Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages*

Azadeh Farzan[1] , Yu-Fang Chen[2],
Edmund M. Clarke[1], Yih-Kuen Tsay[2], and Bow-Yaw Wang[3]

[1]Carnegie Mellon University [2]National Taiwan University [3]Academia Sinica

Abstract. Recent studies have suggested the applicability of learning to automated compositional verification. However, current learning algorithms fall short when it comes to learning *liveness* properties. We extend the automaton synthesis paradigm for the infinitary languages by presenting an algorithm to learn an *arbitrary* regular set of infinite sequences (an ω-regular language) over an alphabet Σ. Our main result is an algorithm to learn a nondeterministic Büchi automaton that recognizes an unknown ω-regular language. This is done by learning a unique projection of it on Σ^* using the framework suggested by Angluin for learning regular subsets of Σ^*.

1 Introduction

Compositional verification is an essential technique for addressing the state explosion problem in Model Checking [1, 7, 8, 10]. Most compositional techniques advocate proving properties of a system by checking properties of its components in an assume-guarantee style. The essential idea is to model check each component independently by making an assumption about its environment, and then discharge the assumption on the collection of the rest of the components. In the paradigm of automated compositional reasoning through learning [8], system behaviors and their requirements are formalized as regular languages. Assumptions in premises of compositional proof rules are often regular languages; their corresponding finite-state automata can therefore be generated by learning techniques for regular languages.

In automated compositional reasoning, a compositional proof rule is chosen *a priori*. The rule indicates how a system can be decomposed. Below is an example of a simple rule:

$$\frac{M_2 \models A \quad M_1||A \models P}{M_1||M_2 \models P}$$

for two components M_1 and M_2, and assumption A, and a property P. Intuitively, this rule says that if M_2 guarantees A, and M_1 guarantees P in an environment that respects A, then the system composed of M_1 and M_2 guarantees P. The goal is to automatically generate the assumption A by learning. One

* This research was sponsored by the iCAST project of the National Science Council, Taiwan, under the grant no. NSC96-3114-P-001-002-Y and the Semiconductor Research Corporation (SRC) under the grant no. 2006-TJ-1366.

naturally wishes to verify all sorts of properties using this framework. However, all existing algorithms fall short when it comes to learning assumptions which involve *liveness* properties. In this paper, we present an algorithm that fills this gap and extends the learning paradigm to the full class of ω-regular languages.

The active learning model used in automated compositional reasoning involves a teacher who is aware of an unknown language, and a learner whose goal is to learn that language. The learner can put two types of queries to the teacher. A *membership query* asks if a string belongs to the unknown language. A *equivalence query* checks whether a conjecture automaton recognizes the unknown language. The teacher provides a counterexample if the conjecture is incorrect [2]. More specifically, in the process of learning an *assumption*, an initial assumption is generated by the learner through a series of membership queries. An equivalence query is then made to check if the assumption satisfies premises of the compositional proof rule. If it does, the verification process terminates with success. Otherwise, the learner refines the assumption by the returned counterexample and more membership queries. Since the weakest assumption either establishes or falsifies system requirements, the verification process eventually terminates when the weakest assumption is attained. A novel idea in [8] uses model checkers to resolve both membership and equivalence queries automatically. By using Angluin's L* [2] algorithm, the verification process can be performed without human intervention.

The product of the learning algorithm L^* is a deterministic finite-state automaton recognizing the unknown regular language [2]. By the Myhill-Nerode Theorem, the minimal deterministic finite-state automaton can be generated from the equivalence classes defined by the coarsest right congruence relation of any regular language [12]. The L^* algorithm computes the equivalence classes by membership queries, and refines them with counterexamples returned by equivalence queries. It can, in fact, infer the minimal deterministic finite-state automaton for any unknown regular language.

Unfortunately, the L* algorithm cannot be directly generalized to learn ω-regular languages. Firstly, deterministic Büchi automata are less expressive than general Büchi automata. Inferred deterministic finite-state automata require more than the Büchi acceptance condition to recognize arbitrary ω-regular languages. Secondly, equivalence classes defined by the coarsest right congruence relation over an ω-regular language do not necessarily correspond to the states of its automaton. The ω-regular language $(a + b)^* a^\omega$ has only one equivalence class. Yet, there is no one-state ω-automaton with Büchi, Rabin, Streett, or even Muller acceptance conditions that can recognize this language.

Maler and Pnueli [13] made an attempt to generalize L* for the ω-regular languages. Their algorithm, L^ω, learns a proper subclass of ω-regular languages which is not expressive enough to cover liveness properties. This restricted class has the useful property of being uniquely identifiable by the syntactic right congruence. Thus, L^ω has the advantage of generating the minimal deterministic Muller automaton (isomorphic to the syntactic right congruence) recognizing a language in the restricted class. The syntactic right congruence, however, can-

not be used to identify an arbitrary ω-regular language. Attempts to use more expressive congruences [3, 20] have been unsuccessful.

Our main ideas are inspired by the work of Calbrix, Nivat, and Podelski [5]. Consider ultimately periodic ω-strings of the form uv^ω. Büchi [4] observed that the set of ultimately periodic ω-strings characterizes ω-regular languages; two ω-regular languages are in fact identical if and only if they have the same set of ultimately periodic ω-strings. Calbrix *et al.* [5] show that the finitary language $\{u\$v \mid uv^\omega \in L\}$ is regular for any ω-regular language L. These properties help uniquely identify a Büchi automaton for the regular language corresponding to ultimately periodic ω-strings of an arbitrary ω-regular language. We develop a learning algorithm for the regular language $\{u\$v \mid uv^\omega \in L\}$ through membership and equivalence queries on the unknown ω-regular language L. A Büchi automaton accepting L can hence be constructed from the finite-state automaton generated by our learning algorithm.

The active learning model and the algorithm L^* were introduced by Angluin [2]. By exploiting the Myhill-Nerode theorem, the L^* algorithm is able to compute the minimal deterministic finite-state automaton for a given regular language with a polynomial number of queries in the size of the target automaton. The upper bound was later improved in [17].

2 Preliminaries

Let Σ be a finite set called the *alphabet*. A finite word over Σ is a finite sequence of elements of Σ. An empty word is represented by ϵ. For two words $u = u_1 \dots u_n$ and $v = v_1 \dots v_n$, define $uv = u_1 \dots u_n v_1 \dots v_m$. For a word u, u^n is recursively defined as uu^{n-1} with $u^0 = \epsilon$. Define $u^+ = \bigcup_{i=1}^{\infty} \{u^i\}$, and $u^* = \{\epsilon\} \cup u^+$. An infinite word over Σ is an infinite sequence of elements of Σ. For a finite word u, define the infinite word $u^\omega = uu \dots u \dots$. Operators $+$, $*$, and ω are naturally extended to sets of finite words.

A word u is a *prefix* (resp. *suffix*) of another word v if and only if there exists a word $w \in \Sigma^*$ such that $v = uw$ (resp. $v = wu$). A set of words S is called *prefix-closed* (resp. *suffix-closed*) if and only if for all $v \in S$, if u is a prefix (resp. suffix) of v then $u \in S$.

The set of all *finite* words on Σ is denoted by Σ^*. Σ^+ is the set of all nonempty words on Σ; therefore, $\Sigma^+ = \Sigma^* \backslash \{\epsilon\}$. Let u be a finite word. $|u|$ is the length of word u with $|\epsilon| = 0$. The set of all infinite words on Σ is denoted by Σ^ω. A language is a subset of Σ^*, and an ω-language is a subset of Σ^ω.

A finite automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, I, F, \delta)$ where Σ is an alphabet, Q is a finite set of states, $I \subseteq Q$ is a set of *initial* states, $F \subseteq Q$ is a set of *final* states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. A finite word $u = u_1 \dots u_n$ is accepted by $\mathcal{A}$ if and only if there exists a sequence $q_{i_0} u_1 q_{i_1} u_2 \dots u_n q_{i_n}$ such that $q_{i_0} \in I$, $q_{i_n} \in F$, and for all j, we have $q_{i_j} \in Q$ and $(q_{i_{j-1}}, u_j, q_{i_j}) \in \delta$. Define $L(\mathcal{A}) = \{u \mid u \text{ is accepted by } \mathcal{A}\}$. A language $L \subseteq \Sigma^*$ is *regular* if and only if there exists an automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

A Büchi automaton has the same structure as a finite automaton, except that it is intended for recognizing infinite words. An infinite word $u = u_1 \dots u_n \dots$ is accepted by a Büchi automaton $\mathcal{A}$ if and only if there exists a sequence $q_{i_0} u_1 q_{i_1} u_2 \dots u_n q_{i_n} \dots$ such that $q_{i_0} \in I$, $q_{i_j} \in Q$ and $(q_{i_j-1}, u_j, q_{i_j}) \in \delta$ (for all j), and there exists a state $q \in F$ such that $q = q_{i_j}$ for infinitely many j's. Again, define $L(\mathcal{A}) = \{u \mid u \text{ is accepted by } \mathcal{A}\}$. An ω-language $L \subseteq \Sigma^\omega$ is ω-regular if and only if there exists a Büchi automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$. For an ω-language L, let $UP(L) = \{uv^\omega \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$. Words of the form uv^ω are called the *ultimately periodic*. Let α be an ultimately periodic word. A word $v \in \Sigma^+$ is a *period* of α if there exists a word $u \in \Sigma^*$ such that $\alpha = uv^\omega$.

Theorem 1. *(Büchi)[4] Let L and L' be two ω-regular languages. $L = L'$ if and only if $UP(L) = UP(L')$.*

The above theorem implies that the set of ultimately periodic words of an ω-regular language L uniquely characterizes L. Define $L_\$$ on $\Sigma \cup \{\$\}$ as

$$L_\$ = \{u\$v \mid uv^\omega \in L\}.$$

Intuitively, the symbol \$ marks the beginning of the period and separates it from the prefix of the ω-word uv^ω. Note that $L_\$ \subseteq \Sigma^*\$\Sigma^+$ We can then say that $L_\$$ uniquely characterizes L.

Theorem 2. *(Büchi)[4] If L is an ω-regular language, then there exists regular languages $L_1, \dots, L_n$ and $L'_1, \dots, L'_n$ such that $L = \bigcup_{i=1}^n L_i (L'_i)^\omega$.*

Theorem 3. *(Calbrix, Nivat, and Podelski)[5] $L_\$$ is regular.*

Moreover, one can show that the *syntactic congruence* of the regular language $L_\$$ and Arnold's congruence [3] for L coincide on the set Σ^+ [6].

3 Ultimately Periodic Words

Define an *equivalence* relation on the words in $\Sigma^*\$\Sigma^+$:

Definition 1. *The* equivalence *relation $\triangleq$ on $\Sigma^*\$\Sigma^+$ is defined by:*

$$u\$v \triangleq u'\$v' \iff uv^\omega = u'v'^\omega$$

$u, u' \in \Sigma^*$ and $v, v' \in \Sigma^+$.

Based on the ω-word ab^ω, we have $a\$b \triangleq ab\$b \triangleq ab\$bb \triangleq \dots \triangleq ab^k\$b^{k'}$, for all k, k'. Therefore, the equivalence class $[a\$b]_\triangleq$ is equal to the set of words $ab^*\$b^+$.

Definition 2. *An equivalence relation $\equiv$* saturates *a language L if and only if for two words u and v, where $u \equiv v$, we have $u \in L$ implies $v \in L$.*

Let L be an ω-regular language, and $L_\$$ its corresponding regular language as defined above. Let $u\$v$ be a word in $L_\$$ and $u'\$v' \in \Sigma^*\$\Sigma^+$ such that $u\$v \doteq u'\$v'$. Since $uv^\omega = u'v'^\omega$, we have $u'v'^\omega \in L$, and therefore (by definition) $u'\$v' \in L_\$$. This implies the following Proposition:

Proposition 1. *The equivalence relation $\doteq$ saturates $L_\$$.*

Let $R \subseteq \Sigma^*\$\Sigma^+$ be a regular language. Proposition 1 suggests that saturating $\doteq$ is a necessary condition for R to be $L_\$$ for some ω-regular language L. The interesting point is that one can show that it is sufficient as well. This can be done by constructing a Büchi automaton $\mathcal{B}$ that recognizes L directly from the automaton $\mathcal{A}$ recognizing R [5]. Since this construction is used in our algorithm, we describe it here. We first need the following lemma:

Lemma 1. *(Calbrix, Nivat, and Podelski) [5] Let $L, L' \subseteq \Sigma^*$ be two regular languages such that $LL'^* = L$ and $L'^+ = L'$. Then, $\alpha \in UP(LL'^\omega)$ if and only if there exist $u \in L$ and $v \in L'$ such that $\alpha = uv^\omega$.*

Let $R \subseteq \Sigma^*\$\Sigma^+$ be a regular language. Let $\mathcal{A} = (\Sigma \cup \{\$\}, Q, I, F, \delta)$ be a deterministic automaton recognizing R. Define $Q_\$$ to be the set of states that can be reached by starting in an initial state and reading the part of a word $u\$v \in M$ that precedes the \$. Formally,

$$Q_\$ = \{q \in Q \mid \exists u\$v \in R, \exists q_i \in I, q = \delta(q_i, u)\}$$

For each state $q \in Q_\$$, let

$$M_q = \{u \mid \exists q_i \in I, \delta(q_i, u) = q\} \tag{1}$$

$$N_q = \{v \mid \exists q_f \in F, \delta(q, \$v) = q_f\}. \tag{2}$$

For each q, M_q and N_q are regular languages; one can easily construct an automaton accepting each by modifying $\mathcal{A}$. Moreover, the definitions of M_q and N_q along side the fact $R \subseteq \Sigma^*\$\Sigma^+$, implies that $R = \bigcup_{q \in Q_\$} M_q\$N_q$.

Next, we partition N_q based on the final states of the automaton. For each final state $q_f \in F$ and $q \in Q_\$$, let the regular language N_{q,q_f} be

$$N_{q,q_f} = \{v \mid \delta(q, v) = q \ \wedge \ \delta(q, \$v) = q_f \ \wedge \ \delta(q_f, v) = q_f\} \tag{3}$$

Finally, we define the ω-regular language L as

$$L = \bigcup_{(q,q_f) \in Q_\$ \times F} M_q N_{q,q_f}^\omega. \tag{4}$$

We refer to L as $\omega(R)$ later in this paper to indicate the fact that it is the corresponding ω-regular language of R. Next we show that L is the ω-regular language whose corresponding regular language is indeed R. The following theorem states this result:

Theorem 4. *Let $R \subseteq \Sigma^*\$\Sigma^+$ be a regular language that is saturated by $\doteq$. Then, there exists an ω-regular language L such that $R = L_\$$.*

Proof. See Appendix A.1 for the proof. □

One can directly build a Büchi automaton recognizing L from $\mathcal{A}$. The set $Q_\$$ can be effectively computed. For each state $q \in Q_\$$, the language M_q is recognized by the automaton $(\Sigma, Q, I, \{q\}, \delta)$. For each final state q_f, the language N_{q,q_f} is the intersection of the languages $L(\Sigma, Q, \{q\}, \{q\}, \delta)$, $L(\Sigma, Q, \{\delta(q, \$)\}, \{q_f\}, \delta)$, and $L(\Sigma, Q, \{q_f\}, \{q_f\}, \delta)$. For each pair (q, q_f), once we have DFAs recognizing M_q and N_{q,q_f}, we can easily construct[1] a Büchi automaton recognizing $M_q N_{q,q_f}^\omega$. The Büchi automaton recognizing L is the union of these automata. Each $M_q N_{q,q_f}^\omega$ is recognized by an automaton of size at most $|\mathcal{A}| + |\mathcal{A}|^3$, which means that L is recognized by an automata of size at most $|\mathcal{A}|^3 + |\mathcal{A}|^5$.

A question that naturally arises is what can one say about the result of the above construction if R is not saturated by $\doteq$? As we will see in Section 4, we need to construct Büchi automata from DFAs guessed in the process of learning which may not be necessarily saturated by $\doteq$. For a regular language $R \subseteq \Sigma^*\$\Sigma^+$ which is not saturated by $\doteq$ and $L = \bigcup_{(q,q_f)\in Q_\$ \times F} M_q N_{q,q_f}^\omega$, it is not necessarily the case that $R = L_\$$ (compare with the statement of Theorem 4). For example, $R = \{a\$b\}$ is not saturated by $\doteq$ since it contains an element of the class $[a\$b]_{\doteq}$ (namely, $a\$b$), but does not contain the whole class (which is the set $ab^*\$b^+$). But, there are a number of other essential properties that L has:

Proposition 2. *If $R = U_\$$ for some arbitrary ω-regular language U, then $L = U$, where L is defined by (4).*

Proof. Direct consequence of Theorem 4. □

Proposition 3. *Let $[u\$v]_{\doteq}$ denote the equivalence class of the word $u\$v$ by the relation $\doteq$. For each pair of words $(u, v) \in \Sigma^* \times \Sigma^+$, if $[u\$v]_{\doteq} \cap R = \emptyset$ then $uv^\omega \notin L$ where L is defined by (4).*

Proof. If $uv^\omega \in L$, there exist a string u' in some M_q and a string v' in some N_{q,q_f} such that $u'v'^\omega = uv^\omega$ (Lemma 1). Since u' is in M_q and v' is in N_{q,q_f}, we have $u'\$v'$ in R. Because $u'v'^\omega = uv^\omega$, we have $u'\$v' \in [u\$v]_{\doteq}$, which contradicts $[u\$v]_{\doteq} \cap R = \emptyset$. □

Proposition 4. *For each pair of words $(u, v) \in \Sigma^* \times \Sigma^+$, if $[u\$v]_{\doteq} \subseteq R$ then $uv^\omega \in L$ where L is defined by (4).*

Proof. If $[u\$v]_{\doteq} \subseteq R$, we can find k and k' satisfying $uv^k(v^{k'})^\omega \in L$ using the same approach as in the $(\mathbf{R} \subseteq \mathbf{L}_\$)$ proof of Theorem 4. Because $uv^\omega = uv^k(v^{k'})^\omega$, we have $uv^\omega \in L$. □

[1] One can connect the final states of $\mathcal{A}(M_q)$ to the initial states of $\mathcal{A}^\omega(N_{q,q_F})$ by ϵ transitions, and let the final states of N_{q,q_f} be the final states of the resulting Büchi automaton. $\mathcal{A}^\omega(N_{q,q_F})$ can be obtained from $\mathcal{A}(N_{q,q_F})$ by normalizing it and connecting the final state to the initial state by an epsilon transition [16].

4 Learning ω-Regular Languages

In this section, we present an algorithm that *learns* an unknown ω-regular language and generates a nondeterministic Büchi automaton which recognizes L as the result. There are well-known and well-studied algorithms for learning a deterministic finite automaton (DFA) [2, 17]. We propose an approach which uses the L* algorithm [2] as the basis for learning an unknown ω-regular language L.

The idea behind L* is learning by experimentation. The learner has the ability to make *membership queries*. An *oracle* (a *teacher* who knows the target language), on any input word v, returns a yes-or-no answer depending on whether v belongs to the target language. The learning algorithm thus chooses particular inputs to classify, and consequently make progress. The learner also has the ability to make *equivalence* queries. A target language is guessed by the learner, which will then be verified by the *teacher* through an equivalence check against the target language. The teacher returns *yes* when the conjecture is correct, or *no* accompanied by a counterexample which witnesses the inequality. This counterexample can be a *positive* counterexample (a word that belongs to the target language but does not belong to the conjecture language) or a *negative* counterexample (a word that does not belong to the the target language but belongs to the conjecture language). We refer the reader unfamiliar with L* to [2] for detailed information on the algorithm.

The goal of our learning algorithm is to come up with a nondeterministic Büchi automaton that recognizes an unknown ω-regular language L. We assume that there is a teacher who can correctly answer the *membership* and *equivalence* queries on L as discussed above. The idea is to learn the language $L_\$$ instead of learning L directly. One can reuse the core of the L* algorithm here, but many changes have to be made. The reason is that the membership and equivalence queries allowed in the setting of our algorithm are for the ω-regular language L and not for the regular language $L_\$$. One has to translate the queries and their responses back and forth from the $L_\$$ level to the L level.

Membership Queries: The L* algorithm frequently needs to ask questions of the form: "does the string w belong to the target language $L_\$$?". We need to translate this query into one that can be posed to our teacher. The following simple steps perform this task:

1. Does w belong to $\Sigma^*\$\Sigma^+$? If no, then the answer is "NO". If yes, then go to the next step.
2. Let $w = u\$v$. Does uv^ω belong to L? if no, then the answer is "NO". If yes, then the answer is "YES".

We know that $L_\$ \subseteq \Sigma^*\$\Sigma^+$ which helps us filter out some strings without asking the teacher. If we have $w \in \Sigma^*\$\Sigma^+$, then w is of the form $u\$v$ which corresponds to the ultimately periodic word uv^ω. The teacher can respond to the membership query by checking whether uv^ω belongs to L. The answer to this query indicates whether $u\$v$ should belong to our current conjecture. Note that by the definition of $L_\$$, we have $u\$v \in L_\$ \Leftrightarrow uv^\omega \in L$.

Equivalence Queries: L* generates conjecture DFAs that need to be verified, and therefore a question of the form "Is the conjecture language M_i equivalent to the target language $L_\$$?" needs to be asked. We need to translate this query into an equivalent one that can be posed to the teacher:

1. Is M_i a subset of $\Sigma^*\$\Sigma^+$? If no, get the counterexample and continue with L*. If yes, then go the next step.
2. Is $\omega(M_i)$ (the corresponding ω-regular language of M_i) equivalent to L? If yes, we are done. If no, we get an ultimately periodic word c that is a (negative or positive) counterexample to the equivalence check. Return "NO" and a finitary interpretation of c (described below) to L*.

Again, the $M_i \subseteq \Sigma^*\$\Sigma^+$ check works as a preliminary test to filter out conjectures that are *obviously* not correct. If a conjecture language (DFA) M_i passes the first test, we construct its corresponding Büchi automaton $\omega(M_i)$. The teacher can then respond by checking the equivalence between L and $\omega(M_i)$. If they are not equivalent, the teacher will return a counterexample witnessing the nonequivalence. In order to proceed with L*, we have to translate these ω-words to finite words that are counterexamples to the equivalence of M_i and $L_\$$. To do this, for the counterexample uv^ω, we construct a DFA that accepts $[u\$v]_{\hat{=}}$. There are two cases for each counterexample uv^ω:

- *The word uv^ω is a positive counterexample:* the word uv^ω should be in $\omega(M_i)$ but is not. Since $uv^\omega \notin \omega(M_i)$, by Proposition 4, $[u\$v]_{\hat{=}} \not\subseteq M_i$ and there exists a word $u'\$v' \in [u\$v]_{\hat{=}}$ such that $u'\$v'$ is not in M_i. Then $u'\$v'$ can serve as an effective positive counterexample for the L* algorithm. To find $u'\$v'$, it suffices to check the emptiness of the language $[u\$v]_{\hat{=}} - M_i$. There are various ways in which one can compute $[u\$v]_{\hat{=}}$. One way is by direct construction of a DFA accepting $[u\$v]_{\hat{=}}$ from the Büchi automaton that accepts the language containing a single word uv^ω. There is a detailed description of this construction in [5]. We use a different construction in our implementation which is presented in Appendix A.2.
- *The word uv^ω is a negative counterexample:* the word uv^ω should not be in $\omega(M_i)$, but it is. Since $uv^\omega \in L$, by Proposition 3, $[u\$v]_{\hat{=}} \cap M_i \neq \emptyset$ and there exists a word $u'\$v' \in [u\$v]_{\hat{=}}$ such that $u'\$v' \in M_i$. One can find this word by checking emptiness of $M_i \cap [u\$v]_{\hat{=}}$. Then $u'\$v'$ works as a proper negative counterexample for the L* algorithm.

Here is why the above procedure works: A conjecture M may not be saturated by $\hat{=}$. Consider the case presented in Figure 1(a). There are four equivalence classes: $[u_1\$v_1]_{\hat{=}}$ is contained in M, $[u_2\$v_2]_{\hat{=}}$ and $[u_3\$v_3]_{\hat{=}}$ have intersections with M but are not contained in it, and $[u_4\$v_4]_{\hat{=}}$ is completely outside M. Now assume L (as defined by 4) is the ω-regular language corresponding to M. Proposition 4 implies that $u_1v_1^\omega \in L$. Proposition 3 implies that $u_4v_4^\omega \notin L$. However, one cannot state anything about $u_2v_2^\omega$ and $u_3v_3^\omega$ with certainty; they may or may not be in L. Let us assume (for the sake of the argument) that $u_2v_2^\omega \in L$ and $u_3v_3^\omega \notin L$. This means that $L_\$$ (which is not equivalent to M) is actually the

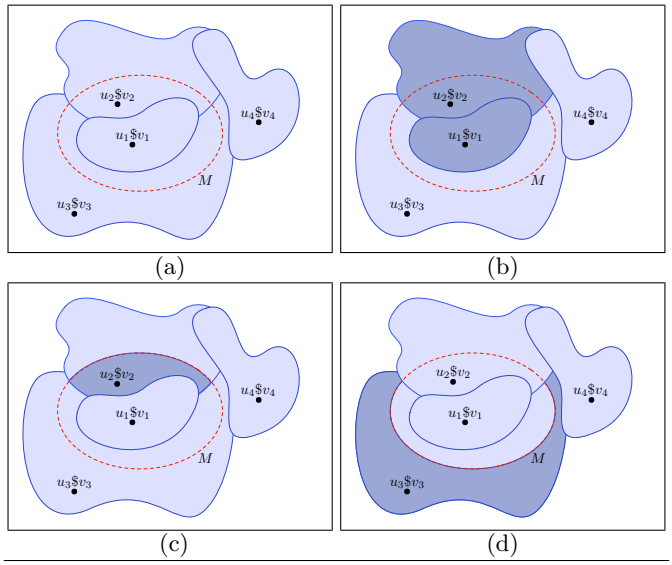


Fig. 1. The Case of Non-saturation

shaded area in Figure 1(b). Now, if L is not the correct conjecture, one will end up with an ω-word uv^ω as a counterexample. As mentioned above, we have one of the following two cases:

(1) The word uv^ω is a negative counterexample. There are two possibilities for the class $[u\$v]_{\triangleq}$:
- $[u\$v]_{\triangleq} \subseteq M$: This case is rather trivial. Any word in $[u\$v]_{\triangleq}$, including $u\$v$, belongs to M while it should not. Therefore, $u\$v$ can serve as a *proper* negative counterexample for the next iteration of L*.
- $[u\$v]_{\triangleq} \not\subseteq M$: This case is more tricky. looking back at Figure 1(b), it is similar to assuming that $u_2v_2^\omega$ was wrongly included in L. But since some of the strings in $[u\$v]_{\triangleq}$ do not belong to M, an arbitrary string from $[u\$v]_{\triangleq}$ does not necessarily work as a negative counterexample for the next iteration of L*. One has to find a string which is in both $[u\$v]_{\triangleq}$ and M, which means it belongs to $[u\$v]_{\triangleq} \cap M$. The *shaded* area in Figure 1(c) demonstrates this set for the example. Note that $[u\$v]_{\triangleq} \cap M$ cannot be empty; By Proposition 3, $[u\$v]_{\triangleq} \cap M = \emptyset$ implies that $uv^\omega \notin L$ which is a contradiction.

(2) The word uv^ω is a positive counterexample. There are two possibilities for the class $[u\$v]_{\triangleq}$:
- $[u\$v]_{\triangleq} \cap M = \emptyset$: This case is rather trivial. All words in $[u\$v]_{\triangleq}$, including $u\$v$, do not belong to M while they should. Therefore, $u\$v$ can serve as a *proper* positive counterexample for the next iteration of L*.
- $[u\$v]_{\triangleq} \cap M \neq \emptyset$: This case is more tricky. looking back at Figure 1(b), it is similar to assuming that $u_3v_3^\omega$ was wrongly left out of L. But since some of the strings in $[u\$v]_{\triangleq}$ do belong to M, an arbitrary string from that

class is not necessarily going to work as a *proper* positive counterexample for the next iteration of L*. We have to make sure to find one that is in $[u\$]_{\triangleq}$ but not in M. The set $[u\$v]_{\triangleq} - M$ contains such a string which is guaranteed to make L* *progress*. The *shaded* area in Figure 1(d) demonstrates this set for the example. Note that $[u\$v]_{\triangleq} - M$ cannot be empty; $[u\$v]_{\triangleq} - M = \emptyset$ implies that $[u\$v]_{\triangleq} \subseteq M$ in which case by Proposition 4, we have $uv^\omega \in L$, which is a contradiction.

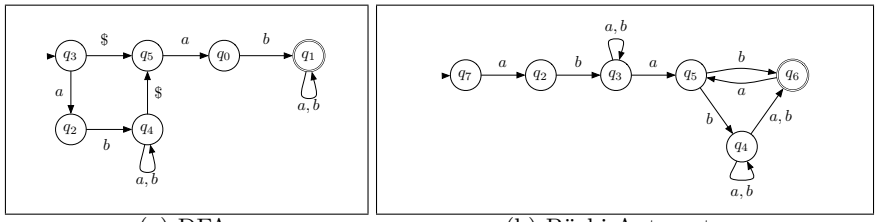
Below, we give a more technical description of our algorithm followed by an example for greater clarity.

Definition 3. *An observation table is a tuple $\langle S, E, T\rangle$ where S is a set of prefix-closed words in Σ^* such that each word in S represents a syntactic right congruence class of $L_\$$, E is a set of suffix-closed strings in Σ^* such that each word in E is a distinguishing word, and $T : (S \cup S\Sigma) \times E \longrightarrow \{-, +\}$ is defined as*

$$T(\alpha, \sigma) = \begin{cases} + & \text{if } \alpha\sigma \in L_\$ \\ - & \text{if } \alpha\sigma \notin L_\$. \end{cases}$$

An observation table is *closed* if for every word $s' \in S\Sigma$, there exists a word $s \in S$ such that $T(s, \bullet) = T(s', \bullet)$ (where $T(s, \bullet)$ indicates the row of the table which starts with s).

The goal of L* here is to learn $L_\$$ for an unknown ω-language L on alphabet $\Sigma \cup \{\$\}$. Our initial setting is the same as L*; the distinguishing experiment set $E = \{\lambda\}$ and the congruence class set $S = \{\lambda\}$. We fill the table by asking membership queries for each pair of strings $(\alpha, \sigma) \in (S \cup S\Sigma) \times E$; a "NO" response sets $T(\alpha, \sigma) = -$, and a "YES" response sets $T(\alpha, \sigma) = +$. Note that the membership queries are translated as discussed above to a format appropriate for the teacher.



(a) DFA. (b) Büchi Automaton.

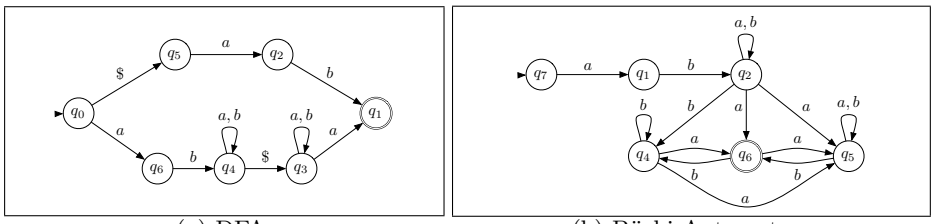
Fig. 2. First Iteration.

When the observation table is closed, a conjecture DFA $A = (S, \Sigma, q_0, F, \delta)$, where $Q = \{u | u \in S\}$, $q_0 = \lambda$, $\delta = \{(u, a, u') | u' \in S \wedge T(u', \bullet) = T(ua, \bullet)\}$, and $F = \{u | T(u, \lambda) = +\}$ is constructed from the table.

We then check if $M_0 = L(A)$ is a subset of $\Sigma^*\$\Sigma^+$. If not, there is a counterexample in $L(A_\$) \cap \overline{\Sigma^*\$\Sigma^+}$ from the language containment check, all of whose suffixes are added to the set of distinguishing words E. If $M_0 \subseteq \Sigma^*\$\Sigma^+$, we construct a Büchi automaton B based on A (see Section 3), and perform the equivalence check. The counterexamples are interpreted (as discussed above) and the

appropriate counterexamples are added to set E. We then proceed to another iteration of this algorithm, until the target language is found.

Example 1. We demonstrate our algorithm by showing the steps performed on a simple example. Assume that the target language is $ab((a+b)^*a)^\omega$. This ω-expression corresponds to the liveness property: "a happens infinitely often in a computation with the prefix ab" which cannot be learned using any of the existing algorithms.



(a) DFA. (b) Büchi Automaton.

Fig. 3. Second Iteration.

Table 1(a) shows the closed observation from the first iteration of our algorithm. Figure 2(a) demonstrates the DFA that corresponds to this observation table, and Figure 2(b) demonstrates the Büchi automaton constructed form this DFA. The first conjecture is not correct; the word $ab(a)^\omega$ belongs to the target language, but is not accepted by the automaton in Figure 2(b). Therefore, the algorithm goes into a second iteration. The counterexample is translated into one appropriate for the L* ($ab\$a$), and all its suffixes are added to the top row of the table. Table 1(b) is the closed table which we acquire after adding the counterexample. Figure 3(a) shows the DFA corresponding to this table, and Figure 3(b) shows the Büchi automaton constructed based on this DFA. This Büchi automaton passes the equivalence check and the algorithm is finished learning the target language.

5 Optimizations

In this section, we briefly discuss some practical optimizations that we have added to our implementation of the algorithm presented in Section 4 to gain a more efficient learning tool.

Equivalence query as the last resort: The equivalence query for an ω-regular language is expensive, even more than equivalence checking for regular languages. The main reason is that it requires complementing the Büchi automaton, which has a proven lower bound of $2^{O(n \log n)}$ [15]. Therefore, having fewer equivalence queries speeds up the process of learning. For each conjecture DFA A that is built during an iteration of the algorithm (more specifically, the incorrect conjectures), it does not have to be the case that $L(A)$ is saturated by $\stackrel{\circ}{=}$. If

one could check for saturation and make $\stackrel{\circ}{=}$ saturate $L(A)$ by adding/removing words, one could avoid going through with an (expensive) equivalence check that will most probably have a "NO" response. Unfortunately, there is no known way of effectively checking for saturation. But all is not lost. One can construct another DFA A' where $L(A') = (\Sigma^*\$\Sigma^+) \cap \overline{L(A)}$. Let B and B' be respectively

	λ	ab	b	$\$ab$	$a\$ab$	$aba\$ab$	$ba\$ab$
λ	−	−	−	+	−	−	+
a	−	−	−	−	−	+	−
b	−	−	−	−	−	−	−
$\$$	−	−	+	−	−	−	−
ab	−	−	−	+	+	+	+
$\$a$	−	+	−	−	−	−	−
$\$ab$	+	+	+	−	−	−	−
aa	−	−	−	−	−	−	−
$a\$$	−	−	−	−	−	−	−
ba	−	−	−	−	−	−	−
bb	−	−	−	−	−	−	−
$b\$$	−	−	−	−	−	−	−
$\$b$	−	−	−	−	−	−	−
$\$\$$	−	−	−	−	−	−	−
aba	−	−	−	+	+	+	+
abb	−	−	−	+	+	+	+
$ab\$$	−	−	+	−	−	−	−
$\$aa$	−	−	−	−	−	−	−
$\$a\$$	−	−	−	−	−	−	−
$\$aba$	+	+	+	−	−	−	−
$\$abb$	+	+	+	−	−	−	−
$\$ab\$$	−	−	−	−	−	−	−

(a) First Iteration.

	λ	ab	b	$\$ab$	$a\$ab$	$aba\$ab$	$ba\$ab$	a	$\$a$	$b\$a$	$ab\$a$
λ	−	−	−	+	−	−	+	−	−	−	+
a	−	−	−	−	−	+	−	−	−	+	−
b	−	−	−	−	−	−	−	−	−	−	−
$\$$	−	−	+	−	−	−	−	−	−	−	−
ab	−	−	−	+	+	+	+	−	+	+	+
$\$a$	−	+	−	−	−	−	−	−	−	−	−
$\$ab$	+	+	+	−	−	−	−	+	−	−	−
$ab\$$	−	−	+	−	−	−	−	+	−	−	−
aa	−	−	−	−	−	−	−	−	−	−	−
$a\$$	−	−	−	−	−	−	−	−	−	−	−
ba	−	−	−	−	−	−	−	−	−	−	−
bb	−	−	−	−	−	−	−	−	−	−	−
$b\$$	−	−	−	−	−	−	−	−	−	−	−
$\$b$	−	−	−	−	−	−	−	−	−	−	−
$\$\$$	−	−	−	−	−	−	−	−	−	−	−
aba	−	−	−	+	+	+	+	−	+	+	+
abb	−	−	−	+	+	+	+	−	+	+	+
$\$aa$	−	−	−	−	−	−	−	−	−	−	−
$\$a\$$	−	−	−	−	−	−	−	−	−	−	−
$\$aba$	+	+	+	−	−	−	−	+	−	−	−
$\$abb$	+	+	+	−	−	−	−	+	−	−	−
$\$ab\$$	−	−	−	−	−	−	−	−	−	−	−
$ab\$a$	+	+	+	−	−	−	−	+	−	−	−
$ab\$b$	−	−	+	−	−	−	−	+	−	−	−
$ab\$\$$	−	−	−	−	−	−	−	−	−	−	−

(b) Second Iteration.

Table 1. Observation Tables.

the corresponding Büchi automata for A and A'. If $L(B) \cap L(B') \neq \emptyset$ then there is $uv^\omega \in L(B) \cap L(B')$, and we know that only a part of the equivalence class $[u\$v]_{\stackrel{\circ}{=}}$ is in $L(A)$ and the rest of it is in $L(A')$. To decide whether the class should go into $L(A)$ (the conjecture) completely, or be altogether removed from it, we can pose a membership query for uv^ω to the teacher. If $uv^\omega \in L$, then the class should belong to the conjecture, and therefore any word in $[u\$v]_{\stackrel{\circ}{=}} \cap L(A')$ works as a positive counterexample for the next iteration of L*. If $uv^\omega \notin L$ then any word in $[u\$v]_{\stackrel{\circ}{=}} \cap L(A)$ can serve as a negative counterexample for the next iteration of the L*. This check is polynomial in the size of A, and saves us an unnecessary equivalence query.

Minimization and Simplification: Our algorithm constructs and handles many DFAs during the construction of the Büchi automaton from the conjecture DFA (from M_q's and N_{q,q_f}'s). Hence, the algorithm can benefit from minimizing all those DFAs in order to reduce the overhead of working with them later on. DFA minimization can be done very efficiently; the complexity is $n \log n$ [11], where n is the size of the source DFA.

When the conjecture Büchi automaton is built, another useful technique is to simplify the Büchi automaton by detecting a simulation relation between states [18]. Intuitively, a state p simulates another state q in the Büchi automaton if all accepting traces starting from q are also accepting traces starting from p. If p simulates q and both transitions $r \xrightarrow{a} p$ and $r \xrightarrow{a} q$ are in the automaton, then $r \xrightarrow{a} q$ can be safely removed without changing the language of the Büchi automaton. Furthermore, if p and q simulate each other, then after redirecting all of q's incoming transitions to p, q can be safely removed. This technique is useful for reducing the size of the result automaton, because the structures of M_q and N^{ω}_{q,q_f} are usually very similar, which provides good opportunities for finding simulation relations.

6 Preliminary Experimental Results

We have implemented our algorithm using JAVA. DFA operations are delegated to the *dk.brics.automaton* package, and the Büchi automaton equivalence checking function is provided by the GOAL tool [19].

	$\|\Sigma\| = 2$			$\|\Sigma\| = 4$		
	Avg	Min	Max	Avg	Min	Max
Target BA recognizing L	5.3	5	7	5.34	5	10
Learned DFA	7.98	3	21	8.7	5	16
Learned BA	6.78	3	11	12.92	5	35
Learned BA (after simplification)	5.36	2	8	9.38	3	24

(Unit: number of states)

Table 2. Results for Randomly Generated Temporal Formulas.

We check the performance of our tool by learning *randomly generated* ω-regular languages. More specifically, we combine the following 5 steps to get a target Büchi automaton:

1. Randomly generate LTL formulas with a length of 6 and with 1 or 2 propositions (which produces respectively Büchi automata with $|\Sigma| = 2$ and 4).
2. If the formula appeared before, discard it and go back to step 1.
3. Use the LTL2BA [9] algorithm to make them Büchi automata.
4. Apply the simplification [18] to make the Büchi automata as small as possible.
5. If the size of the automaton is smaller than 5, discard it and go to step 1.

Note that the combination of these steps does not guarantee the minimality of the resulting Büchi automaton.

Table 2 presents the performance of our algorithm on these randomly generated ω-regular languages. The sizes of the learned automata are compared with the sizes of the target automata. The result shows that the size the learned automaton is comparable with the size of the target automaton. Table 2 presents a summary of the results of 100 learning tasks. We have 50 are with $|\Sigma|= 2$ and another half of them with $|\Sigma| = 4$.

Property Classes	Canonical Formulas	Target		Learned		Responsive Formulas	Target		Learned		$\subseteq$ **DB**$\cap$ **coDB**?
		St	Trans	St	Trans		St	Trans	St	Trans	
Reactive	$\mathbf{FG}p \vee \mathbf{GF}q$	5	26	7	37	$\mathbf{GF}p \rightarrow \mathbf{GF}q$	5	26	7	37	No
Persistence	$\mathbf{FG}p$	2	4	3	7	$\mathbf{G}(p \rightarrow \mathbf{FG}q)$	4	18	11	43	No
Recurrence	$\mathbf{GF}p$	2	12	3	9	$\mathbf{G}(p \rightarrow \mathbf{F}q)$	3	12	13	65	No
Obligation	$\mathbf{G}p \vee \mathbf{F}q$	4	15	6	25	$\mathbf{F}p \rightarrow \mathbf{F}q$	4	15	6	25	Yes
Safety	$\mathbf{G}p$	2	2	2	3	$p \rightarrow \mathbf{G}q$	3	9	9	32	Yes
Guarantee	$\mathbf{F}p$	2	4	3	8	$p \rightarrow \mathbf{F}q$	3	12	4	20	Yes

Table 3. Effectiveness for learning automata from selected temporal formulas.

On a different note, we present the performance of our algorithm on learning properties that are often used in verification. Table 3 presents the result of these experiments. The target languages are described by temporal formulas selected from Manna and Pnueli [14] and classified according to the hierarchy of temporal properties which they proposed in the same paper. We translate those temporal formulas to Büchi automata by the LTL2BA algorithm. The first column of the table lists the six classes of the hierarchy. We select two temporal formulas for each class[2]. One of them is a formula in "canonical form"[3] and the other is a formula in "responsive form"[4]. Maler and Pnueli's algorithm [13] can only handle the bottom three levels of that hierarchy. Their algorithm cannot handle some important properties such as *progress* $\mathbf{G}(p \rightarrow \mathbf{F}q)$ and *strong fairness* $\mathbf{GF}p \rightarrow \mathbf{GF}q$, which can be handled by our algorithm.

7 Conclusions and Future Work

We extend the learning paradigm of the infinitary languages by presenting an algorithm to learn an *arbitrary* ω-regular language L over an alphabet Σ. Our main result is an algorithm to learn a nondeterministic Büchi automaton that recognizes an unknown ω-regular language by learning a unique projection of it ($L_\$$) on Σ^* using the L*[2] algorithm. We also present preliminary experimental results that suggest that algorithms performs well on small examples.

In the future, we would like to extend our experiments by learning bigger Büchi automata. We would also like to use this learning algorithm as a core of a compositional verification tool to equip the tool with the capability to check liveness properties that have been missing from such tools so far. One way of improving our algorithm is to find an effective way of checking for saturation, which appears to be difficult and remains unsolved.

[2] In this table, p and q are propositions. If one replaces p and q in a formula f with temporal formulas containing only past operators, f still belongs to the same class.

[3] The canonical formula is a simple representative formula for each class.

[4] A responsive formula usually contains two propositions p and q. The proposition p represents a stimulus and q is a response to p

References

1. R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *Proceedings of the 17th International Conference on Computer-Aided Verification (2005), LNCS 3576*, pages 548–562. Springer, 2005.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
3. A. Arnold. A syntactic congruence for rational omega-language. *Theoretical Computer Science*, 39:333–335, 1985.
4. J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1962.
5. H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics (1993), LNCS 802*, pages 554–566, 1993.
6. H. Calbrix, M Nivat, and A Podelski. Sur les mots ultimement périodiques des langages rationnels de mots infinis. *Comptes Rendus de l'Académie des Sciences*, 318:493–497, 1994.
7. S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *Proceedings of the 17th International Conference on Computer-Aided Verification (2005), LNCS 3576*, pages 534–547, 2005.
8. J.M. Cobleigh, D. Giannakopoulou, and C.S. Păsăreanu. Learning assumptions for compositional verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), LNCS 2619*, pages 331–346, 2003.
9. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translations. In *Proceedings of CAV (2001), LNCS 2102*, pages 53–65. Springer, 2001.
10. A. Gupta, K.L. McMillan, and Z. Fu. Automated assumption generation for compositional verification. In *Proceedings of the 19th International Conference on Computer-Aided Verification (2005), LNCS 4590*, pages 420–432, 2007.
11. J.E. Hopcroft. A $n \log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford University, 1971.
12. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
13. O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995.
14. Z. Manna and A. Pnueli. A hierarchy of temporal properties. Technical Report STAN-CS-87-1186, Stanford University, Department of Computer Science, 1987.
15. M. Michel. Complementation is more difficult with automata on infinite words. In *CNET, Paris*, 1988.
16. D. Perrin and J.E. Pin. *Infinite Words: Automata, Semigroups, Logic and Games.* Academic Press, 2003.
17. R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.
18. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of CAV (2000), LNCS 1855*, pages 248–263, 2000.
19. Y.-K Tsay, Y.-F Chen, M.-H Tsai, K.-N Wu, and W.-C Chan. GOAL: A graphical tool for manipulating buchi automata and temporal formulae. In *Proceedings of TACAS (2007), LNCS 4424*, pages 466–471.
20. D. L. Van, B. Le Saëc, and I. Litovsky. Characterizations of rational omega-languages by means of right congruences. *Theor. Comput. Sci.*, 143(1):1–21, 1995.

A Appendix

A.1 Proof of Theorem 4

We show that L (defined by (4)) is the ω-regular language for R. To do this, we have to show that $R = L_\$$.

($\mathbf{L_\$ \subseteq R}$): By definition, each pair (M_q, N_{q,q_f}) satisfies the hypothesis of Lemma 1, i.e. $M_q = M_q N^*_{q,q_f}$ and $N^+_{q,q_f} = N_{q,q_f}$. Therefore, if $\alpha \in \bigcup_{(q,q_f)\in Q_\$ \times F} M_q N^\omega_{q,q_f}$, then there exist $q_f \in F$, $u \in M_q$, and $v \in N_{q,q_f}$ such that $\alpha = uv^\omega$. Since $u \in M_q$ and $v \in N_{q,q_f}$, we have $u\$v \in R$. Since R is saturated by $\mathring{=}$, for all u', v' such that $\alpha = u'v'^\omega$, we have $u'\$v' \in R$. This argument holds for all $\alpha \in L$, and thus we have thus shown that $L_\$ \subseteq R$.

($\mathbf{R \subseteq L_\$}$): Let $u\$v$ be in R. We want to show that $uv^\omega \in L$. For all integers k, k', the word $uv^k\$v^{k'}$ corresponds to the same ultimately periodic word as $u\$v$, and therefore $u\$v \mathring{=} uv^k\$v^{k'}$. Since $\mathring{=}$ saturates R, we have $uv^k\$v^{k'} \in R$.

We would like to show that there exist $q \in Q_\$$ and $q_f \in F$ and integers k and k' such that $uv^k \in M_q$ and $v^{k'} \in N_{q,q_f}$, which will imply that $uv^k(v^{k'})^\omega = uv^\omega$ is in L.

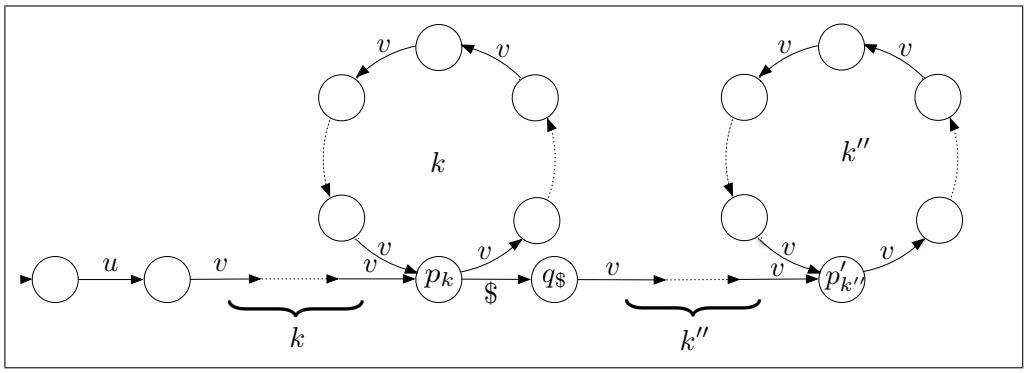


Fig. 4. Proof Diagram.

For each j, we know that $uv^j\$v \in R$. Thus, we have $\delta(q_i, uv^j) = p_j \in Q_\$$ (for some $q_i \in I$). Since $Q_\$$ is finite, there are two integers m, n such that $m \geq 1$ and $p_n = p_{n+m}$, and for each integer $k \leq n + m$, $p_k \notin \{p_0, \ldots p_{k-1}\}$. One can show by simple induction that $p_{k+m} = p_k$ for each integer $k \geq n$.

Let r be the remainder of dividing n by m ($n = lm + r$). Naturally, we have $r < m$. Set k to be $n+m-r = (l+1)m$. This way, we have $p_{k+k} = p_{k+(l+1)m} = p_k$, since $k > n$. By definition $p_k = \delta(q_i, uv^k)$, and therefore we have $\delta(p_k, v^k) = p_k$. We set $q = p_k$.

Using the same method, we can find an integer k'' such that $\delta(q, \$v^{kk''}) = p'_{k''}$ and $\delta(p'_{k''}, v^{kk''}) = p'_{k''}$. We set $k' = kk''$ and $q_f = p_{k''}$. This implies that $v^{k'} \in N_{q,q_f}$. Therefore, $uv^k(v^{k'})^\omega = uv^\omega$ is indeed in L. □

A.2 Generate $[u\$v]_{\triangleq}$ from uv^ω

An alternative to the construction presented here can be found in [5].

The operations *shift-in* and *reduce* on a string $s = u_0u_1 \dots u_n(v_0v_1 \dots v_m)^\omega$ are defined as follows. A shift-in operation is applicable to s if and only if $u_n = v_m$. Applying shift-in to s produces the string $s' = u_0u_1 \dots u_{n-1}(v_mv_1 \dots v_{m-1})^\omega$. Unfolding the ω-clauses of s and s' produces exactly the same infinite string, which implies $s = s'$. A reduce operation is applicable to s if and only if there exists some integers p and q such that $(v_0v_1 \dots v_p)^q = v_0v_1 \dots v_m$. Applying the reduce operation on s results in the string $s' = u_0u_1 \dots u_n(v_0v_1 \dots v_p)^\omega$. Similarly, by unfolding s and s' we can find $s = s'$. *shift-out* and *expand* are the reverse operations of *shift-in* and *reduce*.

uv^ω and $u'v'^\omega$ are *syntactically equivalent*, which is written as $uv^\omega \equiv u'v'^\omega$, if and only if $u = u'$ and $v = v'$. uv^ω is in *canonical form* if and only if for all $u'v'^\omega = uv^\omega, |u| \leq |u'|$ and $|v| \leq |v'|$. $P_n(v)$ denotes the prefix of v with length n.

Lemma 2. *Given two strings v, v' such that $v^\omega = v'^\omega$ and $n = GCD(|v|, |v'|)$, we have $P_n(v)^\omega = v^\omega$, where GCD represents the greatest common divider.*

Proof. Since $n = GCD(|v|, |v'|)$, there are integers k, m such that $|v| = mn$, $|v'| = kn$, and $GCD(m, k) = 1$. Therefore, we have $v = u_1 \dots u_m$ and $v' = u'_1 \dots u'_k$ such that $|u_i| = |u'_i| = n$ for all i. Since $v^\omega = v'^\omega$, we know that all the same-length prefixes of v^ω and v'^ω are equivalent. This means that $u_1 = u'_1$.

We next show that for all i, $u'_i = u_1$. To do this, we show that for all i, there is some prefix of v^ω of the form $\widehat{v}u_1$ and a prefix of v'^ω of the form $\widehat{v'}u'_i$ such that $|\widehat{v}| = |\widehat{v'}|$. This and the fact that $|u_1| = |u'_i| = n$ implies that $u_1 = u'_i$.

We know that $GCD(m, k) = 1$ implies $LCM(m, k) = mk$. Consider the prefix of v^ω and v'^ω which has length $mk-1$. This prefix is of the form $u_1 \dots u_mu_1 \dots u_m \dots = u'_1 \dots u'_ku'_1 \dots u'_k \dots$. Now, consider all positions at which u_1 starts: $0, m, 2m, \dots, (k-1)m$. It suffices that we show that u'_i (for all i) also starts from one of these positions. This means that for all i there is a $j \in \{0, m, 2m, \dots (k-1)m\}$ such that the remainder of $j = \alpha k+i$ (for some α). Let us assume this is not true; this means that there are two elements in that set that have the same remainder divided by k (there are exactly k elements in the set). Let these two be am and bm with $am < bm$. We can say then that $(b-a)m$ is dividable by k and therefore $LCM(m, k) \leq (b-a)k < mk$. This is a contradiction since $LCM(m, k) = mk$.

We have thus shown that $u'_1 = u'_2 = \dots = u'_k = u_1$. Using a similar argument we can show that $u_1 = u_2 = \dots = u_m = u'_1 = u_1 = P_n(v)$. Therefore, $v^\omega = v'^\omega = P_n(v)^\omega$.

Lemma 3. *An ultimately-periodic string uv^ω is not in canonical form if and only if shift-in or reduce operation is applicable on uv^ω.*

Proof. ($\Longleftarrow$) If shift or reduce operation is applicable to uv^ω, then we can obtain a string $u'v'^\omega$ with either a shorter prefix part or suffix part than uv^ω, by applying the applicable operation to uv^ω. Hence uv^ω is not in canonical form.

($\Longrightarrow$) If uv^ω is not in canonical form, then there exists u' and v' such that $u'v'^\omega = uv^\omega$ and $u'v'^\omega$ is in canonical form. By the definition of canonical form then there are two possibilities: (1) ($|u| = |u'|$) and ($|v| > |v'|$) or (2) ($|u| > |u'|$). In the case (1), we should have $GCD(|v'|, |v|) = |v'|$. Otherwise, by Lemma 2, the string $u'P_{GCD(|v|,|v'|)}(v')^\omega = u'v'^\omega$ contradicts the fact that $u'v'^\omega$ is in canonical form by being equivalent with a shorter presentation. Hence $GCD(|v'|, |v|) = |v'|$, and therefore there exists some positive integer $m > 1$ satisfying $v'^m = v$. The reduce operation is then applicable to uv^ω. In case (2), applying $|u|-|u'|$ shift-out operations to $u'v'^\omega$ produces a string $uw^\omega = uv^\omega$. $|v'| = |w| \leq |v|$ because shift operations do not change the size of the suffix part. If $GCD(|w|, |v|) < |w|$, by Lemma 2, we have $P_{GCD(|w|,|v|)}(w)^\omega = w^\omega$. Now, if we reverse the $|u|-|u'|$ shift-out operations by applying $|u| - |u'|$ shift-in operations to $uP_{GCD(|w|,|v|)}(w)^\omega$, we get a string $u'w'^\omega = u'v'^\omega$, $|w'| < |v'|$, which contradicts that $u'v'^\omega$ is in canonical form. Therefore, $GCD(|w|, |v|) = |w|$, and there exists some positive integer $m \geq 1$ satisfying $w^m = v$. When $m > 1$, reduce operation is applicable to uv^ω. If $m = 1$, $uv^\omega \equiv uw^\omega$ and shift-in operation is applicable to uw^ω. □

By lemma 3 and since shift-in and reduce operations are only applicable for a finite number of times (the lengths of the prefix and suffix are strictly decreasing and bounded from below), continuously applying shift-in and reduce operations on uv^ω until neither operation is applicable leads to the canonical form of uv^ω. Reversely, applying shift-out and expand operations on the canonical form of uv^ω by the inverted version of the process generates uv^ω.

Proposition 5. *Any ultimately periodic string uv^ω can be obtained from it its canonical form by finite applications of shift-out and expand operations.*

Proof. Direct consequence of Lemma 3.

Theorem 5. *Given a canonical string $uv^\omega \equiv u_1u_2 \dots u_n(v_1v_2 \dots v_m)^\omega$, the class $[u\$v]_{\triangleq}$ and the regular expression $\bigcup_{0\leq i\leq m} up_i(s_ip_i)^*\$(s_ip_i)^+$ represent the same set of strings, where the pair (p_i, s_i) is defined as follows:*

$$(p_i, s_i) = \begin{cases} (v_1 \dots v_i, v_{i+1} \dots v_m) & \text{if } 1 \leq i < m \\ (\lambda, v_1 \dots v_m) & \text{if } i = 0 \\ (v_1 \dots v_m, \lambda) & \text{if } i = m. \end{cases}$$

Proof. By continuous applications of *shift-out* operations to uv^ω, we can obtain the following set of strings:

$$\begin{aligned} \{u_1u_2 \dots u_n(v_1v_2 \dots v_m)^\omega, \\ u_1u_2 \dots u_nv_1(v_2v_3 \dots v_1)^\omega, \\ u_1u_2 \dots u_nv_1v_2(v_3v_4 \dots v_2)^\omega \cdots, \\ u_1u_2 \dots u_nv_1v_2 \dots v_{m-1}(v_mv_1 \dots v_{m-1})^\omega, \end{aligned}$$

$$
\begin{aligned}
&u_1u_2\dots u_nv_1v_2\dots v_m(v_1v_2\dots v_m)^\omega,\\
&u_1u_2\dots u_nv_1v_2\dots v_mv_1(v_2v_3\dots v_1)^\omega\\
&\dots\\
&u_1u_2\dots u_n(v_1v_2\dots v_m)^2(v_1v_2\dots v_m)^\omega,\\
&u_1u_2\dots u_nv_1(v_2\dots v_mv_1)^2(v_2v_3\dots v_1)^\omega,\cdots\}\\
&=\\
&u_1u_2\dots u_n(v_1v_2\dots v_m)^*(v_1v_2\dots v_m)^\omega \cup\\
&u_1u_2\dots u_nv_1(v_2\dots v_mv_1)^*(v_2v_3\dots v_1)^\omega \cup\dots\cup\\
&u_1u_2\dots u_nv_1v_2\dots v_{m-1}(v_mv_1\dots v_{m-1})^*(v_mv_1\dots v_{m-1})^\omega\\
&=\bigcup_{0\le i\le m} up_i(s_ip_i)^*(s_ip_i)^\omega
\end{aligned}
$$

Each word $u'\$v' \in [u\$v]_{\triangleq}$ corresponds to an ultimately periodic word $u'v'^\omega = uv^\omega$. By Proposition 5, we know that any word $u'v'^\omega$ can be obtained from uv^ω by finitely many shift-out and expand operations, and therefore it has to appear somewhere in $\bigcup_{0\le i\le m} up_i(s_ip_i)^*\$(s_ip_i)^+$.

Furthermore, if we try all possible expand operations after the shift-out operations, We get the set $\bigcup_{0\le i\le m} up_i(s_ip_i)^*((s_ip_i)^+)^\omega$. For each word $u'\$v' \in \bigcup_{0\le i\le m} up_i(s_ip_i)^*\$(s_ip_i)^+$, we have $u'v'^\omega \in \bigcup_{0\le i\le m} up_i(s_ip_i)^*((s_ip_i)^+)^\omega$, and therefore $u'v'^\omega$ equals uv^ω. It follows that $u'\$v' \in [u\$v]_{\triangleq}$ □