

Probabilistic Bisimulation for Parameterized Anonymity and Uniformity Verification

Chih-Duo Hong, Anthony W. Lin, Philipp Rümmer, Rupak Majumdar

Abstract—Bisimulation is crucial for verifying process equivalence in probabilistic systems. This paper presents a novel logical framework for analyzing bisimulation in probabilistic parameterized systems, namely, infinite families of finite-state probabilistic systems. Our framework is built upon the first-order theory of regular structures, which provides a decidable logic for reasoning about these systems. We show that essential properties like anonymity and uniformity can be encoded and verified within this framework in a manner aligning with the principles of deductive software verification, where systems, properties, and proofs are expressed in a unified decidable logic. By integrating language inference techniques, we achieve full automation in synthesizing candidate bisimulation proofs for anonymity and uniformity. We demonstrate the efficacy of our approach by addressing several challenging examples, including cryptographic protocols and randomized algorithms that were previously beyond the reach of fully automated methods.

I. INTRODUCTION

Formal verification techniques are essential for developing reliable and sustainable software systems. Among these techniques, bisimulation equivalence provides a powerful means of establishing that two systems exhibit indistinguishable observable behaviors and meet the same requirements specified in expressive modal logic [1], [2]. In software model checking, bisimulation enables proving a system’s correctness by showing that the system mirrors the specification step by step [3], [4]. In applications involving secrecy and privacy, bisimulation can establish non-leakage of sensitive information by showing that a potential attacker cannot distinguish between the real protocol’s behavior and the idealized confidential behavior [5]–[7]. Other applications of bisimulation equivalence include information flow analysis [8]–[10], knowledge reasoning [11]–[13], runtime verification [14], and computational optimization problems of finite-state automata [15]–[17] and probabilistic systems [18]–[21].

Due to its rich applications, the problem of checking bisimulation equivalence has been extensively studied. This problem

is decidable for both probabilistic and nondeterministic *finite-state* systems [22]–[24]. For infinite-state systems, such as communication protocols involving an unbounded number of processes, the problem is generally undecidable [25]. Therefore, research on the automatic verification of bisimulation for infinite-state systems has primarily taken two directions. The first focuses on developing heuristics and approximation techniques for the general undecidable problems [26]–[28]. The second seeks to identify subclasses of infinite-state systems where bisimulation equivalence remains decidable. For example, studies like [29]–[32] probe into bisimulation equivalence for pushdown systems and their variants in both probabilistic and non-probabilistic settings. As noted by Garavel and Lang [33], most research in infinite-state bisimulation has leaned towards theoretical work over developing practical tools.

In this paper, we introduce a first-order framework for reasoning about bisimulation equivalence in probabilistic infinite-state systems. Drawing from recent advances in deductive verification [34]–[36], we represent these systems in a decidable theory that enables a uniform formalization of the target system, its correctness properties, and the proofs of these properties. Our key contribution is leveraging the first-order theory of *regular structures* [37]–[39] to develop decidable proof rules for probabilistic bisimulation: given a binary relation R encoded in this theory, our rules generate verification conditions determining whether R constitutes a probabilistic bisimulation, which can be automatically checked using theorem provers and constraint solvers. We showcase this approach’s efficacy by examining two essential properties in cryptographic protocols and randomized algorithms: anonymity and uniformity. *Anonymity* protects the identities of participants in a protocol. An anonymous protocol guarantees that external observers cannot identify the individuals involved in the interactions, thereby preserving participant confidentiality [40]–[42]. *Uniformity* focuses on the output distribution of a randomized protocol or algorithm, ensuring that the results are evenly distributed across a specified range [43], [44].

To illustrate the concepts of anonymity and uniformity, consider the *dining cryptographers protocol*. [45]. This protocol involves $n \geq 3$ participants on a ring, each holding a secret bit. Let x_i denote the secret bit held by participant i for $i \in \{0, \dots, n-1\}$. Define the *parity* of these bits as $f(\bar{x}) := x_0 \oplus \dots \oplus x_{n-1}$, where $\oplus$ stands for xor. The participants aim to compute $f(\bar{x})$ without revealing information about the individual bits. To achieve this, they execute the computation in two stages. (As the participants are arranged on a ring of length n , all index arithmetic in the sequel is performed modulo n .) First, each pair of adjacent

Chih-Duo Hong is with the National Chengchi University, Taipei, Taiwan. Anthony W. Lin is with the University of Kaiserslautern-Landau, Kaiserslautern, Germany, and the Max Planck Institute for Software Systems, Kaiserslautern, Germany. Philipp Rümmer is with the University of Regensburg, Regensburg, Germany, and Uppsala University, Sweden. Rupak Majumdar is with the Max Planck Institute for Software Systems, Kaiserslautern, Germany. (Corresponding author: Chih-Duo Hong)

Chih-Duo Hong is supported by the National Science and Technology Council, Taiwan, under grant number NSTC112-2222-E-004-001-MY3. Anthony Lin is supported by the European Research Council under the European Union’s Horizon 2020 research and innovation programme under number 101089343. Philipp Rümmer is supported by the Swedish Research Council through grant 2021-06327.

participants i and $i + 1$ computes a random bit b_i that is visible only to them. Then, each participant i announces $a_i := x_i \oplus b_i \oplus b_{i-1}$ to all participants, allowing them to compute $f(\bar{x})$ since $f(\bar{a}) = f(\bar{x})$. The *anonymity property* of the protocol asserts that any observing participant, say participant k , cannot infer the other participants' secrets from observed information, including her own secret x_k , the random bits b_k and b_{k-1} , and the announcements $a_0, \dots, a_{n-1}$. To establish this property, we may show that the probability distribution of the announcements $a_0, \dots, a_{n-1}$ is solely determined by the values of x_k, b_k, b_{k-1} , and $f(\bar{x})$. Consequently, no information beyond these values can be inferred from the announcements. If we model the dining cryptographers protocol as a Markov decision process (MDP), we can verify the anonymity property by showing that any two initial states of the process, say, one with secrets $\bar{x}$ and the other with secrets $\bar{y}$, are bisimilar as long as $x_k = y_k$ and $f(\bar{x}) = f(\bar{y})$. Alternatively, we can prove anonymity by showing that, starting from an initial state with secrets $\bar{x}$, the announcements observed by participant k are uniformly distributed over

$$\{ \bar{a} \in \{0, 1\}^n : a_k = x_k \oplus b_k \oplus b_{k-1}, f(\bar{a}) = f(\bar{x}) \},$$

which indicates that no information beyond the values of x_k, b_k, b_{k-1} , and $f(x)$ can be inferred from the value of $\bar{a}$. Requirements like this are called *uniformity properties*. We provide more details about these properties in Section V.

We show that anonymity and uniformity properties can be model checked in a unified manner for a class of infinite-state systems called parameterized systems. A *parameterized system* $\{P_n\}_{n \in N}$ comprises an infinite family of finite-state systems indexed by a parameter $n \in N$ [46]–[48]. A parameterized system satisfies a property if every instance P_n of the system satisfies that property. For example, the dining cryptographers protocol is anonymous if it is anonymous for any $n \geq 3$ cryptographers. In this work, we focus on parameterized systems that can be represented within the first-order theory of regular structures, which is expressive enough to capture a range of examples in the literature. Specifically, we examine the (parameterized) dining cryptographers protocol [45], grades protocol [49], crowds protocol [50], random walks, random sums, Knuth-Yao's random number generator [51], and Bertrand's ballot theorem [52]. Drawing inspiration from software verification, where safety proofs are typically separated into proof rules and inductive invariant synthesis, our proofs for anonymity and uniformity similarly distinguish between proof rules and bisimulation synthesis. We show that recent advancements in proof generation and refinement [53]–[55] can be utilized to search for candidate proofs effectively, making our verification procedure truly “push-button”.

Contributions. The main contributions of this paper are as follows. Firstly, we propose to employ bisimulation equivalence as a unified framework for model checking anonymity and uniformity — two properties that have traditionally been addressed with distinct techniques in infinite-state settings.

Secondly, we demonstrate that probabilistic systems satisfying the *minimal deviation assumption* [56] (i.e., all transition probabilities are multiples of some $\varepsilon > 0$) can be faithfully encoded in the first-order theory of regular structures. This

theory has been previously applied to reason about *qualitative* liveness of MDPs [48], where the actual probability values are abstracted away. To our knowledge, this work offers the first quantitative encoding of probabilistic systems in the theory.

Thirdly, assuming minimal deviation, we encode the verification conditions for regular probabilistic bisimulation relations in the aforementioned theory, yielding an algorithmic approach to checking anonymity and uniformity properties. Indeed, since this theory is syntactically reducible to the weak monadic second-order logic with one successor (WS1S) [38], our encodings can be manipulated and analyzed using highly optimized tools like MONA [57] and GASTON [58].

Thus far, our framework has relied on user-provided proofs for verifying anonymity and uniformity. Our final contribution is demonstrating how language inference algorithms [59]–[61] can be leveraged to generalize proofs derived from finite system instances to establish correctness for the entire parameterized system. Thanks to these techniques, we successfully verified challenging examples that were previously beyond the reach of fully automated approaches.

This paper is a significant extension of the conference paper [27], which focused solely on the anonymity verification of parameterized MDPs. This current work expands the previous formalism and results by providing a unified logical framework capable of reasoning about both anonymity and uniformity properties. We also offer new case studies to demonstrate the effectiveness of this extended approach.

II. PRELIMINARIES

A. Weighted transition system

A *weighted transition system* (WTS) is a three-sorted structure $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$, where S is a countable set of configurations, P is a countable set of nonnegative numbers, A is a countable set of actions, $+ : P \times P \rightarrow P$ defines the addition operation over P , and $\delta : S \times A \times S \rightarrow P$ is called a *weighted transition function*. We assume that the elements in S, P, A are *named* for each element e , we can effectively find a constant symbol c_e in $\mathfrak{S}$ that is interpreted to e . For configurations $s, t \in S$, we use $\delta(s, a, t)$ to denote the transition weight from s to t via action a . When $\delta(s, a, t) \neq 0$, we write $s \rightarrow_a t$ to indicate that s can move to a successor t in one step. A finite or infinite sequence $\pi := s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} s_2 \rightarrow_{a_3} \dots$ is called a *path*. A configuration s is *reachable* from s' if there is a path from s' to s . A WTS is *bounded branching* if for every action, the system can only reach a bounded number of configurations in one step. Namely, there exists a universal bound $b < \infty$ such that for each $s \in S$ and $a \in A$, $|\{t \in S : s \rightarrow_a t\}| \leq b$. Since a WTS can have infinitely many actions, a configuration may still have an unbounded number of successors even though the system is bounded branching.

A WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ is called a *Markov chain* if there is a constant $q \in P$ such that $\sum_{a \in A} \sum_{t \in S} \delta(s, a, t) = q$ for each $s \in S$. $\mathfrak{S}$ is a *Markov decision process* (MDP) if there is a constant $q \in P$ such that $\sum_{t \in S} \delta(s, a, t) \in \{0, q\}$ holds for each $s \in S, a \in A$. In both cases, we interpret $\delta(s, a, t)$ as the transition probability $\delta(s, a, t)/q$ of the transition $s \rightarrow_a t$. A Markov chain emits an action after it determines a transition.

In contrast, a Markov decision process needs to select an action before making a transition. If the process selects action a in state s , it moves to a state t with probability $\delta(s, a, t)/q$. Note that a Markov chain has no terminal state by definition, whilst a Markov decision process can get stuck by selecting an action a in a state s such that $\sum_{t \in S} \delta(s, a, t) = 0$.

In [56], Larsen and Skou introduced the *minimal deviation assumption*, restricting a probabilistic system such that all transition probabilities share a common divisor. This assumption holds for most probabilistic systems we encounter, including probabilistic pushdown automata [31], [62], probabilistic parameterized systems [48], and all case studies in Section VI. Notably, a probabilistic system with minimal deviation induces a bounded branching WTS with natural weights. We will adopt this assumption in the sequel and later discuss how to relax it in Section V-C.

B. Bisimulation equivalence

Let $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ be a WTS. A *bisimulation* over $\mathfrak{S}$ is an equivalence relation $R \subseteq S \times S$ such that $(s, s') \in R$ implies

$$\forall a \in A. \forall E \in S/R. \sum_{t \in E} \delta(s, a, t) = \sum_{t' \in E} \delta(s', a, t'), \quad (1)$$

where S/R denotes the set of equivalence classes induced by R . Two configurations s, s' of $\mathfrak{S}$ are *bisimilar* if there exists a bisimulation R over $\mathfrak{S}$ such that $(s, s') \in R$. Intuitively, bisimilar configurations emit the same amount of probability mass to the same equivalence class for any action. The union of all bisimulations over $\mathfrak{S}$ is itself a bisimulation over $\mathfrak{S}$; this maximal bisimulation is also called *bisimulation equivalence* or *bisimilarity* [2].

Bisimilarity can be lifted to relate two systems. Given two WTSs $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ and $\mathfrak{S}' := \langle S', P, A, \delta', + \rangle$ such that $S \cap S' = \emptyset$, we define the *disjoint union* of $\mathfrak{S}$ and $\mathfrak{S}'$ as the WTS $\mathfrak{S} \uplus \mathfrak{S}' := \langle S'', P, A, \delta'', + \rangle$ with $S'' := S \uplus S'$ and

$$\delta''(s, a, t) := \begin{cases} \delta(s, a, t), & s, t \in S; \\ \delta'(s, a, t), & s, t \in S'. \end{cases}$$

A binary relation R over $S \uplus S'$ is called a *bisimulation* between $\mathfrak{S}$ and $\mathfrak{S}'$ if R is a bisimulation over $\mathfrak{S} \uplus \mathfrak{S}'$.

Probabilistic modal logic (PML) [56], originally introduced for probabilistic systems, extends classical propositional logic with formulas of the form $\langle a \rangle_p \phi$, where p is a probability. A state satisfies $\langle a \rangle_p \phi$ if it can move to states satisfying ϕ with probability exceeding p through an a -labeled transition. Many results of PML can be naturally generalized to WTSs. In this work, we will exploit the following key property of PML.

Proposition 1 ([63], [64]). *Two configurations of a WTS are bisimilar if and only if they satisfy the same PML formulas. Furthermore, it is decidable to check whether a configuration satisfies a PML formula in a finite WTS.*

C. A first-order framework for regular relations

An *alphabet* Σ is a finite set of letters, and a *word* is a finite sequence of letters. We use Σ^* to denote the set of words over

alphabet Σ . Note that Σ^* contains ε , the empty word. Given $w \in \Sigma^*$, $|w|$ is the length of w and $w[i]$ is the i th letter of w for $i \in \{1, \dots, |w|\}$. We define $\Sigma^n := \{(a_1, \dots, a_n) : a_1, \dots, a_n \in \Sigma\}$ as the set of n -tuples over Σ , and denote $\Sigma_\#$ by $\Sigma \uplus \{\#\}$, where $\# \notin \Sigma$ is the *blank symbol*. The *convolution* $w_1 \otimes \dots \otimes w_n$ of n words $w_1, \dots, w_n \in \Sigma^*$ is the word $w \in (\Sigma_\#^n)^*$ satisfying (i) $|w| = \max\{|w_1|, \dots, |w_n|\}$, and (ii) $w[i] = (a_1, \dots, a_n)$ for $i \in \{1, \dots, |w|\}$, where

$$a_k := \begin{cases} w_k[i] & |w_k| \geq i, \\ \# & \text{otherwise.} \end{cases}$$

In other words, w is the shortest word obtained by juxtaposing $w_1, \dots, w_n$ and padding the shorter words with the blank symbol $\#$. For example, $abb \otimes aa = (a, a)(b, a)(b, \#) \in (\Sigma_\#^2)^*$ when $\{a, b\} \subseteq \Sigma$. Given an n -ary relation $R \subseteq (\Sigma^*)^n$ over the words Σ^* , we define the *language representation* of R as

$$\mathcal{L}(R) := \{w_1 \otimes \dots \otimes w_n : (w_1, \dots, w_n) \in R\}. \quad (2)$$

We say that an n -ary relation $R \subseteq (\Sigma^*)^n$ is a *regular relation* if $\mathcal{L}(R) \subseteq (\Sigma_\#^n)^*$ is a regular language. We define a structure $\mathfrak{U} := \langle \Sigma^*, \preceq, eqL, \{\prec_a\}_{a \in \Sigma} \rangle$, where $\preceq$ is the prefix-of relation, eqL is the equal-length relation, and $\prec_a$ is the a -successor relation. More precisely, $w \preceq w'$ holds iff there exists $w'' \in \Sigma^*$ such that $w \cdot w'' = w'$, where $\cdot$ denotes word concatenation. Furthermore, $eqL(w, w')$ holds iff $|w| = |w'|$, and $w \prec_a w'$ holds iff $w \cdot a = w'$. Given a formula $\phi(x_1, \dots, x_n)$ defined over $\mathfrak{U}$, we use $\llbracket \phi \rrbracket$ to denote the set $\{(w_1, \dots, w_n) : \phi(w_1, \dots, w_n) \in \text{FO}(\mathfrak{U})\} \subseteq (\Sigma^*)^n$, where $\text{FO}(\mathfrak{U})$ denotes the first-order theory of $\mathfrak{U}$. The following result establishes a connection between logic and automata.

Proposition 2 ([37], [38]). *For any formula $\phi(x_1, \dots, x_n)$ defined over $\mathfrak{U}$, $\llbracket \phi \rrbracket \subseteq (\Sigma^*)^n$ is a regular relation, and we can compute from ϕ a finite automaton $\mathcal{A}$ recognizing $\mathcal{L}(\llbracket \phi \rrbracket)$. Conversely, for a regular relation $R \subseteq (\Sigma^*)^n$, suppose that $\mathcal{A}$ is a finite automaton recognizing $\mathcal{L}(R)$. Then we can compute from $\mathcal{A}$ a formula $\phi(x_1, \dots, x_n)$ over $\mathfrak{U}$ such that $\llbracket \phi \rrbracket = R$.*

An immediate consequence of Proposition 2 is that $\text{FO}(\mathfrak{U})$ is decidable, as can be shown using automata-theoretic arguments [65]. The theory remains decidable even if we extend the structure $\mathfrak{U}$ with arbitrary regular relations. In the sequel, we shall regard relations definable in $\mathfrak{U}$ (such as membership in a regular language) as regular relations, and freely use regular relations as syntactic sugar when we are defining a formula over $\mathfrak{U}$. For a fixed alphabet Σ , we shall use FO_{reg} to denote the decidable first-order theory of regular relations over Σ .

A structure $\mathfrak{S}$ is *regular* if its relational variant is isomorphic to a relational structure $\mathfrak{T}$ such that (i) the universes of $\mathfrak{T}$ are regular languages over a finite alphabet Σ , and (ii) all relations in $\mathfrak{T}$ are regular. In such case, we call $\mathfrak{T}$ a *regular presentation* of $\mathfrak{S}$. It follows by Proposition 2 that the first-order theory of a regular structure is decidable.

III. BISIMULATION IN REGULAR RELATIONS

In this section, we establish how weighted transition systems (WTSs) and bisimulation proof rules can be systematically formulated within our logical framework. We begin by specifying

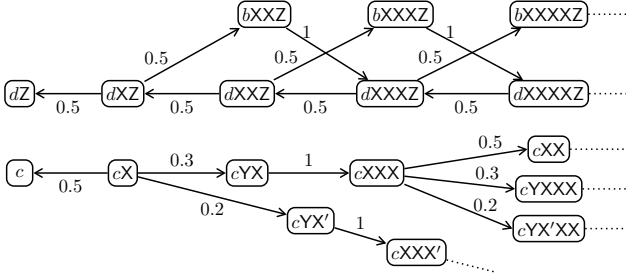


Fig. 1. Part of the configuration graph in Example 3, adapted from [31].

regular WTSs using the first-order theory of regular relations, followed by the definition of verification conditions within the same formalism. These results provide a unified approach for representing and reasoning about WTS properties and proofs.

A. The non-weighted case

As a warm-up, we first consider bisimulation relations over non-weighted transition systems. A *labeled transition system* (LTS) is a structure $\mathfrak{S} := \langle S, A, \delta \rangle$ with a labeled transition relation $\delta \subseteq S \times A \times S$. For convenience, denote $(s, a, t) \in \delta$ as $s \rightarrow_a t$. Then a binary relation $R \subseteq S \times S$ is a bisimulation on the LTS $\mathfrak{S}$ if for all $a \in A$, $t, t' \in S$, and $(s, s') \in R$:

- (i) $s \rightarrow_a t$ only if $s' \rightarrow_a t'$ and $(t, t') \in R$ for some t' ;
- (ii) $s' \rightarrow_a t'$ only if $s \rightarrow_a t$ and $(t, t') \in R$ for some t' .

These conditions can be expressed as a first-order formula:

$$\psi(s, a, s') := (\forall t. \delta(s, a, t) \Rightarrow \exists t''. \delta(s', a, t'') \wedge R(t, t'')) \wedge (\forall t'. \delta(s', a, t') \Rightarrow \exists t''. \delta(s, a, t'') \wedge R(t'', t')).$$

Given a binary relation $R \subseteq S \times S$, let $\mathfrak{S}_R$ denote the structure obtained by extending $\mathfrak{S}$ with R . Then R is a bisimulation on $\mathfrak{S}$ if and only if

$$\mathfrak{S}_R \models \forall s. \forall s'. (R(s, s') \Rightarrow \forall a. \psi(s, a, s')). \quad (3)$$

It is algorithmic to check (3) when both $\mathfrak{S}$ and R are regular. Indeed, as bisimilarity is preserved under isomorphism, we can verify the sentence by reasoning about the regular presentation of $\mathfrak{S}_R$ in the decidable theory FO_{reg} .

While validating a regular bisimulation for a regular LTS is decidable, checking bisimilarity for a regular LTS is not. To see this, notice that the configuration graph of a Turing machine (TM) is regular [66]: it can be modeled by a bounded-branching regular LTS with a dummy action a , such that each configuration encodes the current state and tape content, and the initial configuration s_0 stores an input x . Observe that s_0 is bisimilar to a configuration s with a single outgoing transition $s \rightarrow_a s$ if and only if the TM does not halt on x . Therefore, the halting problem can be reduced to bisimulation checking in a regular LTS, rendering the latter problem undecidable.

B. Specifying and validating weighted systems

Weighted transition systems introduce additional complexity compared to their non-weighted counterparts, requiring careful

specification and validation. Below, we examine how to formally define such systems, and illustrate this process through the regular presentation of a simple probabilistic system.

Example 3. Pushdown automata (PDAs) are finite-state machines equipped with a stack over a finite set of symbols. Each transition in a PDA can modify the stack by pushing or popping a symbol. Probabilistic pushdown automata (pPDAs), a.k.a. recursive Markov chains [62], further enrich PDAs with transition probabilities. Consider a pPDA from Example 1 of [31], which has states $Q = \{b, c, d\}$ and stack symbols $\Gamma = \{X, X', Y, Z\}$. A configuration $qs \in Q\Gamma^*$ is a word that comprises the current state $q \in Q$ and stack content $s \in \Gamma^*$. The transition rules are given by:

$$\begin{array}{lll} dX \xrightarrow{0.5} bXX & dX \xrightarrow{0.5} d & bX \xrightarrow{1} dXX \quad cY \xrightarrow{1} cXX \\ cX \xrightarrow{0.3} cYX & cX \xrightarrow{0.2} cYX' & cX \xrightarrow{0.5} c \\ cX' \xrightarrow{0.4} cYX & cX' \xrightarrow{0.1} cYX' & cX' \xrightarrow{0.5} c \end{array}$$

A rule is applicable if the configuration's prefix matches the left-hand side of the rule.

Figure 1 presents a fragment of the pPDA's configuration graph. We can model this graph as a WTS in FO_{reg} : the configuration set is $Q\Gamma^*$, the weights are encoded in binary after normalizing the probabilities to natural numbers. The transition relation is encoded as a disjunctive formula $\delta(s, a, t, p)$, e.g., the disjunct corresponding to the rule $dX \xrightarrow{0.5} bXX$ is $\exists u \in \Gamma^*. (s = dXu \wedge t = bXXu \wedge p = 101)$. The branching of the resulting WTS is bounded by 3.

Note that FO_{reg} is expressive enough to represent any pPDA with rational transition probabilities, since these probabilities can be encoded as natural weights in its regular presentation, as we have exemplified above. See [66] for further details on infinite-state systems with regular presentations.

We can effectively check whether a set of FO_{reg} formulas properly specifies a WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$, provided that the branching bound of $\mathfrak{S}$ is known. Suppose we have FO_{reg} formulas defining S , P , A , and a formula $\phi(s, a, t, p)$ defining the transition function δ . To check that these formulas indeed specify a WTS with a branching bound n , we essentially need to verify the following three conditions:

- 1) The branching is indeed bounded by n . That is, for each $s \in S$ and $a \in A$, there are at most n distinct t 's in S such that $\phi(s, a, t, p) \wedge p \neq 0$ is satisfiable.
- 2) ϕ encodes a function of sort $S \times A \times S \rightarrow P$. That is, for each $s, t \in S$ and $a \in A$, there exists precisely one $p \in P$ satisfying $\phi(s, a, t, p)$.
- 3) When the WTS is a Markov chain or a Markov decision process, ϕ should encode a mapping from S to the set of probability distributions over S .

The first two conditions are expressible in FO_{reg} and hence are algorithmic by Proposition 2. To verify the third condition, we define an auxiliary formula $\psi(s, a, \bar{t})$, which asserts that the successors of s through action a are among the configurations $\bar{t} := t_1, \dots, t_n$:

$$\psi(s, a, \bar{t}) := \forall t. \forall p. \left(\phi(s, a, t, p) \wedge p \neq 0 \Rightarrow \bigvee_i (t = t_i) \right).$$

The third condition then amounts to checking that there exists a number $q \in P$ such that

$$\forall s. \forall a. \left(\exists \bar{t}. \exists \bar{p}. \left(\psi(s, a, \bar{t}) \wedge \bigwedge_i \phi(s, a, t_i, p_i) \wedge \sum_i p_i = q \right) \right)$$

holds when the WTS is a Markov chain, and

$$\forall s. \forall a. (\forall t. \forall p. (\phi(s, a, t, p) \Leftrightarrow p = 0)) \\ \vee \left(\exists \bar{t}. \exists \bar{p}. \left(\psi(s, a, \bar{t}) \wedge \bigwedge_i \phi(s, a, t_i, p_i) \wedge \sum_i p_i = q \right) \right)$$

holds when the WTS is a Markov decision process. Again, this check is algorithmic in FO_{reg} . Hence, it is decidable to check whether a given regular presentation of a WTS is well-defined.

C. Proof rules for bisimulation over weighted systems

Recall that $\mathfrak{S}_R$ denotes the structure obtained by extending structure $\mathfrak{S}$ with relation R . The following theorem summarizes the main technical result of this work.

Theorem 4. *There is a fixed first-order sentence Φ such that a given binary relation R is a bisimulation on a WTS $\mathfrak{S}$ if and only if $\mathfrak{S}_R \models \Phi$. Furthermore, checking $\mathfrak{S}_R \models \Phi$ is decidable when both $\mathfrak{S}$ and R are regular.*

Proof. Fix a WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ with branching bound n . The fact that a binary relation R is an equivalence relation can be expressed by $\Phi_{\text{eq}} := \forall s. \forall s'. \forall s''. R(s, s) \wedge (R(s, s') \Rightarrow R(s', s)) \wedge (R(s, s') \wedge R(s', s'') \Rightarrow R(s, s''))$. Now, it suffices to define a sentence Φ as

$$\Phi_{\text{eq}} \wedge \forall s. \forall s'. R(s, s') \Rightarrow \forall a. (\psi(s, a, s') \vee \lambda(s, a, s')), \quad (4)$$

such that R is a bisimulation over $\mathfrak{S}$ if and only if $\mathfrak{S}_R \models \Phi$. Here, $\psi(s, a, s')$ is an auxiliary formula asserting that configurations s and s' have no successor through action a :

$$\psi(s, a, s') := \forall t. (\delta(s, a, t) = 0 \wedge \delta(s', a, t) = 0). \quad (5)$$

Before defining $\lambda(s, a, t)$, we first offer some intuition and auxiliary formulas. Given configurations s and t , the formula $\lambda(s, a, t)$ will first guess a set of n configurations $\bar{u}$ containing the successors of s through action a , and a set of n configurations $\bar{v}$ containing the successors of t through action a . The formula will also guess a labeling $\bar{\alpha}$ and $\bar{\beta}$ that corresponds to the partitioning of the configurations $\bar{u}$ and $\bar{v}$, respectively. The intuition here is that the labeling “names” the partitions: $\alpha_i = \alpha_j$ (resp. $\beta_i = \beta_j$) means that u_i and u_j (resp. v_i and v_j) are guessed to be in the same partition. The formula then checks that the guessed partitioning is compatible with the equivalence relation R (i.e. u_i and u_j have the same label iff $R(u_i, u_j)$ holds), and that the probability masses of the partitions assigned by configurations s and t satisfy the constraint given at (1).

For the labeling, define an auxiliary formula $\text{succ}(s, a, \bar{u})$:

$$\left(\bigwedge_{i < j} u_i \neq u_j \right) \wedge \left(\forall t. \delta(s, a, t) \neq 0 \Rightarrow \bigvee_i t = u_i \right),$$

stating that the successors of configuration w on action a are among the n distinct configurations $\bar{u}$. Note that a configuration may have fewer than n successors. In this case, we can set the rest of the variables to arbitrary distinct configurations. Given a labeling, we need to check that R is compatible with the guessed partitions, and that configurations s and t assign the same probability mass to the same partition. Let $\bar{k}$ be a labeling for configurations $\bar{s}$. To check that the partitioning induced by the labeling is compatible with R , we need to express the condition that $k_i = k_j$ if and only if $R(s_i, s_j)$ holds. This condition can be expressed as a formula

$$\text{compat}(\bar{s}, \bar{k}) := \bigwedge_{i < j} (R(s_i, s_j) \Leftrightarrow k_i = k_j).$$

Now, we can define the formula $\lambda(s, a, s')$ at (4) as

$$\begin{aligned} & \exists \bar{u}. \exists \bar{v}. \exists \bar{\alpha}. \exists \bar{\beta}. \text{succ}(s, a, \bar{u}) \wedge \text{succ}(s', a, \bar{v}) \\ & \wedge \text{compat}(\bar{u}, \bar{\alpha}) \wedge \text{compat}(\bar{v}, \bar{\beta}) \\ & \wedge \forall k. \left(\sum_{i: \alpha_i = k} \delta(s, a, u_i) = \sum_{i: \beta_i = k} \delta(s', a, v_i) \right). \end{aligned} \quad (6)$$

Thus, $\mathfrak{S}_R \models \lambda(s, a, s')$ iff $\sum_{t \in E} \delta(s, a, t) = \sum_{t \in E} \delta(s', a, t)$ holds for any equivalence class $E \in S/R$. It follows that the sentence Φ characterizes a bisimulation relation, and $\mathfrak{S}_R \models \Phi$ if and only if R is a bisimulation over $\mathfrak{S}$.

We proceed to show that Φ is expressible in FO_{reg} . Translating Φ_{eq} and ψ to FO_{reg} is straightforward. To translate λ , the key step is to express the summation inside (6). For this, we define a formula that performs iterated additions:

$$\begin{aligned} \chi(s, a, \bar{t}, \bar{\alpha}, k, z) &:= \exists \bar{p}. p_1 = 0 \wedge p_{n+1} = z \\ &\wedge \bigwedge_{1 \leq i \leq n} \chi'(s, a, t_i, \alpha_i, k, p_i, p_{i+1}), \end{aligned}$$

where $s \in S$, $\bar{t} \in S^n$, $k \in P$, $\bar{\alpha} \in P^n$, $\bar{p} \in P^{n+1}$, and

$$\begin{aligned} \chi'(s, a, t, \kappa, k, x, y) &:= (\kappa \neq k \wedge x = y) \vee \\ &(\kappa = k \wedge \exists p. \delta(s, a, t) = p \wedge (x + p = y)). \end{aligned}$$

The formula $\bigwedge_{1 \leq i \leq n} \chi'(s, a, t_i, \alpha_i, k, p_i, p_{i+1})$ effectively sums up the weights in $\{\delta(s, a, t_i) : \alpha_i = k, 1 \leq i \leq n\}$ and stores the result in p_{n+1} . Thus, given $k \in P$, $\bar{u}, \bar{v} \in S^n$ and $\bar{\alpha}, \bar{\beta} \in P^n$, we have

$$\sum_{i: \alpha_i = k} \delta(s, a, u_i) = \sum_{i: \beta_i = k} \delta(s', a, v_i)$$

if and only if

$$\mathfrak{S} \models \exists z. \chi(s, a, \bar{u}, \bar{\alpha}, k, z) \wedge \chi(t, a, \bar{v}, \bar{\beta}, k, z),$$

and the definition at (6) can be equivalently written as

$$\begin{aligned} & \exists \bar{u}. \exists \bar{v}. \exists \bar{\alpha}. \exists \bar{\beta}. \text{succ}(s, a, \bar{u}) \wedge \text{succ}(s', a, \bar{v}) \\ & \wedge \text{compat}(\bar{u}, \bar{\alpha}) \wedge \text{compat}(\bar{v}, \bar{\beta}) \\ & \wedge \forall k. \exists z. \chi(s, a, \bar{u}, \bar{\alpha}, k, z) \wedge \chi(s', a, \bar{v}, \bar{\beta}, k, z). \end{aligned}$$

Consequently, the sentence Φ at (4) is definable in $\text{FO}(\mathfrak{S}_R)$. By Proposition 2, it is decidable to check $\mathfrak{S}_R \models \Phi$ when both $\mathfrak{S}$ and R are regular. This concludes our proof. $\square$

Example 5. Consider the regular WTS from Example 3. Note that the configurations dXZ and cX are bisimilar. This fact can be shown using a bisimulation relation with equivalence classes $\{dX^kZ\} \cup \{dw : w \in \{X, X'\}^k\}$ and $\{bX^{k+2}Z\} \cup \{cYw : w \in \{X, X'\}^{k+1}\}$ for all $k \geq 0$. This bisimulation relation is definable as the reflexive and symmetric closure of a regular relation R , where $(v, u) \in R$ if and only if

$$\begin{aligned} & (v \in dX^*Z \wedge u \in c(X + X')^* \wedge |v| = |u| + 1) \\ & \vee (v \in c(X + X')^* \wedge u \in c(X + X')^* \wedge |v| = |u|) \\ & \vee (v \in bX^*Z \wedge u \in cY(X + X')^* \wedge |v| = |u| + 1) \\ & \vee (v \in cY(X + X')^* \wedge u \in cY(X + X')^* \wedge |v| = |u|). \end{aligned}$$

The formula $\lambda(s, a, s')$ at (6) checks this bisimulation relation for all states. To see the formula in action, fix two bisimilar configurations cX and dXZ . In the WTS, cX has three successors, cYX , cYX' , and c , with probabilities 0.3, 0.2, and 0.5, respectively; dXZ has two successors, $bXXZ$ and dZ , each with probability 0.5. These successors form two equivalence classes $\{dZ, c\}$ and $\{bXXZ, cYX, cYX'\}$. To satisfy formula λ , let $s = cX$ with successors $u_1 = cYX$, $u_2 = cYX'$, $u_3 = c$. Let $s' = dXZ$ with successors $v_1 = bXXZ$, $v_2 = dZ$, and v_3 being any configuration not equal to v_1 and v_2 . To label these successors, we can set $\alpha_3 = \beta_2 = 1$, $\alpha_1 = \alpha_2 = \beta_1 = 2$, and β_3 to an arbitrary number other than 1 and 2. Now, for $k \notin \{1, 2\}$, $\sum_{i: \alpha_i=k} \delta(s, a, u_i) = \sum_{i: \beta_i=k} \delta(s', a, v_i)$ yields $0 = 0$. For $k = 1$, it yields $\delta(s, a, u_3) = \delta(s', a, v_2) = 0.5$. For $k = 2$, it yields $\delta(s, a, u_1) + \delta(s, a, u_2) = 0.3 + 0.2 = \delta(s', a, v_1) = 0.5$. Thus, $\lambda(s, a, s')$ confirms that cX and dXZ are bisimilar.

Theorem 4 leads to the following decidability result.

Theorem 6. Given a regular WTS $\mathfrak{S}$ and a regular relation $E \subseteq S \times S$, there exists a procedure that finds either a non-bisimilar pair $(u, v) \in E$, or a regular bisimulation relation R over $\mathfrak{S}$ such that $E \subseteq R$. Furthermore, the procedure terminates if and only if E contains a non-bisimilar pair or E is a subset of a regular bisimulation relation.

Proof. It suffices to give two semi-procedures, one checking the existence of R and the other finding a non-bisimilar pair (v, w) in E . By Theorem 4, we can enumerate all plausible regular relations R and check that R is a bisimulation over $\mathfrak{S}$. The inclusion of E in R is a first-order property and thus can be checked effectively as well. To see that non-bisimulation is recursively enumerable, let $\mathfrak{S}_v$ denote the tree-structured WTS induced by unfolding $\mathfrak{S}$ from configuration v , and let $\mathfrak{S}_v^d$ denote the finite WTS induced by restricting $\mathfrak{S}_v$ to up to d steps from v . By Proposition 1, two configurations v, w of $\mathfrak{S}$ are non-bisimilar if and only if there is some PML formula ϕ such that $\mathfrak{S}_v \models \phi$ and $\mathfrak{S}_w \not\models \phi$. Since $\mathfrak{S}$ is bounded branching, ϕ can be checked by examining a finite number of configurations. Thus, there exists $d < \infty$ such that v, w are non-bisimilar if and only if $\mathfrak{S}_v^d \models \phi$ and $\mathfrak{S}_w^d \not\models \phi$. It follows that we can locate a non-bisimilar pair by enumerating and checking all pairs $(v, w) \in E$, distances $d \in \mathbb{N}$, and PML formulas ϕ over actions A and weights P . This procedure is effective by Proposition 1, which concludes the proof. $\square$

IV. LEARNING-BASED BISIMULATION SYNTHESIS

While Theorem 6 provides a brute-force method for discovering a regular bisimulation relation, more efficient strategies are needed to identify nontrivial bisimulations in practice. To address this challenge, we propose a learning-based approach for computing bisimulations on weakly finite regular WTSs. A transition system is *weakly finite* [67] if each configuration can reach only a finite number of configurations. Notably, the WTS underlying a parameterized system is weakly finite, as every configuration belongs to a finite instance of the system and thus has access to only finitely many configurations. This local finiteness allows us to effectively compute bisimulations even though the entire system has infinitely many states.

Specifically, our algorithm receives as input a weakly finite regular WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$, along with a regular set $I := \{s_n\}_{n \in \mathbb{N}} \subseteq S$ of initial configurations. The WTS $\mathfrak{S}$ is a regular presentation of a parameterized system $\{P_n\}_{n \in \mathbb{N}}$ such that each P_n starts in the configuration $s_n \in I$. Given a regular set $E \subseteq S \times S$ of the bisimilar pairs to verify, our algorithm employs active automata learning to synthesize a regular bisimulation relation R such that $R \supseteq E$. The learning process computes such a relation by systematically exploring bisimilar and non-bisimilar pairs within the system. A crucial requirement for this process is the ability to determine whether two configurations are bisimilar. For a parameterized system, this task amounts to computing bisimulations over a finite system instance, enabling the use of well-established verification tools from the literature [33].

A. Active automata learning

Automata learning [59]–[61] attempts to infer a DFA for a regular language whose definition is not directly accessible. In *active learning*, this inference is achieved by making queries to a “teacher” who has knowledge of the target regular language, say L . This teacher can respond to two types of queries. The first is membership query $Mem(w)$, asking whether a word w belongs to L . The second is equivalence query $Equ(\mathcal{A})$, asking whether the language $L(\mathcal{A})$ recognized by a DFA $\mathcal{A}$ is identical to L . An active learning algorithm performs the inference iteratively. In each iteration, it makes membership queries to gather information about L . Based on the answers, it builds a hypothesis DFA $\mathcal{A}_{hyp}$ and checks whether $\mathcal{A}_{hyp}$ is a solution through an equivalence query. If it is a solution, the algorithm terminates. If not, the teacher provides a word u as a counterexample, which the algorithm will utilize to refine its hypothesis for the next iteration.

Below, we elaborate on an active learning algorithm proposed by Rivest and Schapire [60], which is an improved version of *L-star* [59]. The algorithm’s foundation builds upon a seminal theorem from Myhill and Nerode [68].

Proposition 7 ([68]). *Given a regular language L , define a relation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ such that $x \equiv_L y$ if and only if $\forall z \in \Sigma^*. xz \in L \Leftrightarrow yz \in L$. Then the following are true:*

- *The relation $\equiv_L$ defines an equivalence relation. The number of distinct equivalence classes produced by this relation corresponds exactly to the number of states in the minimal DFA that can recognize the language L .*

- Any minimal DFA recognizing L is isomorphic to the following DFA: (i) each equivalence class $[x]$ is a state; (ii) the starting state is $[\varepsilon]$; (iii) state transitions are $[x] \rightarrow [xa]$ for $a \in \Sigma$; (iv) the accepting states are $[x]$ for $x \in L$.

Specifically, in the minimal DFA recognizing L , two words x and y are associated with the same state if and only if no suffix z can differentiate between them. In other words, x and y belong to different states in the minimal DFA if and only if there exists a suffix z' such that $xz' \in L$ and $yz' \notin L$.

Algorithm 1: Active automata learning

Input: A teacher that answers $Mem(w)$ and $Equ(\mathcal{A})$ about a target regular language L

Output: A minimal DFA recognizing L

```

1 Initialize the observation table  $(W, D, T)$ ;
2 repeat
3   while  $(W, D, T)$  is not closed do
4     Find a pair  $(x, a) \in W \times \Sigma$  such that
        $\forall y \in W : row_D(xa) \neq row_D(y)$ . Extend  $W$ 
       to  $W \cup \{xa\}$  and update  $T$  using membership
       queries accordingly;
5   Build a candidate DFA  $\mathcal{A}_{hyp} = (W, \Sigma, \delta, \lambda, F)$ ,
       where  $\delta = \{(s, a, s') : s, s' \in W \wedge row_D(sa) = row_D(s')\}$ , the empty string  $\lambda$  is the initial state,
       and  $F = \{s : T(s) = \top \wedge s \in W\}$ ;
6   if  $Equ(\mathcal{A}_{hyp}) = (false, w)$ , where
        $w \in L(\mathcal{A}_{hyp}) \ominus L$  then Analyse  $w$  and add a
       suffix of  $w$  to  $D$ ;
7 until  $Equ(\mathcal{A}_{hyp}) = true$ ;
8 return  $\mathcal{A}_{hyp}$  as the minimal DFA for  $L$ ;
```

Algorithm 1 presents Rivest and Schapire’s version of L-star (see also [53]). It maintains the equivalence classes induced by $\equiv_L$ using a *observation table* (W, D, T) , where W is a set of words representing the identified states of the minimal DFA, D is a set of suffix words distinguishing the states of the minimal DFA, and T is a mapping from $(W \cup (W \cdot \Sigma)) \cdot D$ to $\{\top, \perp\}$, such that $T(w) = \top$ iff $w \in L$. We write $row_D(x) = row_D(y)$ to indicate $\forall z \in D. T(xz) = T(yz)$, meaning that states associated with the words x and y cannot be distinguished using only words in the set D as suffix words. Observe that $x \equiv_L y$ implies $row_D(x) = row_D(y)$ for all $D \subseteq \Sigma^*$. We say that an observation table is *closed* iff it holds that $\forall x \in W. \forall a \in \Sigma. \exists y \in W. row_D(xa) = row_D(y)$. Intuitively, with a closed table, every state can find its successors with respect to all symbols in Σ . Initially, $W = D = \{\lambda\}$, and $T(w) = Mem(w)$ for all $w \in \{\lambda\} \cup \Sigma$.

By construction, two words x, y satisfying $x \equiv_L y$ can never be simultaneously contained in the set W . When the equivalence query $Equ(\mathcal{A})$ is false, the teacher provides a counterexample $w \in L(\mathcal{A}_{hyp}) \ominus L$, namely the symmetric difference between $L(\mathcal{A}_{hyp})$ and L . The algorithm then performs a binary search over w to find a suffix e of w such that $row_D(xa) = row_D(y)$ and $row_{D \cup \{e\}}(xa) \neq row_{D \cup \{e\}}(y)$ hold for some $x, y \in W$ and $a \in \Sigma$. By extending D to

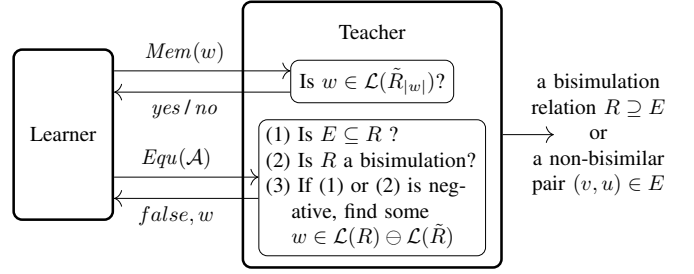


Fig. 2. An overview of using automata learning to synthesize a bisimulation relation $R \supseteq E$. Here, $\ominus$ denotes symmetric set difference, $\tilde{R}$ is the greatest bisimulation relation, $\tilde{R}_n$ is $\tilde{R}$ restricted to configurations of size n , and R denotes the relation represented by automata $\mathcal{A}$.

$D \cup \{e\}$, the algorithm can identify at least one more state needed to recognize the target language L .

Proposition 8 ([60]). *Algorithm 1 can identify a minimal DFA $\mathcal{A}$ that recognizes the language L . This process requires no more than n equivalence queries and $n^2 + n \cdot |\Sigma| + n \log k$ membership queries, where n represents the number of states in $\mathcal{A}$ and k is the length of the longest counterexample provided by the teacher.*

To see why this proposition is true, note that the algorithm’s behavior is governed by two key principles. First, each negative response to an equivalence query extends the learned DFA by at least one state. Since the Myhill-Nerode theorem constrains the candidate DFA’s size to n states, the number of equivalence queries is at most n . Second, to fully populate the observation table, the algorithm requires at most $n(n + n|\Sigma|)$ membership queries. Since the teacher provides at most n counterexamples, and the algorithm employs binary search to analyze counterexamples of lengths at most k , the number of membership queries is bounded by $n^2 + n \cdot |\Sigma| + n \log k$.

B. Learning regular bisimulations

We now explain how to employ active automata learning to compute regular bisimulations for weakly finite regular WTSs. To this end, we first describe a learning procedure under the so-called length-preserving assumption.

Definition 9 (Length-preserving WTS). We say that a WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ is defined over alphabet Σ if S is a subset of Σ^* . A WTS $\mathfrak{S}$ defined over Σ is *length-preserving* if $\delta(s, a, t) > 0$ only when $|s| = |t|$.

Fix a length-preserving regular WTS $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ over Σ , and a regular relation $E \subseteq S \times S$. We describe how to learn a regular bisimulation $R \supseteq E$ using the L-star algorithm. Since L-star requires the target language to be unique, we aim to infer the *greatest* bisimulation, which is the union of all bisimulations. Denote the target language as $\mathcal{L}(\tilde{R})$, the language representation of the greatest bisimulation $\tilde{R}$ over $\mathfrak{S}$. By the length-preserving assumption, a configuration can only reach configurations of the same size. Thus, we can write $\mathcal{L}(\tilde{R}) = \bigcup_{n \geq 1} \mathcal{L}(\tilde{R}_n)$ such that $\tilde{R}_n \subseteq \Sigma^n \times \Sigma^n$ is the greatest bisimulation on $\mathfrak{S}$ restricted to configurations of size n .

L-star needs a teacher for answering membership queries $Mem(w)$ and equivalence queries $Equ(\mathcal{A})$. Below, we explain how to resolve these queries to learn a regular bisimulation.

a) *Membership queries $Mem(w)$* : The teacher checks whether $w = v \otimes u$ for some $v, u \in \Sigma^*$ such that (v, u) is contained in the greatest bisimulation $\tilde{R}$. Since $\mathfrak{S}$ is length-preserving, this amounts to checking whether (v, u) is contained in $\tilde{R}_{|w|}$. As $\tilde{R}_{|w|}$ is defined over a finite-state system, the teacher can answer this query by computing the fixed-length bisimulation $\tilde{R}_{|w|}$ and checking if (v, u) is in $\tilde{R}_{|w|}$.

b) *Equivalence queries $Equ(\mathcal{A})$* : The teacher checks whether $\mathcal{A}$ represents a bisimulation including E . Let R denote the regular relation represented by $\mathcal{A}$. To answer this query, the teacher essentially checks that R satisfies the formula Φ at (4), see Algorithm 2. It first finds a configuration pair violating the formula. If no such pair exists, R is a bisimulation containing E . Suppose a pair (v, u) of size n is found. Then, it is a witness of either $E \not\subseteq R$ or $G_R \not\models \Phi$. If $(v, u) \in E \setminus \tilde{R}_n$, we have found a non-bisimilar pair in E . Otherwise, (v, u) falls in the symmetric difference of R and $\tilde{R}$. It is a valid counterexample since the learner attempts to learn the greatest bisimulation. The teacher thus reports (v, u) for the equivalence query.

Algorithm 2: Answering equivalence queries

Input: A length-preserving regular WTS $\mathfrak{S}$ over Σ ;
 Regular relations $R, E \subseteq \Sigma^* \times \Sigma^*$;
Result: (v, u) : a non-bisimilar pair in E ;
 R : a bisimulation on $\mathfrak{S}$ such that $E \subseteq R$;
 $(false, v \otimes u)$: (v, u) violates $R = \tilde{R}$;

- 1 Check whether $E \subseteq R$, and whether $\mathfrak{S}_R \models \Phi$ holds in the sense of Theorem 4;
- 2 **if** there exists a counterexample (v, u) **then**
- 3 Let $n := \max\{|v|, |u|\}$;
- 4 Compute $\tilde{R}_n$, the greatest bisimulation $\tilde{R}$ restricted to configurations of size n ;
- 5 **if** $(v, u) \in E \setminus \tilde{R}_n$ **then**
- 6 Output (v, u) as a non-bisimilar pair in E ;
- 7 Abort the learning process;
- 8 **else**
- 9 **return** $(false, v \otimes u)$;
- 10 **else**
- 11 Output R as a bisimulation relation;
- 12 Abort the learning process;

Figure 2 outlines our learning procedure for regular bisimulations. This procedure might diverge since bisimulations are not generally computable. It is guaranteed to terminate when the greatest bisimulation $\tilde{R}$ is regular, though the produced bisimulation is not necessarily equal to $\tilde{R}$.

Theorem 10 (Correctness). *When the learning procedure terminates, it provides the correct answer regarding whether all configuration pairs in E are bisimilar.*

Proof. Note that the learning procedure terminates only when the teacher pinpoints a non-bisimilar pair $(v, u) \in E$ or a

bisimulation relation R such that $E \subseteq R$. Therefore, the procedure always outputs a correct answer on termination. $\square$

Theorem 11 (Termination). *When the greatest bisimulation $\tilde{R}$ is regular, the learning procedure is guaranteed to terminate in at most m iterations, where m is the size of the minimal DFA recognizing $\mathcal{L}(\tilde{R})$.*

Proof. All counterexample words reported by the teacher are contained in the symmetric difference of $\mathcal{L}(\tilde{R})$ and the language recognized by $\mathcal{A}$. Thus, by Proposition 8, the learning procedure is guaranteed to terminate when $\mathcal{L}(\tilde{R})$ is regular. Moreover, if $\mathcal{L}(\tilde{R})$ can be recognized by a DFA of m states, the algorithm will terminate in at most m iterations. $\square$

Optimization with inductive invariants. A natural way to optimize the learning procedure is to consider a *regular* inductive invariant Inv such that Inv contains the set of reachable configurations. The optimization is done by simply replacing the greatest finite-length bisimulations $\tilde{R}_n$ with the greatest bisimulation $\tilde{R}'_n := \tilde{R}_n \cap (Inv \times Inv)$ on the inductive invariant Inv when answering the membership and equivalence query. Since $\tilde{R}'_n$ can be much smaller than $\tilde{R}_n$, replacing $\tilde{R}_n$ with $\tilde{R}'_n$ can lead to significant speed-ups. Note that a bisimulation R' on Inv can be extended to a bisimulation R on all configurations by setting $R := R' \cup \{(v, v) : v \notin Inv\}$, which is regular when R' is regular. The inductive invariant Inv may be specified manually or generated automatically [53].

C. Padding WTS for bisimulation learning

We proceed to show that it is possible to relax the length-preserving assumption in the previous section using *padding*. Specifically, if the maximal size of reachable configurations is known in initial configurations, then we can reduce bisimulation inference of a weakly finite WTS $\mathfrak{S}$ to that of a length-preserving WTS, which is essentially a padded version of $\mathfrak{S}$.

Definition 12 (Padded WTS). Let $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ be a WTS over alphabet Σ . For a word $s \in \Sigma^*$, define a language $Pad(s) := s \cdot \#^*$. Namely, $Pad(s) \subseteq \Sigma_\#^*$ is the set of words obtained by appending to s arbitrarily many blank symbols $\#$. We define the *padded variant* of $\mathfrak{S}$ as the length-preserving WTS $\mathfrak{S}_{pad} := \langle \tilde{S}, P, A, \tilde{\delta}, + \rangle$ over alphabet $\Sigma_\#$, where $\tilde{S} := \bigcup_{s \in S} Pad(s)$ and

$$\tilde{\delta}(s, a, t) := \begin{cases} \delta(s', a, t'), & s \in Pad(s'), t \in Pad(t'), |s| = |t|; \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, $\tilde{S}$ is obtained by padding configurations in S , and $\tilde{\delta}_a$ is the length-preserving restriction of δ_a on these padded configurations. Note that $\mathfrak{S}_{pad}$ is definable in FO_{reg} given a regular presentation of $\mathfrak{S}$. Hence, $\mathfrak{S}_{pad}$ is effectively regular for a regular WTS $\mathfrak{S}$. The following result shows that we can encode bisimilar configurations in $\mathfrak{S}$ such that the encoded configurations are bisimilar in $\mathfrak{S}_{pad}$ and vice versa.

Proposition 13. *Let $\mathfrak{S}$ be a weakly finite WTS over alphabet Σ , and $\mathfrak{S}_{pad}$ be the padded variant of $\mathfrak{S}$ over alphabet $\Sigma_\#$.*

Then a relation $E \subseteq \Sigma^* \times \Sigma^*$ consists of bisimilar pairs over $\mathfrak{S}$ if and only if $\tilde{E}$ consists of bisimilar pairs over $\mathfrak{S}_{\text{pad}}$, where

$$\tilde{E} := \bigcup_{(v,u) \in E} \left((\text{Pad}(v) \times \text{Pad}(u)) \cap (\Sigma_{\#} \times \Sigma_{\#})^{f(v,u)} \right)$$

and $f(v,u) := \max\{n(v), n(u)\}$ with $n(s) := \max\{|s'| : s' \text{ is reachable from } s \text{ in } \mathfrak{S}\}$.

Proof. Consider a pair of configurations $(v,u) \in E$ and let $n := \max\{n(v), n(u)\}$. Then $n < \infty$ since $\mathfrak{S}$ is weakly finite. Consider the pair of padded configurations $(v', u') \in \tilde{E} \cap (\text{Pad}(v) \times \text{Pad}(u))$ by padding v and u to size n . Observe that every bisimilar pair of configurations reachable from (v,u) in $\mathfrak{S}$ has its padded version in $\tilde{R}_n$ and vice versa. Therefore, v and u are bisimilar in $\mathfrak{S}$ if and only if $(v', u') \in \tilde{R}_n \subseteq \tilde{R}$. Similarly, for each $(v', u') \in \tilde{E}$, v' and u' are bisimilar in $\mathfrak{S}_{\text{pad}}$ if and only if their unpadded counterparts v and u are bisimilar in $\mathfrak{S}$. This concludes our proof. $\square$

By Proposition 13, checking that E consists of bisimilar pairs in $\mathfrak{S}$ amounts to checking that its padded version $\tilde{E}$ consists of bisimilar pairs in $\mathfrak{S}_{\text{pad}}$. To translate E to $\tilde{E}$, we need to know $n(v)$ and $n(u)$ (i.e., the largest sizes of the configurations reachable from v and u , respectively) for each pair $(v,u) \in E$. In practice, given a regular presentation of a parameterized system $\{P_n\}_{n \in N}$, we can often compute $\tilde{E}$ from E by encoding the parameter $n \in N$ using padding [48], [53], [54]. All the examples discussed in our case study can be adapted to this encoding.

V. A FRAMEWORK FOR ANONYMITY AND UNIFORMITY VERIFICATION

In this section, we introduce how probabilistic bisimulation can be employed to reason about anonymity and uniformity of Markov decision processes (MDPs) and Markov chains. Fix an MDP $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$. Recall that a path of $\mathfrak{S}$ is a sequence $s_0 \rightarrow_{a_1} s_1 \rightarrow_{a_2} \dots$ such that $\delta(s_{i-1}, a_i, s_i) \neq 0$ for each i . We will use $\pi(\mathfrak{S})$ to denote the set of *finite* paths of $\mathfrak{S}$, and use $\mathcal{D}_A$ to denote the set of probability distributions over A . An *adversary* $f : \pi(\mathfrak{S}) \rightarrow \mathcal{D}_A$ resolves the nondeterministic choices of $\mathfrak{S}$ and induces a WTS $\mathfrak{S}_f := \langle \tilde{S}, P, \tilde{A}, \tilde{\delta}, + \rangle$, where $\tilde{S} := \pi(\mathfrak{S})$ and $\tilde{A} := A \uplus \{\alpha\}$ with a dummy action α . The transition function $\tilde{\delta} : \tilde{S} \times \tilde{A} \times \tilde{S} \rightarrow P$ is defined such that for any paths $\pi := s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n$ and $\pi' := s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n \rightarrow_a s_{n+1}$ in $\pi(\mathfrak{S})$ (i.e., π' is a path extending π with one more transition $s_n \rightarrow_a s_{n+1}$), it holds that $\tilde{\delta}(\pi, a, \pi') = f(\pi)(a) \cdot \delta(s_n, a, s_{n+1})$, where $f(\pi)(a)$ denotes the probability of the adversary selecting action a given the path π . We stipulate that $f(\pi)(a) \neq 0$ only if $\delta(s_n, a, s) \neq 0$ for some $s \in S$, i.e., the adversary can only select those actions that lead to a successor configuration in $\mathfrak{S}$. We set $\tilde{\delta}(\pi, \alpha, \pi) = 1$ for every $\pi \in \tilde{S}$ ending with a terminal configuration of $\mathfrak{S}$. Intuitively, $\mathfrak{S}_f$ describes the behavior of $\mathfrak{S}$ under the adversary f by successively extending the paths of $\mathfrak{S}$ according to the actions selected by f . If a path reaches a terminal configuration, it will loop there through the dummy action α .

$\mathfrak{S}_f$ is a Markov chain since $\sum_{a \in A'} \sum_{\pi' \in \tilde{S}} \tilde{\delta}(\pi, a, \pi') = 1$ for every $\pi \in \tilde{S}$. The paths of $\mathfrak{S}_f$ induce a probability measure that can be formalized using standard cylinder construction [69]. Given a finite path $\pi := s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n$, we refer to $\text{tr}(\pi) := a_1 \dots a_n \in A^*$ as the *trace* of π . We call a set of traces $\mathcal{T} \subseteq A^*$ a *trace event*. Let Run_{π} be the set of infinite paths with prefix π . The probability of a trace event $\mathcal{T}$ with respect to a source state s is given by

$$\Pr^s(\mathcal{T} \mid \mathfrak{S}_f) := \Pr\left(\bigcup \{\text{Run}_{\pi} : \text{tr}(\pi) \in \mathcal{T}, s = \pi[0]\} \mid \mathfrak{S}_f\right).$$

For simplicity, we shall write $\Pr^s(\mathcal{T} \mid \mathfrak{S}_f)$ as $\Pr^s(\tau \mid \mathfrak{S}_f)$ when $\mathcal{T} = \{\tau\}$ for a single trace τ .

A. Anonymity verification

We will now describe how to verify anonymity properties using bisimulation. Fix a set $I \subset S$ of initial configurations. An MDP $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ is *anonymous to an adversary* f if for every initial configuration $s \in I$ and trace event $\mathcal{T}$, the probability $\Pr^s(\mathcal{T} \mid \mathfrak{S}_f)$ is solely determined by $\mathcal{T}$ (and thus independent of s). Intuitively, this means that the adversary cannot infer any information about a specific initial configuration by experimenting with the system and observing the traces. An adversary $f : \pi(\mathfrak{S}) \rightarrow \mathcal{D}_A$ is *observational* if $f(\pi) = f(\pi')$ whenever $\text{tr}(\pi) = \text{tr}(\pi')$. That is, the adversary has no access to the system's internal state and determines an action solely based on the trace observed thus far. An MDP is *anonymous* if it is anonymous to any observational adversary. The following result establishes a connection between anonymity and bisimilarity.

Theorem 14. *Let $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ be an MDP and f be an observational adversary. Suppose that $R \subseteq S \times S$ is a bisimulation over $\mathfrak{S}$. Then for any $(u, v) \in R$ and trace event $\mathcal{T}$, we have $\Pr^u(\mathcal{T} \mid \mathfrak{S}_f) = \Pr^v(\mathcal{T} \mid \mathfrak{S}_f)$. That is, u and v induce the same trace distribution under the intervention of f .*

Proof. Fix a trace event $\mathcal{T} \subseteq A^*$. We can assume w.l.o.g. that $\mathcal{T}$ is prefix-free. Indeed, if $\tau, \tau' \in \mathcal{T}$ and τ is a prefix of τ' , then we can remove τ' without changing the probability of $\mathcal{T}$. Thus, we have $\Pr^s(\mathcal{T} \mid \mathfrak{S}_f) = \sum_{\tau \in \mathcal{T}} \Pr^s(\tau \mid \mathfrak{S}_f)$, and it suffices to prove this theorem for the case $\mathcal{T} = \{\tau\}$. We prove it by induction on the length of τ . For the base case, suppose that $\tau = a \in A$. Since u and v are bisimilar, we have $\sum_{s \in S} \delta(u, a, s) = \sum_{s \in S} \delta(v, a, s)$. It follows that $\Pr^u(\tau \mid \mathfrak{S}_f) = f(u)(a) \cdot \sum_{s \in S} \delta(u, a, s) = f(v)(a) \cdot \sum_{s \in S} \delta(v, a, s) = \Pr^v(\tau \mid \mathfrak{S}_f)$ by the definition of observational adversary. For the induction step, suppose that the hypothesis holds for $|\tau| = n$. Consider a trace $\tau := a \cdot \tau'$ with $a \in A$ and $\tau' \in A^n$. For each $s \in S$, define an observational adversary $f_{s,a}$ such that

$$f_{s,a}(s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n) := f(s \rightarrow_a s_0 \rightarrow_{a_1} \dots \rightarrow_{a_n} s_n).$$

Then we have

$$\begin{aligned} \Pr^u(\tau \mid \mathfrak{S}_f) &= f(u)(a) \sum_{s \in S} \delta(u, a, s) \cdot \Pr^s(\tau' \mid \mathfrak{S}_{f_{u,a}}) \quad (\text{def.}) \end{aligned}$$

$$\begin{aligned}
&= f(u)(a) \sum_{[s] \in S/R} \left(\sum_{w \in [s]} \delta(u, a, w) \right) \cdot \Pr^s(\tau' \mid \mathfrak{S}_{f_{u,a}}) \\
&\quad \text{(hypo.)} \\
&= f(v)(a) \sum_{[s] \in S/R} \left(\sum_{w \in [s]} \delta(v, a, w) \right) \cdot \Pr^s(\tau' \mid \mathfrak{S}_{f_{v,a}}) \\
&\quad \text{(bisim.)} \\
&= f(v)(a) \sum_{s \in S} \delta(v, a, s) \cdot \Pr^s(\tau' \mid \mathfrak{S}_{f_{v,a}}) \\
&\quad \text{(hypo.)} \\
&= \Pr^v(\tau \mid \mathfrak{S}_f), \quad \text{(def.)}
\end{aligned}$$

where S/R denotes the set of equivalence classes induced by the bisimulation R , and $[s] := \{s' \in S : (s, s') \in R\}$ is the equivalence class represented by s . Therefore the hypothesis also holds for $|\tau| = n + 1$. The statement hence follows by mathematical induction. $\square$

Based on Theorem 14, we may verify the anonymity of an MDP $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$ as follows. Given a set $I \subseteq S$ of initial states, we specify a reference system $\mathfrak{S}' := \langle S, P, A, \delta', + \rangle$ such that the trace distribution of $\mathfrak{S}'$ is independent of specific initial states regardless of the intervention of any observational adversary f . We then find a bisimulation relation R between $\mathfrak{S}$ and $\mathfrak{S}'$ such that $R \cap (I \times I)$ coincides with the identity relation over I . When such a relation R is found, we can conclude that the trace distribution of $\mathfrak{S}_f$ is also independent of the initial states for any observational adversary f , thereby proving the anonymity property of $\mathfrak{S}$.

B. Uniformity verification

Bisimilarity preserves bounded termination: if two systems $\mathfrak{S}$ and $\mathfrak{S}'$ are bisimilar, then $\mathfrak{S}$ terminates in n steps iff $\mathfrak{S}'$ does. We can generalize this fact and use bisimilarity to establish uniform output distributions for probabilistic programs.

Definition 15 ([44]). A *probabilistic program* is defined by a triple $\mathcal{P} := (\mathfrak{S}, I, \{F_s\}_{s \in I})$, where $\mathfrak{S} := \langle S, P, \{a\}, \delta, + \rangle$ is a Markov chain with dummy action a , $I \subseteq S$ is a set of initial configurations, and $F_s \subseteq S$ is a set of final configurations for each $s \in I$. For simplicity, we will write $\mathfrak{S}$ as $\langle S, P, \delta \rangle$ with probabilistic transition function $\delta : S \times S \rightarrow P$.

Starting from an initial configuration $s \in I$, the probabilistic program $\mathcal{P}$ terminates when the Markov chain $\mathfrak{S}$ reaches some final configuration $s' \in F_s$. The *uniformity property* of $\mathcal{P}$ asserts that, from each initial configuration $s \in I$, the program has the same probability of reaching each final configuration in F_s on termination. In other words, the reachability distribution over F_s is uniform for each initial configuration $s \in I$.

For a Markov chain $\mathfrak{S} := \langle S, P, \delta \rangle$, let $\mathfrak{S}^{-1} := \langle S, P, \delta^{-1} \rangle$ be a WTS where $\delta^{-1}(s, t) := \delta(t, s)$ for all $s, t \in S$. The following result relates uniform reachability distribution of $\mathfrak{S}$ to bisimilarity over $\mathfrak{S}^{-1}$.

Theorem 16. *Let $\mathfrak{S} := \langle S, P, \delta \rangle$ be a Markov chain, $s_0 \in S$ be an initial configuration, and $F \subseteq S$ be a set of final configurations. Then s_0 has a uniform reachability probability over F if there exists a bisimulation relation $R \subseteq S \times S$ over $\mathfrak{S}^{-1}$ such that (i) $F \times F \subseteq R$, and (ii) s_0 is bisimilar only to itself with respect to R .*

Proof. Suppose R is a bisimulation satisfying conditions (i) and (ii). Let $S_n := \{s \in S : \text{there is a path } \pi \text{ from } s_0 \text{ to } s \text{ such that } |\pi| = n\}$. It is not hard to show (e.g., by induction on n) that if $(u, v) \in R$, then $u \in S_n \Leftrightarrow v \in S_n$ for $n \geq 0$. Furthermore, for $s \in S$ and $n \geq 0$, define $p_n(s) := \sum \{\Pr(Run_\pi) : \pi \text{ is a path from } s_0 \text{ to } s \text{ with } |\pi| = n\}$, i.e., $p_n(s)$ is the probability that $\mathfrak{S}$ reaches s from s_0 after making precisely n transitions. Now, we claim that for each $(u, v) \in R$, it holds that $p_n(u) = p_n(v)$ for $n \geq 0$. We prove this claim by induction on n . The base case follows from condition (ii), since $n = 0$ implies that $u = v = s_0$. For the induction step, suppose that $(u, v) \in R$. Since $u \in S_{n+1} \Leftrightarrow v \in S_{n+1}$, either both u and v are not in S_{n+1} , or both u and v are in S_{n+1} . In the former case, we have $p_{n+1}(u) = p_{n+1}(v) = 0$. In the latter case, we have

$$\begin{aligned}
p_{n+1}(u) &= \sum_{s \in S} p_n(s) \cdot \delta^{-1}(u, s) && \text{(def.)} \\
&= \sum_{[s] \in S/R} p_n(s) \cdot \sum_{t \in [s]} \delta^{-1}(u, t) && \text{(hypo.)} \\
&= \sum_{[s] \in S/R} p_n(s) \cdot \sum_{t \in [s]} \delta^{-1}(v, t) && \text{(bisim.)} \\
&= \sum_{s \in S} p_n(s) \cdot \delta^{-1}(v, s) && \text{(hypo.)} \\
&= p_{n+1}(v). && \text{(def.)}
\end{aligned}$$

Here, S/R denotes the set of equivalence classes induced by the bisimulation R , and $[s] := \{s' \in S : (s, s') \in R\}$ is the equivalence class represented by s . By mathematical induction, we see that the hypothesis $p_n(u) = p_n(v)$ holds for all $n \geq 0$. Finally, note that $\sum_{n \geq 0} p_n(s)$ is the reachability probability of configuration s . By condition (i), we have $(u, v) \in R$ for all $u, v \in F$. Thus, $p_n(u) = p_n(v)$ holds for all $u, v \in F$ and $n \geq 0$. It follows that all configurations in F have the same reachability probability. $\square$

We say that a probabilistic program $\mathcal{P} := (\mathfrak{S}, I, \{F_s\}_{s \in I})$ is *regular* if $\mathfrak{S}$ and I are regular, and the relation $E := \{(v, u) \in S \times S : v, u \in F_s \text{ for some } s \in I\}$ is regular (which holds when, for example, there is a regular relation $H \subseteq S \times S \times S$ such that $(v, u) \in F_s$ iff $(s, v, u) \in H$). By Theorem 16, checking uniformity of $\mathcal{P}$ amounts to finding a bisimulation relation R over $\mathfrak{S}^{-1}$ satisfying $E \subseteq R$ and $R \cap (I \times I) = \{(s, s) : s \in I\}$. Since $\mathfrak{S}^{-1}$ is effectively regular when $\mathfrak{S}$ is regular, for a regular probabilistic program $\mathcal{P}$, we can use Theorem 6 to check whether a regular relation R is a proof for the uniformity of $\mathcal{P}$.

Some remarks are in order. In our analysis of uniformity thus far, we have implicitly assumed that the final configurations are reachable with probability 1. Indeed, uniformity holds trivially for unreachable configurations since these configurations are “reached” with the same zero probability. In software model checking, it is common to break down the verification of a correctness property into separate verification tasks for partial correctness and termination [34], [43], [44]. Similarly, for assertions like “the final configurations are reached uniformly at random,” a uniformity proof only provides a partial

correctness guarantee. We need to additionally check *almost-sure termination* to establish total correctness. Almost-sure termination of probabilistic programs can be verified using a wide array of automated techniques in the literature (e.g., [44], [48], [67], [70]) and is not the focus of this work.

C. Extensions

We briefly discuss how to extend our previously introduced formalism for checking probability equivalence and handling models with parametric transition probabilities.

a) Probability equivalence: Although we have assumed that a probabilistic program can reach the relevant final configurations with probability 1, this assumption is not essential. In fact, for a probabilistic program $(\mathfrak{S}, I, \{F_s\}_{s \in I})$, the uniformity properties we verify are *conditional*: the reachability distribution over F_s is uniform whenever the program reaches F_s from the initial configuration $s \in I$. We can exploit this fact to capture and verify a property closely related to uniformity, called *probability equivalence*.

More precisely, fix an initial configuration $s_0 \in I$ and an index set J . Given a set S_j of final configurations for some $j \in J$, let $\mathcal{P}_j$ be the set of paths π from s_0 to S_j . Define $\mathcal{P} := \bigcup_{j \in J} \mathcal{P}_j$. Let $X : \mathcal{P} \rightarrow J$ be a random variable such that $X(\pi) = j$ indicates $\pi \in \mathcal{P}_j$. Then the events $\{\mathcal{P}_j\}_{j \in J}$ have the same probability if and only if the conditional probability $\Pr(X \mid \mathcal{P})$ yields a uniform distribution over J on program termination, which can be directly checked within our framework by augmenting $\mathfrak{S}$. We will demonstrate how this extension applies through probability equivalence checks in random walks, random sums, and the ballot theorem.

b) Parametric probabilities: Fix $\mathfrak{S} := \langle S, P, A, \delta, + \rangle$. For $\mathfrak{S}$ to be regular, the addition operator $+$ must be regular-presentable. It suffices to define the axioms of addition, such as associativity, commutativity, and identity element, in the first-order theory of $\mathfrak{S}$. Based on this observation, we may generalize our proof rule of bisimulation to support (universal) parametric probabilities as follows. We first extend $\mathfrak{S}$ with a regular set Q of symbols to represent parametric probabilities, along with a new addition operator $+$ defined over $Q \cup P$. We then encode the axioms of addition in $\text{FO}(\mathfrak{S})$ such that two probability masses are equal iff they are equal for *any* feasible instantiation of the parametric probabilities. Such encoding is possible as $\mathfrak{S}$ is bounded branching, e.g., when the branching is bounded by 2, we can define the commutativity axiom by

$$\forall x_1. \forall x_2. (x_1 \in (Q \cup P) \wedge x_2 \in (Q \cup P)) \Rightarrow (x_1 + x_2 = x_2 + x_1).$$

In this case, given a bisimulation R , if $u \xrightarrow{1+q}_a E$ holds for an equivalence class $E \in S/R$ and a parametric probability $q \in Q$, then $(u, v) \in R$ implies that $v \xrightarrow{1}_a t$ and $v \xrightarrow{q}_a t'$ hold for some $t, t' \in E$. We illustrate this extension in our evaluation by proving the anonymity of the crowds protocol.

a) Dining cryptographers protocol: As we introduced in Section I, this protocol anonymously computes the parity of the participants' secret bits. We model it as an MDP, allowing the adversary to choose the random bits used by the observing participant k . A configuration is encoded as (s, w) such that s is a control state, $w \in \{0, 1\}^n$ is a bit-vector, and n is the participant number. At an initial configuration, each $w[i]$ represents the secret x_{k+i} held by participant $k+i$. Thus, the interpretation of w depends on who is observing. The system has three types of transitions: the observer choosing head or tail (via actions H or T); a non-observer tossing head or tail with probability 0.5 (both via action X); a participant announcing zero or one (via actions 0 or 1). The random bits computed by the observer are visible as actions H and T, while the random bits computed by the other participants are hidden as the dummy action X.

Starting from an initial configuration (s_I, w) with $|w| = n$, a maximal trace consists of n update followed by n announcements. Let b_i denote the random bit computed by participant $k+i$. For $i \in \{0, \dots, n-1\}$, the i -th update changes the value of $w[j]$ to $w[j] \oplus b_i$ for $j \in \{i, i+1\}$. A configuration (s', w') reached from (s, w) after n updates would satisfy $w'[i] = x_{k+i} \oplus b_i \oplus b_{i-1}$ for $i \in \{0, \dots, n-1\}$. The trace then “prints out” w' by going through n announcements via actions $a_0, \dots, a_{n-1}$, where a_i is 1 if $w'[i] = 1$ and a_i is 0 if $w'[i] = 0$. Here, a_i is interpreted as the announcement made by participant $k+i$. To prove anonymity, we define a reference system where the announcements in a maximal trace starting from an initial configuration (s, w) are uniformly distributed over $\{(a_0, \dots, a_{n-1}) : f(\bar{a}) = f(w), a_0 = w[0] \oplus b_0 \oplus b_{n-1}\}$. In this way, the distribution of the announcements is independent of the initial configuration once the values of x_k, b_0, b_{n-1} , and $f(\bar{a}) := a_0 \oplus \dots \oplus a_{n-1}$ (i.e., the information observed by participant k) are fixed. We then compute a bisimulation between the original system and the reference system and establish the desired anonymity.

In our evaluation, we also examine a generalized version of dining cryptographers where the secret messages $x_0, \dots, x_{n-1}$ are bit-vectors of a parameterized size $m \geq 1$. Unfortunately, the set I of initial configurations is not regular in this setting. To construct a regular model, we allow a configuration to encode secret messages of different sizes. We then devise the transition system such that an initial configuration (s, w) can properly complete the protocol (i.e., it yields a trace containing the n announcements $a_0, \dots, a_{n-1}$) if and only if the messages encoded in w have the same size. The resulting MDP $\mathfrak{S}'$ overapproximates the MDP $\mathfrak{S}$ of the generalized protocol, as the traces of the former subsume those of the latter. Thus, we can verify $\mathfrak{S}'$ to establish the anonymity of $\mathfrak{S}$.

b) Crowds protocol: The crowds protocol [50] enhances anonymous data transmission by randomly routing a message within a group of users before it reaches the true destination. When a user wants to send a message to another user, she transmits it to a random member of the group. If the receiver is corrupt, he will disclose the sender's identity. Otherwise, he will forward the message to the final destination with probability p , and to another random member with probability $1 - p$, extending the route for one more step.

VI. CASE STUDIES

```

def DCP(bool b1, b2, bv[N])
  bv[0] = bv[0] ⊕ b1 ⊕ b2
  bv[1] = bv[1] ⊕ b1
  bv[N-1] = bv[N-1] ⊕ b2
  for i in 2 ... N-1
    int b = coin()
    bv[i] = bv[i] ⊕ b
    bv[i-1] = bv[i-1] ⊕ b
  print bv

def RandWalk(int N)
  int pos = N
  while 0 < pos < 2N
    int d = coin()
    pos = pos - 2d + 1
  print pos

def RandSum(int N)
  int sum = 0
  int p = 0
  for i in 1...2N
    p = coin()
    sum = sum + p
  print sum

def NaiveRNG(int N)
  int x = 0
  int y = 1
  while true
    x = 2x + coin()
    y = 2y
    if y ≥ N
      if x < N
        break
      else
        x = 0
        y = 1
  print x

def KnuthYaoRNG(int N)
  int x = 0
  int y = 1
  while true
    x = 2x + coin()
    y = 2y
    if y ≥ N
      if x < N
        break
      else
        x = x - N
        y = y - N
  print x

def Ballot(int N)
  int a = 0
  int b = 0
  int vote = coin()
  int first = vote
  bool tied = false
  for i = 1 ... N
    if vote = 1
      a = a + 1
    else
      b = b + 1
  if a = b
    tied = true
  vote = coin()
  print (tied, first)

```

Fig. 3. Example probabilistic programs for verifying probability uniformity and equivalence. We use $\oplus$ to denote the XOR Boolean operator, and use N to represent an unbounded natural number parameter. The function `coin()` returns 0 or 1 uniformly at random.

This protocol offers various levels of anonymity guarantees, many of which rely on the following fact: all potential senders are equally likely to be exposed by a corrupt user after the first forwarding. We verify this fact as an anonymity property of an MDP. Each configuration is encoded as (s, x, y, n) , where s is a control state, n is the crowd size, $x \in \{1, \dots, n\}$ is the user currently holding the message, and $y \in \{1, \dots, n\}$ is the original sender. The initial configurations are of the form (s_I, i, i, n) for $1 \leq i \leq n$. The routing process operates in rounds. In each round, the MDP randomly selects a user $z \in \{1, \dots, n\}$ to receive the message from x . Then, an adversary determines whether the receiver z is corrupt. If the receiver is corrupt, the MDP emits action T when $x = y$, indicating that the original sender is revealed, and emits action F when $x \neq y$, indicating that a false sender is revealed. If the receiver is not corrupt, he forwards the message to the final destination with probability p , and proceeds to the next round with probability $1 - p$ and a dummy action X . A trace extends indefinitely until the message is sent to a corrupt user or to the final destination. We verify that all initial configurations in the same crowd are bisimilar, meaning that all potential senders are equally likely to be exposed by an external observer.

c) Grades protocol: The grades protocol [49], [71] is a multi-party computation algorithm aiming to securely compute the sum of the secrets held by the participants. The setting of the protocol is pretty similar to that of the dining cryptographers: given two parameters $n \geq 3$ and $g \geq 2$, we have n participants where each participant i holds an *integer* secret $x_i \in \{0, \dots, g-1\}$. The goal is to compute $x_0 + \dots + x_{n-1}$ without revealing information about the individual secrets. Define $m := (g-1) \cdot n + 1$. The protocol has two steps: (i) Each two adjacent participants i and $i+1$ compute a random number $y_i \in \{0, \dots, m-1\}$; (ii) Each participant i announces $a_i := (x_i + y_i - y_{i-1}) \bmod m$ to the other participants. Thus, the participants can compute $h(\bar{a}) := a_0 + \dots + a_{n-1} \bmod m$, which is equal to $x_0 + \dots + x_{n-1}$ as the y_i 's are canceled out due to the modulo. The anonymity property asserts that no participant can infer the secrets held by the other participants from the observed information. We consider a variant of grades protocol where $m = 2^k \geq (g-1) \cdot n + 1$ with parameters k , n , and g . Clearly, the original protocol's anonymity and correctness properties also hold for this variant. Since m is a power of 2, the MDP of this protocol is similar to the

one we constructed for the generalized dining cryptographers, except that the xor operations are replaced with additions and negations when appropriate. A reference system is specified such that the announcements observed by participant k are uniformly distributed over $\{(a_0, \dots, a_{n-1}) : h(\bar{a}) = h(\bar{x}), a_0 = x_k + y_0 + y_{n-1} \bmod m\}$. We then establish the anonymity property by computing a bisimulation between the original system and the reference system.

d) Dining cryptographers protocol (version 2): In this example, we verify the anonymity of the dining cryptographers by formulating it as a probabilistic program. Let $x_0, \dots, x_{n-1}$ denote the secret bits and $b_0, \dots, b_{n-1}$ denote the random bits computed during execution, such that the announcement made by participant i is $a_i := x_i \oplus b_i \oplus b_{i-1}$. To show that an observing participant k cannot infer information beyond x_k , b_k , b_{k-1} , and $f(\bar{x}) := x_0 \oplus \dots \oplus x_{n-1}$, it suffices to check that the random vector $(a_0, \dots, a_{n-1})$ is uniformly distributed over $\{\bar{a} \in \{0, 1\}^n : f(\bar{a}) = f(\bar{x}), a_k = x_k \oplus b_k \oplus b_{k-1}\}$. Based on this, we model the protocol as the program DCP in Figure 3. The program has arguments $b_1, b_2 \in \{0, 1\}$ and $bv \in \{0, 1\}^n$, whose semantics depends on who the observer is: when participant k is observing, b_1 and b_2 are interpreted to be the values of the random bits b_k and b_{k-1} , respectively, while $bv[i]$ is interpreted to be the value of x_{k+i} for $i \in \{0, \dots, n-1\}$, where the indices are computed modulo n as before. The program DCP computes the announcements $a_k, \dots, a_{k+n-1}$ in order, and stores the result in $bv[0], \dots, bv[n-1]$. We specify this probabilistic program as $(\mathcal{S}, I, \{F_s\}_{s \in I})$, where each configuration $(s, \bar{y}, \bar{x})$ consists of a program location s , the valuation $\bar{y}$ of bv , and the secret values $\bar{x}$. We define $I = \{(s_I, \bar{x}, \bar{x}) : \bar{x} \in \{0, 1\}^n, n \geq 3\}$ and $F_{(s_I, \bar{x}, \bar{x})} = \{(s_F, \bar{y}, \bar{x}) : \bar{y} \in \{0, 1\}^n, f(\bar{y}) = f(\bar{x}), y_k = x_k \oplus b_1 \oplus b_2\}$, where k is the observer, s_I is the initial program location, and s_F is a final program location. We verify the uniformity of this program, which suffices to establish the anonymity of the dining cryptographers protocol.

e) Random walk and random sum: We examine the probability equivalence properties of two probabilistic programs `RandWalk` and `RandSum`, see Figure 3. In `RandWalk`, the walker starts from a position $n \geq 1$ and keeps moving leftward or rightward with equal probability. We check that the walker reaches positions 0 and $2n$ equally likely. The probabilistic program $(\mathcal{S}, I, \{F_s\}_{s \in I})$ has $I := \{(s_I, n, n) : n \geq 1\}$ and $F_{(s_I, n, n)} := \{(s_F, 0, n), (s_F, 2n, n)\}$, with s_I denoting the ini-

tial program location and s_F denoting a final program location. In RandSum, the program computes the sum of $2n$ random bits, and we check that $\Pr(\text{sum} = k) = \Pr(\text{sum} = 2n - k)$ holds for $k \in \{0, \dots, n\}$ on termination. The probabilistic program $(\mathfrak{S}, I, \{F_s\}_{s \in I})$ has $I := \{(s_I, 0, n) : n \geq 1\}$ and $F_{(s_I, 0, n)} := \bigcup_{0 \leq k \leq n} E_{k,n} := \bigcup_{0 \leq k \leq n} \{(s_F, k, n), (s_F, 2n - k, n)\}$ for $n \geq 1$. We verify that the reachability probability on each $E_{k,n}$ is uniform. In both programs, uniformity implies the desired probability equivalence property by construction.

f) *Random number generation*: It is well-known that when n is a power of 2, one can sample a random number x from $\{0, \dots, n-1\}$ using $\lg n$ random bits. For general $n \geq 1$, we may compute x by repeatedly sampling it from $\{0, \dots, 2^{\lceil \lg n \rceil} - 1\}$ until $x \leq n-1$, at which point x 's value is uniformly distributed over $\{0, \dots, n-1\}$. This procedure terminates with probability 1 and uses $2\lceil \lg n \rceil$ random bits on average. Knuth and Yao [51] improved the procedure, obtaining a version using $\lceil \lg n \rceil + \Theta(1)$ random bits on average. We model these two random number generators as probabilistic programs NaiveRNG and KnuthYaoRNG, respectively. Both programs have initial states $I := \{q_n : n \geq 1\} := \{(s_I, 0, 1, n) : n \geq 1\}$ and final states $F_{q_n} := \{(s_F, x, y, n) : 0 \leq x < n, y \geq n\}$ for each $n \geq 1$. We verify the uniformity of these programs, proving that the algorithms indeed compute a uniform random variable over $\{0, \dots, n-1\}$ on termination.

g) *Ballot theorem*: We consider a lemma for proving the ballot theorem [72], which concerns the vote counting process for two candidates. Let n_A and n_B be parameters denoting the votes received by candidates A and B , respectively. Suppose $n_A > n_B$ and the votes are counted in a uniformly random sequence. Then, the theorem asserts that the probability of A maintaining a lead throughout the entire counting process is $(n_A - n_B)/(n_A + n_B)$. One technique, often known as the reflection principle, to prove this theorem involves showing that the probability of counting the first vote for A and subsequently reaching a tie is equal to the probability of counting the first vote for B and then reaching a tie. We design a probabilistic program $(\mathfrak{S}, I, \{F_{s_n}\}_{s_n \in I})$ to simulate the vote counting process. A program configuration (s, a, b, f, t, n) records the program location s and maintains the current votes $a \geq 0$ for A , the current votes $b \geq 0$ for B , $t \in \{\perp, \top\}$ indicating whether a tie has occurred, $f \in \{\perp, \top\}$ indicating whether the first vote is for candidate A , and the total number of votes n . We specify $I := \{q_n : n \geq 1\} = \{(s_I, 0, 0, \perp, \perp, n) : n \geq 1\}$ and $F_{q_n} := \{(s_F, a, b, \top, t, n) : a + b = n\}$. Our program simulates vote counting by drawing uniform samples for the votes [72]. Proving the reflection principle then amounts to showing that, starting from each initial configuration $q_n \in I$, the program reaches F_{q_n} with uniform probability.

VII. EVALUATION

To evaluate our approach, we developed a prototype tool in Scala. This tool can process parameterized systems specified in several languages, including FO_{reg}, WS1S, and Armoise [73]. It utilizes toolkits from MONA [57] and TAPAS [74] to generate automata representations of WTSs and verification conditions for probabilistic bisimulations. Our tool

TABLE I
EXPERIMENTAL RESULTS

Anonymity Examples	#S	#T	Mona	Bisim	Lstar	Total
DCP, single-bit	13	832	1.5s	2.9s	0.1s	5s
DCP, multi-bit	16	1024	1.6s	18s	0.2s	20s
Crowds Protocol	20	1280	2.0s	0.5s	0.8s	4s
Grades Protocol	25	1600	3.1s	23s	0.1s	27s

Uniformity Examples	#S	#T	Mona	Bisim	Lstar	Total
Dining Cryptographers	4	256	0.4s	0.3s	0.5s	2s
Random Walk	6	96	0.3s	0.6s	0.3s	2s
Random Sum	7	448	0.8s	1.1s	1.2s	3s
Knuth-Yao RNG	13	832	1.6s	0.5s	0.8s	3s
Naive RNG	21	1344	2.3s	1.4s	0.6s	5s
Ballot Theorem	52	3328	14s	59s	4.2s	78s

Random Walk Settings	#S	#T	Mona	Bisim	Lstar	Total
$d = 1, k = 10$	6	96	0.5s	1.8s	0.1s	3s
$d = 1, k = 50$	6	96	0.5s	53s	0.3s	54s
$d = 1, k = 100$	6	96	0.5s	529s	0.4s	531s
$d = 1, k = 150$	6	96	0.5s	4863s	0.6s	4865s
$d = 2, k = 1$	32	2048	8.3s	28s	4.5s	42s
$d = 3, k = 1$	200	51200	389s	2685s	1167s	4256s

is built around the active learning procedure described in Section IV. Specifically, membership queries are resolved by computing bisimulations for finite-state system instances, while equivalence queries leverage MONA to verify hypothesis automata and generate counterexamples when necessary. Candidate bisimulations are restricted to binary relations over configuration pairs with valid encoding to streamline proof inference. Below, we discuss the effectiveness, performance, and limitations of our approach based on experiments conducted on a Windows laptop with a 2.3GHz Intel i7-11800H processor and 16GB memory limit.

a) *Effectiveness*: The first two tables in Table I summarize the examples in Section VI, presenting the sizes of synthesized proofs (states #S and transitions #T) and the run-times of MONA verification (Mona), bisimulation computation (Bisim), learner processing (Lstar), and tool execution (Total). These results demonstrate that our tool effectively identifies regular proofs for all examined examples. They also highlight proof complexity and bisimulation computation as the primary performance bottlenecks. Indeed, as proof complexity grows, the search space expands, leading to longer verification times. Also, some examples compute bisimulations over large system instances before convergence, even though the final proof sizes remain relatively small (e.g., the DCP multi-bit example).

b) *Performance*: To further investigate the performance factors of the learning algorithm, we consider a random walk example within $\{\bar{x} \in \mathbb{Z}^d : \forall i. |x_i| \leq n \wedge k \leq |x_i|\}$, where n is a parameter, and k and d are constants. This example can be formulated as a parameterized system $\{P_n\}_{n \geq k}$. We verify that, starting from the origin, the walker reaches the 2^d corner points $p \in \{-n, n\}^d$ with equal probability. The third table in Table I outlines our tool's performance under different k and d . By specifying a larger k in the parameterized system, we require the learner to infer proofs based on bisimulations over larger system instances, amplifying the computational burden even though the inferred proofs remain unchanged. Also, as the dimension d increases, proof complexity escalates and impacts overall performance. These results suggest that the

learning efficiency largely depends on the complexity of the bisimulations and candidate solutions navigated by the learner before it arrives at the final answer.

c) *Limitations*: Although our learning-based approach successfully verifies all the examined examples, it is important to note that the general problem remains undecidable. Indeed, compared to explicit enumeration (cf. Theorem 6), the learning algorithm prioritizes fast convergence over completeness, which means it may fail to find a regular proof even when one exists. In such cases, one could consider combining our learning approach with an enumerative method like solver-based synthesis [75]. It remains an open question to characterize a natural class of parameterized systems for which our learning algorithm is complete.

VIII. RELATED WORK

Our verification framework can be construed as a parameterized variant of probabilistic model checking for anonymity and uniformity. We discuss below the most pertinent literature.

a) *Anonymity*: Formal verification of anonymous and secure communication protocols primarily relies on two methodologies: theorem proving and automated verification. While theorem proving methods [76]–[78] are capable of handling complex properties and systems, these methods often demand nontrivial manual effort and domain expertise when the system induces an unbounded model, which poses an obstacle to automation. Automated verification methods offer various approaches to reason about anonymity. Model checkers based on epistemic logic [79]–[81] and process algebras [82], [83] can analyze qualitative anonymity properties by abstracting randomness into nondeterminism and applying techniques like symbolic model checking [84]. Probabilistic model checkers, including MCSTA [85], PRISM [86], [87], and STORM [88], can express quantitative anonymity properties in probabilistic temporal logic and analyze them through exact computation or numerical approximation [89]. Equivalence checkers [49], [71], [90], [91] formulate anonymity as indistinguishability of system executions, thereby reducing the verification tasks to solving language or trace equivalence problems in probabilistic systems. Despite extensive tool support, most existing equivalence and model checkers are only capable of verifying our case studies in the *finite* setting. To the best of our knowledge, this work provides the first fully automated approach that can verify them in the parameterized setting.

b) *Uniformity*: Uniformity verification is a specialized form of relational verification for probabilistic programs [44]. For finite-state programs, various tools [87], [88] can be employed to model check uniformity properties. For infinite-state programs, Barthe et al. [43] have extended the probabilistic program logic pRHL [77], [78], initially designed for relational properties, to reason about uniformity using *coupling*. Albarghouthi and Hsu [72] further exploited program synthesis techniques to construct coupling proofs. Their work is, to the best of our knowledge, the only fully automated approach for infinite-state uniformity verification aside from our method. Notably, our tool successfully verifies all uniformity examples considered in [72]. A coupling argument aims to show a one-to-one correspondence between relevant execution paths, while

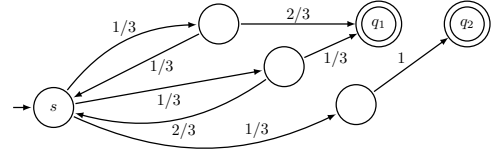


Fig. 4. A Markov chain with uniform output distribution over $F_s = \{q_1, q_2\}$

bisimulation establishes equivalence between path probabilities. One-to-one correspondence is a stronger condition for probability equivalence by proving uniformity modulo permutation of paths. In comparison, bisimulation proves uniformity modulo summation of path probabilities. Figure 4 presents a toy example whose uniformity is trivial for bisimulation proofs but not directly amenable to coupling proofs [72]. On the other hand, coupling arguments can establish probability independence through self-composition [43]. It remains unclear whether probability independence is provable by bisimulation when the output distribution is non-uniform. Symbolic inference [92]–[94] provides automated methods to answer symbolic queries about distributions induced by probabilistic programs. Though it is possible to encode uniformity queries in such methods, existing formalisms of symbolic inference [44] fall short in specifying parameterized systems like those considered by this work.

c) *Bisimulation*: Our method leverages bisimulation to reason about anonymity and uniformity. The concept of bisimulation we consider is also referred to as *strong bisimulation*, and the corresponding behavioral equivalence is termed *equivalence modulo strong bisimilarity* [2], [95]. In the literature, other types of behavioral equivalence have been studied for probabilistic systems, including weak bisimilarity [96], [97] branching bisimilarity [98], and ϵ -bisimilarity [99], with most investigations focusing on finite-state systems. For infinite-state systems, various decidable classes have been identified for strong bisimilarity, including several types of process rewrite systems and pushdown systems in both probabilistic and non-probabilistic settings [29], [31], [100]. Despite these theoretical advances, few of the results have been adapted into practical verification tools [33]. Interestingly, Forejt et al. [31] showed that strong bisimilarity on probabilistic systems can be reduced to strong bisimilarity on nondeterministic LTSs. Thus, specialized proof rules are not essential for verifying probabilistic bisimulations. Unfortunately, their reduction does not preserve regular system encoding, rendering it incompatible with our regular verification framework. In a recent study, Abate et al. [28] proposed a data-driven approach to synthesize bisimulations for infinite-state LTSs. Their approach utilizes SMT solvers to generate candidate bisimulations in a learner-verifier architecture conceptually similar to our learner-teacher framework. Nevertheless, their method is limited to learning bisimulation relations with finitely many equivalence classes, which is often insufficient for parameterized systems.

IX. CONCLUSION

This paper introduces a first-order framework for checking strong bisimulation equivalence in infinite-state probabilistic

systems, with applications to anonymity and uniformity verification. Our approach requires that (i) the examined system has an effective regular presentation, (ii) the system is bounded branching, and (iii) the system is weakly finite, which holds naturally for parameterized systems. We show that, while the general verification problem is undecidable, our framework can effectively encode and automatically verify challenging examples from the literature.

Future research could explore generalizations to weaker versions of bisimulation equivalence like ε -bisimilarity [99] and bisimulation metrics [101], which tolerate slight deviations when comparing system behaviors. Such relaxation is particularly relevant for verifying cryptographic protocols, since most practical protocols do not achieve perfect secrecy but are still sufficiently secure. Another interesting direction is to enhance the expressiveness of our framework by utilizing the recent development of *regular abstraction* [102], [103], which allows for specifying and reasoning about regular structures in background SMT theories. Finally, it is possible to improve our framework's capabilities to handle more complex systems by incorporating probabilistic model checkers and program verifiers, e.g., as oracles for bisimulation learning [28].

REFERENCES

- [1] R. Milner, *A calculus of communicating systems*. Springer, 1980.
- [2] —, *Communication and Concurrency*. Prentice hall Englewood Cliffs, 1989, vol. 84.
- [3] G. J. Holzmann and R. Joshi, “Model-driven software verification,” in *International SPIN Workshop*. Springer, 2004, pp. 76–91.
- [4] K. Banerjee, D. Sarkar, and C. Mandal, “Deriving bisimulation relations from path extension based equivalence checkers,” *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 946–953, 2016.
- [5] J. Gancher, K. Sojakova, X. Fan, E. Shi, and G. Morrisett, “A core calculus for equational proofs of cryptographic protocols,” *Symposium on Principles of Programming Languages (POPL)*, vol. 7, no. POPL, pp. 866–892, 2023.
- [6] A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague, “Probabilistic bisimulation and equivalence for security analysis of network protocols,” in *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*. Springer, 2004, pp. 468–483.
- [7] V. Castiglioni, K. Chatzikokolakis, and C. Palamidessi, “A logical characterization of differential privacy,” *Science of Computer Programming*, vol. 188, p. 102388, 2020.
- [8] A. A. Noroozi, J. Karimpour, and A. Isazadeh, “Bisimulation for secure information flow analysis of multi-threaded programs,” *Mathematical and Computational Applications*, vol. 24, no. 2, p. 64, 2019.
- [9] N. Dong, R. Guanciale, and M. Dam, “Refinement-based verification of device-to-device information flow,” in *Formal Methods in Computer-Aided Design (FMCAD)*, 2021, pp. 123–132.
- [10] K. Salehi, A. A. Noroozi, S. Amir-Mohammadian, and M. Mohagheghi, “An automated quantitative information flow analysis for concurrent programs,” in *International Conference on Quantitative Evaluation of Systems (QEST)*. Springer, 2022, pp. 43–63.
- [11] T. Ágotnes and R. Galimullin, “Quantifying over information change with common knowledge,” *Autonomous Agents and Multi-Agent Systems (AAMAS)*, vol. 37, no. 1, p. 19, 2023.
- [12] T.-L. Tran, Q.-T. Ha, T.-L.-G. Hoang, L. A. Nguyen, and H. S. Nguyen, “Bisimulation-based concept learning in description logics,” *Fundamenta Informaticae*, vol. 133, no. 2-3, pp. 287–303, 2014.
- [13] R. Fervari, F. R. Velázquez-Quesada, and Y. Wang, “Bisimulations for knowing how logics,” *The Review of Symbolic Logic*, vol. 15, no. 2, pp. 450–486, 2022.
- [14] R. G. Scott, M. Dodds, I. Perez, A. E. Goodloe, and R. Dockins, “Trustworthy runtime verification via bisimulation (experience report),” *Proceedings of the ACM on Programming Languages (POPL)*, vol. 7, no. ICFP, pp. 305–321, 2023.
- [15] F. Bonchi and D. Pous, “Checking NFA equivalence with bisimulations up to congruence,” *ACM SIGPLAN Notices*, vol. 48, no. 1, pp. 457–468, 2013.
- [16] D. Kuperberg, L. Pinault, and D. Pous, “Coinductive algorithms for büchi automata,” *Fundamenta Informaticae*, vol. 180, no. 4, pp. 351–373, 2021.
- [17] J. Jacobs and T. Wißmann, “Fast coalgebraic bisimilarity minimization,” *Symposium on Principles of Programming Languages (POPL)*, vol. 7, no. POPL, pp. 1514–1541, 2023.
- [18] C. Dehnert, J.-P. Katoen, and D. Parker, “SMT-based bisimulation minimisation of Markov models,” in *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Springer, 2013, pp. 28–47.
- [19] L. Lin, J. Cao, J. Lam, L. Rutkowski, G. M. Dimirovski, and S. Zhu, “A bisimulation-based foundation for scale reductions of continuous-time markov chains,” *IEEE Transactions on Automatic Control*, 2024.
- [20] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, “Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2024.
- [21] J. Sproston and S. Donatelli, “Backward bisimulation in markov chain model checking,” *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 531–546, 2006.
- [22] C. Baier, “Polynomial time algorithms for testing probabilistic bisimulation and simulation,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 1996, pp. 50–61.
- [23] A. Valmari and G. Franceschinis, “Simple $O(m \log n)$ time markov chain lumping,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2010, pp. 38–52.
- [24] D. Chen, F. van Breugel, and J. Worrell, “On the complexity of computing probabilistic bisimilarity,” in *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*. Springer, 2012, pp. 437–451.
- [25] F. Moller, “Infinite results,” in *International Conference on Concurrency Theory (CONCUR)*. Springer, 1996, pp. 195–216.
- [26] R. Kumar, C. Zhou, and S. Basu, “Finite bisimulation of reactive untimed infinite state systems modeled as automata with variables,” in *American Control Conference*. IEEE, 2006, pp. 6–pp.
- [27] C.-D. Hong, A. W. Lin, R. Majumdar, and P. Rümmer, “Probabilistic bisimulation for parameterised systems,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2019, pp. 455–474.
- [28] A. Abate, M. Giacobbe, and Y. Schnitzer, “Bisimulation learning,” in *International Conference on Computer Aided Verification (CAV)*. Springer, 2024, pp. 161–183.
- [29] J. Srba, “Roadmap of infinite results,” in *Current Trends in Theoretical Computer Science: The Challenge of the New Century*. World Scientific, 2004, pp. 337–350.
- [30] G. Sénizergues, “The bisimulation problem for equational graphs of finite out-degree,” *SIAM Journal on Computing*, vol. 34, no. 5, pp. 1025–1106, 2005.
- [31] V. Forejt, P. Jančar, S. Kiefer, and J. Worrell, “Game characterization of probabilistic bisimilarity, and applications to pushdown automata,” *Logical Methods in Computer Science (LMCS)*, vol. 14, no. 4, 2018.
- [32] V. Koutavas, Y.-Y. Lin, and N. Tzevelekos, “Pushdown normal-form bisimulation: A nominal context-free approach to program equivalence,” in *Symposium on Logic in Computer Science (LICS)*, 2024, pp. 1–15.
- [33] H. Garavel and F. Lang, “Equivalence checking 40 years after: a review of bisimulation tools,” *A Journey from Process Algebra via Timed Automata to Model Learning: Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, pp. 213–265, 2022.
- [34] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, *Deductive Software Verification - The KeY Book - From Theory to Practice*. Springer, 2016, vol. 10001.
- [35] R. Hähnle and M. Huisman, “Deductive software verification: from pen-and-paper proofs to industrial tools,” *Computing and Software Science: State of the Art and Perspectives*, pp. 345–373, 2019.
- [36] C. A. Furia, B. Meyer, and S. Velder, “Loop invariants: Analysis, classification, and examples,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–51, 2014.
- [37] A. Blumensath and E. Grädel, “Finite presentations of infinite structures: Automata and interpretations,” *Theory of Computing Systems (TCS)*, vol. 37, no. 6, pp. 641–674, 2004.
- [38] T. Colcombet and C. Löding, “Transforming structures by set interpretations,” *Logical Methods in Computer Science (LMCS)*, vol. 3, no. 2, pp. paper–4, 2007.
- [39] A. W. Lin and P. Rümmer, “Regular model checking revisited,” in *Model Checking, Synthesis, and Learning: Essays Dedicated to Bengt Jonsson on The Occasion of His 60th Birthday*. Springer, 2022, pp. 97–114.
- [40] R. Beauxis and C. Palamidessi, “Probabilistic and nondeterministic aspects of anonymity,” *Theoretical Computer Science*, vol. 410, no. 41, pp. 4006–4025, 2009.
- [41] M. Bhargava and C. Palamidessi, “Probabilistic anonymity,” in *International Conference on Concurrency Theory (CONCUR)*. Springer, 2005, pp. 171–185.
- [42] J. Y. Halpern and K. R. O’Neill, “Anonymity and information hiding in multiagent systems,” *Journal of Computer Security*, vol. 13, no. 3, pp. 483–514, 2005.

- [43] G. Barthe, T. Espitau, B. Grégoire, J. Hsu, and P.-Y. Strub, “Proving uniformity and independence by self-composition and coupling,” *arXiv preprint arXiv:1701.06477*, 2017.
- [44] G. Barthe, J.-P. Katoen, and A. Silva, *Foundations of Probabilistic Programming*. Cambridge University Press, 2020.
- [45] D. Chaum, “The dining cryptographers problem: Unconditional sender and recipient untraceability,” *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.
- [46] P. A. Abdulla, A. P. Sistla, and M. Talupur, “Model checking parameterized systems,” *Handbook of model checking*, pp. 685–725, 2018.
- [47] G. Kourtis, C. Dixon, and M. Fisher, “Parameterized verification of leader/follower systems via arithmetic constraints,” *IEEE Transactions on Software Engineering*, 2024.
- [48] A. W. Lin and P. Rümmer, “Liveness of randomised parameterised systems under arbitrary schedulers,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2016, pp. 112–133.
- [49] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, “APEX: an analyzer for open probabilistic programs,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2012, pp. 693–698.
- [50] M. K. Reiter and A. D. Rubin, “Crowds: Anonymity for web transactions,” *ACM transactions on information and system security*, vol. 1, no. 1, pp. 66–92, 1998.
- [51] D. Knuth, “The complexity of nonuniform random number generation,” *Algorithms and Complexity, New Directions and Results*, pp. 357–428, 1976.
- [52] W. Feller, *An introduction to probability theory and its applications*. John Wiley & Sons, 1991, vol. 81.
- [53] Y.-F. Chen, C.-D. Hong, A. W. Lin, and P. Rümmer, “Learning to prove safety over parameterised concurrent systems,” in *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. Springer, 2017, pp. 76–83.
- [54] O. Markgraf, C.-D. Hong, A. W. Lin, M. Najib, and D. Neider, “Parameterized synthesis with safety properties,” in *Asian Symposium on Programming Languages and Systems*. Springer, 2020, pp. 273–292.
- [55] P. Garg, C. Löding, P. Madhusudan, and D. Neider, “ICE: A robust framework for learning invariants,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2014, pp. 69–87.
- [56] K. G. Larsen and A. Skou, “Bisimulation through probabilistic testing,” *Information and Computation*, vol. 94, no. 1, pp. 1–28, 1991.
- [57] N. Klarlund and A. Möller, *Mona Version 1.4: User Manual*. BRICS, Department of Computer Science, University of Aarhus Denmark, 2001.
- [58] T. Fiedor, L. Holík, P. Janků, O. Lengál, and T. Vojnar, “Lazy automata techniques for WSIS,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2017, pp. 407–425.
- [59] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.
- [60] R. L. Rivest and R. E. Schapire, “Inference of finite automata using homing sequences,” *Information and Computation*, vol. 103, no. 2, pp. 299–347, 1993.
- [61] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*. MIT press, 1994.
- [62] K. Etessami and M. Yannakakis, “Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations,” *Journal of the ACM (JACM)*, vol. 56, no. 1, pp. 1–66, 2009.
- [63] A. Bianco and L. De Alfaro, “Model checking of probabilistic and nondeterministic systems,” in *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Springer, 1995, pp. 499–513.
- [64] F. Clerc, N. Fijalkow, B. Klin, and P. Panangaden, “Expressiveness of probabilistic modal logics: A gradual approach,” *Information and Computation*, vol. 267, pp. 145–163, 2019.
- [65] C.-D. Hong, “Symbolic techniques for parameterised verification,” Ph.D. dissertation, University of Oxford, 2022.
- [66] V. Bárány, “Automatic presentations of infinite structures.” Ph.D. dissertation, RWTH Aachen University, Germany, 2007.
- [67] J. Esparza, A. Gaiser, and S. Kiefer, “Proving termination of probabilistic programs using patterns,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2012, pp. 123–138.
- [68] A. Nerode, “Linear automaton transformations,” *Proceedings of the American Mathematical Society*, vol. 9, no. 4, pp. 541–544, 1958.
- [69] W. Woess, *Denumerable Markov chains*. European Mathematical Society, 2009.
- [70] A. Abate, M. Giacobbe, and D. Roy, “Learning probabilistic termination proofs,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2021, pp. 3–26.
- [71] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, “Algorithmic probabilistic game semantics,” *Formal Methods in System Design (FMSD)*, vol. 43, no. 2, pp. 285–312, 2013.
- [72] A. Albarghouthi and J. Hsu, “Constraint-based synthesis of coupling proofs,” in *International Conference on Computer Aided Verification (CAV)*. Springer, 2018, pp. 327–346.
- [73] J. Leroux and G. Point, “The Armoire language,” https://tapas.labri.fr/wp/?page_id=17, 2010.
- [74] —, “Tapas: The talence presburger arithmetic suite,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2009, pp. 182–185.
- [75] M. J. Heule and S. Verwer, “Software model synthesis using satisfiability solvers,” *Empirical Software Engineering*, vol. 18, no. 4, pp. 825–856, 2013.
- [76] A. McIver and C. Morgan, “The thousand-and-one cryptographers,” *Engineering Secure and Dependable Software Systems*, vol. 53, p. 137, 2019.
- [77] G. Barthe, B. Grégoire, and S. Zanella Béguelin, “Formal certification of code-based cryptographic proofs,” in *Symposium on Principles of programming languages (POPL)*. ACM, 2009, pp. 90–101.
- [78] M. Avanzini, G. Barthe, D. Davoli, and B. Grégoire, “A quantitative probabilistic relational hoare logic,” *arXiv preprint arXiv:2407.17127*, 2024.
- [79] O. Al Bataineh and R. van der Meyden, “Abstraction for epistemic model checking of dining cryptographers-based protocols,” in *Conference on Theoretical Aspects of Rationality and Knowledge*, 2011, pp. 247–256.
- [80] A. Lomuscio, H. Qu, and F. Raimondi, “MCMAS: an open-source model checker for the verification of multi-agent systems,” *International Journal on Software Tools for Technology Transfer*, vol. 19, pp. 9–30, 2017.
- [81] M. Knapik, A. Niewiadomski, W. Penczek, A. Pórtola, M. Szreter, and A. Zbrzezny, “Parametric model checking with ver ics,” *Transactions on Petri Nets and Other Models of Concurrency*, pp. 98–120, 2010.
- [82] V. Cheval, “Apte: an algorithm for proving trace equivalence,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2014, pp. 587–592.
- [83] A. Tiu and J. Dawson, “Automating open bisimulation checking for the spi calculus,” in *Computer Security Foundations Symposium*. IEEE, 2010, pp. 307–321.
- [84] V. Cortier, S. Kremer, and B. Warinschi, “A survey of symbolic methods in computational analysis of cryptographic systems,” *Journal of Automated Reasoning*, vol. 46, pp. 225–259, 2011.
- [85] A. Hartmanns and H. Hermanns, “The modest toolset: An integrated environment for quantitative modeling and verification,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 593–598.
- [86] V. Shmatikov, “Probabilistic analysis of an anonymity system,” *Journal of Computer Security*, vol. 12, no. 3–4, pp. 355–377, 2004.
- [87] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2011, pp. 585–591.
- [88] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, “The probabilistic model checker Storm,” *International Journal on Software Tools for Technology Transfer*, pp. 1–22, 2022.
- [89] J. J. Rutten, *Mathematical techniques for analyzing concurrent and probabilistic systems*. American Mathematical Society, 2004, no. 23.
- [90] M. S. Bauer, R. Chadha, A. Prasad Sistla, and M. Viswanathan, “Model checking indistinguishability of randomized security protocols,” in *International Conference on Computer Aided Verification (CAV)*. Springer, 2018, pp. 117–135.
- [91] V. Cheval, R. Crubillé, and S. Kremer, “Symbolic protocol verification with dice: process equivalences in the presence of probabilities,” in *Computer Security Foundations Symposium (CSF)*. IEEE, 2022, pp. 319–334.
- [92] T. Gehr, S. Misailovic, and M. Vechev, “Psi: Exact symbolic inference for probabilistic programs,” in *International Conference on Computer Aided Verification (CAV)*. Springer, 2016, pp. 62–83.
- [93] M. Cusumano-Towner, B. Bichsel, T. Gehr, M. Vechev, and V. K. Mansinghka, “Incremental inference for probabilistic programs,” in *Conference on Programming Language Design and Implementation (PLDI)*, 2018, pp. 571–585.
- [94] Z. Susag, S. Lahiri, J. Hsu, and S. Roy, “Symbolic execution for randomized programs,” *Proceedings of the ACM on Programming Languages*, vol. 6, no. OOPSLA2, pp. 1583–1612, 2022.
- [95] M. Hennessy, “Exploring probabilistic bisimulations, part i,” *Formal Aspects of Computing*, vol. 24, no. 4, pp. 749–768, 2012.
- [96] C. Baier and H. Hermanns, “Weak bisimulation for fully probabilistic processes,” in *International Conference on Computer Aided Verification (CAV)*. Springer, 1997, pp. 119–130.
- [97] A. Philippou, I. Lee, and O. Sokolsky, “Weak bisimulation for probabilistic systems,” in *International Conference on Concurrency Theory (CONCUR)*. Springer, 2000, pp. 334–349.
- [98] S. Andova and T. A. Willemse, “Branching bisimulation for probabilistic systems: Characteristics and decidability,” *Theoretical Computer Science (TCS)*, vol. 356, no. 3, pp. 325–355, 2006.
- [99] T. Spork, C. Baier, J.-P. Katoen, J. Piribauer, and T. Quatmann, “A spectrum of approximate probabilistic bisimulations,” *arXiv preprint arXiv:2407.07584*, 2024.

- [100] L. Aceto, A. Ingólfssdóttir, J. Srba *et al.*, “The algorithmics of bisimilarity,” *Advanced Topics in Bisimulation and Coinduction*, vol. 52, pp. 100–172, 2012.
- [101] F. Van Breugel and J. Worrell, “A behavioural pseudometric for probabilistic transition systems,” *Theoretical Computer Science*, vol. 331, no. 1, pp. 115–142, 2005.
- [102] C.-D. Hong and A. W. Lin, “Regular abstractions for array systems,” *Symposium on Principles of Programming Languages (POPL)*, vol. 8, no. POPL, pp. 638–666, 2024.
- [103] P. Czerner, J. Esparza, V. Krasotin, and C. Welzel-Mohr, “Computing inductive invariants of regular abstraction frameworks,” *arXiv preprint arXiv:2404.10752*, 2024.