SymMC: Approximate Model Enumeration and Counting Using Symmetry Information for Alloy Specifications

Wenxi Wang
The University of Texas at Austin
Austin, Texas, USA
wenxiw@utexas.edu

Yang Hu
The University of Texas at Austin
Austin, Texas, USA
huyang@utexas.edu

Kenneth L. McMillan
The University of Texas at Austin
Austin, Texas, USA
kenmcm@cs.utexas.edu

Sarfraz Khurshid
The University of Texas at Austin
Austin, Texas, USA
khurshid@ece.utexas.edu

ABSTRACT

Specifying and analyzing critical properties of software systems plays an important role in the development of reliable systems. Alloy is a mature tool-set that provides a first-order relational logic for writing specifications, and a fully automatic powerful backend for analyzing the specifications. It has been widely applied in areas including verification, security, and synthesis.

Symmetry breaking is a useful approach for pruning the search space to efficiently check the satisfiability of combinatorial problems. As the backend solver of Alloy, Kodkod does the partial symmetry breaking (PaSB) for Alloy specifications. While full symmetry breaking remains challenging to scale, a recent study showed that Kodkod PaSB could significantly reduce the model counting time, albeit at the cost of producing only partial model counts. However, the desired term is either the *isomorphic* count under *no* symmetry breaking, or the *non-isomorphic* models/count under *full* symmetry breaking. This paper presents an approach called SymMC, which utilizes the symmetry information to compute all the desired terms for Alloy specifications. To make SymMC scalable, we propose approximate algorithms based on sampling to estimate the desired terms. We show that our proposed estimators have consistency and upper bound properties. To our knowledge, SymMC is the first approach that automatically approximates non-isomorphic model enumeration/counting for Alloy specifications. Thanks to the non-isomorphic model counting, SymMC also provides the first automatic quantification measurement on the solution space pruning ability of Kodkod PaSB. Furthermore, empirical evaluations show that SymMC provides a competitive isomorphic counting approach for Alloy specifications compared to the state-of-the-art model counters.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3549161

CCS CONCEPTS

• **Software and its engineering → Software verification**; • **Theory of computation → Automated reasoning**.

KEYWORDS

Symmetry Breaking, Permutation Sampling, Alloy specifications

ACM Reference Format:
Wenxi Wang, Yang Hu, Kenneth L. McMillan, and Sarfraz Khurshid. 2022. SymMC: Approximate Model Enumeration and Counting Using Symmetry Information for Alloy Specifications. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549161

1 INTRODUCTION

Alloy [27] is a specification language combining first-order logic with relational algebra. Alloy and its analyzer have been widely used for complex system modeling in a variety of fields such as security [2, 56], system analysis and verification [8, 28, 43], and design [3, 6, 7]. In the backend, Alloy analyzer is supported by a highly optimized constraint solver called Kodkod [54], which does symmetry breaking at the problem domain level and efficiently translates Alloy specifications into SAT formulas. To check the satisfiability of Alloy specifications, Alloy analyzer calls its off-the-shelf SAT solvers to solve the SAT formulas.

A symmetry is a *permutation* of atoms in a problem's universe that takes models of the problem to other models, and non-models to other non-models. Kodkod performs *static* symmetry breaking where the symmetry breaking predicates (SBPs) [13] are pregenerated and added to the original SAT formula. Unfortunately, generating a *full* symmetry breaking predicate to make exactly one representative per isomorphism class is NP-complete [13]. Regarding this, Kodkod generates a *partial* SBP which is true of at least one (typically more than one) representative per isomorphism class.

This paper focuses on addressing three challenging problems for Alloy specifications by exploiting the Kodkod PaSB.

Non-Isomorphic Model Enumeration Non-isomorphic models are often desirable because they amount to a significant computational save without reducing the effectiveness of achieving the goal. For example, non-isomorphic test inputs save a significant amount of time to test the program without reducing the code coverage. A

lot of work has been done to efficiently enumerate non-isomorphic models in various kinds of areas such as test generation [32, 42, 59], configurations of robotics [11, 34], and graphs [37, 41, 55]. To our knowledge, there is only one existing tool called TestEra [29, 31, 35] that utilizes Alloy to generate non-isomorphic test inputs for Java programs. However, this approach has one limitation: it requires users to manually add the symmetry breaking specification to realize the full symmetry breaking for their Alloy specifications. Motivated by this, we aim to propose a fully automated approach to enumerate non-isomorphic models for Alloy specifications without requiring manual input from users.

Quantification Measurement of Kodkod Partial SBPs Kodkod partial SBPs are often effective—they are often able to rule out a large fraction of models from each isomorphism class. However, the pruning ability of Kodkod partial SBPs is unpredictable, which means that the partial SBPs can result in totally different ruled-out fractions for different input problems. Shlyakhter et al. did some initial measurements for the effectiveness of the partial SBPs applied on only a few examples with the known number of isomorphism classes [48]. However, to our knowledge, no work has been done to automatically measure the effectiveness of Kodkod partial SBPs for an arbitrary Alloy specification due to the difficulty of counting the number of isomorphism classes of the specification. Regarding this, we aim to automatically measure the pruning ability of the applied Kodkod partial SBPs for an arbitrary Alloy specification, by getting the non-isomorphic count for the specification.

Isomorphic Model Counting Model counting is a classical problem of computing the number of models for a given formula. *Projected model counting* [4] is a kind of model counting problem which counts only the unique models with respect to designated variables. Consider a simple SAT formula $(x_1 \vee x_2) \wedge (x_3 \vee \neg x_4)$, the projected model count over variables x_1 and x_2 is 3; over variables x_1 and x_4 is 4. There have been several recent works in doing model counting for Alloy specifications [57, 60, 62]. The model counting for Alloy specifications belongs to projected model counting, which counts the number of satisfiable models of the translated SAT formulas over only primary variables (i.e., the variables directly encoding Alloy specification components) excluding auxiliary variables introduced during the translation. A recently published tool called AlloyMC [62] has added off-the-shelf projected model counters to the Alloy backend. A recent study [60] found that Kodkod partial SBPs can substantially reduce the counting time taken by the-state-of-art model counters for counting satisfiable models of Alloy specifications. However, the addition of partial SBPs means that the reported counts are accurate only with respect to *partial* symmetry breaking (PaSB). Indeed, it is the isomorphic count (i.e., the model count with no symmetry breaking) that is commonly desirable. Inspired by the findings of the study, we aim to efficiently get the isomorphic count for Alloy specifications by utilizing the Kodkod partial SBPs.

We propose an automated tool called SymMC to solve all these three challenging problems by solving two key technical problems: the non-isomorphic model enumeration/counting and the isomorphic model counting for Alloy specifications. The two technical problems are in general hard problems—they would quickly become intractable as the number of all possible permutations in the input problem increases substantially. Regarding this, the core idea inside SymMC is to efficiently approximate the non-isomorphic models and the isomorphic model count, by *sampling* the permutations instead of considering all possible permutations.

We first convert the non-isomorphic model enumeration problem into a graph theory problem. The non-isomorphic models can be over-approximated through the *weakly connected components* of the converted graph which are constructed by the randomly sampled permutations. Based on the formulation, we propose our non-isomorphic model estimator and show that the estimator is able to provide high approximation accuracy. The isomorphic counting estimator is built upon the non-isomorphic model estimator. We formulate the isomorphic counting problem into a statistical inference problem. Based on the formulation, we then propose our isomorphic counting estimator based on the simple random sampling. We prove that both estimators provide the upper bound/over-approximation of the count/enumeration and have consistency property. Finally, we present two practical approximate algorithms of SymMC for realizing the two estimators, respectively.

We evaluate SymMC mainly in two aspects: the approximation efficiency and the approximation accuracy. To do the empirical evaluation, we collect 110 Alloy specifications as our subjects from four sources relating to a variety of real-world applications such as security, protocols and test generation. For non-isomorphic model approximation, experimental results show that SymMC successfully solves 73 subjects within the standard time limit of 5,000 seconds; for all those subjects, SymMC approximates with surprisingly zero error rate. For quantification measurement of the Kodkod partial SBP, SymMC found that its pruning ability is often effective and even perfect in some subject types (e.g., n-Queen problems), while its ability is sometimes very limited in other subjects (e.g., singly linked list data structure). For isomorphic model counting, experimental results show that SymMC is more efficient than the state-of-the-art exact counter GANAK [47] in 77.2% subjects and the state-of-the-art approximate counter ApproxMC [51] in 79.5% subjects; SymMC approximates with lower error rate than ApproxMC in 95.6% subjects. The source code of SymMC is publicly available at https://github.com/wenxiwang/SymMC-Tool.

The contributions of this paper are:

- *Idea.* We introduce the idea of utilizing symmetry information and permutation sampling to approximate the non-isomorphic model enumeration and isomorphic counting for Alloy specifications.
- *Approximate Non-isomorphic Enumeration.* To our knowledge, SymMC is the first to do automatic non-isomorphic model enumeration for Alloy specifications.
- *Quantification Measurement.* SymMC provides an automatic way of measuring the pruning ability of the Kodkod partial SBPs for an arbitrary Alloy specification.
- *Approximate Isomorphic Counting.* We present an approximate isomorphic counting algorithm based on sampling which is competitive with state-of-the-art counters.

2 RELATED WORK

For non-isomorphic model enumeration for Alloy specifications, the most related tool is TestEra [29, 31, 35] which applies Alloy as

```
1. sig Node { link: one Node }
2. pred Cyclic {
3.   all n: Node | n.^link = Node
4. }
5. run Cyclic for 5 Node
```

Figure 1: The Alloy specification of the cyclic linked list.

its backend to enumerate non-isomorphic test inputs for Java programs with limited data structure types. TestEra requires users to manually add domain-specific specifications as symmetry breaking constraints to eliminate all the isomorphic models. It is shown in the recent study [60] that the manually added domain-specific symmetry breaking constraints proposed by TestEra for six basic data structures are able to do efficient full symmetry breaking. However, to fully break the symmetries for even the classic data structures (e.g., linked lists and binary search trees), it took the authors (i.e., the experts) a few hours to write the specification [30]. Not to mention all kinds of complex domain-specific problems defined by Alloy users. In general, the limitation of this approach is obvious—it requires users to have enough domain knowledge and expert knowledge in symmetry breaking to be able to manually write the symmetry breaking constraints for their own domain-specific problems. In contrast, by exploiting the symmetry information of the specifications, SymMC is able to *automatically* approximate the non-isomorphic models for *arbitrary* Alloy specifications, without requiring any manual effort from the user side.

There are two recent works which consider doing model counting with Alloy. Wang et al. in the study [60] of symmetry breaking and model counting found that counting under Alloy partial symmetry breaking is much faster than counting under no symmetry breaking, which shows a promising direction to improve the model counting efficiency. However, using partial symmetry breaking predicates creates a different counting problem, thus could introduce bias to the original counting problem. Unfortunately, the study did not attempt to address this issue. Inspired by the findings of the study, SymMC aims to efficiently approximate the counts both under no symmetry breaking (isomorphic counting) and under full symmetry breaking (non-isomorphic counting) by enumerating models under partial symmetry breaking. AlloyMC [62] is a tool implementation which adds off-the-shelf model counters to the Alloy backend and provides a GUI which invokes these counters to get the count of the input Alloy specification under no symmetry breaking or partial symmetry breaking. In essence, AlloyMC simply applies off-the-shelf model counters on Alloy specifications, by extending the original Alloy grammars and GUI. It neither considers symmetry breaking during counting, nor is able to count the non-isomorphic models.

Much work has been done in approximate model counting with sampling [9, 17, 21–25, 33, 50, 51, 61]. A significant amount of research has been attempted in utilizing symmetry breaking in SAT and constraint solving [5, 14, 15, 18–20, 38–40, 44, 46]. However, only a few works focus on applying symmetry in the area of model counting. Besides the recent study [60] by Wang et al. as discussed above, we are only aware of one very recent paper by Bremen et al. [58], which is closely related to our paper. The authors exploited the inherent symmetry exhibited in combinatorial problems for component caching-based model counters. Their approach is implemented in a propositional counter called SymGanak. There are

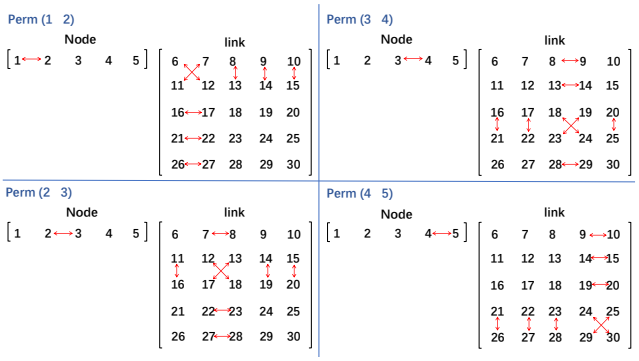


Figure 2: Permutations of the cyclic linked list specification explicitly eliminated by Kodkod symmetry breaking.

two main differences between SymMC and SymGanak: 1) SymMC presents a specialized counting approach for Alloy specifications which can do both non-isomorphic model enumeration/counting and isomorphic counting, while SymGanak can only do the isomorphic counting for propositional formulas; 2) SymGanak exploits symmetry dynamically among the components encountered during counting, while SymMC dose its counting based on the static symmetry info. In addition, the current implementation of SymGanak does not support projected model counting[1].

3 BACKGROUND

We introduce the basics of Alloy specifications and Kodkod partial symmetry breaking for Alloy specifications, using a simple illustrative example in Figure 1.

3.1 Alloy Specifications

An Alloy specification usually consists of three key components: *signature declarations* which define sets or relations, *constraint paragraphs* which define the constraints over the signatures, and *commands* which provide instructions for the Alloy analyzer to carry out various analyses. Figure 1 shows a specification example which aims to generate a cyclic linked list. A signature (keyword `sig`) named `Node` represents a set of atoms (line 1). In the body of the signature declaration, the field named `link` represents a binary relation over the `Node` set (i.e., $link \subseteq Node \times Node$). Here, the keyword `one` means that for any atom $n \in Node$, there exists *exactly* one atom $n' \in Node$ satisfying that $(n, n') \in link$. A predicate `Cyclic` defines a constraint saying that all nodes are reachable from every node `n` (keyword `all`) following one or more traversals (symbol `^`) along with the `link` (line 2-4). The `run` command requests the Alloy analyzer to search for an instance (i.e., a model) of the predicate `Cyclic`. Here, `5 Node` means that the `Node` set contains at most 5 atoms. Given a specification, the Alloy analyzer invokes the backend constraint solver called Kodkod [54] to translate it into a SAT formula. The formula is then solved by an off-the-shelf SAT solver to check the satisfiablity of the predicate constraint. For more details of Alloy, please refer to the book [27].

[1]Please refer to the discussion at: https://github.com/meelgroup/ganak/issues/14

3.2 Kodkod Partial Symmetry Breaking for Alloy Specifications

As a backend solver of Alloy, Kodkod [54] is an efficient constraint solver, which supports static partial symmetry breaking at the problem domain level [49, 53].

Kodkod Translation on Relations When Kodkod translates an Alloy specification into a SAT formula, it treats every relation as a matrix of Boolean variables [26]. The dimension of the matrix equals to the relation's arity. A signature (i.e., a set of atoms) is specially viewed as a unary relation and is represented by a vector. The vector and the matrix representing the `Node` and `link` relations respectively in the illustrative example are shown in Figure 2 (each Boolean variable is indexed with a unique number). Since the `run` command defines that at most 5 atoms are in the `Node` relation, the `Node` relation is represented by a vector with length 5; the `link` relation which maps `Node` to `Node` is represented by a 5×5 matrix.

Symmetry Type, Permutations and Transpositions Kodkod symmetry detection for Alloy specifications is straightforward: the symmetry of Alloy specifications happens in each declared signature that can contain more than one atom. In Kodkod, such signature is called a *symmetry type*. In the illustrative example, the `Node` signature is a symmetry type. The interchange of any atoms within symmetry types is taken as one *permutation*. Figure 2 shows four permutations of the illustrative example. The top left of Figure 2 shows one permutation which interchanges atom 1 and atom 2 in the symmetry type `Node`; the variables in row 1 and row 2, and column 1 and column 2 of the `link` relation matrix are also exchanged correspondingly (as indicated by the red arrows), since the `link` relation has the symmetry type `Node` in both dimensions. A permutation can be represented as a set of *transpositions* (i.e., permutations which only exchange two atoms and keep all others fixed). For example, all the permutations shown in Figure 2 are actually transpositions.

Partial Symmetry Breaking Kodkod performs partial symmetry breaking using the *lexicographical order* (lex-order) predicate [53]. For each symmetry type, Kodkod defines a lex-order predicate over all its atoms. The predicates over atoms are then transformed into the lex-order predicates over the variables in relation matrices which contain the symmetry type. Figure 2 shows all four permutations of the example that are explicitly eliminated by Kodkod defined lex-order predicates. To eliminate the permutation $(1, 2)$ shown in the top left of Figure 2, Kodkod defines the lex-order predicate over atoms that atom `1` is *lex-smaller* (denoted as $\preceq_{lex}$) than atom `2` in the symmetry type `Node`. This defines the lex-order of the variables in the relation matrix `link`: variables in row 1 are lex-smaller than the corresponding variables in row 2, which also applies to column 1 and column 2. Thus, the lex-order predicate over atoms ($atom\ 1 \preceq_{lex} atom\ 2$) is equivalent to the lex-order predicate over variables in relation matrices: $(1 \preceq_{lex} 2) \wedge (6 \preceq_{lex} 12) \wedge (7 \preceq_{lex} 11) \wedge (8 \preceq_{lex} 13) \wedge (9 \preceq_{lex} 14) \wedge (10 \preceq_{lex} 15) \wedge (16 \preceq_{lex} 17) \wedge (21 \preceq_{lex} 22) \wedge (26 \preceq_{lex} 27)$. This way, by transforming the predicate over atoms: (atom 1 $\preceq_{lex}$ atom 2) $\wedge$ (atom 2 $\preceq_{lex}$ atom 3) $\wedge$ (atom 3 $\preceq_{lex}$ atom 4) $\wedge$ (atom 4 $\preceq_{lex}$ atom 5) into the predicate over variables, the Kodkod symmetry breaking predicate is created, with which all four permutations in Figure 2 are explicitly eliminated.

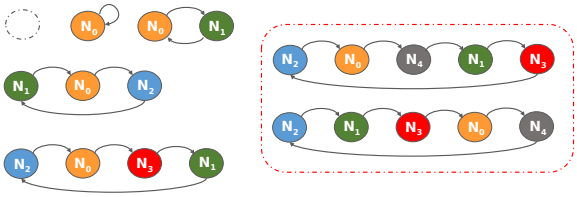


Figure 3: All models of the illustrative example under Kodkod partial symmetry breaking; the dashed circle represents an empty linked list.

Figure 4: Applying the transposition list of permutation (1, 2) on model m_1 to obtain the permuted model m_2. For each model, the values of 30 primary variables are shown.

As shown in the example, Kodkod only explicitly considers eliminating a linear number of permutations in each symmetry type, *which might implicitly eliminate other permutations.* Although efficient, there is no guarantee to break all symmetries. As shown in Figure 3, two satisfying models under the defined SBP are isomorphic to each other (circled by the red dashed line). *In sum, Kodkod's partial symmetry breaking is based on heuristics, and there is no characterization of what portion of permutations is eliminated by the defined SBP.*

4 SYMMC

Given that the pruning ability of Kodkod SBPs is unpredictable, instead of only considering the permutations that are explicitly eliminated by the Kodkod SBPs, SymMC enhances Kodkod to extract the symmetry information which is able to generate *all* possible permutations. It then samples permutations which are usually much larger than the permutations explicitly eliminated by Kodkod SBPs, using well-designed algorithms with theoretical guarantees to approximate the non-isomorphic model set/count and the isomporhic model count.

4.1 Overview

The overview of SymMC is shown in Figure 5. The input of SymMC is an arbitrary Alloy specification. SymMC has three key functionalities which correspond to three outputs: the non-isomorphic models/count of the specification, the isomorphic model count of the specification, and the quantification metric in evaluating the pruning ability of the applied Kodkod partial SBP. In order to realize these functionalities, SymMC consists of three modules: 1) enhanced Kodkod which not only encodes the Alloy specification under PaSB into a SAT formula (with primary variables indicated), but also extracts the symmetry info of the specification; 2) the all-satisfiable model enumerator which generates all the satisfying models projected over primary variables under PaSB; 3) the estimator module including the two estimators for approximating the non-isomorphic model set/count and the isomorphic model count, respectively. The quantification metric is the by-product of the non-isomorphic estimator. The following subsections introduce these three modules in detail.

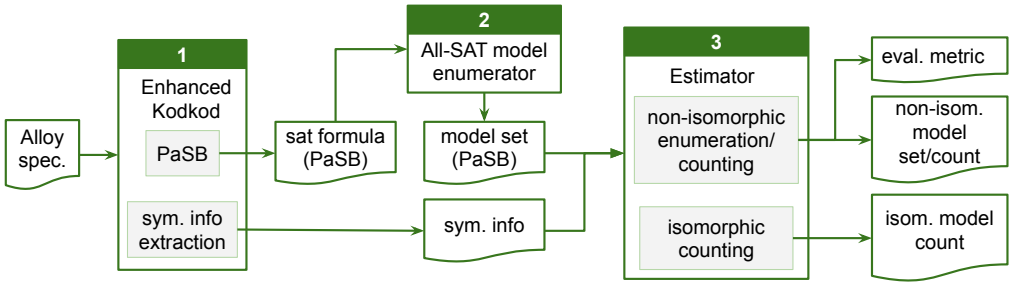


Figure 5: SymMC overview

4.2 Enhanced Kodkod with Symmetry Info Extraction

The enhanced Kodkod module can not only perform partial symmetry breaking but also extract the symmetry info of the input specification, with which all possible permuted models of a given model can be efficiently generated. To illustrate why and how we enhance Kodkod for symmetry info extraction, we first present how one permuted model of a given model is created with the symmetry info available from the vanilla Kodkod; we then show that extra symmetry info is needed for efficient permuted model generation.

With our illustrative example, we explain how the permuted model m_2 of a given model m_1 is obtained under permutation (1, 2), as shown in Figure 4. As mentioned above, the detailed info of the permutation (1, 2) w.r.t. the symmetry type `Node` is a transposition list of the corresponding Boolean variables in `Node` and `link`: (1, 2), (6, 12), (7, 11), (8, 13), (9, 14), (10, 15), (16, 17), (21, 22) and (26, 27). For the model m_1, values of the Boolean variables in each transposition in the list are exchanged to obtain the permuted model m_2 under the permutation (1, 2).

As known, every permutation can be decomposed as products of a *linear* number of transpositions of any two atoms, while every permutation can be decomposed as products of a *quadratic* number of *nearest-neighbor* transpositions (i.e., the transpositions of two atoms which are next to each other). As illustrated in Section 3, Kodkod only considers nearest-neighbor transpositions for each symmetry type (e.g., (1, 2), (2, 3), (3, 4) and (4, 5) in our example). To get a simple decomposition of any permutation in the example, besides the nearest-neighbor transpositions, we also need the info (i.e., the transposition list of the corresponding Boolean variables) of other transpositions (e.g., (1, 3), (1, 4), (1, 5), (2, 4), (2, 5), and (3, 5) in our example). In sum, to efficiently generate any permuted model w.r.t. one symmetry type, we need the info of transpositions of *any* two atoms in that type. Thus, we enhance Kodkod to extract the info of all possible transpositions for each symmetry type. A permuted model w.r.t. *all* symmetry types $A_1, \ldots A_n$ is obtained with the combination of the permutation in each type, thus the number of all permutations is $\prod_{i=1}^{n} |A_i|!$, which is usually a huge number. Therefore, the naive non-isomorphic model enumeration/counting approach as well as the naive isomorphic counting approach could quickly fail when the number increases substantially. In Section 4.4 and Section 4.5, we will introduce how SymMC utilizes permutation sampling to make it scalable.

4.3 The AllSAT Model Enumerator

Generating all satisfiable models (AllSAT for short) is a variant of the propositional satisfiability problem. According to the recent survey [52], AllSAT solvers can be classified into three categories: blocking clause-based, chronological backtracking-based and Binary Decision Diagram (BDD)-based. The blocking clause-based AllSAT solver is the most typical and easiest to implement. It iteratively computes satisfying models using a traditional Boolean satisfiability (SAT) solver and adds blocking clauses which are the complement of the already enumerated models. In the current implementation of SymMC, we build a blocking clause based ALLSAT solver on top of a classic SAT solver called MiniSat-2.2.0 [16] to enumerate projected models over primary variables for the SAT formula encoding the input Alloy specification under PaSB.

4.4 The Non-Isomorphic Model Estimator

One key functionality of SymMC is to approximately enumerate/count the non-isomorphic models for the input Alloy specification, with its model set under PaSB (denoted as $\mathcal{M}_P$) produced by All-SAT enumerator and the extracted symmetry info (i.e., the symmetry information containing the transpositions of any two atoms; denoted as *Sym*) produced by enhanced Kodkod module. We first formulate the non-isomorphic model enumeration/counting problem as a graph theory problem. We then propose our estimator and show that the estimator is able to provide high approximation accuracy thanks to the problem formulation. Finally, we present a practical approximate model enumeration/counting algorithm realizing the estimator.

4.4.1 *The Estimator.*

Problem Definition 1. *Let $\mathcal{E}$ be the set of all possible permutations. Let $f(m, e)$ be a function which outputs a permuted model by applying a permutation $e \in \mathcal{E}$ to a model $m \in \mathcal{M}_P$. Let $=_F$ be an equivalence relation on $\mathcal{M}_P$ such that $m =_F m'$ iff $\exists e \in \mathcal{E}. f(m, e) = m'$. The goal is to count the number of equivalence classes defined by $=_F$, which is the non-isomorphic model count C_F; and to select one model from each equivalence class to create a non-isomorphic model set $\mathcal{M}_F$.*

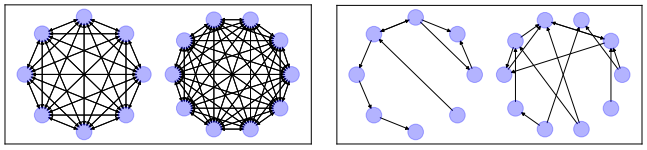
The non-isomorphic model enumeration/counting problem can be converted into a *graph theory* problem defined as follows.

Problem Definition 2. *The equivalence relation $=_F$ on $\mathcal{M}_P$ can be represented as a directed graph $G(\mathcal{M}_P, =_F)$, where each node represents a model in $\mathcal{M}_P$, and each edge from the node of $m \in \mathcal{M}_P$ to*

Algorithm 1 SymMC approximate non-isomorphic model enumeration and counting.

Input: all the models under PaSB $\mathcal{M}_P$; symmetry info Sym produced by enhanced Kodkod module.
Output: the non-isomorphic model set $\hat{\mathcal{M}}_F$, count $\hat{C}_F$, and the sampled permutation set $\hat{\mathcal{E}}$.
Parameters: initial sample ratio d; sample growth rate r; the early stopping criterion θ; numbers of all perm $totalperm$

1: **procedure** APPROXNONISOM($\mathcal{M}_P$, Sym)
2: $\hat{\mathcal{E}} \leftarrow \emptyset$
3: $\hat{=}_F \leftarrow \emptyset$
4: $\hat{\mathcal{M}}_F \leftarrow \mathcal{M}_P$
5: $sampsize \leftarrow setSampSize(totalperm, d)$
6: **do**
7: $b \leftarrow |\hat{\mathcal{M}}_F|$ ▷ back up the size of $\hat{\mathcal{M}}_F$.
8: $\Delta \leftarrow sampPerms(Sym, sampsize, \hat{\mathcal{E}})$
9: **for each** $m \in \mathcal{M}_P$ **do**
10: $\mathcal{M}_\Delta \leftarrow applyPerms(m, \Delta, \mathcal{M}_P)$
11: $\hat{=}_F \leftarrow \hat{=}_F \cup (\{m\} \times \mathcal{M}_\Delta)$
12: **end for**
13: $\hat{\mathcal{M}}_F \leftarrow wcc(G(\mathcal{M}_P, \hat{=}_F))$
14: $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \Delta$
15: $sampsize \leftarrow setSampSize(totalperm, r)$
16: **while** $|\hat{\mathcal{E}}| < totalperm \wedge b - |\hat{\mathcal{M}}_F| > \theta$
17: $\hat{C}_F \leftarrow |\hat{\mathcal{M}}_F|$
18: **return** $\hat{\mathcal{M}}_F, \hat{C}_F, \hat{\mathcal{E}}$
19: **end procedure**



(a) The graph $G(\mathcal{M}_P, =_F)$. **(b) The graph $G(\mathcal{M}_P, \hat{=}_F)$.**

Figure 6: An intuitive illustration of why our non-isomorphic model estimator is able to provide high accuracy. (a) shows two components of the graph, each of which is fully connected; (b) shows that 85% random edge deletion of the graph still preserves its weak connectivity.

the node of $m' \in \mathcal{M}_P$ represents $m =_F m'$. The non-isomorphic count C_F is equal to the number of weakly connected components in the graph. One node in each weakly connected component is arbitrarily selected, and the models represented by these nodes constitute a non-isomorphic model $\mathcal{M}_F$.

THEOREM 1. *Let $\hat{=}_F$ be a relation on $\mathcal{M}_P$, such that $m \hat{=}_F m'$ iff $\exists e \in \hat{\mathcal{E}}. f(m, e) = m'$, where $\hat{\mathcal{E}}$ is a sample of $\mathcal{E}$. The relation $\hat{=}_F$ on $\mathcal{M}_P$ can be represented by a directed graph $G(\mathcal{M}_P, \hat{=}_F)$. Let $\hat{C}_F$ be the number of weakly connected components in the graph $G(\mathcal{M}_P, \hat{=}_F)$. One node in each weakly connected component of $G(\mathcal{M}_P, \hat{=}_F)$ is arbitrarily selected; and let $\hat{\mathcal{M}}_F$ be the set of models represented by the selected nodes. $\hat{C}_F$ is the upper bound of C_F; and $\hat{\mathcal{M}}_F$ is the over-approximation of $\mathcal{M}_F$, meaning that for any model $m \in \mathcal{M}_F$, there exists a model $m' \in \hat{\mathcal{M}}_F$ such that $m =_F m'$.*

PROOF. According to the definitions of $=_F$ and $\hat{=}_F$, if $m \hat{=}_F m'$ then $m =_F m'$. Therefore, all edges in $G(\mathcal{M}_P, \hat{=}_F)$ are also in $G(\mathcal{M}_P, =_F)$. Thus, the weakly connected components in $G(\mathcal{M}_P, =_F)$ are no more than those in $G(\mathcal{M}_P, \hat{=}_F)$, thus $\hat{C}_F \geq C_F$.

We prove $\hat{\mathcal{M}}_F$ is the over-approximation of $\mathcal{M}_F$ by contradiction. Assume $\hat{\mathcal{M}}_F$ is not the over-approximation of $\mathcal{M}_F$, meaning that there exists a model $m \in \mathcal{M}_F$, for any model $m' \in \hat{\mathcal{M}}_F$ such that $m \neq_F m'$. Therefore, if $m =_F m'$ then $m' \notin \hat{\mathcal{M}}_F$, which means $\hat{\mathcal{M}}_F$ does not include any models in the equivalence class of m w.r.t. $=_F$. This contradicts to the definition of $\hat{\mathcal{M}}_F$. □

Constructing the graph $G(\mathcal{M}_P, \hat{=}_F)$ is equivalent to deleting the edges of the graph $G(\mathcal{M}_P, =_F)$ corresponding to the unsampled permutations.

THEOREM 2. *$\hat{\mathcal{M}}_F \xrightarrow{=_F} \mathcal{M}_F$ and $\hat{C}_F \to C_F$ when $\hat{\mathcal{E}} \to \mathcal{E}$. Here, $\xrightarrow{=_F}$ refers to the convergence w.r.t. $=_F$.*

PROOF. According to the definitions of $=_F$ and $\hat{=}_F$, if $\hat{\mathcal{E}} \to \mathcal{E}$, then $\hat{=}_F \to =_F$. Therefore, we have $\hat{\mathcal{M}}_F \xrightarrow{=_F} \mathcal{M}_F$ and $\hat{C}_F \to C_F$. □

In sum, we use $\hat{C}_F$ as the estimator of C_F, which provides the upper bound of C_F; and $\hat{\mathcal{M}}_F$ as the estimator of $\mathcal{M}_F$, which provides the over-approximation of $\mathcal{M}_F$. As shown in Theorem 2, they both have the consistency property. It is important to note that the consistency property can be achieved when $|\hat{\mathcal{E}}|$ is much smaller than $|\mathcal{E}|$. Because each weakly connected component w.r.t. $=_F$ is in fact fully connected (i.e., every two nodes in the component are directly and mutually connected), for which randomly deleting a large number of edges might still preserve the weak connectivity. This explains SymMC surprisingly high estimation accuracy for non-isomorphic model enumeration and counting, as shown in Section 5. Figure 6 shows a conceptual example to illustrate our intuition, where after randomly deleting 85% of the edges in a fully connected graph with two components, its weak connectivity still preserves.

4.4.2 The Algorithm. The practical algorithm based on the proposed estimators for approximating the non-isomorphic models or count is presented in Algorithm 1. Initially, the permutation sample set $\hat{\mathcal{E}}$ is set to empty (line 2), and the graph $G(\mathcal{M}_P, \hat{=}_F)$ is constructed with only nodes representing the models under PaSB and no edges (line 3-4). In addition, the sample size is initialized based on the total number of permutations and the initial sample ratio, using the *setSampSize* function, which is shown in Algorithm 2 (line 5).

The general idea of the algorithm is to incrementally add edges to the graph based on each round of sampled permutations and update the estimated non-isomorphic model set and count iteratively. To do so, in each round, a set of *new* permutations Δ is randomly sampled from the total permutation set $\mathcal{E}$ *without replacement* (line 8); Δ is then used to generate permuted models in $\mathcal{M}_P$ (line 10) and add new edges to the graph (line 11); The Union-Find algorithm

Algorithm 2 SymMC permutation sample size setting.

Input: perms number *totalperm*; the sampling rate *ratio*.
Output: the sample size *sampsize*
Parameters: sample size range [*minperm*, *maxperm*];

```
1:  procedure setSampSize(totalperm, ratio)
2:      if totalperm ≤ minperm then
3:          return totalperm
4:      end if
5:      α ← totalperm · ratio
6:      if α < minperm then
7:          return minperm
8:      end if
9:      if α ≤ maxperm then
10:         return α
11:     end if
12:     return maxperm                          ▷ α > maxperm
13: end procedure
```

[12] is then performed on the graph $G(\mathcal{M}_P, \hat{=}_F)$ to find its weakly connected components and construct $\hat{\mathcal{M}}_F$ (line 13)[2]; The sampled permutation set $\hat{\mathcal{E}}$ gets updated with the set of new permutations Δ (line 14). As the number of added edges grows, the number of weakly connected components decreases, so as the estimated count. The approximation process continues until the decrease of the estimated count is not significant under the early stopping criterion θ or all permutations have been sampled (line 16).

Algorithm 2 shows the details of how we set the sample size given the total permutation number *totalperm* and the sampling rate *ratio*. The idea is that if the total permutation number is smaller than the threshold *minperm*, we utilize all the permutations without doing the sampling (lines 2-4); otherwise, we first set the sample size based on the sampling rate and normalize it within a parameterized range [*minperm*, *maxperm*] (lines 5-12).

4.4.3 The Quantification Metric. There is a by-product of the non-isomorphic model approximation, which is the quantification metric for approximately evaluating the pruning ability of the applied Kodkod partial symmetry breaking predicate. Inspired by Shlyakhter [48], we define the metric as the ratio of non-isomorphic count to the count under PaSB: $\frac{\hat{C}_F}{|\mathcal{M}_P|}$. The range of the metric is (0,1]. Larger value indicates higher pruning ability. The value of 1 indicates that the Kodkod partial SBP is actually doing full symmetry breaking. Note that since $\hat{C}_F$ is the upper bound of C_F, the metric is an optimistic estimation in evaluating the pruning ability.

4.5 The Isomorphic Count Estimator

The isomorphic count estimator takes $\mathcal{M}_P$ and *Sym* as inputs. It first utilizes the non-isomorphic model estimator to generate the estimated non-isomorphic model set $\hat{\mathcal{M}}_F$. Based on $\hat{\mathcal{M}}_F$, the isomorphic counting problem is formulated as a statistical inference problem, where the desired isomorphic count denoted by C_N is estimated using *simple random sampling* techniques [45] (Chapter 7.3, Page 202-220). We show that the expectation of the estimator provides the upper bound of the isomorphic count and has consistency property.

4.5.1 The Estimator.

Problem Definition 3. *Let $=_N$ be an equivalence relation on $\mathcal{M}_F \times \mathcal{E}$ such that $(m,e) =_N (m',e')$ iff $f(m,e) = f(m',e')$. Our goal is to calculate the number of equivalence classes defined by $=_N$, which is the count C_N.*

Lemma 1. *Let $\mathcal{I} : \mathcal{E} \mapsto \{1,\dots,|\mathcal{E}|\}$ be a permutation ordering function which returns the index of a permutation in $\mathcal{E}$ based on a predefined ordering of the permutations in $\mathcal{E}$. Let $\mathcal{G}_N : \mathcal{M}_F \times \mathcal{E} \mapsto \{0,1\}$ be a global labeling function over all model-permutation pairs:*

$$\mathcal{G}_N(m,e) = \begin{cases} 0, & \exists e' \in \mathcal{E}. \mathcal{I}(e') < \mathcal{I}(e) \wedge (m,e) =_N (m,e') \\ 1, & otherwise. \end{cases}$$

Thus, we have $C_N = \sum_{m\in\mathcal{M}_F} \sum_{e\in\mathcal{E}} \mathcal{G}_N(m,e)$.

Proof. First, we prove that for any $m, m' \in M_F$ and $e, e' \in \mathcal{E}$, if $m \neq m'$ then $(m,e) \neq_N (m',e')$ by contradiction. Assuming there exists $m, m' \in M_F$ and $e, e' \in \mathcal{E}$ such that $m \neq m' \wedge (m,e) =_N (m',e')$. By the definition of $=_N$, we have $f(m,e) = f(m',e')$. We then have $f(m, ee'^{-1}) = m'$, thus $m =_F m'$. By the definitions of $=_F$ and $\mathcal{M}_F$, we know that if $m, m' \in \mathcal{M}_F$ and $m =_F m'$ then $m = m'$. This contradicts to $m \neq m'$.

Thus, we have that the number of the equivalence classes (w.r.t. $=_N$) in $\mathcal{M}_F \times \mathcal{E}$, which is C_N, is the sum of the number of the equivalence classes (w.r.t. $=_N$) of $\{m\} \times \mathcal{E}$ for every $m \in \mathcal{M}_F$. For each $m \in \mathcal{M}_F$, the number of equivalence classes (w.r.t. $=_N$) in $\{m\} \times \mathcal{E}$ is the sum of all labels $\sum_{e\in\mathcal{E}} \mathcal{G}_N(m,e)$. Because the model-permutation pair with the smallest permutation index in each equivalence class (w.r.t. $=_N$) is taken as the representative of that class and labeled as 1, and the rest pairs are labeled as 0. Thus, we have $C_N = \sum_{m\in\mathcal{M}_F} \sum_{e\in\mathcal{E}} \mathcal{G}_N(m,e)$. □

Theorem 3. *Let $\hat{\mathcal{E}}$ be a set of permutations uniformly sampled from $\mathcal{E}$. One estimator of C_N is $\hat{C}_N = \frac{|\mathcal{E}|}{|\hat{\mathcal{E}}|} \sum_{m\in\mathcal{M}_F} \sum_{e\in\hat{\mathcal{E}}} \mathcal{G}_N(m,e)$. $\hat{C}_N$ is an unbiased estimator of C_N.*

Proof. Let $\hat{\mu}_m = \frac{1}{|\hat{\mathcal{E}}|} \sum_{e\in\hat{\mathcal{E}}} \mathcal{G}_N(m,e)$ be the mean of the labels over $\{m\} \times \hat{\mathcal{E}}$, and $\mu_m = \frac{1}{|\mathcal{E}|} \sum_{e\in\mathcal{E}} \mathcal{G}_N(m,e)$ be the mean of labels over $\{m\} \times \mathcal{E}$. Based on the properties of simple random sampling [45] (Chapter 7.3, Page 205), we have $\mathbb{E}(\hat{\mu}_m) = \mu_m$. Thus, we have

$$\mathbb{E}(\hat{C}_N) = |\mathcal{E}| \sum_{m\in\mathcal{M}_F} \mathbb{E}(\hat{\mu}_m) = \sum_{m\in\mathcal{M}_F} |\mathcal{E}|\mu_m = C_N.$$

□

However, the labeling via the global labeling function $\mathcal{G}_N$ could be expensive when the permutation size becomes large. To make it scalable, we propose a sample labeling function $\hat{\mathcal{G}}_N$ as follows, with which the labeling is done only w.r.t. the model-permutation pairs in $\hat{\mathcal{M}}_F \times \hat{\mathcal{E}}$.

[2] In our implementation, for efficiency purposes, we utilize the Union-Find operations to combine disjoint sets as long as new edges are found, instead of constructing a complete graph before computing WCCs.

Theorem 4. *Let $\hat{\mathcal{M}}_F$ be the estimated non-isomorphic model set generated by our non-isomorphic model estimator. Let $\hat{\mathcal{I}} : \hat{\mathcal{E}} \mapsto \{1, \ldots, |\hat{\mathcal{E}}|\}$ be a sample ordering function which returns the index of a sampled permutation in $\hat{\mathcal{E}}$ based on a predefined ordering of the permutations in $\hat{\mathcal{E}}$. Let $\hat{\mathcal{G}}_N$ be a sample labeling function:*

$$\hat{\mathcal{G}}_N(m,e) = \begin{cases} 0, & \exists e' \in \hat{\mathcal{E}}.\hat{\mathcal{I}}(e') < \hat{\mathcal{I}}(e) \wedge (m,e) =_N (m,e') \\ 1, & otherwise. \end{cases}$$

One estimator of C_N is defined as $\hat{C}'_N = \frac{|\mathcal{E}|}{|\hat{\mathcal{E}}|} \sum_{m \in \hat{\mathcal{M}}_F} \sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e)$. $\mathbb{E}(\hat{C}'_N)$ is the upper bound of C_N.

Proof. For any $m, m' \in \hat{\mathcal{M}}_F$, if $m =_F m'$, then the number of equivalence classes in $\{m\} \times \hat{\mathcal{E}}$ equals to that in $\{m'\} \times \hat{\mathcal{E}}$. Thus, $\sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e) = \sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m',e)$. Since $\hat{\mathcal{M}}_F$ is the over-approximation of $\mathcal{M}_F$ (see Theorem 1), we have

$$\hat{C}'_N \geq \frac{|\mathcal{E}|}{|\hat{\mathcal{E}}|} \sum_{m \in \mathcal{M}_F} \sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e). \tag{1}$$

Next, for each $m \in \mathcal{M}_F$, we divide model-permutation pairs of $\{m\} \times \hat{\mathcal{E}}$ into equivalence classes w.r.t. $=_N$: if we label the pairs via $\hat{\mathcal{G}}_N$, there is exactly one pair labeled as 1 for each equivalence class (w.r.t. $=_N$); if we label the pairs via $\mathcal{G}_N$, there is at most one pair labeled as 1 for each equivalence class (w.r.t. $=_N$). Thus, we have

$$\sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e) \geq \sum_{e \in \hat{\mathcal{E}}} \mathcal{G}_N(m,e). \tag{2}$$

Based on Inequation 1, Inequation 2 and Theorem 3, we have

$$\mathbb{E}(\hat{C}'_N) \geq \mathbb{E}(\frac{|\mathcal{E}|}{|\hat{\mathcal{E}}|} \sum_{m \in \mathcal{M}_F} \sum_{e \in \hat{\mathcal{E}}} \mathcal{G}_N(m,e)) = \mathbb{E}(\hat{C}_N) = C_N.$$

□

Theorem 5. *$\hat{C}'_N \to C_N$ when $\hat{\mathcal{E}} \to \mathcal{E}$.*

Proof. If $\hat{\mathcal{E}} \to \mathcal{E}$, then $\hat{\mathcal{M}}_F \xrightarrow{=_F} \mathcal{M}_F$ based on Theorem 2, and $\hat{\mathcal{G}}_N \to \mathcal{G}_N$ based on Theorem 4. Thus, we have $\hat{C}'_N \to C_N$. □

We use $\hat{C}'_N = \frac{|\mathcal{E}|}{|\hat{\mathcal{E}}|} \sum_{m \in \hat{\mathcal{M}}_F} \sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e)$ as our final estimator of C_N. The estimator has the consistency property and its expectation provides the upper bound of C_N.

4.5.2 The Algorithm. Our isomorphic model counting algorithm for realizing the estimator $\hat{C}'_N$ is shown in Algorithm 3. The algorithm is built upon the non-isomorphic model enumeration algorithm, which reuses the sampled permutations and the non-isomorphic models produced by the non-isomorphic algorithm (line 2). Note that for each model $m \in \hat{\mathcal{M}}_F$, all the models in the permuted model set $\mathcal{M}_{\hat{\mathcal{E}}}$ are labeled as 1 and the rest (the duplicated models with the models in $\mathcal{M}_{\hat{\mathcal{E}}}$) are labeled as 0. Therefore, for each $m \in \hat{\mathcal{M}}_F$, the sum of the labels $\sum_{e \in \hat{\mathcal{E}}} \hat{\mathcal{G}}_N(m,e)$ is the size of the permuted model (line 7).

Algorithm 3 SymMC approximate isomorphic model counting.

Input: all the models under PaSB $\mathcal{M}_P$; symmetry info *Sym* produced by enhanced Kodkod module.
Output: the estimated isomorphic model count $\hat{C}'_N$.

```
1:  procedure ApproxIsom(M_P, Sym)
2:      M̂_F, _, Ê ← approxNonIsom(M_P, Sym)
3:      totalsamp ← |Ê|
4:      sum ← 0
5:      for each m ∈ M̂_F do
6:          M_Ê ← applyPerms(m, Ê)
7:          sum ← sum + |M_Ê|
8:      end for
9:      Ĉ'_N ← totalperm/totalsamp · sum
10:     return Ĉ'_N
11: end procedure
```

5 EXPERIMENTAL EVALUATION

Subjects We take the specifications from four sources as our subjects, including 1) 53 specifications from Alloy standard distribution, arising from a variety of real-world applications such as security in protocols and file systems; 2) 14 specifications from Kodkod standard distribution including constraint satisfaction problems such as the graph coloring problem and the Latin squares problem; 3) n-Queen (10 specifications) and 3-Queen problems (9 specifications) from the recent study of the symmetry breaking impact in model counting [60]; 4) 30 specifications from the recent study [36] that counts the models for 6 data structures (e.g., red-black trees and double linked lists), each with 5 different scopes. In total, there are 116 specifications, out of which 6 specifications have no detected symmetries (4 from Alloy category and 2 from Kodkod category). We exclude those subjects in our experiments. Therefore, there are 110 subjects in our benchmark, including 49 subjects in Alloy category, 12 subjects in Kodkod category, 19 subjects in n-Queen category, and 30 subjects in Data Structure category.
Platform All the experiments were performed on a machine with a 3.7 GHz Intel Core i7-8700K CPU and 32 GB RAM.
Research Questions Our research questions are summarized as follows:

- RQ1: How does SymMC perform in approximating the non-isomorphic models/count?
- RQ2: How does SymMC perform in approximating the isomorphic count?
- RQ3: What does the quantification metric of SymMC indicate about the pruning ability of Kodkod partial SBPs?

5.1 RQ1: SymMC Performance in Approximating Non-Isomorphic Models

The Baseline Since there is no existing automatic tool for non-isomorphic model enumeration for Alloy specifications, we make an exact model enumeration and counting variant of SymMC called SymMC-exact as our baseline, by turning off the permutation sampling of SymMC (i.e., do the enumeration and counting with all possible permutations).

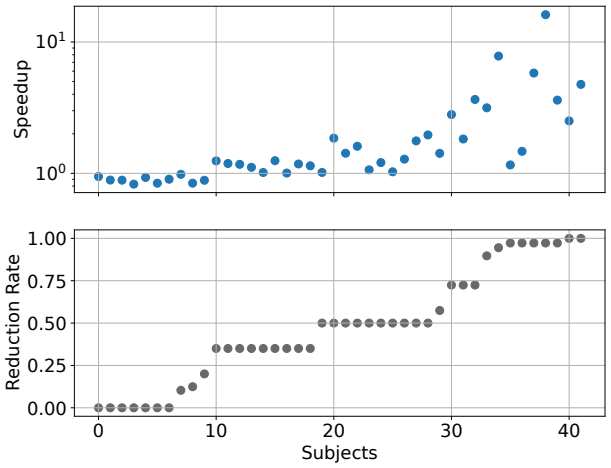


Figure 7: The top shows SymMC approximation speedup over SymMC-exact. The bottom shows SymMC permutation reduction rate, indicating the amount of work that can be saved with our permutation sampling approach.

Evaluation Metrics We evaluate SymMC in approximating non-isomorphic models in three aspects: time efficiency, the computations saved by our permutation sampling approach, and the approximation accuracy. For time efficiency, we report the speedup of SymMC over SymMC-exact. For the amount of saved computation, we use the permutation reduction rate $\frac{totalperm-totalsamp}{totalperm}$ as our metric. For evaluating the approximation accuracy, in line with prior work of ApproxMC, we use the error rate as $\max(\frac{approx}{exact}, \frac{exact}{approx}) - 1$, defined based on multiplicative guarantees [10].

Parameter Settings For parameters in Algorithm 2, we set the minimal permutation sample size *minperm* as 2,000; and the maximum permutation sample size *maxperm* as 100,000. For parameters in Algorithm 1, we set the initial sample ratio d as 0.5; the sample growth rate r as 0.15; and the early stopping criterion θ as 0.

Time Limit We use the standard time limit of 5,000 seconds in solving each subject in all categories.

The Non-Isomorphic Count Collection In order to validate the correctness and the approximation accuracy of SymMC non-isomorphic model enumeration and counting, we need to get the ground truth of the number of non-isomorphic models. For the subjects in Data Structure category and n-Queen category, we collect the non-isomorphic count in the On-line Encyclopedia of Integer Sequences (OEIS) [1] as the ground truths. For the other two categories, we get the ground truths by applying SymMC-exact with the extended time limit of 20,000 seconds.

5.1.1 Approximation Efficiency. We apply SymMC and SymMC-exact on all the subjects for enumerating the non-isomorphic models. Within the time limit, out of 110 subjects, SymMC solves 73 subjects (66.3%) while SymMC-exact solves 68 subjects (61.8%). There are no cases that cannot be solved by SymMC but solved by SymMC-exact. Note that there are 31 subjects out of the 73 subjects solved by SymMC that the total number of permutations *totalperm* is less than the minimal permutation sampling size *minperm*. Based on the *setSampSize* function (see Section 4.4.2), for those 31 subjects, SymMC does the exact enumeration without permutation sampling as SymMC-exact. To compare the SymMC approximation efficiency with SymMC-exact, we omit such subjects and only show the results of the rest 42 subjects.

Figure 7 shows the speedup of SymMC over SymMC-exact (top) and the permutation reduction rate of SymMC (bottom) for the 42 subjects. For demonstration purposes, we sort the subjects based on the permutation reduction rate in its ascending order. We can observe that, in general, the speedup of SymMC increases as the reduction rate increases. Overall, SymMC speeds up SymMC-exact in 74.4% subjects with up to 16.1x. On average, SymMC speeds up SymMC-exact 1.9x. We notice that, for two subjects, the permutation reduction rate is near 1 which means only a very small portion of permutations are applied in the approximation. In addition, we can see that when the reduction rate is near 0 which means all permutations are applied for the approximation, the speedup is slightly below 1 which means SymMC is slightly slower than SymMC-exact. This is because there is a slight overhead for SymMC to incrementally add the permutations. *In general, SymMC is able to approximate the non-isomorphic models of Alloy specifications with better time efficiency than the baseline.*

5.1.2 Approximation Accuracy. For 73 solved subjects by SymMC in enumerating non-isomorphic models, the ground truths of 71 subjects are obtained. Surprisingly, SymMC approximates the non-isomorphic models with 0.0 error rate for all these 71 subjects. *The results suggest that SymMC is able to approximate the non-isomorphic models with high accuracy, which is consistent with our intuition as discussed in Section 4.4.1.*

5.2 RQ2: SymMC Performance in Approximating the Isomorphic Count

Baselines For comparison in getting the isomorphic count, we select two robust state-of-the-art model counters (both in their default parameter settings) as our baselines, including one exact model counter called GANAK [47], and one approximate model counter called ApproxMC [10, 51].

Evaluation Metrics We evaluate SymMC in approximating the isomorphic count in two aspects: time efficiency and the approximation accuracy. For time efficiency, we report the actual wall-clock time cost of the counters. For approximation accuracy, we apply the approximation error rate $\max(\frac{approx}{exact}, \frac{exact}{approx}) - 1$ as used in evaluating the non-isomorphic model approximation.

Other Settings We utilize the same parameter setting and time limit of 5,000 seconds as in non-isomorphic approximation.

The Isomorphic Count Collection To evaluate the approximation accuracy, we get the ground truth of the isomorphic counts of our subjects with two tools. We apply SymMC-exact and GANAK on all the subjects with an extended time limit of 20,000 seconds.

5.2.1 Approximation Efficiency. We apply SymMC, GANAK, and ApproxMC on all the subjects for counting the isomorphic counts. Within the time limit, SymMC solves 72 subjects; ApproxMC solves 60 subjects; and GANAK solves 54 subjects, out of 110 subjects. Figure 8 shows the solving time (in seconds) of the three counters for the subjects that are solved by at least one counter (there are in

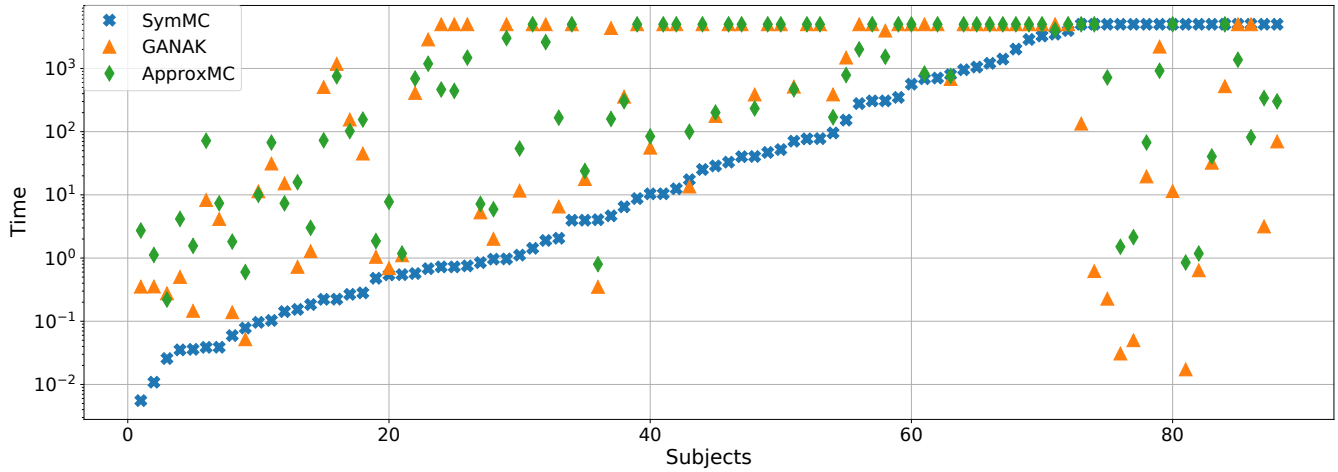


Figure 8: Sovling time (in seconds) of SymMC, GANAK, and ApproxMC in getting isomorphic counts for all the subjects solved by at least one counter.

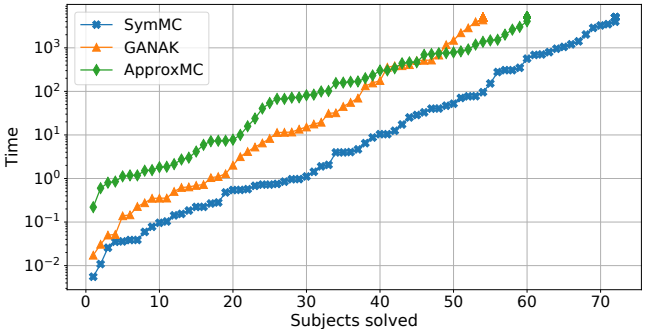


Figure 9: Cactus plot showing the behavior of SymMC, GANAK, and ApproxMC (time in seconds).

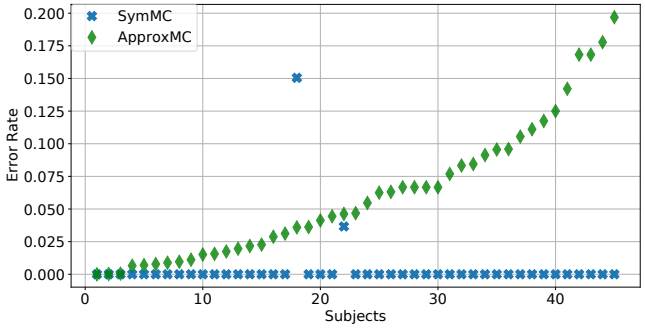


Figure 10: Error rates of SymMC and ApprxoMC in approximating the isomorphic count.

total 88 subjects); for demonstration purposes, we sort the subjects based on the solving time of SymMC in the ascending order. We can observe that SymMC clearly outperforms both GANAK and ApproxMC in more than three quarters of the subjects. In detail, SymMC outperforms GANAK in 68 out of 88 solved subjects (77.2%), and outperforms ApproxMC in 70 out of 88 subjects (79.5%). Overall, SymMC speeds up GANAK 1.8x, taking 973 seconds less in counting each subject on average; speeds up ApproxMC 1.6x, taking 703 seconds less in counting each subject on average.

Figure 9 shows the cactus plot for SymMC, GANAK and ApproxMC, where x-asix presents the number of subjects solved and y-axis presents the solving time. The cactus plot is commonly used in model counting community, which demonstrates the solving progress of the counters over time. We can see that SymMC takes the lead at the beginning, shows its clear superiority at 10 seconds, and maintains its advantage over two baselines until the end.

We further studied the subjects that are not solved by SymMC but solved by either of the two baselines. There are in total 15 such subjects. We found that 13 out of 15 subjects have a large number of models w.r.t. PaSB, ranging from 10,259,500 to 3,941,750,000 which chokes the blocking clause-based AllSAT solver in the current implementation of SymMC. However, based on the experimental results in the recent survey [52], the BDD-based AllSAT solver (mentioned in Section 4.3) is able to enumerate more than one quadrillion models, with which SymMC might be able to solve more such subjects. We will leave this possible improvement as our future work.

The results suggest that, for computing the isomorphic count, when models of the specification w.r.t. PaSB is enumerable by the AllSAT solver and there are symmetries present in the specification, SymMC could be a preferable choice. *Overall, the results show that SymMC is able to approximate the isomorphic model count of Alloy specifications with better time efficiency than the baseline counters.*

5.2.2 Approximation Accuracy. To evaluate the approximation accuracy of SymMC, we take the state-of-the-art approximate model counter ApproxMC as our baseline. Since GANAK is an exact model counter, we do not take GANAK as the baseline for approximation accuracy evaluation. For 72 solved subjects by SymMC, the ground truths of 64 subjects are obtained; for 60 solved subjects by ApproxMC, the ground truths of 56 subjects are obtained. The results show that SymMC solves the 64 subjects with the maximum error rate of 0.15, the minimum error rate of 0.0, and the average error rate of 0.003; ApproxMC solves the 56 subjects with the maximum error rate of 0.20, the minimum error rate of 0.0, and the average

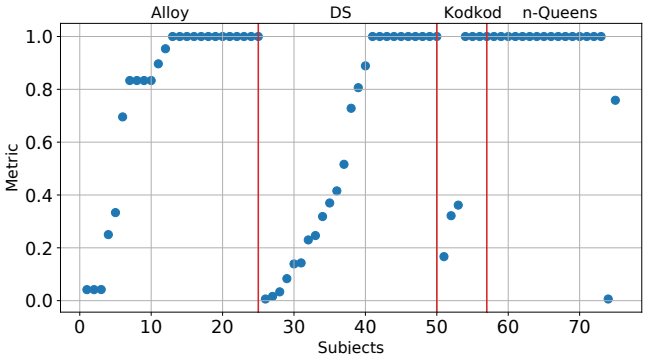


Figure 11: SymMC quantification metric for the pruning ability of Alloy SBPs.

error rate of 0.06. In addition, for 58 out of 64 subjects (90.6%), SymMC approximates the count accurately (with 0.0 error rate), while ApproxMC can approximate accurately for only one subject.

To further compare SymMC and ApproxMC in terms of the accuracy in approximating the isomorphic counts, we collect all the subjects that are solved by both counters and also have the ground truth. There are in total 45 subjects. The error rate of SymMC and ApproxMC in approximating the isomorphic count of these 45 subjects is shown in Figure 10. For demonstration purposes, we sort the subjects based on the error rate of ApproxMC in the ascending order. We can clearly observe that, as the error rate of ApproxMC increases, the error rate of SymMC stays almost stable at around 0.0. In detail, for 43 out of 45 subjects (95.6%), SymMC approximates the count in a lower error rate than ApproxMC. *Overall, results suggest that SymMC is able to approximate the isomorphic model count of Alloy specifications with a lower error rate than ApproxMC.*

5.3 RQ3: SymMC quantification measurement

We use the 73 solved subjects by SymMC non-isomorphic enumeration as the subjects in studying SymMC quantification measurement for the pruning ability of Kodkod partial SBPs. Figure 11 shows the values of SymMC quantification metric in evaluating the pruning ability of the applied Kodkod partial SBPs for each solved subject classified by categories. For demonstration purposes, we sort the subjects within each category based on the metric value in its ascending order. Overall, the quantification metric ranges from 0.006 to 1 with an average of 0.759; and the metric value is below 0.5 in 19 subjects. To be specific, for 25 solved Alloy subjects, the quantification metric ranges from 0.041 to 1 with an average of 0.784; for 25 solved data structure subjects, the metric ranges from 0.006 to 1 with an average of 0.598; for 7 solved Kodkod subjects, the metric ranges from 0.167 to 1 with an average of 0.693; the metric is 1 for all 16 solved n-Queen subjects. *The results show that the pruning ability of Kodkod partial SBPs is often effective and even perfect in many subjects (e.g., n-Queen problems), while the ability is sometimes limited in some other subjects (e.g., singly linked list data structure).*

6 CONCLUSION

This paper presented a symmetry exploitation tool called SymMC, which provides the first automatic non-isomorphic models/count approximation approach for Alloy specifications and provides a competitive isomorphic count approximation approach for Alloy specifications. In addition, SymMC provides an automatic quantification measurement on the solution space pruning ability of Kodkod PaSB. We hope that SymMC could shed light on specialized model counting/enumeration for other specifications (e.g., the specifications of SAT-based finite model finders) or more generalized model counting approaches applicable to multiple kinds of specifications.

ACKNOWLEDGEMENT

We thank Darko Marinov and reviewers for very helpful comments and feedback. This work was supported by CCF-1718903, and a grant from the Army Research Office accomplished under Cooperative Agreement Number W911NF-19-2-0333. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] 2019. The On-Line Encyclopedia of Integer Sequences. https://oeis.org/.
[2] Devdatta Akhawe, Adam Barth, Peifung E Lam, John Mitchell, and Dawn Song. 2010. Towards a formal foundation of web security. In *2010 23rd IEEE Computer Security Foundations Symposium*. IEEE, 290–304.
[3] Sven Apel, Wolfgang Scholz, Christian Lengauer, and Christian Kastner. 2010. Detecting dependences and interactions in feature-oriented design. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 161–170.
[4] Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey. 2015. SAT: Projected Model Counting. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 121–137.
[5] Rolf Backofen and Sebastian Will. 1999. Excluding Symmetries in Constraint-Based Search. In *Principles and Practice of Constraint Programming – CP'99*, Joxan Jaffar (Ed.). Springer, Berlin, Heidelberg, 73–87.
[6] Biljana Bajić-Bizumić, Claude Petitpierre, Hieu Chi Huynh, and Alain Wegmann. 2013. A model-driven environment for service design, simulation and prototyping. In *International Conference on Exploring Services Science*. Springer, 200–214.
[7] Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Feature and meta-models in Clafer: mixed, specialized, and coupled. In *International Conference on Software Language Engineering*. Springer, 102–122.
[8] Fabian Büttner, Marina Egea, Jordi Cabot, and Martin Gogolla. 2012. Verification of ATL transformations using transformation models and model finders. In *International Conference on Formal Engineering Methods*. Springer, 198–213.
[9] Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. 2013. A scalable and nearly uniform generator of SAT witnesses. In *International Conference on Computer Aided Verification*. Springer, 608–623.
[10] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, 3569–3576.
[11] I-Ming Chen and Joel W Burdick. 1998. Enumerating the non-isomorphic assembly configurations of modular robotic systems. *The International Journal of Robotics Research* 17, 7 (1998), 702–719.
[12] Thomas H. Cormen. 2009. *Introduction to Algorithms, Third Edition*. (3rd ed. ed.). MIT Press, Cambridge.
[13] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. 1996. Symmetry-breaking predicates for search problems. *KR* 96 (1996), 148–159.
[14] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. 2017. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 83–100.
[15] Jo Devriendt, Bart Bogaerts, Broes de_ Cat, Marc Denecker, and Christopher Mears. 2012. Symmetry propagation: Improved dynamic symmetry breaking in SAT. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, Vol. 1. IEEE, 49–56.
[16] Niklas Eén and Niklas Sörensson. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*. Springer, 502–518.
[17] Stefano Ermon, Carla Gomes, and Bart Selman. 2012. Uniform Solution Sampling Using a Constraint Solver as an Oracle. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence* (Catalina Island, CA) *(UAI'12)*. AUAI Press, Arlington, Virginia, USA, 255–264.

[18] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. 2001. Symmetry Breaking. In *Principles and Practice of Constraint Programming — CP 2001*, Toby Walsh (Ed.). Springer, Berlin, Heidelberg, 93–107.

[19] Filippo Focacci and Michaela Milano. 2001. Global Cut Framework for Removing Symmetries. In *Principles and Practice of Constraint Programming — CP 2001*, Toby Walsh (Ed.). Springer, Berlin, Heidelberg.

[20] Ian P Gent and Barbara Smith. 2000. Symmetry Breaking in Constraint Programming. In *ECAI*.

[21] Vibhav Gogate and Rina Dechter. 2006. A new algorithm for sampling CSP solutions uniformly at random. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 711–715.

[22] Vibhav Gogate and Rina Dechter. 2007. Approximate counting by sampling the backtrack-free search space. In *AAAI*. 198–203.

[23] Vibhav Gogate and Rina Dechter. 2011. SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence* 175, 2 (2011), 694–729.

[24] Carla P Gomes, Joerg Hoffmann, Ashish Sabharwal, and Bart Selman. 2007. From Sampling to Model Counting.. In *IJCAI*, Vol. 2007. 2293–2299.

[25] Carla P Gomes, Ashish Sabharwal, and Bart Selman. 2006. Model counting: A new strategy for obtaining good bounds. In *AAAI*. 54–61.

[26] Daniel Jackson. 2000. Automating first-order relational logic. In *Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications*. 130–139.

[27] Daniel Jackson. 2012. *Software Abstractions: logic, language, and analysis*. MIT press.

[28] Eunsuk Kang and Daniel Jackson. 2008. Formal modeling and analysis of a flash filesystem in Alloy. In *International Conference on Abstract State Machines, B and Z*. Springer, 294–308.

[29] Shadi Abdul Khalek, Guowei Yang, Lingming Zhang, Darko Marinov, and Sarfraz Khurshid. 2011. Testera: A tool for testing java programs using alloy specifications. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 608–611.

[30] Sarfraz Khurshid and Darko Marinov. 2004. TestEra: Specification-based testing of Java programs using SAT. *Automated Software Engineering* 11, 4 (2004), 403–434.

[31] Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. 2003. A case for efficient solution enumeration. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 272–286.

[32] Sarfraz Khurshid, Corina S Păsăreanu, and Willem Visser. 2003. Generalized symbolic execution for model checking and testing. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 553–568.

[33] Lukas Kroc, Ashish Sabharwal, and Bart Selman. 2008. Leveraging belief propagation, backtrack search, and statistics for model counting. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 127–141.

[34] Jinguo Liu, Yuechao Wang, Shugen Ma, and Yangmin Li. 2010. Enumeration of the non-isomorphic configurations for a reconfigurable modular robot with square-cubic-cell modules. *International Journal of Advanced Robotic Systems* 7, 4 (2010), 31.

[35] Darko Marinov and Sarfraz Khurshid. 2001. TestEra: A novel framework for automated testing of Java programs. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*. IEEE, 22–31.

[36] Darko Marinov and Sarfraz Khurshid. 2001. TestEra: A novel framework for automated testing of Java programs. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*. IEEE, 22–31.

[37] Brendan D McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of symbolic computation* 60 (2014), 94–112.

[38] C Mears. 2009. *Automatic symmetry detection and dynamic symmetry breaking for constraint programming*. Ph. D. Dissertation. Ph. D. thesis, Clayton School of Information Technology, Monash University.

[39] Christopher Mears, Maria Garcia De La Banda, Bart Demoen, and Mark Wallace. 2014. Lightweight dynamic symmetry breaking. *Constraints* 19, 3 (2014), 195–242.

[40] Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. 2018. CDCLSym: Introducing effective symmetry breaking in SAT solving. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 99–114.

[41] Patryk Mikos. 2021. Efficient enumeration of non-isomorphic interval graphs. *Discrete Mathematics & Theoretical Computer Science* 23 (2021).

[42] Aleksandar Milicevic, Sasa Misailovic, Darko Marinov, and Sarfraz Khurshid. 2007. Korat: A tool for generating structurally complex test inputs. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 771–774.

[43] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M Pai, and Sanjay Singh. 2011. Formal verification of OAuth 2.0 using Alloy framework. In *2011 International Conference on Communication Systems and Network Technologies*. IEEE, 655–659.

[44] Karen E Petrie, Barbara M Smith, and Neil Yorke-Smith. 2004. Dynamic symmetry breaking in constraint programming and linear programming hybrids. In *European starting AI researcher symp*. Citeseer.

[45] John A Rice. 2007. *Mathematical statistics and data analysis, 3rd Edition*. Thomson Higher Education.

[46] Bas Schaafsma, Marijn JH Heule, and Hans Van Maaren. 2009. Dynamic symmetry breaking by simulating zykov contraction. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 223–236.

[47] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S Meel. 2019. GANAK: a scalable probabilistic exact model counter. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 1169–1176.

[48] Ilya Shlyakhter. 2007. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics* 155, 12 (2007), 1539–1548.

[49] Ilya Shlyakhter. 2007. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics* 155, 12 (2007), 1539–1548.

[50] Michael Sipser. 1983. A complexity theoretic approach to randomness. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. 330–335.

[51] Mate Soos and Kuldeep S Meel. 2019. Bird: Engineering an efficient CNF-XOR sat solver and its applications to approximate model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1592–1599.

[52] Takahisa Toda and Takehide Soh. 2016. Implementing efficient all solutions SAT solvers. *Journal of Experimental Algorithmics (JEA)* 21 (2016), 1–44.

[53] Emina Torlak. 2009. *A constraint solver for software engineering: finding models and cores of large relational specifications*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[54] Emina Torlak and Daniel Jackson. 2007. Kodkod: A relational model finder. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 632–647.

[55] Dat Hoang Tran and Ryuhei Uehara. 2020. Efficient enumeration of non-isomorphic ptolemaic graphs. In *International Workshop on Algorithms and Computation*. Springer, 296–307.

[56] Caroline Trippel, Daniel Lustig, and Margaret Martonosi. 2018. Checkmate: Automated synthesis of hardware exploits and security litmus tests. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 947–960.

[57] Muhammad Usman, Wenxi Wang, and Sarfraz Khurshid. 2020. TestMC: testing model counters using differential and metamorphic testing. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 709–721.

[58] Timothy Van Bremen, Vincent Derkinderen, Shubham Sharma, Subhajit Roy, and Kuldeep S Meel. 2021. Symmetric Component Caching for Model Counting on Combinatorial Instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 3922–3930.

[59] Willem Visser, Corina S Păsăreanu, and Sarfraz Khurshid. 2004. Test input generation with Java PathFinder. In *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. 97–107.

[60] Wenxi Wang, Muhammad Usman, Alyas Almaawi, Kaiyuan Wang, Kuldeep S Meel, and Sarfraz Khurshid. 2020. A Study of Symmetry Breaking Predicates and Model Counting. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 115–134.

[61] Wei Wei and Bart Selman. 2005. A new approach to model counting. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 324–339.

[62] Jiayi Yang, Wenxi Wang, Darko Marinov, and Sarfraz Khurshid. 2020. AlloyMC: Alloy Meets Model Counting. In *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Demo Papers*. 1541–1545.