MCBAT: A Practical Tool for Model Counting Constraints on Bounded Integer Arrays

Abtin Molavi
amolavi@hmc.edu
Harvey Mudd College
Claremont, California, USA

Mara Downing
mdowning@hmc.edu
Harvey Mudd College
Claremont, California, USA

Tommy Schneider
tschneider@hmc.edu
Harvey Mudd College
Claremont, California, USA

Lucas Bang
lbang@hmc.edu
Harvey Mudd College
Claremont, California, USA

ABSTRACT

Model counting procedures for data structures are crucial for advancing the field of automated quantitative program analysis. We present a tool for Model Counting for Bounded Array Theory (MCBAT). MCBAT works on quantified integer array constraints in which all arrays have a finite length. We employ reductions from the theory of arrays to uninterpreted functions and linear integer arithmetic (LIA). Once reduced to LIA, we leverage Barvinok's polynomial time integer lattice point enumeration algorithm. Finally, we present a case study demonstrating applicability to automated quantitative program analysis. MCBAT is available for immediate use as a Docker image and the source code is freely available in our Github repository.

CCS CONCEPTS

• **Software and its engineering → Formal methods**; • **Mathematics of computing → Combinatoric problems**.

KEYWORDS

Model Counting, Array Theory, SMT

ACM Reference Format:
Abtin Molavi, Mara Downing, Tommy Schneider, and Lucas Bang. 2020. MCBAT: A Practical Tool for Model Counting Constraints on Bounded Integer Arrays. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3368089.3417937

1 INTRODUCTION

Model counting is the enabling technology and theory behind automated quantitative program analyses. The ability to count the number of solutions to a constraint allows one to perform reliability analysis [13], probabilistic symbolic execution [15], quantitative information flow analysis [16, 23, 29, 31], Bayesian inference [8, 11, 26], and compiler optimization [25]. Originally stated with respect to Boolean formulas [5], more recent advances in model counting have extended counting capabilities to the theories of linear integer arithmetic [19, 30], non-linear numeric constraints [6], strings [20, 27, 28], word-level counting for bit-vectors applied to the problem of automatic inference [7], and more recent work has begun to combine theories of strings and integers [2]. This paper presents the first tool with the capability of performing model counting directly on formulas with symbolic arrays.

The current space of exploration in model counting is driven by the ubiquity of the types found in common programming languages– Booleans, integers, and strings. In this paper, we expand the space of model counting tools by introducing MCBAT for counting the number of models to constraints over the theory of bounded integer arrays. We hypothesize that MCBAT can be used by quantitative program analysis researchers as a basis for analyzing other structures that can be modeled as arrays: vectors, maps, hash tables, memory caches, and so on. Our tool, MCBAT, is freely available as a Docker image for immediate use[1] and the source code of MCBAT is located in our public Github repository[2].Typical use cases for MCBAT are highlighted in the demonstration video[3].

Our previous work [22] details the algorithms and theory behind MCBAT. Namely, that work proves that our sequence of reductions that are used to convert a formula over the theory of arrays into a pure linear integer arithmetic formula are model-count preserving (MCBAT is theoretically correct). In addition, that earlier work empirically verified the correctness of MCBAT by comparing to a naive model enumeration algorithm using Z3 (MCBAT is experimentally verified to be correct). This paper presents MCBAT as a practical tool to be used by researchers in areas that require model counting software like quantitative information flow, probabilistic symbolic execution, or expected cost analysis of programs, when the program under test is one that operates on arrays. A case study on QIF demonstrating the applicability of MCBAT is given in Section 5.

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3417937

[1] https://hub.docker.com/repository/docker/abtinm/mcbat
[2] https://github.com/hmc-alpaqa/mcbat
[3] https://www.cs.hmc.edu/ bang/mcbat.html

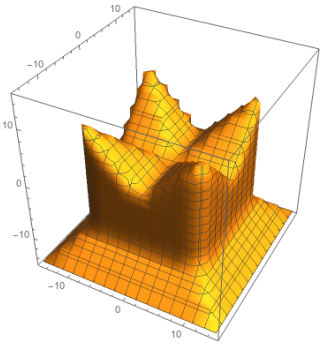

Figure 1: Polytope defined by constraint of Example.

2 BACKGROUND

Let's start with a simple example of an array constraint. Consider a constraint over integer array a and integer variable k:

$$\textsc{length}(A) = 2 \wedge (k \geq -15) \wedge \forall i\ (k \leq a[i] \leq 10 \vee k \leq -a[i] \leq 10)$$

This constraint is equivalent to a constraint with three variables $a_0, a_1, k \in \mathbb{Z}$:

$$k \geq -15 \wedge (k \leq a_0 \leq 10 \vee k \leq -a_0 \leq 10) \wedge$$
$$(k \leq |a_1| \leq 10 \vee k \leq -a_1 \leq 10)$$

This constraint over three variables defines a polytope P in $\mathbb{R}^3$ (Fig. 1). Observe that integer lattice points in P correspond to integer triples (k, a_0, a_1), corresponding to the free variables k and a of the original constraint. Our procedure counts the number of possible models for all free variables in a constraint. For this example, the number of integer lattice points in this polytope, and therefore the number of models to the original constraint, is 10076.

This example illustrate the main idea of our approach: count models for an array constraint by transforming it into an instance of lattice point counting within a polytope. While this example is easy to visualize, in general, a finite array constraint over integers is model-count equivalent to a set of lattice points in a multi-dimensional polytope.

3 ALGORITHM

MCBAT Input. MCBAT takes a formula in the theory of bounded arrays of the form

$$\phi(a_1, \ldots, a_n; k_1, \ldots, k_w) = \textsc{length}(a_1, \ell_1) \wedge \cdots \textsc{length}(a_n, \ell_n) \wedge \phi_A$$

where ϕ_A is a Boolean combination of quantified array formulas. Here, we've explicitly denoted the n free array-variables and the w free integer-variables. Throughout our algorithm some steps may introduce new free variables, but we ensure that the model count is preserved.

MCBAT Output. We output the number of models there are for $\phi(a_1, \ldots, a_n; k_1, \ldots, k_w)$.

High-Level Overview. MCBAT (Algorithm 1) has these main steps:

- Decompose a boolean combination of quantified array formulas into individual quantified array formulas.
- Replace index terms that occur within array access terms with auxiliary integer variables; introduce auxiliary integer constraints to capture this replacement.
- Each array-store term is replaced by equivalent constraints that do not contain array-store expressions.
- Rewrite expressions that are universally quantified over array index variables as a conjunctions over all possible indices, with upper bounds enforced by each array s LENGTH predicate.
- Perform Ackermann's reduction, converting array access terms into integer terms.
- Send the resulting linear integer arithmetic constraint to BARVINOK [30] to compute the final model count.

Algorithm 1 MCBAT: Compute the model count for $\phi(a_1, \ldots, a_n; k_1, \ldots, k_w)$

procedure MCBAT($\phi(a_1, \ldots, a_n; k_1, \ldots, k_w)$)
  Decompose ϕ into a tree $\mathbf{T}$ of array formulas $\phi_1, \phi_2, \ldots, \phi_m$.
  Create a tree $\mathbf{T}'$ and a label-formula map $\mathbf{M}$ using $\phi_1, \phi_2, \ldots, \phi_m$ as labels.
  for ϕ_i **do**
    $\phi_i^{(1)} \longleftarrow \textsc{removeTermsInAccess}(\phi_i)$
    $\phi_i^{(2)} \longleftarrow \textsc{replaceAllArrayStores}(\phi_i^{(1)})$
    $\phi_i^{(3)} \longleftarrow \textsc{removeQuants}(\phi_i^{(2)})$
  end for
  Construct $\mathbf{M}'$ from $\mathbf{M}$ and the formulas $\phi_1^{(3)}, \ldots, \phi_m^{(3)}$.
  Construct $\phi^{(4)}$ by applying the label-formula map $\mathbf{M}'$ to the Boolean tree $\mathbf{T}'$.
  $\phi^{(5)} \longleftarrow \textsc{AckermannReduction}(\phi^{(4)})$
  return $\textsc{Barvinok}(\phi^{(5)})$
end procedure

4 IMPLEMENTATION

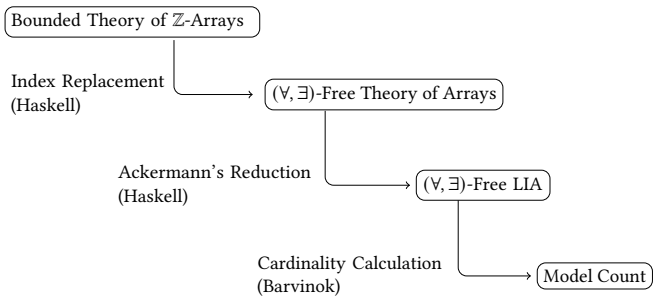

Figure 2: High-level view of MCBAT implementation architecture.

We implemented the MCBAT algorithm in a tool, also called MCBAT. A high level architecture of MCBAT can be seen in Fig. 2. The core MCBAT algorithm is implemented in a series of Haskell functions, eventually passing a quantifier-free linear integer arithmetic formula to the Barvinok library which returns the final model count.

Array constraints may be entered directly as Haskell expressions at the REPL. MCBAT also supports reading and writing constraint files in an SMT-LIB2-like format. This allows use of MCBAT as a command line utility that takes an SMT file as input and prints the count to the terminal. For example, an interaction with MCBAT might look like

```
> mcbat example1.smt
{10076}
```

The complete implementation is freely available along with the source code. In addition, our implementation has an associated Docker image, so that one can immediately download and run MCBAT in a virtual environment using a single terminal command. Assuming one has docker installed, MCBAT can be (downloaded and) run on the command line as

```
> docker run -it abtinm/mcbat
```

Once the MCBAT Docker container running, the user can explore the source code of MCBAT in the `src` directory, and example constraints arising from symbolic execution of array programs are contained in `src/test/ed/`.

The experimental evaluation of MCBAT is also replicable using this Docker image and the expected depth experiment can easily be extended to new sets of constraints using a special flag at the command line.

5 CASE STUDY

In order to demonstrate the usefulness of MCBAT as a tool for quantitative program analysis, here we present a short case study that makes use of model counting. Within the field of quantitative information flow (QIF) one often needs to compute probabilities of program events. Those probabilities can be computed using model counting. Here we discuss MCBAT's applicability to security analysis by providing a tool for computing those probabilities.

Consider the problem of determining if a piece of code might be vulnerable to side-channel attacks. In a timing-based side channel exploit, an attacker is able to observe the exection time of a function that operates over secret values. If the running time of the code is affected by the combination of the attacker's input values and the secret values stored in the program, an attacker can measure the running time of the system to gain partial information about the unknown secret [3, 16, 23, 29].

We briefly summarize how automatic static side-channel analysis techniques work. First symbolic execution is run on on a piece of code. The length of the symbolic execution path to each leaf node is taken as a proxy for the running time of that execution path. Each leaf has a path constraint ϕ_i that is then associated with a static estimate of the a timing observation o_i, based on the path length. Path constraints are merged into observation constraints ψ_j where each $\psi_j = \phi_{i_1} \vee \phi_{i_2} \vee \ldots \vee \phi_{i_{n_j}}$, the disjunction of the n_j path constraints that lead to the same observation (i.e. $o_{i_1} = o_{i_2} = \ldots = o_{i_{n_j}}$. Each ψ_j describes all program observations that can lead to the same timing observations from the point of view of the attacker.

A bound on the amount of information $\mathcal{H}$ that an attacker can gain about a secret value is then given by the Shannon entropy of the distribution of the m possible observations. We compute $\mathcal{H} = \sum_{j=1}^{m} p(o_j) \log_2(1/p(o_j))$. Now, observe that, assuming a uniform distribution of progam inputs, the probability of an observation can be computed by performing model counting on the observation contraints: $p(o_j) = \#(\psi_j)/\sum_{i=1}^{m} \#(\psi_i)$. (This is the fundamental idea behind probabilistic symbolic execution [31].) One then computes the Shannon entropy to compute an upper bound on the amount of *information leaked* to an attacker through the timing side channel.

Table 1: Information leakage in number of bits for a set of functions computed using MCBAT.

Function	Leakage
Check for element at index	0.00
Search for a number	0.00
Check array equality	0.00
Lexicographic compare	1.00
Insert in sorted list	3.81
Insertion sort	3.45
Bubble sort	0.00
Find Maximum	0.00
Merge two sorted arrays	2.47
Longest common subsequence	0.00
Check if sorted	1.63

We applied this idea to a small case study of programs that use arrays. We performed symbolic execution using a custom Python-based symbolic execution engine to generate path constraints over arrays, and performed the above-described leakage computation. We found that our implementations of common array operations leaked the amounts of information shown in Table 1. Functions with zero leakage are likely to be safer with respect to timing side channels. All of the constraints used to perform this computation are provided in our repository.

6 RELATED WORK

The Theory of Arrays. In 1962, McCarthy introduced a formal theory of arrays based the two *select and store* axioms [21]. In more recent times, decision procedures for the theory of arrays have been developed and implemented, for instance in the Z3 SMT solver [9, 10]. A comprehensive treatment of satisfiability checking for array constraints is given in Kroening and Strichman's "Decision Procedures: An Algorithmic Point of View" [17].

We find that the most closely related work to ours is that of Plazar, et. al [24]. While their work is focused primarily on satisfiability checking of array constraints over arbitrary value types, the bounded fragment of array theory that they focus on and the resulting algorithm bear resemblances to our approaches that focus on bounded integer arrays. While the authors observe a "strong correspondence between the models to the input and transformed formulas", their algorithm, as it is not concerned with model counting, is not strong enough to fully maintain the model correspondence across transformations.

Applications of Array Constraint Procedures. SMT solving for arrays is an important component of symbolic execution for programs that operate on arrays, as in Symbolic Path Finder for Java[14]. Another useful application of satisfiability checking for

arrays is the synthesis of invariants over arrays [18] by Larraz, et. al., whose constraints we used as a benchmark for our experimental analysis. In this paper, we applied model counting to a case study on computing information leakage of functions operating on arrays. Existing work for computing information leakage of function that operate on arrays work by performing symbolic execution by creating arrays of symbolic integers, and therefore generating constraints on integers, rather than by creating symbolic arrays and dealing with array constraints directly [3, 29].

Model Counting. Initial work in model counting applied to formulas of propositional logic. Of particular interest is the use of DPLL as a model counting procedure [5]. Recent years have seen a significant increase in interest in model counting for domains beyond propositional logic. LattE [19] and Barvinok [30] are popular model counters for the theory of linear integer arithmetic. Closely related to the theory of arrays is the theory of strings, which are also an indexable type. Recent approaches to model counting for strings make use of generating functions [20], recurrence relations [28], and automata theory [1]. Earlier work on model counting for data structures exists in which Java code that defines a data structure is symbolically executed and the resulting constraints are model counted using LattE during analysis [12]. Finally, recent theoretical results have been shown for the problem of weighted model counting for constraints containing uninterpreted functions [4].

7 CONCLUSION

We presented tool, MCBAT, for performing model counting on constraints over integer arrays of bounded length. MCBAT performs a series of transformations on constraints in order to accomplish model counting. In addition, we demonstrated its usefulness on a case study regarding quantitative information flow analysis. It is our hope that MCBAT can be used by other researchers in applications requiring model counting for array constraints.

There are many avenues for future work. Extending our approach to higher dimensional arrays would increase the expressiveness of MCBAT, as would handling arrays of types other than integers. In addition, we would like to allow for reasoning over arrays of symbolic lengths. Finally, as arrays can be used to model vectors, hash maps, memory accesses, heaps, and so on, we model counting for arrays is a first step toward building practical tools for model counting for these more complex data structures.

In performing this research, we observed that in many works on model counting the fundamental insights are to (1) convert elements of satisfiability checking procedure into a model counting procedure or (2) convert elements a model enumeration procedure into a model counting procedure. This is the case with model counting algorithms that are based on DPLL, automata, and generating functions. The challenging question arises: to what degree can satisfiability checking and model enumeration algorithms be converted, perhaps automatically, into counting algorithms?

To conclude, we note that Satisfiability Modulo Theories has dramatically increased the ability to perform program analyses. SMT solvers combine decision procedures for Boolean combinations of constraints from various theories. Because there are model counting algorithms for Boolean formulas, linear integer arithmetic, strings, and now integer arrays, we look forward to a future in which Model Counting Modulo Theories combines model counting procedures for various theories to become the fundamental enabling technology behind quantitative program analysis.

REFERENCES

[1] Abdulbaki Aydin, Lucas Bang, and Tevfik Bultan. 2015. Automata-Based Model Counting for String Constraints. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. 255–272.

[2] Abdulbaki Aydin, William Eiers, Lucas Bang, Tegan Brennan, Miroslav Gavrilov, Tevfik Bultan, and Fang Yu. 2018. Parameterized model counting for string and numeric constraints. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. 400–410.

[3] Lucas Bang, Nicolás Rosner, and Tevfik Bultan. 2018. Online Synthesis of Adaptive Side-Channel Attacks Based On Noisy Observations. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. 307–322.

[4] Vaishak Belle. 2017. Weighted Model Counting With Function Symbols. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*.

[5] Elazar Birnbaum and Eliezer L. Lozinskii. 1999. The Good Old Davis-Putnam Procedure Helps Counting Models. *J. Artif. Int. Res.* 10, 1 (June 1999), 457–477.

[6] Mateus Borges, Quoc-Sang Phan, Antonio Filieri, and Corina S. Păsăreanu. 2017. Model-Counting Approaches for Nonlinear Numerical Constraints. In *NASA Formal Methods*, Clark Barrett, Misty Davies, and Temesghen Kahsai (Eds.). Springer International Publishing, Cham, 131–138.

[7] Supratik Chakraborty, Kuldeep Meel, Rakesh Mistry, and Moshe Vardi. 2015. Approximate Probabilistic Inference via Word-Level Counting. (11 2015).

[8] Mark Chavira and Adnan Darwiche. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172, 6 (2008), 772 – 799.

[9] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*. 337–340.

[10] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2009. Generalized, efficient array decision procedures. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. 45–52.

[11] Rodrigo De Salvo Braz, Ciaran O'Reilly, Vibhav Gogate, and Rina Dechter. 2016. Probabilistic Inference Modulo Theories. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence* (New York, New York, USA) *(IJCAI'16)*. AAAI Press, 3591–3599.

[12] Antonio Filieri, Marcelo F. Frias, Corina S. Pasareanu, and Willem Visser. 2015. Model Counting for Complex Data Structures. In *Model Checking Software - 22nd International Symposium, SPIN 2015, Stellenbosch, South Africa, August 24-26, 2015, Proceedings*. 222–241.

[13] Antonio Filieri, Corina S. Pasareanu, and Willem Visser. 2013. Reliability analysis in symbolic pathfinder. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 622–631.

[14] Aymeric Fromherz, Kasper Søe Luckow, and Corina S. Pasareanu. 2016. Symbolic Arrays in Symbolic PathFinder. *ACM SIGSOFT Software Engineering Notes* 41, 6 (2016), 1–5.

[15] Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. 2012. Probabilistic Symbolic Execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis* (Minneapolis, MN, USA) *(ISSTA 2012)*. ACM, New York, NY, USA, 166–176.

[16] Vladimir Klebanov. 2014. Precise quantitative information flow analysis - a symbolic approach. *Theor. Comput. Sci.* 538 (2014), 124–139.

[17] Daniel Kroening and Ofer Strichman. 2008. *Decision Procedures: An Algorithmic Point of View* (1 ed.). Springer Publishing Company, Incorporated.

[18] Daniel Larraz, Enric Rodríguez-Carbonell, and Albert Rubio. 2013. SMT-Based Array Invariant Generation. In *Verification, Model Checking, and Abstract Interpretation*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 169–188.

[19] Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. 2004. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation* 38, 4 (2004), 1273 – 1302.

[20] Loi Luu, Shweta Shinde, Prateek Saxena, and Brian Demsky. 2014. A Model Counter for Constraints over Unbounded Strings. In *Proceedings of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. ACM, New York, NY, USA, 565–576.

[21] John McCarthy. 1962. Towards a Mathematical Science of Computation. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany,*

August 27 - September 1, 1962. 21–28.

[22] Abtin Molavi, Thomas Schneider, Mara Downing, and Lucas Bang. 2020. MCBAT: Model Counting for Constraints over Bounded Integer Arrays. In *Verified Software. Theories, Tools, and Experiments - 12th International Conference, VSTTE 2020, Los Angeles, CA, USA, July 20-21, 2020, Revised Selected Papers (Lecture Notes in Computer Science)*. Springer.

[23] Quoc-Sang Phan, Pasquale Malacaria, Corina S. Pasareanu, and Marcelo d'Amorim. 2014. Quantifying information leaks using reliability analysis. In *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*. 105–108.

[24] Quentin Plazar, Mathieu Acher, Sébastien Bardin, and Arnaud Gotlieb. 2017. Efficient and Complete FD-solving for extended array constraints. 1231–1238.

[25] William Pugh. 1994. Counting Solutions to Presburger Formulas: How and Why. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation* (Orlando, Florida, USA) *(PLDI '94)*. ACM, New York, NY, USA, 121–134.

[26] Tian Sang, Paul Bearne, and Henry Kautz. 2005. Performing Bayesian Inference by Weighted Model Counting. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1* (Pittsburgh, Pennsylvania) *(AAAI'05)*. AAAI Press, 475–481.

[27] Elena Sherman and Andrew Harris. 2019. Accurate String Constraints Solution Counting with Weighted Automata. In *Proceedings of The 34th IEEE/ACM International Conference on Automated Software Engineering ASE*.

[28] Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. 2017. Model Counting for Recursively-Defined Strings. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. 399–418.

[29] Nestan Tsiskaridze, Lucas Bang, Joseph McMahan, Tevfik Bultan, and Timothy Sherwood. 2018. Information Leakage in Arbiter Protocols. In *Automated Technology for Verification and Analysis - 16th Int. Symp, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*. 404–421.

[30] Sven Verdoolaege, Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. 2007. Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions. *Algorithmica* 48, 1 (March 2007), 37–66.

[31] Willem Visser and Corina S. Pasareanu. 2017. Probabilistic programming for Java using symbolic execution and model counting. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists, SAICSIT 2017, Thaba Nchu, South Africa, September 26-28, 2017*. 35:1–35:10.