Learning Assumptions for Compositional Verification of Timed Systems

Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun, and Jin Song Dong

Abstract—Compositional techniques such as assume-guarantee reasoning (AGR) can help to alleviate the state space explosion problem associated with model checking. However, compositional verification is difficult to be automated, especially for timed systems, because constructing appropriate assumptions for AGR usually requires human creativity and experience. To automate compositional verification of timed systems, we propose a compositional verification framework using a learning algorithm for automatic construction of timed assumptions for AGR. We prove the correctness and termination of the proposed learning-based framework, and experimental results show that our method performs significantly better than traditional monolithic timed model checking.

Index Terms—automatic assume-guarantee reasoning, model checking, timed systems

1 Introduction

Model checking [10], [32] is one of the most successful formal verification techniques because it can be automatically applied if the following two inputs are given: a *system model* describing the system behavior and a *property* specifying what the system should satisfy. However, model checking suffers from the *state space explosion* problem [10], [32] because the number of states increases exponentially with the number of components involved.

To alleviate the state space explosion problem, *assume-guarantee reasoning* (AGR) [12], [17], [31], a well-known compositional technique, has been applied to model checking. The most common proof rule used in AGR is the following non-circular assume-guarantee (AG-NC) rule:

$$\begin{array}{rcl} M_1 \parallel A & \models & \varphi \\ M_2 & \models & A \\ \hline M_1 \parallel M_2 & \models & \varphi \end{array}$$

- *This article is a revised and substantially extended version of the paper presented at the 9th International Symposium on Automated Technology for Verification and Analysis (ATVA 2011) [25]. This work proposes a compositional framework for timed systems, which uses and guides the learning algorithm proposed in [25] to automatically generate timed assumptions needed by AGR. The tool paper [29] about the implementation of the proposed framework has been presented at the 18th International Symposium on Formal Methods (FM 2012).*

- *Shang-Wei Lin, the corresponding author, is with Temasek Laboratories, National University of Singapore. E-mail: tsllsw@nus.edu.sg*
- *Étienne André is with Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France. E-mail: Etienne.Andre@lipn.univ-paris13.fr*
- *Yang Liu is with the School of Computer Engineering, Nanyang Technological University. E-mail: yangliu@ntu.edu.sg*
- *Jun Sun is with Singapore University of Technology and Design. E-mail: sunjun@sutd.edu.sg*
- *Jin Song Dong is with the School of Computing, National University of Singapore. E-mail: dongjs@comp.nus.edu.sg*

- *This work was supported by the TRF project R394-000-063-23 and the seed project R394-000-068-232 in Temasek Laboratories at National University of Singapore, and partially supported by the CNRS STIC-Asie project "Compositional Analysis of Timed Systems."*

Given a system with two components modeled by M_1 and M_2 and a property φ, the AG-NC proof rule tells us that if M_1 can satisfy a property φ under an assumption A and M_2 can guarantee the assumption A, then we can conclude that $M_1 \parallel M_2$ satisfies φ. However, the assumption A in AGR usually requires nontrivial human creativity and experience. Thus, practical impact of AGR is limited if the assumption A is not automatically constructed.

Cobleigh et al. [13] proposed a framework that can generate assumptions for AGR automatically using the L* algorithm [5]. This framework is guaranteed to terminate when the verification problem $M_1 \parallel M_2 \models \varphi$ is either proved or disproved with a counterexample. To infer the assumption needed by AGR, the L* algorithm is not the only solution. Bobaru et al. [7] adopted the *abstraction-refinement* paradigm [11]. The assumption A is constructed as an abstraction of M_2. If $M_1 \parallel A \models \varphi$ holds, then $M_1 \parallel M_2 \models \varphi$ can be concluded. If $M_1 \parallel A \models \varphi$ does not hold, A is refined by the counterexample given by model checking until a conclusive result can be concluded.

However, these frameworks are only applicable to untimed systems. The demand for compositional model checking of timed systems is even greater than that of untimed systems because the state space explosion problem is graver in timed model checking. As a solution, we propose an automatic learning-based compositional verification framework for timed systems[1]. We focus on timed systems modeled by *event-recording automata* (ERAs) [3], which is a determinizable class of timed automata. ERAs are as powerful as timed transition systems [3], [19] and are sufficiently expressive to model many interesting timed systems. The proposed framework consists of a compositional verification flow based on the AG-NC proof rule and uses a

1. In [7], the comparison between the learning-based and abstraction-refinement-based approaches for generating untimed assumptions in AGR did not indicate a clear winner. Therefore, it would be interesting as well to study a similar abstraction-refinement-based approach in a timed setting.

learning algorithm to automatically generate timed assumptions for AGR. The verification flow is designed as a two-phase process. It generates untimed assumptions first, which guarantees the sequence of events on assumptions is correct. Then it refines untimed assumptions into timed ones, which guarantees that the occurrences of events on assumptions satisfy time constraints. We prove the *correctness* and *termination* of the learning-based compositional verification framework for timed systems. Experimental results show that the proposed framework performs significantly better than traditional monolithic timed model checking [3] that constructs the timed global state space based on zone abstraction. Our contributions can be summarized as follows.

- We propose a learning-based compositional verification framework for timed systems. To the best of our knowledge, this is the first work of fully automated compositional verification for timed systems.
- Our compositional verification framework is based on a novel algorithm that we proposed for learning ERAs. This algorithm is particularly efficient in the context of our framework where the models of the system components are available.
- We prove the correctness of the proposed framework and show that it is always terminating.
- We implement the proposed framework as a self-contained toolkit and evaluate its scalability, usefulness, and reliability via a variety of systems.

The rest of this paper is organized as follows. Section 2 introduces background knowledge. Section 3 presents the TL* algorithm for learning ERAs. The proposed learning-based compositional verification framework is described in Section 4. The experiment results are given in Section 5. Related works are discussed in Section 6. The conclusion and the future work are given in Section 7.

2 PRELIMINARIES

We give some background knowledge about timed languages and event-recording automata in Section 2.1. The proposed algorithm for learning ERAs is inspired by the L* algorithm, which we recall in Section 2.2.

2.1 Background Knowledge

Let Σ be a finite alphabet. We use λ to denote the empty word. A *timed word* over Σ is a finite sequence $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \dots, n\}$ that are paired with nonnegative real numbers $t_i \in \mathbb{R}^+$ such that the sequence $t_1 t_2 \dots t_n$ of timed stamps is nondecreasing. For a timed word w_t, we can obtain its *untimed word*, denoted by $ut(w_t)$, by discarding all the time stamps, i.e., $ut(w_t) = a_1 a_2 \dots a_n$. Given another alphabet Σ', we use $w_t{\downarrow_{\Sigma'}}$ to denote the timed word obtained by removing from w_t all pairs (a_i, t_i) such that $a_i \notin \Sigma'$.

For every symbol $a \in \Sigma$, we use x_a to denote the *event-recording clock* [3] of a. Intuitively, x_a records the time elapsed since the last occurrence of a, i.e., once a occurs, clock x_a is reset. We use $C_\Sigma = \{x_a \mid a \in \Sigma\}$ to denote the set of event-recording clocks over Σ. A *clock valuation* $\gamma : C_\Sigma \mapsto \mathbb{R}^+$ is a function assigning a nonnegative real number to an event-recording clock.

A *clocked word* over Σ is a finite sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \dots, n\}$ that are paired with clock valuations γ_i such that $\gamma_1(x_a) = \gamma_1(x_b)$ for all $a, b \in \Sigma$ and $\gamma_i(x_a) = \gamma_{i-1}(x_a) + \gamma_i(x_{a_{i-1}})$ when $1 < i \le n$ and $a \ne a_{i-1}$. Each timed word $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ can be naturally transformed into a clocked word $cw(w_t) = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ where $\gamma_i(x_a) = t_i$ if $a_j \ne a$ for $1 \le j < i$; $\gamma_i(x_a) = t_i - t_j$ if there exists a_j such that $a_j = a$ for $1 \le j < i$ and $a_k \ne a$ for $j < k < i$. For example, the timed word $(a, 1)(b, 3)(a, 7)$ can be transformed into a clocked word $(a, \gamma_1)(b, \gamma_2)(a, \gamma_3)$ such that $\gamma_1(x_a) = \gamma_1(x_b) = 1$, $\gamma_2(x_a) = 2$, $\gamma_2(x_b) = 3$, $\gamma_3(x_a) = 6$, and $\gamma_3(x_b) = 4$.

An *atomic clock constraint* η is defined as $\eta = x_a \sim n \mid x_a - x_b \sim n$ where $x_a, x_b \in C_\Sigma$, $\sim \in \{<, \le, \ge, >\}$, and $n \in \mathbb{N}$. A *clock constraint* ϕ is a conjunction of atomic clock constraints. We say $\eta \in \phi$ if η is one of the conjuncts of ϕ. An *atomic clock guard* τ is defined as $\tau = x_a \sim n$ where $x_a \in C_\Sigma$, $\sim \in \{<, \le, >, \ge\}$, and $n \in \mathbb{N}$. A *clock guard* g is a conjunction of atomic clock guards. We say $\tau \in g$ if τ is one of the conjuncts of g.

A clock constraint ϕ identifies a $|\Sigma|$-dimensional polyhedron $[\![\phi]\!] \subseteq (\mathbb{R}^+)^{|\Sigma|}$, whereas a clock guard g identifies a $|\Sigma|$-dimensional hypercube $[\![g]\!] \subseteq (\mathbb{R}^+)^{|\Sigma|}$. We use G_Σ to denote the set of clock guards over C_Σ.

A *guarded word* over Σ is a sequence $w_g = (a_1, g_1)(a_2, g_2)\dots(a_n, g_n)$ where $a_i \in \Sigma$ and $g_i \in G_\Sigma$ for all $i \in \{1, 2, \dots, n\}$. The sub word of w_g, denoted by $[w_g]_i^j$, is the sequence $(a_i, g_i)(a_{i+1}, g_{i+1})\dots(a_j, g_j)$ for $1 \le i \le j \le n$. Given a clocked word $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ and a guarded word $w_g = (a_1, g_1)(a_2, g_2)\dots(a_n, g_n)$, we use $w_c \models w_g$ to denote $\gamma_i \models g_i$ for all $i \in \{1, 2, \dots, n\}$.

Given a clock constraint ϕ, if ϕ is satisfiable, there is a unique canonical clock constraint, denoted by $Can(\phi)$, among all the clock constraints identifying the polyhedron $[\![\phi]\!]$, obtained by closing ϕ under all consequences of pairs of conjuncts in ϕ. For example, given a constraint $\phi_1 : 0 \le x_a \le 3 \wedge 0 \le x_b \le 2$, its canonical form is $Can(\phi_1) : 0 \le x_a \le 3 \wedge 0 \le x_b \le 2 \wedge -3 \le x_b - x_a \le 2$.

For a clock constraint ϕ, we define the *reset* of an event-recording clock x_a in ϕ, denoted by $\phi[x_a \mapsto 0]$, as $Can(\phi')$ where ϕ' is obtained from $Can(\phi)$ by removing all conjunctions where x_a is included, and adding the conjunct $x_a \le 0$. For example, $\phi_1[x_a \mapsto 0] : x_a = 0 \wedge 0 \le x_b \le 2 \wedge 0 \le x_b - x_a \le 2$.

For a clock constraint ϕ, we define the *time elapsing* of ϕ, denoted by $\phi\!\uparrow$, as $Can(\phi'')$ where ϕ'' is obtained from $Can(\phi)$ by removing all clock upper bounds. For example, time elapsing of ϕ_1 is $\phi_1\!\uparrow : 0 \le x_a \wedge 0 \le x_b \wedge -3 \le x_b - x_a \le 2$.

Given a guarded word w_g and a clock constraint ϕ, the *strongest postcondition* of w_g given a precondition ϕ,

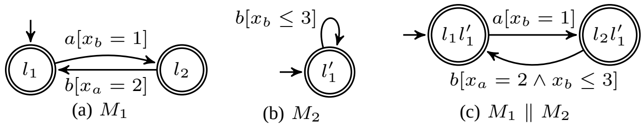


Fig. 1. Event-recording automata and timed language

denoted by $sp(\phi, w_g)$, is defined inductively as follows: $sp(\phi, \lambda) = \phi$; $sp(\phi, w_g(a, g)) = ((sp(\phi, w_g) \wedge g)[x_a \mapsto 0])\uparrow$. We often omit the initial clock constraint $\phi_0 = \bigwedge_{a,b \in \Sigma}(x_a = x_b)$, i.e., $sp(w_g) = sp(\phi_0, w_g)$.

The target model in this work, event-recording automata (ERAs), is formulated in Definition 1, and the parallel composition between two ERAs is formulated in Definition 2.

Definition 1: **(ERA)**. An *event-recording automaton* (ERA) $M = (\Sigma, L, L^0, \delta, L^f)$ consists of a finite input alphabet Σ, a finite set L of locations, a set of initial locations $L^0 \subseteq L$, a set of accepting locations $L^f \subseteq L$, and a transition function $\delta\colon L \times \Sigma \times G_\Sigma \mapsto 2^L$. We use $l \xrightarrow{a[g]} l'$ to denote $l' \in \delta(l, a, g)$ for $l, l' \in L$, $a \in \Sigma$, and $g \in G_\Sigma$. An ERA is *deterministic* if $|L^0| \leq 1$ and $|\delta(l, a, g)| \leq 1$, and if both $\delta(l, a, g_1)$ and $\delta(l, a, g_2)$ are defined and $g_1 \neq g_2$, then $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$ where $g_1, g_2 \in G_\Sigma$. A deterministic ERA is *complete* if $\bigcup_{g_i \in \{g | \delta(l,a,g) \neq \emptyset\}} [\![g_i]\!] = [\![true]\!]$ for all $l \in L$ and $a \in \Sigma$. $\square$

Note that in ERAs each event-recording clock $x_a \in C_\Sigma$ is implicitly and automatically reset when a transition with event a is taken. Fig. 1 (a) shows an example of a deterministic ERA M_1.

Given an ERA $M = (\Sigma, L, l_0, \delta, L^f)$, a clocked word $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\ldots(a_n, \gamma_n)$ is *accepted* by M if there exists a sequence of transitions $l_0 \xrightarrow{a_1[g_1]} l_1 \xrightarrow{a_2[g_2]} \cdots \xrightarrow{a_n[g_n]} l_n$ on M such that $l_0 \in L^0$, $l_n \in L^f$, and $\gamma_i \models g_i$ for all $i \in \{1, 2, \ldots, n\}$. A timed word w_t is accepted by M, if its clocked word w_c is accepted by M. The *timed language* accepted by M, denoted by $\mathcal{L}(M)$, is the set of timed words accepted by M. We give in Fig. 1 (a) an ERA M_1 that accepts the timed language $(a, t_1)(b, t_2)(a, t_3)(b, t_4)\ldots$ such that $t_1 = 1$, $t_{2i} - t_{2i-1} = 2$ and $t_{2i+1} - t_{2i} = 1$, and we give in Fig. 1 (b) an ERA M_2 that accepts the timed language $(b, t_1)(b, t_2)\ldots$ such that $t_1 \leq 3$ and $t_{i+1} - t_i \leq 3$. For a timed language L, we can obtain its *untimed language*, denoted by $ut(L)$, by collecting all the untimed words of L, i.e., $ut(L) = \{ut(w_t) \mid w_t \in L\}$.

Definition 2: **(Parallel Composition)**. Given two ERAs $M_i = (\Sigma_i, L_i, L_i^0, \delta_i, L_i^f)$ for $i \in \{1, 2\}$, their *parallel composition* is the ERA $M_1 \parallel M_2 = (\Sigma_1 \cup \Sigma_2, L_1 \times L_2, L_1^0 \times L_2^0, \delta, L_1^f \times L_2^f)$ where the set of event-recording clocks becomes $C_{\Sigma_1} \cup C_{\Sigma_2}$ and the transition relation δ is defined as follows where $[\![g_1]\!] \cap [\![g_2]\!] \neq \emptyset$.

$$\begin{cases} (l_1, l_2) \xrightarrow{a[g_1 \wedge g_2]} (l_1', l_2') & \text{if} \quad l_1 \xrightarrow{a[g_1]} l_1' \text{ and } l_2 \xrightarrow{a[g_2]} l_2' \\ (l_1, l_2) \xrightarrow{a[g_1]} (l_1', l_2) & \text{if} \quad l_1 \xrightarrow{a[g_1]} l_1' \text{ and } a \notin \Sigma_2 \\ (l_1, l_2) \xrightarrow{a[g_2]} (l_1, l_2') & \text{if} \quad l_2 \xrightarrow{a[g_2]} l_2' \text{ and } a \notin \Sigma_1 \end{cases}$$

Fig. 1 (a) and (b) give two deterministic ERAs M_1 and M_2, respectively, and their parallel compositional $M_1 \parallel M_2$ is shown in Fig. 1 (c).

In this work, we assume timed models and properties are all represented using ERAs. Given two ERAs M_1 and M_2 whose alphabets are Σ_1 and Σ_2, respectively, M_1 *satisfies* M_2, denoted by $M_1 \models M_2$, if $\mathcal{L}(M_1)\downarrow_{\Sigma_2} \subseteq \mathcal{L}(M_2)$ where $\mathcal{L}(M_1)\downarrow_{\Sigma_2} = \{w_t\downarrow_{\Sigma_2} \mid w_t \in \mathcal{L}(M_1)\}$. Figs. 1 (a) and (b) give two ERAs M_1 and M_2 such that $M_1 \models M_2$.

2.2 The L* Algorithm

The L* algorithm [5], [34] is a formal method to learn a minimal DFA (with the minimal number of locations) that accepts an unknown language U over an alphabet Σ. During the learning process, the L* algorithm interacts with a *Minimal Adequate Teacher* (Teacher for short) to make two types of queries: membership queries and candidate queries. A *membership query* for a string σ is a function $\mathcal{Q}_m$ such that if $\sigma \in U$, then $\mathcal{Q}_m(\sigma) = 1$; otherwise, $\mathcal{Q}_m(\sigma) = 0$. A *candidate query* for a DFA M is a function $\mathcal{Q}_c$ such that if $\mathcal{L}(M) = U$, then $\mathcal{Q}_c(M) = 1$; otherwise, $\mathcal{Q}_c(M) = 0$. During the learning process, the L* algorithm stores the membership query results in an *observation table* (S, E, T) where $S \subseteq \Sigma^*$ is a set of prefixes, $E \subseteq \Sigma^*$ is a set of suffixes, and $T : (S \cup S \cdot \Sigma) \times E \mapsto \{0, 1\}$ is a mapping function such that if $s \cdot e \in U$, then $T(s, e) = 1$; otherwise, i.e., $s \cdot e \notin U$, then $T(s, e) = 0$, where $s \in (S \cup S \cdot \Sigma)$ and $e \in E$. In the observation table, the L* algorithm categorizes strings based on Myhill-Nerode Congruence [21], as formulated in Definition 3.

Definition 3: **(Myhill-Nerode Congruence)**. For any two strings $\sigma, \sigma' \in \Sigma^*$, we say that they are *equivalent*, denoted by $\sigma \equiv \sigma'$, if $\sigma \cdot \rho \in U \Leftrightarrow \sigma' \cdot \rho \in U$, for all $\rho \in \Sigma^*$. Under the equivalence relation, we can say σ and σ' are the *representing strings* of each other with respect to U, denoted by $\sigma = [\sigma']_r$ and $\sigma' = [\sigma]_r$.

The L* algorithm always keeps the observation table *closed* and *consistent*. An observation table is *closed* if for all $s \in S$ and $\alpha \in \Sigma$, there always exists $s' \in S$ such that $s \cdot \alpha \equiv s'$. An observation table is *consistent* if for every two elements $s, s' \in S$ such that $s \equiv s'$, then $(s \cdot \alpha) \equiv (s' \cdot \alpha)$ for all $\alpha \in \Sigma$. If the observation table (S, E, T) is closed and consistent, the L* algorithm constructs a corresponding candidate DFA $C = (\Sigma_C, L_C, l_C^0, \delta_C, L_C^f)$ such that $\Sigma_C = \Sigma$, $L_C = S$, $l_C^0 = \{\lambda\}$, $\delta_C(s, \alpha) = [s \cdot \alpha]_r$ for $s \in S$ and $\alpha \in \Sigma$, and $L_C^f = \{s \in S \mid T(s, \lambda) = 1\}$. Subsequently, L* makes a candidate query for C.

If $\mathcal{Q}_C(M) = 0$, i.e., $\mathcal{L}(C) \neq U$, then Teacher gives a counterexample σ_{ce}. The counterexample σ_{ce} is *positive* if $\sigma_{ce} \in U \setminus \mathcal{L}(C)$, or *negative* if $\sigma_{ce} \in \mathcal{L}(C) \setminus U$. The L* algorithm then analyzes the counterexample σ_{ce} to find the witness suffix. For two strings that are classified by L* into an equivalence class, a *witness suffix* is a string that when appended to the two strings provides enough evidence for the two strings to be classified into two different equivalence classes under the Myhill-Nerode Congruence. Given an observation table (S, E, T) and a counterexample σ_{ce} given by Teacher, we define an

Algorithm 1: L* Algorithm

input : Σ: alphabet
output: a DFA accepting the unknown language U

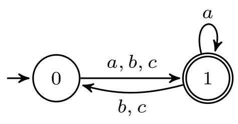
```
Let S = E = {λ} ;
Update T by Q_m(λ) and Q_m(λ · α), for all α ∈ Σ ;
while true do
    while there is (s · α) s.t. (s · α) ≢ s' for all s' ∈ S do
        S ⟵ S ∪ {s · α} ;
        Update T by Q_m((s · α) · β), for all β ∈ Σ ;
    Construct candidate DFA M from (S, E, T) ;
    if Q_c(M) = 1 then return M ;
    else
        σ_ce ⟵ the counterexample given by Teacher ;
        v ⟵ WS(σ_ce) ;
        E ⟵ E ∪ {v} ;
        Update T by Q_m(s · v) and Q_m(s · α · v), for
        all s ∈ S and α ∈ Σ ;
```

	λ
λ	0 (s_0)
a	1
b	1
c	1

(a)

	λ
λ	0 (s_0)
a	1 (s_1)
b	1
c	1
$a \cdot a$	1
$a \cdot b$	0
$a \cdot c$	0

(b)



(c)

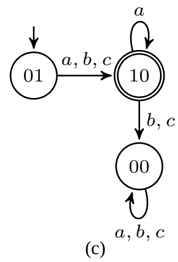
Fig. 2. L* Observation Table and Candidate DFA M_1

	λ	c
λ	0	1 (s_0)
a	1	0 (s_1)
b	1	0
c	1	0
$a \cdot a$	1	0
$a \cdot b$	0	0
$a \cdot c$	0	0

(a)

	λ	c
λ	0	1 (s_0)
a	1	0 (s_1)
b	1	0
c	1	0
$a \cdot a$	1	0
$a \cdot b$	0	0 (s_2)
$a \cdot c$	0	0
$ab \cdot a$	0	0
$ab \cdot b$	0	0
$ab \cdot c$	0	0

(b)



(c)

Fig. 3. L* Observation Table and Candidate DFA M_2

i-decomposition query of σ_{ce}, denoted by $\mathcal{Q}_m^i(\sigma_{ce})$, as follows: $\mathcal{Q}_m^i(\sigma_{ce}) = \mathcal{Q}_m([u_i]_r \cdot v_i)$ where $\sigma_{ce} = u_i \cdot v_i$ is a decomposition of σ_{ce} such that $|u_i| = i$, and $[u_i]_r$ is the representing string of u_i in S with respective to $\mathcal{L}(C)$. The *witness suffix* of σ_{ce}, denoted by $WS(\sigma_{ce})$, is the suffix v_i of the decomposition of σ_{ce} such that $\mathcal{Q}_m^i(\sigma_{ce}) \neq \mathcal{Q}_m^0(\sigma_{ce})$. Once the witness suffix $WS(\sigma_{ce})$ is obtained, L* uses $WS(\sigma_{ce})$ to refine the candidate DFA C until $\mathcal{L}(C) = U$. The pseudo-code of the L* algorithm is given in Algorithm 1.

We use an example to illustrate how the L* algorithm works to learn a minimal DFA accepting an unknown language. Suppose the unknown language $U = (a|b|c) \cdot a^*$ over $\Sigma = \{a, b, c\}$ needs to be learned. Initially, S and E are initialized to $\{\lambda\}$ and then the membership queries of λ, a, b, and c are performed. At this point, the observation table with $S = \{\lambda\}$, $E = \{\lambda\}$ is shown in Fig. 2 (a). The observation table now is not closed because there is no $s \in S$ such that $a \equiv s$. Therefore, a is added into S, and then the membership queries of aa, ab, and ac are performed respectively. At this point, the observation table with $S = \{\lambda, a\}$, $E = \{\lambda\}$ is closed as shown in Fig. 2 (b). The corresponding DFA M_1 is shown in Fig. 2 (c). The candidate query of M_1 is performed.

However, Teacher gives a negative counterexample abc that is accepted by M_1 but not in U. The L* algorithm analyzes the negative counterexample abc to get the witness suffix as follows: $\mathcal{Q}_m^0(abc) = 0$. $\mathcal{Q}_m^1(abc) = \mathcal{Q}_m([a]_r \cdot bc) = \mathcal{Q}_m(abc) = 0$, $\mathcal{Q}_m^2(abc) = \mathcal{Q}_m([ab]_r \cdot c) = \mathcal{Q}_m(\lambda \cdot c) = \mathcal{Q}_m(c) = 1 \neq \mathcal{Q}_m^0(abc)$. After analyzing the counterexample abc, the witness suffix is c. So, c is added into E, and the membership queries of c, ac, bc, cc, aac, abc, and acc are performed. The observation table now with $S = \{\lambda, a\}$, $E = \{\lambda, c\}$ is shown in Fig. 3 (a). However, the observation table is not closed because there is no $s \in S$ such that $ab \equiv s$. So, ab is added into S, and then the membership queries of aba, abb, abc, $abac$, $abbc$, and $abcc$ are performed. At this point, the observation table with $S = \{\lambda, a, ab\}$, $E = \{\lambda, c\}$ is closed as shown in Fig. 3 (b). The corresponding DFA M_2 is shown in Fig. 3 (c) and $\mathcal{L}(M_2) = U$.

Assume Σ is the alphabet of the unknown regular language U and the number of states of the minimal DFA is n. The L* algorithm needs $n - 1$ candidate queries and $O(|\Sigma|n^2 + n \log m)$ membership queries to learn the minimal DFA, where m is the length of the longest counterexample returned by Teacher. Angluin [5] proved that as long as the unknown language U is regular, the L* algorithm will learn a complete minimal DFA M such that $\mathcal{L}(M) = U$ in at most $n - 1$ iterations.

3 A Learning Algorithm for ERAs

This section is devoted to the TL* algorithm. Inspired by the L* algorithm, we develop a TL* algorithm, introduced in Section 3.1, to learn event-recording automata that accept timed languages. An example for illustrating the TL* algorithm is given in Section 3.2. Further discussions are given in Section 3.3. The correctness and termination of TL* are proved in Section 3.4.

3.1 The TL* Algorithm

In order to infer an ERA accepting an unknown timed language, the proposed TL* algorithm deals with guarded words. Before we get into the details, let us define the acceptance of a guarded word by an ERA.

Given a guarded word w_g, we use $\mathcal{L}(w_g)$ to denote the set of timed words w_t that are contained in w_g. That is, $\mathcal{L}(w_g) = \{w_t \mid cw(w_t) \models w_g\}$, e.g., $\mathcal{L}((a, x_a \geq 2))$ represents the timed language $\{(a, t) \mid t \geq 2\}$.

Definition 4: **(Acceptance of Guarded Words)**. Given an ERA $M = (\Sigma, L, l_0, \delta, L^f)$, a guarded word $w_g = (a_1, \hat{g_1})(a_2, \hat{g_2}) \ldots (a_n, \hat{g_n})$ is *accepted* by M, denoted by

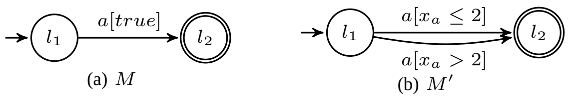


Fig. 4. Acceptance of Guarded Words

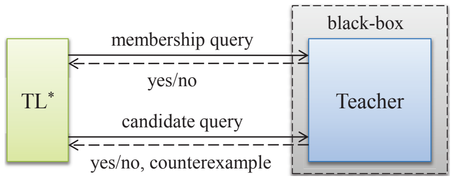


Fig. 5. Interaction between TL* and Teacher

$\mathcal{L}(w_g) \subseteq \mathcal{L}(M)$, if there exists a sequence of transitions $l_0 \xrightarrow{a_1[g_1]} l_1 \xrightarrow{a_2[g_2]} \cdots \xrightarrow{a_n[g_n]} l_n$ on M such that $l_0 \in L^0$, $l_n \in L^f$, and $[\![\hat{g_i}]\!] \subseteq [\![sp([w_g]_1^{i-1})]\!] \cap [\![g_i]\!]$ for all $1 \leq i \leq n$, where $[w_g]_1^0 = \lambda$.

Fig. 4 gives an example of the acceptance of guarded words. The guarded word $(a, true)$ is accepted by the ERA M as shown in Fig. 4 (a), and the two guarded words $(a, x_a \leq 2)$ and $(a, x_a > 2)$ are accepted by the ERA M' as shown in Fig. 4 (b). Note that the guarded word $(a, x_a \leq 3)$ is not accepted by M' because $[\![x_a \leq 3]\!] \not\subseteq [\![x_a \leq 2]\!]$ and $[\![x_a \leq 3]\!] \not\subseteq [\![x_a > 2]\!]$, while the guarded $(a, x_a \leq 1)$ is accepted by M' because $[\![x_a \leq 1]\!] \subseteq [\![x_a \leq 2]\!]$.

One may find that according to Definition 4, there might be a situation where we can construct two equivalent ERAs such that there exists a guarded word accepted by one but not the other. Fig. 4 shows such a case where M and M' are equivalent, and $(a, true)$ is accepted by M but not accepted by M'. This situation is not a problem because we define timed language on timed words instead of guarded words. Although M and M' accept different guarded words, they accept the same timed language (a, t) where $t \geq 0$.

Given a timed language U_T accepted by an ERA M_{U_T}, the proposed TL* algorithm interacts with a timed Teacher to make two types of queries: *timed membership queries* for guarded words and *timed candidate queries* for ERAs. Fig. 5 shows the interaction between the TL* algorithm and the timed Teacher. Note that our TL* algorithm is a black-box learning algorithm since only the Teacher knows about the timed language U_T to be learned. TL* views the Teacher as a black box and constructs an ERA according to the query results from the Teacher.

A *timed membership query* for a guarded word w_g is a function $\mathcal{Q}_{m^T}$ such that $\mathcal{Q}_{m^T}(w_g) = 1$ if w_g is accepted by M_{U_T}; otherwise $\mathcal{Q}_{m^T}(w_g) = 0$. A *timed candidate query* for an ERA M is a function $\mathcal{Q}_{c^T}$ such that $\mathcal{Q}_{c^T}(M) = 1$ if $\mathcal{L}(M) = U_T$; otherwise, $\mathcal{Q}_{c^T}(M) = 0$ and a guarded word as a counterexample will be given by the Teacher. A guarded word counterexample w_g is *negative* if $\mathcal{L}(w_g) \subseteq \mathcal{L}(M)$ and $\mathcal{L}(w_g) \not\subseteq U_T$. A guarded word counterexample w_g is *positive* if $\mathcal{L}(w_g) \subseteq U_T$ and $\mathcal{L}(w_g) \not\subseteq \mathcal{L}(M)$.

The idea behind the TL* algorithm is to first learn a DFA M accepting U, the untimed language of U_T, i.e., $U = ut(U_T)$, and then to refine the DFA M into a timed version, i.e., an ERA. Although the timed refinement may sometimes only add constraints on the transitions, it usually changes the structure of M by adding more locations and transitions. Indeed, it is well-known that adding constraints on the transitions of M is not sufficient in general to accept the timed language U_T. However, we still consider a two-phase algorithm consisting of an *untimed learning phase* and a *timed learning phase*. The reasons are as follows: (1) not all events are restricted by time conditions, and (2) if an event is restricted by time conditions, we do not want to actively guess all the possible time conditions for the event, which increases the number of membership queries exponentially and slows down the learning process. Instead, we passively assume the event is not restricted by any time condition and deduce the conditions from the counterexamples given by the Teacher. Algorithm 2 shows the pseudo-code of the TL* algorithm. The details are described in the following.

Untimed Learning. In this phase, the L* algorithm is used to learn a DFA M accepting the untimed language U with respect to U_T (Line 1 of Algorithm 2). The observation table (S, E, T) constructed in the learning process of L* is preserved before starting the timed learning phase (Line 2).

Timed Learning. In this phase, the TL* algorithm tries to refine the DFA M learned in the untimed learning phase into an ERA. The untimed alphabet Σ is extended into a timed alphabet $\Sigma_T \subseteq \Sigma \times G_\Sigma$ such that the observation table obtained from the untimed learning phase becomes a timed one. The results of membership queries for guarded words are stored in the timed observation table. This phase consists of the following steps.

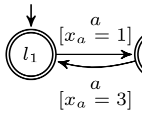
1) Perform a candidate query for the ERA M (Line 4). If the answer is "yes", M accepts the language U_T to be learned, and M is returned (Line 21).
2) If the answer to the candidate query for M is "no" with a counterexample $(a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$, TL* splits prefixes (rows) and suffixes (columns) in the observation table as follows. If a prefix p or a suffix e in the observation table has a substring of the form (a_i, g) for some $i \in \{1, 2, \ldots, n\}$ and $[\![g_i]\!] \cap [\![g]\!] \neq \emptyset$, then $[\![g]\!]$ is partitioned using g_i such that $[\![g]\!] = [\![g_i]\!] \cup G$ where $G = \{\hat{g_1}, \hat{g_2}, \ldots, \hat{g_m}\}$ is obtained by $[\![g]\!] - [\![g_i]\!]$ using DBM subtraction [27], [28]. The prefix p is split into $\{\hat{p_0}, \hat{p_1}, \hat{p_2}, \ldots, \hat{p_m}\}$ where (a_i, g_i) is a substring of $\hat{p_0}$ and $(a_i, \hat{g_j})$ is a substring of $\hat{p_j}$ for all $j \in \{1, 2, \ldots, m\}$ (Line 10). Similarly, the suffix e is also split into $\{\hat{e_0}, \hat{e_1}, \hat{e_2}, \ldots, \hat{e_m}\}$ where (a_i, g_i) is a substring of $\hat{e_0}$ and $(a_i, \hat{g_j})$ is a substring of $\hat{e_j}$ for all $j \in \{1, 2, \ldots, m\}$ (Line 11). Then the observation table is updated by performing timed membership queries $Q_{m^T}(\hat{p_j} \cdot \hat{e_j})$ for all $j \in \{0, 1, 2, \ldots, m\}$ (Line 12).
3) If the observation table (S, E, T) is not closed, i.e., there is a prefix $s \cdot \alpha$ with no $s' \in \Sigma_T$ such that $(s \cdot \alpha) \equiv s'$, then $s \cdot \alpha$ is added into S (Lines 13-14). The observation table is updated by performing the timed membership queries $Q_{m^T}(s \cdot \alpha \cdot \beta)$ for all $\beta \in \Sigma_T$ (Line 15).

Algorithm 2: TL* Algorithm

input : Σ: alphabet
output: an deterministic ERA M

1 Use L^* to learn a DFA M accepting U;
2 Let (S, E, T) be the observation table during the L* learning process;
3 $\alpha \leftarrow (\alpha, true)$; $s \leftarrow (s, true)$; $e \leftarrow (e, true)$ for each $\alpha \in \Sigma$, $s \in S$ and $e \in E$;
4 **while** $Q_{c^T}(M) = 0$ **do**
5   Let $(a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ be the counterexample given by the Teacher ;
6   **foreach** (a_i, g_i), $i \in \{1, 2, \dots, n\}$ **do**
7   **if** (a_i, g) is a substring of p or e for some $p \in S \cup (S \cdot \Sigma_T)$ and $e \in E$ such that $[\![g_i]\!] \cap [\![g]\!] \neq \emptyset$ **then**
8   Let $G = \{\hat{g_1}, \hat{g_2}, \dots, \hat{g_m}\}$ obtained by $[\![g]\!] - [\![g_i]\!]$;
9       $\Sigma_T = \Sigma_T \setminus \{(a_i, g)\} \cup \{(a_i, g_i), (a_i, \hat{g_1}), (a_i, \hat{g_2}), \dots, (a_i, \hat{g_m})\}$;
10   Split p into $\{\hat{p_0}, \hat{p_1}, \hat{p_2}, \dots, \hat{p_m}\}$ where (a_i, g_i) is a substring of $\hat{p_0}$ and $(a_i, \hat{g_j})$ is a substring of $\hat{p_j}$ for all $j \in \{1, 2, \dots, m\}$;
11   Split e into $\{\hat{e_0}, \hat{e_1}, \hat{e_2}, \dots, \hat{e_m}\}$ where (a_i, g_i) is a substring of $\hat{e_0}$ and $(a_i, \hat{g_j})$ is a substring of $\hat{e_j}$ for all $j \in \{1, 2, \dots, m\}$;
12   Update T by $Q_{m^T}(\hat{p_j} \cdot \hat{e_j})$ for all $j \in \{0, 1, 2, \dots, m\}$;
13   **while** *there exists* $(s \cdot \alpha)$ *such that* $(s \cdot \alpha) \not\equiv s'$ *for all* $s' \in S$ **do**
14     $S \longleftarrow S \cup \{s \cdot \alpha\}$;
15   Update T by $Q_{m^T}((s \cdot \alpha) \cdot \beta)$ for all $\beta \in \Sigma_T$;
16   $v \longleftarrow WS((a_1, g_1)(a_2, g_2) \cdots (a_n, g_n))$;
17   **if** $|v| > 0$ **then**
18     $E \longleftarrow E \cup \{v\}$;
19   Update T by $Q_{m^T}(s \cdot v)$ and $Q_{m^T}(s \cdot \alpha \cdot v)$ for all $s \in S$ and $\alpha \in \Sigma_T$;
20   Construct candidate M from (S, E, T);
21 **return** M;

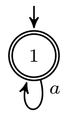
4) Analyze the counterexample π to find the witness suffix (Line 16). We define an i-decomposition query of π, denoted by $Q^i_{m^T}(\pi)$, as follows: $Q^i_{m^T}(\pi) = Q_{m^T}(s_i \cdot v_i)$ where $\pi = u_i \cdot v_i$ is a decomposition of π such that $|u_i| = i$ and $u_i \equiv s_i$ for some $s_i \in S$. The witness suffix of π, denoted by $WS(\pi)$, is the suffix v_i of π such that $Q^i_{m^T}(\pi) \neq Q^0_{m^T}(\pi)$. If there is a witness suffix v_i, i.e., $|v_i| > 0$, then v_i is added into the set of suffixes E (Lines 17-18). Then the observation table is updated by the timed membership queries $Q_{m^T}(s \cdot v_i)$ and $Q_{m^T}(s \cdot \alpha \cdot v_i)$ for each $s \in S$ and $\alpha \in \Sigma_T$ (Line 19).
5) Construct the ERA $M = (\Sigma_M, L_M, l^0_M, \delta_M, L^f_M)$ corresponding to the observation table (S, E, T) such that $\Sigma_M = \Sigma_T$, $L_M = S$, $l^0_M = \{\lambda\}$, $\delta_M(s, \alpha) = [s \cdot \alpha]_r$ for $s \in S$ and $\alpha \in \Sigma_T$, and $L^f_M = \{s \in S \mid T(s, \lambda) = 1\}$. Go to Step 1 (Line 20).



(a) ERA $\mathcal{A}_1$

	λ
λ	1 (s_0)
a	1

(b) T_1



(c) M_1

	λ
λ	1 (s_0)
$(a, true)$	1

(d) T_2

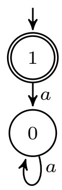
Fig. 6. Untimed Learning Phase

	λ
λ	1 (s_0)
$(a, x_a < 1)$	0
$(a, x_a \geq 1)$	0

(a) T_3

	λ
λ	1 (s_0)
$(a, x_a < 1)$	0 (s_1)
$(a, x_a \geq 1)$	0
$(a, x_a < 1)(a, x_a < 1)$	0
$(a, x_a < 1)(a, x_a \geq 1)$	0

(b) T_4



(c) M_2

Fig. 7. Timed Refinement 1

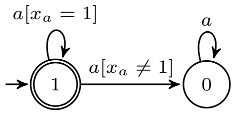
3.2 An Example

We use an example to illustrate the TL* algorithm. Suppose the timed language U_T to be learned is accepted by the ERA $\mathcal{A}_1$ as shown in Fig. 6 (a). In the untimed learning phase, L* is used to learn the DFA M_1, as shown in Fig. 6 (c), accepting the untimed language a^*, and the observation table (S, E, T) obtained by L* is shown in Fig. 6 (b). At this time, $\Sigma = \{a\}$, $S = \{\lambda\}$, and $E = \{\lambda\}$.

In the timed refinement phase, TL* first modifies the alphabet and the observation table into a timed version, i.e., $\Sigma_T = \{(a, true)\}$, $S = \{(\lambda, true)\}$, and $E = \{(\lambda, true)\}$. The current timed observation table T_2 is shown in Fig. 6 (d). Then, TL* performs the timed candidate query for the first candidate ERA M_1. However, the answer to the candidate query is "no" with a negative counterexample $(a, x_a < 1)$. Because there is a prefix $(a, true)$ in the observation such that $[\![x_a < 1]\!] \cap [\![true]\!] \neq \emptyset$, the prefix $(a, true)$ is split into $(a, x_a < 1)$ and $(a, x_a \geq 1)$, and the

	λ
λ	$1\ (s_0)$
$(a, x_a < 1)$	$0\ (s_1)$
$(a, x_a = 1)$	1
$(a, x_a > 1)$	0
$(a, x_a < 1)(a, x_a < 1)$	0
$(a, x_a < 1)(a, x_a = 1)$	0
$(a, x_a < 1)(a, x_a > 1)$	0

(a) T_5

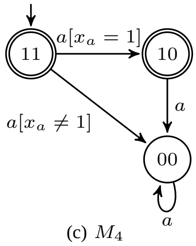


(b) M_3

Fig. 8. Timed Refinement 2

	λ	$(a, x_a = 1)$
λ	1	$1\ (s_0)$
$(a, x_a < 1)$	0	$0\ (s_1)$
$(a, x_a = 1)$	1	$0\ (s_2)$
$(a, x_a > 1)$	0	0
$(a, x_a < 1)(a, x_a < 1)$	0	0
$(a, x_a < 1)(a, x_a = 1)$	0	0
$(a, x_a < 1)(a, x_a > 1)$	0	0
$(a, x_a = 1)(a, x_a < 1)$	0	0
$(a, x_a = 1)(a, x_a = 1)$	0	0
$(a, x_a = 1)(a, x_a > 1)$	0	0

(b) T_7

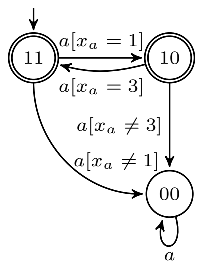


(c) M_4

Fig. 9. Timed Refinement 3

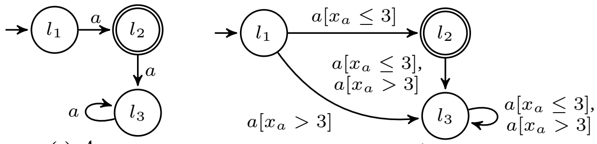
	λ	$(a, x_a = 1)$
λ	1	$1\ (s_0)$
$(a, x_a < 1)$	0	$0\ (s_1)$
$(a, x_a = 1)$	1	$0\ (s_2)$
$(a, 1 < x_a < 3)$	0	0
$(a, x_a = 3)$	0	0
$(a, x_a > 3)$	0	0
$(a, x_a < 1)(a, x_a < 1)$	0	0
$(a, x_a < 1)(a, x_a = 1)$	0	0
$(a, x_a < 1)(a, 1 < x_a < 3)$	0	0
$(a, x_a < 1)(a, x_a = 3)$	0	0
$(a, x_a < 1)(a, x_a > 3)$	0	0
$(a, x_a = 1)(a, x_a < 1)$	0	0
$(a, x_a = 1)(a, x_a = 1)$	0	0
$(a, x_a = 1)(a, 1 < x_a < 3)$	0	0
$(a, x_a = 1)(a, x_a = 3)$	1	1
$(a, x_a = 1)(a, x_a > 3)$	0	0

(a) T_8



(b) M_5

Fig. 10. Timed Refinement 4



(a) $\mathcal{A}_2$ (b) $\mathcal{A}_3$

Fig. 11. Learning Guided by a Friendly Teacher

timed membership queries for $(a, x_a < 1)$ and $(a, x_a \geq 1)$ are performed, respectively. The current observation table T_3 is shown in Fig. 7 (a). However, T_3 is not closed because there is $(a, x_a < 1)$ with no $s \in S$ such that $s \equiv (a, x_a < 1)$, so $(a, x_a < 1)$ is added into S and the membership queries for $(a, x_a < 1)(a, x_a < 1)$ and $(a, x_a < 1)(a, x_a \geq 1)$ are performed, respectively. The closed observation table T_4 and its corresponding ERA M_2 are shown in Fig. 7 (b) and (c), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a \geq 1)\}$, $S = \{(\lambda, true), (a, x_a < 1)\}$, and $E = \{(\lambda, true)\}$.

In the second iteration of the timed refinement phase, TL* performs the timed candidate query for M_2. However, the answer is still "no" with a positive counterexample $(a, x_a = 1)$. Because there are two prefixes $(a, x_a \geq 1)$ and $(a, x_a < 1)(x_a \geq 1)$ in the observation table (S, E, T) such that $[\![x_a = 1]\!] \cap [\![x_a \geq 1]\!] \neq \emptyset$, the prefix $(a, x_a \geq 1)$ is split into $(a, x_a = 1)$ and $(a, x_a > 1)$, and the prefix $(a, x_a < 1)(x_a \geq 1)$ is split into $(a, x_a < 1)(x_a = 1)$ and $(a, x_a < 1)(x_a > 1)$, respectively. The timed membership queries for the new prefixes are performed. The current closed observation table T_5 and its corresponding ERA M_3 are shown in Fig. 8 (a) and (b), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a > 1)\}$, $S = \{(\lambda, true), (a, x_a < 1)\}$, and $E = \{(\lambda, true)\}$.

In the third iteration of the timed refinement phase, TL* performs the timed candidate query for the ERA M_3. However, the answer is still "no" with a negative counterexample $\pi = (a, x_a = 1)(a, x_a = 1)$. This time, no prefix or suffix in the observation table has to be split. TL* analyzes the counterexample as follows. $Q^0_{m^T}(\pi) = Q_{m^T}((a, x_a = 1)(a, x_a = 1)) = 0$. $Q^1_{m^T}(\pi) = Q^1_{m^T}([(a, x_a = 1)]_r(a, x_a = 1)) = Q_{m^T}((a, x_a = 1)) = 1 \neq Q^0_{m^T}(\pi)$. Thus, we have a witness suffix $v = (a, x_a = 1)$, and v is added into the set E. Then the membership queries for $s \cdot (a, x_a = 1)$ for all $s \in S$ are performed. The closed observation table T_7 and its corresponding ERA M_4 are shown in Fig. 9 (a) and (b), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a > 1)\}$, $S = \{(\lambda, true), (a, x_a < 1), (a, x_a = 1)\}$, and $E = \{(\lambda, true), (a, x_a = 1)\}$.

In the fourth iteration of the timed refinement phase, TL* performs the timed candidate query for the ERA M_4 again. However, the answer is still "no" with a positive counterexample $\pi = (a, x_a = 1)(a, x_a = 3)$. Three prefixes $(a, x_a > 1)$, $(a, x_a < 1)(a, x_a > 1)$, and $(a, x_a = 1)(a, x_a > 1)$ in the observation table T_7 have to be split, and the new split prefixes are shown in Fig. 10 (a). The timed membership queries for the new split prefixes concatenated with e for all $e \in E$ are performed. Then the TL* algorithm analyzes the counterexample. Since $Q^0_{m^T}(\pi) = Q^1_{m^T}(\pi) = Q^2_{m^T}(\pi)$, there is no witness suffix for π. The closed observation table T_8 is shown in Fig. 10 (a), and its corresponding ERA M_5 is constructed as shown in Fig. 10 (b). At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, 1 < x_a < 3), (a, x_a = 3), (a, x_a > 3)\}$, $E = \{(\lambda, true), (a, x_a < 1), (a, x_a = 1)\}$, and $E = \{(\lambda, true), (a, x_a = 1)\}$.

In the fifth iteration of the timed refinement, TL* performs the timed candidate query for M_5. This time, Teacher says that $\mathcal{L}(M_5) = U_T$, and the learning process of the TL* algorithm is finished.

3.3 Discussion Regarding the Teacher

Since TL* is a black-box learning algorithm, one may find that the guidance of the Teacher affects the learning of TL*. Thus, we give a discussion for the guidance of the Teacher in this section. Note that the reason for the discussion here is that the proposed TL* is a generic algorithm, which is not limited to our setting and might be used in different contexts for learning ERAs. Let us consider a timed language accepting timed words (a, t) where $t \leq 3$. In the untimed learning phase, TL* performs

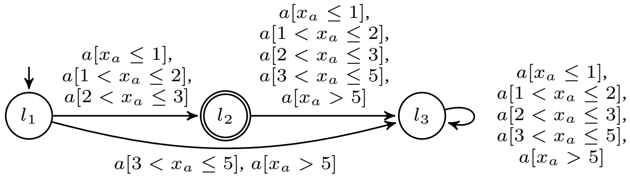


Fig. 12. Learning Guided by a Bad Teacher

the L* algorithm to learn a DFA M accepting the untimed word a, as shown in Fig. 11 (a). When the Teacher answers the timed candidate query for $\mathcal{A}_2$, if it returns a beautiful negative counterexample $(a, x_a > 3)$, the alphabet a is split into $(a, x_a \leq 3)$ and $(a, x_a > 3)$, and the final learned ERA $\mathcal{A}_3$ is as shown in Fig. 11 (b).

What if the Teacher is not friendly? That is, Teacher always gives counterexamples whose time constraints are not exactly the boundary guards. Let us consider the above example again. Suppose Teacher gives a negative counterexample $(a, x_a > 5)$ instead of $(a, x_a > 3)$ when answering the timed candidate query of M. The alphabet a is split into $(a, x_a \leq 5)$ and $(a, x_a > 5)$, and both of them are not accepted. After this, Teacher can only return positive counterexamples of the form $(a, x_a \sim c)$ where $\sim \in \{<, \leq\}$ and $c \leq 3$. Let us suppose that Teacher gives the positive counterexamples in the worst way. It gives a positive counterexample $(a, x_a \leq 1)$ which causes the split of the alphabet a into $(a, x_a \leq 1)$, $(a, 1 < x_a \leq 5)$ and $(a, x_a > 5)$ where only $(a, x_a \leq 1)$ is accepted. And Teacher gives another positive counterexample $(a, x_a \leq 2)$, which causes the split of the alphabet as: $(a, x_a \leq 1)$, $(a, 1 < x_a \leq 2)$, $(a, 2 < x_a \leq 5)$ and $(a, x_a > 5)$, where $(a, x_a \leq 1)$ and $(a, 1 < x_a \leq 2)$ are accepted. Then Teacher gives the final positive counterexample $(a, x_a \leq 3)$, which causes the split of the alphabet as: $(a, x_a \leq 1)$, $(a, 1 < x_a \leq 2)$, $(a, 2 < x_a \leq 3)$, $(a, 3 < x_a \leq 5)$ and $(a, x_a > 5)$, where $(a, x_a \leq 1)$, $(a, 1 < x_a \leq 2)$ and $(a, 2 < x_a \leq 3)$ are accepted. The final learned ERA is as shown in Fig. 12.

We can observe that with a friendly Teacher, unnecessary alphabet split can be avoided, while with a bad Teacher, unnecessary split might occur, but they are always in the same class (leading to the same state), as shown in Fig. 12. However, even with the worst Teacher, the alphabet split will be approaching the boundary as illustrated in the above example and in Fig. 12. Recall from Section 2.1 that the constant in a clock constraint is necessarily an integer.

In our setting of compositional verification based on the TL* algorithm (c.f. Section 4), we implement the Teacher by model checking, and the boundary time constraint is specified either in the models or in the property, i.e., a friendly Teacher, which avoids unnecessary split – this is also confirmed by our experiments.

3.4 Termination and Correctness

Given a timed language U_T accepted by a deterministic ERA $\mathcal{A} = (\Sigma, L, l_0, \delta, L^f)$, TL* learns an ERA to accept U_T. After the untimed learning phase, each untimed alphabet $(\alpha, true)$, $\alpha \in \Sigma$, may be split according to the guard condition of the counterexamples returned by Teacher. With a friendly Teacher, each untimed word $(\alpha, true)$ will be split into $|G_{\mathcal{A}}|$ guarded words, where $G_{\mathcal{A}}$ is the set of clock zones partitioned by the clock guards appearing in $\mathcal{A}$. For example, the clock guard appearing in $\mathcal{A}_3$, as shown in Fig. 11 (b), is $x_a > 3$, so $G_{\mathcal{A}_3} = \{x_a \leq 3, x_a > 3\}$.

With a bad Teacher, the number of alphabet split is more than $|G_{\mathcal{A}}|$. For each event $\alpha \in \Sigma$, if $(\alpha, true)$ needs to be split, Teacher will give a negative counterexample (α, g) and g is of the form $(\alpha, x_\beta \sim \overrightarrow{c})$ or $(\alpha, x_\beta \sim' \overleftarrow{c})$, where $\beta \in \Sigma$, $\sim \in \{<, \leq\}$, $\sim' \in \{>, \geq\}$, and $\overrightarrow{c}, \overleftarrow{c} \in N$. Basically, $\overrightarrow{c}$ and $\overleftarrow{c}$ are the upper and lower bounds of the clock x_β, respectively. We can construct a set of regions with respect to $\overrightarrow{c}$ and $\overleftarrow{c}$, denoted by $\mathcal{R}_{\overleftrightarrow{c}}$. For example, given $\overrightarrow{c} = 3$ and $\overleftarrow{c} = 1$, $\mathcal{R}_{\overleftrightarrow{c}} = \{x_\beta = 1, 1 < x_\beta < 2, x_\beta = 2, 2 < x_\beta < 3, x_\beta = 3, x_\beta > 3\}$. Thus, with a bad Teacher, each event $\alpha \in \Sigma$ might be split at most $|C_\Sigma| \cdot |\mathcal{R}_{\overleftrightarrow{c}}|$ times.

Let $\varrho = \max\{|G_{\mathcal{A}}|, |C_\Sigma| \cdot |\mathcal{R}_{\overleftrightarrow{c}}|\}$. In general, each membership query of untimed word $(\alpha, true)$ gives rise to at most ϱ timed membership queries. In total, TL* needs to perform $O(|\Sigma| \cdot \varrho \cdot |L|^2 + |L| \log |\pi|)$ timed membership queries, where π is the counterexample given by the Teacher. We will show in Theorem 1 that TL* needs to perform $O(|L| + \varrho \cdot |\Sigma|)$ candidate queries.

Lemma 1: Given a closed and consistent observation table (S, E, T), any deterministic ERA consistent with T has at least $|S|$ locations.

Proof: We first define a row in the observation table. If $p \in S \cup (S \cdot \Sigma)$ is a prefix (row) of the table, we use $row(p)$ to denote the function $f : E \mapsto \{0, 1\}$ which is defined by $f(e) = T(p \cdot e)$ for $e \in E$. Let $M = (\Sigma, L, l^0, \delta, L^f)$ be an ERA consistent with T. We then define $f'(s) = \delta(l^0, s)$ for every $s \in S$. For any two $s_1, s_2 \in S$, we have $row(s_1) \neq row(s_2)$ implying that there exists $e \in E$ such that $T(s_1 \cdot e) \neq T(s_2 \cdot e)$. Since M is consistent with T, exactly one of $\delta(l^0, s_1 \cdot e)$ and $\delta(l^0, s_2 \cdot e)$ is in L^f implying that $\delta(l^0, s_1)$ and $\delta(l^0, s_2)$ are distinct locations. Thus, $f'(s)$ takes on at least $|S|$ values implying that M has at least $|S|$ locations. □

Theorem 1: The TL* algorithm is correct and terminates in a finite number of iterations.

Proof: The correctness is based on the fact that the TL* algorithm returns an ERA only if it accepts the unknown timed language U_T. Let $\mathcal{A} = (\Sigma, L, l^0, \delta, L^f)$ be an ERA accepting U_T. In each iteration, the TL* algorithm either adds a row into S in the observation table (S, E, T) or splits a clock guard of an event $\alpha \in \Sigma$ into at least two disjoint clock guards. Since the observation table should be consistent with $\mathcal{A}$ (otherwise, the Teacher must have given wrong answers to the membership queries), TL* adds at most $|L|$ rows into S. Lastly, each untimed alphabet $(\alpha, true)$ splits at most ϱ times. Thus, the TL* algorithm terminates after $O(|L| + \varrho \cdot |\Sigma|)$ iterations. □

Theorem 2: The ERA learned by the TL* algorithm has the minimal number of locations.

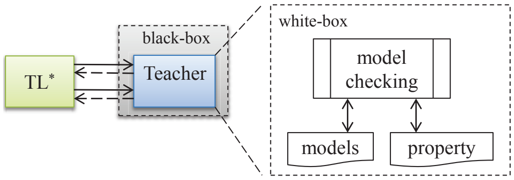


Fig. 13. Model Checking Plays the Teacher Role

Proof: Given a closed and consistent observation table (S, E, T), TL* constructs an ERA M exactly with $|S|$ locations. By Lemma 1, we can conclude that M has the minimal number of locations. $\square$

From the above arguments, we can conclude the followings: even if the teacher is bad, i.e., it gives on purpose counterexamples as little helpful as possible, as long as the it answers the membership and candidate queries correctly, our TL* algorithm can learn an ERA with the minimal number of locations to accept the unknown timed language and terminate in a finite number of iterations.

4 An Automatic Compositional Verification Framework for Timed Systems

This section is devoted to an automatic learning-based compositional verification framework for timed systems. The proposed framework is introduced in Section 4.1. An example is given in Section 4.2 for illustrating the framework. The correctness and termination of the framework are proved in Section 4.3.

4.1 Automatic Verification Framework

To learn an ERA accepting a timed language, the TL* algorithm needs the guidance of the Teacher to answer membership and candidate queries. Thus, to use TL* to automatically generate the assumption for AGR, the proposed framework has to play the Teacher role to answer the membership and candidate queries needed by TL*. In the proposed compositional verification framework, we adopt model checking to answer the queries from TL*. Fig. 13 shows the big picture of the TL* algorithm, the Teacher, and model checking. Note that the Teacher itself, played by model checking, is a white-box setting since it knows the component models and the property. However, the Teacher is still a black box to the TL* algorithm.

Fig. 14 shows the overall flow of the learning-based compositional verification for timed systems based on the AG-NC proof rule. It consists of two phases, namely *untimed verification phase* for constructing the untimed assumption (environment) for M_1 to satisfy the property, and *timed verification phase* for refining the untimed assumption into a timed one and concluding the result of the timed verification.

The target ERA to be learned by TL* is the *weakest assumption* A_w under which M_1 satisfies φ, i.e., for any environment E, $M_1 \parallel E \models \varphi$ iff $E \models A_w$. To guide TL* to learn the weakest assumption A_w, model checking is used to answer the membership and candidate queries needed by TL*. Although the target ERA for TL* is the weakest assumption A_w, the proposed framework terminates as soon as compositional verification gets conclusive results, which is often before the weakest assumption A_w is learned. The details of the learning-based compositional verification framework are described as follows. Note that the alphabet of the assumption ranges over $\Sigma_A = (\Sigma_{M_1} \cup \Sigma_\varphi) \cap \Sigma_{M_2}$.

Algorithm 3: Untimed Candidate Query Q_c

input : C: an untimed ERA
output: (0/1, a counterexample π)

if $\mathcal{L}((M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut} \parallel C) = \emptyset$ **then**
  if $\mathcal{L}((M_2)^{ut} \parallel \overline{C}) = \emptyset$ **then** **return** $(1, \lambda)$;
  else **return** $(0, \pi)$, $\pi \in \mathcal{L}((M_2)^{ut} \parallel \overline{C})$;
else **return** $(0, \pi)$, $\pi \in \mathcal{L}((M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut} \parallel C)$;

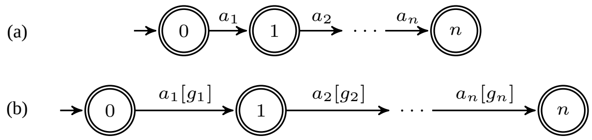


Fig. 15. Prefix-Accepting Automata

Untimed Verification Phase. In this phase, the L* algorithm [5] is used to learn an untimed assumption according to the AG-NC proof rule such that $(M_1)^{ut} \parallel (M_2)^{ut} \models (\varphi)^{ut}$ is proved or disproved. We use $(M_1)^{ut}$ to denote the untimed version of M_1, i.e., all the time constraints on transitions are ignored. The L* algorithm constructs a candidate DFA A after several untimed membership queries. The answer to an untimed membership query for an untimed behavior σ is positive only if the behavior σ does not violate the property $(\varphi)^{ut}$ when interacting with $(M_1)^{ut}$, i.e., $\sigma \notin \mathcal{L}((M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut})$. This is basically an emptiness problem of $\mathcal{L}(M_\sigma \parallel (M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut})$ where M_σ is a DFA accepting all the prefixes of σ. For an untimed behavior $\sigma = a_1 a_2 \ldots a_n$, we can easily construct M_σ as shown in Fig. 15 (a). The emptiness problem can be checked by model checking.

The candidate query for A is answered by the Q_c algorithm, as given in Algorithm 3. If $(M_1)^{ut} \parallel (M_2)^{ut} \models (\varphi)^{ut}$ is disproved in this phase with a untimed counterexample π, we have to check whether it is a real timed counterexample, i.e., $\pi \in \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$ and $\pi \in \mathcal{L}(M_2)$. If yes, we can conclude $M_1 \parallel M_2 \not\models \varphi$. If not, we cannot conclude anything here and the flow goes to the timed verification phase.

Timed Verification Phase. In this phase, the TL* algorithm is used to learn the timed assumption A according to the AG-NC proof rule such that $M_1 \parallel M_2 \models \varphi$ is proved or disproved. The TL* algorithm constructs a timed assumption A after several timed membership queries. The answer to the timed membership query for a guarded word σ is positive only if the behavior σ does not violate the property φ when interacting with M_1, i.e., $\mathcal{L}(\sigma) \not\subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$. Basically, this is an emptiness problem of $\mathcal{L}(M_\sigma \parallel M_1 \parallel M_{\overline{\varphi}})$ where M_σ is an ERA

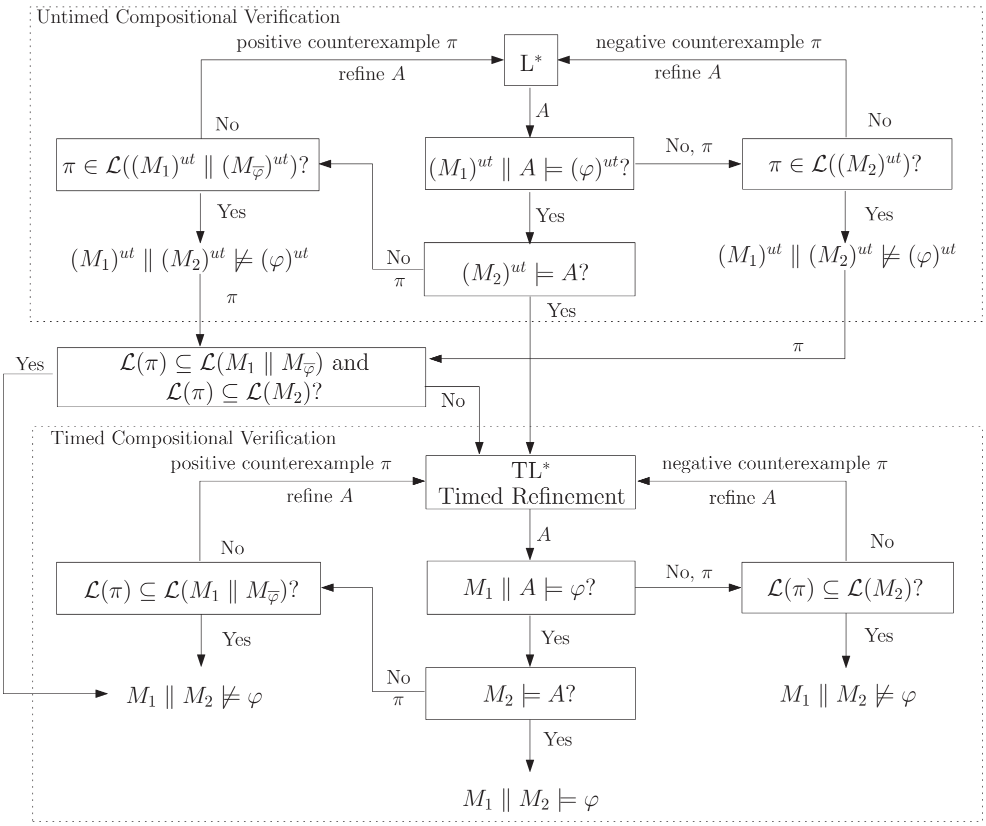


Fig. 14. Flow of compositional verification framework for timed systems

such that all the prefixes of σ are accepted by M_σ. For a guarded word $\sigma = (a_1, g_1)(a_2, g_n) \ldots (a_n, g_n)$, we can easily construct M_σ as shown in Fig. 15 (b). The emptiness problem can be checked by timed model checking.

The candidate query of the timed assumption A is answered by the Q_{c^T} algorithm, as given in Algorithm 4. The details are described in the following.

1) If $M_1 \parallel C \models \varphi$ and $M_2 \models C$, we can conclude $M_1 \parallel M_2 \models \varphi$ (Lines 1-2 of Algorithm 4).
2) If $M_1 \parallel C \not\models \varphi$, a counterexample π is given (Line 12). We check whether the untimed trace $(\pi)^{ut}$ is also an untimed counterexample. If yes, the sequence of events is wrong no matter how it is restricted by time constraints and the projected counterexample $\pi{\downarrow_\Sigma}$ is returned as a negative counterexample (Lines 13-14). If not, the sequence of events is allowed but the time constraints of events lead to an error. The strategy of refining the time constraints is as follows. Given any clock constraint η in $sp(\pi)$, if any event of the counterexample makes η unsatisfiable, then π will not violate the property φ anymore (see Theorem 3). Suppose the projected counterexample is $\pi{\downarrow_\Sigma} = (a_1, g_1)(a_2, g_2) \cdots (a_m, g_m)$. For simplicity, we always select the clock constraint $\eta = x_{a_{m-1}} - x_{a_m} \sim c$ representing the time difference between the occurrences of a_{m-1} and a_m. If a_m is not performed in $[\![x_{a_{m-1}} \sim c]\!]$, then η becomes unsatisfiable. Thus, the negative counterexample $(a_1, g_1)(a_2, g_2) \cdots (a_m, x_{a_{m-1}} \sim c)$ is returned to the TL* algorithm (Lines 15-19).
3) If $M_1 \parallel C \models \varphi$ but $M_2 \not\models C$, a counterexample π is given. We check whether $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$. If yes, we can conclude $M_1 \parallel M_2 \not\models \varphi$ (Line 5). If not, π is a positive counterexample. Note that each time a counterexample is returned to TL*, some events in the alphabet Σ might be split. We want to reduce the split as much as possible. Thus, before directly returning π to TL*, we try to find a counterexample π' similar to π but with less split of events in Σ. First, we obtain a normalized behavior π' w.r.t. Σ by replacing each event/guard pair (a, g) appearing in π with $(a, g') \in \Sigma$ and $[\![g]\!] \subseteq [\![g']\!]$ (Line 7). We consider a behavior $\sigma_1 \cdot \sigma_2$ where $\sigma_1 = (\pi')_1^i$ is a prefix of π' and $\sigma_2 = (\pi)_{i+1}^n$ is a suffix of π for i from $n-1$ to 1 and $n = |\pi|$. If $\mathcal{L}(\sigma_1 \cdot \sigma_2) \not\subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$ for some $1 \leq i \leq n-1$, then $\sigma_1 \cdot \sigma_2$ is a better candidate than π (Lines 8-9). Otherwise, π is returned (Line 10).

Theorem 3: Let $\pi = (a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ be a guarded word. Given any clock constraint $\eta \in sp(\pi)$ of the form $x_{a_i} - x_{a_j} \sim c$ for some i and j, $1 \leq i < j \leq n$, we can obtain $\pi' = (a_1, g_1) \cdots (a_j, x_{a_i} \overline{\sim} c) \cdots (a_n, g_n)$ and $a_i \neq a_j \neq a_k$ for all k, $j < k \leq n$ such that $[\![sp(\pi)]\!] \cap [\![sp(\pi')]\!] = \emptyset$.

Proof: Let $\overline{\sim}$ be the complement of $\sim$ where the complement of $<, \leq, \geq, >$ is $\geq, >, <, \leq$, respectively. $x_{a_i} - x_{a_j}$ represents the time difference between the occurrences of a_i

Algorithm 4: Timed Candidate Query Q_{c^T}

input : C: the candidate ERA; Σ: the alphabet of TL*
output: $(0/1$, a counterexample $\pi)$

if $\mathcal{L}(M_1 \parallel M_{\overline{\varphi}} \parallel C) = \emptyset$ **then**
  if $\mathcal{L}(M_2 \parallel \overline{C}) = \emptyset$ **then return** $(1, \lambda)$;
  else
    Let π be a trace in $\mathcal{L}(M_2 \parallel \overline{C})$ and $|\pi| = n$;
    if $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$ **then return** $(0, \pi)$;
    else
      $\pi' \longleftarrow$ NORM(π, Σ) ;
      for $i = n - 1$ **to** 1 **do**
        if $\mathcal{L}((\pi')_1^i \cdot (\pi)_{i+1}^n) \not\subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$ **then return** $(0, (\pi')_1^i \cdot (\pi)_{i+1}^n)$;
      return $(0, \pi)$;
else
  Let π be a trace in $\mathcal{L}(M_1 \parallel M_{\overline{\varphi}} \parallel C)$;
  if $(\pi)^{ut} \in \mathcal{L}((M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut})$ **then**
    return $(0, \pi\downarrow_\Sigma)$;
  else
    $\pi' \longleftarrow \pi\downarrow_\Sigma$ and $\pi' = (a_1, g_1) \cdots (a_m, g_m)$;
    Let $\eta \in sp(\pi)$ and $\eta = x_{a_{m-1}} - x_{a_m} \sim c$;
    $\pi' \longleftarrow (a_1, g_1)(a_2, g_2) \cdots (a_m, x_{m-1} \sim c)$;
    return $(0, \pi')$;

and a_j for some i and j, $1 \le i < j \le n$. If a_j is performed when $x_{a_i} \overline{\sim} c$ in π' such that $a_k \ne a_i$ and $a_k \ne a_j$ for all k, $j < k \le n$, then $x_{a_i} - x_{a_j}$ is not changed after a_i and a_j are performed and $x_{a_i} - x_{a_j} \overline{\sim} c \in sp(\pi')$. Since $[\![x_{a_i} - x_{a_j} \overline{\sim}\, c]\!] \cap [\![x_{a_i} - x_{a_j} \sim c]\!] = \emptyset$ and $x_{a_i} - x_{a_j} \sim c \in sp(\pi)$, we can conclude $[\![sp(\pi')]\!] \cap [\![sp(\pi)]\!] = \emptyset$. $\square$

4.2 An Example

We use an example to illustrate the proposed framework. Fig. 16 shows an I/O system [13] consisting of two components, INPUT and OUTPUT. There are four events, *input*, *send*, *output*, and *ack* in the system. The pairs of event-recording clocks and their corresponding events are: x_i : *input*, x_s : *send*, x_o : *output*, and x_a : *ack*. The model of the INPUT component is shown in Fig. 16 (a). INPUT performs an *input* event within one time unit once it receives an *ack* event from OUTPUT. Subsequently, it performs a *send* event to notify OUTPUT that an *input* event has been performed and waits another *ack* event from OUTPUT. The model of the OUTPUT component is shown in Fig. 16 (b). After receiving a *send* event, OUTPUT performs an *output* event within one time unit and then performs an *ack* event within one time unit after the *output* event. The system property φ, as shown in Fig. 16 (c), is that *input* and *output* events should alternate and the time difference between every two consecutive events should not exceed five time units. The negation of the property is given in Fig. 16 (d) where τ is the error location, and we assume that the negation of the property is specified by users.

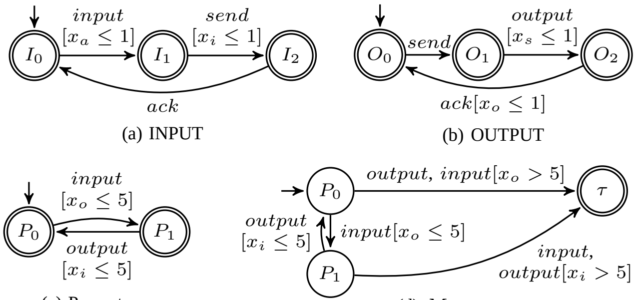
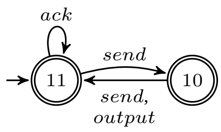


Fig. 16. Models and property of the I/O system

	λ	ack
λ	1	1 (s_0)
$send$	1	0 (s_2)
$output$	0	0 (s_1)
ack	1	1
$output \cdot send$	0	0
$output \cdot output$	0	0
$output \cdot ack$	0	0
$send \cdot send$	1	1
$send \cdot output$	1	1
$send \cdot ack$	0	0

(a) T_2



(b) A_2

Fig. 17. Untimed assumption A_2

We skip the details on the untimed verification phase, which can be found in [13]. After the untimed verification phase, the untimed assumption A_2, as shown in Fig. 17 (b), is learned by L* to prove (INPUT)^{ut} $\parallel$ (OUTPUT)^{ut} $\models$ $(\varphi)^{ut}$. We remark the assumption as A_2 instead of A_1 because it is the second assumption generated in the untimed verification phase. For simplicity, we omit non-accepting locations of ERAs in the following. The untimed observation table of A_2 is shown in Fig. 17 (a). The flow goes to the timed verification phase, and the untimed observation table is modified into a timed version.

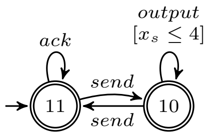
In the first iteration, the timed candidate query for A_2 is performed and the result is negative because INPUT $\parallel$ A_2 $\not\models$ φ with a counterexample $\pi = (input, x_a \le 1)(send, x_i \le 1)(output, x_i > 5)$. The counterexample projected to Σ_A is $\pi' = (send, true)(output, true)$. The strongest post conditions $sp(\pi)$ are as follows: $x_o = 0$, $x_s > 4$, $x_i > 4$, $x_a > 5$, $0 \le x_i - x_s \le 1$, $0 \le x_a - x_s \le 2$, $0 \le x_a - x_i \le 1$, $x_s - x_o > 4$, $x_i - x_o > 5$, and $x_a - x_o > 5$. We select $x_s - x_o > 4$, and $(send, true)(output, x_s > 4)$ is returned to TL*. The observation table is split according to the returned counterexample as shown in Fig. 18 (a) and its corresponding ERA A_3 is shown in Fig. 18 (b).

In the second iteration, the timed candidate query for A_3 is performed and the result is negative because INPUT $\parallel$ A_3 $\not\models$ φ with a counterexample $\pi = (input, x_a \le 1)(send, x_i \le 1)(output, x_s \le 4)(output, x_s \le 4)$ whose projection to Σ_A is $\pi' = (send, true)(output, x_s \le 4)(output, x_s \le 4)$. Because $(\pi)^{ut} \in \mathcal{L}((M_1)^{ut} \parallel (M_{\overline{\varphi}})^{ut})$, the negative counterexample π' is returned to TL*. After analyzing π', TL* adds the witness suffix $\sigma_2 = (output, x_s \le 4)$ into the set E as shown in Fig. 19 (a). The corresponding ERA A_4 is shown in Fig. 19 (b).

In the third iteration, the timed candidate query for

	λ	σ_1
λ	1	1 (s_0)
$(send, true)$	1	0 (s_2)
$(output, x_s \leq 4)$	0	0 (s_1)
$(output, x_s > 4)$	0	0
$(ack, true)$	1	1
$(output, x_s \leq 4)(send, true)$	0	0
$(output, x_s > 4)(send, true)$	0	0
$(output, x_s \leq 4)(output, x_s \leq 4)$	0	0
$(output, x_s \leq 4)(output, x_s > 4)$	0	0
$(output, x_s > 4)(output, x_s \leq 4)$	0	0
$(output, x_s > 4)(output, x_s > 4)$	0	0
$(output, x_s \leq 4)(ack, true)$	0	0
$(output, x_s > 4)(ack, true)$	0	0
$(send, true)(send, true)$	1	1
$(send, true)(output, x_s \leq 4)$	1	0
$(send, true)(output, x_s > 4)$	0	0
$(send, true)(ack, true)$	0	0

(a) $\sigma_1 = (ack, true)$

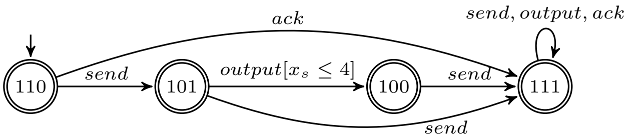


(b) A_3

Fig. 18. First timed assumption

	λ	σ_1	σ_2
λ	1	1	0 (s_0)
$(send, true)$	1	0	1 (s_2)
$(output, x_s \leq 4)$	0	0	0 (s_1)
$(output, x_s > 4)$	0	0	0
$(ack, true)$	1	1	1 (s_3)
$(output, x_s \leq 4)(send, true)$	0	0	0
$(output, x_s > 4)(send, true)$	0	0	0
$(output, x_s \leq 4)(output, x_s \leq 4)$	0	0	0
$(output, x_s \leq 4)(output, x_s > 4)$	0	0	0
$(output, x_s > 4)(output, x_s \leq 4)$	0	0	0
$(output, x_s > 4)(output, x_s > 4)$	0	0	0
$(output, x_s \leq 4)(ack, true)$	0	0	0
$(output, x_s > 4)(ack, true)$	0	0	0
$(send, true)(send, true)$	1	1	1
$(send, true)(output, x_s \leq 4)$	1	0	0 (s_4)
$(send, true)(output, x_s > 4)$	0	0	0
$(send, true)(ack, true)$	0	0	0
$(ack, true)(send, true)$	1	1	1
$(ack, true)(output, x_s \leq 4)$	1	1	1
$(ack, true)(output, x_s > 4)$	1	1	1
$(ack, true)(ack, true)$	1	1	1
$(send, true)(output, x_s \leq 4)(send, true)$	1	1	1
$(send, true)(output, x_s \leq 4)(output, x_s \leq 4)$	0	0	0
$(send, true)(output, x_s \leq 4)(output, x_s > 4)$	0	0	0
$(send, true)(output, x_s \leq 4)(ack, true)$	0	0	0

(a) $\sigma_1 = (ack, true)$ and $\sigma_2 = (output, x_s \leq 4)$



(b) A_4

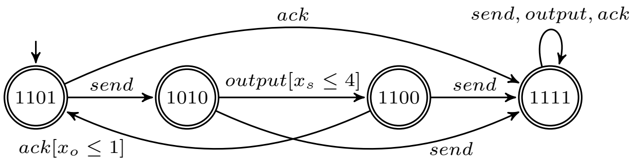
Fig. 19. Second timed assumption

A_4 is performed and the result is still negative with a positive counterexample $\pi = (send, true)(output, x_s \leq 1)(ack, x_o \leq 1)$. The normalized counterexample w.r.t. Σ_A is $\pi' = (send, true)(output, x_s \leq 4)(ack, true)$. A better counterexample $(\pi')_1^2 \cdot (\pi)_3^3 = (send, true)(output, x_s \leq 4)(ack, x_o \leq 1)$ is returned to TL*. The observation table is split according to the positive counterexample as shown in Fig. 20 (a), and the third timed assumption A_5 is constructed as shown in Fig. 20 (b).

In the fourth iteration, the result of the timed candidate query for A_5 is positive since INPUT $\parallel A_5 \models \varphi$ and OUTPUT $\models A_5$. By the AG-NC proof rule, the I/O system satisfies the timed property φ is concluded, and the verification framework is finished. Although the size of the assumption A_5 is bigger than OUTPUT in this

	λ	σ_1	σ_2	σ_3
λ	1	1	0	1 (s_0)
$(send, true)$	1	0	1	0 (s_2)
$(output, x_s \leq 4)$	0	0	0	0 (s_1)
$(output, x_s > 4)$	0	0	0	0
$(ack, x_o \leq 1)$	1	1	1	1 (s_3)
$(ack, x_0 > 1)$	1	1	1	1
$(output, x_s \leq 4)(send, true)$	0	0	0	0
$(output, x_s > 4)(send, true)$	0	0	0	0
$(output, x_s \leq 4)(output, x_s \leq 4)$	0	0	0	0
$(output, x_s \leq 4)(output, x_s > 4)$	0	0	0	0
$(output, x_s > 4)(output, x_s \leq 4)$	0	0	0	0
$(output, x_s > 4)(output, x_s > 4)$	0	0	0	0
$(output, x_s \leq 4)(ack, x_o \leq 1)$	0	0	0	0
$(output, x_s \leq 4)(ack, x_o > 1)$	0	0	0	0
$(output, x_s > 4)(ack, x_o \leq 1)$	0	0	0	0
$(output, x_s > 4)(ack, x_o > 1)$	0	0	0	0
$(send, true)(send, true)$	1	1	1	1
$(send, true)(output, x_s \leq 4)$	1	1	0	0 (s_4)
$(send, true)(output, x_s > 4)$	0	0	0	0
$(send, true)(ack, x_o \leq 1)$	0	0	0	0
$(send, true)(ack, x_o > 1)$	0	0	0	0
$(ack, x_o \leq 1)(send, true)$	1	1	1	1
$(ack, x_o > 1)(send, true)$	1	1	1	1
$(ack, x_o \leq 1)(output, x_s \leq 4)$	1	1	1	1
$(ack, x_o > 1)(output, x_s \leq 4)$	1	1	1	1
$(ack, x_o \leq 1)(output, x_s > 4)$	1	1	1	1
$(ack, x_o > 1)(output, x_s > 4)$	1	1	1	1
$(ack, x_o \leq 1)(ack, x_o \leq 1)$	1	1	1	1
$(ack, x_o \leq 1)(ack, x_o > 1)$	1	1	1	1
$(ack, x_o > 1)(ack, x_o \leq 1)$	1	1	1	1
$(ack, x_o > 1)(ack, x_o > 1)$	1	1	1	1
$(s_4)(send, true)$	1	1	1	1
$(s_4)(output, x_s \leq 4)$	0	0	0	0
$(s_4)(output, x_s > 4)$	0	0	0	0
$(s_4)(ack, x_o \leq 1)$	1	1	0	1
$(s_4)(ack, x_o > 1)$	0	0	0	0

(a) $\sigma_1 = (ack, x_o \leq 1)$, $\sigma_2 = (output, x_s \leq 4)$ and $\sigma_3 = (ack, x_o > 1)$



(b) A_5

Fig. 20. Third Timed Assumption

small example, our experiments in Section 5 shows that the proposed framework performs well in large scale systems.

4.3 Correctness and Termination

Theorem 4: AG-NC for ERAs is sound and complete.

Proof: Given two system models M_1, M_2 and a property φ represented by ERAs, to establish the soundness, we want to prove that $(M_1 \parallel A \models \varphi) \wedge (M_2 \models A) \rightarrow (M_1 \parallel M_2 \models \varphi)$. Let us prove this by contradiction. Assume $M_1 \parallel M_2 \not\models \varphi$, which implies that there exists a guarded word π such that $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_1)$, $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_2)$, and $\mathcal{L}(\pi) \subseteq \mathcal{L}(\overline{\varphi})$. Because $M_2 \models A$, therefore $\mathcal{L}(M_2) \subseteq \mathcal{L}(A)$, which implies $\mathcal{L}(\pi) \subseteq \mathcal{L}(A)$. Thus we can conclude that $M_1 \parallel A \not\models \varphi$ because $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_1)$, $\mathcal{L}(\pi) \subseteq \mathcal{L}(A)$, and $\mathcal{L}(\pi) \subseteq \mathcal{L}(\overline{\varphi})$, which contradicts to the promise $M_1 \parallel A \models \varphi$. To establish the completeness, given any two ERAs M_1 and M_2 and a property φ such that $M_1 \parallel M_2 \models \varphi$, we can always choose M_2 as the assumption A to satisfy the rule because $M_1 \parallel M_2 \models \varphi$ and $M_2 \models M_2$. $\square$

Theorem 5: The proposed learning-based compositional verification is sound and complete.

Proof: Our framework answers candidate queries needed by TL* according to the AG-NC proof rule, i.e., it concludes $M_1 \parallel M_2 \models \varphi$ when both $M_1 \parallel A \models \varphi$ and $M_2 \models A$ hold. By Theorem 4, the AG-NC proof rule is sound for ERAs, which implies our framework is sound. On the other hand, our framework returns a counterexample π only if $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_1 \parallel M_{\overline{\varphi}})$ and $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_2)$, which implies that $M_1 \parallel M_2 \not\models \varphi$. Given any two ERAs M_1 and M_2 and a property φ such that $M_1 \parallel M_2 \models \varphi$, our framework learns an assumption as M_2 in the worst case, which implies our framework is complete. □

Theorem 6: The proposed learning-based compositional verification terminates.

Proof: The proposed framework consists of two phases. The overall framework is terminating because both phases are terminating. In [13], it has been already proven that the untimed verification phase is terminating. Here, we only have to prove that the timed verification phase is terminating. In any iteration of the timed verification phase, our framework either concludes whether $M_1 \parallel M_2 \models \varphi$ holds and then terminates, or continues by providing counterexamples to the TL* algorithm. Since the target ERA to be learned by TL* is the weakest assumption A_w, by the correctness and termination of TL* in Theorem 1, it eventually constructs A_w in some iteration. In this iteration, A_w will pass the check $M_1 \parallel A_w \models \varphi$ according to the definition of the weakest assumption. We then check whether $M_2 \models A_w$ holds. If $M_2 \models A_w$, then $M_1 \parallel M_2 \models \varphi$ is concluded, and the framework terminates. If $M_2 \not\models A_w$, then $M_1 \parallel M_2 \not\models \varphi$ is concluded, and the framework also terminates and returns a counterexample $\mathcal{L}(\pi) \subseteq \mathcal{L}(M_2) \setminus \mathcal{L}(A_w)$. □

Generalization. The proposed compositional framework for verifying timed systems is presented in the context of two components. If a system consists of n components modeled by $M = \{M_1, M_2, \ldots, M_n\}$ where $n \geq 3$, one intuitive approach to generalize our framework is to partition the components into two higher level components to fit the AG-NC proof rule, e.g., if $n = 4$, we can obtain $H_1 = M_1 \parallel M_2$ and $H_2 = M_3 \parallel M_4$ and apply our approach on H_1 and H_2. Another way is to recursively apply the AG-NC proof rule, which constitutes the following generalized AG-NC proof rule for n components for $n \geq 2$.

$$
\begin{array}{rcl}
M_1 \parallel A_1 & \models & \varphi \\
M_2 \parallel A_2 & \models & A_1 \\
\vdots & & \\
M_{n-1} \parallel A_{n-1} & \models & A_{n-2} \\
M_n & \models & A_{n-1} \\
\hline
M_1 \parallel M_2 \parallel \ldots \parallel M_n & \models & \varphi
\end{array}
$$

Currently, we adopt the first approach to partition components into two groups. However, we found that the ways of partitioning components affect the verification result significantly. An investigation [14] reported that finding the best partition is hard. In our implementation, we use a heuristic that collects in H_1 the components containing the events specified in the property, and the heuristic yielded good performance in most of the cases in our experiments.

5 Experimental Results

The proposed learning-based compositional verification framework for timed systems has been implemented in the PAT model checker [36] such that PAT can automatically generate the assumptions for assume-guarantee reasoning when verifying timed systems modeled by ERAs. To demonstrate the feasibility and benefits of the framework, three applications were modeled and verified.

- **GSS**. A gas station system [18] consists of five components: one operator, one queue, one pump, and two customers. Two customers can fill gas at this gas station. Properties require that customers should be served in order and that each customer can start filling gas within three time units after payment.
- **FMS**. A flexible manufacturing system (FMS) [33] produces blocks with a cylindrical painted pin from raw blocks and raw pegs. It consists of fourteen devices: three conveyors, two mills, a lathe, a painting device, six robots, and an assembly machine. The devices are connected through nine buffers, and the capacity of each buffer is one. We modeled the FMS system in a constructive way such that four versions of models have been obtained, namely FMS-1 (the simplest one), FMS-2 (the medium one), FMS-3 (a complex one), and FMS-4 (the most complex one). Properties require that each buffer should not overflow or underflow and that output of each buffer should be within three time units after its input.
- **AIP**. The AIP manufacturing system [24] produces two products from two different materials. It consists of ten components: an I/O station, three transport units, two assembly stations, three external loops, and a central loop. Properties require that the routes of the two materials should be opposite and output of each loop should be within three time units after its input.

The ERA models of the applications, the verified properties, and the PAT model checker can be found in [1]. Tables 1-6 show the detailed verification results for each property of the three timed systems using the proposed approach and traditional monolithic timed model checking that constructs the timed global state space based on zone abstraction, respectively. The experimental results were obtained by running the PAT model checker on a 64-bit Windows 7 machine with a 23.4 GHz Intel(R) Core(TM) i7-2600 processor and 8 GB RAM.

As mentioned in the end of Section 4.3, for a system with more than two components, the way of partitioning them into two groups (M_1 and M_2) affects the verification result significantly. Thus, we also compare the results of applying our partition heuristic (c.f. Section 4.3) with those without any heuristic. We randomly generate 5 different partitions, and calculate the average results for the compositional approaches with and without our partition heuristic. The randomly generated partitions and the detailed verification results for each partition can be found in [1].

TABLE 1
Verification Results of GSS

			Monolithic			Compositional				Compositional+ Partition Heuristic				UPPAAL
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L\rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	Time (secs)
1	3	NO	16	0.01	0.1	55	1	0.01	0.8	56	1	0.01	0.8	0.03
2	3	NO	17	0.01	0.1	55	1	0.01	0.8	56	1	0.01	0.8	0.03
3	3	YES	29	0.01	0.2	97	19	0.25	2.0	33	1	0.01	0.8	0.02
Total				0.03				0.27				0.03		0.08

$\lvert C_\Sigma \rvert$: number of event-recording clocks; $\lvert L\rvert_{max}$: average of the maximum number of locations among all partitions during verification; $\lvert L_A\rvert$: average number of locations of the learned assumption among all partitions

TABLE 2
Verification Results of FMS-1

			Monolithic			Compositional				Compositional+ Partition Heuristic				UPPAAL
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L\rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	Time (secs)
1	3	YES	193	0.01	1.6	50	1	0.01	1.0	50	1	0.04	0.8	0.02
2	3	NO	9	0.01	0.1	108	2	0.01	1.7	8	1	0.01	0.4	0.02
3	3	YES	193	0.01	1.6	295	12	0.18	0.7	51	1	0.01	0.8	0.03
Total				0.03				0.20				0.05		0.07

TABLE 3
Verification Results of FMS-2

			Monolithic			Compositional				Compositional+ Partition Heuristic				UPPAAL
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L\rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	Time (secs)
1	6	YES	76,769	4.64	163.1	2,047	1	0.15	4.7	2,047	1	0.11	4.8	0.87
2	6	YES	76,769	4.64	165.2	5,621	8	13.10	25.7	2,252	1	0.16	5.7	0.27
3	6	NO	34	0.01	0.4	187	2	0.02	1.1	12	1	0.01	0.5	0.03
4	6	NO	48	0.01	0.6	1,462	2	0.06	2.3	17	1	0.01	0.9	0.03
5	6	YES	76,769	4.51	159.4	2,047	1	0.12	4.7	2,047	1	0.12	4.3	0.26
6	6	NO	4,270	0.21	7.6	2,347	3	1.00	3.5	4,815	4	0.50	13.3	0.63
Total				14.02				14.45				0.91		2.06

TABLE 4
Verification Results of FMS-3

			Monolithic			Compositional				Compositional+ Partition Heuristic				UPPAAL
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L\rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	$\lvert L\rvert_{max}$	$\lvert L_A\rvert$	Time (secs)	Mem (MB)	Time (secs)
1	6	YES	480,481	36.05	1052.1	2,265	1	0.15	7.4	2,265	1	0.15	5.9	5.02
2	6	YES	480,481	40.88	1065.4	6,801	6	13.61	26.2	2,211	1	0.17	4.1	5.19
3	6	NO	36	0.01	0.3	392	2	0.03	1.4	11	1	0.01	0.5	0.03
4	6	NO	52	0.01	0.7	1,679	2	0.09	5.6	16	1	0.01	0.9	0.04
5	6	YES	480,481	39.73	1051.7	2,864	3	0.49	8.3	2,472	1	0.16	6.4	1.53
6	6	YES	480,481	43.39	1065.1	6,624	16	15.01	19.8	11,789	1	0.95	24.1	0.06
7	6	YES	480,481	33.44	1065.0	6,624	16	7.81	14.3	5,400	1	0.41	11.8	0.05
Total				193.51				37.19				1.86		11.92

For the results of GSS and FMS-1 in Tables 1–2, the system size in terms of the number of locations is small and our compositional approach does not outperform the monolithic approach and even consumes more memory because of the overhead of learning iterations.

For the results of the FMS-2 system, as shown in Table 3, the compositional approach without the partition heuristic outperforms the monolithic in most of the cases expect Specs 2, 4, and 6. In Spec 2, the randomly generated partitions are not good, and it takes 8 times of candidate queries in average to learn the assumption, which is even worse than the monolithic approach. With our partition heuristic, it only takes 1 candidate query to learn the assumption, which significantly speeds up the verification process. In Specs 4 and 6, the properties are violated. If the verified property is violated, the monolithic verification might find a counterexample faster than the compositional approach because of the on-the-fly technique, which terminates the verification once a counterexample is found to avoid constructing the whole state space.

For the results of the FMS-3 system as shown in Table 4, the compositional approach without the partition heuristic

TABLE 5
Verification Results of FMS-4 (ROM: run out of memory)

			Monolithic			Compositional				Compositional+ Partition Heuristic				UPPAAL
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L \rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L \rvert_{max}$	$\lvert L_A \rvert$	Time (secs)	Mem (MB)	$\lvert L \rvert_{max}$	$\lvert L_A \rvert$	Time (secs)	Mem (MB)	Time (secs)
1	8	YES	–	–	ROM	34,008	1	4.46	70.9	34,008	1	4.18	74.8	ROM
2	8	YES	–	–	ROM	34,008	1	4.32	72.7	34,008	1	4.54	81.1	ROM
3	8	NO	62	0.01	1.0	2,276	2	0.20	8.2	17	1	0.01	0.9	0.04
4	8	NO	96	0.01	1.6	12,499	2	1.29	36.7	20	1	0.01	1.1	0.05
5	8	YES	–	–	ROM	78,955	31	476.84	158.5	36,680	1	5.19	77.8	ROM
6	8	NO	–	–	ROM	44,196	18	33.13	145.8	4,405	7	13.55	12.0	ROM
7	8	YES	–	–	ROM	–	–	–	ROM	30,144	1	4.78	71.2	ROM
8	8	NO	111	0.01	2.0	116,748	2	14.24	270.9	27	1	0.01	1.3	0.04
9	8	YES	–	–	ROM	59,410	37	1419.83	150.9	31,656	1	3.39	66.4	ROM
Total				N/A				N/A				35.66		N/A

TABLE 6
Verification Results of AIP

			Monolithic			Compositional				Compositional+ Partition Heuristic			
Spec	$\lvert C_\Sigma \rvert$	Valid?	$\lvert L \rvert_{max}$	Time (secs)	Mem (MB)	$\lvert L \rvert_{max}$	$\lvert L_A \rvert$	Time (secs)	Mem (MB)	$\lvert L \rvert_{max}$	$\lvert L_A \rvert$	Time (secs)	Mem (MB)
1	4	NO	137	0.01	1.7	211	1	0.02	1.2	210	1	0.02	1.1
2	4	NO	368	0.02	0.4	680	2	0.05	1.8	680	2	0.05	2.0
3	4	NO	651	0.03	1.0	474	1	0.03	1.6	435	1	0.03	1.4
4	4	NO	63	0.01	0.8	636	5	1.05	4.5	345	2	0.02	3.0
5	4	YES	104,651	29.59	231.4	2,301	1	0.15	6.2	2,301	1	0.15	6.0
6	4	YES	104,651	33.21	229.2	2,561	4	0.70	7.6	2,512	1	0.16	6.7
7	4	YES	86,051	22.03	190.9	2,088	3	0.25	6.3	2,189	1	0.15	5.8
8	4	YES	86,051	20.81	188.2	1,948	1	0.12	5.5	2,733	1	0.19	7.6
9	4	YES	104,651	26.50	227.7	15,538	45	87.70	54.0	2,605	1	0.25	8.2
10	4	NO	14	0.01	0.2	25	2	0.01	1.1	15	1	0.01	1.0
Total				132.22				89.03				1.03	

outperforms the monolithic one in all cases where the properties are satisfied because the learning iterations compensate for the large global state space such that the verification time and the memory usage are significantly reduced. In addition, with the partition heuristic, the verification time and memory usage are even further reduced dramatically.

For the results of the FMS-4 system as shown in Table 5, the monolithic approach cannot even finish the verification for each satisfied properties using 8 GB memory, while the compositional approach without the partition heuristic can finish the verification for all properties except for Spec 7. In the case of Spec 7, two randomly generated partitions are not good and cannot be verified by the compositional approach without the partition heuristic using 8 GB memory. With the partition heuristic, the total verification time only takes less than 36 seconds, and the maximal memory usage is less than 82 MB, which is a significant improvement.

The verification results for AIP are shown in Table 6. For Spec 9, the compositional approach without the partition heuristic performs seriously worse than the monolithic one because some generated partitions are very bad, which need 45 candidate queries in average (each of which requires model checking). Again, the partition heuristic improves the performance significantly. We can observe that the way of partitioning components really dominates the performance of the learning-based compositional verification.

We also compared the verification time between our approach and UPPAAL [2]; however, we do not list the verification time of UPPAAL for the AIP system because UPPAAL does not support events on transitions so that the AIP system cannot be modeled in UPPAAL. Our compositional approach with the partition heuristic outperforms UPPAAL in all cases. For FMS-4, UPPAAL cannot even verify the satisfied properties using 8 GB memory.

6 Related Work

Model checking [10], [32] suffers from the *state space explosion* problem, especially for timed systems. To alleviate the problem, Pnueli firstly proposed the assume-guarantee paradigm [31] to verify system components individually and use the individual verification results to deduce additional properties of the system. Clarke et al. [12] used interface processes to model the abstract environment for a component, which is much smaller than the real one, such that the state space is reduced. For formal verification that is not based on model checking, Xu et al. [37] proposed a proof system based on the assume-guarantee paradigm for verifying shared variable concurrent programs. Henzinger et al. [20] reported several case studies about applying assume-guarantee reasoning on real world systems.

For model checking of timed systems, Laroussinie et al. [23] proposed a technique to transform the property w.r.t. individual components one by one using quotient

$$\frac{\begin{array}{c} M_1 \parallel A_{M_1} \models \varphi \\ \vdots \\ M_n \parallel A_{M_n} \models \varphi \\ \overline{A_{M_1}} \parallel \overline{A_{M_2}} \parallel \cdots \parallel \overline{A_{M_n}} \models \varphi \end{array}}{M_1 \parallel M_2 \parallel \cdots \parallel M_n \models \varphi}$$

Fig. 21. The AGC proof rule

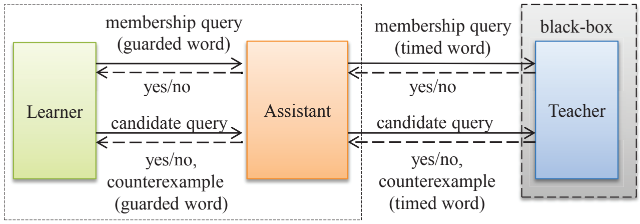


Fig. 22. Interaction between TL^*_{sg} and Teacher

construction such that the global state space need not be generated, but the minimization for the transformed property is needed because repeated quotient constructions lead to an explosion on the transformed property.

Cobleigh et al. [13] proposed a framework that generates the abstract environment of components automatically using the L^* algorithm [5]. This work is a pioneer of automating the untimed compositional verification based on learning techniques. Consequently, several improvements [9], [15], [35] have been proposed to further reduce the complexity. These improvements focus on reducing the size of the alphabet during learning, which dominates the time complexity of the membership query in the L^* algorithm. Inspired by [13], Lin and Hsiung [26] proposed a compositional synthesis framework which can help a system designer to automatically synthesize component models to satisfy the given property based on the L^* algorithm and causality semantics.

Barringer et al. [6] also used the L^* algorithm to learn assumptions automatically for AGR with the circular and symmetric (AGC) proof rule as shown in Fig. 21. In contrast to the AG-NC proof rule, the components of the system do not have to be grouped when applying the AGC proof rule. However, the number of premises to be proved in the AGC proof rule and the number of assumptions to be learned increase linearly with the number of the components. To reduce the number of premises and assumptions, Nam and Alur [30] proposed a method to automatically group the n components into m groups, where $m < n$, by reducing the problem to the *hypergraph partition problem*. Alur et al. [4] proposed a symbolic implementation of AGR for the AGC proof rule. They used *Binary Decision Diagrams* (BDD) [8] to symbolically encode the observation table maintained by the L^* algorithm.

However, the works based on the L^* algorithm mentioned above are only applicable for untimed systems. To infer timed assumptions for AGR, a learning algorithm for timed models is needed. Grinchtein et al. [16] proposed three algorithms TL^*_{sg}, TL^*_{nsg}, and TL^*_{s} for learning ERAs. Their learning algorithms deal with timed words, while our TL^* algorithm deals with guarded words. Theoretically, they are not comparable since the target words to be dealt with are different. More specifically, the learning problem handled by Grinchtein et al. [16] is more difficult because the interface between the learning algorithm and the Teacher is based on timed words and the learning algorithm has to actively infer the time condition of each event.

We briefly introduce the TL^*_{sg} algorithm here to see how TL^*_{sg} and TL^* perform in the context of the goal of this work, compositional verification. Grinchstein et al.'s TL^*_{sg} algorithm consists of two components, namely a learner and an assistant, as shown in Fig. 22. The learner acts almost like the L^* algorithm except that it interacts with the assistant instead of the Teacher and asks membership queries for guarded words instead of untimed words. The assistant translates a membership query for a guarded word into several membership queries of timed words and forwards these translated membership queries to the Teacher. After getting the results of membership queries of timed words from the Teacher, the assistant returns the result of the membership query for the guarded word to the learner according to the results from the Teacher.

Let us use the example in Section 3.2 for illustration. Suppose the timed language to be learned is accepted by the ERA as shown in Fig. 6 (a). The TL^*_{sg} algorithm assumes that the maximum constant of the clock guard, K, is known (here, $K = 3$). Note that our TL^* algorithm does not make this assumption. For each event, TL^*_{sg} actively guesses all possible guards for the event. In this example, all the possible guards for event a are as shown in the first 2–11 rows in Fig. 23. For the membership query of the guarded word $(a, 1 \leq x_a \leq 3)$, the assistant performs the membership queries for the following timed words: $(a, 0)$, $(a, 1)$, $(a, 2)$, $(a, 3)$, and $(a, 4)$. According to the results from the Teacher, the assistant finds that $1 \leq x_a \leq 3$ is not the sharpest guard [16] for event a (the sharpest guard is $x_a = 1$). Thus, the assistant answers “no” to the learner for the guarded word $(a, 1 \leq x_a \leq 3)$. The final closed observation table is as shown in Fig. 23, and the final learned ERA is as shown in Fig. 10 (b).

From the example, we can observe that the number of membership queries increases exponentially to K, the maximum constant [16]. If we change K from 3 to 1000 in this example, Grinchstein et al.'s TL^*_{sg} algorithm requires a huge number of membership queries, which makes it unsuitable to be used in compositional verification setting. This is because the learning problem in the context of compositional verification is not as difficult as that in [16]. In our setting of using TL^* to learn timed assumptions for AGR, the Teacher can be the most friendly one since the component models are transparent to model checking.

Gheorghiu et al. [15] used the abstraction-refinement paradigm [11] to infer the necessary alphabet of the untimed assumption A for AGR. Howar et al. [22] also used the paradigm on the alphabet for inferring abstract

	λ	$(a, x_a = 3)$
λ	1	0 (s_0)
$(a, x_a = 0)$	0	0 (s_1)
$(a, x_a = 1)$	1	1 (s_2)
$(a, x_a = 2)$	0	0
$(a, x_a = 3)$	0	0
$(a, x_a \geq 3)$	0	0
$(a, 0 \leq x_a \leq 1)$	0	0
$(a, 1 \leq x_a \leq 2)$	0	0
$(a, 2 \leq x_a \leq 3)$	0	0
$(a, 0 \leq x_a \leq 2)$	0	0
$(a, 1 \leq x_a \leq 3)$	0	0
$(a, 0 \leq x_a \leq 3)$	0	0
$(a, x_a = 0)(a, x_a = 0)$	0	0
$(a, x_a = 0)(a, x_a = 1)$	0	0
$(a, x_a = 0)(a, x_a = 2)$	0	0
$(a, x_a = 0)(a, x_a = 3)$	0	0
$(a, x_a = 0)(a, x_a \geq 3)$	0	0
$(a, x_a = 0)(a, 0 \leq x_a \leq 1)$	0	0
$(a, x_a = 0)(a, 1 \leq x_a \leq 2)$	0	0
$(a, x_a = 0)(a, 2 \leq x_a \leq 3)$	0	0
$(a, x_a = 0)(a, 0 \leq x_a \leq 2)$	0	0
$(a, x_a = 0)(a, 1 \leq x_a \leq 3)$	0	0
$(a, x_a = 0)(a, 0 \leq x_a \leq 3)$	0	0
$(a, x_a = 1)(a, x_a = 0)$	0	0
$(a, x_a = 1)(a, x_a = 1)$	0	0
$(a, x_a = 1)(a, x_a = 2)$	0	0
$(a, x_a = 1)(a, x_a = 3)$	1	0
$(a, x_a = 1)(a, x_a \geq 3)$	0	0
$(a, x_a = 1)(a, 0 \leq x_a \leq 1)$	0	0
$(a, x_a = 1)(a, 1 \leq x_a \leq 2)$	0	0
$(a, x_a = 1)(a, 2 \leq x_a \leq 3)$	0	0
$(a, x_a = 1)(a, 0 \leq x_a \leq 2)$	0	0
$(a, x_a = 1)(a, 1 \leq x_a \leq 3)$	0	0
$(a, x_a = 1)(a, 0 \leq x_a \leq 3)$	0	0

Fig. 23. Observation Table Constructed by TL^*_{sg}

automata with respect to given concrete behavior such that determinism is preserved. Our TL* algorithm may benefit from the abstraction refinement paradigm if the alphabet of the ERA to be learned can be smaller.

7 Conclusion and Future Work

Assume-guarantee reasoning (AGR) can help to alleviate the state explosion problem. However, constructing assumptions for AGR usually requires human creativity and experience. To automate compositional verification for timed systems, we propose a framework consisting of a learning algorithm and a timed teacher. The algorithm, TL*, automatically learns the timed assumption by asking membership and candidate queries, and the timed teacher answers the queries based on the AG-NC proof rule of AGR. With the proposed framework, compositional verification for timed systems is fully automated, and the state explosion problem can be effectively alleviated. In the future, we plan to extend the TL* algorithm with one-phase learning instead of two phases and to investigate the differences between them. We also plan to use different techniques to generate the assumptions as well as to extend the framework using other proof rules of AGR.

Acknowledgment

We thank Dr. Dimitra Giannakopoulou for her helpful and constructive suggestions and the anonymous reviewers for their detailed review comments.

References

[1] https://sites.google.com/site/shangweilin/era-pat.

[2] http://www.uppaal.org/.

[3] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

[4] R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 548–562, 2005.

[5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[6] H. Barringer, D. Giannakopoulou, and C. S. Păsăreanu. Proof rules for automated compositional verification through learning. In *Proceedings of Workshop on Specification and Verification of Component-Based Systems*, pages 14–21, 2003.

[7] M. G. Bobaru, C. S. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 5123 of *LNCS*, pages 135–148, 2008.

[8] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[9] S. Chaki and O. Strichman. Optimized L*-based assume-guarantee reasoning. In *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 276–291, 2007.

[10] E. M. Clarke and E. A. Emerson. Design and sythesis of synchronization skeletons using branching time temporal logic. In *the Logics of Programs Workshop*, volume 131, pages 52–71, 1981.

[11] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 154–169, 2000.

[12] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Proceedings of Symposium on Logic in Computer Science (LICS)*, pages 353–362, 1989.

[13] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *LNCS*, pages 331–346, 2003.

[14] J.M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An investigation of decomposition for assume-guarantee reasoning. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 97–108, 2006.

[15] M. Gheorghiu, D Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In *Proceeding of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 292–307, 2007.

[16] O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. *Theoretical Computer Science*, 411(47):4029–4054, 2010.

[17] O. Grumberg and D. E. Long. Model checking and modular verification. In *CONCUR 1991*, volume 527 of *LNCS*, pages 250–265, 1991.

[18] D. Helmbold and D. Luckhan. Debugging Ada tasking programs. *IEEE Software*, 2(2):47–57, 1985.

[19] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112:273–337, 1994.

[20] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 1427 of *LNCS*, pages 440–451, 1998.

[21] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[22] F. Howar, B. Steffen, and M. Merten. Automata learning with automated alphabet abstraction refinement. In *Proceeding of International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 6538 of *LNCS*, pages 263–277, 2011.

[23] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proceedings of International Conference*

on Concurrency Theory (CONCUR)*, volume 962 of *LNCS*, pages 27–41, 1995.

[24] R. J. Leduc, M. Lawford, and P. C. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4):654–668, 2006.

[25] S.-W. Lin, É. André, J. S. Dong, J. Sun, and Y. Liu. An efficient algorithm for learning event-recording automata. In *Proceedings of International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 6996 of *LNCS*, pages 463–472, 2011.

[26] S.-W. Lin and P.-A. Hsiung. Counterexample-guided assume-guarantee synthesis through learning. *IEEE Transactions on Computers*, 60(5):734–750, 2011.

[27] S.-W. Lin and P.-A. Hsiung. Model checking prioritized timed systems. *IEEE Transactions on Computers*, 61(5):843–856, 2012.

[28] S.-W. Lin, P.-A. Hsiung, C. H. Huang, and Y. R Chen. Model checking prioritized timed automata. In *Proceedings of International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 3707 of *LNCS*, pages 370–384, 2005.

[29] S.-W. Lin, Y. Liu, J. Sun, J. S. Dong, and É. André. Automatic compositional verification of timed systems. In *Proceedings of International Symposium on Formal Methods (FM)*, volume 7436 of *LNCS*, pages 272–276, 2012.

[30] W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In *Proceedings of International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *LNCS*, pages 170–185, 2006.

[31] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, pages 123–144, 1985.

[32] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *the International Symposium on Programming*, volume 137, pages 337–351, 1982.

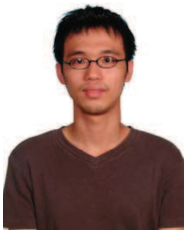
[33] M. H. Queiroz, J. E. R. Cury, and W. M. Wonham. Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4), 2005.

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

[35] N. Sinha and E. M. Clarke. SAT-based compositional verification using lazy learning. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 4590 of *LNCS*, pages 39–54, 2007.

[36] J. Sun, Y Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 709–714, 2009.

[37] Q. Xu, W. P de Roever, and J. He. The rely-guarantee method for verifying shared variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.



Shang-Wei Lin received his Bachelor degree in management information system from National Chung Cheng University (Taiwan) in 2003, and his Ph.D. degree in Computer Science and Information Engineering from National Chung Cheng University in 2010. From 2011 to 2012, he was a research fellow in School of Computing, National University of Singapore (NUS). Currently, he is a research scientist in Temasek Laboratories at NUS. His research interests include formal verification, formal synthesis, scheduling, embedded software synthesis and verification, and component-based object-oriented application frameworks for real-time embedded systems. More details can be found at https://www.sites.google.com/site/shangweilin/.

Étienne André received his Master degree (with honors) from the Université de Rennes 1 (France) in 2007, and his Ph.D. degree in computer science from École Normale Supérieure de Cachan (France) in 2010. He was then a research fellow in Prof. Dong Jin Song's team in the National University of Singapore for 9 months. Since September 2011, he has been an Associate Professor in the Laboratoire d'Informatique de Paris Nord, in the University of Paris 13 (Sorbonne Paris Cité) in France. His current research interests include the specification and verification of real-time concurrent systems. More details about his information and research background can be found at http://lipn.univ-paris13.fr/~andre/.

Yang Liu graduated in 2005 with a Bachelor of Computing in the National University of Singapore (NUS). In 2010, he obtained his Ph.D. and continued with his post doctoral work in NUS. Since 2012, he joined Nanyang Technological University as an Assistant Professor. His research focuses on software engineering, formal methods and security. Particularly, he specializes in software verification using model checking techniques. This work led to the development of a state-of-the-art model checker, Process Analysis Toolkit.

Jun Sun received Bachelor and Ph.D. degrees in computing science from National University of Singapore (NUS) in 2002 and 2006. In 2007, he received the prestigious LEE KUAN YEW postdoctoral fellowship in School of Computing of NUS. Since 2010, he joined Singapore University of Technology and Design (SUTD) as an Assistant Professor. He was a visiting scholar at MIT from 2011-2012. Jun's research focuses on software engineering and formal methods, in particular, system verification and model checking. He is the co-founder of the PAT model checker.

Jin Song Dong received Bachelor and Ph.D. degrees in Computing from University of Queensland in 1992 and 1996. From 1995-1998, he was Research Scientist at CSIRO in Australia. Since 1998 he has been in the School of Computing at the National University of Singapore (NUS) where he is currently Associate Professor and a member of PhD supervisors at NUS Graduate School. He is on the editorial board of Formal Aspects of Computing and Innovations in Systems and Software Engineering. His research interests include formal methods, software engineering, pervasive computing and semantic technologies.