Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Early Career Track

Counting, Sampling, and Synthesis: The Quest for Scalability

Kuldeep S. Meel
National University of Singapore

Abstract

The current generation of symbolic reasoning techniques excel at the qualitative tasks (i.e., when the answer is Yes or No); such techniques sufficed for traditional systems whose design sought to achieve deterministic behavior. In contrast, modern computing systems crucially rely on the statistical methods to account for the uncertainty in the environment, and to reason about behavior of these systems, there is need to look beyond qualitative symbolic reasoning techniques. We will discuss our work focused on the development of the next generation of automated reasoning techniques that can perform higher-order tasks such as quantitative measurement, sampling of representative behavior, and automated synthesis of systems.

From a core technical perspective, our work builds on the *SAT revolution*, which refers to algorithmic advances in combinatorial solving techniques for the fundamental problem of satisfiability (SAT), i.e., whether it is possible to satisfy a given set of constraints. The SAT revolution offers the opportunity to develop scalable techniques for problems that lie *beyond SAT* from complexity perspective and, therefore, stand to benefit from the availability of powerful SAT engines. Our work seeks to enable a *Beyond SAT revolution* via design of scalable techniques for three fundamental problems that lie beyond SAT: *constrained counting, constrained sampling,* and *automated synthesis*.

1 Formal Definitions

In constrained counting, the task is to count the total weight, subject to a given weight function, of the set of solutions of input constraints. In constrained sampling, the task is to sample randomly, subject to a given weight function, from the set of solutions of input constraints. In the case of automated synthesis, given a set of constraints capturing the functional specification between inputs and outputs, the task is to construct a program (represented as a circuit) whose outputs meet the desired specification.

In order to state the problems formally, we introduce some necessary notations. Let $X = \{x_1, x_2, \ldots x_n\}$ be the set of proositional variables, F be a formula over X, and weight function $W : \{0,1\}^n \mapsto [0,1]$. To ease the notations, we overload W to denote the weight of a literal, assignment or formula, depending on the context. Let $\mathsf{sol}(F)$ represent the set of satisfying assignments of F and let $W(F) = \sum_{\sigma \in \mathsf{sol}(F)} W(\sigma)$. The weight function is often presented implicitly. One such formulation is where weights are assigned to literals and the weight of an assignment is the product of weights of its literals. For a variable x_i a weight function W , we use $W(x_i)$ and $W(\neg x_i)$ to denote the non-negative, real-valued weights of the positive and negative literals such that $W(x_i) + W(\neg x_i) = 1$; Thus, for an assignment σ we denote $W(\sigma) = \prod_{x_i \in \sigma} W(x_i) \prod_{x_i \notin \sigma} W(\neg xi)$

Definition 1 (Constrained Counting)**.** *Given F and W, compute $W(F)$*

Definition 2 (Constrained Sampling)**.** *Given F and W, sample $\sigma \in \mathsf{sol}(F)$ such that $\Pr[\sigma$ is output$] = \frac{W(\sigma)}{W(F)}$*

In order to define the problem of synthesis formally, we introduce another set Y of propositional variables and now consider $F(X, Y)$ to be defined over $X \cup Y$.

Definition 3 (Functional Synthesis)**.** *Given F, determine G such that $\forall X \left(\exists Y F(X, Y) \leftrightarrow F(X, G(X))\right)$*

Counting, sampling, and synthesis are fundamental problems with numerous applications including in probabilistic reasoning, machine learning, planning, statistical physics, computational biology, inexact computing, quantitative information flow analysis and constrained-random verification. We refer the reader to [Meel *et al.*, 2016; Golia *et al.*, 2020] and references therein for details on applications.

2 Constrained Counting

Constrained counting is known to be computationally hard [Valiant, 1979; Jerrum and Sinclair, 1996; Toda, 1989]. Significant effort has therefore been invested in theoretical work on studying the complexity of *approximate* variants of these problems. Key theoretical results pertaining to approximate counting were already known as early as the early 1980s [Stockmeyer, 1983; Jerrum *et al.*, 1986]. In a seminal paper [Stockmeyer, 1983], Stockmeyer introduced the idea of using pairwise independent hash functions [Carter and Wegman, 1977] to reduce the approximate counting problem to polynomially many propositional satisfiability (SAT)

calls. While theoretically elegant, translating the proposed technique to practice suffered from the requirement of a prohibitively large number of invocations of a SAT-solver to perform symbolic reasoning on input formulas. This meant that the technique could not be translated to practical tools that could be applied to industrial-scale problems.

We introduced a novel universal-hashing based paradigm, ApproxMC, wherein the problem of constrained counting over a n-dimensional space is reduced to solving a small number of queries to a combinatorial solver wherein each query is the underlying set of constraints augmented with randomly generated parity constraints [Chakraborty *et al.*, 2013b; Chakraborty *et al.*, 2014a; Chakraborty *et al.*, 2016a; Chakraborty *et al.*, 2016b; Soos and Meel, 2019; Agrawal *et al.*, 2020; Wang *et al.*, 2020; Soos *et al.*, 2020]. The core idea of ApproxMC is to use pairwise independent hash functions, expressed as randomly generated XOR constraints, to randomly partition the solution space of the original formula into "small" enough cells. The sizes of sufficiently many randomly chosen cells are then determined using calls to a SAT solver, and a scaled median of these sizes is used to estimate the desired total weight. Consequently, ApproxMC provides (ε, δ)-guarantees, i.e., the computed value is within $(1+\varepsilon)$-factor approximation with confidence at least $1-\delta$ for any user-provided values of ε and δ.

ApproxMC seeks to build on the success of modern SAT solvers, and therefore, has led to novel and unconventional algorithmic constructs. For example, prior work used independence among different SAT queries as a central component, which was preferred owing to the ease of theoretical analysis. In contrast, we have focused on the introduction of dependence among different SAT-solver queries to maximize the sharing of information among different calls by an underlying incremental SAT solver [Chakraborty *et al.*, 2016b]. The use of hash functions that allow dependent SAT-solver queries also allows a linear search to be replaced by a logarithmic search. The sizes of XOR constraints in the formulas that are fed to a modern SAT solver have a significant impact on the running time of the solver. By exploiting the connection between definability of formulas and variance of the distribution of solutions in a cell defined by 3-wise independent hash functions, we introduced an algorithmic technique, MIS, that reduced the size of required XOR constraints dramatically and improved the runtime performance of ApproxMC by orders of magnitude. We further exploited a beautiful connection between concentration measures of XOR-based hash functions and isoperimetric inequalities on Boolean hypercubes to resolve the long-standing open problem of the design of logarithmic size XORs that provide the desired theoretical guarantees and achieve practical scalability [Meel, 2020].

A fundamental contribution of our approach is its generalizability. In particular, if the set of constraints is expressed in Disjunctive Normal Form (DNF), ApproxMC directly yields a Fully Polynomial Randomized Approximation Scheme (FPRAS) for counting – the only known FPRAS for DNF that does not involve Monte Carlo steps [Chakraborty *et al.*, 2016b]. The resulting approach has also been demonstrated to be superior in performance to the classical Monte Carlo technique [Meel *et al.*, 2018]. Furthermore, we demonstrated ApproxMC can be lifted to count the minimal unsatisfiable subsets of a given formula, which is an important metric of diagnosis of faults in a system.

Data Streams and Counting We established a deep and natural connection between constrained counting and F_0 estimation [Pavan, 2021]. Given a stream of sets $a = a_1, a_2, \ldots, a_m$ wherein each set $a_i \subseteq \Omega$ the zeroth frequency moment, denoted as F_0, of a is the number of distinct elements appearing in a, i.e., $|\cup_i a_i|$. F_0 computation is a fundamental problem in data streaming with a rich history of prior work. In particular, we design a recipe to transform streaming algorithms for F_0 estimation to those for constrained counting. Such a transformation yields new (ε, δ)-approximate algorithms for constrained counting, which are different from currently known algorithms, and vice versa. In a folow-up work, we designed the first algorithm with $\mathrm{poly}(\log|\Omega|, m)$-space and time complexity for the case where each a_i allows *efficient* (i.e., $\mathrm{poly}(\log|\Omega|, m)$-time complexity) access to membership, sampling, and counting [Meel, 2021]. In particular, our result resolved the *open problem* of the existence of $\mathrm{poly}(\log|\Omega|, m)$-space and time complexity algorithms for Klee's measure problem in discrete settings.

2.1 Applications

Quantitative Verification of Neural Networks Given a system $\mathcal{N}$ and a property $\mathcal{P}$, the classical verification methodology focuses on the qualitative question of detecting whether there exists some input for which the output of $\mathcal{N}$ does not satisfying $\mathcal{P}$. While the qualitative verification methodology has been dominant paradigm in the context of traditional hardware and software systems wherein presence of bug is often viewed unacceptable, the advent of neural networks requires a rethinking given the statistical guarantees on the behavior of these systems. We proposed the notion of quantitative verification to capture how often a property $\mathcal{P}$ is satisfied by a network $\mathcal{N}$ [Baluta *et al.*, 2019; Baluta *et al.*, 2021]. We demonstrated how the framework can be applied in the context of properties such as robustness, susceptibility to trojan attacks, and fairness. As a concrete example, an analyst can analyze whether a statistically significant number of adversarial examples exist for a given input under user-defined distributions of perturbations. We have designed the first scalable framework, called NPAQ, with formal guarantees of correctness for binarized neural networks [Baluta *et al.*, 2019; Dudek *et al.*, 2020]. NPAQ reduces the problem of quantitative verification of binarized neural networks to projected model counting, therefore allowing us to rely on the scalability of ApproxMC.

Resilience of Critical Infrastructure The availability of critical facilities and utility services, such as power, telecommunications, water, gas, and transportation is crucial to an increasingly connected world. Natural disasters often result in disruption of underlying networks. We introduced a new approach, RelNet, to measure the resilience of underlying systems in events of natural disasters [Duenas-Osorio *et al.*, 2017; Paredes *et al.*, 2019]. The approach relies on the representation of the problem of computation of resilience as

constrained counting over the configurations of the edges in the underlying graph. The computational engine of RelNet was applied to ten power transmission networks powering small to medium sized cities in the states of Texas, Florida, California, Tennessee, Georgia, and South Carolina. Our approach was successfully applied to obtain *the first theoretically sound estimates of resilience* in these power transmission networks.

Quantitative Information Flow Analysis: Quantitative information flow analysis is a powerful code-analysis technique to detect leakage of secret program information to public program outputs. A specific fragment of the program (e.g., a function, or the whole program) is modeled as an information-theoretic channel from its input to its output. To compute the maximum amount of information that can leak from the program fragment of interest, a constrained counter is used to determine the number of distinct outputs of the fragment (e.g., return values of the function, or the outputs of the program). By counting the number of cases where information is leaked, an information-theoretic estimate of the quantified information flow can be obtained [Biondi *et al.*, 2018].

3 Constrained Sampling

We exploited the inter-reducibility of constrained counting and sampling and the properties of 3-wise independent hash functions to design the first scalable constrained sampler, UniGen, that can sample solutions of a given set of propositional a set of constraints [Chakraborty *et al.*, 2013a; Chakraborty *et al.*, 2014b; Chakraborty *et al.*, 2015; Soos *et al.*, 2020]. Unlike prior approaches pioneered that either required a linear number of calls to a constrained counter or computationally prohibitive steps of inverting n-wise independent hash functions, UniGen requires *a single call* to a constrained counter and employs hash functions that can be inverted reasonably efficiently in practice using state-of-the-art SAT solvers. The algorithm first invokes a constrained counter, viz. ApproxMC, to get an approximate count of the size of the solution space. Having this count enables automatically determining the parameters for using 3-wise independent hash functions, expressed as random XOR constraints. These constraints are used to partition the solution space in such a way that a randomly chosen cell is "small" enough in expectation, and the solutions in it can be enumerated by an existing SAT solver and sampled accordingly. While this yields a provably *almost uniform* sampling, the practical performance is, surprisingly, even better: in particular, for benchmarks where the ideal distribution can be generated using exhaustive enumeration, the distributions from UniGen are statistically indistinguishable from the ideal distributions [Chakraborty *et al.*, 2014b].

Distribution Testing-Driven Development Given the widespread applications of sampling, scalability remains a major challenge despite recent developments in the community, including UniGen. Consequently, for applications where samplers such as UniGen do not scale, the samplers based on heuristics and lacking theoretical analysis are the last resort. Recognizing the gap between the existence of heuristic-based samplers and lack of testing tools with rigorous guarantees, we initiated Barbarik project that combines distribution testing paradigm with modeling of the behavior of solvers to design a tester that can provably test whether a given sampler's output distribution is uniform or not [Chakraborty and Meel, 2019; Meel *et al.*, 2020; Pote, 2021]. We demonstrated that Barbarik is able to successfully *reject* samplers' whose distribution is far from uniform and would *accept* samplers such as UniGen. Next, we pursued a test-driven methodology in the design of CMSGen [Golia *et al.*, 2021c], a sampler based on state of the art SAT solver, whose theoretical analysis is beyond the reach of the currently existing techniques but for which Barbarik returns *accept*. Crucially, CMSGen is able to achieve significant scalability in comparison to existing samplers with theoretical guarantees, and thereby achieving the right trade-off between scalability and sample quality.

3.1 Applications

Testing of Highly Configurable Software Systems The widespread and diverse usage has led to the design of highly configurable software systems operating in diverse environments. A fundamental problem is the generation of test configurations that maximize coverage while respecting constraints that rule out invalid configurations. Building on the scalability of our samplers, we proposed an adaptive weighted sampling approach, called baital, that achieves significantly higher t-wise coverage [Baranov *et al.*, 2020].

Hardware-Design Verification: Every major Electronic Design Automation company employs simulation-based techniques for functional verification, which heavily depend on the *quality* of test inputs with which the design is simulated. The standard industrial approach is *constrained-random verification*, in which random solutions of carefully crafted constraints are used as test inputs. Prior techniques either fail to provide theoretical guarantees on the quality of test inputs generated or fail to scale beyond toy designs. We extended our hashing-based framework, UniGen, to design a scalable distributed method for test-input generation that comes with rigorous approximation guarantees on the quality of generated stimuli [Chakraborty *et al.*, 2015].

4 Functional Synthesis

Functional synthesis concerns with the automatic synthesis of programs (also represented as circuits) that provably meet the end user's functional requirements. We introduced a novel data-driven approach, manthan, that combines advances in machine learning and formal methods [Golia *et al.*, 2020; Golia *et al.*, 2021b]: we rely on machine learning to generate candidate functions and then rely on formal methods to repair the generated candidate functions to synthesize functions that provably meet the end user's functional requirements. manthan consists of three phases: we first exploit the recent advances in constrained sampling to generate samples from the given relational specification, say $R(X, Y)$ over inputs X and outputs Y. Next, we cast the functional synthesis as a classification problem wherein the input variables (X)

in samples correspond to features while output variables (Y) correspond to labels. Consequently, the generated samples are fed as training data, and the learned classifier is encoded as a Boolean function. Often, machine learning techniques are capable of generating *almost correct* and our approach relies on advances in SAT and MaxSAT to generate a counterexample if the generated candidate functions do not satisfy specification. In contrast to the traditional approach of counterexample guided abstraction refinement (CEGAR), wherein a refinement is applied for the generated counterexample, we employ MaxSAT solver to search for a more *generalizable counterexample*. Finally, we employ the UNSAT core to repair the generated function, and we terminate once the repaired function satisfies the specification.

4.1 Applications

Program Synthesis The approach of combining formal methods and machine learning has led to dramatic scalability: on the standard suite of 609 instances, the prior state of the art techniques ranged from 210 to 280 instances while manthan can solve 509 instances; therefore, manthan is the only functional synthesis technique to solve more than 300 instances. Motivated by the impressive scalability, we turned our attention to program synthesis tasks commonly represented in Syntax Guided Synthesis (SyGuS) form. We demonstrated that the problem of program synthesis, represented in SyGuS, reduces to functional synthesis when there are no restrictions on the grammar [Golia *et al.*, 2021a]. Our reduction allows us to transform manthan as a state of the art approach for program synthesis tasks over bit-vector theory.

Scalable Interpretable Rule Learning We developed MLIC, a scalable interpretable decision rule learning framework that provides a precise control of accuracy vs. interpretability [Malioutov and Meel, 2018; Ghosh and Meel, 2019]. The key strength of the framework lies its ability to separate the modeling from the optimization and therefore has applications in a wide variety of interpretable classification formulations, including group-sparsity and having prior knowledge on variable importance. We observed that a major bottleneck to scalability of MLIC is the size (i.e., number of clauses) of optimization query, which would be linear in the number of samples in the training data. To achieve scalability, we introduced a novel partitioning-based incremental optimization solving approach that achieves scalability by invoking optimization solver linearly many times wherein each query is over a fixed size formula. The resulting open-source tool can now handle problems involving up to million samples in training data.

5 Data-Driven System Design

The modern SAT solvers are designed to be general purpose but the algorithmic frameworks for *Beyond SAT* present opportunities for tighter integration between solvers and algorithms so as to allow the solvers to exploit the structure of the queries. We exploit such opportunities via new paradigms for SAT sovlers, and the resulting tight coupling leads to significant scalability gains.

Native Support for Parity (XOR) Constraints The constrained counting and sampling techniques heavily rely on the usage of pairwise independent hash functions encoded as random XOR (parity) constraints. Consequently, the SAT solver is queried with conjunction of CNF and XOR constraints, also known as CNF-XOR formulas. Profiling of counters and samplers revealed that over 99% of the runtime is consumed by the underlying SAT solvers, thereby critically highlighting the need for efficient design of solvers with native support for CNF-XOR formulas. We introduced a novel framework, called bird, which put forth an unconventional architecture: bird first blasts the XOR constraints into CNF so that the entire formula is available to the solver in CNF [Soos and Meel, 2019; Soos *et al.*, 2020]. Furthermore, after every inprocessing step, bird recovers XOR constraints so that CDCL can be performed with on-the-fly Gauss-Jordan Elimination on the recovered XOR constraints. Our specialized data structure operations based on matrix row handling achieve efficient propagation and conflict checking of XOR constraints. Our project on verification of Binarized Neural Networks (BNN) showed that BNNs can be encoded naturally into Pseudo-Boolean (PB) constraints, and highlighted the need for counting tool with native support for PB. Consequently, we developed LinPB, a PB solver with native support for PB-XOR formulas [Yang and Meel, 2021], and ApproxMC augmented with LinPB is able to handle significantly larger neural networks.

Acknowledgments

The author is deeply grateful to his co-authors, without whom this work would not have been possible. The author would like to give a special shout out to his friend and long-term collaborator, Dr. Mate Soos. It is hard to imagine the fate of ApproxMC and UniGen had Soos, in 2018, not agreed to be the lead designer.

References

[Agrawal *et al.*, 2020] D. Agrawal, Bhavishya, and K. S. Meel. On the sparsity of xors in approximate model counting. In *Proc. of SAT*, 2020.

[Baluta *et al.*, 2019] T. Baluta, Shiqi Shen, Shweta Shine, K. S. Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In *Proc. of CCS*, 11 2019.

[Baluta *et al.*, 2021] T. Baluta, Zheng Leong Chua, K. S. Meel, and Prateek Saxena. Scalable quantitative verification for deep neural networks. In *Proc. of ICSE*, 2021.

[Baranov *et al.*, 2020] E. Baranov, A. Legay, and K. S. Meel. Baital: An adaptive weighted sampling approach for improved t-wise coverage. In *Proc. of FSE*, 2020.

[Biondi *et al.*, 2018] F. Biondi, M. Enescu, A. Heuser, A. Legay, K. S. Meel, and J. Quilbeuf. Scalable approximation of quantitative information flow in programs. In *Proc. of VMCAI*, 2018.

[Carter and Wegman, 1977] L. J. Carter and M. N Wegman. Universal classes of hash functions. In *Proc. of STOC*, 1977.

[Chakraborty and Meel, 2019] S. Chakraborty and K. S. Meel. On testing of uniform samplers. In *Proc. of AAAI*, 2019.

[Chakraborty *et al.*, 2013a] S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable and nearly uniform generator of sat witnesses. In *Proc. of CAV*, 2013.

[Chakraborty *et al.*, 2013b] S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In *Proc. of CP*, 2013.

[Chakraborty *et al.*, 2014a] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, 2014.

[Chakraborty *et al.*, 2014b] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Balancing scalability and uniformity in sat-witness generator. In *Proc. of DAC*, 2014.

[Chakraborty *et al.*, 2015] S. Chakraborty, D. J Fremont, K. S. Meel, and Moshe Y Seshia, S. A.and Vardi. On parallel scalable uniform sat witness generation. In *Proc. of TACAS*, 2015.

[Chakraborty *et al.*, 2016a] S. Chakraborty, K. S. Meel, R. Mistry, and M. Y. Vardi. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*, 2016.

[Chakraborty *et al.*, 2016b] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In *Proc. of IJCAI*, 2016.

[Dudek *et al.*, 2020] Jeffrey M. Dudek, Dror Fried, and K. S. Meel. Taming discrete integration via the boon of dimensionality. In *Proc. of NeurIPS*, 12 2020.

[Duenas-Osorio *et al.*, 2017] L. Duenas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proc. of AAAI*, 2017.

[Ghosh and Meel, 2019] B. Ghosh and K. S. Meel. Imli: An incremental framework for maxsat-based learning of interpretable classification rules. In *Proc. of AIES*, 2019.

[Golia *et al.*, 2020] P. Golia, S. Roy, and K. S. Meel. Manthan: A data-driven approach for boolean function synthesis. In *Proc. of CAV*, 2020.

[Golia *et al.*, 2021a] P. Golia, S. Roy, and K. S. Meel. Program synthesis as dependency quantified formula modulo theory. In *Proc. of IJCAI*, 2021.

[Golia *et al.*, 2021b] P. Golia, Friedrich Slivovsky, S. Roy, and K. S. Meel. Engineering an efficient boolean functional synthesis engine. In *Proc. of ICCAD*, 2021.

[Golia *et al.*, 2021c] P. Golia, M. Soos, S. Chakraborty, and K. S. Meel. Designing samplers is easy: The boon of testers. In *Proc. of FMCAD*, 2021.

[Jerrum and Sinclair, 1996] 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.

[Jerrum *et al.*, 1986] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986.

[Malioutov and Meel, 2018] Dmitry Malioutov and K. S. Meel. Mlic: A maxsat-based framework for learning interpretable classification rules. In *Proc. of CP*, 08 2018.

[Meel *et al.*, 2016] K. S. Meel, M. Y. Vardi, S. Chakraborty, D. J. Fremont, S. A. Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained sampling and counting: Universal hashing meets sat solving. In *Proc. of Beyond NP Workshop*, 2016.

[Meel *et al.*, 2018] K. S. Meel, Aditya A. Shrotri, and M. Y. Vardi. Not all fprass are equal: Demystifying fprass for dnf-counting. In *Proc. of CP*, 08 2018.

[Meel *et al.*, 2020] K. S. Meel, Y. Pote, and S. Chakraborty. On testing of samplers. In *Proc. of NeurIPS*, 12 2020.

[Meel, 2020] S. Meel, K. S. ⓡ Akshay. Sparse hashing for scalable approximate model counting: Theory and practice. In *Proc. of LICS*, 2020.

[Meel, 2021] NV ⓡ Chakraborty S. Meel, K. S. ⓡ Vinodchandran. Estimating the size of union of sets in streaming models. In *Proc. of PODS*, 2021.

[Paredes *et al.*, 2019] R. Paredes, L. Duenas-Osorio, K. S. Meel, and M. Y. Vardi. Principled network reliability approximation: A counting-based approach. *Reliability Engineering and System Safety*, 11 2019.

[Pavan, 2021] N. V. ⓡ Bhattacharya A. ⓡ Meel K. S. Pavan, A. ⓡ Vinodchandran. Model counting meets F_0 estimation. In *Proc. of PODS*, 2021.

[Pote, 2021] K. S. Pote, Y. ⓡ Meel. Testing probabilistic circuits. In *Proc. of NeurIPS*, 12 2021.

[Soos and Meel, 2019] M. Soos and K. S. Meel. Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proc. of AAAI*, 2019.

[Soos *et al.*, 2020] M. Soos, Stephan Gocht, and K. S. Meel. Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling. In *Proc. of CAV*, 2020.

[Stockmeyer, 1983] L. Stockmeyer. The complexity of approximate counting. In *Proc. of STOC*, 1983.

[Toda, 1989] S. Toda. On the computational power of PP and (+)P. In *Proc. of FOCS*, pages 514–519. IEEE, 1989.

[Valiant, 1979] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 1979.

[Wang *et al.*, 2020] W. Wang, M. Usman, A. Almaawi, K. Wang, K. S. Meel, and S. Khurshid. A study of symmetry breaking predicates and model counting. In *Proc. of TACAS*, 2020.

[Yang and Meel, 2021] J. Yang and K. S. Meel. Engineering an efficient pb-xor solver. In *Proc. of CP*, 2021.