The Opacity of Real-Time Automata

Lingtai Wang, Naijun Zhan, and Jie An

***Abstract*—Opacity is an important property on information flow to guarantee that a system under attack keeps its "secrets", possibly subsets of traces (language-based opacity) or subsets of states (state-based opacity), opaque to the outside intruder with partial observability. In this paper, we investigate the opacity problems of real-time automata (RTA), which is a popular model for real-time systems. In order to prove that the language-opacity problem of RTA is decidable, we introduce the notion of trace-equivalence and then translate RTA into finite-state automata (FA) with timed alphabets. Besides, we also introduce the notions of partitioned timed alphabet and language to guarantee trace equivalence is preserved by complementation and product operations over FA with timed alphabets. Thus, our decision procedure can be sketched as follows: first, translate the RTA to model a system under attack and the RTA to specify the secret behavior of the system into FA, respectively; then, compute another FA, which accepts all traces accepted by the first FA, but not by the second one; afterwards, project these FA onto the given observable set; finally, unify the alphabets of these FA such that for any two timed actions with the same event, their time parts do not have any overlap. Thus, whether the original system is language-opaque with respect to the secret RTA and the observable set is reduced to the inclusion problem of regular languages. Similarly, we can show decidability of initial-opacity of RTA.**

***Index Terms*—Decidability, initial-state opacity, language-opacity, real-time automata (RTA), trace-equivalence.**

I. Introduction

As NETWORK communications and online services are ubiquitous in modern life, security and privacy have become more and more important. Opacity is an information flow property aiming at keeping the "secret" of a system opaque to its outsider (called the intruder, who is believed to know everything about the structure of the system, but only has partial observability over it). Once the intruder has observed the execution, he can get an estimation whether the execution belongs to the secret. There are two types of secrets: subsets of traces and subsets of states. Opacity properties are divided into language-based opacity and state-based opacity, according to in which type the secrets are.

A system is called *language-opaque* if an intruder with partial observability can never determine whether a trace of the system is secret no matter what he has observed, while a system is called *initial-state opaque* if the intruder is unable to determine whether it starts from a secret state.

The notion of *opacity* was first introduced in the context of security protocols in [1]. Opacity has then been modeled in Petri nets (PNs), for example, in [2] it is proved that the problems of initial-opacity, final-opacity and always-opacity are all decidable if the set of all markings reachable from any initial markings is finite. In [3], an approach based on basis reachability graph was proposed so that initial-state opacity of bounded PNs can be verified. Bryans *et al.* [4] generalized opacity to labeled transition systems and proved that opacity is undecidable in such systems. To this end, they further proposed a decidable approximation to the original opacity, which was named as under/over-opacity. In the framework of automata, verification of initial-opacity is PSPACE-complete [5]. State estimators are constructed in [5]–[8] for verification of different kinds of state-based opacity. Probabilistic models are also taken into consideration, such as [9]–[12].

In [13], the notion of opacity was first extended to time settings, with the result that the language-based opacity problem is already undecidable for a very restrictive class of event recording timed automata (ERA).

As time is an important attack vector against secure systems, we still would like to consider the language-based and initial-state-based opacity problems on a timed model. In this paper, we concentrate on real-time automata (RTA) [14], a subclass of timed automata with a single clock to be reset at each transition, which can be regarded as finite automata with time information for each transition. RTA is a popular model for real-time systems. Note that RTA is not comparable with ERA as pointed out in [14].

In this paper, we show that the language-opacity problem of RTA is decidable by reduction to the inclusion problem of regular languages, which is decidable from automata theory [15]. The basic idea can be sketched as follows. First, we introduce the notion of *trace-equivalence* between languages of RTA and finite-state automata (FA) with timed alphabets. Second, in order to guarantee trace-equivalence to be preserved by the complementation and product operation over FA with timed alphabets, we introduce the concepts of partitioned timed alphabet and partitioned language. So refined FA with partitioned timed alphabet are constructed. Third, we define a projection operation on FA onto the given observable set Σ_o, by removing all unobservable transitions and

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. Date of current version October 18, 2018. This work was supported in part by the NSFC under Grant 61625206, Grant 61732001, and Grant 61472279, in part by the "973 Program" under Grant 2014CB340701, and in part by the CAS/SAFEA International Partnership Program for Creative Research Teams. This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Naijun Zhan.)*

L. Wang and N. Zhan are with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Computer Science and Control, University of Chinese Academy of Sciences, Beijing 100190, China (e-mail: wanglt@ios.ac.cn; znj@ios.ac.cn).

J. An is with the School of Software Engineering, Tongji University, Shanghai 201804, China (e-mail: 1510796@tongji.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857363

merging their time durations into the subsequent observable transition.

Given $\mathcal{A}$ (the system under attack), $\mathcal{A}_s$ (specifying secret behaviors), and Σ_o (the observables), the whole process is shown below, where 0 stands for translation from RTA to FA, 1 for refinement by partition, 2 for complementation, 3 for production, and 4 for projection.

$$\begin{tikzcd}
\mathcal{A}_s \arrow[r, "0"] & \mathcal{B}_s \arrow[r, "1\&2"] & \mathcal{B}_s^c \arrow[r, "3"] & \mathcal{B}_{ns} \arrow[r, "4"] & \mathcal{B}_{ns} \uparrow_{\Sigma_o} \arrow[r, "1\&2"] & (\mathcal{B}_{ns} \uparrow_{\Sigma_o})^c \arrow[d, "3"] \\
\mathcal{A} \arrow[r, "0"] & \mathcal{B} \arrow[urr, "1\&3"] \arrow[rrr, "4"] & & & \mathcal{B} \uparrow_{\Sigma_o} \arrow[r, "1\&3"] & \mathcal{B}_{\text{final}}
\end{tikzcd}$$

Among those automata, $\mathcal{B}_{ns}$ accepts the language trace-equivalent to $L(\mathcal{A}) \setminus L_f(\mathcal{A}_s)$, and $\mathcal{B}_{\text{final}}$ accepts the language trace-equivalent to $P_{\Sigma_o,t}(L(\mathcal{A})) \setminus (P_{\Sigma_o,t}(L(\mathcal{A})) \setminus L_f(\mathcal{A}_s))$. Thus, the original $\mathcal{A}$ is language-opaque with respect to $L_f(\mathcal{A}_s)$ and Σ_o iff $L_f(\mathcal{B}_{\text{final}})$ is empty.

Besides, the decidability of the initial-state opacity of RTA can be proved by reduction to the inclusion problem of regular languages similarly.

A. Related Work

The technique used in this paper is similar to [14] to establish Kleene theorem and pumping lemma for RTA. In [14], the Floyd–Warshall algorithm and partition are used for transforming an augmented RTA into a stuttering-free one and then into a deterministic one, for a given RTA, so that the resulted RTA is closed under complementation, which can correspond to an FA. In this paper, we consider the opacity problems of RTA. To this end, partition is used to eliminate overlap of time domains so that a trace-equivalence between RTA and FA can be preserved by their product and complementation operations; in addition, the Floyd–Warshall algorithm is utilized only once for merging successive unobservable transitions and constructing the projection of FA. So, the complexity of Dima's [14] approach is much higher than ours, with his approach, the number of states in the stuttering-free one is $n = 2^{|\Sigma|} \cdot |S|$, and the number of states in the deterministic one is at most 3^n.

B. Organization

The remainder of this paper is organized as follows. In Section II, we recall preliminaries including FA, regular expressions, RTA, and the problems of language/initial-opacity of RTA. Translation from RTA to FA preserving trace-equivalence is introduced in Section III. Section IV provides the method to project an FA obtained in the previous section onto an observable alphabet. In Section V, we come to the conclusion that the problems of language/initial-opacity are decidable for RTA, and provide a very simple example for illustration. A prototypical implementation is presented in Sections VI and VII concludes this paper.

II. Preliminaries

We use $\mathbb{R}_{\geq 0}$, $\mathbb{Q}_{\geq 0}$, and $\mathbb{N}$ to denote the set of non-negative real numbers, non-negative rational numbers, and natural numbers, respectively.

Let Σ, a set of events, be the *alphabet*. A *word* or *string* over Σ is a finite sequence $w = \sigma_1\sigma_2 \cdots \sigma_n$, where $\sigma_i \in \Sigma$ for $i = 1, 2, \ldots, n$. $|w| = n$ is the length of w. ε is the empty word, with $|\varepsilon| = 0$. Σ^* is the set of all the finite words over Σ including ε. L is a *language* over Σ if $L \subseteq \Sigma^*$.

Commonly used operations on languages include union, intersection, and difference as in set theory, as well as concatenation, Kleene closure and projection defined below.

1) *Concatenation:* Let $L_1, L_2 \subseteq \Sigma^*$, the concatenation $L_1 \cdot L_2 = \{s_1 \cdot s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}$. The "·" can be omitted if no confusion.
2) *Kleene Closure:* Let $L \subseteq \Sigma^*$, and $L^0 = \{\varepsilon\}$, $L^k = (L^{k-1})L$ for $k > 0$, then the Kleene closure of L is $L^* = \bigcup_{k\in\mathbb{N}} L^k = \{\varepsilon\} \cup L \cup LL \cup \cdots$.
3) *Projection:* Given Σ and a subset $\Sigma_o \subseteq \Sigma$, we can define a projection $P_o : \Sigma^* \to \Sigma_o^*$, where

$$P_o(\varepsilon) = \varepsilon$$
$$P_o(\sigma s) = \begin{cases} \sigma P_o(s), & \text{if } \sigma \in \Sigma_o \\ P_o(s), & \text{otherwise} \end{cases}, \text{ for } \sigma \in \Sigma \text{ and } s \in \Sigma^*.$$

Given any $B \subseteq \Sigma^*$ and $C \subseteq \Sigma_o^*$, the image of B under P_o is $P_o(B) = \{P_o(s) \mid s \in B\} \subseteq \Sigma_o^*$ and the inverse image of C under P_o is $P_o^{-1}(C) = \{s \in \Sigma^* \mid P_o(s) \in C\} \subseteq \Sigma^*$.

Consider the alphabet $\Sigma \times \mathbb{R}_{\geq 0}$. A *timed word* over Σ is a finite word over the alphabet $\Sigma \times \mathbb{R}_{\geq 0}$ with the form of $w_t = (\sigma_1, t_1)(\sigma_2, t_2) \ldots (\sigma_n, t_n)$, where $0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$, meaning that σ_i occurs at t_i successively for $1 \leq i \leq n$. $\text{TW}^*(\Sigma)$ denotes the set of all timed words over Σ. A subset of $\text{TW}^*(\Sigma)$ is a *timed language*. If $\Sigma_o \subseteq \Sigma$ is the observable alphabet, $P_{\Sigma_o,t}$ denotes the projection from $\text{TW}^*(\Sigma)$ into $\text{TW}^*(\Sigma_o)$. For example, if $w_t = (a, 2)(b, 3)(a, 5)(b, 8)$, $P_{\{b\},t}(w_t) = (b, 3)(b, 8)$ and $P_{\{a\},t}(w_t) = (a, 2)(a, 5)$.

A. Finite-State Automata and Regular Expressions

Automata is a kind of well-known and commonly used model to study discrete transition systems and their behaviors. FA are automata with finite states, including deterministic and nondeterministic ones.

Definition 1:

1) A deterministic finite-state automaton (DFA) is a five-tuple $A_d = (S, \Sigma, \delta, s_0, F)$, where:
   a) S is a finite set of states;
   b) Σ is a finite alphabet;
   c) $\delta : S \times \Sigma \to S$ is the transition relation, a partial function on $S \times \Sigma$;
   d) $s_0 \in S$ is the initial state;
   e) $F \subseteq S$ is the set of accepting states.
2) A nondeterministic finite-state automaton (NFA) is a five-tuple $\mathcal{A}_n = (S, \Sigma \cup \{\varepsilon\}, \delta, \text{Init}, F)$, where:
   a) S is a finite set of states;
   b) Σ is a finite alphabet;
   c) $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
   d) $\text{Init} \subseteq S$ is the set of initial states;
   e) $F \subseteq S$ is the set of accepting states.

Obviously, a DFA can be viewed as a special kind of NFA, where there is only one initial state, one or zero state in each $\delta(s, \sigma)$, and no ε-transition.

For an NFA $\mathcal{A}$, (s_1, σ, s_2) is called a σ-transition if $s_2 \in \delta(s_1, \sigma)$. Pre_σ and Post_σ denotes the set of states from which and to which are σ-transitions, respectively.

A *run* of $\mathcal{A}$ is either a single state s_0 where $s_0 \in \text{Init}$, or a sequence $s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} s_n$, where $n > 0$, $s_0 \in \text{Init}$, $\sigma_i \in \Sigma \cup \{\varepsilon\}$, and $s_i \in \delta(s_{i-1}, \sigma_{i-1})$ for $1 \leq i \leq n$. An *accepting run* is a run ending in a state $s_n \in F$.

The trace of a run ρ is a finite word over Σ, written as $\text{trace}(\rho)$. $\text{trace}(s_0)$ is ε, and the trace of the sequence from s_0 to s_n is a finite word obtained by projecting $\sigma_1\sigma_2 \ldots \sigma_n$ onto Σ, that is, the string $a_1 a_2 \ldots a_m$ obtained by removing each ε from $\sigma_1\sigma_2 \ldots \sigma_n$; hence the length of the trace is m, less than or equal to n.

Let $\text{Tr}(s_0)$ be the set of traces of runs from s_0, and $\text{Tr}(S_0)$ be the set of traces of runs from any state $s_0 \in S_0$.

The language generated by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of traces of runs of $\mathcal{A}$, i.e., $L(\mathcal{A}) = \text{Tr}(\text{Init})$; the language accepted by $\mathcal{A}$, denoted by $L_f(\mathcal{A})$, is the set of traces of accepting runs. A language is said to be *regular* if it can be accepted by a finite-state automaton.

Regular expressions is another way to define regular languages.

Definition 2: Regular expressions over alphabet Σ can be defined recursively as follows.

1) *Base Clause:* $\emptyset$, ε, $\sigma \in \Sigma$ are regular expressions, where $\emptyset$ denotes the empty set, ε denotes the set $\{\varepsilon\}$, and σ denotes the set $\{\sigma\}$ for $\sigma \in \Sigma$.
2) *Inductive Clause:* If r, r_1, and r_2 are regular expressions, so are $r_1 \cdot r_2$, $r_1 + r_2$, and r^*. $r_1 \cdot r_2$ denotes the concatenation of the languages defined by r_1 and r_2, $r_1 + r_2$ denotes the union of the two languages, and r^* denotes the Kleene closure of the language defined by r.
3) *External Clause:* Regular expressions can only be constructed by applying steps 1 and 2.

Theorem 1 (Kleene's Theorem): Any regular language is accepted by an FA; any language accepted by an FA is regular.

1) Complementation and Product Over DFA: Two automata are called *language-equivalent*, or *equivalent* for short, if they generate and accept the same languages. An NFA $\mathcal{A}_n = (S, \Sigma, \delta, \text{Init}, F)$ can be transformed into an equivalent DFA $\mathcal{A}_d = (S', \Sigma, \delta', \text{Init}', F')$ defined below. Let $\varepsilon R(s, \varepsilon)$ denote the set of states which are reachable from state s via no transitions or only ε-transitions, and $\varepsilon R(s, \sigma)$ the set of states which are reachable from state s via one σ-transition together with ε-transitions before and after it. Then in $\mathcal{A}_d$, $S' = 2^S$; $\delta'(S_1, \sigma) = \bigcup_{s_1 \in S_1} \varepsilon R(s_1, \sigma)$; $\text{Init}' = \varepsilon R(s_0, \varepsilon)$; and $F' = \{S_1 \mid S_1 \cap F \neq \emptyset\}$.

Consider a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$. The complement automaton $\mathcal{A}^{\text{comp}}$ which accepts $L_f(\mathcal{A})^c = \Sigma^* \setminus L_f(\mathcal{A})$ can be constructed as follows.

1) Augment S with a new state $s_{\text{new}} \notin S$.
2) Augment δ such that it becomes a total function, denoted as δ^{comp}. For all $(s, \sigma) \in S \times \Sigma$, if $\delta(s, \sigma)$ is defined, let $\delta^{\text{comp}}(s, \sigma) = \delta(s, a)$; if $\delta(s, \sigma)$ is not defined, let $\delta^{\text{comp}}(s, \sigma) = s_{\text{new}}$. Also $\delta(s_{\text{new}}, \sigma) = s_{\text{new}}$ for each $\sigma \in \Sigma$. After that Σ^* becomes the language generated, while the language accepted keeps unchanged.
3) Let $(S \setminus F) \cup \{s_{\text{new}}\}$ be the set of accepting states.

To sum up, $\mathcal{A}^{\text{comp}} = (S \cup \{s_{\text{new}}\}, \Sigma, \delta^{\text{comp}}, s_0, S \setminus F \cup \{s_{\text{new}}\})$.

Given two DFA $\mathcal{A}_1 = (S_1, \Sigma_1, \delta_1, s_{0,1}, F_1)$ and $\mathcal{A}_2 = (S_2, \Sigma_2, \delta_2, s_{0,2}, F_2)$ with $S_1 \cap S_2 = \emptyset$, the product of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}^p = \mathcal{A}_1 \times \mathcal{A}_2 = (S^p, \Sigma^p, \delta^p, s_0^p, F^p)$, defined as follows: $S^p = S_1 \times S_2$; $\Sigma^p = \Sigma_1 \cap \Sigma_2$; $\delta^p((s_1, s_2), \sigma) = (s_1', s_2')$ if $\delta(s_1, \sigma) = s_1'$ and $\delta(s_2, \sigma) = s_2'$, and is not defined otherwise; $s_0^p = (s_{0,1}, s_{0,2})$; $F^p = F_1 \times F_2$

$$L_f(\mathcal{A}_1 \times \mathcal{A}_2) = L_f(\mathcal{A}_1) \cap L_f(\mathcal{A}_2). \tag{1}$$

B. Real-Time Automata

RTA are very similar to classical automata despite their taking time into account as well. We can easily get an RTA by attaching time information to each transition of a given automaton.

Definition 3: An RTA is a six-tuple $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$, where:

S	is a finite set of states;
Σ	is a finite alphabet;
$\Delta \subseteq S \times \Sigma \times S$	is the transition relation;
$\text{Init} \subseteq S$	is the set of initial states;
$F \subseteq S$	is the set of accepting states;
$\mu : \Delta \to 2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\}$	is the time labeling function.

Transitions $(s_1, \sigma, s_2) \in \Delta$ are called σ-transitions. Pre_σ and Post_σ denotes the set of states from which and to which are σ-transitions, respectively.

A *run* of $\mathcal{A}$ is either a single initial state $s_0 \in \text{Init}$ or a finite sequence $\rho = s_0 \xrightarrow[\lambda_1]{\sigma_1} s_1 \xrightarrow[\lambda_2]{\sigma_2} \cdots \xrightarrow[\lambda_n]{\sigma_n} s_n$ where $n > 0$, $s_0 \in \text{Init}$, $(s_{i-1}, \sigma_i, s_i) \in \Delta$, and $\lambda_i \in \mu(s_{i-1}, \sigma_i, s_i)$ for $1 \leq i \leq n$.

The trace of a run ρ, denoted by $\text{trace}(\rho)$, is a timed word defined as follows: $\text{trace}(s_0) = \varepsilon_t$, where subscript "$t$" is used to emphasize the time factor; if $\rho = s_0 \xrightarrow[\lambda_1]{\sigma_1} s_1 \xrightarrow[\lambda_2]{\sigma_2} \ldots \xrightarrow[\lambda_n]{\sigma_n} s_n$, $\text{trace}(\rho) = (\sigma_1, t_1)(\sigma_2, t_2) \ldots (\sigma_n, t_n)$, where $t_i = \sum_{j=1}^{i} \lambda_j$ for $1 \leq i \leq n$.

Let $\text{Tr}(s_0)$ be the set of traces of runs from s_0, and $\text{Tr}(S_0)$ be the set of traces of runs from any state $s_0 \in S_0$. Then languages generated and accepted by $\mathcal{A}$ can be defined: $L(\mathcal{A}) = \text{Tr}(\text{Init}) = \bigcup_{s_0 \in \text{Init}} \text{Tr}(s_0)$, and $L_f(\mathcal{A}) = \{\text{trace}(\rho) \mid \rho$ starts from $s_0 \in S_0$ and ends in $s_f \in F\}$.

Example 1: Consider the two RTA in Fig. 1. In $\mathcal{A}_1$, $\text{Tr}(s_0) = \{\varepsilon_t\} \cup \{(a, t_a) \mid t_a \in [1, 2]\} \cup \{(a, t_a)(b, t_b) \mid t_a \in [1, 2], t_b - t_a \in [2, 3]\}$, and $\text{Tr}(s_3) = \{\varepsilon_t\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$. $\mathcal{A}_1$ generates $L(\mathcal{A}_1) = \text{Tr}(s_0) \cup \text{Tr}(s_3)$; and $\mathcal{A}_1$ accepts $L_f(\mathcal{A}_1) = \{(a, t_a) \cdot (b, t_b) \mid t_a \in [1, 2], t_b - t_a \in [2, 3]\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$.

In $\mathcal{A}_2$, let $a^* = \{\varepsilon\} \cup \bigcup_{k \geq 1} \{(a, t_1) \ldots (a, t_k) \mid t_i - t_{i-1} \in [1, 2]\}$ $(t_0 = 0)$, and $a^*b = a^* \cdot (b, t_b)$, where $t_b \in [3, 5]$. Then $\text{Tr}(s_0') = a^* \cup a^*b$; $L(\mathcal{A}_2) = \text{Tr}(s_0')$; and $L_f(\mathcal{A}_2) = a^*b$.

C. Language and Initial-State Opacity of RTA

Given an RTA $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$ and an observable alphabet $\Sigma_o \subseteq \Sigma$, although $L(\mathcal{A})$ is generated, intruders can only observe timed words in $P_{\Sigma_o, t}(L(\mathcal{A}))$.

Suppose we have a secret timed language, L_{secret}, over Σ. In this case, can intruders make sure that a trace of the system falls into the secret set L_{secret} according to what they have

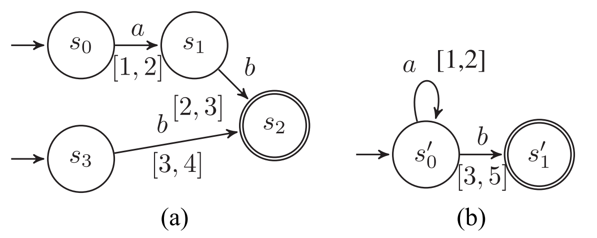

Fig. 1. RTA (a) $\mathcal{A}_1$ and (b) $\mathcal{A}_2$.

observed? This is considered in the language-opacity problem. Similarly, suppose we have some secret states, $S_{\text{secret}} \subseteq S$. The problem of initial-state opacity (initial-opacity for short) considers whether intruders can make sure that a trace starts from S_{secret}. Formally,

Definition 4: Given an RTA $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$, an observable alphabet $\Sigma_o \subseteq \Sigma$, a secret timed language L_{secret} over Σ and a secret set of states $S_{\text{secret}} \subseteq S$,

1) *Language-Opacity:* $\mathcal{A}$ is language-opaque with respect to L_{secret} and Σ_o iff for all $w_t \in L(\mathcal{A}) \cap L_{\text{secret}}$, $\exists w'_t \in L(\mathcal{A}) \setminus L_{\text{secret}}$ s.t.
$$P_{\Sigma_o,t}(w_t) = P_{\Sigma_o,t}(w'_t)$$
equivalently
$$P_{\Sigma_o,t}(L(\mathcal{A}) \cap L_{\text{secret}}) \subseteq P_{\Sigma_o,t}(L(\mathcal{A}) \setminus L_{\text{secret}}), \text{ or}$$
$$P_{\Sigma_o,t}(L(\mathcal{A})) \subseteq P_{\Sigma_o,t}(L(\mathcal{A}) \setminus L_{\text{secret}}).$$
2) *Initial-State Opacity:* $\mathcal{A}$ is initial-state opaque with respect to S_{secret} and Σ_o iff for all $s_0 \in \text{Init} \cap S_{\text{secret}}$ and all $w_t \in \text{Tr}(s_0)$, $\exists s'_0 \in \text{Init} \setminus S_{\text{secret}}$, $\exists w'_t \in \text{Tr}(s'_0)$ s.t.
$$P_{\Sigma_o,t}(w_t) = P_{\Sigma_o,t}\big(w'_t\big)$$
equivalently
$$P_{\Sigma_o,t}(\text{Tr}(\text{Init} \cap S_{\text{secret}})) \subseteq P_{\Sigma_o,t}(\text{Tr}(\text{Init} \setminus S_{\text{secret}})), \text{ or}$$
$$P_{\Sigma_o,t}(L(\mathcal{A})) \subseteq P_{\Sigma_o,t}(\text{Tr}(\text{Init} \setminus S_{\text{secret}})).$$
3) *The Language-Opacity Problem of RTA:* Is an RTA $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$ language-opaque with respect to some given secret timed language L_{secret} and $\Sigma_o \subseteq \Sigma$?
4) *The Initial-State Opacity Problem of RTA:* Is an RTA $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$ initial-state opaque with respect to some given secret set $S_{\text{secret}} \subseteq S$ and $\Sigma_o \subseteq \Sigma$?

Example 2: We still consider the automata $\mathcal{A}_1$ shown in Fig. 1(a). Let $\Sigma_o = \{b\}$, $L_{\text{secret}} = \{(a, t_a) \cdot (b, t_b) \mid 1 \leq t_a \leq 3 \wedge 2 \leq t_b \leq 5\}$, and $S_{\text{secret}} = \{s_3\}$.

First, $\mathcal{A}_1$ is not language-opaque with respect to L_{secret} and Σ_o. This is because there only exists $w_t = (a, 2) \cdot (b, 5)$ in $L(\mathcal{A}_1)$ satisfying $P_{\Sigma_o,t}(w_t) = (b, 5)$ and w_t is from L_{secret}. So if $P_{\Sigma_o,t}(w_t) = (b, 5)$, we can easily know that the trace is $w_t = (a, 2) \cdot (b, 5)$ and the run is $\rho = s_0 \xrightarrow[2]{a} s_1 \xrightarrow[3]{b} s_2$. Thus, the secret is exposed in this case.

From another perspective, $L(\mathcal{A}_1) \cap L_{\text{secret}} = \{(a, t_a) \cdot (b, t_b) \mid t_a \in [1, 2], t_b - t_a \in [2, 3]\}$, and $L(\mathcal{A}_1) \setminus L_{\text{secret}} = \{\varepsilon_t\} \cup \{(a, t_a) \mid t_a \in [1, 2]\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$. The projections are $P_{\Sigma_o,t}(L(\mathcal{A}_1) \cap L_{\text{secret}}) = \{(b, t) \mid t \in [3, 5]\}$, $P_{\Sigma_o,t}(L(\mathcal{A}_1) \setminus L_{\text{secret}}) = \{\varepsilon_t\} \cup \{(b, t) \mid t \in [3, 4]\}$, and $P_{\Sigma_o,t}(L(\mathcal{A}_1)) = \{\varepsilon_t\} \cup \{(b, t) \mid t \in [3, 5]\}$. So $\mathcal{A}_1$ is not language-opaque with respect to L_{secret} and Σ_o.

Second, $\mathcal{A}_1$ is initial-opaque with respect to S_{secret} and Σ_o. This is because for any $w_t = (b, t_b)$ in $\text{Tr}(s_3)$ with $3 \leq t_b \leq 4$, there always exists $w'_t = (a, 1) \cdot (b, t_b - 1) \in \text{Tr}(s_0)$ such that $P_{\Sigma_o,t}(w'_t) = P_{\Sigma_o,t}(w_t)$, whose corresponding run is $\rho = s_0 \xrightarrow[1]{a} s_1 \xrightarrow[t_b-1]{b} s_2$. Hence the secret is concealed.

From another perspective, $P_{\Sigma_o,t}(\text{Tr}(s_0)) = \{\varepsilon_t\} \cup \{(b, t) \mid t \in [3, 5]\}$, and $P_{\Sigma_o,t}(\text{Tr}(s_3)) = \{\varepsilon_t\} \cup \{(b, t) \mid t \in [3, 4]\}$. So $\mathcal{A}_1$ is initial-opaque with respect to $S_{\text{secret}} = \{s_3\}$ and $\Sigma_o = \{b\}$.

III. From RTA to FA

In this section, we construct FA from RTA, such that their languages are "equivalent" in some sense, which is called the *trace-equivalence relation* in this paper. Also, we put some restriction on the alphabet of the derived FA, called the *partitioned alphabet*, so that product and complementation operations still work on those FA.

A. Trace-Equivalence Relation

RTA and FA are similar in their structure, except that RTA maintain time information as well. Thus, it is natural to consider the transformation of an RTA into its corresponding FA, by attaching time information to the respective event in each transition, in order to utilize the existing results of FA.

We first introduce the concept of trace-equivalence between a timed language over a finite alphabet Σ and a language over the alphabet $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$ where each T_σ is a finite subset of $2^{\mathbb{R}_{\geq 0}}$. Σ_t is called a *timed alphabet*. Σ_t is finite as Σ is finite and T_σ is finite for each $\sigma \in \Sigma$.

Given a fixed word $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n)$, where $\Lambda_i \subseteq \mathbb{R}_{\geq 0}$ and $\Lambda_i \neq \emptyset$, we write $[w]$ to denote the set of all the timed words of the form $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n)$ with $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \leq n$.

Definition 5: Given L_1, a timed language over Σ, and L_2, a language over $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$, where each T_σ is a finite subset of $2^{\mathbb{R}_{\geq 0}}$, L_2 is said to be *trace-equivalent* to L_1, denoted by $L_2 \approx_{\text{tr}} L_1$, if:

1) $\varepsilon_t \in L_1$ iff $\varepsilon \in L_2$;
2) if any timed word $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n) \in L_1$, then there exists some $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n) \in L_2$ such that $w_t \in [w]$;
3) if $w = (\sigma_1, \Lambda_1) \cdot (a_2, \Lambda_2) \cdot \ldots \cdot (a_n, \Lambda_n) \in L_2$, then all timed words $w_t \in [w]$ are in L_1, i.e., $[w] \subseteq L_1$.

B. First Trial: Directly Derived FA

Given an RTA $\mathcal{A} = (S, \Sigma, \Delta, \text{Init}, F, \mu)$, an FA $\mathcal{B} = (S', \Sigma_t, \delta, \text{Init}', F')$ can be directly built which has the same sets of states, initial states and accepting states as $\mathcal{A}$, that is, $S' = S$, $\text{Init}' = \text{Init}$, and $F' = F$. The difference is that time information of each transition described by μ in $\mathcal{A}$ is transferred into its label in $\mathcal{B}$. Formally, $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$, where $T_\sigma = \{\mu(s, \sigma, s') \mid \exists (s, \sigma, s') \in \Delta\}$, and $\delta(s, (\sigma, \Lambda)) = \{s' \mid \exists (s, \sigma, s') \in \Delta \wedge \mu(s, \sigma, s') = \Lambda\}$. $\mathcal{B}$ constructed as above is called the *directly derived FA* from $\mathcal{A}$. $\mathcal{B}_0$ is constructed same as $\mathcal{B}$ except that its accepting states is set to be S. This means that $\mathcal{B}_0$ accepts $L(\mathcal{B})$.

Lemma 1: $L(\mathcal{B}) \approx_{\mathrm{tr}} L(\mathcal{A})$, and $L_f(\mathcal{B}) \approx_{\mathrm{tr}} L_f(\mathcal{A})$.

Proof: If $\varepsilon_t \in L(\mathcal{A})$, a possible run is $s_0 \in \mathrm{Init}$. Thus, $s_0 \in \mathrm{Init}'$. It means that $\mathcal{B}$ has a run s_0 whose trace is ε, i.e., $\varepsilon \in L(\mathcal{B})$. On the other hand, suppose $\varepsilon \in L(\mathcal{B})$, which implies there exists an initial state $s_0 \in \mathrm{Init}'$. It follows that $s_0 \in \mathrm{Init}$. Hence, $\varepsilon_t \in L(\mathcal{A})$.

If $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n) \in L(\mathcal{A})$, there exists a run of $\mathcal{A}$, say $\rho = s_0 \xrightarrow[t_1]{\sigma_1} s_1 \xrightarrow[t_2 - t_1]{\sigma_2} \cdots \xrightarrow[t_n - t_{n-1}]{\sigma_n} s_n$, such that $s_0 \in \mathrm{Init}$, $(s_{i-1}, \sigma_i, s_i) \in \Delta$ and $t_i - t_{i-1} \in \mu(s_{i-1}, \sigma_i, s_i)$ for $1 \le i \le n$. So there exists a run of $\mathcal{B}$ from $s_0 \in \mathrm{Init}'$, that is, $\rho' = s_0 \xrightarrow{(\sigma_1, \Lambda_1)} s_1 \xrightarrow{(\sigma_2, \Lambda_2)} \cdots \xrightarrow{(\sigma_n, \Lambda_n)} s_n$, where $\Lambda_i = \mu(s_{i-1}, \sigma_i, s_i)$ for $1 \le i \le n$. Thus, there exists a trace $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n) \in L(\mathcal{B})$ such that $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \le n$, in other words, $w_t \in [w]$.

Given a word $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n) \in L(\mathcal{B})$, there must be a run of the form $s_0 \xrightarrow{(\sigma_1, \Lambda_1)} s_1 \xrightarrow{(\sigma_2, \Lambda_2)} \cdots \xrightarrow{(\sigma_n, \Lambda_n)} s_n$, where $s_0 \in \mathrm{Init}'$. Hence $(s_{i-1}, \sigma_i, s_i) \in \Delta$ and $\mu(s_{i-1}, \sigma_i, s_i) = \Lambda_i$ for $1 \le i \le n$. It follows that for any t_1, t_2,..., t_n such that $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \le n$, there exists a run $s_0 \xrightarrow[t_1]{\sigma_1} s_1 \xrightarrow[t_2 - t_1]{\sigma_2} \cdots \xrightarrow[t_n - t_{n-1}]{\sigma_n} s_n$; therefore, $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n) \in L(\mathcal{A})$. Hence, $[w] \subseteq L(\mathcal{A})$.

The proof for $L_f(\mathcal{B}) \approx_{\mathrm{tr}} L_f(\mathcal{A})$ is similar, except that the considered run should end in an accepting state. ■

However, such trace-equivalence relationship may not be preserved by intersection and complementation operations. For instance, let $\Sigma = \{a\}$ and $\Sigma_t = \{(a, [2, 4]), (a, [3, 5])\}$, and $L_{11} = \{(a, t) \mid t \in [3, 5]\}$, $L_{12} = \{(a, t) \mid t \in [2, 4]\}$, $L_{21} = \{(a, [3, 5])\}$, and $L_{22} = \{(a, [2, 4])\}$. Clearly, $L_{21} \approx_{\mathrm{tr}} L_{11}$ and $L_{22} \approx_{\mathrm{tr}} L_{12}$, but it can easily follow that $L_{21} \cap L_{22}$ is not trace-equivalent to $L_{11} \cap L_{12}$, since $L_{11} \cap L_{12} = \{(a, t) \mid t \in [3, 4]\}$, while $L_{21} \cap L_{22} = \emptyset$. This is because $(a, [2, 4])$ and $(a, [3, 5])$ are different events in the timed alphabet Σ_t, but the actual timed words they represent overlap. As for the complementation operation, we only choose and compare subsets of words/timed words with length 1. Timed words in $\mathrm{TW}^*(\Sigma) \setminus L_{11}$ with length 1 is $\{(a, t) \mid t \in [0, 3) \cup (5, +\infty)\}$, while words in $\Sigma_t^* \setminus L_{21}$ with length 1 is $\{(a, [2, 4])\}$. This is because the union of $[3, 5]$ and $[2, 4]$ does not cover $\mathbb{R}_{\ge 0}$ and the intersection of $[3, 5]$ and $[2, 4]$ is not empty.

C. Partitioned Timed Alphabet and Partitioned Language

As a consequence, restrictions should be placed on the timed alphabet. Suppose the timed alphabet under consideration is $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$. There are two main restrictions on each T_σ.

1) Any two different elements of T_σ should be disjoint.

2) The union of all elements of T_σ should be equal to $\mathbb{R}_{\ge 0}$.

Hence the concept of *partition* in mathematics can be exploited here. A partition of a nonempty set B is a set of B's nonempty subsets satisfying each element $x \in B$ is in one and only one of those subsets. T_σ satisfies the two restrictions above if it is a partition of $\mathbb{R}_{\ge 0}$. We introduce the definition of *partitioned alphabet* and *partitioned language* below.

Definition 6: A finite timed alphabet $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$ is called *partitioned* if for any $\sigma \in \Sigma$, T_σ is a partition of $\mathbb{R}_{\ge 0}$.

A language L over a timed alphabet Σ_t is called *partitioned* if Σ_t is a partitioned alphabet.

Partitioned languages can guarantee that the relation of trace-equivalence is preserved by language complementation and intersection.

Lemma 2: If L_1 is a timed language over Σ, L_2 is a partitioned language over $\Sigma_t = \bigcup_{\sigma \in \Sigma} (\{\sigma\} \times T_\sigma)$ with $L_2 \approx_{\mathrm{tr}} L_1$, then we have $(\Sigma_t^* \setminus L_2) \approx_{\mathrm{tr}} (\mathrm{TW}^*(\Sigma) \setminus L_1)$.

Proof: First, $\varepsilon_t \in \mathrm{TW}^*(\Sigma) \setminus L_1 \Leftrightarrow \varepsilon_t \notin L_1 \Leftrightarrow \varepsilon \notin L_2 \Leftrightarrow \varepsilon \in \Sigma_t^* \setminus L_2$.

Second, suppose $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n) \in \mathrm{TW}^*(\Sigma) \setminus L_1$, then there must be a unique $\Lambda_i \in T_{\sigma_i}$ containing $t_i - t_{i-1}$ for each σ_i (t_0 is deemed to be 0 here), as T_{σ_i} is a partition of $\mathbb{R}_{\ge 0}$. Thus, it follows $(\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n) \in \Sigma_t^*$. We can easily know $(\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n)$ is not in L_2, otherwise w_t would be in L_1, which is a contradiction. So $(\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n)$ is in $\Sigma_t^* \setminus L_2$.

On the other hand, suppose $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \ldots \cdot (\sigma_n, \Lambda_n) \in \Sigma_t^* \setminus L_2$. Let $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n)$ be any timed word with $t_1 \in \Lambda_1$ and $t_i - t_{i-1} \in \Lambda_i$ for $i = 2, \ldots, n$. It holds that $w_t \notin L_1$, otherwise there would exist some $w' = (\sigma_1, \Lambda'_1) \cdot (\sigma_2, \Lambda'_2) \cdot \ldots \cdot (\sigma_n, \Lambda'_n) \in L_2$ such that $t_1 \in \Lambda'_1$ and $t_i - t_{i-1} \in \Lambda'_i$ for $1 < i \le n$ since $L_2 \approx_{\mathrm{tr}} L_1$. The fact that Λ_i and Λ'_i are both in a partition T_{σ_i} of $\mathbb{R}_{\ge 0}$ and $\Lambda_i \cap \Lambda'_i \ne \emptyset$ indicates that $\Lambda_i = \Lambda'_i$ and therefore $w = w'$, which results in a contradiction. Hence $w_t \in \mathrm{TW}^*(\Sigma) \setminus L_1$.

To sum up, $(\Sigma_t^* \setminus L_2) \approx_{\mathrm{tr}} (\mathrm{TW}^*(\Sigma) \setminus L_1)$ by Definition 5. ■

Lemma 3: If L_{11} and L_{12} are timed languages over Σ, L_{21} and L_{22} are partitioned languages over the same timed alphabet $\Sigma_t = \bigcup_{\sigma \in \Sigma} (\{\sigma\} \times T_\sigma)$, and $L_{21} \approx_{\mathrm{tr}} L_{11}$ and $L_{22} \approx_{\mathrm{tr}} L_{12}$, then it follows that $(L_{21} \cap L_{22}) \approx_{\mathrm{tr}} (L_{11} \cap L_{12})$.

Proof: First, $\varepsilon_t \in L_{11} \cap L_{12} \Leftrightarrow \varepsilon_t \in L_{11} \wedge \varepsilon_t \in L_{12} \Leftrightarrow \varepsilon \in L_{21} \wedge \varepsilon \in L_{22} \Leftrightarrow \varepsilon \in L_{21} \cap L_{22}$ by Definition 5.

Second, if $w_t = (\sigma_1, t_1) \cdot (\sigma_2, t_2) \cdot \ldots \cdot (\sigma_n, t_n) \in L_{11} \cap L_{12}$, then $w_t \in L_{11} \wedge w_t \in L_{12}$. As $w_t \in L_{11}$, there exists a $w^1 = (\sigma_1, \Lambda_1^1) \cdot (\sigma_2, \Lambda_2^1) \cdot \ldots \cdot (\sigma_n, \Lambda_n^1) \in L_{21}$ such that $t_i - t_{i-1} \in \Lambda_i^1$ for each i (here $t_0 = 0$). Similarly, as $w_t \in L_{12}$, there exists a $w^2 = (\sigma_1, \Lambda_1^2) \cdot (\sigma_2, \Lambda_2^2) \cdot \ldots \cdot (\sigma_n, \Lambda_n^2) \in L_{22}$ such that $t_i - t_{i-1} \in \Lambda_i^2$ for each i (also $t_0 = 0$). Since L_{21} and L_{22} are over a common alphabet Σ_t, Λ_i^1 and Λ_i^2 are both in T_{σ_i}, a partition of $\mathbb{R}_{\ge 0}$. $\Lambda_i^1 \cap \Lambda_i^2 \ne \emptyset$ means that $\Lambda_i^1 = \Lambda_i^2$, for $i = 1, \ldots, n$. Therefore, $w^1 = w^2$. So there exists a $w = w^1 = w^2$ such that $w \in L_{21} \wedge w \in L_{22}$, i.e., $w \in L_{21} \cap L_{22}$.

On the other hand, if $w = (a_1, \Lambda_1)(a_2, \Lambda_2) \ldots (a_n, \Lambda_n) \in L_{21} \cap L_{22}$, obviously $w \in L_{21}$ and $w \in L_{22}$. This implies that $[w] \subseteq L_{11}$ and $[w] \subseteq L_{12}$, and therefore $[w] \subseteq L_{11} \cap L_{12}$.

To sum up, $(L_{21} \cap L_{22}) \approx_{\mathrm{tr}} (L_{11} \cap L_{12})$ by Definition 5. ■

According to the above lemmas, if timed languages over an alphabet Σ are represented by FA accepting their trace-equivalent languages over a partitioned timed alphabet Σ_t, then the trace-equivalence relationship can be preserved under complementation and intersection over these FA.

D. Refined FA Over Partitioned Timed Alphabets

In this section, we construct another FA $\mathcal{B}'$ with a partitioned timed alphabet Σ'_t, such that $L_f(\mathcal{B}')$ is also trace-equivalent to $L_f(\mathcal{A})$.

The key point is to obtain the partitioned timed alphabet Σ'_t. Each transition of $\mathcal{B}$, say $(s_1, (\sigma, \Lambda), s_2)$, should be split into finite transitions in $\mathcal{B}'$, with the same pre- and post-states: $(s_1, (\sigma, \Lambda_1), s_2), (s_1, (\sigma, \Lambda_2), s_2), \dots, (s_1, (\sigma, \Lambda_k), s_2)$, such that $\bigcup_{j=1}^{k} \Lambda_j = \Lambda$ and $\Lambda_i \cap \Lambda_j = \emptyset$ for any $i \neq j$.

The finite set T_σ of $\mathcal{B}$ contains time labels of all σ-transitions for each σ. Then T'_σ meeting the above requirements can be computed from T_σ. The alphabet of $\mathcal{B}'$ is thus constructed as $\Sigma'_t = \cup_{\sigma\in\Sigma}\{\sigma\} \times T'_\sigma$.

An auxiliary function $\mathfrak{P}$ is recursively defined here to compute a finite partition given a finite set $C \subseteq 2^{\mathbb{R}_{\geq 0}}$.

1) $\mathfrak{P}(C) = \{\mathbb{R}_{\geq 0}\}$ if $|C| = 0$, i.e., $C = \emptyset$.
2) $\mathfrak{P}(C \cup \Lambda) = (\{\Lambda_1 \cap \Lambda, \dots, \Lambda_m \cap \Lambda\} \cup \{\Lambda_1 \setminus \Lambda, \dots, \Lambda_m \setminus \Lambda\}) \setminus \{\emptyset\}$, if $\mathfrak{P}(C) = \{\Lambda_1, \dots, \Lambda_m\}$ and $\Lambda \notin C$.

Additionally, $|\mathfrak{P}(C)|$ is no more than $2^{|C|}$.

By the definition of $\mathfrak{P}$, each nonempty $\Lambda \in C$ is partitioned into several subsets which are mutually disjoint, and the set of these subsets is denoted as $\mathfrak{P}_{C,\Lambda}$, or $\mathfrak{P}_\Lambda$ if there is no ambiguity. For instance, given $C = \{[2, 5], [3, 6]\}$, we can construct $\mathfrak{P}(C) = \{[3, 5], [2, 3), (5, 6], [0, 2) \cup (6, +\infty)\}$, satisfying $[2, 5] = [2, 3) \uplus [3, 5]$, and $[3, 6] = [3, 5] \uplus (5, 6]$.

Now given an FA $\mathcal{B} = (S, \Sigma_t, \delta, \text{Init}, F)$ whose alphabet $\Sigma_t = \bigcup_{\sigma\in\Sigma}\{\sigma\} \times T_\sigma$ is finite, we can obtain another FA $\mathcal{B}' = (S', \Sigma'_t, \delta', \text{Init}', F')$ such that Σ'_t is partitioned. We fix the set T^+_σ, a finite superset of T_σ in advance, and let $T'_\sigma = \mathfrak{P}(T^+_\sigma)$ for each σ. The selection strategy of T^+_σ is discussed at the end of this section.

Definition 7: Given $\mathcal{B}$ and $\{T'_\sigma\}_{\sigma\in\Sigma}$ as above, the refined FA, also called the refinement of $\mathcal{B}$, $\mathcal{B}' = (S', \Sigma'_t, \delta', \text{Init}', F')$ is as follows.

1) $S' = S$, $\text{Init}' = \text{Init}$, $F' = F$.
2) $\Sigma'_t = \bigcup_{\sigma\in\Sigma}\{\sigma\} \times T'_\sigma$.
3) $\delta'(s_1, (\sigma, \Lambda')) = \{s_2 \mid s_2 \in \delta(s_1, (\sigma, \Lambda)) \wedge \Lambda' \subseteq \Lambda\} = \bigcup\{\delta(s_1, (\sigma, \Lambda)) \mid \Lambda' \subseteq \Lambda\}$.

Lemma 4: Let $\mathcal{B}'$ be a refinement of $\mathcal{B}$, then $L(\mathcal{B}')$ is trace-equivalent to a timed language L_t if $L(\mathcal{B})$ is trace-equivalent to L_t. And $L_f(\mathcal{B}')$ is trace-equivalent to a timed language $L_{t,f}$ if $L_f(\mathcal{B})$ is trace-equivalent to $L_{t,f}$.

Proof: If $\varepsilon_t \in L_t$ (resp. $L_{t,f}$), $\varepsilon \in L(\mathcal{B})$ [resp. $L_f(\mathcal{B})$]. A possible run is s_0, where $s_0 \in \text{Init}$ (resp. $\text{Init} \cap F$). So $s_0 \in \text{Init}'$ (resp. $\text{Init}' \cap F'$) and $\varepsilon \in L(\mathcal{B}')$ [resp. $L_f(\mathcal{B})$]. On the other hand, suppose $\varepsilon \in L(\mathcal{B})$ [resp. $L_f(\mathcal{B})$], which implies there exists $s_0 \in \text{Init}'$ (resp. $\text{Init}' \cap F'$). It follows that $s_0 \in \text{Init}$ (resp. $\text{Init} \cap F$); therefore $\varepsilon \in L(\mathcal{B})$ [resp. $L_f(\mathcal{B})$] and $\varepsilon_t \in L_t$ (resp. $L_{t,f}$).

Suppose $w_t = (\sigma_1, t_1) \cdot \dots \cdot (\sigma_n, t_n) \in L_t$ (resp. $L_{t,f}$), where $n \geq 1$, so there exists a word $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \dots \cdot (\sigma_n, \Lambda_n) \in L(\mathcal{B})$ [resp. $L_f(\mathcal{B})$] such that $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \leq n$. Thus, there exists some $w' = (\sigma_1, \Lambda'_1) \cdot (\sigma_2, \Lambda'_2) \cdot \dots \cdot (\sigma_n, \Lambda'_n) \in L(\mathcal{B}')$ [resp. $L_f(\mathcal{B}')$] such that $t_1 \in \Lambda'_1$ and $(t_i - t_{i-1}) \in \Lambda'_i$ for $1 < i \leq n$ according to the construction above.

Given a word $w' = (\sigma_1, \Lambda'_1) \cdot (\sigma_2, \Lambda'_2) \cdot \dots \cdot (\sigma_n, \Lambda'_n) \in L(\mathcal{B}')$ (resp. $L_f(\mathcal{B}')$), there exists a word $w = (\sigma_1, \Lambda_1) \cdot (\sigma_2, \Lambda_2) \cdot \dots \cdot (\sigma_n, \Lambda_n) \in L(\mathcal{B})$ (resp. $L_f(\mathcal{B})$) such that $\Lambda' \subseteq \Lambda$ for $1 \leq i \leq n$. So any word in $[w]$ is also in L_t (resp. $L_{t,f}$). Since $[w'] \subseteq [w]$, all words in $[w']$ is in L_t (resp. $L_{t,f}$). ■

Now let us come back to the selection strategy of T^+_σ for each σ. Suppose the FA under consideration are $\mathcal{B}$, $\mathcal{B}_1$, and $\mathcal{B}_2$ which have n, n_1, and n_2 states, respectively. Let T_σ (resp. $T_{\sigma,1}$ and $T_{\sigma,2}$) be the duration parts of σ-transitions in $\mathcal{B}$ (resp. in $\mathcal{B}_1$ and $\mathcal{B}_2$), and k_σ, $k_{\sigma,1}$, and $k_{\sigma,2}$ be their cardinalities, respectively.

If we need to get the "complement" of $L_f(\mathcal{B})$, we should choose $T^+_\sigma = T_\sigma$, as $T'_\sigma = \mathfrak{P}(T_\sigma)$ is sufficient for each σ. Accordingly, the refined FA $\mathcal{B}'$ has n states, at most $\sum_{\sigma\in\Sigma} 2^{k_\sigma}$ timed events and at most $\sum_{\sigma\in\Sigma} k_\sigma 2^{k_\sigma}$ transitions.

If we need to get the "intersection," "union," or "minus" of $L_f(\mathcal{B}_1)$ and $L_f(\mathcal{B}_2)$, we should choose $T^+_\sigma = T_{\sigma,1} \cup T_{\sigma,2}$ for all of them. Thus, $T'_\sigma = \mathfrak{P}(T_{\sigma,1} \cup T_{\sigma,2})$. $\mathcal{B}_i$ has n_i states, at most $\sum_{\sigma\in\Sigma}(2^{k_{\sigma,1}+k_{\sigma,2}})$ timed events and at most $\sum_{\sigma\in\Sigma} k_{\sigma,i}(2^{k_{\sigma,1}+k_{\sigma,2}})$ transitions for $i = 1, 2$.

Note that the idea of partition is similar to the proof in [14, Th. 4.4]. But our calculation of partition of non-negative reals is more flexible by choosing different parameters for $\mathfrak{P}$ for different operations, that can reduce the number of states and/or transitions essentially.

By translating RTA into FA, further into refined FA, we can get a better understanding of their timed languages. Moreover, the trace-equivalence relationship allows to compute complementation and intersection of such languages.

IV. Projection

In this section, we concentrate on projection of FA over timed alphabets. Given $\mathcal{B}$ and a timed language L_t satisfying $L_f(\mathcal{B}) \approx_{\text{tr}} L_t$, we would like to build $\mathcal{B} \uparrow_{\Sigma_o}$ from $\mathcal{B}$, such that $L_f(\mathcal{B} \uparrow_{\Sigma_o}) \approx_{\text{tr}} P_{\Sigma_o,t}(L_t)$, where Σ_o denotes the observable alphabet.

We use symbols $a, b, c, \dots$ (with or without subscripts) to denote the observable symbols from Σ_o, and the single symbol τ (with or without subscripts) to denote all the unobservable ones from $\Sigma \setminus \Sigma_o$ in the following discussions.

A. Projection of Languages

We start our analysis with an arbitrary timed word

$$w'_t = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n) \tag{2}$$

in the timed language $P_{\Sigma_o,t}(L_t)$. The case where w'_t is ε_t is also contained in the form (2), by letting $n = 0$. There must exist some w_t in L_t such that $P_{\Sigma_o,t}(w_t) = w'_t$. Without loss of generality, w_t is of the form

$$w_t = u^1_t \cdot (a_1, t_1) \cdot u^2_t \cdot (a_2, t_2) \cdots u^n_t \cdot (a_n, t_n) \cdot u^{n+1}_t \tag{3}$$

where for $1 \leq j \leq n+1$, u^j_t is either ε_t or an "unobservable" timed word $(\tau, t_{j1}) \dots (\tau, t_{jm_j})$ with $t_{j-1} \leq t_{j1} \leq \dots \leq t_{jm_j} \leq t_j$ (t_0 and t_{n+1} are deemed to be 0 and $+\infty$, respectively).

As $L_f(\mathcal{B}) \approx_{\text{tr}} L_t$, there also exists in $L_f(\mathcal{B})$ some w similar to w_t in the structure, that is,

$$w = u^1 \cdot (a_1, \Lambda_1) \cdot u^2 \cdot (a_2, \Lambda_2) \cdots u^n \cdot (a_n, \Lambda_n) \cdot u^{n+1} \qquad (4)$$

where for $1 \le j \le n+1$, u^j is either ε or a timed word $(\tau, \Lambda_{j1}) \ldots (\tau, \Lambda_{jm_j})$ such that for $1 \le j \le n+1$, $t_j - t_{jk_j} \in \Lambda_j$ and for $1 \le k \le m_i$ $t_{jk} - t_{j(k-1)} \in \Lambda_{jk}$ (t_0 and t_{n+1} are deemed to be 0 and $+\infty$, respectively, and each t_{j0} is t_{j-1} for $1 \le j \le n+1$).

Although the projection of a timed word w_t as in (3) over Σ_o is $P_{\Sigma_o,t}(w_t) = (a_1, t_1)(a_2, t_2) \ldots (a_n, t_n) = w_t'$, this is not suitable for the projection of w as in (4). If we delete all the unobservable u^j roughly, the result would be $(a_1, \Lambda_1) \cdot (a_2, \Lambda_2) \cdot \ldots \cdot (a_n, \Lambda_n)$, which does not satisfy $w_t' \in [w']$ any more, because in a timed word w_t, each time stamp is the amount of time elapsed since the beginning of w_t (i.e., the physical time when the event occurs), while in w, each duration contains all the possibilities of time elapsed since the occurrence of its previous action (i.e., the logical time related to its previous event). In other words, each symbol-duration pair, no matter observable or not, plays a part both in w and in its projection. The projection of w, denoted as $w' = P_{\uparrow\Sigma_o}(w)$ and to guarantee $\{P_{\uparrow\Sigma_o}(w)\} \approx_{\text{tr}} P_{\Sigma_o,t}([w])$, will be defined below. We use a different notation here to avoid confusion with the projection of normal words and timed words. And slightly overloading the notation, $P_{\uparrow\Sigma_o}(L)$ denotes the set $\{P_{\uparrow\Sigma_o}(w) \mid w \in L\}$.

There still remains one thing to do before formally defining the function $P_{\uparrow\Sigma_o}$, to recall the addition operation "+" (and "Σ") on sets of non-negative real numbers. Let Λ_1, $\Lambda_2, \ldots, \Lambda_m$ be subsets of $\mathbb{R}_{\ge 0}$. $\Lambda_1 + \Lambda_2 := \{\lambda_1 + \lambda_2 \mid \lambda_1 \in \Lambda_1 \wedge \lambda_2 \in \Lambda_2\}$, and $\sum_{j=1}^{m} \Lambda_j = \Lambda_1 + \Lambda_2 + \cdots + \Lambda_m$. This operation is commutative and associative. Moreover the star operation can be defined: $\Lambda^* = \bigcup_{k \in \mathbb{N}} k\Lambda$, where $0\Lambda = \{0\}$ and $(k+1)\Lambda = k\Lambda + \Lambda$.

For w as in (4), $P_{\uparrow\Sigma_o}(w)$ is inductively defined based on the number of observable symbol-duration pairs in w

$$P_{\uparrow\Sigma_o}(u^1) = \varepsilon;$$
$$P_{\uparrow\Sigma_o}\Big(u^1(a_1, \Lambda_1)\Big) = \begin{cases} (a_1, \Lambda_1), & \text{if } u^1 = \varepsilon \\ \left(a_1, \sum_{k=1}^{m_1} \Lambda_{1k} + \Lambda_1\right), & \text{otherwise} \end{cases}$$
$$P_{\uparrow\Sigma_o}\Big(\Big(u^1(a_1, \Lambda_1)\Big) \cdot w_{\text{suffix}}\Big) = P_{\uparrow\Sigma_o}\Big(u^1(a_1, \Lambda_1)\Big) \times P_{\uparrow\Sigma_o}(w_{\text{suffix}})$$

where u^1 is either ε or $(\tau, \Lambda_{11}) \ldots (\tau, \Lambda_{1m_1})$. According to its definition, if we factorize the duration word w into $w_1 \cdot \ldots \cdot w_n \cdot u^{n+1}$ where $w_j = u^j \cdot (a_j, \Lambda_j)$ for each $1 \le j \le k$, we will have $P_{\uparrow\Sigma_o}(w) = P_{\uparrow\Sigma_o}(w_1) \cdot P_{\uparrow\Sigma_o}(w_2) \cdot \ldots \cdot P_{\uparrow\Sigma_o}(w_n) \cdot P_{\uparrow\Sigma_o}(u^{n+1}) = P_{\uparrow\Sigma_o}(w_1) \cdot P_{\uparrow\Sigma_o}(w_2) \cdot \ldots \cdot P_{\uparrow\Sigma_o}(w_n)$.

Lemma 5: $\{P_{\uparrow\Sigma_o}(w)\} \approx_{\text{tr}} P_{\Sigma_o,t}([w])$.

Proof: First, $P_{\uparrow\Sigma_o}(w) = \varepsilon$ if w is either ε or $(\tau, \Lambda_{11}) \cdot \ldots \cdot (\tau, \Lambda_{1m_1})$. In either case, $P_{\Sigma_o,t}(w_t) = \varepsilon$ for each w_t in $[w]$, so $[w] = \{\varepsilon_t\}$. Then $\{P_{\uparrow\Sigma_o}(w)\} \approx_{\text{tr}} P_{\Sigma_o,t}([w])$.

Second, let $w = w_1 \cdot \ldots \cdot w_n \cdot u^{n+1}$, and $w' = P_{\uparrow\Sigma_o}(w) = P_{\uparrow\Sigma_o}(w_1) \cdot P_{\uparrow\Sigma_o}(w_2) \cdot \ldots \cdot P_{\uparrow\Sigma_o}(w_n) \cdot P_{\uparrow\Sigma_o}(u^{n+1}) = (a_1, \Lambda_1') \cdot (a_2, \Lambda_2') \cdot \ldots \cdot (a_n, \Lambda_n')$.

For any $w_t \in [w]$, $w_t = (\tau, t_{11}) \cdot \ldots \cdot (\tau, t_{1m_1}) \cdot (a_1, t_1) \cdot \ldots \cdot (\tau, t_{n1}) \cdot \ldots \cdot (\tau, t_{nm_n}) \cdot (a_n, t_n) \cdot (\tau, t_{(n+1)1}) \cdot \ldots \cdot (\tau, t_{(n+1)m_{n+1}})$, and $w_t' = (a_1, t_1) \cdot \ldots \cdot (a_n, t_n)$. If $u^j = \varepsilon$, $t_j - t_{j-1} \in \Lambda_j = \Lambda_j'$. If u^j is not an empty word, as $t_j - t_{jk_j} \in \Lambda_j$ and $t_{jk} - t_{j(k-1)} \in \Lambda_{jk}$ for $1 \le j \le n+1$ and $1 \le k \le m_i$, $t_j - t_{j-1} \in \Lambda_j + \sum_{k=1}^{m_j} \Lambda_{jk} = \Lambda_j'$.

Similarly, for any $w_t' = (a_1, t_1) \cdot (a_2, t_2) \cdot \ldots \cdot (a_n, t_n) \in P_{\Sigma_o,t}([w])$, it holds that each $t_j - t_{j-1} \in \Lambda_j'$ ($t_0 = 0$).

To sum up, $\{P_{\uparrow\Sigma_o}(w)\} \approx_{\text{tr}} P_{\Sigma_o,t}([w])$. ■

B. Projection of FA

We show how to construct $\mathcal{B} \uparrow_{\Sigma_o}$.

Suppose the FA under consideration is $\mathcal{B} = (S, \Sigma_t, \delta, \text{Init}, F)$, where $\Sigma_t = \bigcup_{\sigma \in \Sigma} \{\sigma\} \times T_\sigma$ and $\Sigma = \Sigma_o \uplus \{\tau\}$ is partitioned into an observable set Σ_o and an unobservable set $\{\tau\}$. Fix a run ρ of $\mathcal{B}$, whose trace is $w \in L_f(\mathcal{B})$, such that $w = w_1 \cdot \ldots \cdot w_n \cdot u^{n+1}$, where $w_j = u^j \cdot (a_j, \Lambda_j)$ for each $1 \le j \le k$. Its projection is $w' = P_{\uparrow\Sigma_o}(w) = P_{\uparrow\Sigma_o}(w_1) \cdot P_{\uparrow\Sigma_o}(w_2) \cdot \ldots \cdot P_{\uparrow\Sigma_o}(w_n) \cdot P_{\uparrow\Sigma_o}(u^{n+1})$. Although $P_{\uparrow\Sigma_o}(u^{n+1}) = \varepsilon$, u^{n+1} still matters since the trace is an accepted one. Hence we can deal with each segment w_j separately and finally put them together. Each w_j also comprises two parts: the first is u^j which is unobservable or empty, and the second is observable (a_j, Λ_j) meaning that there is a transition $(s_{jm_j}, (a_j, \Lambda_j), s_{jm_j}')$.

1) Sum of Successive Unobservable Transitions: We first deal with the case where u^j is a sequence of unobservable transitions, that is, $u^j = (\tau, \Lambda_{j1}) \ldots (\tau, \Lambda_{jm_j})$. So the corresponding segment of the run ρ is $s_{j0} \xrightarrow{(\tau, \Lambda_{j1})} s_{j1} \cdots \xrightarrow{(\tau, \Lambda_{jm_j})} s_{jm_j} \xrightarrow{(a_j, \Lambda_j)} s_{jm_j}'$. If $j = 1$, the starting state s_{10} should be an initial state of $\mathcal{B}$. If $j > 1$, s_{j0} should be the same as the ending state of its previous segment, that is, $s_{j0} = s_{(j-1)m_{j-1}}'$ is also the post-state of the observable transition $(s_{(j-1)m_{j-1}}, (a_{j-1}, \Lambda_{j-1}), s_{(j-1)m_{j-1}}')$.

In the case where $u^j = \varepsilon$, the corresponding segment of ρ is therefore $s_{j0} \xrightarrow{(a_j, \Lambda_j)} s_{j0}'$, such that s_{j0} is initial if $j = 1$ and is the post-state of $(s_{(j-1)m_{j-1}}, (a_{j-1}, \Lambda_{j-1}), s_{(j-1)m_{j-1}}')$ if $j > 1$.

As a consequence, we should delete all the unobservable transitions from $\mathcal{B}$, and add new transitions with labels $P_{\uparrow\Sigma_o}(w_j)$.

A new transition $(s, (a_j, \Lambda_j'), s')$ with an observable label should be subject to the following restrictions.

1) Its prestate s is an initial state, or the post-state of an observable transition.
2) Its post-state s' is the post-state of an observable transition with symbol a_j, say $(s_{jm_j}, (a_j, \Lambda_j), s_{jm_j}')$.
3) There is a segment of run $s_{j0} \xrightarrow{(\tau, \Lambda_{j1})} s_{j1} \cdots \xrightarrow{(\tau, \Lambda_{jm_j})} s_{jm_j} \xrightarrow{(a_j, \Lambda_j)} s_{jm_j}'$, where $m_j \ge 1$ such that $s = s_{j0}$, $s' = s_{jm_j}'$, and $\Lambda_j' = \sum_{k=1}^{m_j} \Lambda_{jk} + \Lambda_j$.

In the sequel we explain the steps to obtain such new transitions based on $\mathcal{B}$.

First, we calculate the sum of durations of a sequence of unobservable transitions, of which the first starts in either an initial state of $\mathcal{B}$ or the post-state of an observable transition, and of which the last ends in the prestate of an observable transition. Slightly overloading the notation, Pre_a and Post_a

denote the set of states which are pre and post-states of transitions with the observable symbol "a" in its label. To this end, a new FA $\mathcal{B}_\tau$ is constructed according to $\mathcal{B}$.

Definition 8: $\mathcal{B}_\tau = (S_\tau, \Sigma_\tau, \delta_\tau, \mathrm{Init}_\tau, F_\tau)$, where:

S_τ	S;
Σ_τ	T_τ, the set of the duration parts of τ-transitions in $\mathcal{B}$;
$\delta_t(s_1, \Lambda)$	$\{s_2 \mid s_2 \in \delta(s_1, (\tau, \Lambda))\}$, in other words, (s_1, Λ, s_2) is a transition in $\mathcal{B}_\tau$ iff $(s_1, (\tau, \Lambda), s_2)$ is a transition in $\mathcal{B}$;
Init_τ	$\mathrm{Init} \cup \bigcup_{a\in\Sigma_o} \mathrm{Post}_a$;
F_τ	$F \cup \bigcup_{a\in\Sigma_o} \mathrm{Pre}_a$.

Below is shown how to calculate regular expressions corresponding to traces in $\mathcal{B}_\tau$ from one state to another and to transform each regular expression into a duration, i.e., a subset of non-negative real numbers. This can be done by following the Floyd–Warshall algorithm (a well-known method for proving the Kleene's Theorem).

We first rename states in S_τ as $s_1, \ldots, s_{|S_\tau|}$, since S_τ is finite. Suppose $s_i \to \cdots \to s_j$ is a run from s_i to s_j whose labels above the right arrows are omitted for simplicity here. If there is no state between s_i and s_j, the set of all such runs is denoted as $\mathrm{Run}_{i,j}(0)$, and $RE_{i,j}(0)$ is used to denote the regular expression expressing all the traces of runs in $\mathrm{Run}_{i,j}(0)$. If all the states between s_i and s_j have subscripts less or equal to k, the set of all such runs is denoted as $\mathrm{Run}_{i,j}(k)$, and $RE_{i,j}(k)$ is defined similarly. It is trivial that $\mathrm{Run}_{i,j}(k) \subseteq \mathrm{Run}_{i,j}(k+1)$ for $0 \le k < |S_\tau|$; therefore $RE_{i,j}(k)$ can be calculated inductively for $0 \le k \le |S_\tau|$. Thus, we can finally obtain the required $RE_{i,j}(|S_\tau|)$ for each pair of $(s_0, s_f) \in \mathrm{Init}_\tau \times F_\tau$.

In the base case, we compute $RE_{i,j}(0)$ for each pair (s_i, s_j) as: 1) If $s_i \neq s_j$ and there is no transition from s_i to s_j, $RE_{i,j}(0) = \emptyset$. 2) If $s_i \neq s_j$ and the set of labels of transitions from s_i to s_j, $\Sigma_{i,j} = \{\Lambda \mid s_j \in \delta_t(s_i, \Lambda)\}$ is nonempty, we rename those labels as $\Lambda_1, \ldots, \Lambda_{|\Sigma_{i,j}|}$. Then $RE_{i,j}(0) = \Lambda_1 + \cdots + \Lambda_{|\Sigma_{i,j}|}$. 3) If $s_i = s_j$ and there is no transition from s_i to s_j, $RE_{i,j}(0) = \varepsilon$. 4) If $s_i = s_j$ and the set of labels of transitions from s_i to s_j, $\Sigma_{i,j} = \{\Lambda \mid s_j \in \delta_t(s_i, \Lambda)\}$ is nonempty, we rename those labels as $\Lambda_1, \ldots, \Lambda_{|\Sigma_{i,j}|}$. Then $RE_{i,j}(0) = \varepsilon + \Lambda_1 + \cdots + \Lambda_{|\Sigma_{i,j}|}$.

In the step case, we recursively compute $RE_{i,j}(k+1) = RE_{i,j}(k) + RE_{i,k+1}(k) \cdot RE_{k+1,k+1}(k)^* \cdot RE_{k+1,j}(k)$. This is because any run ρ in $\mathrm{Run}_{i,j}(k+1)$ is either in $\mathrm{Run}_{i,j}(k)$ or is a run where the state s_{k+1} occurs at least once. If the state s_{k+1} occurs at least once in a run ρ, this run can also be split into three segments: the first part is from its start s_i to the first s_{k+1}, the second part is from the first s_{k+1} to the last s_{k+1}, and the third one is from the last s_{k+1} to its end s_j. Also the second part is split into at least one segments, each of which starts in s_{k+1}, ends in s_{k+1}, and has no states with subscripts larger than k in between.

Finally, we can obtain $RE_{i,j}(|S_\tau|)$ for each pair of states (s_i, s_j), which expresses all the possible traces of runs from s_i to s_j.

After regular expressions have been obtained, it is necessary to transform them into appropriate durations, i.e., subsets of $\mathbb{R}_{\ge 0}$. For the base clause, $\emptyset$ is transformed into the empty set $\emptyset$, ε into the set $\{0\}$, and Λ into the set Λ itself. For the inductive clause, if r, r_1, and r_2 are regular expressions transformed into Λ, Λ_1, and Λ_2, respectively, then $r_1 \cdot r_2$ is transformed into $\Lambda_1 + \Lambda_2 := \{\lambda_1 + \lambda_2 \mid \lambda_1 \in \Lambda_1 \wedge \lambda_2 \in \Lambda_2\}$, $r_1 + r_2$ into $\Lambda_1 \cup \Lambda_2 := \{\lambda \mid \lambda \in \Lambda_1 \vee \lambda \in \Lambda_2\}$, and r^* into $\Lambda^* := \bigcup_{k\in\mathbb{N}} k\Lambda$, where $0\Lambda = \{0\}$ and $(k+1)\Lambda = k\Lambda + \Lambda$.

In conclusion, for each pair of states $(s_i, s_j) \in \mathrm{Init}_\tau \times F_\tau$, we can obtain Λ_{s_i,s_j} from $RE_{i,j}(|S_\tau|)$, so as to determine: 1) whether s_i can reach s_j via unobservable transitions by checking whether Λ_{s_i,s_j} is nonempty and 2) the time duration from s_i to s_j if it is reachable. In this step the complexity is $O(|S_\tau|^3)$.

2) Merging Unobservable Durations Into Observable Transitions: After the analysis of all the u^j above, we would like to consider w_j by merging the duration on u^j into the observable (a_j, Λ_j) so as to build the automaton $\mathcal{B} \uparrow_{\Sigma_o} = (S_p, \Sigma_p, \delta_p, \mathrm{Init}_p, F_p)$.

The states and initial states of $\mathcal{B} \uparrow_{\Sigma_o}$ can be set to be the same as those of the original $\mathcal{B}$.

Transitions of $\mathcal{B} \uparrow_{\Sigma_o}$ are all of the form $(s_i, (a, \Lambda_{\mathrm{new}}), s_k)$. A new transition $(s_i, (a, \Lambda_{\mathrm{new}}), s_k)$ should be included if there exists s_j and Λ such that $s_k \xrightarrow{(a,\Lambda)} s_j$, s_i can reach s_j via unobservable transitions in $\mathcal{B}$, and $\Lambda_{\mathrm{new}} = \Lambda + \Lambda_{s_i,s_j}$. Formally δ_p is defined such that $\delta_p(s_i, (a, \Lambda_{\mathrm{new}})) = \{s_k \mid \exists s_j \exists \Lambda \big(s_k \in \delta(s_j, (a, \Lambda)) \wedge \Lambda_{s_i,s_j} \neq \emptyset \wedge \Lambda_{\mathrm{new}} = \Lambda + \Lambda_{s_i,s_j}\big)\}$.

And the alphabet can be derived from the transitions. That is, $\Sigma_p = \{(a, \Lambda) \mid \exists s \big(\delta_p(s, (a, \Lambda)) \neq \emptyset\big)\}$.

The accepting states needs special attention. s_k is an accepting state if and only if i) s_k is an initial state or the post-state of some observable transition, and ii) s_k can reach an accepting state via zero or more unobservable transitions. Formally, let $F_{\mathrm{reach}} = \{s_k \mid \exists s_f \in F\big(\Lambda_{s_k,s_f} \neq \emptyset\big)\}$, so $F_p = (\mathrm{Init} \cup \bigcup_{a\in\Sigma_o} \mathrm{Post}_a) \cap F_{\mathrm{reach}}$.

Thus, the construction of $\mathcal{B} \uparrow_{\Sigma_o}$ is done. $\mathcal{B} \uparrow_{\Sigma_o}$ has the same number of states as the original $\mathcal{B}$ and the unobservable $\mathcal{B}_\tau$, and at most $|\delta_o| * |S_\tau|^2$ transitions, where δ_o stands for restricting δ, the set of transitions of $\mathcal{B}$, to the observable set.

Theorem 2: The accepting language of $\mathcal{B} \uparrow_{\Sigma_o}$ is exactly the projection language $P_{\uparrow\Sigma_o}(L_f(\mathcal{B}))$.

Proof: Follows from the construction in this section. ■

Note that the proposed scheme in this paper of projection is similar to [14] in that the Floyd–Warshall algorithm is utilized in both work to compute the "sum" of a sequel of non-negative (resp. positive) sets.

Besides, as there is no special constraints on the time labeling function in our definition of RTA, i.e., any nonempty subset of $\mathbb{R}_{\ge 0}$ is allowed, so normal form of sets in $\mathcal{K}(\mathrm{Int})$ can be circumvented in our theoretical analysis. But in our implementation, we utilize normal forms to obtain a finite subset of $2^{\mathbb{R}_{\ge 0}}$ for any regular expression over finite subsets of $2^{\mathbb{R}_{\ge 0}}$.

V. Decidability

Now let us come back to the language-opacity and initial-opacity problems for RTA, and fix an RTA $\mathcal{A}$ with an alphabet Σ.

First we focus on language-opacity. Let a secret RTA $\mathcal{A}_{\mathrm{secret}}$ accept the secret language. $\mathcal{A}$ and $\mathcal{A}_{\mathrm{secret}}$ are assumed to share a common alphabet Σ, as we can always expand the alphabet to the union of their alphabets. We need to determine whether $P_{\Sigma_o,t}(L(\mathcal{A})) \subseteq P_{\Sigma_o,t}(L(\mathcal{A}) \setminus L(\mathcal{A}_{\text{secret}}))$.

The following theorem indicates that the language-opacity problem of RTA is decidable.

Theorem 3: The language-opacity problem of RTA is decidable.

Proof: Consider an RTA $\mathcal{A}$ and a secret RTA $\mathcal{A}_{\text{secret}}$ accepting the secret language.

According to Section III, two FA, say $\mathcal{B}$ and $\mathcal{B}_{ns}$, can be constructed such that they, respectively, accept languages trace-equivalent to $L(\mathcal{A})$ and $L(\mathcal{A}) \setminus L(\mathcal{A}_{\text{secret}})$.

According to Section IV, it is natural to obtain $\mathcal{B} \uparrow_{\Sigma_o}$ and $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$ which accept $P_{\uparrow\Sigma_o}(L_f(\mathcal{B}))$ and $P_{\uparrow\Sigma_o}(L_f(\mathcal{B}_{ns}))$, respectively.

Finally, we can apply the complementation and product operations on them again, and obtain $\mathcal{B}_{\text{final}} = \mathcal{B} \uparrow_{\Sigma_o} \times(\mathcal{B}_{ns} \uparrow_{\Sigma_o})^{\text{comp}}$. Clearly, $\mathcal{A}$ is language-opaque with respect to $\mathcal{A}_{\text{secret}}$ and Σ_o if and only if $L_f(\mathcal{B}_{\text{final}})$ is empty. It is well-known that the latter is decidable [15]. ■

In the following theoretical complexity in the worst case is discussed. Suppose the common alphabet Σ comprises l events, $\mathcal{A}$ has n states and m transitions (k_σ transitions for each σ), and $\mathcal{A}_{\text{secret}}$ has n' states and m' transitions (k'_σ transitions for each σ).

First, B has n states and m transitions, while $\mathcal{B}_{ns}$ has $n(n'+1)$ states and $n(n'+1)2^{k_\sigma+k'_\sigma}$ σ-transitions for each σ. Second, for their projections, $\mathcal{B} \uparrow_{\Sigma_o}$ has n states and $n^2 2^{k_\sigma+k'_\sigma}$ σ-transitions for each $\sigma \in \Sigma_o$, and $\mathcal{B}_{\text{secret}} \uparrow_{\Sigma_o}$ has $n(n'+1)$ states and $(n(n'+1))^3 2^{k_\sigma+k'_\sigma}$ transitions for each $\sigma \in \Sigma_o$. When we unify alphabets of the two projections, it's possible to obtain two NFA, so we must transform $\mathcal{B} \uparrow_{\Sigma_o}$ into DFA, whose number of states becomes $2^{n(n'+1)}$. Thus, the final result $\mathcal{B}_{\text{final}}$ has $n(2^{n(n'+1)}+1)$ states, and $2^{O(n^4 n'^4 2^{k_\sigma+k'_\sigma})}$ transitions for each $\sigma \in \Sigma_o$. The last thing is to check emptiness of $\mathcal{B}_{\text{final}}$, in linear time with respect to $|\mathcal{B}_{\text{final}}.S| \cdot |\mathcal{B}_{\text{final}}.\delta|$.

In practice the numbers of states and transitions are much smaller, since we can simplify the resulted automata and prune useless states and transitions.

Then we turn to initial-opacity. Let S_{secret} be a secret set of states, and we use $L_s(\mathcal{A})$ and $L_{ns}(\mathcal{A})$ as the abbreviations of $\text{Tr}(\text{Init} \cap S_{\text{secret}})$ and $\text{Tr}(\text{Init} \setminus S_{\text{secret}})$, respectively.

Let $L_{\text{secret}} = L(\mathcal{A}) \setminus L_{ns}(\mathcal{A})$. Then $\mathcal{A}$ is initial-state opaque with respect to S_{secret} and Σ_o iff $P_{\Sigma_o,t}(L(\mathcal{A})) \subseteq P_{\Sigma_o,t}(L(\mathcal{A}) \setminus L_{\text{secret}})$ iff $\mathcal{A}$ is language-opaque with respect to L_{secret} and Σ_o. Therefore, the initial-opacity problem of RTA is also decidable.

Theorem 4: The initial-state opacity problem for RTA is decidable.

A. Simple Example of Language-Opacity

We illustrate our method with a simple example.

The two RTA under study are as in Fig. 2(a) and (b). $L(\mathcal{A})$ is $\{\varepsilon_t\} \cup \{(a,t_1) \mid t_1 \in [2,5]\} \cup \{(b,t_1) \mid t_1 \in [2,4]\} \cup \{(a,t_1)(a,t_2) \mid t_1 \in [2,5] \wedge (t_2-t_1) \in [1,3]\} \cup \{(a,t_1)(b,t_2) \mid t_1 \in [2,5] \wedge (t_2-t_1) \in [3,4]\} \cup \{(b,t_1)(a,t_2) \mid t_1 \in [2,4] \wedge (t_2-t_1) \in [1,3]\} \cup \{(b,t_1)(b,t_2) \mid t_1 \in [2,4] \wedge (t_2-t_1) \in [3,4]\}$.

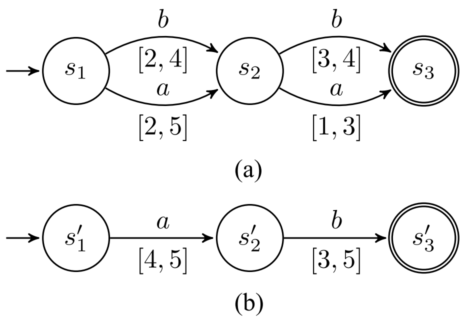

Fig. 2. (a) $\mathcal{A}$. (b) $\mathcal{A}_{\text{secret}}$.

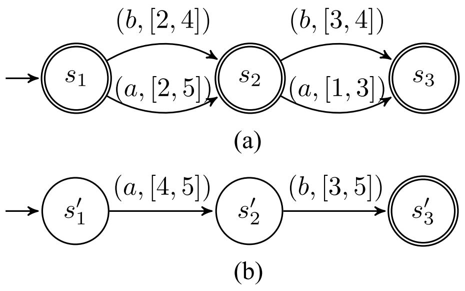

Fig. 3. FA directly derived from $\mathcal{A}$ and $\mathcal{A}_{\text{secret}}$. (a) $\mathcal{B}$. (b) $\mathcal{B}_{\text{secret}}$.

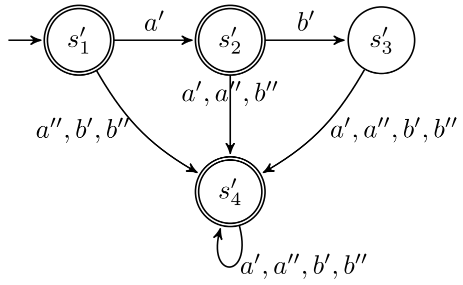

Fig. 4. $\mathcal{B}^{\text{comp}}_{\text{secret}}$, the complement of $\mathcal{B}'_{\text{secret}}$.

And $L_{\text{secret}} = L_f(\mathcal{A}_{\text{secret}})$ is $\{(a,t_1)(b,t_2) \mid t_1 \in [4,5] \wedge (t_2-t_1) \in [3,5]\}$.

The whole process is as follows.

First, we construct $\mathcal{B}$ and $\mathcal{B}_{ns}$ which accept languages trace-equivalent to $L(\mathcal{A})$ and $L(\mathcal{A}) \setminus L(\mathcal{A}_{\text{secret}})$, respectively.

1) Construct $\mathcal{B}$: This can be done directly, and the resulting $\mathcal{B}$ is depicted in Fig. 3(a). Its timed alphabet Σ_{t1} is $\{(a,[1,3]),(a,[2,5]),(b,[2,4]),(b,[3,4])\}$.

2) Construct $\mathcal{B}^{\text{comp}}_{\text{secret}}$: We first construct $\mathcal{B}_{\text{secret}}$ as in Fig. 3(b). Then we refine $\mathcal{B}_{\text{secret}}$ into $\mathcal{B}'_{\text{secret}}$, which looks the same as $\mathcal{B}_{\text{secret}}$, whose timed alphabet is Σ_{t2} is $\{(a,[4,5]),(a,[0,4)\cup(5,+\infty))\} \cup \{(b,[3,5]),(b,[0,3)\cup(5,+\infty))\}$ instead. Hence $\mathcal{B}^{\text{comp}}_{\text{secret}}$ can be constructed, as shown in Fig. 4. Elements in Σ_{t2} are abbreviated to a', a'', b', b'', respectively.

3) Construct $\mathcal{B}_{ns}$: We first refine $\mathcal{B}$ and $\mathcal{B}^{\text{comp}}_{\text{secret}}$ by the partitioned timed alphabet $\{(a,[1,2)),(a,[2,3]),(a,(3,4)),(a,[4,5]),(a,[0,1)\cup(5,+\infty)),(b,[2,3)),(b,[3,4]),(b,(4,5]),(b,[0,2)\cup(5,+\infty))\}$ obtained from Σ_{t1} and Σ_{t2}. We use a_1, a_2, a_3, a_4, a_5, b_1, b_2, b_3, b_4 as

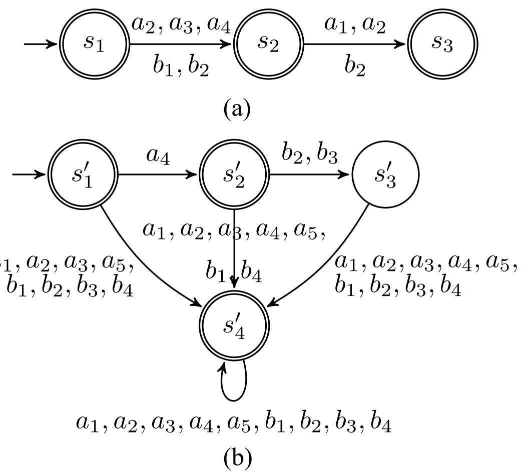

Fig. 5. Refined FA of $\mathcal{B}$ and $\mathcal{B}_{\text{secret}}^{\text{comp}}$. (a) $\mathcal{B}'$. (b) $\mathcal{B}_{\text{secret}}^{\text{comp}\,\prime}$.

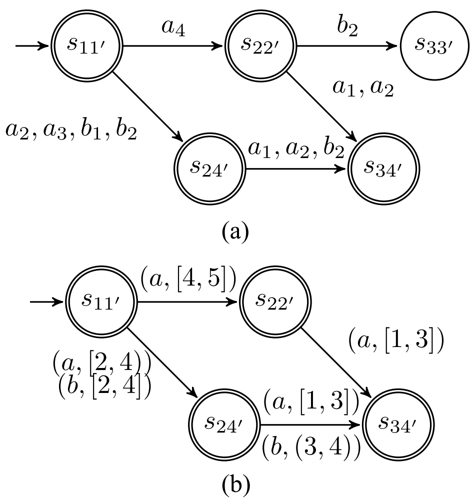

Fig. 6. (a) $\mathcal{B}^p$, the product of $\mathcal{B}'$ and $\mathcal{B}_{\text{secret}}^{\text{comp}}$. (b) $\mathcal{B}_{ns}$.

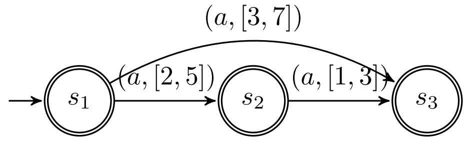

Fig. 7. $\mathcal{B} \uparrow_{\Sigma_o}$, the projection FA from $\mathcal{B}$.

abbreviations for the nine labels in order. And the resulting automata are in Fig. 5(a) and (b).

By constructing the product of $\mathcal{B}'$ and $\mathcal{B}_{\text{secret}}^{\text{comp}}$, we can obtain FA $\mathcal{B}^p$, as shown in Fig. 6(a). Then $\mathcal{B}_{ns}$ is constructed, by simplifying $\mathcal{B}^p$, depicted in Fig. 6(b).

Second, we compute the projection of $\mathcal{B}$ and $\mathcal{B}_{ns}$, denoted as $\mathcal{B} \uparrow_{\Sigma_o}$ and $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$, respectively, with respect to $\Sigma_o = \{b\}$.

4) Construct $\mathcal{B} \uparrow_{\Sigma_o}$: We first construct $\mathcal{B}_\tau$ with two transitions: $(s_1, [2,4], s_2)$ and $(s_2, [3,4], s_3)$. All the three states are initial and accepting. $\Lambda_{s_1 s_2} = [2,4]$, $\Lambda_{s_2 s_3} = [3,4]$, $\Lambda_{s_1 s_3} = [5,8]$, and all the others are equal to $\{0\}$. Then in our $\mathcal{B} \uparrow_{\Sigma_o}$, the transitions are $(s_1, (a, [2,5]), s_2)$, $(s_2, (a, [1,3]), s_3)$ and $(s_1, (a, [3,7]), s_3)$. $S = F_p = \{s_1, s_2, s_3\}$, and $\text{Init}_p = \{s_1\}$, as shown in Fig. 7.

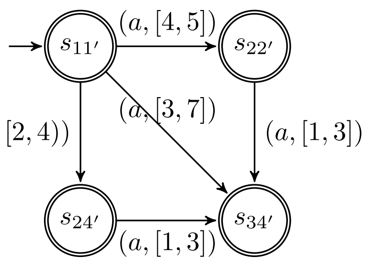

Fig. 8. $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$, the projection FA from $\mathcal{B}_{ns}$.

5) Construct $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$: The construction is similar to that of $\mathcal{B} \uparrow_{\Sigma_o}$, shown in Fig. 8.

(6) Construct another FA $(\mathcal{B}' \uparrow_{\Sigma_o}) \times (\mathcal{B}'_{ns} \uparrow_{\Sigma_o})^{\text{comp}}$ in order to compare languages accepted by $\mathcal{B} \uparrow_{\Sigma_o}$ and $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$. This is similar to steps 2 and 3, so we omit the details here.

As $(\mathcal{B}' \uparrow_{\Sigma_o}) \times (\mathcal{B}'_{ns} \uparrow_{\Sigma_o})^{\text{comp}}$ accepts nothing, the RTA $\mathcal{A}$ is language-opaque with respect to $L(\mathcal{A}_{\text{secret}})$ and Σ_o.

VI. Implementation

Based on the theory reported above, we have developed a prototypical tool for deciding the language-opacity problem of RTA. The tool is implemented with Python and thus can run on Linux, Window, and MAC smoothly.

The tool needs two input files: one is the system model (`a.json`) and the other is the secret one (`a_secret.json`). Both models are json files of RTA models.

A json file of an RTA model consists of seven parts.

1) "*name*" is the name of the RTA model.
2) "*s*" is the list of states' names.
3) "*sigma*" is the alphabet.
4) "*tran*" is the list of transitions of the RTA model, where each transition is of the key-value form "id: [source, label, guard, target]." The key "id" is the index of the transition, and the value consists of four parts.
    a) "source" is the name of the prestate of the transition.
    b) "target" is the name of the post-state of the transition.
    c) "label" is one of the letters in the alphabet.
    d) "guard" is the union of the time intervals, represented in Dima's [14] normal forms,
5) "*init*" is the name of the initial state, and
6) "*observable*" is the list of observable letters.

Specifically, the json file of the system model $\mathcal{A}$ in Section V-A is in Listing 1 below.

First, we load the json files of two RTA models $\mathcal{A}$ and $\mathcal{A}_{\text{secret}}$, and transform them into two FA $\mathcal{B}$ and $\mathcal{B}_{ns}$, respectively. These two FA accept languages trace-equivalent to $L(\mathcal{A})$ and $L_f(\mathcal{A}) \setminus L(\mathcal{A}_{\text{secret}})$, respectively.

Second, we obtain $\mathcal{B} \uparrow_{\Sigma_o}$ and $\mathcal{B}_{ns} \uparrow_{\Sigma_o}$ which are the projections $\mathcal{B}$ and $\mathcal{B}_{ns}$, respectively, based on the observable action set Σ_o, by constructing $\mathcal{B}_\tau$ and calculating the regular expressions *RE*s corresponding to traces in $\mathcal{B}_\tau$. Union, intersection, complement, addition, and Kleene star operations of Dima's normal forms are implemented according to [14].

```
{
  "name": "A",
  "s": ["s1", "s2", "s3"],
  "sigma": ["a", "b"],
  "tran": {
  "0": ["s1", "b", "[2,4]", "s2"],
  "1": ["s1", "a", "[2,5]", "s2"],
  "2": ["s2", "b", "[3,4]", "s3"],
  "3": ["s2", "a", "[1,3]", "s3"]
  },
  "init": "s1",
  "accept": ["s3"],
  "observable": ["a"]
}
```

Listing 1. Json file of $\mathcal{A}$ (`a.json`).

The well-known Floyd–Warshall–Kleene Algorithm [16]–[18] is utilized to construct the matrix of regular expressions $RE_{i,j}$. For calculating and simplifying the $k+1$ round formula $RE_{i,j}(k+1) = RE_{i,j}(k)+RE_{i,k+1}(k)\cdot RE_{k+1,k+1}(k)^{*}\cdot RE_{k+1,j}(k)$, the Horner's rule [19] is applied where the addition operation, union operation, and $\{0\}$ and $\emptyset$ in Dima's normal forms are treated as the production operation, addition operation, 1 and 0 in the polynomials in rational number domain, respectively.

Finally, after transforming the two finite automata into DFA, we apply the complement and production operations on them again, and obtain $\mathcal{B}\uparrow_{\Sigma_o} \times (\mathcal{B}_{ns}\uparrow_{\Sigma_o})^{\text{comp}}$. Clearly, $\mathcal{A}$ is language-opaque with respect to $\mathcal{A}_{\text{secret}}$ and Σ_o if and only if $L_f(\mathcal{B}\uparrow_{\Sigma_o} \times (\mathcal{B}_{ns}\uparrow_{\Sigma_o})^{\text{comp}})$ is empty.

The output of the tool is as follows: it returns "language-opaque: true" if the RTA $\mathcal{A}$ is language-opaque with respect to $L_f(\mathcal{A}_{\text{secret}})$ and Σ_o; otherwise, it returns "language-opaque: false." Intermediate automata during the computation are also printed, including $\mathcal{A}$, $\mathcal{A}_{\text{secret}}$, $\mathcal{B}$, $\mathcal{B}_{ns}$, $\mathcal{B}\uparrow_{\Sigma_o}$, $\mathcal{B}_{ns}\uparrow_{\Sigma_o}$, and the final automaton $\mathcal{B}\uparrow_{\Sigma_o} \times (\mathcal{B}_{ns}\uparrow_{\Sigma_o})^{\text{comp}}$. Besides, after the deciding procedure, the total elapsed time will be given (in seconds).

The simple example of language-opacity in Section V-A is as a demo in the tool and the result of the example is language-opaque with taking 1.67 s in our test environment (Inter Core i3-5005U at 2.0 GHz and 4-GB DDR3L-1600-MHz RAM). The prototypical tool and its information can be found at https://github.com/Leslieaj/RTAOpacity.

VII. Conclusion

In this paper, we investigate the opacity problems of RTA, including language opacity and initial-state opacity. We prove the two opacity problems both are decidable by reduction to the language inclusion problem of FA. To this end, we first translate a given RTA into an FA with the corresponding timed alphabet, such that the language accepted by the FA and that generated by the RTA are equivalent in the sense of *trace-equivalence* proposed in this paper. Then, in order to guarantee trace-equivalence relation is preserved by the complement and product operation over translated FA, we propose the notions of partitioned timed alphabet and partitioned language. Based on them, we further refine the translated FA to an FA with partitioned timed alphabet, of which the accepted language is still trace-equivalent to the generated language of the original RTA. Then, we define a projection operation on refined FA with partitioned timed alphabets onto the given observable set by removing all unobservable transitions and merging their time durations into the subsequent observable transition. With such projection, we can show that the accepted language of the projection of the refined FA onto the observable set is still trace-equivalent to the projection of the language generated by the original RTA onto the same set. Thus, the decidability of the language and initial-state opacity problems of RTA can be reduced to the regular language inclusion problem.

Applicability: Bryans *et al.* [4] investigated how to formalize security properties like *anonymity*, *noninterference*, etc., which can be essentially reduced to *opacity problems*, using labeled transition systems. Gardey *et al.* [20] generalized these notions to timed setting in the formalism of timed automata, and proved that *noninterference* is still decidable. However, unlike in untimed setting, Cassez [13] proved that for the very restrictive class of ERA, the language opacity problem is already undecidable, that leaves little hope for an algorithmic solution to the opacity problem in dense-time. Dima [14] pointed out that ERA is incomparable with RTA. Fortunately, in this paper, we prove that the language and initial-state opacity problems of RTA both are decidable, which reembarks the hope to automatically verify security properties in timed settings. Actually, many commonly used communication protocols with time can be modeled using RTA, for example, Denning and Sacco [21] extended Needham and Schroeder's key distribution protocols for both single key and public key systems with timestamps in order to avoid attacks by replay. It is not hard to model the extended protocols with RTA as all timing constraints are of the form $|\text{clock} - T| \in [a,b]$, where clock stands for a local clock, and T is the given timestamp. Furthermore, it is possible to model the commonly used authentication technology *Kerberos* [22] using RTA, as its timing part is essentially based on the extended Needham and Schroeder's protocols. As one of major future work, we will investigate how to apply RTA to model more security protocols.

Another interesting future work is to extend HCSP [23]–[25] and HHL [25], [26] to security for modeling and reasoning about more general security properties, and to investigate automatic verification techniques for them based on the work reported in this paper.

References

[1] R. Bailliage and L. Mazaré, "Using unification for opacity properties," in *Proc. Workshop Issues Theory Security (WITS)*, 2004, pp. 165–176.

[2] J. W. Bryans, M. Kounty, and P. Y. A. Ryan, "Modelling opacity using Petri nets," *Electron. Notes Theor. Comput. Sci.*, vol. 121, pp. 101–115, Feb. 2005.

[3] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Verification of state-based opacity using Petri nets," *IEEE Trans. Autom. Control*, vol. 62, no. 6, pp. 2823–2837, Jun. 2017.

[4] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *Int. J. Inf. Security*, vol. 7, no. 6, pp. 421–435, 2008.

[5] A. Saboori and C. N. Hadjicostis, "Verification of infinite-step opacity and complexity considerations," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1265–1269, May 2012.

[6] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of discrete event systems," *Inf. Sci.*, vol. 246, pp. 115–132, Oct. 2013.

[7] A. Saboori and C. N. Hadjicostis, "Verification of K-step opacity and analysis of its complexity," in *Proc. 48th Held Join. 28th Chin. Control Conf. Decis. Control (CDC)*, Shanghai, China, 2009, pp. 205–210.

[8] A. Saboori and C. N. Hadjicostis, "Opacity-enforcing supervisory strategies via state estimator constructions," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1155–1165, May 2012.

[9] C. Keroglou and C. Hadjicostis, "Initial state opacity in stochastic DES," in *Proc. IEEE 18th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Cagliari, Italy, 2013, pp. 1–8.

[10] B. Bérard, K. Chatterjee, and N. Sznajder, "Probabilistic opacity for Markov decision processes," *Inf. Process. Lett.*, vol. 115, no. 1, pp. 52–59, 2015.

[11] B. Bérard, J. Mullins, and M. Sassolas, "Quantifying opacity," in *Proc. 7th Int. Conf. Quant. Eval. Syst.*, 2010, pp. 263–272.

[12] M. Ibrahim, J. Chen, and R. Kumar, "Secrecy in stochastic discrete event systems," in *Proc. 11th IEEE Int. Conf. Netw. Sens. Control*, 2014, pp. 48–53.

[13] F. Cassez, "The dark side of timed opacity," in *Proc. Adv. Inf. Security Assurance*, 2009, pp. 21–30.

[14] C. Dima, "Real-time automata," *J. Automata Lang. Comb.*, vol. 6, no. 1, pp. 3–24, 2001.

[15] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation—International Edition*, 2nd ed. Milan, Italy: Addison-Wesley, 2003.

[16] S. C. Kleene, "Representation of events in nerve nets and finite automata," in *Automata Studies*. Princeton, NJ, USA: Princeton Univ. Press, 1956, pp. 3–41.

[17] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

[18] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[19] P. Borwein and T. Erdélyi, *Polynomials and Polynomial Inequalities*. New York, NY, USA: Springer-Verlag, 1995.

[20] G. Gardey, J. Mullins, and O. H. Roux, "Non-interference control synthesis for security timed automata," *Electr. Notes Theor. Comput. Sci.*, vol. 180, no. 1, pp. 35–53, 2007.

[21] D. E. Denning and G. M. Sacco, "Timestamps in key distribution protocols," *Commun. ACM*, vol. 24, no. 8, pp. 533–536, 1981.

[22] J. Kohl and B. C. Neuman, "The Kerberos network authentication service (V5)," RFC 1510, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–112, 1993.

[23] J. He, "From CSP to hybrid systems," in *A Classical Mind, Essays in Honour of C.A.R. Hoare*. New York, NY, USA: Prentice-Hall, 1994, pp. 171–189.

[24] C. Zhou, J. Wang, and A. P. Ravn, "A formal description of hybrid systems," in *Hybrid Systems* (LNCS 1066). Heidelberg, Germany: Springer, 1996, pp. 511–530.

[25] N. Zhan, S. Wang, and H. Zhao, *Formal Verification of Simulink/Stateflow Diagrams*. New York, NY, USA: Springer, 2017.

[26] J. Liu *et al.*, "A calculus for hybrid CSP," in *Proc. APLAS*, LNCS, vol. 6461. Heidelberg, Germany: Springer, 2010, pp. 1–15.

Lingtai Wang received the B.Sc. degree in information and computation sciences from Peking University, Beijing, China, in 2015. She is currently pursuing the Ph.D. degree with the Institute of Software, Chinese Academy of Sciences, Beijing, and the University of Chinese Academy of Sciences, Beijing.

Her current research interest includes modeling and verification of hybrid systems.

Naijun Zhan received the B.Sc. degree in mathematics and the M.Sc. degree in computer science from Nanjing University, Nanjing, China, in 1993 and 1996, respectively, and the Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences, Beijing, China.

He was a Research Fellow with the Faculty of Mathematics and Information, University of Mannheim, Mannheim, Germany, from 2001 to 2004. Since then he joined the Institute of Software, Chinese Academy of Sciences, as an Associate Research Professor, and was promoted to be a Full Research Professor in 2008. His current research interests include real-time, embedded and hybrid systems, program verification, concurrent computation models, and modal and temporal logics.

Jie An received the bachelor's and master's degrees in software engineering from Tongji University, Shanghai, China, in 2012 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the School of Software Engineering.

Since 2016, he has been an intern PhD student with the State Key Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China. His current research interests include formal verification, embedded systems, and machine learning.