CSB: A Counting and Sampling tool for Bit-vectors

Arijit Shaw[1,2], Kuldeep S. Meel[3]

[1]Chennai Mathematical Institute, India
[2]IAI, TCG-CREST, Kolkata, India
[3]University of Toronto, Canada

Abstract
Satisfiability Modulo Theory (SMT) solvers have significantly advanced automated reasoning due to their effectiveness in solving problems across various fields. With the advancement in SMT solvers, there is growing interest in exploring capabilities beyond mere satisfiability, similar to the progression observed in Boolean satisfiability solvers that expanded into counting and sampling. In this study, we investigate the following question: *Can we rely on modern CNF model counters and CNF samplers to extend a modern SMT solvers to handle the problems of counting and sampling over bit-vectors?*

The main contribution of this work is the development of an efficient approximate model counter along with the first almost-uniform and uniform-like samplers for the theory of bit-vectors. Our tool[1] csb converts the bit-vector formula into a CNF formula using bit-blasting techniques before applying CNF model counters or samplers to perform counting or sampling. Our tool enhances the SMT solver STP with an approximate model counter ApproxMC, an almost-uniform sampler UniGen, and a uniform-like sampler CMSGen. Our experiments demonstrate significant performance improvements over existing methods.

1. Introduction

The paradigm of Satisfiability Modulo Theory (SMT) solving has been central to advances in hardware and software verification over the past two decades. Over time, the community has developed several scalable state-of-the-art SMT solvers. For many problem instances, satisfiability alone does not suffice, and one is often interested in computations over the solution set. Two such problems are computing an estimate of the cardinality of the solution set and uniformly sampling a solution from the entire solution set. As a starting point, we focus on the case when the underlying formula is expressed in QF_BV*, or quantifier-free bit-vector arithmetic* (referred to as *bit-vectors* hereafter). Our choice of QF_BV stems from its being one of the first theories to be investigated in the context of SMT solving, as well as recent empirical studies showing the importance of counting problems over bit-vector formulas in domains such as cryptography [1] and software verification [2, 3].

The problem of counting and sampling over bit-vectors can be addressed through two methods: (i) reasoning directly over bit-vectors, or (ii) reducing the problem to CNF. While the former approach has been studied in recent years using lifting techniques for word-level constraints [4, 5, 6, 7], the latter has not been thoroughly assessed. In this paper, we focus on whether a modern SMT solver can be extended with CNF-based model counters and CNF samplers to efficiently address the problems of counting and sampling over bit-vectors. Our investigation into the design of a counting tool for bit-vector formulas relies on bit-blasting, followed by the use of CNF-based samplers and counters, leveraging the latter's scalability.

To test the scalability of our system, we compiled a collection of 679 benchmarks from various practical domains, such as cryptography and software verification, encoded as the problem of bit-vector model counting. The primary contribution of this paper is the development of an efficient bit-vector counting and sampling tool. Specifically,

1. We introduce csb, a bit-vector model counter that extends the SMT solver STP to incorporate off-the-shelf CNF-counters. Out of a set of 679 bit-vector counting problems, csb successfully

[1]Open source tool available at: https://github.com/meelgroup/csb

22nd International Workshop on Satisfiability Modulo Theories, July 22-23, 2024, Montreal, Canada.

counts 643 benchmarks, surpassing the previous state-of-the-art counter, SMTApproxMC, which counts 135 benchmarks.

2. Furthermore, we extend csb to function as a sampler by integrating CNF-samplers. On average, it takes 7.4 seconds and 273.6 seconds to generate 500 uniform-like and almost-uniform samples, respectively, from our benchmark set in these modes.

The rest of the paper is organized as follows: We introduce the preliminaries and related work in Section 2. In Section 3, we present an overview of our framework, csb. We describe our experimental methodology and results in Section 4. Finally, we conclude in Section 5.

2. Background

A *word* (or *bit-vector*) is a vector of bits of a given size. Let X be the set of word-level variables, and let F be a formula in the theory of *quantifier free bit-vectors.* A *model* or *solution* of F is an assignment of word-level constants to variables in X such that F evaluates to True. Let $\mathsf{Sol}(F)$ denote the set of models of F. The problem of counting is to compute $|\mathsf{Sol}(F)|$.

An *approximate model counter* takes in a formula F, tolerance parameter ε, confidence parameter δ and returns c such that $\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathsf{Sol}(F)|\right] \geq 1-\delta$. An *almost uniform sampler* G takes a tolerance parameter ε along with F , and guarantees $\forall y \in \mathsf{Sol}(F)$, $\frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)|} \leq \Pr[G(F,\varepsilon) = y] \leq \frac{(1+\varepsilon)}{|\mathsf{Sol}(F)|}$. An *uniform-like sampler* also takes in a Boolean formula F and returns $\sigma \in \mathsf{Sol}(F)$, and is designed to behave like a uniform sampler, albeit without theoretical guarantees. Instead of theoretical analysis, the design of these samplers are influenced by recently proposed distribution testing-based tool, Barbarik [8, 9].

Related Work. The success of propositional model counters, especially approximate ones, led to the adaptation of these techniques for word-level constraints. Chistikov et al. [4] extended hashing-based model counting from the approximate CNF model counter ApproxMC [10] to word-level benchmarks using bit-blasting. Chakraborty et al. [5] developed SMTApproxMC, which lifts hash functions to handle word-level constraints. Kim et al. introduced a system for statistical estimation to continuously refine the probabilistic estimate of model counts, although it lacks (ε,δ)-guarantees [11].

In the sampling domain, the need for uniform samplers and those achieving high coverage is well-established. There have been considerable efforts to sample from SMT formulas with high coverage [6, 7, 12], yet uniform sampling from these formulas remains largely unexplored, representing a significant challenge. The technique of lifting hash functions, as used in [5], is ineffective for uniform sampling due to the need for 3-wise independence, whereas the lifted hash functions in [13] only ensure 2-wise independence.

3. Approach

We develop csb, a model counting and sampling tool for the quantifier-free fragment of the theory of bit-vectors. To achieve this, csb first bit-blasts the formula to a Boolean CNF formula, then applies an off-the-shelf CNF model counter or CNF sampler to solve the problems of model counting and sampling, respectively. By reducing the problem to CNF, our tool csb leverages ongoing improvements in propositional model counting and sampling.

3.1. Model Counting

As shown in Figure 1, the model counting part of csb comprises two primary components: (i) a tool for bit-blasting SMT2 formulas into CNFs, and (ii) an off-the-shelf model counter.

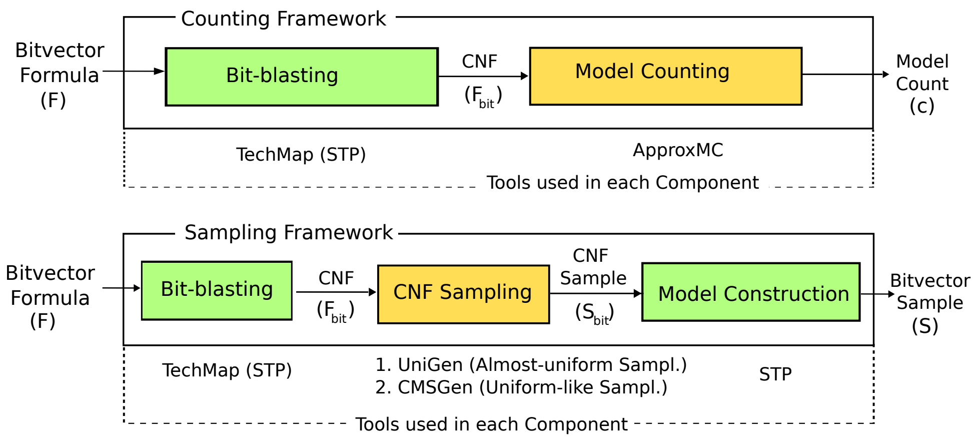

Figure 1: The architecture for counting and sampling in csb.

Phase 1: Bit-blasting. The bit-blasting component takes an input bit-vector formula F and outputs a CNF formula F_{bit} such that $|\mathsf{Sol}(F_{\text{bit}})| = |\mathsf{Sol}(F)|$. The most commonly used method for converting a bit-vector formula into a propositional Boolean formula involves representing the formula as an And-Inverter Graph (AIG) circuit, which is subsequently converted into conjunctive normal form (CNF). There are multiple techniques available for converting an AIG to CNF, with the two most commonly used being: (i) TseitinEnc: the standard Tseitin encoding [14] and (ii) TechMap: technology mapping-based logic synthesis [15]. The TseitinEnc method uses auxiliary variables for sub-circuits of AIG, whereas the TechMap method performs various optimizations on the circuit to produce a minimized CNF. From the perspective of satisfiability, studies have revealed that the performance of the encoding scheme is reliant on the benchmark set being investigated, with TseitinEnc exhibiting superior results in some benchmark sets and TechMap being more effective in others [16]. In context of csb, TseitinEnc and TechMap showed similar performance, and we use TechMap for bit-blasting.

Phase 2: Propositional Model Counting. The second phase of csb involves a propositional model counter that takes the CNF formula F_{bit} and returns c, an approximation of $|\mathsf{Sol}(F_{\text{bit}})|$. We investigated various approaches to model counting, including both compilation-based counters and hashing-based approximate counters. In recent years, there has been a proliferation of model counters, and the annual model counting competitions have demonstrated that different model counters have distinct strengths and weaknesses for various problem types. In the context of this research, we examine the applicability of five model counters within the framework of csb: four compilation-based counters SharpSAT-TD [17], Ganak [18], ADDMC [19], and ExactMC [20], and a hashing-based approximate counter ApproxMC [21]. In csb, we use ApproxMC as the model counter because it shows the best performance on the test instances.

3.2. Sampling

As shown in Figure 1, the sampling mode of csb comprises three primary components: (i) a tool for bit-blasting SMT2 formulas into CNFs, (ii) a CNF sampler to sample solutions (almost) uniformly from the solution space, and (iii) constructing the bit-vector model from the sampled CNF solution. In the following part of this section, we briefly describe the task and available tools for each component.

Phase 1: Bit-blasting. The bit-blasting component generates a propositional formula F_{bit} from the input bit-vector formula F, such that there is a one-to-one correspondence between $\mathsf{Sol}(F)$ and $\mathsf{Sol}(F_{\text{bit}})$. The techniques used in this part are the same as those in the bit-blasting component of the model counting mode. At this stage, the underlying SMT solver maintains the mapping between variables of F and F_{bit}, allowing it to map an element of $\text{Sol}(F_{\text{bit}})$ to the corresponding element of $\text{Sol}(F)$.

Phase 2: CNF Sampling. The bit-blasted formula F_{bit} is passed to a CNF sampler to generate a sample S_{bit}, a randomly selected element from $\text{Sol}(F_{\text{bit}})$. UniGen [22] uses hashing-based techniques to achieve almost-uniform sampling. On the other hand, CMSGen [9] incorporates randomization at various stages of a SAT solver's process, effectively making the solver's output mimic that of a uniform sampler. This approach resembles uniform sampling and meets distribution tests [23], validating its utility. In practical scenarios, uniform-like samplers often exhibit superior performance, although some applications necessitate strict uniformity guarantees. Depending on the specific sampling requirements, we employ different CNF samplers: for almost-uniform sampling, we utilize UniGen; for uniform-like sampling, we select CMSGen.

Phase 3: Model Construction. Once the CNF sampler returns a solution S_{bit}, it is passed to the SMT solver, which generates the corresponding bit-vector solution S. The SMT solver uses the variable mapping from phase 1 to perform this operation.

3.3. Preprocessing

In model counting and sampling, preprocessors attempt to simplify the original problem instance. In practical scenarios, the problem instances are typically rooted in circuits within a specific domain. These circuits are comprised of gates, and the variables correspond to either input or output variables, with the output variables being determined by the input variables. Preprocessing techniques have been developed to effectively handle the *input-output bipartition* property, as discussed in several studies [24, 25, 26]. In csb, we utilize Arjun [26] as a preprocessor.

3.4. Implementation

A simple approach to build the tool is to use a standalone shell script that interfaces with an SMT solver to generate a CNF file, followed by processing with a model counter or sampler. However, using shell scripts could lead to performance bottlenecks due to the overhead of file read-writes, complicate error handling, and create challenges in deployment and portability by requiring specific environmental setups and external binaries. Therefore, we chose a tightly integrated approach, embedding the counter and samplers directly within the SMT solver as a library. This integration yields a single binary that efficiently produces both model counts and samples.We implement csb using STP as the underlying SMT solver. We integrate ApproxMC to implement the model counter, and UniGen and CMSGen to implement the samplers. We also integrate Arjun as a preprocessor. The default parameters for the approximate model counting mode are $\varepsilon = 0.8$ and $\delta = 0.2$, adhering to the standard values used in the model counting community.

4. Experimental Evaluation

The evaluation procedure was conducted using the following setup. We conducted all our experiments on a high-performance computer cluster, with each node consisting of an AMD EPYC 7713 CPU. We set the memory limit to 16 GB for all configurations and ran each solver instance on a single core. To adhere to the standard timeout used in model counting competitions, we set the timeout for all experiments to 3600 seconds.

Baseline. For the problem of model counting, we compare our performance with the prior state-of-the-art bit-vector model counter, SMTApproxMC. As noted in the related works, there are no available uniform samplers for bit-vectors, so we evaluate the relative performance of uniform samplers based on the total number of benchmarks.

Counter	Instances Counted	Average Runtime (s)
SMTApproxMC	135	5854.8
csb	**643**	**653.5**

Sampling Mode in csb	Instances Sampled	Average Runtime (s)	Median Runtime (s)
Almost-uniform sampling	641	273.6	74.6
Uniform-like sampling	662	7.4	1.24

Table 1
Number of instances finished by counting and sampling mode of csb, out of 679 instances.

Benchmarks. We collected a substantial set of benchmarks that pertain to the problem of model counting and naturally encode them into bit-vector formulas. The benchmarks were produced using several software tools for various purposes. These include CounterSharp [3], which is a quantitative software reliability estimation tool; an algorithm designed for testing robust reachability [2]; Delphinium, which is a cryptographic tool for automating chosen ciphertext attacks [1]; and SMC, a previous work on bit-vector counting [11]. The total number of benchmarks used in our evaluation amounts to 679. In this work, we sought to answer the following questions:

RQ1. How does csb's performance compare with the existing state-of-the-art?

RQ2. How do the various components of csb, such as model counting, sampling, and bit-blasting, impact its overall performance?

Summary of Results. In model counting mode, csb can solve over four times more instances than SMTApproxMC. Out of a total of 679 instances, csb can count 643, whereas SMTApproxMC is able to count 135. In sampling mode, csb generated samples from almost all the problem formulas. The uniform-like sampling mode showed better efficiency in generating samples.

RQ1. Comparison with Prior State-of-the-Art

We experimented with configurations for csb and compared its performance with SMTApproxMC, the current state-of-the-art bit-vector model counter. Table 1 shows the improvement in performance demonstrated by csb. In the cactus plot in Figure 2, we compare the performance. The x-axis represents the number of instances, while the y-axis shows the time taken. A point (i, j) in the plot indicates that a counter counted j benchmarks out of the total benchmarks in a set in less than or equal to i seconds. The results indicate that csb counted 643 instances, which is four times the number of instances counted by SMTApproxMC, which could count 135 instances.

As there is no existing uniform bit-vector sampling tool, we do not have a baseline for comparing csb against. We tested csb on our benchmarks in both sampling modes by asking it to generate 500 samples for each input formula. In Table 1, we compare the performance in terms of number of instances solved and average runtime[1]. Out of 679 instances, in almost-uniform sampling mode csb generated samples for 641 instances, while in uniform-like sampling mode, it solved 662 instances.

RQ2. Impact of Different Components

As shown in Section 3, there are multiple components for csb, and there are multiple tools available for each component. To determine which tools to use for each component, we performed experiments. Below, we summarize the impact of each component.

[1]We use the PAR-2 score as a proxy for average runtime. PAR-2 score (penalized average runtime) assigns a runtime of two times the time limit for each benchmark not solved by a counter.

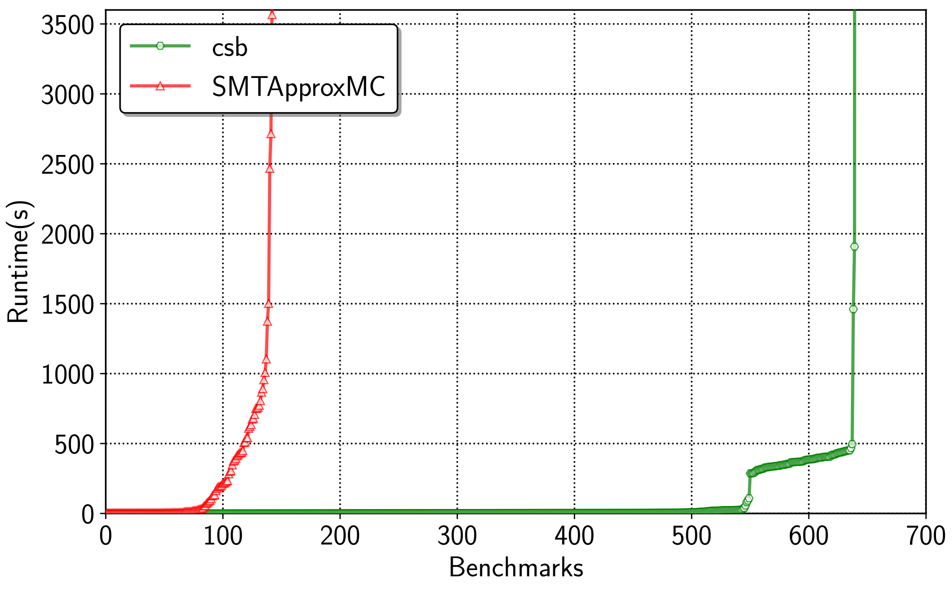

Figure 2: Performance of csb model counter w.r.t. state-of-the-art

Impact of CNF Counters. In the cactus plot of Figure 3, we compare the performance of the model counters. When csb is combined with the best possible preprocessing settings for each model counter, the best performance is shown by ApproxMC, solving 643 out of 679 benchmarks. The model counter SharpSAT-TD shows the second-best performance, solving 409 benchmarks. Conversely, the performance of other model counters such as ExactMC, ADDMC, and Ganak was poorer, with each solving nearly 300 benchmarks.

Impact of CNF Samplers. Both CNF samplers demonstrate strong performance, solving over 94% of instances. The average runtime is significantly improved when CMSGen is used as the sampler. In Table 1, we compare the performance metrics. Using CMSGen as the sampler, csb solves 21 more instances and reduces the average runtime to 3% of that with UniGen.

Impact of Preprocessing. The preprocessors had a minimal positive impact on the performance of exact model counting tools, and in a few cases, they even showed a slight negative effect. For the approximate model counting tool ApproxMC, Arjun demonstrated a positive impact, enabling the solving of 643 instances, which is 473 more than without preprocessing.

5. Conclusion

This paper introduces csb, an extension of the SMT solver STP that leverages CNF counters, and samplers for bit-vector formulas. csb enhances performance, achieving a fourfold improvement in bit-vector model counting and efficiently solving uniform sampling of bit-vector constraints. We identify optimal methods for bit-vector counting and explore uniform sampling from SMT formulas.

Acknowledgements. We are thankful to anonymous reviewers for their constructive comments to improve this paper. 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 this work was completed while Arijit Shaw was a visiting graduate student at the University of Toronto. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg).

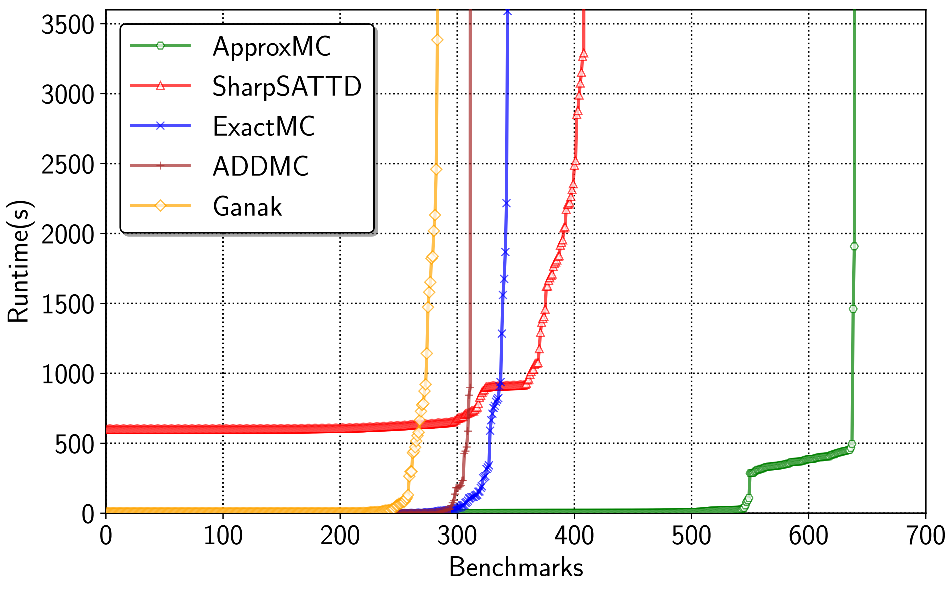

Figure 3: Performance of csb with different model counters.

References

[1] G. Beck, M. Zinkus, M. Green, Automating the development of chosen ciphertext attacks, in: Proc. of USENIX Security, 2020.

[2] G. Girol, B. Farinier, S. Bardin, Not all bugs are created equal, but robust reachability can tell the difference, in: Proc. of CAV, 2021.

[3] S. Teuber, A. Weigl, Quantifying software reliability via model-counting, in: Proc. of Quantitative Evaluation of Systems, 2021.

[4] D. Chistikov, R. Dimitrova, R. Majumdar, Approximate counting in SMT and value estimation for probabilistic programs, in: Proc. of TACAS, 2015.

[5] S. Chakraborty, K. Meel, R. Mistry, M. Vardi, Approximate probabilistic inference via word-level counting, in: Proc. of AAAI, 2016.

[6] R. Dutra, J. Bachrach, K. Sen, Smtsampler: Efficient stimulus generation from complex smt constraints, in: Proc. of ICCAD, 2018.

[7] R. Dutra, J. Bachrach, K. Sen, Guidedsampler: coverage-guided sampling of smt solutions, in: Proc. of FMCAD, 2019.

[8] S. Chakraborty, K. S. Meel, On testing of uniform samplers, in: Proc. of AAAI, 2019.

[9] P. Golia, M. Soos, S. Chakraborty, K. S. Meel, Designing samplers is easy: The boon of testers, in: Proc. of FMCAD, 2021.

[10] S. Chakraborty, K. S. Meel, M. Y. Vardi, A scalable approximate model counter, in: Proc. of CP, 2013.

[11] S. Kim, S. McCamant, Bit-vector model counting using statistical estimation, in: Proc. of TACAS, 2018.

[12] M. I. Peled, B.-C. Rothenberg, S. Itzhaky, Smt sampling via model-guided approximation, in: Proc. of FM, 2023.

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

[14] G. S. Tseitin, On the complexity of derivation in propositional calculus, in: Automation of reasoning, Springer, 1983.

[15] N. Eén, A. Mishchenko, N. Sörensson, Applying logic synthesis for speeding up SAT, in: Proc. of SAT, 2007.

[16] S. Jha, R. Limaye, S. A. Seshia, Beaver: Engineering an efficient smt solver for bit-vector arithmetic,

in: Proc. of CAV, 2009.
[17] T. Korhonen, M. Järvisalo, Integrating tree decompositions into decision heuristics of propositional model counters, in: Proc. of CP, 2021.
[18] S. Sharma, S. Roy, M. Soos, K. S. Meel, Ganak: A scalable probabilistic exact model counter., in: Proc. of IJCAI, 2019.
[19] J. M. Dudek, V. H. Phan, M. Y. Vardi, ADDMC: Weighted model counting with algebraic decision diagrams, Proc. of AAAI (2020).
[20] Y. Lai, K. S. Meel, R. H. Yap, The power of literal equivalence in model counting, in: Proc. of AAAI, 2021.
[21] M. Soos, K. S. Meel, BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting, in: Proc. of AAAI, 2019.
[22] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, M. Y. Vardi, On parallel scalable uniform sat witness generation, in: Proc. of TACAS, 2015.
[23] K. S. Meel, Y. P. Pote, S. Chakraborty, On testing of samplers, Proc. of NeurIPS (2020).
[24] J.-M. Lagniez, E. Lonca, P. Marquis, Improving model counting by leveraging definability., in: Proc. of IJCAI, 2016.
[25] J.-M. Lagniez, E. Lonca, P. Marquis, Definability for model counting, Artificial Intelligence (2020).
[26] M. Soos, K. S. Meel, Arjun: An efficient independent support computation technique and its applications to counting and sampling, in: Proc. of ICCAD, 2022.