Using Language Inference to Verify omega-regular Properties

Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, Gul Agha *
Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, USA
{vardhan,ksen,vmahesh,agha}@cs.uiuc.edu

Abstract. A novel machine learning based approach was proposed recently as a complementary technique to the acceleration based methods for verifying infinite state systems. In this method, the set of states satisfying a fixpoint property is learnt as opposed to being iteratively computed. We extend the machine learning based approach to verifying general ω-regular properties that include both safety and liveness. To achieve this, we first develop a new fixpoint based characterization for the verification of ω-regular properties. Using this characterization, we present a general framework for verifying infinite state systems. We then instantiate our approach to the context of regular model checking where states are represented as strings over a finite alphabet and the transition relation of the system is given as a finite state transducer; unlike previous learning based algorithms, we make no assumption about the transducer being length-preserving. Using Angluin's L* algorithm for learning regular languages, we develop an algorithm for verification of ω-regular properties of such infinite state systems. The algorithm is a complete verification procedure for systems for whom the fixpoint can be represented as a regular set. We have implemented the technique in a tool called Lever and use it to analyze some examples.

1 Introduction

Automated verification of systems with respect to temporal properties involves computing fixpoints of functionals on sets of states of the system. This is often calculated by iteratively computing approximations to the fixpoint, until the process converges. When verifying infinite state systems, this iterative computation must necessarily be performed symbolically, using a suitably chosen representation for sets of states. However, since fixpoint computations are no longer guaranteed to converge within finitely many steps, a variety of *acceleration methods*, such as widening [15, 4] and abstraction [3], have been proposed. These methods have been used successfully to verify many practical examples and can be used to obtain complete verification procedures for special subclasses of systems (such as bounded local depth or simple transition relations [10]).

* The third author was supported in part by DARPA/AFOSR MURI Award F49620-02-1-0325 and NSF 04-29639. The other three authors were supported in part by DARPA IPTO TASK Program (contract F30602-00-2-0586), ONR Grant N00014-02-1-0715, and Motorola Grant MOTOROLA RPS #23 ANT

Recently, a complementary, machine learning based approach has been independently proposed in [17] and [9]. In this approach, the fixpoint is *learnt* from examples of states belonging to the fixpoint and states not belonging to the fixpoint. The advantage of the learning based approach is that termination does not depend on how long it takes to converge to the fixpoint, and hence this approach yields a complete verification procedure even when the fixpoint does not converge within a finite bound. Second, because intermediate approximations to the fixpoint are never computed, it avoids the space overhead of storing fixpoint approximations that may have a large symbolic representation. Preliminary experimental results based on this approach are promising [17, 9, 16].

In this paper, we present a general framework to verify infinite state systems with respect to specifications presented as non-deterministic Büchi automata. One of the central requirements of our framework is a learning algorithm that can learn concepts encoded using a chosen symbolic representation. The learning algorithm is used to learn the fixpoint of a specific function, such that the initial state of the system belongs to the learnt set if and only if the system satisfies the specification. This yields a complete verification procedure, *provided* the fixpoint can be represented in the chosen representation. We then instantiate the framework to the specific context of regular model checking, where states are encoded as strings over some finite alphabet, and the system's transition relation is presented as a transducer over such strings. Unlike previous work in this area, we *do not* assume that the transducer is length preserving. If the fixpoint of our functional can be expressed as a regular language, then our algorithm is guaranteed to terminate and either prove the system to be correct or demonstrate that it is faulty. We use Angluin's L* [2] algorithm to learn the regular set representation of the fixpoint.

The results presented here significantly advance the state of the art in learning based verification. First, our method verifies ω-regular properties which can express safety as well as liveness properties. This generalizes our previous work on safety properties reported in [17, 16]. Second, our instantiation to regular model checking is not confined to analyzing systems such as FIFO automata. We also do not need the transition relation to be restricted to be length-preserving as has been assumed in some other approaches such as [9]. Moreover, our general framework can potentially be used to verify systems symbolically represented using *polyhedra* or *ellipsoids*, not just regular languages, provided appropriate learning algorithms can be plugged in. Third, our algorithm for checking containment of the system's trace language in the specification automata's language, is not based on discovering loops where final states of the automata are visited infinitely often (as is the case in [9]). Thus, our algorithm will successfully identify faulty systems, even when there is no ultimately periodic execution that witnesses the violation. This is important because for general infinite state systems, it is often the case that there is no such ultimately periodic execution witnessing the violation of a liveness property. Finally, since we use Angluin's L* algorithm, we are guaranteed to not only learn the smallest automaton representing the fixpoint, but are also guaranteed to only make polynomially many calls to the learning algorithm.

The rest of the paper is organized as follows. We first outline results that are closely related to this paper. In Section 2, we introduce basic concepts and notation that are used in the paper. The general learning based verification framework for ω-regular languages is presented in Section 3. We first identify a functional whose fixpoint helps us verify ω-regular properties (Section 3.1) and then show how a learning algorithm can be used to compute the fixpoint of this functional (Section 3.2). In Section 4, we instantiate this general framework to the specific context of regular model checking, where states are represented as strings over a finite alphabet and the system's transition relation is represented as a transducer. We give detailed algorithms for the various operations that are needed in the learning based algorithm. Finally, in Section 5 we discuss the analysis of two examples using the implementation of this verification method in a tool called Lever and present our conclusions in Section 6.

Related Work. We introduced the learning to verify approach in [17], where we used RPNI [12] to learn the regular set from positive and negative examples without active queries. In [16], we improved our learning procedure for FIFO automata by using a more powerful active learning framework and a better encoding for witnesses for membership queries. Concurrently and independently of our work, Habermehl *et al.* [9] have also proposed a learning based approach for verification of systems whose transition can be represented by a length-preserving transducer. The algorithm presented there crucially depends on the length-preserving nature of the transition relation for its completeness. An earlier use of regular inference techniques for reachability in parameterized rings of processes also appears in [8]. Verification of ω-regular properties for infinite state systems has also been addressed in [4] and [13]. Abdulla *et al.* [1] present a "two-dimensional" modal logic called LTL(MSO) for verification of liveness properties. The above approaches rely on loop detection for checking liveness and assume that the transition relation is length preserving. Recently, Bouajjani *et al.* [5] have analyzed liveness properties of non-length preserving systems using a notion of simulation between states.

2 Preliminaries

In this section, we present the learning framework that we will consider in this paper and basic definitions of Kripke structures and Büchi automata.

2.1 Learning with membership and equivalence queries

A learning algorithm is usually set in a framework which describes the types of input data and queries available to the learner. In the framework of *active learning* [2]), the learning algorithm is given access to a knowledgeable teacher, often called a *minimally adequate teacher.* The knowledgeable teacher can be thought of as a pair of oracles: a *membership oracle* and an *equivalence oracle*. The membership oracle provides answers to queries about whether an example belongs to the concept being learnt or not. The equivalence oracle is a more powerful oracle which answers question about whether a hypothesis proposed by the learning algorithm is indeed equivalent to the concept being learnt. If at some point the learning algorithm's hypothesis is deemed correct by the equivalence oracle then the learning process stops. If on the other hand, the learner submits a hypothesis which is not equivalent to the target concept, the equivalence oracle not only says no, but also provides a counter-example to demonstrate when the hypothesis is wrong. The counter-example is either an example belonging to the hypothesis but not to the target concept, or it is an example belonging to the target concept but not to the submitted hypothesis. The active learning framework can be contrasted with the *passive learning* framework where the learner is simply provided a set of examples labeled as either belonging to the target concept or not; there is no knowledgeable teacher involved. The active learning algorithm is a powerful framework that in many cases admits efficient learning of concepts which otherwise cannot be learnt passively.

Our learning based verification approach uses a learning algorithm in the active learning framework. In particular, when we instantiate our learning based approach to verify a class of infinite state systems, we use a classical algorithm due to Angluin [2] which learns the smallest automaton recognizing the regular language, when it is allowed to interact with a knowledgeable teacher. Angluin's L* algorithm is also highly efficient; it can be shown that the number of queries made to the membership and equivalence oracles by the learning algorithm is bounded by a polynomial in the size of the smallest DFA recognizing the regular language. The main idea behind Angluin's L* algorithm is to systematically explore strings in the alphabet for membership and create a DFA with minimum number of states to make a conjecture for the target set. If the conjecture is incorrect, the string returned by the teacher is used to make corrections, possibly after more membership queries. The algorithm maintains a prefix closed set S representing different possible states of the target DFA, a set SA for the transition function consisting of strings from S extended with one letter of the alphabet, and a suffix closed set E denoting *experiments* to distinguish between states. An *observation table* with rows from $(S \cup SA)$ and columns from E stores results of the membership queries for strings in $(S \cup SA).E$ and is used to create the DFA for a conjecture.

2.2 Kripke structures and Büchi automaton

We use Kripke structure to model the system being verified and Büchi automaton for the specification. We now formally define these.

Kripke structure. A *Kripke structure* K is a quintuple $(S^k, \Sigma, R^k, S_0^k, \mathcal{L})$ where S^k is the set of (possibly infinite) states, Σ is a finite alphabet, $R^k \subseteq S^k \times S^k$ is the (total) transition relation, $S_0^k \subseteq S^k$ is the set of initial states and $\mathcal{L} : S^k \to \Sigma$ is the labeling function. We restrict ourselves to Kripke structures that are finitely branching, *i.e.*, for any state s, the set $\{s' \mid R^k(s,s')\}$ is finite. We say $s \to s'$ iff $(s,s') \in R^k$. A *path starting from state* s is an infinite sequence $s_0, s_1, s_2 \ldots$ such that $s = s_0$ and for every i, $(s_i, s_{i+1}) \in R^k$. A *path* of a Kripke structure K is just a path starting from some initial state $s \in S_0^k$. The set of all paths of K will be denoted by $\mathcal{P}(K)$. For a path $\pi = s_0, s_1, s_2, \ldots$, $\mathit{KTrace}(\pi)$ is the sequence of labels $\ell_0, \ell_1, \ell_2, \ldots$ such that for every i, $\mathcal{L}(s_i) = \ell_i$. For a set of paths Π, $\mathit{KTrace}(\Pi)$ is taken to be $\{\mathit{KTrace}(\pi) \mid \pi \in \Pi\}$.

Büchi automaton. A *Büchi automaton* [14] M is a quintuple $(S^m, \Sigma, S_0^m, \delta, F^m)$ where S^m is a finite set of states, $S_0^m \subseteq S^m$ is the set of initial states, $\delta : S^m \times \Sigma \to 2^{S^m}$ is the transition function, $F^m \subseteq S^m$ is a set of *accepting* states. For an infinite word $v = v_0, v_1, v_2, \ldots \in \Sigma^\omega$, the run of M on v is a sequence of states $\rho = s_0, s_1, s_2, \ldots$, such that $s_{i+1} \in \delta(s_i, v_i)$ for every i. An infinite word v is accepted by M if there is some run ρ of M on v such that some state $s \in F^m$ appears infinitely often in ρ. The language accepted by M, which we denote by $\mathcal{S}(M)$, is the set of all words v accepted by M. A set of infinite words L is said to be *ω-regular* if there is some Büchi automaton such that $L = \mathcal{S}(M)$.

CTL^* Various modal and temporal logics such as CTL^* are often used for specifying the acceptable behaviors of a system. For a comprehensive introduction to this subject, the reader is referred to [7]. In this paper we will be concerned with only one specific CTL^* property, namely $EGFp$. A state s in a Kripke structure K satisfies $EGFp$ if and only if there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting from s such that for all i, $\mathcal{L}(s_j) = p$ for some $j \geq i$; in other words, the path encounters states labelled p infinitely often. When s satisfies $EGFp$, we will say $s, K \models EGFp$; when K is clear from the context we will simply write this as $s \models EGFp$. We will denote by $[\![EGFp]\!]_K$ the set of all states s, such that $s, K \models EGFp$.

Satisfying Specifications. Similar to the traditional approach used in model checking using automata theory, we assume that the system specification is given in terms of the *bad behaviors* that the implementation must not exhibit. The bad behaviors are specified using a Büchi automaton. For a Kripke structure K and a Büchi automaton M, K is said to be *correct* with respect to M iff $\mathit{KTrace}(\mathcal{P}(K)) \cap \mathcal{S}(M) = \emptyset$. Since Büchi automata are closed under complementation even if we are given the specification as an automaton M_g specifying the *good behaviors*, we can complement M_g to get M which specifies the bad behaviors.

We will reduce the problem of checking if the system satisfies the specification to the problem of checking if the CTL^* formula $EGFp$ is satisfied. In order to do this, we first define the Kripke structure obtained by taking the cross product of a Kripke structure and a Büchi automaton.

Definition 1. *The cross-product of a Büchi automaton* $M = (S^m, \Sigma, S_0^m, \delta, F^m)$ *and a Kripke structure* $K = (S^k, \Sigma, R^k, S_0^k, \mathcal{L})$ *is the Kripke structure* $M \times K = (S^m \times S^k, \{f, \tilde{f}\}, R', S_0^m \times S_0^k, \mathcal{L}')$. *Here,* $((s_1^m, s_1^k), (s_2^m, s_2^k)) \in R'$ *if and only if*

$(s^k_1, s^k_2) \in R^k$ *and* $s^m_2 \in \delta(s^m_1, \mathcal{L}(s^k_1))$. *A state* (s^m, s^k) *in* $M \times K$ *is labelled by* f *if* $s^m \in F^m$ *and by* $\tilde{f}$ *otherwise.*

Lemma 1. *There is a path* $\pi = (s^m_0, s^k_0)(s^m_1, s^k_1)(s^m_2, s^k_2)\ldots$ *in the product Kripke structure* $M \times K$ *if and only if* $s^m_0 s^m_1 s^m_2 \ldots$ *is a run in the Büchi automaton* M *on* $KTrace(s^k_0 s^k_1 s^k_2 \ldots)$ *where* $s^k_0 s^k_1 s^k_2 \ldots$ *is a path in* K.

Proposition 1. *For an automaton* M, $KTrace(\mathcal{P}(K)) \cap \mathcal{S}(M) = \emptyset$ *if and only if* $[\![EGFf]\!]_{M\times K} \cap (S^m_0 \times S^k_0) = \emptyset$ *(In other words, no initial state of* $M \times K$ *satisfies* $EGFf$*).*

Proof. Suppose $KTrace(\mathcal{P}(K)) \cap \mathcal{S}(M) \neq \emptyset$. Then there is a path $\pi \in \mathcal{P}(K)$ such that $KTrace(\mathcal{P}(K))$ is accepted by M. Let $s^m_0 s^m_1 s^m_2 \ldots$ be the accepting run in M. By Lemma 1, there is a path $\pi = (s^m_0, s^k_0)(s^m_1, s^k_1)(s^m_2, s^k_2)\ldots$ in $M \times K$. But since an accepting run of a Büchi automata visits a accepting state infinitely often, then by the product construction, the path $\pi = (s^m_0, s^k_0)(s^m_1, s^k_1)(s^m_2, s^k_2)\ldots$ in $M \times K$ visits states labelled f infinitely often. Thus, $M \times K$ satisfies $EGFf$.

If $M \times K$ satisfies $EGFf$ then there is a path $\pi = (s^m_0, s^k_0)(s^m_1, s^k_1)(s^m_2, s^k_2)\ldots$ which infinitely often visits states labeled f. By Lemma 1, there is a run $s^m_0 s^m_1 s^m_2 \ldots$ in M on $KTrace(s^k_0 s^k_1 s^k_2 \ldots)$. This is an accepting run because the product construction labels a state $(s^m, s^k) \in M \times K$ as f only if s^m is an accepting state. But then M accepts $KTrace(s^k_0 s^k_1 s^k_2 \ldots)$. Hence, $KTrace(\mathcal{P}(K)) \cap \mathcal{S}(M) \neq \emptyset$.

3 Learning to verify ω-regular properties

In this section, we present a general framework to verify a system described as a Kripke structure K. We assume that we are given a Büchi automaton M that describes the set of behaviors that the system K *must not* exhibit. Recall, that in Section 2.2, we observed that the problem of checking if $KTrace(\mathcal{P}(K)) \cap \mathcal{S}(M) = \emptyset$ can be reduced to the problem of checking if an initial state of $M \times K$ satisfies $EGFf$. We first characterize $[\![EGFf]\!]$ using fixpoints of a functional that we define in Section 3.1. Next, we show that the fixpoint is unique and has certain key properties that we need for our problem. Finally, we will show how a learning algorithm can be used to learn the fixpoint, and therefore help verify if K satisfies M.

3.1 Fixpoint characterization of $EGFf$

From now on, we assume that we are interested in checking if some initial state of a Kripke structure $K = (S, \{f, \tilde{f}\}, R, S_0, \mathcal{L})$ satisfies $EGFf$. Traditionally, the fixpoint characterization of $EGFf$ is given by $\nu Z_1.EX(\mu Z_2.Z_1 \wedge (f \vee EXZ_2))$ (see [6]). Notice that this formula involves nesting of the fixpoint operators which we wish to avoid in our learning-based technique for technical reasons. Therefore, we develop a novel characterization of $EGFf$ that does not use nesting. Further, we also obtain a unique fixpoint which make it possible to answer equivalence queries exactly. As far as we know, this is a new characterization and may be of independent interest. We now proceed to describe this fixpoint.

Let X be a set of triples (s, i, j) such that $s \in S$ and $i, j \in \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers. We define the functional $\Gamma : 2^{S\times\mathbb{N}\times\mathbb{N}} \rightarrow 2^{S\times\mathbb{N}\times\mathbb{N}}$ such that $\Gamma(X) = \Gamma_1(X) \cup \Gamma_2(X) \cup \Gamma_3(X)$, where

$$\begin{aligned}
\Gamma_1(X) &= \{(s, 0, j) \mid \mathcal{L}(s) = f \text{ and } j \in \mathbb{N}\} \\
\Gamma_2(X) &= \{(s, i, j) \mid \mathcal{L}(s) = \tilde{f} \text{ and } \exists s'.\, s \rightarrow s'\ \exists j' < j.\, (s', i, j') \in X\} \\
\Gamma_3(X) &= \{(s, i, j) \mid \mathcal{L}(s) = f \text{ and } \exists s'.\, s \rightarrow s'\ \exists j' < j.\, (s', i-1, j') \in X\}
\end{aligned}$$

The intuition behind the definition of Γ is as follows. Consider a property $\eta_f^{i,j}$ such that a state s satisfies $\eta_f^{i,j}$ if there is a path of length j such that we encounter (at least) $i+1$ states that are labeled f. Formally, $s \models \eta_f^{i,j}$ iff there is a finite path $s_0, s_1, s_2, \ldots, s_j$ from state s such that there are indices $k_1, k_2, \ldots k_{i+1}$ such that $\mathcal{L}(s_{k_\ell}) = f$ for every $1 \leq \ell \leq i+1$. Now the intuition behind Γ is that if X is a fixpoint of Γ and $(s, i, j) \in X$ then $s \models \eta_f^{i,j}$

Clearly, Γ is monotonic and hence has fixpoints. In addition, we can show that Γ has a unique fixpoint. This is the objective of the next few observations.

Lemma 2. *Let X be a fixpoint of Γ. The following two facts hold about elements of X.*

1. *If $\mathcal{L}(s) = \tilde{f}$ then $\forall i \geq 0. \forall j.\ (s, i, j) \in X$ if and only if $\exists s'.\, s \rightarrow s'\ \exists j' < j.\, (s', i, j') \in X$*
2. *If $\mathcal{L}(s) = f$ then $\forall i \geq 1. \forall j.\ (s, i, j) \in X$ if and only if $\exists s'.\, s \rightarrow s'\ \exists j' < j.\, (s', i-1, j') \in X$*

Proof. The results follow from the definition of the fixpoint under Γ. We illustrate this for one direction of 1; the proof for other cases is similar. Suppose $\mathcal{L}(s) = \tilde{f}$ and suppose $(s, i, j) \in X$. If $\exists s'.\, s \rightarrow s'\ \exists j' < j.\, (s', i, j') \in X$ does not hold then $(s, i, j) \notin \Gamma(X)$ which contradicts the fact that X is a fixpoint.

Proposition 2. *If X_1 is a fixpoint of Γ and X_2 is also a fixpoint of Γ then $X_1 \subseteq X_2$. Hence there is a unique fixpoint of Γ.*

Proof. Let $(s, i, j) \in X_1$. We show that then $(s, i, j) \in X_2$. The proof will proceed by induction on i and j.

Consider the base case when $i = 0$. We will prove the claim by induction on j. Clearly $(s, 0, 0) \in X_1$ iff $\mathcal{L}(s) = f$ iff $(s, 0, 0) \in X_2$. Suppose the claim holds for $(s, 0, j')$ for all $j' < j$. Consider $(s, 0, j) \in X_1$. If $\mathcal{L}(s) = f$ then $(s, 0, j) \in X_2$ for every j by the definition of Γ_1. Now if $\mathcal{L}(s) = \tilde{f}$ then by Lemma 2, it must be the case that there is s' and j' such that $s \rightarrow s'$, $j' < j$ and $(s', 0, j') \in X_1$. By the induction hypothesis, we know that $(s', 0, j') \in X_2$. Again, by Lemma 2, this means that $(s, 0, j) \in X_2$.

Assume that for every $i' < i$ and for every j', if $(s, i', j') \in X_1$ then $(s, i', j') \in X_2$. The induction step for (s, i, j) is proved by induction on j. For the base case when $j = 0$, we observe that $(s, i, 0)$ is not a member of any fixpoint of Γ

(Lemma 2). The proof of the induction step is similar to the case of $i = 0$, and is skipped in the interests of space.

By symmetry, $X_2 \subseteq X_1$, hence $X_1 = X_2$ giving the uniqueness of the fixpoint for Γ.

Henceforth, we use X to denote the *unique* fixpoint of Γ. We are now ready to state the proposition that formally proves our intuition behind defining Γ.

Proposition 3. *Suppose X is the fixpoint of Γ. Then, $(s, i, j) \in X$ if and only if $s \models \eta_f^{i,j}$*

Proof. ($\Rightarrow$) We prove this by induction on i and j. For the base case consider $i = 0$. We now induct on j. When $j = 0$, $(s, 0, 0) \in X$ iff $\mathcal{L}(s) = f$, which means that there is a path of length 0 starting from s where we encounter one state labeled f. Now suppose $j > 0$. If $\mathcal{L}(s) = f$ then it trivially follows that there is a path of length $j > 0$ starting from s where we encounter at least one state labeled f. Suppose $\mathcal{L}(s) = \tilde{f}$. Then by Lemma 2, there is s' and $j' < j$ such that $s \rightarrow s'$ and $(s', 0, j') \in X$. Then by induction hypothesis, $s' \models \eta_f^{0,j'}$ which then implies that $s \models \eta_f^{0,j}$.

Consider $i > 0$. Once again we induct on j. Observe that since by Lemma 2, $(s, i, 0)$ is not in any fixpoint when $i > 0$, the claim holds vacuously. The induction step goes through in manner similar to the case of $i = 0$ and the proof is therefore skipped.

($\Leftarrow$) We prove the converse direction also by induction. Consider $i = 0$. If $j = 0$ and $s \models \eta_f^{0,0}$ then it must be the case that $\mathcal{L}(s) = f$. This means that $(s, 0, 0) \in X$. Suppose $j > 0$ and $s \models \eta_f^{0,j}$. If $\mathcal{L}(s) = f$ then once again $(s, 0, j) \in X$. If $\mathcal{L}(s) = \tilde{f}$ then it must be the case that there is some s' such that $s \rightarrow s'$ and $s' \models \eta_f^{0,j-1}$. Thus by induction hypothesis $(s', 0, j-1) \in X$ and therefore by Lemma 2, $(s, 0, j) \in X$.

Consider $i > 0$ and $s \models \eta_f^{i,j}$. If $\mathcal{L}(s) = f$ then it is definitely the case that there is s' such that $s \rightarrow s'$ and $s' \models \eta_f^{i-1,j-1}$. By induction hypothesis, $(s', i-1, j-1) \in X$, and that implies (by Lemma 2) that $(s, i, j) \in X$. On the other hand, if $\mathcal{L}(s) = \tilde{f}$ then we can conclude that there is s' such that $s \rightarrow s'$ and $s' \models \eta_f^{i,j-1}$. By induction hypothesis this means that $(s', i, j-1) \in X$, and by this we can conclude that $(s, i, j) \in X$ because of Lemma 2.

We are now ready to characterize $[\![EGFf]\!]$ in terms of the fixpoint X of Γ. This is the formal content of Proposition 4. But before presenting that proposition, we need a technical definition.

Definition 2. $\sigma(X) = \{s \mid \forall i \exists j.(s, i, j) \in X\}$

Proposition 4. *Suppose X is the fixpoint of Γ. Then $s \in \sigma(X)$ if and only if $s \models EGFf$*

Proof. ($\Leftarrow$) Suppose $s \models EGFf$. Then there is a path $\pi = s_0, s_1, s_2, \ldots$ starting from s, such that for infinitely many k, $\mathcal{L}(s_k) = f$. Define j_i to be the least

k such that $\mathcal{L}(s_k) = f$ and there are $i+1$ states before s_k on π that are also labeled f. It is clear that $s \models \eta_f^{i,j_i}$ and therefore by Proposition 3, $(s, i, j_i) \in X$. Hence $s \in \sigma(X)$.

($\Rightarrow$) Suppose $s \in \sigma(X)$. By definition, for every i, there is some j such that $(s, i, j) \in X$. Hence, by Proposition 3, $s \models \eta_f^{i,j}$. Construct a tree with root s, containing edges appearing in all shortest paths that witness s satisfying $\eta_f^{i,j}$. A few observations about this tree are in order. First, the tree is finite branching; an immediate consequence of the Kripke structure being finite branching. Second, all leaves are labeled f since the tree is constructed using the shortest witnesses. Third, if s' is an internal node in the tree then every path from s' in the tree will reach a state labeled f. Finally, this tree has infinitely many vertices. By König's Lemma, there must be an infinite path in the tree. Let us call this infinite path π. We claim that this infinite path witnesses $EGFf$. Consider any state s' on path π. Since s' is an internal node in the tree, it must be the case that on every path from s' in the tree we encounter a state labeled f. In particular on the path π, we encounter a state labeled f beyond s'. Thus π has infinitely many states labeled f.

3.2 Learning fixpoints

We are now ready to present our general framework for verifying ω-regular properties using learning. We make the following assumptions about the system K being verified.

1. The system K can be simulated from any state.
2. There is a convenient symbolic representation $\mathcal{R}$ for sets consisting of triples (s, i, j), where s is a state and i, j are natural numbers. This means that the representation is closed under complementation and decision procedures are available for membership in a set, containment of one set in another, and emptiness of a set.
3. Given the representation of a set Y of triples (s, i, j) and a state s it is possible to check if $s \in \sigma(Y)$
4. Given a representation of a set Y of triples (s, i, j) it is possible to compute the representation of $\Gamma(Y)$
5. There is an active learning algorithm for concepts encoded in the symbolic representation.

Based on these assumptions, we show how learning can be used to verify ω-regular properties. The central idea is to use the learning algorithm to learn the fixpoint X of Γ. After we learn the fixpoint, based on Propositions 1 and 4, we can reliably answer whether or not the system satisfies the specification. Thus to verify ω-regular properties using learning, we need to implement the membership and equivalence oracles that the learning algorithm needs.

Proposition 3 suggests a method to answer membership queries about whether (s, i, j) belongs to the fixpoint X of Γ. To check if (s, i, j) belongs to X, we will simulate the system for j steps starting from state s and check if on some path, we encounter $i+1$ states labeled f. Further, given a representation for a set Y, we can also answer whether Y is in fact equal to X. Since Γ has a unique fixpoint, all we need to do is check if $\Gamma(Y) = Y$. If $\Gamma(Y) \neq Y$ then the equivalence query must provide a counterexample. In other words, we need to produce an element in the symmetric difference of Y and X. This can be done as follows for the different possible cases.

- $\Gamma(Y) \setminus Y \neq \emptyset$. Let $l = (s, i, j)$ be some element in this set. If $l = (s, 0, 0)$ then $l \in X$, because the only way we can have any $(s, 0, 0)$ in $\Gamma(Y)$ is if $\mathcal{L}(s) = f$. In this case, l is in X and hence in $X \oplus Y$. If $l = (s, 0, j)$ and $\mathcal{L}(s) = f$ then once again $l \in X$ and hence in $X \oplus Y$. If $l = (s, i, j)$ for some $j \neq 0$, we can check if $l \in X$ using the membership query. If yes, then l is also in $X \oplus Y$ and we are done. Otherwise, $l \in \Gamma(Y)$ because of the existence of some triple $(s', i', j') \in Y$ which satisfies the conditions Γ_2 or Γ_3. (s', i', j') cannot be in X otherwise (s, i, j) would have to be in X. Hence $(s', i', j') \in X \oplus Y$.
- $\Gamma(Y) \subsetneq Y$. From standard fixpoint theory, since X happens to also be the least fixpoint under Γ, it must be the intersection of all prefixpoints of Γ (a set Z is a prefixpoint if it *shrinks* under the functional Γ, *i.e.* $\Gamma(Z) \subseteq Z$). Now, Y is clearly a prefixpoint. Applying Γ to both sides of the equation $\Gamma(Y) \subsetneq Y$ and using monotonicity of Γ, we get $\Gamma(\Gamma(Y)) \subsetneq \Gamma(Y)$. Thus, $\Gamma(Y)$ is also a prefixpoint. Let l be some string in the set $Y \setminus \Gamma(Y)$. Since l is outside the intersection of two prefixpoints, it is not in the least fixpoint X. Hence, l is in $X \oplus Y$.

Once we have learned the fixpoint X, we can verify if the initial states of the Kripke structure satisfy $EGFf$ using Proposition 4. By Proposition 1, this provides an answer to the verification problem. The overall procedure is summarized in Figure 1. This procedure yields a complete verification method when the fixpoint X of Γ can be symbolically represented in the chosen representation. This is the content of the following theorem.

Theorem 1. *If the fixpoint X of Γ can be represented using the chosen symbolic data structure and a learning algorithm using membership and equivalence queries is available for this data structure, the verification procedure is guaranteed to terminate and correctly infer whether the system satisfies the specification.*

The theorem follows from observations made in this section.

4 Infinite state systems using regular languages

In Section 3.2 we presented a general set of conditions under which we can use a learning based approach to verify systems with respect to ω-regular properties. In this section, we demonstrate this can be achieved within the context of using regular languages to represent sets of states.

Regular sets are a popular symbolic representation for sets of states of for infinite state systems. *Regular model checking* [4] has been applied to modeling

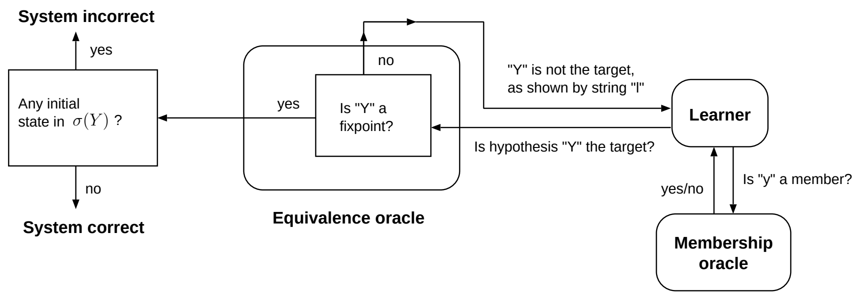

Fig. 1. Verification procedure

parameterized systems, FIFO automata, systems with integer variables and *push down stacks*. Based on the practical success that has been enjoyed by *regular model checking* and the efficient learning algorithms available for regular languages, we apply our learning technique on regular sets. As mentioned before, we use Angluin's L^* [2] algorithm.

We assume that the states of the system can be encoded as strings over some finite alphabet ρ^k. The transition relation is given as a transducer τ^k which takes an input string corresponding to some state s and outputs a string for the state related to s. The transition relation is assumed to be total. The set of initial states is given by a regular set S_0^k and the set of states with a label a is given as regular sets S_a^k. Let K be the Kripke structure defined by the above sets.

4.1 Construction of the product Kripke structure

Let M be the Büchi automaton specifying the bad behaviors that must not be exhibited by the system. Since ω-regular languages are powerful enough to express fairness constraints, we assume that such constraints, if any, are already embodied in the Büchi automaton. We now show how to construct the product Kripke structure $M \times K$. We extend the alphabet ρ^k to $\rho^{M \times K}$ with new symbols b_{s^m}, one for each state s^m in M. A state (s^m, s^k) in $M \times K$ is encoded as a string with the first letter as b_{s^m} and the remaining part of the string as the original string encoding s^k. Initial states in $S_0^{M \times K}$ are given by concatenating a letter b_{s_0} for $s_0 \in S_0^m$ and a string in S_0^k. The set of states $S_{\tilde{f}}$ (resp. S_f) labelled with $\tilde{f}$ (resp. f) is given by a DFA which looks at the first letter of the input string and accepts if this is b_{s^m} for some $s^m \notin F^m$ (resp. $s^m \in F^m$). The transducer $\tau^{M \times K}$ representing the transition relation for $M \times K$ is a bit more tedious but can be constructed using standard automata operations.

Henceforth, we restrict our attention to the Kripke structure $M \times K$. For ease of notation, we drop the superscript $M \times K$ in τ, S_0, ρ and so on.

4.2 Symbolic representation for the fixpoint X

As discussed in Section 3.2, we now need to learn the fixpoint X of the functional Γ. In general, X is a subset of $\rho^* \times \mathbb{N} \times \mathbb{N}$. To encode X as a regular set we use the alphabet ρ^X given by $(\rho \cup \{\perp\}) \times \{0, \perp\} \times \{0, \perp\}$. This is the alphabet that will be used by Angluin's L^* learning algorithm. Here 0 is a unary symbol for natural numbers and $\perp$ is a new "filler" symbol. An element (s, i, j) is encoded as string over ρ^X such that projecting the symbols on the first component gives us s (the $\perp$ symbols are ignored); and projecting on the second and third components gives i and j respectively in unary notation.

4.3 Membership and Equivalence queries

As discussed before, membership queries for X can be answered using Proposition 3. For answering equivalence queries, we need a symbolic way to calculate $\Gamma(X)$. Apart from the standard operations on regular set we define the following.

Definition 3. *Given Y a set of strings in the alphabet of ρ^X, define*

$$Inc^i(Y) = \{(s, i, j) \mid (s, i-1, j) \in Y\}$$
$$Inc^j(Y) = \{(s, i, j) \mid (s, i, j-1) \in Y\}$$

Given a DFA M_Y for Y, the DFA for $Inc^i(Y)$ can be constructed as follows. $Inc^i(Y)$ keeps two copies of M_Y, with initial states in the first copy and final states in the second copy. Any transition t with the $\perp$ symbol for the i component in the first copy is changed to a transition to the state corresponding to the target of t in the second copy and the i component symbol is changed to 0. We also add a transition from a state in the first copy which used to be final in M_Y to the corresponding state in the second copy with symbol $(\perp, 0, \perp)$. A similar construction can be used for $Inc^j(Y)$.

Checking hypothesis for upward closure in j. A property that we will find useful in answering equivalence queries is that by definition of Γ, its fixpoint X is upward closed in the j component, *i.e.*, if (s, i, j) in X then for all $j' > j$, (s, i, j') is also in X. A set Y is upward closed in the j component if and only if $Inc^j(Y) \subseteq Y$. If Y is not upward closed then let (s, i, j) be the string in $Inc^j(Y) \setminus Y$. Clearly, $(s, i, j) \notin Y$. Now we use membership query to check if $(s, i, j) \in X$. If (s, i, j) is indeed in X then (s, i, j) is in the symmetric difference $X \oplus Y$. Otherwise $(s, i, j-1)$ is also not in X (since X has the upward closed property). In this case $(s, i, j-1) \in X \oplus Y$.

Symbolic computation of Γ_1. A finite automaton for $\Gamma_1(Y)$ is obtained by taking the DFA for f and taking its cross product with a DFA that accepts 0 for the i component and another DFA which accepts any j.

Symbolic computation of Γ_2. If we always first check for upward closure in j, we can assume that we would need to compute Γ_2 only for sets which are upward closed. Let $\tau^{-1}(Y)$ be the inverse of τ lifted to the triples (s, i, j) so that it simply copies the second and the third components. It can be seen that if Y is upward closed then $\Gamma_2(Y) = S_{\bar{f}} \cap Inc^j(\tau^{-1}(Y))$.

Symbolic computation of Γ_3. For Γ_3, $\tau^{-1}(Y)$ gives the set of states which have a successor in Y. It is easy to see that $\Gamma_3(Y) = S_f \cap Inc^i(Inc^j(\tau^{-1}(Y)))$.

Using the fixpoint check. From the previous paragraphs, we have a symbolic method to compute $\Gamma(Y) = \Gamma_1(Y) \cup \Gamma_2(Y) \cup \Gamma_3(Y)$. Now, the equivalence oracle simply needs to check if $Y = \Gamma(Y)$. We also need a method of extracting strings in the symmetric difference of Y and the fixpoint in case Y is not the fixpoint. It can be seen that the approach outlined in Section 3.2 can be applied to regular sets.

4.4 Checking for $s_0 \in \sigma(X)$.

Proposition 5. $\sigma(X) = \overline{Proj_1(\overline{Proj_{1,2}(X)})}$. *Here, $Proj_1$ is the projection to the first component and $Proj_{1,2}$ the projection to the first and second components.*

Proof (Sketch). Recall that $\sigma(X) = \{s \mid \forall i \exists j.(s, i, j) \in X\}$. Equivalently, $\sigma(X) = \{s \mid \neg(\exists i \neg(\exists j.(s, i, j) \in X))\}$. The claim follows from the fact that $\exists$ can be eliminated using projection and the $\neg$ operator corresponds to taking the complement.

Given a regular representation of X we can calculate $\sigma(X)$ using standard regular set operations. Then the system is correct if and only if $S_0 \cap \sigma(X) = \emptyset$.

The verification algorithm is summarized in Figure 2.

4.5 Complexity analysis

Let m be the length of the longest string returned by the teacher in a negative answer to an equivalence query, n be the number of states of the minimal automaton representing the fixpoint X, k be the size of the alphabet of the learned language and t be the number of states of the automaton representing the transducer for the transition relation. As shown in [2], Angluin's algorithm makes $O(kmn^2)$ membership queries and $O(n)$ equivalence queries. The worst case for the equivalence query for a hypothesis Y occurs when we look for a string in the difference of Y and $\Gamma(Y)$. The size of DFA representing Y is bounded by n. Looking at Γ, it can be seen that the DFA representing the difference of Y and $\Gamma(Y)$ would be $O(nt)$. Thus the length of the longest string returned by an equivalence query is $m = O(nt)$.

The cost of answering membership queries dominates the total runtime cost of the algorithm. Using $m = O(nt)$, the number of membership queries is $O(ktn^3)$. For efficiency, given a query for (s, i, j), we build a DFA D_j for $\Gamma^{j+1}(\emptyset)$ where

algorithm *learner*
begin
Angluin's L^* algorithm
end

algorithm *isMember*
Input: (s, i, j)
Output: is $(s, i, j) \in X$?
```
begin
  From s simulate system for j steps
    Does any path in above encounter
    at least i + 1 states labelled f?
  If yes return true
  else return false
end
```

algorithm *Equivalence Check*
Input: Hypothesis Y
Output: For fixpoint X, is $Y = X$?
If not, then some string in $Y \oplus X$
```
begin
  If Inc^j(Y) \ Y ≠ ∅ {upward closure check}
    let (s, i, j) ∈ Inc^j(Y) \ Y
    if isMember((s, i, j))
      return (no, (s, i, j))
    else
      return (no, (s, i, j − 1))
  else if Γ(Y) \ Y ≠ ∅ {fixpoint check}
    let (s, i, j) ∈ Γ(Y) \ Y
    Find (s', i', j') which causes (s, i, j) to be in Γ(Y)
    if isMember((s, i, j))
      return (no, (s, i, j))
    else
      return (no, (s', i', j'))
  else if Γ(Y) ⊊ Y
    return (no, l ∈ (Y \ Γ(Y)))
  else {found fixpoint}
    if S_0 ∩ Proj_1(complement(Proj_{1,2}(X))) ≠ ∅
      print "System incorrect"
    else
      print "System correct"
    end
```

Fig. 2. Verifying ω-regular properties for regular set based systems

Γ^{j+1} denotes the composition of Γ $j+1$ times with itself. Once D_j has been built, all queries with the same value of j can be answered by checking if the queried element is accepted by D_j. Thus the cost of the membership queries is equal to the number of membership queries and the cost of building the DFAs. The cost for D_j is $(O(t))^j$ which leads to the total cost of membership queries of $O(t^{O(nt)} + ktn^3)$ (using maximum value of j to be $m = O(nt)$).

5 Examples

We have extended our learning based verification tool suite called LEVER [11] with the algorithm presented in this paper and have successfully analyzed liveness properties for two examples of infinite state systems. The Büchi automaton forms the specification and also describes the fairness constraints on the system. The states of the system considered are encoded as strings over an alphabet as described in [4]. We now briefly discuss the examples analyzed.

Token passing We consider a parameterized system of processes in which each process can send a token to the process to its right. There is a single token in the system and initially it rests with the leftmost process. The liveness property that is encoded with the Büchi automaton is, "every process eventually receives a token". The fixpoint for Γ is found to be regular and the system shown to be correct using our verification procedure.

Producer consumer This consists of a FIFO automata with a single channel, in which one part of the system constantly produces messages while another part consumes them. We verify the property, "a message produced by the producer is eventually consumed". Again, the fixpoint for Γ is found to be regular and the system verified to be correct.

Both the examples take just a few seconds to analyze on a 1.5 GHz computer. We continue to optimize the implementation in LEVER, and in future plan on analyzing more examples of infinite state systems and comparing our running time with other tools that are available.

6 Conclusion

In this paper we presented a general learning based verification framework to verify ω-regular properties of infinite state systems. We instantiated the framework in the context of regular model checking giving detailed algorithms for the various primitive operations that are needed in order to perform the learning based verification procedure. The algorithm is a significant improvement in the current state of the art in learning based verification, as it verifies general ω-regular properties, while not making restrictive assumptions about the way the transition relation of the system is represented as a transducer. Furthermore, the algorithm can detect buggy implementations, even when the implementations do not have an ultimately periodic counter-example for the property.

References

1. P. A. Abdulla, B. Jonsson, M. Nilson, J. d'Orso, and M. Saksena. Regular model checking for LTL(MSO). In *Proc. of CAV'04, USA, LNCS 3114*, 2004.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, Nov. 1987.
3. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV'04, LNCS 3114*, 2004.
4. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 403–418. Springer, 2000.
5. A. Bouajjani, A. Legay, and P. Wolper. Handling liveness properties in (ω-)regular model-checking. In *Proc. of Infinity'04, London, UK*, 2004.
6. E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mucalculus. In *Proccedings of the First Annual Symposium on Logic in Computer Science*, pages 267–278, Washington, D.C., 1986. IEEE Computer Society Press.
7. E. A. Emerson. Temporal and modal logic. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier, Amsterdam, 1990.
8. L. Fribourg and H. Olsén. Reachability sets of parametrized rings as regular languages. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97), Bologna, Italy, July 1997*, volume 9. Elsevier Science, 1997.
9. P. Habermehl and T. Vojnar. Regular model checking using inference of regular languages. In *Proc. of Infinity'04, London, UK*, 2004.
10. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 220–234. Springer, 2000.
11. LEVER. Learning to verify tool. http://osl.cs.uiuc.edu/~vardhan/lever.html, 2004.
12. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, Singapore, 1992.
13. A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *CAV'00*, 2000.
14. W. Thomas. Automata on infinite objects. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier, Amsterdam, 1990.
15. T. Touili. Regular model checking using widening techniques. In *ENTCS*, volume 50. Elsevier, 2001.
16. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for fifo automata. In *LNCS 3328, Proc. of FSTTCS'04, Chennai, India*, pages 494–505, 2004.
17. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. In *LNCS 3308, Proc. of ICFEM'04, Seattle, USA*, pages 274–288, 2004.