

# Automated Assume-Guarantee Reasoning for Omega-Regular Systems and Specifications

Sagar Chaki · Arie Gurfinkel

Received: date / Accepted: date

**Abstract** We develop a learning-based automated Assume-Guarantee (AG) reasoning framework for verifying  $\omega$ -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  $\infty$ -regular systems, that combine finite and infinite behaviors. We show that for  $\infty$ -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  $\infty$ -regular, and  $\omega$ -regular, systems using both **AG-NC** and **AG-C** as proof rules.

## 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 [19, 14]. 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

in order 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 [18, 16], 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:

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

where  $M_1, M_2, A$  and  $S$  are finite automata,  $\parallel$  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 [5, 1, 21]. However, the question of whether this paradigm is applicable to verifying  $\omega$ -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  $\omega$ -regular languages has been proposed by Farzan et al. [10]. However, to our knowledge, the AG proof rules have not been extended to

$\omega$ -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.,  $\omega$ -regular). This is surprising and interesting. In contrast, we show that **AG-C** is both sound and complete for  $\omega$ -regular languages. *Second*, we extend our AG proof rules to systems and specifications expressible in  $\infty$ -regular languages (i.e., unions of regular and  $\omega$ -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  $\infty$ -regular languages – one using a learning algorithm for  $\omega$ -regular languages (see Theorem 8(a)) with an augmented alphabet, and another combining a learning algorithm for  $\omega$ -regular languages with **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  $\infty$ -regular and  $\omega$ -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 Section 2. In Section 3, we review our model of concurrency. In Section 4, we study the soundness and completeness of AG rules, and present our LAG framework in Section 5. We review the related work in Section 6, and conclude in Section 7.

## 2 Preliminaries

We write  $\Sigma^*$  and  $\Sigma^\omega$  for the set of all finite and infinite words over  $\Sigma$ , respectively, and write  $\Sigma^\infty$  for  $\Sigma^* \cup \Sigma^\omega$ . We use the standard notation of regular expressions:  $\lambda$  for empty word,  $a \cdot b$  for concatenation,  $a^*$ ,  $a^+$ , and  $a^\omega$  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  $(L, \Sigma)$  such that  $\Sigma$  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 sim-

plicity, we often refer to a language as  $L$  and mention  $\Sigma$  separately. For instance, we write “language  $L$  over alphabet  $\Sigma$ ” to mean the language  $(L, \Sigma)$ , 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  $\bar{L}$ , is defined as:  $\bar{L} = \Sigma(L)^\infty \setminus L$ . A finitary language ( $\Sigma^*$ -language) is a subset of  $\Sigma^*$ . An infinitary language ( $\Sigma^\omega$ -language) is a subset of  $\Sigma^\omega$ . 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  $\Sigma(\bar{L}) = \Sigma(*(L)) = \Sigma(\omega(L)) = \Sigma(L)$ .

**Transition Systems.** A labeled transition system (LTS) is a 4-tuple  $M = (S, \Sigma, Init, \delta)$ , where  $S$  is a finite set of states,  $\Sigma$  is an alphabet,  $Init \subseteq S$  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  $\Sigma$ .  $M$  is deterministic if  $|Init| \leq 1$ , and  $\forall s \in S. \forall \alpha \in \Sigma. |\{s' \mid s \xrightarrow{\alpha} s'\}| \leq 1$ . A run  $r$  over a word  $w = \alpha_0, \alpha_1, \dots \in \Sigma(M)^\infty$  is a sequence of states  $s_0, s_1, \dots$ , such that  $\forall i \geq 0. s_i \xrightarrow{\alpha_i} s_{i+1}$ . We write  $First(r)$ ,  $Last(r)$ , and  $Inf(r)$  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  $Run(w, M)$  for the set of runs of  $w$  on  $M$ .

**Automata.** A Finite Automaton (FA) is a 5-tuple  $A = (S, \Sigma, Init, \delta, F)$ , where  $(S, \Sigma, Init, \delta)$  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  $First(r) \in Init \wedge Last(r) \in F$ . A Büchi Automaton (BA) is a 5-tuple  $B = (S, \Sigma, Init, \delta, F)$ , where  $(S, \Sigma, Init, \delta)$  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  $First(r) \in Init \wedge Inf(r) \cap F \neq \emptyset$ . A BA or FA is deterministic if its underlying LTS is deterministic.

**Regularity.** A language is regular ( $\omega$ -regular) iff it is accepted by a FA (BA). A language  $L \subseteq \Sigma^\infty$  is  $\infty$ -regular iff  $*(L)$  is regular and  $\omega(L)$  is  $\omega$ -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  $\Sigma$ . 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  $\omega$ -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  $\omega$ -regular languages.

Farzan et al. [10] show how to learn a Büchi automaton for an  $\omega$ -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  $\omega$ -regular languages. In particular,  $\mathbf{L}^{\$}$  is an existing candidate.

### 3 Model of Concurrency

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

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

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

A process is modeled by a language of all of its behaviors (or computations). *Parallel composition* ( $\parallel$ ) of

two processes/languages synchronizes on common actions while executing local actions asynchronously. For languages  $(L_1, \Sigma_1)$  and  $(L_2, \Sigma_2)$ ,  $L_1 \parallel L_2$  is the language over  $\Sigma_1 \cup \Sigma_2$  defined as follows:

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

Intuitively,  $L_1 \parallel L_2$  consists of all permutations of words from  $L_1$  and  $L_2$  that have a common synchronization sequence. For example,  $(b^* \cdot a \cdot b^*) \parallel (c^* \cdot a \cdot c^*)$  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  $\Sigma^*$ ,  $\Sigma^\omega$ , and  $\Sigma^\infty$  languages are all closed under parallel composition. The following theorem illustrates some other useful properties of parallel composition.

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

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  $L_1 \preceq L_2$ , if  $L_1 \downarrow \Sigma(L_2) \subseteq L_2$ . Thus,  $L_1 \preceq L_2$  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  $L_M \preceq L_S$ .

### 4 Proof Rules for Assume-Guarantee Reasoning

In this section, we study the applicability of a non-circular and a circular AG rule to proving 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  $\Sigma^*$ ) behaviors by Barringer et al. [3]. In Section 4.1, we show that the non-circular AG rule is sound for both  $\Sigma^\infty$  and  $\Sigma^\omega$  behaviors. However, it is complete only when the assumptions are allowed to combine *both* finite and infinite behaviors (i.e., in  $\Sigma^\infty$ ). In Section 4.2, we show that the circular AG rule is sound and complete for  $\Sigma^\omega$  and  $\Sigma^\infty$  behaviors.

#### 4.1 Non-Circular Assume-Guarantee Rule

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

$$\frac{(L_1 \parallel L_A) \preceq L_S \quad L_2 \preceq L_A}{(L_1 \parallel L_2) \preceq L_S}$$

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  $\Sigma^*$ -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  $\Sigma_A$  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  $\Sigma^\omega$ -languages:

**Theorem 2** *There exists  $L_1, L_2, L_S \subseteq \Sigma^\omega$  such that  $(L_1 \parallel L_2) \preceq L_S$ , but there does not exist 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:

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

The conclusion of **AG-NC** rule is satisfied since  $(L_1 \parallel L_2) \downarrow \Sigma_S = (a + b)^* b^\omega = L_S$ . The alphabet  $\Sigma_A$  of  $L_A$  is  $(\Sigma_1 \cup \Sigma_S) \cap \Sigma_2 = \{a\}$ . Since  $L_A \subseteq \Sigma_A^\omega$ , it can only be  $a^\omega$  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.  $\square$

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

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

$$L_1 \preceq L_2 \Leftrightarrow \omega(L_1 \downarrow \Sigma(L_2)) \subseteq L_2.$$

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:

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

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

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

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

*Remark 2* **AG-NC** is complete if the alphabet  $\Sigma_A$  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  $\Sigma^\omega$  because  $\Sigma^\omega$  is not closed under projection. However, we show that the rule is complete for  $\Sigma^\infty$  – the smallest projection-closed extension of  $\Sigma^\omega$ . We first show that for any languages  $L_1$  and  $L_S$ , there always exists a unique weakest assumption  $L_A$ , such that  $L_1 \parallel L_A \preceq L_S$ .

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

*Proof* Let us write  $\Sigma_1, \Sigma_S$  and  $\Sigma_{1S}$  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  $w \downarrow \Sigma_A \in L_A$ . This implies that  $w \downarrow \Sigma_S \in (L_1 \parallel \{w \downarrow \Sigma_A\}) \downarrow \Sigma_S \subseteq L_S$ . Since  $w$  is any word in  $L_1 \parallel L_A$ , we have  $L_1 \parallel L_A \preceq L_S$ . To show that  $L_A$  is the weakest assumption, let  $L'_A \subseteq \Sigma_A^\infty$  be any language such that  $L_1 \parallel L'_A \preceq L_S$  and let  $w$  be any word in  $L'_A$ . Then,  $(L_1 \parallel \{w\}) \subseteq (L_1 \parallel L'_A) \preceq L_S$ . But this implies that  $w \in L_A$ , and, therefore,  $L'_A \subseteq L_A$ .  $\square$

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

**Theorem 4** *Let  $L_1, L_2, L_S$ , and  $L_A$  be in  $\Sigma^\infty$ . 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  $L_1 \parallel L_2 \preceq L_S$ , and let  $L_A$  be the weakest assumption such that  $L_1 \parallel L_A \preceq L_S$ . By Theorem 3,  $L_A$  is well-defined and satisfies the first premise of **AG-NC**. The second premise holds because  $L_2 \downarrow \Sigma_A \subseteq \Sigma_A^\infty$ , and  $L_A$  is the weakest  $\Sigma_A^\infty$  assumption (see Theorem 3).  $\square$

Theorem 4 implies that **AG-NC** is sound for any fragment of  $\Sigma^\infty$ . 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  $\Sigma_A$  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  $L_1 \parallel L_A \preceq L_S$  iff  $L_A = \overline{(L_1 \parallel \overline{L_S})} \downarrow \Sigma_A$ .*

*Proof* Let us write  $\Sigma_1$ ,  $\Sigma_S$  and  $\Sigma_{1S}$  to mean  $\Sigma(L_1)$ ,  $\Sigma(L_S)$  and  $\Sigma(L_1) \cup \Sigma(L_S)$ , respectively. For any  $w \in \Sigma_A^\infty$ :

$$\begin{aligned} & w \in \overline{(L_1 \parallel \overline{L_S})} \downarrow \Sigma_A \\ \text{iff } \forall w' \in \Sigma_{1S}^\infty \cdot \{w'\} \preceq \{w\} & \implies w' \notin (L_1 \parallel \overline{L_S}) \\ \text{iff } \forall w' \in \Sigma_{1S}^\infty \cdot \{w'\} \preceq \{w\} & \implies \\ & (\{w'\} \not\preceq L_1 \vee \{w'\} \preceq L_S) \\ \text{iff } \forall w' \in \Sigma_{1S}^\infty \cdot (\{w'\} \preceq \{w\} \wedge \{w'\} \preceq L_1) & \implies \\ & \{w'\} \preceq L_S \\ \text{iff } \forall w' \in \Sigma_{1S}^\infty \cdot (\{w'\} \preceq (L_1 \parallel \{w\})) & \implies \{w'\} \preceq L_S \\ \text{iff } L_1 \parallel \{w\} \preceq L_S & \end{aligned}$$

Together with Theorem 3, this completes the proof.  $\square$

Theorem 5 implies **AG-NC** is complete for any class of languages closed under complementation and projection, e.g., regular and  $\infty$ -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,  $\omega$ -regular and  $\infty$ -regular languages.

## 4.2 Circular Assume-Guarantee Rule

The Circular Assume-Guarantee proof rule (**AG-C** for short) is stated as follows:

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

where  $L_1$ ,  $L_2$ , and  $L_S$  are languages over alphabets  $\Sigma_1$ ,  $\Sigma_2$ ,  $\Sigma_S$ , 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  $\Sigma^*$ -languages. Note that in comparison with **AG-NC**, there are two assumptions  $L_{A1}$  and  $L_{A2}$  over a larger alphabet  $\Sigma_A$ . Informally, the rule is sound for the following reason. Let  $w$  be a word in  $L_1 \parallel L_2$ , and  $u = w \downarrow \Sigma_A$ . Then  $u \in L_{A1}$ , or  $u \in L_{A2}$ , or  $u \in \overline{L_{A1}} \cup \overline{L_{A2}} = \overline{(\overline{L_{A1}} \parallel \overline{L_{A2}})}$ . If  $u \in L_{A1}$  then the first premise implies that  $\{w\} \preceq L_1 \parallel \{u\} \preceq L_S$ ; if  $u \in L_{A2}$  then the second premise implies that  $\{w\} \preceq L_2 \parallel \{u\} \preceq L_S$ ; otherwise, the third premise implies that  $\{w\} \preceq \{u\} \preceq L_S$ .

*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  $L_1 \parallel L_2 \preceq L_S$ . We show that no  $L_{A1}$  and  $L_{A2}$  can satisfy the three premises of **AG-C**. Premise 1  $\implies b \notin L_{A1} \implies b \in \overline{L_{A1}}$ . Similarly, premise 2  $\implies a \notin L_{A2} \implies a \in \overline{L_{A2}}$ . But then  $ab \in \overline{L_{A1}} \parallel \overline{L_{A2}}$ , violating premise 3.

In this section, we show that **AG-C** is sound and complete for both  $\Sigma^\omega$  and  $\Sigma^\infty$  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  $\Sigma^\omega$  languages as defined in the proof of Theorem 2. In this case, the alphabet  $\Sigma_A$  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  $\Sigma^\omega$  or  $\Sigma^\infty$ . 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  $L_1 \parallel L_2 \preceq L_S$ . Let  $L_{A1}$  and  $L_{A2}$  be the weakest assumptions such that  $L_1 \parallel L_{A1} \preceq L_S$ , and  $L_2 \parallel L_{A2} \preceq L_S$ , 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,  $(\overline{L_{A1}} \parallel \overline{L_{A2}}) = \overline{(L_{A1} \cap L_{A2})}$ . Assume that  $(\overline{L_{A1}} \cap \overline{L_{A2}}) \not\preceq L_S$ . Then, there exists a word  $w \in (\overline{L_{A1}} \parallel \overline{L_{A2}})$  such that  $w \notin L_{A1}$ , and  $w \notin L_{A2}$ , and  $w \downarrow \Sigma_S \notin L_S$ . By the definition of weakest assumption (see Theorem 3),  $L_1 \parallel \{w\} \not\preceq L_S$  and  $L_2 \parallel \{w\} \not\preceq L_S$ . Pick any  $w_1 \in L_1 \parallel \{w\}$  and  $w_2 \in L_2 \parallel \{w\}$ . Let  $w'_1 = w_1 \downarrow \Sigma_1$  and  $w'_2 = w_2 \downarrow \Sigma_2$ . We know that  $\{w'_1\} \parallel \{w'_2\} \subseteq L_1 \parallel L_2$ . Also,  $w \in (\{w'_1\} \parallel \{w'_2\}) \downarrow \Sigma_A$ . Now since  $\{w'_1\} \parallel \{w'_2\} \subseteq L_1 \parallel L_2$ , we have  $w \in (L_1 \parallel L_2) \downarrow \Sigma_A$ . Since  $\Sigma_S \subseteq \Sigma_A$ ,  $w \downarrow \Sigma_S \in (L_1 \parallel L_2) \downarrow \Sigma_S$ . But  $w \downarrow \Sigma_S \notin L_S$ , which contradicts  $L_1 \parallel L_2 \preceq L_S$ .  $\square$

Intuitively, **AG-C** is complete even for  $\omega$ -regular languages because, unlike **AG-NC**, assumptions always appear to the left of  $\preceq$  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  $(\infty, \omega)$ -regular if  $L_1$ ,  $L_2$ , and  $L_S$  are  $(\infty, \omega)$ -regular, respectively. Thus, **AG-C** is complete for  $(\infty, \omega)$ -regular languages. Since **AG-NC** is incomplete for

$\omega$ -regular languages, a learning algorithm for  $\omega$ -regular languages (such as  $\mathbf{L}^{\mathbb{S}}$ ) cannot be applied directly for AG reasoning for  $\omega$ -regular systems and specifications. In the next section, we overcome this challenge by developing automated AG algorithms for  $\infty$ -regular and  $\omega$ -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  $\Sigma$  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  $(\mathbf{Cand}, \mathbf{LearnCE})$  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  $P = (\mathbf{Cand}, \mathbf{LearnCE})$  we write  $P.\mathbf{Cand}$  and  $P.\mathbf{LearnCE}$  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  $P_0, P_1, \dots$  and an infinite sequence of counterexamples  $CE_1, CE_2, \dots$  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  $A_1, \dots, A_k$  of  $\mathcal{A}$  and returns either (i) **SUCCESS**, or (ii) a pair  $(\mathbf{FAILURE}, CE)$  such that  $CE \in \Sigma^\infty$ , or (iii) a triple  $(\mathbf{FEEDBACK}, i, CE)$  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  $(\mathcal{A}, k)$ .

### Input:

$P_1 \dots P_k : \mathbf{Learner}_{\mathcal{A}};$

$Q_1, \dots, Q_k : \mathbf{Oracle};$

$V : \mathbf{Checker}_{(\mathcal{A}, k)}$

### forever do

  for  $i = 1$  to  $k$  do  $C_i := P_i.\mathbf{Cand}(Q_i)$

$R := V(C_1, \dots, C_k)$

  if  $(R = (\mathbf{FEEDBACK}, i, CE))$  then

$P_i := P_i.\mathbf{LearnCE}(CE)$

  else return  $R$

Fig. 1 Algorithm for overall LAG procedure.

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)  $(\mathbf{FAILURE}, CE)$  where  $CE$  is a real counterexample; (iii)  $(\mathbf{FEEDBACK}, i, CE)$  where  $CE$  is a counterexample for the  $i$ -th 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  $C_1, \dots, C_k$  using the learners and oracles.
2. Invokes the checker with the candidates constructed in Step 1 above.
3. If the checker returns **SUCCESS** or  $(\mathbf{FAILURE}, CE)$ , 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  $U_1, \dots, U_k$  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  $V(C_1, \dots, C_k) = (\mathbf{FEEDBACK}, i, CE)$ , 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  $U_i$ .  $\square$

### 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  $\Sigma$  be any alphabet. We

---

```

Checker:  $V_{NC}(L_1, L_2, L_S)$ 


---


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

```

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

write  $Q(L_1, L_S, \Sigma)$  to denote the oracle such that  $Q(L_1, L_S, \Sigma) \models (L_1 \parallel \overline{L_S}) \downarrow \Sigma$ .  $Q(L_1, L_S, \Sigma)$  is typically implemented via model checking since, by Theorems 3 and 5,  $Q(L_1, L_S, \Sigma)(u) = 1 \iff u \in \Sigma^\infty \wedge L_1 \parallel \{u\} \preceq L_S$ .

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

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

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

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

---

```

Checker:  $V_C(L_1, L_2, L_S)$ 


---


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

```

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

### 5.3 LAG Instantiations

In this section, we present several instantiations of LAG for checking  $L_1 \parallel L_2 \preceq L_S$ . 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  $(L_1 \parallel \overline{L_S}) \downarrow \Sigma$ . 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  $(L_i \parallel \overline{L_S}) \downarrow \Sigma$  for  $i \in \{1, 2\}$ .

**New Contribution: Learning Infinite Behavior.** Let  $\mathbf{L}^\omega$  be any active learning algorithm for  $\omega$ -regular languages (e.g.,  $\mathbf{L}^s$ ). Since **AG-NC** is incomplete for  $\omega$ -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  $\infty$ -regular languages. Therefore, a learning algorithm for  $\infty$ -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  $\infty$ -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  $\Sigma$  be the alphabet of  $U$ . We use  $\mathbf{L}^\omega$  to learn an  $\omega$ -regular language  $U'$  over the alphabet

$\Sigma' = \Sigma \cup \{\tau\}$  such that  $U' \downarrow \Sigma = U$ , and  $\tau \notin \Sigma$ . Let  $U' = U \cdot \tau^\omega$ . We assume that the MAT  $X$  for  $U$  accepts candidate queries of the form  $(M_1, M_2) \in \text{DFA} \times \text{BA}$ , and returns “Yes” if  $U = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$ , and a  $CE$  otherwise. Then, a MAT for  $U'$  is implemented using  $X$  as follows: (i) **Membership**:  $u \in U'$  iff  $u \in \Sigma^\infty \cdot \tau^\omega \wedge u \downarrow \Sigma \in U$ , where  $u \downarrow \Sigma \in U$  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  $\mathcal{L}(M_1) = *(C' \downarrow \Sigma)$  and  $\mathcal{L}(M_2) = \omega(C' \downarrow \Sigma)$ , 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  $(M_1, M_2) \in \text{DFA} \times \text{BA}$ , 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.  $\square$

**LAG instances for  $\infty$ -regular Trace Containment.** Suppose that  $L_1, L_2$  and  $L_S$  are  $\infty$ -regular and we wish to verify  $L_1 \parallel L_2 \preceq L_S$ . 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  $(L_1 \parallel \overline{L_S}) \downarrow \Sigma$ . 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  $(L_i \parallel \overline{L_S}) \downarrow \Sigma$  for  $i \in \{1, 2\}$ .

**LAG instances for  $\omega$ -regular Trace Containment.** Suppose that  $L_1, L_2$  and  $L_S$  are  $\omega$ -regular and we wish to check  $L_1 \parallel L_2 \preceq L_S$ . When using **AG-NC**, restricting assumptions to  $\omega$ -regular languages is incomplete (cf. Theorem 2). Hence, the situation is the same as for  $\infty$ -regular languages (cf. row 5 of Table 1). When using **AG-C**, restricting assumptions to be  $\omega$ -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  $\omega$ -regular languages. This instance terminates with the correct result due to Theorem 7, Proposition 2, and because  $(P_i, Q_i)$  learns  $(L_i \parallel \overline{L_S}) \downarrow \Sigma$  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  $\omega$ -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 [9, 5, 6]. 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  $\infty$ -regular systems is reducible the standard AG framework. In contrast, our focus is on the verification of arbitrary  $\omega$ -regular-properties of  $\omega$ -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  $\omega$ -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  $\omega$ -regular properties of reactive systems with  $\omega$ -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  $\omega$ -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  $\omega$ -regular-properties as well.



Conformance	Rule	$\mathcal{A}$	Learner(s)	Oracle(s)	Checker
Regular Trace Containment	<b>AG-NC</b> [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	<b>AG-C</b> [3]	DFA	$P_1 = P_2 = P(\mathbf{L}^*)$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$
$\infty$ -regular Trace Containment	<b>AG-NC</b>	DFA $\times$ BA	$P_1 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_{NC})$	$V_{NC}(L_1, L_2, L_S)$
$\infty$ -regular Trace Containment	<b>AG-C</b>	DFA $\times$ BA	$P_1 = P_2 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$
$\omega$ -regular Trace Containment	<b>AG-NC</b>	DFA $\times$ BA	$P_1 = P(\mathbf{L})$	$Q_1 = Q(L_1, L_S, \Sigma_{NC})$	$V_{NC}(L_1, L_2, L_S)$
$\omega$ -regular Trace Containment	<b>AG-C</b>	BA	$P_1 = P_2 = P(\mathbf{L}^\omega)$	$Q_1 = Q(L_1, L_S, \Sigma_C)$ $Q_2 = Q(L_2, L_S, \Sigma_C)$	$V_C(L_1, L_2, L_S)$

**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.

## References

- 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), Springer-Verlag, Edinburgh, Scotland, July 6–10, 2005. New York, NY, Lecture Notes in Computer Science, vol 3576, pp 548–562
- Angluin D (1987) Learning Regular Sets from Queries and Counterexamples. *Information and Computation* 75(2):87–106
- Barringer H, Giannakopoulou D, Păsăreanu CS (2003) Proof Rules for Automated Compositional Verification through Learning. In: Proceedings of the 2nd Workshop on Specification and Verification of Component Based Systems (SAVCBS '03), Helsinki, Finland, September 1–2, 2003. Ames, Iowa: Iowa State University, pp 14–21
- Calbrix H, Nivat M, Podelski A (1993) Ultimately Periodic Words of Rational  $\omega$ -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), Springer-Verlag, New Orleans, LA, USA, Lecture Notes in Computer Science, vol 802, pp 554–566
- 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), IEEE Computer Society, San Jose, CA, November 12–16, 2006. Washington, DC, pp 134–144
- 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), Springer-Verlag, Braga, Portugal, March 24–April 1, 2007. New York, NY, Lecture Notes in Computer Science, vol 4424, pp 276–291
- 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), Springer-Verlag, Edinburgh, Scotland, Lecture Notes in Computer Science, vol 3576, pp 534–547
- 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), IEEE Computer Society, Pacific Grove, CA, June 5–8, 1989. Washington, DC, pp 353–362
- 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), Springer-Verlag, Warsaw, Poland, April 7–11, 2003. New York, NY, Lecture Notes in Computer Science, vol 2619, pp 331–346
- 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), Springer-Verlag, Budapest, Hungary, Lecture Notes in Computer Science, vol 4963, pp 2–17
- 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), Springer-Verlag, Braga, Portugal, March 24–April 1, 2007. New York, NY, Lecture Notes in Computer Science, vol 4424, pp 292–307
12. Gold EM (1967) Language Identification in the Limit. *Information and Control* 10(5):447–474
  13. Grumberg O, Long DE (1994) Model Checking and Modular Verification. *ACM Transactions on Programming Languages and System (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)*, IEEE Computer Society, San Jose, 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, Information Processing, vol 83*, pp 321–332
  17. Maler O, Pnueli A (1995) On the learnability of infinitary regular sets. *Information and Computation* 118(2):316–326
  18. Misra J, Chandy KM (1981) Proofs of Networks of Processes. *IEEE Transactions on Software Engineering (TSE)* 7(4):417–426
  19. Pnueli A (1985) In Transition from Global to Modular Temporal Reasoning About Programs. *Logics and Models of Concurrent Systems* 13:123–144
  20. Rivest RL, Schapire RE (1993) Inference of Finite Automata Using Homing Sequences. *Information and Computation* 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)*, Springer-Verlag, Berlin, Germany, July 3–7, 2007. New York, NY, Lecture Notes in Computer Science, vol 4590, 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)*, Springer-Verlag, San Francisco, California, USA, July 21–24, 2008. New York, NY, Lecture Notes in Computer Science, vol 5148, 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)*, Springer-Verlag, Lisbon, Portugal, September 3–8, 2007. New York, NY, Lecture Notes in Computer Science, vol 4703, pp 303–316