Learning to Prove Safety over Parameterised Concurrent Systems

Yu-Fang Chen
Academia Sinica

Chih-Duo Hong
Oxford University

Anthony W. Lin
Oxford University

Philipp Rümmer
Uppsala University

***Abstract*—We revisit the classic problem of proving safety over parameterised concurrent systems, i.e., an infinite family of finite-state concurrent systems that are represented by some finite (symbolic) means. An example of such an infinite family is a dining philosopher protocol with any number n of processes (n being the parameter that defines the infinite family). Regular model checking is a well-known generic framework for modelling parameterised concurrent systems, where an infinite set of configurations (resp. transitions) is represented by a regular set (resp. regular transducer). Although verifying safety properties in the regular model checking framework is undecidable in general, many sophisticated semi-algorithms have been developed in the past fifteen years that can successfully prove safety in many practical instances. In this paper, we propose a simple solution to synthesise regular inductive invariants that makes use of Angluin's classic L^* algorithm (and its variants). We provide a termination guarantee when the set of configurations reachable from a given set of initial configurations is regular. We have tested L^* algorithm on standard (as well as new) examples in regular model checking including the dining philosopher protocol, the dining cryptographer protocol, and several mutual exclusion protocols (e.g. Bakery, Burns, Szymanski, and German). Our experiments show that, despite the simplicity of our solution, it can perform at least as well as existing semi-algorithms.**

I. Introduction

Parameterised concurrent systems are infinite families of finite-state concurrent systems, parameterised by the number n of processes. There are numerous examples of parameterised concurrent systems, including models of distributed algorithms which are typically designed to handle an arbitrary number n of processes [32], [53]. Verification of such systems, then, amounts to proving that a desired property holds for *all* permitted values of n. For example, proving that the safety property holds for a dining philosopher protocol entails proving that the protocol with any given number n of philosophers ($n \geq 3$) can never reach a state when two neighbouring philosophers eat simultaneously. For each given value of n, verifying safety/liveness is decidable, albeit the exponential state-space explosion in the parameter n. However, when the property has to hold for each value of n, the number of system configurations a verification algorithm has to explore is potentially infinite. Indeed, even safety checking is already undecidable for parameterised concurrent systems [9], [12], [30]; see [13] for a comprehensive survey on the decidability aspect of the parameterised verification problem.

Various sophisticated semi-algorithms for verifying parameterised concurrent systems are available. These semi-algorithms typically rely on a symbolic framework for representing infinite sets of system configurations and transitions. *Regular model checking* [42], [7], [14], [15], [6], [1], [22], [45], [65] is one well-known symbolic framework for modelling and verifying parameterised concurrent systems. In regular model checking, configurations are modelled using words over a finite alphabet, sets of configurations are represented as regular languages, and the transition relation is defined by a regular transducer. From the research programme of regular model checking, not only are regular languages/transducers known to be highly expressive symbolic representations for modelling parameterised concurrent systems, they are also amenable to an automata-theoretic approach (due to many nice closure properties of regular languages/transducers), which have often proven effective in verification.

In this paper, we revisit the classic problem of verifying safety in the regular model checking framework. Many sophisticated semi-algorithms for dealing with this problem have been developed in the literature using methods such as abstraction [4], [5], [21], [20], widening [15], [23], acceleration [57], [42], [11], and learning [54], [55], [38], [63], [62]. One standard technique for proving safety for an infinite-state systems is by exhibiting an *inductive invariant* Inv (i.e. a set of configurations that is closed under an application of the transition relation) such that **(i)** Inv subsumes the set $Init$ of all initial configurations, but **(ii)** Inv does not intersect with the set Bad of unsafe configurations. In regular model checking, the sets $Init$ and Bad are given as regular sets. For this reason, a natural method for proving safety in regular model checking is to exhibit a *regular* inductive invariant satisfying (i) and (ii). The regular set Inv can be constructed as a "regular proof" for safety since checking that a candidate regular set Inv is a proof for safety is decidable. A few semi-algorithms inspired by automata learning — some based on the passive learning algorithms [38], [55], [2] and some others based on active learning algorithms [55], [62]— have been proposed to synthesise a regular inductive invariant in regular model checking. Despite these semi-algorithms, not much attention has been paid to applications of automata learning in regular model checking.

In this paper, we are interested in one basic research question in regular model checking: *can we effectively apply the classic Angluin's L^* automata learning [8] (or variants [58], [44]) to learn a regular inductive invariant?* Hitherto this question, perhaps surprisingly, has no satisfactory answer in the literature. A more careful consideration reveals at least two problems. Firstly, membership queries (i.e. is a word w reachable from Init?) may be asked by the L^* algorithm, which amounts to checking reachability in an infinite-state system, which is undecidable in general. This problem was already noted in [54], [55], [62], [63]. Secondly, a regular inductive invariant satisfying (i) and (ii) might not be unique, and so strictly speaking we are not dealing with a well-defined learning problem. More precisely, consider the question of *what the teacher should answer when the learner asks whether v is in the desired invariant, but v turns out not to be reachable from Init?* Discarding v might not be a good idea, since this could force the learning algorithm to look for a *minimal* (in the sense of set inclusion) inductive invariant, which might not be regular. Similarly, let us consider what the teacher should answer in the case when we found a pair (v, w) of configurations such that (1) v is in the candidate Inv, (2) $w \notin \mathit{Inv}$, and (3) there is a transition from v to w. In the ICE-learning framework [35], [34], [54], the pair (v, w) is called an *implication counterexample*. To satisfy the inductive invariant constraint, the teacher may respond that w should be added to Inv, or that v should be removed from Inv. Some works in the literature have proposed using a three-valued logic/automaton (with "don't know" as an answer) because of the teacher's incomplete information [37], [26].

a) Contribution: In this paper, we propose a simple and practical solution to the problem of applying the classic L^* automata learning algorithm and its variants to synthesise a regular inductive invariant in regular model checking. To deal with the first problem mentioned in the previous paragraph, we propose to restrict to *length-preserving* regular transducers. In theory, length-preservation is not a restriction for safety analysis, since it just implies that each instance of the considered parameterised system is operating on bounded memory of size n (but the parameter n is unbounded). Experience shows that many practical examples in parameterised concurrent systems can be captured naturally in terms of length-preserving systems, e.g., see [52], [7], [6], [42], [22], [57], [1]. The benefit of the restriction is that the problem of membership queries is now decidable, since the set of configurations that may reach (be reachable from) any given configuration w is finite and can be solved by a standard finite-state model checker. For the second problem mentioned in the previous paragraph, we propose that a *strict teacher* be employed in L^* learning for regular inductive invariants in regular model checking. A strict teacher attempts to teach the learner the minimal inductive invariant (be it regular or not), but is satisfied when the candidate answer posed by the learner is an inductive invariant satisfying (i) and (ii) without being minimal. [In this sense, perhaps a more appropriate term is a *strict but generous teacher*, who tries to let a student pass a final exam whenever possible.] For this reason, when the learner asks whether w is in the desired inductive invariant, the teacher will reply NO if w is not reachable from Init. The same goes with an implication counterexample (v, w) such that the teacher will say that an unreachable v is not in the desired inductive invariant.

We have implemented the learning-based approach in a prototype tool with an interface to the libalf library, which includes the L^* algorithm and its variants. Despite the simplicity of our solution, it (perhaps surprisingly) works extremely well in practice, as our experiments suggest. We have taken numerous standard examples from regular model checking, including cache coherence protocols (German's Protocol), self-stabilising protocols (Israeli-Jalfon's Protocol and Herman's Protocol), synchronisation protocols (Lehmann-Rabin's Dining Philosopher Protocol), secure multi-party computation protocols (Dining Cryptographers Protocol [25]), and mutual exclusion protocols (Szymanski's Protocol, Burn's Protocol, Dijkstra's Protocol, Lamport's bakery algorithm, and Resource-Allocator Protocol). We show that L^* algorithm can perform at least as well as (and, in fact, often outperform) existing semi-algorithms. We compared the performance of our algorithm with well-known and established techniques such as SAT-based learning [55], [54], [51], [52], abstract regular model checking (ARMC), which is based on abstraction-refinement using predicate abstractions and finite-length abstractions [20], [21], and T(O)RMC, which is based on extrapolation (a widening technique) [16]. Our experiments show that, despite the simplicity of our solution, it can perform at least as well as existing semi-algorithms.

b) Related Work: The work of Vardhan *et al.* [63], [62] applies L^* learning to infinite-state systems and, amongst other, regular model checking. The learning algorithm attempts to learn an inductive invariant enriched with "distance" information, which is one way to make membership queries (i.e. reachability for general infinite-state systems) decidable. This often makes the resulting set not regular, even if the set of reachable configurations is regular, in which case our algorithm is guaranteed to terminate (recall our algorithm is only learning a regular invariant without distance information). Conversely, when an inductive invariant enriched with distance information is regular, so is the projection that omits the distance information. Unfortunately, neither their tool Lever [63], nor the models used in their experiments are available, so that we cannot make a direct comparison to our approach. A learning algorithm allowing incomplete information [37] has been applied in [55] for inferring inductive invariants of regular model checking. Although the learning algorithm in [37] uses the same data structure as the standard L^* algorithm, it is essentially a SAT-based learning algorithm (its termination is not guaranteed by the Myhill-Nerode theorem).

Despite our results that SAT-based learning seems to be less efficient than L^* learning for synthesising regular inductive invariants in regular model checking, SAT-based learning is more general and more easily applicable when verifying other properties, e.g., liveness [52], fair termination [48], and safety games [56]. View abstraction [5] is a novel technique for parameterised verification. Comparing to parameterised verification based on view abstraction, our framework (i.e. general regular model checking framework with transducers) provides a more expressive modelling language that is required in specifying protocols with near-neighbour communication (e.g. Dining Cryptographers and Dining Philosophers).

When preparing the final version, we found that a very similar algorithm had already appeared in Vardhan's thesis [61, Section 6] from 2006; in particular, including the trick to make a membership query (i.e. point-to-point reachability) decidable by bounding the space of the transducers. The research presented here was conducted independently, and considers several aspects that were not yet present in [61], including experimental results on systems that are not counter systems (parameterised concurrent systems with topologies), and heuristics like the use of shortest counterexamples and caching. We cannot compare our implementation in detail with the one from [61], since the latter tool is not publicly available.

c) Organisation: The notations are defined in Section II. A brief introduction to regular model checking and automata learning is given in Section III and Section IV, respectively. The learning-based algorithm is provided in Section V. The result of the experiments is in Section VI.

II. Preliminaries

a) General Notations: Let Σ be a finite set of symbols called *alphabet*. A word over Σ is a finite sequence of symbols of Σ. We use λ to represent an empty word. For a set $I \subseteq \Sigma^*$ and a relation $T \subseteq \Sigma^* \times \Sigma^*$, we define $T(I)$ to be the post-image of I under T, i.e., $T(I) = \{y \mid \exists x.\ x \in I \wedge (x,y) \in T\}$. Let $id = \{(x,x) \mid x \in \Sigma^*\}$ be the *identity relation*. We define T^n for all $n \in \mathbb{N}$ in the standard way by induction: $T^0 = id$, and $T^k = T \circ T^{k-1}$, where $\circ$ denotes the *composition* of relations. Let T^* denote the transitive closure of T, i.e., $T^* = \bigcup_{i=1}^{\infty} T^i$. For any two sets A and B, we use $A \ominus B$ to denote their *symmetric difference*, i.e., the set $A \setminus B \cup B \setminus A$.

b) Finite Automata and Transducer: In this paper, automata/transducers are denoted in calligraphic fonts $\mathcal{A}, \mathcal{B}, \mathcal{I}, \mathcal{T}$ to represent automata/transducers, while the corresponding languages/relations are denoted in roman fonts A, B, I, T.

A *finite automaton* (FA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. A *run* of $\mathcal{A}$ on a word $w = a_1 a_2 a_3 \cdots a_n$ is a sequence of states $q_0, q_1, \cdots, q_n$ such that $(q_i, a_{i+1}, q_{i+1}) \in \delta$. A run is *accepting* if the last state $q_n \in F$. A word is *accepted* by $\mathcal{A}$ if it has an accepting run. The *language* of $\mathcal{A}$, denoted by A, is the set of word accepted by $\mathcal{A}$. A language is *regular* if it can be recognised by a finite automaton. $\mathcal{A}$ is a *deterministic finite automaton* (DFA) if $|\{q' \mid (q,a,q') \in \delta\}| \leq 1$ for each $q \in Q$ and $a \in \Sigma$.

Let $\Sigma_\lambda = \Sigma \cup \{\lambda\}$. A *(finite) transducer* is a tuple $\mathcal{T} = (Q, \Sigma_\lambda, \delta, q_0, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Sigma_\lambda \times \Sigma_\lambda \times Q$ is a transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. We say that $\mathcal{T}$ is *length-preserving* if $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$. We define relation $\delta^* \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ as the smallest relation satisfying (1) $(q, \lambda, \lambda, q) \in \delta^*$ for any $q \in Q$ and (2) $(q_1, x, y, q_2) \in \delta^* \wedge (q_2, a, b, q_3) \in \delta \implies (q_1, xa, yb, q_3) \in \delta^*$. The relation represented by $\mathcal{T}$ is the set $\{(x,y) \mid (q_0, x, y, q) \in \delta^* \wedge q \in F\}$. A relation is *regular and length-preserving* if it can be represented by a length-preserving transducer.

III. Regular model checking

Regular model checking (RMC) is a uniform framework for modelling and automatically analysing parameterised concurrent systems. In the paper, we focus on the regular model checking framework for safety properties. Under the framework, each *system configuration* is represented as a word in Σ^*. The sets of *initial configurations* and of *bad configurations* are captured by regular languages over Σ. The *transition relation* is captured by a regular and length-preserving relation on Σ^*. We use a triple $(\mathcal{I}, \mathcal{T}, \mathcal{B})$ to denote a *regular model checking problem*, where $\mathcal{I}$ is an FA recognizing the set of initial configurations, $\mathcal{T}$ is a transducer representing the transition relation, and $\mathcal{B}$ is an FA recognizing the set of bad configurations. Then the regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$ asks if $T^*(I) \cap B = \emptyset$. A standard way to prove $T^*(I) \cap B = \emptyset$ is to find a proof based on a set V satisfying the following three conditions: (1)$I \subseteq V$ (i.e. all initial configurations are contained in V), (2) $V \cap B = \emptyset$ (i.e. V does not contain bad configurations), (3) $T(V) \subseteq V$ (i.e. V is *inductive*: applying T to any configuration in V does not take it outside V). We call the set V an *inductive invariant* for the regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. In the framework of regular model checking, a standard method for proving safety (e.g. see [55], [7]) is to find a *regular proof*, i.e., an inductive invariant that can be captured by finite automaton. Because regular languages are effectively closed under Boolean operations and taking pre-/post-images w.r.t. finite transducers, an algorithm for verifying whether a given regular language is an inductive invariant can be obtained by using language inclusion algorithms for FA [3], [19].

Example 1 (Herman's Protocol). *Herman's Protocol is a* self-stabilising *protocol for n processes (say with ids $0, \ldots, n-1$) organised as a ring structure. A* configuration *in the Herman's Protocol is* correct *iff only one process has a token. The protocol ensures that any system configuration where the processes collectively holding any odd number of tokens will almost surely be recovered to a correct configuration. More concretely, the protocol works iteratively. In each iteration, the scheduler randomly chooses a process. If the process with the number i is chosen by the scheduler, it will toss a coin to decide whether to keep the token or pass the token to the next process, i.e. the one with the number $(i+1)\%n$. If a process holds two tokens in the same iteration, it will discard* both *tokens. One safety property the protocol guarantees is that every system configuration has at least one token.*

The protocol and the corresponding safety property can be modelled as a regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. Each process has two states; the symbol T *denotes the state that the process has a token and* N *denotes the state that the process does not have a token. The word* NNTTNN *denotes a system configuration with six processes, where only the processes with numbers 2 and 3 are in the state with tokens. The set of initial configurations is $I = \mathsf{N}^* \mathsf{T} (\mathsf{N}^* \mathsf{T} \mathsf{N}^* \mathsf{T} \mathsf{N}^*)^*$, i.e., an odd number of processes has tokens. The set of bad configuration is $B = \mathsf{N}^*$, i.e., all tokens have disappeared. We use the regular* *language* $E = ((\mathsf{T}, \mathsf{T}) + (\mathsf{N}, \mathsf{N}))$ *to denote the relation that a process is idle. The transition relation* T *can be specified as a union of the following regular expressions: (1)* E^* *[Idle], (2)* $E^*(\mathsf{T}, \mathsf{N})(\mathsf{T}, \mathsf{N})E^* + (\mathsf{T}, \mathsf{N})E^*(\mathsf{T}, \mathsf{N})$ *[Discard both tokens], and (3)* $E^*(\mathsf{T}, \mathsf{N})(\mathsf{N}, \mathsf{T})E^* + (\mathsf{N}, \mathsf{T})E^*(\mathsf{T}, \mathsf{N})$ *[Pass the token].*

IV. Automata Learning

Suppose R is a regular *target* language whose definition is not directly accessible. *Automata learning* algorithms [8], [58], [44], [17] automatically infer a FA $\mathcal{A}$ recognising R. The setting of an online learning algorithm assumes a *teacher* who has access to R and can answer the following two queries: (1) Membership query $Mem(w)$: is the word w a member of R, i.e., $w \in R$? (2) Equivalence query $Equ(\mathcal{A})$: is the language of FA $\mathcal{A}$ equal to R, i.e., $A = R$? If not, it returns a counterexample $w \in A \ominus R$. The learning algorithm will then construct a FA $\mathcal{A}$ such that $A = R$ by interacting with the teacher. Such an algorithm works iteratively: In each iteration, it performs membership queries to get from the teacher information about R. Using the results of the queries, it proceeds by constructing a candidate automaton $\mathcal{A}_h$ and makes an equivalence query $Equ(\mathcal{A}_h)$. If $A_h = R$, the algorithm terminates with $\mathcal{A}_h$ as the resulting FA. Otherwise, the teacher returns a word w distinguishing A_h from R. The learning algorithm uses w to refine the candidate automaton of the next iteration. In the last decade, automata learning algorithms have been frequently applied to solve formal verification and synthesis problems, c.f., [27], [24], [38], [37], [26], [31].

More concretely, below we explain the details of the automata learning algorithm proposed by Rivest and Schapire [58] (RS), which is an improved version of the classic L^* learning algorithm by Angluin [8]. The foundation of the learning algorithm is the Myhill-Nerode theorem, from which one can infer that the states of the minimal DFA recognizing R are isomorphic to the set of equivalence classes defined by the following relations: $x \equiv_R y$ iff $\forall z \in \Sigma^* : xz \in R \leftrightarrow yz \in R$. Informally, two strings x and y belong to the same state of the minimal DFA recognising R iff they cannot be distinguished by any suffix z. In other words, if one can find a suffix z' such that $xz' \in R$ and $yz' \notin R$ or vice versa, then x and y belong to different states of the minimal DFA.

The algorithm uses a data structure called *observation table* (S, E, T) to find the equivalence classes correspond to $\equiv_R$, where S is a set of strings denoting the set of identified states, E is the set of suffixes to distinguish if two strings belong to the same state of the minimal DFA, and T is a mapping from $(S \cup (S \cdot \Sigma)) \cdot E$ to $\{\top, \bot\}$. The value of $T(w) = \top$ iff $w \in R$. We use $row_E(x) = row_E(y)$ as a shorthand for $\forall z \in E : T(xz) = T(yz)$. That is, the strings x and y cannot be identified as two different states using only strings in the set E as the suffixes. Observe that $x \equiv_R y$ implies $row_E(x) = row_E(y)$ for all $E \subseteq \Sigma^*$. We say that an observation table is *closed* iff $\forall x \in S, a \in \Sigma : \exists y \in S : row_E(xa) = row_E(y)$. Informally, with a closed table, every state can find its successors wrt. all symbols in Σ. Initially, $S = E = \{\lambda\}$, and $T(w) = Mem(w)$ for all $w \in \{\lambda\} \cup \Sigma$.

Algorithm 1: The improved L^* algorithm by Rivest and Schapire

```
Input: A teacher answers $Mem(w)$ and $Equ(\mathcal{A})$ about a
       target regular language $R$ and the initial
       observation table $(S, E, T)$.
repeat
    while $(S, E, T)$ is not closed do
        Find a pair $(x, a) \in S \times \Sigma$ such that
          $\forall y \in S : row_E(xa) \neq row_E(y)$. Extend $S$ to
          $S \cup \{xa\}$ and update $T$ using membership
          queries accordingly;
    Build a candidate DFA $\mathcal{A}_h = (S, \Sigma, \delta, \lambda, F)$, where
      $\delta = \{(s, a, s') \mid s, s' \in S \wedge row_E(sa) = row_E(s)\}$,
      the empty string $\lambda$ is the initial state, and
      $F = \{s \mid T(s) = \top \wedge s \in S\}$;
    if $Equ(\mathcal{A}_h) = (\mathsf{false}, w)$, where $w \in A \ominus R$ then
      Analyse $w$ and add a suffix of $w$ to $E$;
until $Equ(\mathcal{A}_h) = \mathsf{true}$;
return $\mathcal{A}_h$ is the minimal DFA for $R$;
```

The details of of the improved L^* algorithm by Rivest and Schapire can be found in Algorithm 1. Observe that, in the algorithm, two strings x, y with $x \equiv_R y$ will never be simultaneously contained in the set S. When the equivalence query $Equ(\mathcal{A})$ returns false together with a counterexample $w \in A \ominus R$, the algorithm will perform a binary search over w using membership queries to find a suffix e of w and extend E to $E \cup \{e\}$. The suffix e has the property that $\exists x, y \in S, a \in \Sigma : row_E(xa) = row_E(y) \wedge row_{E\cup\{e\}}(xa) \neq row_{E\cup\{e\}}(y)$, that is, add e to E will identify at least one more state. The existence of such a suffix is guaranteed. We refer the readers to [58] for the proof.

Proposition 1. *[58] Algorithm 1 will find the minimal DFA* $\mathcal{R}$ *for* R *using at most* n *equivalence queries and* $n(n + n|\Sigma|) + n \log m$ *membership queries, where* n *is the number of state of* $\mathcal{R}$ *and* m *is the length of the longest counterexample returned from the teacher.*

Because each equivalence query with a false answer will increase the size (number of states) of the candidate DFA by at least one and the size of the candidate DFA is bounded by n according to the Myhill-Nerode theorem, the learning algorithm uses at most n equivalence queries. The number of membership queries required to fill in the entire observation table is bounded by $n(n + n|\Sigma|)$. Since a binary search is used to analyse the counterexample and the number of counterexample from the teacher is bounded by n, the number of membership queries required is bounded by $n \log m$.

We would like to introduce the other two important variants of the L^* learning algorithm. The algorithm proposed by Kearns and Vazirani [44] (KV) uses a *classification tree* data structure to replace the observation table data structure of the classic L^* algorithm. The algorithm of Kearns and Vazirani has a similar query complexity to the one of Rivest and

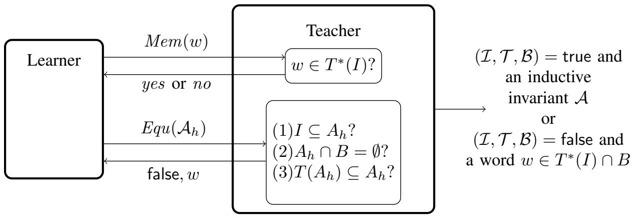

Learner
Teacher
$Mem(w)$
$w \in T^*(I)$?
yes or *no*
$Equ(\mathcal{A}_h)$
$(1) I \subseteq A_h$?
$(2) A_h \cap B = \emptyset$?
$(3) T(A_h) \subseteq A_h$?
false, w
$(\mathcal{I}, \mathcal{T}, \mathcal{B})$ = true and
an inductive
invariant $\mathcal{A}$
or
$(\mathcal{I}, \mathcal{T}, \mathcal{B})$ = false and
a word $w \in T^*(I) \cap B$

Fig. 1. Overview: using automata learning to solve the regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. Recall that we use calligraphy font for automata/transducers and roman font for the corresponding languages/relations.

Schapire [58]; it uses at most n equivalence queries and $n^2(n|\Sigma| + m)$ membership queries. However, the worst case bound of the number of membership queries is very loose. It assumes the structure of the classification tree is linear, i.e., each node has at most one child, which happens very rarely in practice. In our experience, the algorithm of Kearns and Vazirani usually requires a few more equivalence queries, with a significant lower number of membership queries comparing to Rivest and Schapire when applied to verification problems.

The NL^* algorithm [17] learns a non-deterministic finite automaton instead of a deterministic one. More concretely, it makes use of a canonical form of nondeterministic finite automaton, named *residual finite-state automaton (RFSA)* to express the target regular language. In some examples, RFSA can be exponentially more succinct than DFA recognising the same languages. In the worst case, the NL^* algorithm uses $O(n^2)$ equivalence queries and $O(m|\Sigma|n^3)$ membership queries to infer a canonical RFSA of the target language.

V. Algorithm

We apply automata learning algorithms, including Angluin's L^* and its variants, to solve the regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. Those learning algorithms require a teacher answering both equivalence and membership queries. Our strategy is to design a "strict teacher" targeting the minimal inductive invariant $T^*(I)$. For a membership query on a word w, the teacher checks if $w \in T^*(I)$, which is decidable under the assumption that $\mathcal{T}$ is length-preserving. For an equivalence query on a candidate FA $\mathcal{A}_h$, the teacher analyses if $\mathcal{A}_h$ can be used as an inductive invariant in a proof of the problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. It performs one of the following actions depending on the result of the analysis (Fig. 1):

- Determine that $\mathcal{A}_h$ does not represent an inductive invariant, and return false together with an explanation $w \in \Sigma^*$ to the learner.
- Conclude that $(\mathcal{I}, \mathcal{T}, \mathcal{B}) =$ true, and terminate the learning process with an inductive invariant $\mathcal{A}_h$ as the proof.
- Conclude that $(\mathcal{I}, \mathcal{T}, \mathcal{B}) =$ false, and terminate the learning with a word $w \in T^*(I) \cap B$ as an evidence.

Similar to the typical regular model checking approach, our learning-based technique tries to find a "regular proof", which amounts to finding an inductive invariant in the form of a regular language. Our approach is incomplete in general since it could happen that there only non-regular inductive invariants exist. Pathological cases where only non-regular inductive invariant exist do not, however, seem to occur frequently in practice, c.f., [21], [38], [20], [22], [60], [57], [50].

Answering a membership query on a word w, i.e., checking whether $w \in T^*(I)$, is the easy part: since $\mathcal{T}$ is length-preserving, we can construct an FA recognising $Post^{|w|} = \{w' \mid |w'| = |w| \wedge w' \in T^*(I)\}$ and then check if $w \in Post^{|w|}$. In practice, $Post^{|w|}$ can be efficiently computed and represented using BDDs and symbolic model checking.

For an equivalence query on a candidate FA $\mathcal{A}_h$, we need to check if $\mathcal{A}_h$ can be used as an inductive invariant for the regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. More concretely, we check the three conditions (1) $I \subseteq A_h$, (2) $A_h \cap B = \emptyset$, and (3) $T(A_h) \subseteq A_h$ using Algorithm 2.

Algorithm 2: Answer equivalence query on candidate FA

```
Input: An FA $\mathcal{A}_h$ and an RMC problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$
if $I \not\subseteq A_h$ then
    Find a word $w \in \mathcal{I} \setminus A_h$;
    return (false, $w$) to the learner;
else if $A_h \cap B \neq \emptyset$ then
    Find a word $w \in A_h \cap B$;
    if $w \in T^*(I)$ then Output {$cex = w$,
      $(\mathcal{I}, \mathcal{T}, \mathcal{B}) =$ false} and halt;
    else return (false, $w$) to the learner;
else if $T(A_h) \not\subseteq A_h$ then
    Find a pair of words $(w, w') \in T$ such that $w \in A_h$
      but $w' \notin A_h$;
    if $w \in T^*(I)$ then return (false, $w'$) to the learner;
    else return (false, $w$) to the learner;
else Output {$inv = \mathcal{A}_h$, $(\mathcal{I}, \mathcal{T}, \mathcal{B}) =$ true} and halt;
```

If the condition (1) is violated, i.e., $I \not\subseteq A_h$, there is a word $w \in I \setminus A_h$. Since $I \subseteq T^*(I)$, the teacher can infer that $w \in T^*(I) \setminus A_h$ and return w as a *positive* counterexample to the learner. A counterexample is positive if it represents a word in the target language that was missing in the candidate language. The definition negative counterexamples is symmetric.

If the condition (2) is violated, i.e., $A_h \cap B \neq \emptyset$, there is a word $w \in A_h \cap B$. The teacher checks if $w \in T^*(I)$ by constructing $Post^{|w|}$ and checking if $w \in Post^{|w|}$. If $w \notin T^*(I)$, the teacher obtains that $w \in A_h \setminus T^*(I)$ and returns false together with w as a negative counterexample to the learner. Otherwise, the teacher infers that $w \in T^*(I) \cap B$ and outputs $(\mathcal{I}, \mathcal{T}, \mathcal{B}) =$ false with the word w as an evidence.

The case that the condition (3) is violated, i.e., $T(A_h) \not\subseteq A_h$, is more involved. There exists a pair of words $(w, w') \in T$ such that $w \in A_h \wedge w' \notin A_h$. The teacher will check if $w \in T^*(I)$. If it is, then the teacher knows that $w' \in T^*(I) \wedge w' \notin A_h$ and hence returns false together with w' as a positive counterexample to the learner. If $w \notin T^*(I)$, then the teacher knows that $w \notin T^*(I) \wedge w \in A_h$ and hence returns false together with w as a negative counterexample to the learner.

If all conditions hold, the "strict teacher" shows its generosity (A_h might not equal to $T^*(I)$, but it will still pass) and concludes that $(\mathcal{I}, \mathcal{T}, \mathcal{B}) = \mathsf{true}$ with a proof using $\mathcal{A}_h$ as the inductive invariant.

Theorem 1 (Correctness)**.** *If the algorithm from Fig. 1 terminates, it gives correct answer to the RMC problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$.*

To see this, observe that the algorithm provides an inductive invariant when it concludes $(\mathcal{I}, \mathcal{T}, \mathcal{B}) = \mathsf{true}$ and a word in $T^*(I) \cap B$ when it concludes $(\mathcal{I}, \mathcal{T}, \mathcal{B}) = \mathsf{false}$. In addition, if one of the L^* learning algorithms[1] from Section IV is used, we can obtain an additional result about termination:

Theorem 2 (Termination)**.** *When $T^*(I)$ is regular, the algorithm from Fig. 1 is guaranteed to terminate in at most k iterations, where k is the size of the minimal DFA of $T^*(I)$.*

Proof. Observe that in the algorithm, the counterexample obtained by the learner in each iteration locates in the symmetric difference of the candidate language and $T^*(I)$. Hence, when $T^*(I)$ can be recognized by a DFA of k states, the algorithm will not execute more than k iterations by Proposition 1. □

Two remarks are in order. Firstly, the set $T^*(I)$ tends to be regular in practice, e.g., see [21], [38], [20], [22], [60], [57], [10], [11], [50], [49]. In fact, it is known that $T^*(I)$ is regular for many subclasses of infinite-state systems that can be modelled in regular model checking [60], [50], [40], [11], [49] including pushdown systems, reversal-bounded counter systems, two-dimensional VASS (Vector Addition Systems with States), and other subclasses of counter systems. Secondly, even in the case when $T^*(I)$ is not regular, termination may still happen due to the "generosity" of the teacher, which will accept any inductive invariant as an answer.

Considerations on Implementation: The implementation of the learning-based algorithm is very simple. Since it is based on standard automata learning algorithms and uses only basic automata/transducer operations, one can find existing libraries for them. The implementation only need to take care of how to answer queries. The core of our implementation has only around 150 lines of code (excluding the parser of the input models). We provide a few suggestions to make the implementation more efficient. First, each time when an FA recognising $Post^k$ is produced, we store the pair $(k, Post^k)$ in a cache. It can be reused when a query on any word of length k is posed. We can also check if $Post^k \cap B = \emptyset$. The algorithm can immediately terminate and return $(\mathcal{I}, \mathcal{T}, \mathcal{B}) = \mathsf{false}$ if $Post^k \cap B \neq \emptyset$. Second, for each language inclusion test, if the inclusion does not hold, we suggest to return the shortest counterexample. This heuristic helped to shorten the average length of strings sent for membership queries and hence reduced the cost of answering them. Recall that the algorithm needs to build the FA of $Post^k$ to answer membership queries. The shorter the average length of query strings is, the fewer instances of $Post^k$ have to be built.

VI. Evaluation

To evaluate our techniques, we have developed a prototype[2] in Java and used the libalf library [18] as the default inference engine. We used our prototype to check safety properties for a range of parameterised systems, including cache coherence protocols (German's Protocol), self-stabilising protocols (Israeli-Jalfon's Protocol and Herman's Protocol), synchronisation protocols (Lehmann-Rabin's Dining Philosopher Protocol), secure multi-party computation protocol (David Chaums' Dining Cryptographers Protocol), and mutual exclusion protocols (Szymanski's Protocol, Burn's Protocol, Dijkstra's Protocol, Lamport's Bakery Algorithm, and Resource-Allocator Protocol). Most of the examples we consider are standard benchmarks in the literature of regular model checking (c.f. [4], [5], [20], [22], [57]). Among them, German's Protocol and Kanban are more difficult than the other examples for fully automatic verification (c.f. [4], [5], [43]).

Based on these examples, we compare our learning method with existing techniques such as SAT-based learning [54], [55], [51], [52], extrapolating [16], [46], and abstract regular model checking (ARMC) [20], [21]. The SAT-based learning approach encodes automata as Boolean formulae and exploits a SAT-solver to search for candidate automata representing inductive invariants. It uses automata-based algorithms to either verify the correctness of the candidate or obtain a counterexample that can be further encoded as a Boolean constraint. T(O)RMC [16], [46] extrapolates the limit of the reachable configurations represented by an infinite sequence of automata. The extrapolation is computed by first identifying the increment between successive automata, and then over-approximating the repetition of the increment by adding loops to the automata. ARMC is an efficient technique that integrates abstraction refinement into the fixed-point computation. It begins with an existential abstraction obtained by merging states in the automata/transducers. Each time a spurious counterexample is found, the abstraction can be refined by splitting some of the merged states. ARMC is among the most efficient algorithms for regular model checking [38].

The comparison of those algorithms are reported in Table I, running on a MinGW64 system with 3GHz Intel i7 processor, 2GB memory limit, and 60-second timeout. The experiments show that the learning method is quite efficient: the results of our prototype are comparable with those of the ARMC algorithm[3] on all examples but Kanban, for which the minimal inductive invariant, if it is regular, has at least 400 states. On the other hand, our algorithm is significantly faster than ARMC in two cases, namely German's Protocol and Dining Cryptographers. ARMC comes with a bundle of options and heuristics, but not all of them work for our benchmarks. We have tested all the heuristics available from the tool and adopted the ones[4] that had the best performance in our experi-

[1] If NL^* is used, the bound in Theorem 2 will increase to $O(k^2)$.

[2] Available at https://github.com/ericpony/safety-prover.

[3] Available at http://www.fit.vutbr.cz/research/groups/verifit/tools/hades.

[4] The heuristics are structure preserving, backward computation, and backward collapsing with all states being predicates. See [21] for explanations.

The RMC problems								RS			SAT			T(O)RMC			ARMC
Name	#label	S_{init}	T_{init}	S_{trans}	T_{trans}	S_{bad}	T_{bad}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time
Bakery [32]	3	3	3	5	19	3	9	0.0s	6	18	0.5s	2	5	0.0s	6	11	0.0s
Burns [4]	12	3	3	10	125	3	36	0.2s	8	96	1.1s	2	10	0.1s	7	38	0.0s
Szymanski [59]	11	9	9	118	412	13	40	0.3s	43	473	1.6s	2	21	2.0s	51	102	0.1s
German [36]	581	3	3	17	9.5k	4	2112	4.8s	14	8134	t.o.	–	–	t.o.	–	–	10s
Dijkstra [4]	42	1	1	13	827	3	126	0.1s	9	378	1.7s	2	24	6.1s	8	83	0.3s
Dijkstra, ring [28], [33]	12	3	3	13	199	3	36	1.4s	22	264	0.9s	2	14	t.o.	–	–	0.1s
Dining Crypto. [25]	14	10	30	17	70	12	70	0.1s	32	448	t.o.	–	–	t.o.	–	–	7.2s
Coffee Can [52]	6	8	18	13	34	5	8	0.0s	3	18	0.2s	2	7	0.1s	6	13	0.0s
Herman, linear [39]	2	2	4	4	10	1	1	0.0s	2	4	0.2s	2	4	0.0s	2	4	0.0s
Herman, ring [39]	2	2	4	9	22	1	1	0.0s	2	4	0.4s	2	4	0.0s	2	4	0.0s
Israeli-Jalfon [41]	2	3	6	24	62	1	1	0.0s	4	8	0.1s	2	4	0.0s	4	8	0.0s
Lehmann-Rabin [47]	6	4	4	14	96	3	13	0.1s	8	48	0.5s	2	11	0.8s	19	105	0.0s
LR Dining Philo. [52]	4	4	4	3	10	3	4	0.0s	4	16	0.2s	2	6	0.1s	7	18	0.0s
Mux Array [33]	6	3	3	4	31	3	18	0.0s	5	30	0.4s	2	7	0.2s	4	14	0.0s
Res. Allocator [29]	3	3	3	7	25	4	9	0.0s	5	15	0.0s	1	3	0.0s	4	9	0.0s
Kanban [5], [43]	3	25	48	98	250	37	68	t.o.	–	–	t.o.	–	–	t.o.	–	–	3.5s
Water Jugs [64]	11	5	6	23	132	5	12	0.1s	24	264	t.o.	–	–	t.o.	–	–	0.0s

TABLE I

Comparing the performance of different RMC techniques. #label stands for the size of alphabet; S_x and T_x stand for the numbers of states and transitions, respectively, in the automata/transducers. **RS** is the result of our prototype using Rivest and Schapire's version of L^*; **SAT**, **T(O)RMC**, and **ARMC** are the results of the other three techniques.

ments. The performance of SAT-based learning is comparable to the previous two approaches whenever inductive invariants representable by automata with few states exist. However, as its runtime grows exponentially with the sizes of candidate automata, the SAT-based algorithm fails to solve four examples that do not have small regular inductive invariants. T(O)RMC seems to suffer from similar problems as it timeouts on all examples that cannot be proved by the SAT-based approach.

Table II reports the results of the learning-based algorithm geared with different automata learning algorithms implemented in libalf. As the table shows, these algorithms have similar performance on small examples; however, the algorithm of Rivest and Schapire [58] and the algorithm of Kearns and Varzirani [44] are significantly more efficient than the other algorithms on some large examples such as Szymanski and German. Table II shows that Kearns and Varzirani's algorithm can often find smaller inductive invariants (fewer states) than the other L^* variants, which explains the performance difference. For NL^*, our implementation pays an additional cost to determinise the learned FA in order to answer the equivalence queries; this cost is significant when a large invariant is needed.

Recall that our approach uses a "strict but generous teacher". Namely, the target language of the teacher is $T^*(I)$ for an RMC problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$. We have tried the version where a "flexible and generous teacher" is used, that is, the target language of the teacher is the complement of $(T^{-1})^*(B)$. The performance, however, is worse than that of our current version. This result may reflect the fact that the set $T^*(I)$ is "more regular" (i.e., can be expressed by a DFA with fewer states) than the set $(T^{-1})^*(B)$ in practical cases.

VII. Conclusion

The encouraging experimental results suggest that the performance of the L^* algorithm for synthesising regular inductive invariants is comparable to the most sophisticated algorithm for regular model checking for proving safety. From a theoretical viewpoint, learning-based approaches (including ours and [54], [55], [38]) have a termination guarantee when the set $T^*(I)$ is regular, which is not guaranteed by approaches based on a fixed-point computation (e.g., the ARMC [21]). An interesting research question is whether L^* algorithm can be effectively used for verifying other properties, e.g., liveness.

Acknowledgements: We thank anonymous referees for their useful comments. Rümmer was supported by the Swedish Research Council under grant 2014-5484.

References

[1] P. A. Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012.

[2] P. A. Abdulla, M. F. Atig, Y. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. String constraints for verification. In *CAV'14*, pages 150–166.

[3] P. A. Abdulla, Y. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS'10*, pages 158–174.

[4] P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS'07*, pages 721–736.

[5] P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI'13*, pages 476–495.

[6] P. A. Abdulla, B. Jonsson, M. Nilsson, J. d'Orso, and M. Saksena. Regular model checking for LTL(MSO). *STTT*, 14(2):223–241, 2012.

[7] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR'04*, pages 35–48.

[8] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[9] K. R. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *IPL*, 22(6):307–309, 1986.

[10] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

[11] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA'05*, pages 474–488.

[12] N. Bertrand and P. Fournier. Parameterized verification of many identical probabilistic timed processes. In *FSTTCS'13*, volume 24 of *LIPIcs*, pages 501–513. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[13] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

[14] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1999.

[15] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *CAV'03*, pages 223–235.

[16] B. Boigelot, A. Legay, and P. Wolper. Omega-regular model checking. In *TACAS'04*, pages 561–575.

[17] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *IJCAI'09*, pages 1004–1009.

	RS			**L****			**L*****c**			**KV**			**NL***		
	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}
Bakery	0.0s	6	18	0.0s	6	18	0.1s	6	18	0.0s	6	18	0.1s	6	18
Burns	0.2s	8	96	0.5s	8	96	0.2s	8	96	0.2s	8	96	0.4s	6	72
Szymanski	0.3s	43	473	2.4s	51	561	1.2s	41	451	0.3s	41	451	1.4s	59	649
German	4.8s	14	8134	13s	15	8715	26s	15	8715	4.2s	14	8134	40s	15	8715
Dijkstra	0.1s	9	378	0.4s	9	378	0.1s	9	378	0.2s	9	378	0.2s	10	420
Dijkstra, ring	1.4s	22	264	2.7s	20	240	8.9s	22	264	1.5s	14	168	1.8s	20	240
Dining Crypto.	0.1s	32	448	0.2s	34	476	0.2s	38	532	0.1s	19	266	0.3s	36	504
Coffee Can	0.0s	3	18	0.0s	3	18	0.0s	4	24	0.0s	3	18	0.0s	4	24
Herman, linear	0.0s	2	4	0.0s	2	4	0.0s	2	4	0.0s	2	4	0.0s	2	4
Herman, ring	0.0s	2	4	0.0s	2	4	0.0s	2	4	0.0s	2	4	0.0s	2	4
Israeli-Jalfon	0.0s	4	8	0.0s	4	8	0.0s	4	8	0.0s	4	8	0.0s	4	8
Lehmann-Rabin	0.1s	8	48	0.2s	8	48	0.1s	8	48	0.1s	8	48	0.2s	8	48
LR D. Philo.	0.0s	4	16	0.2s	4	16	0.0s	5	20	0.0s	4	16	0.0s	8	32
Mux Array	0.0s	5	30	0.0s	5	30	0.0s	5	30	0.0s	5	30	0.0s	5	30
Res. Allocator	0.0s	5	15	0.0s	4	12	0.0s	5	15	0.0s	5	15	0.0s	5	15
Kanban	>60s	–	–	>60s	–	–	>60s	–	–	>60s	–	–	>60s	–	–
Water Jugs	0.1s	24	264	0.5s	25	275	0.5s	25	275	0.1s	24	264	0.5s	25	275

TABLE II

COMPARING THE PERFORMANCE BASED ON DIFFERENT AUTOMATA LEARNING ALGORITHMS. THE COLUMNS $\mathbf{L}^*$, $\mathbf{L}^*\mathbf{c}$, $\mathbf{RS}$, $\mathbf{KV}$, AND $\mathbf{NL}^*$ ARE THE RESULTS OF THE ORIGINAL L^* ALGORITHM BY ANGLUIN [8], A VARIANT OF L^* THAT ADDS ALL SUFFIXES OF THE COUNTEREXAMPLE TO COLUMNS, THE VERSION BY RIVEST AND SHAPIRE [58], THE VERSION BY KEARNS AND VAZIRANI [44], AND THE NL^* ALGORITHM [17], RESPECTIVELY.

[18] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. libalf: The automata learning framework. In *CAV'10*, pages 360–364.

[19] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL'13*, pages 457–468.

[20] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking. *ENTCS*, 149(1):37–48, 2006.

[21] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV'04*, pages 372–386.

[22] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV'00*, pages 403–418.

[23] A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. *STTT*, 14(2):145–165, 2012.

[24] M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the language of error. In *ATVA'15*, pages 114–130.

[25] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[26] Y. Chen, A. Farzan, E. M. Clarke, Y. Tsay, and B. Wang. Learning minimal separating DFA's for compositional verification. In *TACAS'09*, pages 31–45.

[27] Y. Chen, C. Hsieh, O. Lengál, T. Lii, M. Tsai, B. Wang, and F. Wang. PAC learning-based verification and model synthesis. In *ICSE'16*, pages 714–724.

[28] E. W. Dijkstra, R. Bird, M. Rogers, and O.-J. Dahl. Invariance and non-determinacy [and discussion]. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 312(1522):491–499, 1984.

[29] A. F. Donaldson. *Automatic techniques for detecting and exploiting symmetry in model checking*. PhD thesis, University of Glasgow, 2007.

[30] J. Esparza. Parameterized verification of crowds of anonymous processes. *Dependable Software Systems Engineering*, 45:59–71, 2016.

[31] A. Farzan, Y. Chen, E. M. Clarke, Y. Tsay, and B. Wang. Extending automated compositional verification to the full class of omega-regular languages. In *TACAS'08*, pages 2–17.

[32] W. Fokkink. *Distributed Algorithms*. MIT Press, 2013.

[33] L. Fribourg and H. Olsén. Reachability sets of parameterized rings as regular languages. *ENTCS*, 9:40, 1997.

[34] P. Garg, C. Löding, P. Madhusudan, and D. Neider. ICE: A robust framework for learning invariants. In *CAV'14*, pages 69–87.

[35] P. Garg, C. Löding, P. Madhusudan, and D. Neider. Learning universally quantified invariants of linear data structures. In *CAV'13*, pages 813–829.

[36] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *JACM*, 39(3):675–735, 1992.

[37] O. Grinchtein, M. Leucker, and N. Piterman. Inferring network invariants automatically. In *IJCAR'06*, pages 483–497.

[38] P. Habermehl and T. Vojnar. Regular model checking using inference of regular languages. *ENTCS*, 138(3):21–36, 2005.

[39] T. Herman. Probabilistic self-stabilization. *IPL*, 35(2):63–67, 1990.

[40] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[41] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC'90*, pages 119–131.

[42] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00*, pages 220–234.

[43] A. Kaiser, D. Kroening, and T. Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV'10*, pages 645–659.

[44] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.

[45] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256(1-2):93–112, 2001.

[46] A. Legay. T(O)RMC: A tool for (ω)-regular model checking. In *CAV'08*, pages 548–551.

[47] D. Lehmann and M. O. Rabin. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *POPL'81*, pages 133–138.

[48] O. Lengál, A. W. Lin, R. Majumdar, and P. Rümmer. Fair termination for parameterized probabilistic concurrent systems. In *TACAS'17*.

[49] J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *ATVA'05*, pages 489–503.

[50] A. W. Lin. Accelerating tree-automatic relations. In *FSTTCS'12*, pages 313–324.

[51] A. W. Lin, T. K. Nguyen, P. Rümmer, and J. Sun. Regular symmetry patterns. In *VMCAI'16*, pages 455–475.

[52] A. W. Lin and P. Rümmer. Liveness of randomised parameterised systems under arbitrary schedulers. In *CAV'16*, pages 112–133.

[53] N. A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *PODC'94*, pages 314–323.

[54] D. Neider. *Applications of Automata Learning in Verification and Synthesis*. PhD thesis, RWTH Aachen, 2014.

[55] D. Neider and N. Jansen. Regular model checking using solver technologies and automata learning. In *NFM*, pages 16–31, 2013.

[56] D. Neider and U. Topcu. An automaton learning approach to solving safety games over infinite graphs. In *TACAS'16*, pages 204–221.

[57] M. Nilsson. *Regular Model Checking*. PhD thesis, Uppsala Univ., 2005.

[58] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.

[59] B. K. Szymanski. A simple solution to Lamport's concurrent programming problem with linear wait. In *ICS'88*, pages 621–626.

[60] A. W. To and L. Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *FoSSaCS'10*, pages 221–236.

[61] A. Vardhan. *Learning To Verify Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2006.

[62] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. In *ICFME'04*, pages 274–289.

[63] A. Vardhan and M. Viswanathan. LEVER: A tool for learning based verification. In *CAV'06*, pages 471–474.

[64] Wikipedia. Liquid water pouring puzzles. https://en.wikipedia.org/w/index.php?title=Liquid_water_pouring_puzzles&oldid=764748113, 2017. [Accessed: 24-February-2017].

[65] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*, pages 88–97.

	λ
λ	$\bot$
T	$\top$
N	$\bot$
TT	$\bot$
TN	$\top$

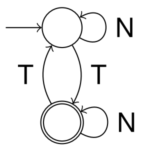


Fig. 2. Using learning to solve the RMC problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$ of Herman's Protocol. The table on the left is the content of the *observation table* used by the automata learning algorithm of Rivest and Shaphire [58] and the automaton on the right is the inferred candidate DFA.

VIII. Appendix

We provide some more examples for regular model checking in the appendix.

Example 2 (RMC of the Herman's Protocol)**.** *Consider the RMC problem of Herman's Protocol in Example 1. Initially, several membership queries will be posed to the teacher to produce the closed observation table on the left of Fig. 2. In this example, the teacher returns $\top$ only for words containing an odd number of the symbol T. The learner will then construct the candidate FA $\mathcal{A}_h$ on the right of Fig. 2 and pose an equivalence query on $\mathcal{A}_h$. Observe that $\mathcal{A}_h$ can be used as an inductive invariant in a regular proof. It is easy to verify that $I = \mathsf{N}^*\mathsf{T}(\mathsf{N}^*\mathsf{T}\mathsf{N}^*\mathsf{T}\mathsf{N}^*)^* \subseteq A_h$ and $A_h \cap B = A_h \cap \mathsf{N}^* = \emptyset$. The condition (3) $T(A_h) \subseteq A_h$ of a regular proof can be proved to be correct based on the following observation: the FA $\mathcal{A}_h$ recognises exactly the set of all configurations with an odd number of tokens. When tokens are discarded in a transition, the total number of discarded tokens in all processes is always 2. The other two types of transitions will not change the total number of tokens in the system. It follows that taking a transition from any configurations in A_h will arrive a configuration with an odd number of tokens, which is still in A_h.*

The verification of Herman's Protocol finishes after the first iteration of learning and hence we cannot see how the learning algorithm uses a counterexample for refinement. Below we introduce a slightly more difficult problem.

Example 3 (RMC of the Israeli-Jalfon's Protocol)**.** *Israeli-Jalfon's Protocol is a routing protocol of n processes organised in a ring-shaped topology, where a process may hold a token. Again we assume the processes are numbered from 0 to $n-1$. If the process with the number i is chosen by the scheduler, it will toss a coin to decide to pass the token to the left or right, i.e. the one with the number $(i-1)\%n$ or $(i+1)\%n$. When two tokens are held by the same process, they will be merged. The safety property of interest is that every system configuration has at least one token. The protocol and the corresponding safety property can be modelled as a regular model checking problem $(\mathcal{I}, \mathcal{T}, \mathcal{B})$, together with the set of initial configurations $I = (\mathsf{T}+\mathsf{N})^*\mathsf{T}(\mathsf{T}+\mathsf{N})^*\mathsf{T}(\mathsf{T}+\mathsf{N})^*$, i.e., at least two processes have tokens, and the set of bad configurations $B = \mathsf{N}^*$, i.e., all tokens have disappeared. Again we use the regular language $E = ((\mathsf{T},\mathsf{T}) + (\mathsf{N},\mathsf{N}))$*

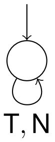
to denote the relation that a process is idle, i.e, the process does not change its state. The transition relation T then can be specified as a union of the regular expressions in Figure 3.

$$\begin{array}{ll} E^*(\mathsf{T},\mathsf{N})((\mathsf{T},\mathsf{T})+(\mathsf{N},\mathsf{T}))E^*,\ ((\mathsf{T},\mathsf{T})+(\mathsf{N},\mathsf{T}))E^*(\mathsf{T},\mathsf{N}) & (\textit{Pass the token right}) \\ E^*((\mathsf{T},\mathsf{T})+(\mathsf{N},\mathsf{T}))(\mathsf{T},\mathsf{N})E^*,\ (\mathsf{T},\mathsf{N})E^*((\mathsf{T},\mathsf{T})+(\mathsf{N},\mathsf{T})) & (\textit{Pass the token left}) \end{array}$$

Fig. 3. The Transition Relation of Israeli-Jalfon's Protocol

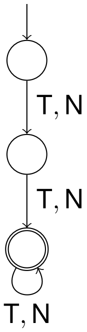
When the automata learning algorithm of Rivest and Shaphire is applied to solve the RMC problem, we can obtain an inductive invariant with 4 states in 3 iterations. The first candidate FA in Fig. 4(a) is incorrect because it does not include the initial configuration TT*. By analysing the counterexample* TT*, the learning algorithm adds the suffix* T *to the set E. The second candidate FA in Fig. 4(b) is still incorrect because it contains an unreachable bad configuration* NNN*. The learning algorithm analyses the counterexample* NNN *and adds the suffix* N *to the set E. This time it obtains the candidate FA in Fig. 4(c), which is a valid regular inductive invariant.*

	λ
λ	$\bot$
T	$\bot$
N	$\bot$



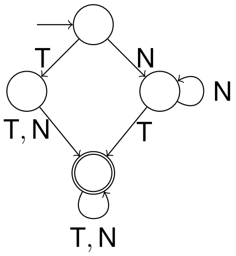
(a) First candidate automaton

	λ	T
λ	$\bot$	$\bot$
T	$\bot$	$\top$
TT	$\top$	$\top$
N	$\bot$	$\top$
TN	$\top$	$\top$
TTT	$\top$	$\top$
TTN	$\top$	$\top$



(b) Second candidate automaton

	λ	T	N
λ	$\bot$	$\bot$	$\bot$
T	$\bot$	$\top$	$\top$
N	$\bot$	$\top$	$\bot$
TT	$\top$	$\top$	$\top$
TN	$\top$	$\top$	$\top$
NT	$\top$	$\top$	$\top$
NN	$\top$	$\top$	$\bot$
TTT	$\top$	$\top$	$\top$
TTN	$\top$	$\top$	$\top$



(c) Third candidate automaton

Fig. 4. Using learning to solve the RMC problem $(\mathcal{I},\mathcal{T},\mathcal{B})$ of Israeli-Jalfon's Protocol. The table on the left of each sub-figure is the content of the *observation table* used by the automata learning algorithm of Rivest and Shaphire [58].