Innovations Syst Softw Eng (2011) 7:131–139
DOI 10.1007/s11334-011-0148-1

SI: NFM 2010

Automated assume-guarantee reasoning for omega-regular systems and specifications

Sagar Chaki · Arie Gurfinkel

Received: 13 September 2010 / Accepted: 26 February 2011 / Published online: 12 March 2011

Abstract We develop a learning-based automated assume-guarantee (AG) reasoning framework for verifying ω-regular properties of concurrent systems. We study the applicability of non-circular (**AG-NC**) and circular (**AG-C**) AG proof rules in the context of systems with infinite behaviors. In particular, we show that **AG-NC** is incomplete when assumptions are restricted to strictly infinite behaviors, while **AG-C** remains complete. We present a general formalization, called LAG, of the learning based automated AG paradigm. We show how existing approaches for automated AG reasoning are special instances of LAG. We develop two learning algorithms for a class of systems, called ∞-regular systems, that combine finite and infinite behaviors. We show that for ∞-regular systems, both **AG-NC** and **AG-C** are sound and complete. Finally, we show how to instantiate LAG to do automated AG reasoning for ∞-regular, and ω-regular, systems using both **AG-NC** and **AG-C** as proof rules.

Keywords Verification · Compositional reasoning · Assume guarantee · Learning · Omega-regular languages

1 Introduction

Compositional reasoning [8,13] is a widely used technique for tackling the *statespace explosion* problem while verifying concurrent systems. Assume guarantee (AG) is one of the most well-studied paradigms for compositional reasoning [14,19]. In AG-style analysis, we infer global properties of a system from the results of local analysis on its components. Typically, to analyze a system component C locally, we use an appropriate "assumption", a model of the rest of the system that reflects the behavior expected by C from its environment to operate correctly. The goal of the local analyses is then to establish that every assumption made is also "guaranteed"—hence assume guarantee.

Since its inception [16,18], the AG paradigm has been explored in several directions. However, a major challenge in automating AG reasoning is constructing appropriate assumptions. For realistic systems, such assumptions are often complicated, and, therefore, constructing them manually is impractical.

In this context, Cobleigh et al. [9] proposed the use of learning to automatically construct appropriate assumptions to verify a system composed of finite automata against a finite automaton specification (i.e., to verify safety properties). They used the following sound and complete AG proof rule:

$$\frac{M_1 \parallel A \sqsubseteq S \qquad M_2 \sqsubseteq A}{M_1 \parallel M_2 \sqsubseteq S}$$

where M_1, M_2, A and S are finite automata, $||$ is a parallel composition, and $\sqsubseteq$ denotes language containment. The essential idea is to use the $\mathbf{L}^*$ algorithm [2] to learn an assumption A that satisfies the premises of the rule, and implement the minimally adequate teacher required by $\mathbf{L}^*$ via model checking.

The learning-based automated AG paradigm has been extended in several directions [1,5,21]. However, the question of whether this paradigm is applicable to verifying ω-regular properties (i.e., liveness and safety) of reactive systems is open. In this paper, we answer this question in the affirmative. An automated AG framework requires: (i) an algorithm that uses queries and counterexamples to learn an appropriate assumption, and (ii) a set of sound and complete AG rules. Recently, a learning algorithm for ω-regular languages has been proposed by Farzan et al. [10]. However,

S. Chaki (✉) · A. Gurfinkel
Software Engineering Institute, Carnegie Mellon University,
4500 Fifth Ave., Pittsburgh, PA 15213-2612, USA
e-mail: chaki@sei.cmu.edu

to our knowledge, the AG proof rules have not been extended to ω-regular properties. This is the problem we address in this paper.

First, we study the applicability of non-circular (**AG-NC**) and circular (**AG-C**) AG proof rules in the context of systems with infinite behaviors. We assume that processes synchronize on shared events and proceeding asynchronously otherwise, i.e., as in CSP [15]. We prove that, in this context, **AG-NC** is sound but *incomplete* when restricted to languages with strictly infinite behaviors (e.g., ω-regular). This is surprising and interesting. In contrast, we show that **AG-C** is both sound and complete for ω-regular languages. *Second*, we extend our AG proof rules to systems and specifications expressible in ∞-regular languages (i.e., unions of regular and ω-regular languages). We show that both **AG-C** and **AG-NC** are sound and complete in this case. To the best of our knowledge, these soundness and completeness results are new. We develop two learning algorithms for ∞-regular languages—one using a learning algorithm for ω-regular languages (see Theorem 8(a)) with an augmented alphabet, and another combining a learning algorithm for ω-regular languages with $\mathbf{L}^*$ (see Theorem 8(b)) without alphabet augmentation. *Finally*, we present a very general formalization, called LAG, of the learning based automated AG paradigm. We show how existing approaches for automated AG reasoning are special instances of LAG. Furthermore, we show how to instantiate LAG to develop automated AG algorithms for ∞-regular and ω-regular languages using both AG-NC and AG-C as proof rules.

The rest of the paper is structured as follows. We present the necessary background in Sect. 2. In Sect. 3, we review our model of concurrency. In Sect. 4, we study the soundness and completeness of AG rules, and present our LAG framework in Sect. 5. We review the related work in Sect. 6, and conclude in Sect. 7.

2 Preliminaries

We write Σ^* and Σ^ω for the set of all finite and infinite words over Σ, respectively, and write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. We use the standard notation of regular expressions: λ for empty word, $a \cdot b$ for concatenation, a^*, a^+, and a^ω for finite, finite and non-empty, and infinite repetition of a, respectively. When $a \in \Sigma^\omega$, we define $a \cdot b = a$. These operations are extended to sets in the usual way, e.g., $X \cdot Y = \{x \cdot y \mid x \in X \wedge y \in Y\}$.

Language A language is a pair (L, Σ) such that Σ is an alphabet and $L \subseteq \Sigma^\infty$. The alphabet is an integral part of a language. In particular, $(\{a\}, \{a\})$ and $(\{a\}, \{a, b\})$ are different languages. However, for simplicity, we often refer to a language as L and mention Σ separately. For instance, we write "language L over alphabet Σ" to mean the language (L, Σ), and $\Sigma(L)$ to mean the alphabet of L. Union and intersection are defined as usual, but only for languages over the same alphabet. The complement of L, denoted $\overline{L}$, is defined as: $\overline{L} = \Sigma(L)^\infty \setminus L$. A finitary language ($\Sigma^*$-language) is a subset of Σ^*. An infinitary language (Σ^ω-language) is a subset of Σ^ω. For $L \subseteq \Sigma^\infty$, we write $*(L)$ for the finitary language $L \cap \Sigma^*$ and $\omega(L)$ for the infinitary language $L \cap \Sigma^\omega$. Note that $\Sigma(\overline{L}) = \Sigma(*(L)) = \Sigma(\omega(L)) = \Sigma(L)$.

Transition systems A labeled transition system (LTS) is a 4-tuple $M = (S, \Sigma, \mathit{Init}, \delta)$, where S is a finite set of states, Σ is an alphabet, $\mathit{Init} \subseteq S$ is the set of initial states, and $\delta \subseteq S \times \Sigma \times S$ is a transition relation. We write $s \xrightarrow{\alpha} s'$ for $(s, \alpha, s') \in \delta$, and $\Sigma(M)$ for Σ. M is deterministic if $|\mathit{Init}| \leq 1$, and $\forall s \in S \centerdot \forall \alpha \in \Sigma \centerdot |\{s' \mid s \xrightarrow{\alpha} s'\}| \leq 1$. A run r over a word $w = \alpha_0, \alpha_1, \ldots, \in \Sigma(M)^\infty$ is a sequence of states $s_0, s_1, \ldots$, such that $\forall i \geq 0 \centerdot s_i \xrightarrow{\alpha_i} s_{i+1}$. We write $\mathit{First}(r)$, $\mathit{Last}(r)$, and $\mathit{Inf}(r)$ to denote the first state of r, the last state of r (assuming $r \in S^*$), and states that occur infinitely often in r (assuming $r \in S^\omega$), respectively. We write $\mathit{Run}(w, M)$ for the set of runs of w on M.

Automata A finite automaton (FA) is a 5-tuple $A = (S, \Sigma, \mathit{Init}, \delta, F)$, where $(S, \Sigma, \mathit{Init}, \delta)$ is an LTS and $F \subseteq S$ is a set of accepting states. The language accepted by A, $\mathcal{L}(A)$, is the set of all words $w \in \Sigma^*$ s.t. there exists a run r of w on A, with $\mathit{First}(r) \in \mathit{Init} \wedge \mathit{Last}(r) \in F$. A Büchi Automaton (BA) is a 5-tuple $B = (S, \Sigma, \mathit{Init}, \delta, F)$, where $(S, \Sigma, \mathit{Init}, \delta)$ is an LTS and $F \subseteq S$ is a set of accepting states. The language accepted by B, $\mathcal{L}(B)$, is the set of all words $w \in \Sigma^\omega$ s.t. there exists a run r of w on A with $\mathit{First}(r) \in \mathit{Init} \wedge \mathit{Inf}(r) \cap F \neq \emptyset$. A BA or FA is deterministic if its underlying LTS is deterministic.

Regularity A language is regular (ω-regular) iff it is accepted by a FA (BA). A language $L \subseteq \Sigma^\infty$ is ∞-regular iff $*(L)$ is regular and $\omega(L)$ is ω-regular. Deterministic FA (DFA) and non-deterministic FA (NFA) are equally expressive. Deterministic BA are strictly less expressive than non-deterministic BA.

Learning A learning algorithm for a regular language is any algorithm that learns an unknown, but fixed, language U over a known alphabet Σ. Such an algorithm is called *active* if it works by querying a Minimally Adequate Teacher (MAT). The MAT can answer "Yes/No" to two types of queries about U:

Membership query Given a word w, is $w \in U$?
Candidate query Given an automaton B, is $\mathcal{L}(B) = U$? If the answer is "No", the MAT returns a counterexample (CE), which is a word such that $CE \in \mathcal{L}(B) \ominus U$, where $X \ominus Y = (X \setminus Y) \cup (Y \setminus X)$.

An active learning algorithm begins by asking membership queries of the MAT until it constructs a candidate, with which it makes a candidate query. If the candidate query is successful, the algorithm terminates; otherwise, it uses the *CE* returned by the MAT to construct additional membership queries. The family of active learning algorithms was originated by Angluin via $\mathbf{L}^*$ [2] for learning a minimal DFA that accepts an unknown regular language. $\mathbf{L}^*$ was further optimized by Rivest and Schapire [20].

The problem of learning a *minimal* automaton which accepts an unknown ω-regular language is still open. It is known [17] that for any language U one can learn in the limit an automaton that accepts U via the *identification by enumeration* approach proposed by Gold [12]. However, the automaton learned via enumeration may, in the worst case, be exponentially larger than the minimal automaton accepting U. Furthermore, there may be multiple minimal automata [17] accepting U. Maler et al. [17] have shown that $\mathbf{L}^*$ can be extended to learn a minimal (Müller) automaton for a fragment of ω-regular languages.

Farzan et al. [10] show how to learn a Büchi automaton for an ω-regular language U. Specifically, they use $\mathbf{L}^*$ to learn the language $U_\$ = \{u\$v \mid u \cdot v^\omega \in U\}$, where \$ is a fresh letter not in the alphabet of U. The language $U_\$$ was shown to be regular by Calbrix et al. [4]. In the sequel, we refer to this algorithm as $\mathbf{L}^\$$. The complexity of $\mathbf{L}^\$$ is exponential in the minimal BA for U. Our LAG framework can use any active algorithm for learning ω-regular languages. In particular, $\mathbf{L}^\$$ is an existing candidate.

3 Model of concurrency

Let w be a word and Σ an arbitrary alphabet. We write $w \downharpoonright \Sigma$ for the projection of w onto Σ defined recursively as follows (recall that λ denotes the empty word):

$$\lambda \downharpoonright \Sigma = \lambda \qquad (a \cdot u) \downharpoonright \Sigma = \begin{cases} a \cdot (u \downharpoonright \Sigma) & \text{if } a \in \Sigma \\ u \downharpoonright \Sigma & \text{otherwise} \end{cases}$$

Clearly, both Σ^* and Σ^∞ are closed under projection, but Σ^ω is not. For example, $(a^* \cdot b^\omega \downharpoonright \{a\}) = a^*$, and a^* consists only of finite words. Projection preservers regularity. If L is a regular (∞-regular) language and Σ is any alphabet, then $L \downharpoonright \Sigma$ is also regular (∞-regular).

A process is modeled by a language of all of its behaviors (or computations). *Parallel composition* ($||$) of two processes/languages synchronizes on common actions while executing local actions asynchronously. For languages (L_1, Σ_1) and (L_2, Σ_2), $L_1||L_2$ is the language over $\Sigma_1 \cup \Sigma_2$ defined as follows:

$$L_1 \parallel L_2 = \{w \in (\Sigma_1 \cup \Sigma_2)^\infty \mid w \downharpoonright \Sigma_1 \in L_1 \wedge w \downharpoonright \Sigma_2 \in L_2\} \qquad \text{(def. of } ||)$$

Intuitively, $L_1||L_2$ consists of all permutations of words from L_1 and L_2 that have a common synchronization sequence. For example, $(b^* \cdot a \cdot b^*)||(c^* \cdot a \cdot c^*)$ is $(b+c)^* \cdot a \cdot (b+c)^*$. Note that when L_1 and L_2 share an alphabet, the composition is their intersection; when their alphabets are disjoint, the composition is their language shuffle. The set of Σ^*, Σ^ω, and Σ^∞ languages are all closed under parallel composition. The following theorem illustrates some other useful properties of parallel composition.

Theorem 1 *The* $||$ *operator is associative, commutative, distributive over union and intersection. It is also monotone, i.e., for any two languages* L_1, L_2, *and* L_3: $L_2 \subseteq L_3 \Rightarrow (L_1||L_2) \subseteq (L_1||L_3)$.

Let L_1 and L_2 be two languages such that $\Sigma(L_1) \supseteq \Sigma(L_2)$. We say that L_1 is subsumed by L_2, written $L_1 \preccurlyeq L_2$, if $L_1 \downharpoonright \Sigma(L_2) \subseteq L_2$. Thus, $L_1 \preccurlyeq L_2$ means that L_1 is subsumed by L_2 when projected on the alphabet of common actions. Let L_S be the language of a specification S, and L_M be the language of a system M. Then, M satisfies S, written $M \models S$, iff $L_M \preccurlyeq L_S$.

4 Proof rules for assume-guarantee reasoning

In this section, we study the applicability of a non-circular and a circular AG rule to prove properties of processes with infinite behaviors (e.g., reactive systems that neither terminate nor deadlock). These rules were shown to be sound and complete for systems with finite (i.e., in Σ^*) behaviors by Barringer et al. [3]. In Sect. 4.1, we show that the non-circular AG rule is sound for both Σ^∞ and Σ^ω behaviors. However, it is complete only when the assumptions are allowed to combine *both* finite and infinite behaviors (i.e., in Σ^∞). In Sect. 4.2, we show that the circular AG rule is sound and complete for Σ^ω and Σ^∞ behaviors.

4.1 Non-circular assume-guarantee rule

The non-circular AG proof rule (**AG-NC** for short) is stated as follows:

$$\frac{(L_1 \parallel L_A) \preccurlyeq L_S \qquad L_2 \preccurlyeq L_A}{(L_1 \parallel L_2) \preccurlyeq L_S}$$

where L_1, L_2, L_S, and L_A are languages with the alphabets $\Sigma_1, \Sigma_2, \Sigma_S, \Sigma_A$, respectively, $\Sigma_S \subseteq (\Sigma_1 \cup \Sigma_2)$, and $\Sigma_A = (\Sigma_1 \cup \Sigma_S) \cap \Sigma_2$. **AG-NC** is known to be sound and complete for Σ^*-languages. Intuitively, it says that if there exists an assumption L_A such that: (a) L_1 composed with L_A is contained in L_S, and (b) L_2 is contained in L_A, then the composition of L_1 with L_2 is contained in L_S as well. Note that the alphabet Σ_A is the smallest alphabet containing: (a) actions at the interface between L_1 and L_2, i.e., actions common to the alphabets of L_1 and L_2, and (b) external actions of L_2, i.e., actions common to the alphabets of L_2 and L_S. Any smaller alphabet makes the rule trivially incomplete; any larger alphabet exposes internal (i.e., non-external) actions of L_2. It is not surprising that **AG-NC** remains sound even when applied to languages with infinite words. However, **AG-NC** is *incomplete* when L_A is restricted to Σ^ω-languages:

Theorem 2 *There exists $L_1, L_2, L_S \subseteq \Sigma^\omega$ such that $(L_1||L_2) \preccurlyeq L_S$, but there does not exists an assumption $L_A \subseteq \Sigma^\omega$ that satisfies all of the premises of* **AG-NC**.

Proof By example. Let L_1, L_2, L_S, and their alphabets be defined as follows:

$$\Sigma_1 = \{a, b\} \qquad \Sigma_2 = \{a, c\} \qquad \Sigma_S = \{a, b\}$$
$$L_1 = (a+b)^\omega \qquad L_2 = a^* c^\omega \qquad L_S = (a+b)^* b^\omega$$

The conclusion of **AG-NC** rule is satisfied since $(L_1||L_2) \downharpoonleft \Sigma_S = (a+b)^* b^\omega = L_S$. The alphabet Σ_A of L_A is $(\Sigma_1 \cup \Sigma_S) \cap \Sigma_2 = \{a\}$. Since $L_A \subseteq \Sigma_A^\omega$, it can only be a^ω or $\emptyset$. The only way to satisfy the first premise of **AG-NC** is to let $L_A = \emptyset$, but this is too strong to satisfy the second premise. □

Note that the proof of Theorem 2 shows that **AG-NC** is incomplete even for ∞-regular languages.

Remark 1 One may conjecture that the **AG-NC** rule becomes complete for Σ^ω if subsumption is redefined to only consider infinite words. That is, by redefining subsumption as:

$$L_1 \preccurlyeq L_2 \Leftrightarrow \omega(L_1 \downharpoonleft \Sigma(L_2)) \subseteq L_2.$$

However, under this interpretation, **AG-NC** is no longer sound. For example, let the languages L_1, L_2, L_S, and their alphabets be defined as follows:

$$\Sigma_1 = \{a, b\} \qquad \Sigma_2 = \{a, c\} \qquad \Sigma_S = \{a, b\}$$
$$L_1 = (a+b)^\omega \qquad L_2 = a^* c^\omega \qquad L_S = b^\omega$$

Then, the conclusion of **AG-NC** does not hold:

$$\omega((L_1||L_2) \downharpoonleft \Sigma_S) = (a+b)^* b^\omega \not\subseteq b^\omega.$$

But, $L_A = \emptyset$ satisfies both premises: $(L_1||L_A) = b^\omega$, and $\omega(L_2 \downharpoonleft \{a\}) = L_A$.

Remark 2 **AG-NC** is complete if the alphabet Σ_A is redefined to be $\Sigma_1 \cup \Sigma_2$. However, in this case, the rule is no longer "compositional" since the assumption L_A can be as expressive as the component L_2.

Intuitively, **AG-NC** is incomplete for Σ^ω because Σ^ω is not closed under projection. However, we show that the rule is complete for Σ^∞ – the smallest projection-closed extension of Σ^ω. We first show that for any languages L_1 and L_S, there always exists a unique weakest assumption L_A, such that $L_1||L_A \preccurlyeq L_S$.

Theorem 3 *Let L_1 and L_S be two languages, and Σ_A be any alphabet s.t. $\Sigma(L_1) \cup \Sigma_A = \Sigma(L_1) \cup \Sigma(L_S)$. Then, $L_A = \{w \in \Sigma_A^\infty \mid (L_1||\{w\}) \preccurlyeq L_S\}$ satisfies $L_1||L_A \preccurlyeq L_S$, and is the weakest such assumption.*

Proof Let us write Σ_1, Σ_S and Σ_{1S} to mean $\Sigma(L_1)$, $\Sigma(L_S)$ and $\Sigma(L_1) \cup \Sigma(L_S)$ respectively. To show that L_A is a valid assumption, pick any $w \in L_1 \parallel L_A$. Then $w \downharpoonleft \Sigma_A \in L_A$. This implies that $w \downharpoonleft \Sigma_S \in (L_1 \parallel \{w \downharpoonleft \Sigma_A\}) \downharpoonleft \Sigma_S \subseteq L_S$. Since w is any word in $L_1 \parallel L_A$, we have $L_1 \parallel L_A \preccurlyeq L_S$. To show that L_A is the weakest assumption, let $L'_A \subseteq \Sigma_A^\infty$ be any language such that $L_1 \parallel L'_A \preccurlyeq L_S$ and let w be any word in L'_A. Then, $(L_1 \parallel \{w\}) \subseteq (L_1 \parallel L'_A) \preccurlyeq L_S$. But this implies that $w \in L_A$, and, therefore, $L'_A \subseteq L_A$. □

Note that Σ_A^∞ subsumes both finite (Σ_A^*) and infinite (Σ_A^ω) words. Thus, if L_A is a Σ_A^∞ weakest assumption, then $*(L_A)$ and $\omega(L_A)$ are the weakest Σ_A^* and Σ_A^ω assumptions, respectively.

Theorem 4 *Let L_1, L_2, L_S, and L_A be in Σ^∞. Then, the* **AG-NC** *rule is sound and complete.*

Proof The proof of soundness is trivial and is omitted. For the proof of completeness, we only show the key step. Assume that $L_1||L_2 \preccurlyeq L_S$, and let L_A be the weakest assumption such that $L_1||L_A \preccurlyeq L_S$. By Theorem 3, L_A is well defined and satisfies the first premise of **AG-NC**. The second premise holds because $L_2 \downharpoonleft \Sigma_A \subseteq \Sigma_A^\infty$, and L_A is the weakest Σ_A^∞ assumption (see Theorem 3). □

Theorem 4 implies that **AG-NC** is sound for any fragment of Σ^∞. Of course, this is not true for completeness of the rule. For practical purposes, we would like to know that the rule remains complete when its languages are restricted to the regular subset. We show that this is so by showing that under the assumption that L_1 and L_S are regular, the weakest assumption is regular as well.

Theorem 5 *Let L_1 and L_S be two languages, and Σ_A be any alphabet such that $\Sigma(L_1) \cup \Sigma_A = \Sigma(L_1) \cup \Sigma(L_S)$. Then, $L_A \subseteq \Sigma_A^\infty$ is the weakest assumption such that $L_1||L_A \preccurlyeq L_S$ iff $L_A = \overline{(L_1 \parallel \overline{L_S}) \downharpoonleft \Sigma_A}$.*

Proof Let us write Σ_1, Σ_S and Σ_{1S} to mean $\Sigma(L_1)$, $\Sigma(L_S)$ and $\Sigma(L_1) \cup \Sigma(L_S)$, respectively. For any $w \in \Sigma_A^\infty$:

$$
\begin{aligned}
& w \in \overline{(L_1 \parallel \overline{L_S}) \downharpoonleft \Sigma_A} \\
\textbf{iff } & \forall w' \in \Sigma_{1S}^\infty \centerdot \{w'\} \preccurlyeq \{w\} \implies w' \notin (L_1 \parallel \overline{L_S}) \\
\textbf{iff } & \forall w' \in \Sigma_{1S}^\infty \centerdot \{w'\} \preccurlyeq \{w\} \implies \\
& \qquad\qquad (\{w'\} \not\preccurlyeq L_1 \vee \{w'\} \preccurlyeq L_S) \\
\textbf{iff } & \forall w' \in \Sigma_{1S}^\infty \centerdot (\{w'\} \preccurlyeq \{w\} \wedge \{w'\} \preccurlyeq L_1) \implies \\
& \qquad\qquad \{w'\} \preccurlyeq L_S \\
\textbf{iff } & \forall w' \in \Sigma_{1S}^\infty \centerdot (\{w'\} \preccurlyeq (L_1 \parallel \{w\})) \implies \{w'\} \preccurlyeq L_S \\
\textbf{iff } & L_1 \parallel \{w\} \preccurlyeq L_S
\end{aligned}
$$

Together with Theorem 3, this completes the proof. □

Theorem 5 implies **AG-NC** is complete for any class of languages closed under complementation and projection, e.g., regular and ∞-regular languages. In addition, Theorem 5 implies that learning-based automated AG reasoning is effective for any class of languages whose weakest assumptions fall in a "learnable" fragment. In particular, this holds for regular, ω-regular and ∞-regular languages.

4.2 Circular assume-guarantee rule

The circular assume-guarantee proof rule (**AG-C** for short) is stated as follows:

$$\frac{(L_1 \parallel L_{A1}) \preccurlyeq L_S \quad (L_2 \parallel L_{A2}) \preccurlyeq L_S \quad (\overline{L_{A1}} \parallel \overline{L_{A2}}) \preccurlyeq L_S}{(L_1 \parallel L_2) \preccurlyeq L_S}$$

where L_1, L_2, and L_S are languages over alphabets Σ_1, Σ_2, Σ_S, respectively; $\Sigma_S \subseteq \Sigma_1 \cup \Sigma_2$, and L_{A1} and L_{A2} share a common alphabet $\Sigma_A = (\Sigma_1 \cap \Sigma_2) \cup \Sigma_S$. **AG-C** is known to be sound and complete for Σ^*-languages. Note that in comparison with **AG-NC**, there are two assumptions L_{A1} and L_{A2} over a larger alphabet Σ_A. Informally, the rule is sound for the following reason. Let w be a word in $L_1||L_2$, and $u = w \downharpoonright \Sigma_A$. Then, $u \in L_{A1}$, or $u \in L_{A2}$, or $u \in \overline{L_{A1} \cup L_{A2}} = (\overline{L_{A1}}||\overline{L_{A2}})$. If $u \in L_{A1}$ then the first premise implies that $\{w\} \preccurlyeq L_1||\{u\} \preccurlyeq L_S$; if $u \in L_{A2}$ then the second premise implies that $\{w\} \preccurlyeq L_2||\{u\} \preccurlyeq L_S$; otherwise, the third premise implies that $\{w\} \preccurlyeq \{u\} \preccurlyeq L_S$.

Remark 3 Note that the assumption alphabet for **AG-C** is larger than **AG-NC**. In fact, using $\Sigma_{A1} = (\Sigma_1 \cup \Sigma_S) \cap \Sigma_2$ and $\Sigma_{A2} = (\Sigma_2 \cup \Sigma_S) \cap \Sigma_1$ makes **AG-C** incomplete. Indeed, let $L_1 = \{aa\}$ with $\Sigma_1 = \{a\}$, $L_2 = \{bb\}$ with $\Sigma_2 = \{b\}$ and $\overline{L_S} = \{aab, abb, ab\}$. Note that $L_1||L_2 \preccurlyeq L_S$. We show that no L_{A_1} and L_{A_2} can satisfy the three premises of **AG-C**. Premise 1 $\Rightarrow b \notin L_{A_1} \Rightarrow b \in \overline{L_{A_1}}$. Similarly, premise 2 $\Rightarrow a \notin L_{A_2} \Rightarrow a \in \overline{L_{A_2}}$. But then $ab \in \overline{L_{A1}}||\overline{L_{A_2}}$, violating premise 3.

In this section, we show that **AG-C** is sound and complete for both Σ^ω and Σ^∞ languages. First, we illustrate an application of the rule to the example from the proof of Theorem 2. Let L_1, L_2, and L_S be Σ^ω languages as defined in the proof of Theorem 2. In this case, the alphabet Σ_A is $\{a, b\}$. Letting $L_{A1} = (a + b)^* b^\omega$, and $L_{A2} = (a + b)^\omega$ satisfies all three premises of the rule.

Theorem 6 *Let L_1, L_2, L_S, L_{A1}, and L_{A2} be in Σ^ω or Σ^∞. Then, the* **AG-C** *rule is sound and complete.*

Proof The proof of soundness is sketched in the above discussion. For the proof of completeness, we only show the key steps. Assume that $L_1||L_2 \preccurlyeq L_S$. Let L_{A1} and L_{A2} be the weakest assumptions such that $L_1||L_{A1} \preccurlyeq L_S$, and $L_2||L_{A2} \preccurlyeq L_S$, respectively. By Theorem 3, both L_{A1} and L_{A2} are well defined and satisfy the first and the second premises of **AG-C**, respectively. We prove the third premise by contradiction. Since L_{A1} and L_{A2} have the same alphabet, $(\overline{L_{A1}}||\overline{L_{A2}}) = (\overline{L_{A1}} \cap \overline{L_{A2}})$. Assume that $(\overline{L_{A1}} \cap \overline{L_{A2}}) \not\preccurlyeq L_S$. Then, there exists a word $w \in (\overline{L_{A1}}||\overline{L_{A2}})$ such that $w \notin L_{A1}$, and $w \notin L_{A2}$, and $w \downharpoonright \Sigma_S \notin L_S$. By the definition of weakest assumption (see Theorem 3), $L_1||\{w\} \not\preccurlyeq L_S$ and $L_2||\{w\} \not\preccurlyeq L_S$. Pick any $w_1 \in L_1||\{w\}$ and $w_2 \in L_2||\{w\}$. Let $w_1' = w_1 \downharpoonright \Sigma_1$ and $w_2' = w_2 \downharpoonright \Sigma_2$. We know that $\{w_1'\}||\{w_2'\} \subseteq L_1||L_2$. Also, $w \in (\{w_1'\}||\{w_2'\}) \downharpoonright \Sigma_A$. Now since $\{w_1'\}||\{w_2'\} \subseteq L_1||L_2$, we have $w \in (L_1||L_2) \downharpoonright \Sigma_A$. Since $\Sigma_S \subseteq \Sigma_A$, $w \downharpoonright \Sigma_S \in (L_1||L_2) \downharpoonright \Sigma_S$. But $w \downharpoonright \Sigma_S \notin L_S$, which contradicts $L_1||L_2 \preccurlyeq L_S$. □

Intuitively, **AG-C** is complete even for ω-regular languages because, unlike **AG-NC**, assumptions always appear to the left of $\preccurlyeq$ in the premises of **AG-C**. Thus, restricting the assumptions to only infinitary behaviors does not invalidate any of the premises of **AG-C**. This is true even though the assumptions appear in negated form in the third premise of **AG-C**.

The completeness part of the proof of Theorem 6 is based on the existence of the weakest assumption. We already know from Theorem 5 that the weakest assumption is (∞-,ω-)regular if L_1, L_2, and L_S are (∞-,ω-)regular, respectively. Thus, **AG-C** is complete for (∞-,ω-)regular languages. Because **AG-NC** is incomplete for ω-regular languages, a learning algorithm for ω-regular languages (such as $\mathbf{L}^\$$) cannot be applied directly for AG reasoning for ω-regular systems and specifications. In the next section, we overcome this challenge by developing automated AG algorithms for ∞-regular and ω-regular languages.

5 Automated assume-guarantee reasoning

In this section, we present our LAG framework, and its specific useful instances. LAG uses membership oracles, learners, and checkers, which we describe first.

Definition 1 (*Membership Oracle and Learner*) A membership oracle Q for a language U over alphabet Σ is a procedure that takes as input a word $u \in \Sigma^\infty$ and returns 0 or 1 such that $Q(u) = 1 \iff u \in U$. We say that $Q \models U$. The set of all membership oracles is denoted by **Oracle**. Let $\mathcal{A}$ be any set of automata. We write $\mathbf{Learner}_\mathcal{A}$ to denote the set of all learners of type $\mathcal{A}$. Formally, a learner of type $\mathcal{A}$ is a pair (**Cand**, **LearnCE**) such that: (i) **Cand** : **Oracle** $\mapsto \mathcal{A}$ is a procedure that takes a membership oracle as input and outputs a candidate $C \in \mathcal{A}$, and (ii) **LearnCE** : $\Sigma^\infty \mapsto \mathbf{Learner}_\mathcal{A}$ is a procedure that takes a counterexample as input and returns a new learner of type $\mathcal{A}$. For any learner $P =$ (**Cand**, **LearnCE**) we write P.**Cand** and P.**LearnCE** to mean **Cand** and **LearnCE** respectively.

Intuitively, a membership oracle is the fragment of a MAT that only answers membership queries, while a learner encapsulates an active learning algorithm that is able to construct candidates via membership queries, and learn from counterexamples of candidate queries.

Learning Let U be any unknown language, Q be an oracle, and P be a learner. We say that (P, Q) learns U if the following holds: if $Q \models U$, then there does not exist an infinite sequence of learners $P_0, P_1, \ldots$ and an infinite sequence of counterexamples $CE_1, CE_2, \ldots$ such that: (i) $P_0 = P$, (ii) $P_i = P_{i-1}.\mathbf{LearnCE}(CE_i)$ for $i > 0$, and (iii) $CE_i \in \mathcal{L}(P_{i-1}.\mathbf{Cand}(Q)) \ominus U$ for $i > 0$.

Definition 2 (*Checker*) Let $\mathcal{A}$ be a set of automata, and k be an integer denoting the number of candidates. A checker of type $(\mathcal{A}, k)$ is a procedure that takes as input k elements $A_1, \ldots, A_k$ of $\mathcal{A}$ and returns either (i) **SUCCESS**, or (ii) a pair (**FAILURE**, CE) such that $CE \in \Sigma^\infty$, or (iii) a triple (**FEEDBACK**, i, CE) such that $1 \leq i \leq k$ and $CE \in \Sigma^\infty$. We write $\mathbf{Checker}_{(\mathcal{A},k)}$ to mean the set of all checkers of type $(\mathcal{A}, k)$.

Intuitively, a checker generalizes the fragment of a MAT that responds to candidate queries by handling multiple (specifically, k) candidates. This generalization is important for circular proof rules. The checker has three possible outputs: (i) **SUCCESS** if the overall verification succeeds; (ii) (**FAILURE**, CE) where CE is a real counterexample; (iii) (**FEEDBACK**, i, CE) where CE is a counterexample for the ith candidate.

5.1 LAG procedure

Our overall LAG procedure is presented in Fig. 1. We write $X : T$ to mean that "X is of type T". LAG accepts a set of k membership oracles, k learners, and a checker, and repeats the following steps:

1. Constructs candidate automata $C_1, \ldots, C_k$ using the learners and oracles.
2. Invokes the checker with the candidates constructed in Step 1 above.
3. If the checker returns **SUCCESS** or (**FAILURE**, CE), then exits with this result. Otherwise, updates the appropriate learner with the feedback and repeats from Step 1.

Theorem 7 *LAG terminates if there exists languages $U_1, \ldots, U_k$ such that:* (i) $Q_i \models U_i$ *for* $1 \leq i \leq k$, (ii) (P_i, Q_i) *learns* U_i *for* $1 \leq i \leq k$, *and* (iii) *if* $V(C_1, \ldots, C_k) =$ (**FEEDBACK**, i, CE), *then* $CE \in \mathcal{L}(C_i) \ominus U_i$.

Proof By contradiction. If LAG does not terminate there exists some P_i such that $P_i.\mathbf{LearnCE}$ is called infinitely often. This, together with assumptions (i) and (iii), contradicts (ii), i.e., (P_i, Q_i) learns U_i. □

```
Input:
    $P_1 \ldots P_k$ : Learner$_{\mathcal{A}}$;
    $Q_1, \ldots, Q_k$ : Oracle;
    $V$ : Checker$_{(\mathcal{A},k)}$
forever do
    for $i = 1$ to $k$ do $C_i := P_i$.Cand$(Q_i)$
    $R := V(C_1, \ldots, C_k)$
    if ($R =$ (FEEDBACK, $i$, $CE$)) then
        $P_i := P_i$.LearnCE($CE$)
    else return $R$
```

Fig. 1 Algorithm for overall LAG procedure

5.2 Oracle, learner, and checker instantiations

We now describe various implementations of oracles, learners and checkers. We start with the notion of an oracle for weakest assumptions.

Oracle for weakest assumption Let L_1, L_S be any languages and Σ be any alphabet. We write $Q(L_1, L_S, \Sigma)$ to denote the oracle such that $Q(L_1, L_S, \Sigma) \models \overline{(L_1 \parallel \overline{L_S}) \downharpoonleft \Sigma}$. $Q(L_1, L_S, \Sigma)$ is typically implemented via model checking since, by Theorems 3 and 5, $Q(L_1, L_S, \Sigma)(u) = 1 \iff u \in \Sigma^\infty \wedge L_1 \parallel \{u\} \preccurlyeq L_S$.

Learner instantiations In general, a learner $P(\mathbf{L})$ is derived from an active learning algorithm $\mathbf{L}$ as follows: $P(\mathbf{L}) =$ (**Cand**, **LearnCE**) s.t. **Cand** $=$ part of $\mathbf{L}$ that constructs a candidate using membership queries, and **LearnCE** $=$ part of $\mathbf{L}$ that learns from a counterexample to a candidate query.

Non-circular checker Let $\mathcal{A}$ be a type of automata, and L_1, L_2 and L_S be any languages. Then, $V_{NC}(L_1, L_2, L_S)$ is the checker of type $(\mathcal{A}, 1)$ defined in Fig. 2. Note that $V_{NC}(L_1, L_2, L_S)$ is based on the **AG-NC** proof rule. The following proposition about $V_{NC}(L_1, L_2, L_S)$ will be used later.

```
Checker: $V_{NC}(L_1, L_2, L_S)$
Input: $A$: $\mathcal{A}$
if $(L_1 \parallel \mathcal{L}(A)) \preccurlyeq L_S$ then
    if $L_2 \preccurlyeq \mathcal{L}(A)$ then return SUCCESS
    else
        let $w$ be a CEX to $L_2 \preccurlyeq \mathcal{L}(A)$
        if $L_1 \parallel \{w\} \preccurlyeq L_S$ then
            return (FEEDBACK, $1, w \downharpoonleft \Sigma(A)$)
        else
            let $w'$ be a CEX to $L_1 \parallel \{w\} \preccurlyeq L_S$
            return (FAILURE, $w'$)
else
    let $w$ be a CEX to $(L_1 \parallel \mathcal{L}(A)) \preccurlyeq L_S$
    return (FEEDBACK, $1, w \downharpoonleft \Sigma(A)$)
```

Fig. 2 V_{NC}—a checker based on **AG-NC**

```
Checker: $V_C(L_1, L_2, L_S)$
Input: $A_1, A_2 : \mathcal{A}$
for $i = 1, 2$ do
    if $L_i \parallel \mathcal{L}(A_i) \not\preccurlyeq L_S$ then
        let $w$ be a CEX to $L_i \parallel \mathcal{L}(A_i) \preccurlyeq L_S$
        return (FEEDBACK, $i, w \downharpoonright \Sigma_A$)
if $\overline{\mathcal{L}(A_1)} \parallel \overline{\mathcal{L}(A_2)} \preccurlyeq L_S$ then return SUCCESS
else
    let $w$ be a CEX to $\overline{\mathcal{L}(A_1)} \parallel \overline{\mathcal{L}(A_2)} \preccurlyeq L_S$
    for $i = 1, 2$ do
        if $L_i \parallel \{w\} \preccurlyeq L_S$ then
            return (FEEDBACK, $i, w \downharpoonright \Sigma_A$)
        else let $w_i$ be a CEX to $L_i \parallel \{w\} \preccurlyeq L_S$
    pick $w' \in \{w_1\} \parallel \{w_2\}$
    return (FAILURE, $w'$)
```

Fig. 3 V_C—a checker based on **AG-C**

Proposition 1 *If $V_{NC}(L_1, L_2, L_S)(A)$ returns* **SUCCESS**, *then $L_1 \parallel L_2 \preccurlyeq L_S$. Otherwise, if $V_{NC}(L_1, L_2, L_S)(A)$ returns* (**FAILURE**, *CE*)*, then CE is a valid counterexample to $L_1 \parallel L_2 \preccurlyeq L_S$. Finally, if $V_{NC}(L_1, L_2, L_S)(A)$ returns* (**FEEDBACK**, 1, *CE*)*, then $CE \in \mathcal{L}(A) \ominus \overline{(L_1 \parallel \overline{L_S})} \downharpoonright \Sigma$.*

Circular checker Let $\mathcal{A}$ be a type of automata, and L_1, L_2 and L_S be any languages. Then $V_C(L_1, L_2, L_S)$ is the checker of type $(\mathcal{A}, 2)$ defined in Fig. 3. Note that $V_C(L_1, L_2, L_S)$ is based on the **AG-C** proof rule. The following proposition about $V_C(L_1, L_2, L_S)$ will be used later.

Proposition 2 *If $V_C(L_1, L_2, L_S)(A_1, A_2)$ returns* **SUCCESS**, *then $L_1 \parallel L_2 \preccurlyeq L_S$. Otherwise, if $V_C(L_1, L_2, L_S)(A_1, A_2)$ returns* (**FAILURE**, *CE*)*, then CE is a valid counterexample to $L_1 \parallel L_2 \preccurlyeq L_S$. Finally, if $V_C(L_1, L_2, L_S)(A_1, A_2)$ returns* (**FEEDBACK**, *i*, *CE*)*, then*

$$CE \in \mathcal{L}(A_i) \ominus \overline{(L_i \parallel \overline{L_S})} \downharpoonright \Sigma.$$

5.3 LAG instantiations

In this section, we present several instantiations of LAG for checking $L_1 \parallel L_2 \preccurlyeq L_S$. Our approach extends to systems with finitely many components, as for example in [9,3].

Existing work as LAG instances: regular trace containment. Table 1 instantiates LAG for existing learning-based algorithms for AG reasoning. The first row corresponds to the work of Cobleigh et al. [9]; its termination and correctness follow from Theorem 7, Proposition 1, and the fact that (P_1, Q_1) learns the language $\overline{(L_1 \parallel \overline{L_S})} \downharpoonright \Sigma$. The second row corresponds to Barringer et al. [3]; its termination and correctness follow from Theorem 7, Proposition 2, and the fact that (P_i, Q_i) learns $\overline{(L_i \parallel \overline{L_S})} \downharpoonright \Sigma$ for $i \in \{1, 2\}$.

New contribution: learning infinite behavior. Let $\mathbf{L}^\omega$ be any active learning algorithm for ω-regular languages (e.g., $\mathbf{L}^\$$). Since **AG-NC** is incomplete for ω-regular languages, $\mathbf{L}^\omega$ is not applicable directly in this context. On the other hand, both **AG-NC** and **AG-C** are sound and complete for ∞-regular languages. Therefore, a learning algorithm for ∞-regular languages yields LAG instances for systems with infinite behavior. We now present two such algorithms. The first (see Theorem 8 (a)) uses $\mathbf{L}^\omega$ only, but augments the assumption alphabet. The second (see Theorem 8(b)) combines $\mathbf{L}^\omega$ and $\mathbf{L}^*$, but leaves the assumption alphabet unchanged. We present both schemes since neither is objectively superior.

Theorem 8 *We can learn a ∞-regular language U using a MAT for U in two ways: (a) using only $\mathbf{L}^\omega$ but with alphabet augmentation, and (b) without alphabet augmentation, but using both $\mathbf{L}^*$and $\mathbf{L}^\omega$.*

Proof Part (a): Let Σ be the alphabet of U. We use $\mathbf{L}^\omega$ to learn an ω-regular language U' over the alphabet $\Sigma' = \Sigma \cup \{\tau\}$ such that $U' \downharpoonright \Sigma = U$, and $\tau \notin \Sigma$. Let $U' = U \cdot \tau^\omega$. We assume that the MAT X for U accepts candidate queries of the form $(M_1, M_2) \in$ DFA × BA, and returns "Yes" if

Table 1 Existing learning-based AG algorithms as instances of LAG; $\Sigma_{NC} = (\Sigma(L_1) \cup \Sigma(L_S)) \cap \Sigma(L_2)$; $\Sigma_C = (\Sigma(L_1) \cap \Sigma(L_2)) \cup \Sigma(L_S)$; $\mathbf{L}$ is a learning algorithm from Theorem 8

Conformance	Rule	$\mathcal{A}$	Learner(s)	Oracle(s)	Checker
Regular trace containment	**AG-NC** [9]	DFA	$P_1 = P(\mathbf{L}^*)$	$Q_1 = Q(L_1, L_S, \Sigma_{NC})$	$V_{NC}(L_1, L_2, L_S)$
Regular trace containment	**AG-C** [3]	DFA	$P_1 = P_2 = P(\mathbf{L}^*)$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$
∞-regular trace containment	**AG-NC**	DFA × BA	$P_1 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_{NC})$	$V_{NC}(L_1, L_2, L_S)$
∞-regular trace containment	**AG-C**	DFA × BA	$P_1 = P_2 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$
ω-regular trace containment	**AG-NC**	DFA × BA	$P_1 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_{NC})$	$V_{NC}(L_1, L_2, L_S)$
ω-regular trace containment	**AG-C**	BA	$P_1 = P_2 = P(\mathbf{L}^\omega)$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$

$U = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$, and a *CE* otherwise. Then, a MAT for U' is implemented using X as follows: (i) **Membership**: $u \in U'$ iff $u \in \Sigma^\infty \cdot \tau^\omega \wedge u \downharpoonright \Sigma \in U$, where $u \downharpoonright \Sigma \in U$ is decided using X; (ii) **Candidate** with C': If $\mathcal{L}(C') \not\subseteq \Sigma^\infty \cdot \tau^\omega$, return $CE' \in \mathcal{L}(C') \setminus \Sigma^\infty \cdot \tau^\omega$. Otherwise, make a candidate query to X with (M_1, M_2) such that $\mathcal{L}(M_1) = *(C' \downharpoonright \Sigma)$ and $\mathcal{L}(M_2) = \omega(C' \downharpoonright \Sigma)$, and turn any *CE* to $CE' = CE \cdot \tau^\omega$.

Part (b): We use $\mathbf{L}^*$ to learn $*(U)$ and $\mathbf{L}^\omega$ to learn $\omega(U)$. We assume that the MAT X for U accepts candidate queries of the form $(M_1, M_2) \in$ DFA × BA, and returns "Yes" if $U = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$, and a *CE* otherwise. We run $\mathbf{L}^*$ and $\mathbf{L}^\omega$ concurrently, and iterate the two next steps: (1) answer membership queries with X until we get candidates M_1 and M_2 from $\mathbf{L}^*$ and $\mathbf{L}^\omega$ respectively; (2) make candidate query (M_1, M_2) to X; return any finite (infinite) *CE* back to $\mathbf{L}^*$ ($\mathbf{L}^\omega$); repeat from Step 1. □

LAG instances for ∞-regular trace containment Suppose that L_1, L_2 and L_S are ∞-regular and we wish to verify $L_1 \parallel L_2 \preccurlyeq L_S$. The third row of Table 1 show how to instantiate LAG to solve this problem using **AG-NC**. This instance of LAG terminates with the correct result due to Theorem 7, Proposition 1, and the fact that (P_1, Q_1) learns $\overline{(L_1 \parallel \overline{L_S})} \downharpoonright \Sigma$. The fourth row of Table 1 show how to instantiate LAG to solve this problem using **AG-C**. This instance of LAG terminates correctly due to Theorem 7, Proposition 2, and because (P_i, Q_i) learns $\overline{(L_i \parallel \overline{L_S})} \downharpoonright \Sigma$ for $i \in \{1, 2\}$.

LAG instances for ω-regular trace containment. Suppose that L_1, L_2 and L_S are ω-regular and we wish to check $L_1 \parallel L_2 \preccurlyeq L_S$. When using **AG-NC**, restricting assumptions to ω-regular languages is incomplete (cf. Theorem 2). Hence, the situation is the same as for ∞-regular languages (cf. row 5 of Table 1). When using **AG-C**, restricting assumptions to be ω-regular is complete (cf. Theorem 6). Hence, we use $\mathbf{L}^\omega$ without augmenting the assumption alphabet, as summarized in row 6 of Table 1. This is a specific benefit of the restriction to ω-regular languages. This instance terminates with the correct result due to Theorem 7, Proposition 2, and because (P_i, Q_i) learns $\overline{(L_i \parallel \overline{L_S})} \downharpoonright \Sigma$ for $i \in \{1, 2\}$.

6 Related work

Automated AG reasoning with automata-based learning was pioneered by Cobleigh et al. [9] for checking safety properties of finite state systems. In this context, Barringer et al. [3] investigate the soundness and completeness of a number of decomposition proof rules, and Wang [23] proposed a framework for automatic derivation of sound decomposition rules.

Here, we extend the AG reasoning paradigm to arbitrary ω-regular properties (i.e., both safety and liveness) using both non-circular and circular rules.

The idea behind (particular instances of) Theorem 5 is used implicitly in almost all existing work on automated assume-guarantee reasoning [5,6,9]. However, we are not aware of an explicit closed-form treatment of the weakest assumption in a general setting such as ours.

The learning-based automated AG reasoning paradigm has been extended to check simulation [7] and deadlock [5]. Alur et al. [1], and Sinha et al. [21], have investigated symbolic and lazy SAT-based implementations, respectively. Tsay and Wang [22] show that verification of safety properties of ∞-regular systems is reducible the standard AG framework. In contrast, our focus is on the verification of arbitrary ω-regular properties of ω-regular systems.

For safety properties, further optimizations to the learning algorithm that reduce the number of queries [6], and gradually refine the alphabet of the assumption [11] have been proposed. We believe that these optimizations are also applicable in the context of ω-regular properties.

7 Conclusion

In summary, we present a very general formalization, called LAG, of the learning-based automated AG paradigm. We instantiate LAG to verify ω-regular properties of reactive systems with ω-regular behavior. We also show how existing approaches for automated AG reasoning are special instances of LAG. In addition, we prove the soundness and completeness of circular and non-circular AG proof rules in the context of ω-regular languages. Recently, techniques to reduce the number of queries [6], and refine the assumption alphabet [11], have been proposed in the context of using automated AG to verify safety properties. We believe that these techniques are applicable for ω-regular-properties as well.

References

1. Alur R, Madhusudan P, Nam W (2005) Symbolic compositional verification by learning assumptions. In: Etessami K, Rajamani SK (eds) Proceedings of the 17th international conference on computer aided verification (CAV '05), New York, July 6–10, 2005. Lecture notes in computer science, vol 3576. Springer, Edinburgh, pp 548–562
2. Angluin D (1987) Learning regular sets from queries and counterexamples. Inform Comput 75(2):87–106
3. Barringer H, Giannakopoulou D, Păsăreanu CS (2003) Proof rules for automated compositional verification through learning. In: Proceedings of the second workshop on specification and verification of component based systems (SAVCBS '03), Helsinki, September 1–2, 2003. Iowa State University, Ames, pp 14–21
4. Calbrix H, Nivat M, Podelski A (1993) Ultimately periodic words of rational ω-languages. In: Brookes SD, Main MG, Melton A,

Mislove MW, Schmidt DA (eds) Proceedings of the 9th international conference on mathematical foundations of programming semantics (MFPS '93). Lecture notes in computer science, vol 802. Springer, New Orleans, pp 554–566

5. Chaki S, Sinha N (2006) Assume-guarantee reasoning for deadlock. In: Proceedings of the 6th international conference on formal methods in computer-aided design (FMCAD '06), San Jose, November 12–16, 2006. IEEE Computer Society, Washington, DC, pp 134–144
6. Chaki S, Strichman O (2007) Optimized L* for assume-guarantee reasoning. In: Grumberg O, Huth M (eds) Proceedings of the 13th international conference on tools and algorithms for the construction and analysis of systems (TACAS '07), Braga, March 24–April 1, 2007. Lecture notes in computer science, vol 4424. Springer, New York, pp 276–291
7. Chaki S, Clarke EM, Sinha N, Thati P (2005) Automated assume-guarantee reasoning for simulation conformance. In: Etessami K, Rajamani SK (eds) Proceedings of the 17th international conference on computer aided verification (CAV '05). Lecture notes in computer science, vol 3576. Springer, Edinburgh, pp 534–547
8. Clarke E, Long D, McMillan K (1989) Compositional model checking. In: Proceedings of the 4th annual IEEE symposium on logic in computer science (LICS '89), Pacific Grove, June 5–8, 1989. IEEE Computer Society, Washington, DC, pp 353–362
9. Cobleigh JM, Giannakopoulou D, Păsăreanu CS (2003) Learning assumptions for compositional verification. In: Garavel H, Hatcliff J (eds) Proceedings of the 9th international conference on tools and algorithms for the construction and analysis of systems (TACAS '03), Warsaw, April 7–11, 2003. Lecture notes in computer science, vol 2619. Springer, New York, pp 331–346
10. Farzan A, Chen Y, Clarke E, Tsan Y, Wang B (2008) Extending automated compositional verification to the full class of omega-regular languages. In: Ramakrishnan CR, Rehof J (eds) Proceedings of the 14th international conference on tools and algorithms for the construction and analysis of systems (TACAS '08), Budapest. Lecture notes in computer science, vol 4963. Springer, Berlin, pp 2–17
11. Gheorghiu M, Giannakopoulou D, Păsăreanu CS (2007) Refining interface alphabets for compositional verification. In: Grumberg O, Huth M (eds) Proceedings of the 13th international conference on tools and algorithms for the construction and analysis of systems (TACAS '07), Braga, March 24–April 1, 2007. Lecture notes in computer science, vol 4424. Springer, New York, pp 292–307
12. Gold EM (1967) Language identification in the limit. Inform Control 10(5):447–474
13. Grumberg O, Long DE (1994) Model checking and modular verification. ACM Trans Program Lang Syst (TOPLAS) 16(3):843–871
14. Henzinger TA, Qadeer S, Rajamani SK (2000) Decomposing refinement proofs using assume-guarantee reasoning. In: Proceedings of the 2000 international conference on computer-aided design (ICCAD '00), San Jose. IEEE Computer Society, Washington, DC, pp 245–252
15. Hoare CAR (1985) Communicating sequential processes. Prentice Hall, London
16. Jones CB (1983) Specification and design of (Parallel) programs. In: Mason REA (ed) Proceedings of the 9th IFIP World Congress, Paris, France, vol 83. Information processing, pp 321–332
17. Maler O, Pnueli A (1995) On the learnability of infinitary regular sets. Inform Comput 118(2):316–326
18. Misra J, Chandy KM (1981) Proofs of networks of processes. IEEE Trans Softw Eng (TSE) 7(4):417–426
19. Pnueli A (1985) In transition from global to modular temporal reasoning about programs. Logics Models Concurrent Syst 13:123–144
20. Rivest RL, Schapire RE (1993) Inference of finite automata using homing sequences. Inform Comput 103(2):299–347
21. Sinha N, Clarke E (2007) SAT-based compositional verification using lazy learning. In: Damm W, Hermanns H (eds) Proceedings of the 19th international conference on computer aided verification (CAV '07), Berlin, July 3–7, 2007. Lecture notes in computer science, vol 4590. Springer, New York, pp 39–54
22. Tsay YK, Wang BY (2008) Automated compositional reasoning of intuitionistically closed regular properties. In: Ibarra OH, Ravikumar B (eds) Proceedings of the 13th international conference on implementation and applications of automata (CIAA '08), San Francisco, July 21–24, 2008. Lecture notes in computer science, vol 5148. Springer, New York, pp 36–45
23. Wang BY (2007) Automatic derivation of compositional rules in automated compositional reasoning. In: Caires L, Vasconcelos VT (eds) Proceedings of the 18th international conference on concurrency theory (CONCUR '07), Lisbon, September 3–8, 2007. Lecture notes in computer science, vol 4703. Springer, New York, pp 303–316