor of $(1+\varepsilon)$ of the correct answer, and does so with probability at least $(1-\delta)$, in polynomial time with respect to the size of the input, $1/\varepsilon$, and $\log(1/\delta)$.

4 Algorithm

In this section, we introduce the primary contribution of our paper: the SkolemFC algorithm. The algorithm takes in a formula $F(X, Y)$ and returns an estimate for $\log(|\mathsf{Skolem}(F, Y)|)$. We first outline the key technical ideas that inform the design of SkolemFC and then present the pseudocode for implementing this algorithm.

4.1 Core Ideas

Since finding even a single Skolem function is computationally expensive, our approach is to estimate the count of Skolem functions without enumerating even a small number of Skolem functions. The key idea is to observe that the number of Skolem functions can be expressed as a product of the model counts of formulas. A Skolem function $\Psi \in \mathsf{Skolem}(F, Y)$ is a function from 2^X to 2^Y. A useful quantity in the context of counting Skolem functions is to define, for every assignment $\sigma \in 2^X$, the set of elements in 2^Y that $\Psi(X)$ can belong to. We refer to this quantity as $\mathsf{range}(\sigma)$ and formally define it as follows:

Algorithm 2: SkolemFC$(F(X, Y), \varepsilon, \delta)$

```
1: $\varepsilon_f \leftarrow 0.6\varepsilon, \delta_f \leftarrow 0.4\delta, s \leftarrow 4\ln(2/\delta_f)(1+\varepsilon_f)/{\varepsilon_f}^2$
2: $\varepsilon_s \leftarrow 0.2\varepsilon, \delta_c = 0.4\delta/ms, \varepsilon_c \leftarrow 4\sqrt{2}-1$
3: $\varepsilon_g = 0.1\varepsilon, \delta_g = 0.1\delta$
4: $G(X, Y, Y') := F(X, Y) \wedge F(X, Y') \wedge (Y \neq Y')$
5: while $x < s$ do
6:     $\sigma \leftarrow$ AlmostUniformSample$(G, X, \varepsilon_s)$
7:     $c \leftarrow \log(\mathsf{ApproxCount}(F \wedge (X = \sigma), \varepsilon_c, \delta_c))/m$
8:     $x \leftarrow x + c$
9:     $t \leftarrow t + 1$
10: $g \leftarrow$ ApproxCount$(G, X, \varepsilon_g, \delta_g)$
11: Est $\leftarrow s/t \times m \times g$
12: if $(g \log(1+\varepsilon_c) > 0.1\mathsf{Est})$ then return $\perp$
13: return Est
```

Definition 4.1.

$$\mathsf{range}(\sigma) = \begin{cases} \mathsf{Sol}(F \wedge (X = \sigma))_{\downarrow Y} & |\mathsf{Sol}(F \wedge (X = \sigma))| > 0 \\ 1 & \text{otherwise} \end{cases}$$

Lemma 4.2. $|\mathsf{Skolem}(F, Y)| = \prod_{\sigma \in 2^X} |\mathsf{range}(\sigma)|$

Proof. First of all, we observe that

$$\forall \sigma \in 2^X, \forall \pi \in \mathsf{range}(\sigma), \exists \Psi \text{ s.t. } \Psi(\sigma) = \pi$$

which is easy to see for all $\sigma \in 2^X$ for which there exists $\pi \in 2^Y$ such that $F(\sigma, \pi) = 1$. As for $\sigma \in 2^X$ for which there is no π such that $F(\sigma, \pi) = 1$, Skolem functions that differ solely on inputs $\sigma \notin \mathsf{Sol}(F)_{\downarrow X}$ are regarded as identical. Consequently, such inputs have no impact on the count of distinct Skolem functions, resulting in $\mathsf{range}(\sigma) = 1$ for these cases. Recall, $\mathsf{Skolem}(F, Y)$ is the set of all functions from 2^X to 2^Y. It follows that $|\mathsf{Skolem}(F, Y)| = \prod_{\sigma \in 2^X} |\mathsf{range}(\sigma)|$. □

Lemma 4.2 allows us to develop a connection between Skolem function counting and propositional model counting. As stated in the problem statement, we focus on estimating $\log |\mathsf{Skolem}(F, Y)|$. To formalize our approach, we need to introduce the following notation:

Proposition 4.3.

$$\log |\mathsf{Skolem}(F, Y)| = \sum_{\sigma \in S_2} \log |\mathsf{Sol}(F \wedge (X = \sigma))|$$
$$\text{where, } S_2 := \{\sigma \in 2^X \mid |\mathsf{Sol}(F \wedge (X = \sigma))| \geq 2\}$$

Proof. From Lemma 4.2, we have $|\mathsf{Skolem}(F, Y)| = \prod_{\sigma \in 2^X} |\mathsf{range}(\sigma)|$. Taking logs on both sides, partitioning 2^X into S_2 and $2^X \backslash S_2$, and observing that $\log |\mathsf{range}(\sigma)| = 0$ for $\sigma \notin S_2$, we get the desired result. □

4.2 Algorithm Description

The pseudocode for SkolemFC is delineated in Algorithm 2. It accepts a formula $\exists Y F(X,Y)$, a tolerance level ε, and a confidence parameter δ. The algorithm SkolemFC then provides an approximation of $\log|\mathsf{Skolem}(F,Y)|$ following Proposition 4.3. To begin, SkolemFC almost-uniformly samples σ from S_2 at random in line 6, utilizing an almost-uniform sampler. Subsequently, SkolemFC approximates $|\mathsf{Sol}(F \wedge (X=\sigma))|$ through an approximate model counter at line 7. The estimate Est is computed by taking the product of the mean of c's and $|S_2|$. In order to sample $\sigma \in S_2$, SkolemFC constructs the formula G whose solutions, when projected to X represent all the assignments $\sigma \in S_2$ (line 4). Finally, SkolemFC returns the estimate Est as logarithm of the Skolem function count.

The main loop of SkolemFC (from lines 5 to 9) is based on the Stopping Rule algorithm presented in Section 2. The Stopping Rule algorithm is utilized to approximate the mean of a collection of i.i.d. random variables that fall within the range $[0,1]$. The method repeatedly adds the outcomes of these variables until the cumulative value reaches a set threshold s. This threshold value is influenced by the input parameters ε and δ. The result yielded by the algorithm is represented as s/t, where t denotes the number of random variables aggregated to achieve the threshold s. In the context of SkolemFC, this random variable is defined as $\log(\mathsf{ApproxCount}(F \wedge (X=\sigma),\varepsilon_c,\delta_c))/m$. Line 12 asserts that the error introduced by the approximate model counting oracle is within some specific bound.

Oracle Access. We assume access to approximate model counters and almost-uniform samplers as oracles. The notation $\mathsf{ApproxCount}(F,P,\varepsilon,\delta)$ represents an invocation of the approximate model counting oracle on Boolean formula F with a projection set P, tolerance parameter ε, and confidence parameter δ. $\mathsf{AlmostUniformSample}(F,S,\varepsilon)$ denotes an invocation of the almost uniform sampler on a formula F, with projection set S and tolerance parameter ε. The particular choice of values of ε_s, ε_c, δ_c, ε_g, δ_g used in the counting and sampling oracle aids the theoretical guarantees.

4.3 Illustrative Example

We will now examine the specification of factorization as outlined in Section 3, and investigate how SkolemFC estimates the count of Skolem functions meeting that specification.

1. In line 4, SkolemFC constructs G such that $|\mathsf{Sol}(G)_{\downarrow X}| = 7$, $\mathsf{Sol}(G)_{\downarrow X} = \{12,16,18,20,24,28,30\}$.
2. In line 6, it samples σ from $\mathsf{Sol}(G)_{\downarrow X}$. Let's consider $\sigma = 30$. Then $\mathsf{Sol}(F \wedge (X=\sigma))_{\downarrow Y_0,Y_1} = \{(2,15),(3,10),(5,6)\}$. Therefore, $c = \log(3)$ in line 7.
3. Suppose in the next iteration it samples $\sigma = 16$. Then $\mathsf{Sol}(F \wedge (X=\sigma)) = \{(2,8),(4,4)\}$. Therefore, $c = \log(2)$ in line 7.
4. Now suppose that the termination condition of line 5 is reached. At this point, the estimate Est returned from line 11 will be $\approx \frac{(\log(3)+\log(2))}{2} \times 7 \approx 6$.
5. Finally, SkolemFC will return the value Est= 6.

Note that the approach is in stark contrast to the state-of-the-art counting techniques in the context of propositional models, which either construct a compact representation of the entire solution space or rely on the enumeration of a small number of models.

4.4 Analysis of SkolemFC

Let $F(X,Y)$ be a propositional CNF formula over variables X and Y. In the section we'll show that SkolemFC works as an approximate counter for the number of Skolem functions. We create a formula $G(X,Y,Y') = F(X,Y) \wedge F(X,Y') \wedge (Y \neq Y')$ from $F(X,Y)$, where Y' is a fresh set of variables and $m = |Y'|$. We show that, if we pick a solution from $G(X,Y,Y') = F(X,Y) \wedge F(X,Y') \wedge (Y \neq Y')$, then the assignment to X in that solution to G will have at least two solutions in $F(X,Y)$.

Lemma 4.4.

$$\mathsf{Sol}(G)_{\downarrow X} = \left\{\sigma \mid \sigma \in 2^X \wedge |\mathsf{Sol}(F \wedge (X=\sigma))| \geq 2\right\}$$

Proof. We can write the statement alternatively as,

$$\sigma \in \mathsf{Sol}(G,X) \iff |\mathsf{Sol}(F \wedge (X=\sigma))| \geq 2$$

($\implies$) For every element $\sigma \in \mathsf{Sol}(G)$, we write σ as $\langle \sigma_{\downarrow X}, \sigma_{\downarrow Y}, \sigma_{\downarrow Y'} \rangle$. Now according to the definition of G, both $\langle \sigma_{\downarrow X}, \sigma_{\downarrow Y} \rangle$ and $\langle \sigma_{\downarrow X}, \sigma_{\downarrow Y'} \rangle$ satisfy F. Moreover, $\sigma_{\downarrow Y}$ and $\sigma_{\downarrow Y'}$ are not equal. Therefore, $|\mathsf{Sol}(F \wedge (X=\sigma))| \geq 2$.

($\impliedby$) If $|\mathsf{Sol}(F \wedge (X=\sigma))| \geq 2$, then $F(X,Y)$ has solutions of the form $\langle \sigma, \gamma_1 \rangle$ and $\langle \sigma, \gamma_2 \rangle$, where $\gamma_1 \neq \gamma_2$. Now $\langle \sigma \gamma_1 \gamma_2 \rangle$ satisfies G. □

Theorem 4.5. SkolemFC takes in input $F(X,Y)$, $\varepsilon > 0$, and $\delta \in (0,1]$, and returns Est such that

$$\Pr\left[(1-\varepsilon)\ell \leq \mathsf{Est} \leq (1+\varepsilon)\ell\right] \geq 1-\delta$$

where, $\ell = \log(|\mathsf{Skolem}(F,Y)|)$. Furthermore, it makes $\tilde{O}\left(\frac{m}{\varepsilon^2}\ln\frac{2}{\delta}\right)$ many calls to a SAT oracle, where $\tilde{O}$ hides poly-log factors in parameters m,n,ε,δ.

We defer the proof to the full version due to space constraints.

5 Experiments

We conducted a thorough evaluation of the performance and accuracy of results of the SkolemFC algorithm by implementing a functional prototype[1] in C++. The following experimental setup was used to evaluate the performance and quality of results of the SkolemFC algorithm[2].

Baseline. A possible approach to count Skolem functions, following Lemma 4.2, is given in Algorithm 3. The $\mathsf{Count}(F)$ oracle denotes an invocation of exact model counter. We implemented that to compare with SkolemFC. In the implementation, we relied on the latest version of Ganak (Sharma et al. 2019) to get the necessary exact model counts. We use a modified version of the SAT solver CryptoMiniSat (Soos, Nohl, and Castelluccia 2009) as AllSAT solver to find all solutions of a given formula, projected on X variables. We call this implementation Baseline in the following part of the paper.

[1]Source code: https://github.com/meelgroup/skolemfc/

[2]All benchmarks and experimental data are available at https://doi.org/10.5281/zenodo.10404174

Algorithm 3: Baseline$(F(X,Y))$

```
1: Est ← 0
2: $G(X,Y,Y') := F(X,Y) \wedge F(X,Y') \wedge (Y \neq Y')$
3: SolG ← AllSAT$(G, X)$
4: for each $\sigma \in$ SolG do
5:     $c \leftarrow \log(\mathsf{Count}(F \wedge (X = \sigma)))$
6:     Est ← Est + $c$
7: return Est
```

Environment. All experiments were carried out on a cluster of nodes consisting of AMD EPYC 7713 CPUs running with 2x64 real cores. All tools were run in a single-threaded mode on a single core with a timeout of 10 hrs, i.e., 36000 seconds. A memory limit was set to 32 GB per core.

Parameters for Oracles and Implementation. In the implementation, we utilized different state-of-the-art tools as counting and sampling oracles, including UniSamp (Delannoy and Meel 2022) as an almost uniform sampling oracle, and the latest version of ApproxMC (Yang and Meel 2023) as an approximate counting oracle. SkolemFC was tested with $\varepsilon = 0.8$ and $\delta = 0.4$. That gave the following values to error and tolerance parameters for model counting and sampling oracles. The almost uniform sampling oracle UniSamp is run with $\varepsilon_s = 0.16$. The approximate model counting oracle ApproxMC in line 7 was run with $\varepsilon_c = 4\sqrt{2} - 1$ and $\delta_c = \frac{0.32}{m \cdot s}$, where s comes from the algorithm, based on input (ε, δ) and m is number of output variables in the specification. We carefully select error and tolerance values $\varepsilon_s, \varepsilon_c, \delta_c$ for counting and sampling oracles to ensure the validity of final bounds for SkolemFC while also aiming for optimal performance of the counter based on these choices. The relationship between these values and the validity of bound of SkolemFC is illustrated in the proof of Theorem 4.5.

In our experiments, we sought to evaluate the run-time performance and approximation accuracy of SkolemFC. Specifically, the following questions guided our investigation:

RQ1. How does SkolemFC scale in terms of solving instances and the time taken in our benchmark set?

RQ2. What is the precision of the SkolemFC approximation, and does it outperform its theoretical accuracy guarantees in practical scenarios?

Benchmarks. To evaluate the performance of SkolemFC, we chose two sets of benchmarks.

1. *Efficiency benchmarks.* 609 instances from recent works on Boolean function synthesis (Golia, Roy, and Meel 2020; Akshay et al. 2017), which includes different sources: the Prenex-2QBF track of QBF Evaluation 2017 and 2018, disjunctive (Akshay et al. 2017), arithmetic (Tabajara and Vardi 2017) and factorization (Akshay et al. 2017).
2. *Correctness Benchmarks.* The benchmarks described in the paragraph above are too hard for the baseline algorithm to solve. As Section 5.1 reveals, the number of instances solved by the baseline is just eight out of the 609 instances. Therefore, to check the correctness of SkolemFC (RQ2), we used a set of 158 benchmarks from SyGuS instances (Golia, Roy, and Meel 2021). These benchmarks have very few input variables ($m \leq 8$), and takes seconds for SkolemFC to solve.

Algorithm	# Instances solved
Baseline	8
SkolemFC	375

Table 1: Instances solved (out of 609).

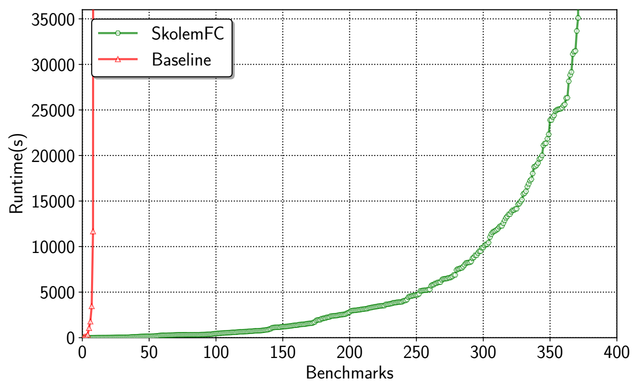

Figure 1: Runtime performance of SkolemFC and Baseline.

Summary of Results. SkolemFC achieves a huge improvement over Baseline by resolving 375 instances in a benchmark set consisting of 609, while Baseline only solved 8. The accuracy of the approximate count is also noteworthy, with an average error of a count by SkolemFC of only 21%.

5.1 Performance of SkolemFC

We evaluate the performance of SkolemFC based on two metrics: the number of instances solved and the time taken to solve the instances.

Instances Solved. In Table 1, we compare the number of benchmarks that can be solved by Baseline and SkolemFC. First, it is evident that the Baseline only solved 8 out of the 609 benchmarks in the test suite, indicating its lack of scalability for practical use cases. Conversely, SkolemFC solved 375 instances, demonstrating a substantial improvement compared to Baseline.

Solving Time Comparison. A performance evaluation of Baseline and SkolemFC is depicted in Figure 1, which is a cactus plot comparing the solving time. The x-axis represents the number of instances, while the y-axis shows the time taken. A point (i, j) in the plot represents that a solver solved j benchmarks out of the 609 benchmarks in the test suite in less than or equal to j seconds. The curves for Baseline and SkolemFC indicate that for a few instances, Baseline was able to give a quick answer, while in the long run, SkolemFC could solve many more instances given any fixed timeout.

Counter Call Comparison. We analyze the algorithms' complexity in terms of counter calls, comparing Baseline and

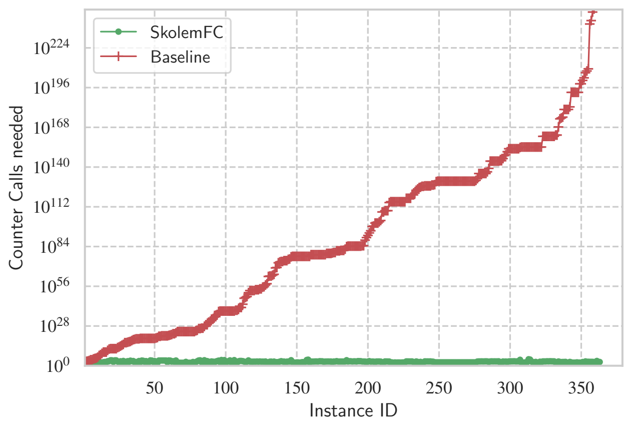

Figure 2: Counter calls needed by SkolemFC and Baseline to solve the benchmarks.

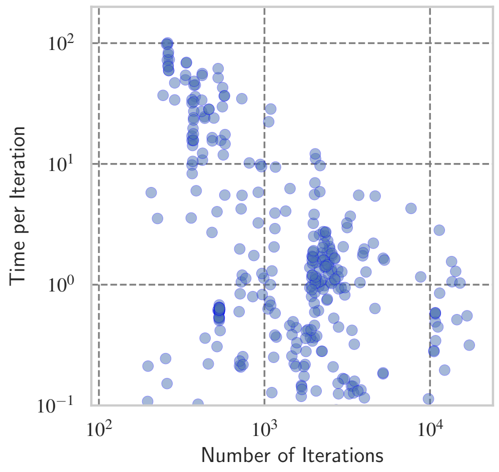

Figure 3: Relation between number of iterations needed by SkolemFC and average time taken in each iteration.

SkolemFC across benchmarks in Figure 2. The x axis represents benchmarks, and the y axis shows required counter calls, sorted by the increasing order of calls needed by Baseline. A red or green point (i, j) signifies that Baseline or SkolemFC, respectively, requires j counting oracle calls for the i^{th} instance. Baseline requires up to a staggering 10^{230} counter calls for some instances, emphasizing the need for a scalable algorithm like SkolemFC, which incurs significantly fewer counter calls.

We analyze the scalability of SkolemFC by examining the correlation between average time per iteration and total number of iterations, depicted in Figure 3. The point (i, j) means that if SkolemFC needs i counter calls, the average time per call is j seconds. The figure showcases diverse scenarios: some with fewer iterations and longer durations per call, others with high counts and minimal time per call.

5.2 Quality of Approximation

In the experiments, 158 accuracy benchmarks were measured using Baseline, enabling comparison between Baseline and SkolemFC results, shown in Figure 4. The counts' close alignment and error reduction below theoretical guarantees were observed. We quantify the SkolemFC performance with error $e = \frac{|b-s|}{b}$, where b is the count from Baseline and s from SkolemFC. Analysis of all 158 cases found the average e to be 0.21, geometric mean 0.19, and maximum 0.496, contrasting sharply with a theoretical guarantee of 0.8. This signifies SkolemFC substantially outperforms its theoretical bounds. Our findings underline SkolemFC's accuracy and potential as a dependable tool for various applications.

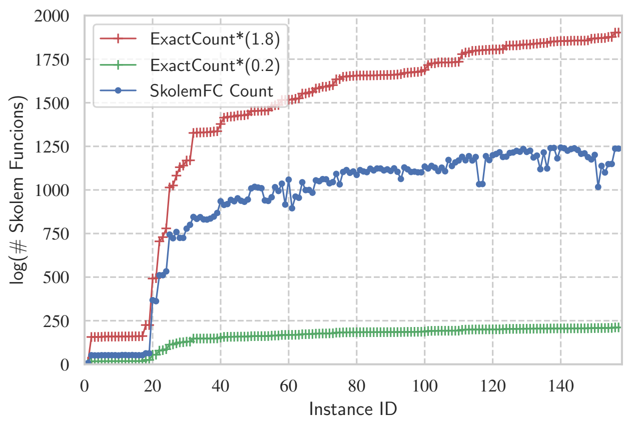

Figure 4: SkolemFC's estimate vs. the theoretical bounds.

6 Conclusion

In conclusion, this paper presents the first scalable approximate Skolem function counter, SkolemFC, which has been successfully tested on practical benchmarks and showed impressive performance. Our proposed method employs probabilistic techniques to provide theoretical guarantees for its results. The implementation leverages the progress made in the last two decades in the fields of constrained counting and sampling, and the practical results exceeded the theoretical guarantees. These findings open several directions for further investigation. One such area of potential extension is the application of the algorithm to other types of functions, such as counting uninterpreted functions in SMT with a more general syntax. This extension would enable the algorithm to handle a broader range of applications and provide even more accurate results. In summary, this research contributes significantly to the field of Skolem function counting and provides a foundation for further studies.

Acknowledgements

We are thankful to Tim van Bremen and Priyanka Golia for providing detailed feedback on the early drafts of the paper and grateful to the anonymous reviewers for their constructive comments to improve this paper. We thank Martina Seidl, Andreas Plank and Sibylle Möhle for pointing out an error in the first implementation of the code. This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 Grant [MOE-T2EP20121-0011], Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114], and NSF awards IIS-1908287, IIS-1939677, and IIS-1942336. Part of the work was done during Arijit Shaw's internship at the National University of Singapore. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg).

References

Akshay, S.; Arora, J.; Chakraborty, S.; Krishna, S.; Raghunathan, D.; and Shah, S. 2019. Knowledge Compilation for Boolean Functional Synthesis. In *Proc. of FMCAD*.

Akshay, S.; Chakraborty, S.; Goel, S.; Kulal, S.; and Shah, S. 2018. What's hard about Boolean Functional Synthesis? In *Proc. of CAV*.

Akshay, S.; Chakraborty, S.; John, A. K.; and Shah, S. 2017. Towards parallel Boolean functional synthesis. In *Proc. of TACAS*.

Albarghouthi, A.; Dillig, I.; and Gurfinkel, A. 2016. Maximal specification synthesis. *ACM SIGPLAN Notices*.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. SAT-based MaxSAT algorithms. *Artificial Intelligence*.

Balabanov, V.; and Jiang, J.-H. R. 2011. Resolution proofs and Skolem functions in QBF evaluation and applications. In *Proc. of CAV*.

Baudry, B.; and Monperrus, M. 2015. The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Computing Surveys (CSUR)*.

Becker, B.; Ehlers, R.; Lewis, M.; and Marin, P. 2012. ALLQBF solving by computational learning. In *Proc. of ATVA*.

Chakraborty, D.; Chakraborty, S.; Kumar, G.; and Meel, K. S. 2023. Approximate Model Counting: Is SAT Oracle More Powerful than NP Oracle? In *Proc. of ICALP*.

Chakraborty, S.; Fried, D.; Tabajara, L. M.; and Vardi, M. Y. 2018. Functional synthesis via input-output separation. In *Proc. of FMCAD*.

Chakraborty, S.; Meel, K.; Mistry, R.; and Vardi, M. 2016. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A scalable approximate model counter. In *Proc. of CP*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2014. Balancing Scalability and Uniformity in SAT-Witness Generator. In *Proc. of DAC*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proc. of IJCAI*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2021. Approximate model counting. In *Handbook of Satisfiability*.

Chistikov, D.; Dimitrova, R.; and Majumdar, R. 2015. Approximate counting in SMT and value estimation for probabilistic programs. In *Proc. of TACAS*.

Dagum, P.; Karp, R.; Luby, M.; and Ross, S. 1995. An optimal algorithm for Monte Carlo estimation. In *Proc. of FOCS*.

Dagum, P.; and Luby, M. 1997. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*.

Delannoy, R.; and Meel, K. S. 2022. On Almost-Uniform Generation of SAT Solutions: The power of 3-wise independent hashing. In *Proc. of LICS*.

Dudek, J. M.; Phan, V. H.; and Vardi, M. Y. 2020. ADDMC: Weighted model counting with algebraic decision diagrams. In *Proc. of AAAI*.

Durand, A.; Haak, A.; Kontinen, J.; and Vollmer, H. 2021. Descriptive complexity of #P functions: A new perspective. *Journal of Computer and System Sciences*.

Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M. 2021. SAT competition 2020. *Artificial Intelligence*.

Golia, P.; Roy, S.; and Meel, K. S. 2020. Manthan: A data-driven approach for Boolean function synthesis. In *Proc. of CAV*.

Golia, P.; Roy, S.; and Meel, K. S. 2021. Program Synthesis as Dependency Quantified Formula Modulo Theory. In *Proc. of IJCAI*.

Golia, P.; Slivovsky, F.; Roy, S.; and Meel, K. S. 2021. Engineering an efficient boolean functional synthesis engine. In *Proc. of ICCAD*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2021. Model counting. In *Handbook of satisfiability*.

Haak, A.; and Vollmer, H. 2019. A model-theoretic characterization of constant-depth arithmetic circuits. *Annals of Pure and Applied Logic*.

Jiang, J.-H. R. 2009. Quantifier elimination via functional composition. In *Proc. of CAV*.

Jiang, J. R.; Lin, H.; and Hung, W. 2009. Interpolating functions from large Boolean relations. In *Proc. of ICCAD*.

John, A. K.; Shah, S.; Chakraborty, S.; Trivedi, A.; and Akshay, S. 2015. Skolem functions for factored formulas. In *Proc. of FMCAD*.

Kroening, D.; and Strichman, O. 2016. *Decision procedures*. Springer.

Li, C. M.; and Manya, F. 2021. MaxSAT, hard and soft constraints. In *Handbook of satisfiability*.

Prabhu, S.; Fedyukovich, G.; Madhukar, K.; and D'Souza, D. 2021. Specification synthesis with constrained Horn clauses. In *Proc. of PLDI*.

Rabe, M. N. 2019. Incremental Determinization for Quantifier Elimination and Functional Synthesis. In *Proc. of CAV*.

Rabe, M. N.; and Seshia, S. A. 2016. Incremental Determinization. In *Proc. of SAT*.

Rabe, M. N.; Tentrup, L.; Rasmussen, C.; and Seshia, S. A. 2018. Understanding and extending incremental determinization for 2QBF. In *Proc. of CAV*.

Sharma, S.; Roy, S.; Soos, M.; and Meel, K. S. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In *Proc. of IJCAI*.

Shukla, A.; Möhle, S.; Kauers, M.; and Seidl, M. 2022. Outercount: A first-level solution-counter for quantified boolean formulas. In *Proc. of CICM*.

Soos, M.; and Meel, K. S. 2019. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proc. of AAAI*.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *Proc. of SAT*.

Tabajara, L. M.; and Vardi, M. Y. 2017. Factored Boolean functional synthesis. In *Proc. of FMCAD*.

Thurley, M. 2006. sharpSAT–counting models with advanced component caching and implicit BCP. In *Proc. of SAT*.

Yang, J.; and Meel, K. S. 2023. Rounding Meets Approximate Model Counting. In *Proc. of CAV*.