Decidability of the initial-state opacity of real-time automata

Lingtai Wang[1,2] and Naijun Zhan[1,2]

[1] State Key Lab. of Comp. Sci., Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. In this paper, we investigate the initial-state opacity of real-time automata. A system is called *initial-state opaque* if an intruder with partial observability is unable to determine whether or not the execution starts from a secret state. In order to prove that the initial-state opacity problem is decidable, we first calculate the lapse of time between each pair of observable events. Two real-time automata are constructed which accept the projection of languages from secret initial states and non-secret ones, respectively. Then, the two real-time automata are further transformed into trace-equivalent finite-state automata. Subsequently, we adapt complement and product on the finite-state automata, and check accepting language of the finally-obtained automaton. The system is initial-state opaque if it accepts nothing or only empty trace, and not initial-state opaque otherwise.

Keywords: Real-time automata, initial-state opacity, decidability, trace-equivalence

1 Introduction

In the wake of development of network communications and online services, security and privacy have become more significant and thus received more and more attention. Opacity is an information flow property aiming at keeping the "secret" of a system opaque to its outsider (called the intruder). There are two types of "secrets": subsets of traces and subsets of states. This divides opacity properties into language-based opacity and state-based opacity. The intruder is believed to know 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. This paper focuses on initial-state opacity, which is state-based, that is, the secret S is a set of states. The system is initial-state opaque if the intruder can never determine whether it starts from a secret state or a non-secret one no matter what he has observed. Examples from tracking problems in sensor networks have been used to motivate initial-state opacity in [1], where the sensor network only has partial observation.

Systems being investigated are often modelled as discrete event systems (DES), for example, Petri nets [2, 3], labeled transition systems (LTS) [4] and finite-state automata (FSA) [1,5–7]. Probabilistic models are also taken into consideration, such as [8–11]. However, in [12], the notion of opacity was extended to dense-time systems, with the result that the (language-based) opacity problem is already undecidable for a very restrictive class of event-recording automata (ERA).

As time is an important attack vector against secure systems, we extend the notion of initial-state opacity to real-time automata [13]. Real-time automata is a class of timed automata with a single clock which is reset at each transition, also regarded as finite automata with time information for each transition. Classical results for finite automata can thus be extended to real-time automata such as Kleene's theorem, Pumping Lemma and the closure under complementation [13]. Besides, as pointed out in [13], RTA is not comparable with ERA.

Our analysis mainly focuses on calculating time taken by unobservable transitions and then constructing two real-time automata accepting the projection of languages from secret initial states and non-secret ones respectively. A relationship between languages of real-time automata and their corresponding finite-state automata, called trace-equivalence, is introduced, so that the initial-opacity problem is transformed into the problem of language inclusion of finite-state automata. Thus, the initial-state opacity problem is proved to be *decidable.*

The remainder of this paper is organized as follows. In Section 2, we recall preliminaries for finite-state automata, regular expressions, real-time automata, and the initial-state opacity problem of real-time automata. The correspondence of real-time automata and finite-state automata is introduced in Section 3. Section 4 provides a procedure to determine whether a real-time automaton is initial-state opaque w.r.t. a given set of secret states and an observable alphabet, and Section 5 concludes this paper.

2 Preliminaries

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

Let E, a set of events, be the *alphabet.* A *word* or *string* over E is a finite sequence $w = a_1 a_2 \dots a_n$, where $a_i \in E$ for $i = 1, 2, \dots, n$. $|w| = n$ is the length of w. ε is the empty word, whose length $|\varepsilon| = 0$. E^* is the set of all the finite words over E including ε. L is a *language* over E if $L \subseteq E^*$.

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

Concatenation: Let $L_1, L_2 \subseteq E^*$, the concatenation $L_1 L_2 = \{s_1 s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}$.

Kleene closure: Let $L \subseteq E^*$, and $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^k = (L^{k-1})L$ for $k > 1$, then the Kleene closure of L is $L^* = \bigcup_{k \in \mathbb{N}} L^k = \{\varepsilon\} \cup L \cup LL \cup \cdots$.

Projection: Given E and a subset $E_o \subseteq E$, we can define a projection $P_{E_o} : E^* \to E_o^*$, where

$$P_{E_o}(\varepsilon) = \varepsilon$$
$$P_{E_o}(as) = \begin{cases} aP_{E_o}(s), & \text{if } a \in E_o \\ P_{E_o}(s), & \text{otherwise} \end{cases}, \text{ for } a \in E \text{ and } s \in E^*.$$

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

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 $w_t = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$, where $0 \leq t_1 \leq t_2 \leq \dots \leq t_n$, meaning that a_i occurs at t_i successively for $1 \leq i \leq n$. $TW^*(\Sigma)$ denotes the set of all timed words over Σ. A subset of $TW^*(\Sigma)$ is a *timed language*. If $\Sigma_o \subseteq \Sigma$ is the observable alphabet, $P_{\Sigma_o,t}$ denotes the projection from $TW^*(\Sigma)$ into $TW^*(\Sigma_o)$. For example, if $w_t = (a, 2)(b, 3)(a, 5)(b, 8)$, $P_{\{b\},t}(w_t) = (b, 3)(b, 8)$ and $P_{\{a\},t}(w_t) = (a, 2)(a, 5)$.

2.1 Finite-state automata and regular expressions

Automata are a kind of well-known model to study discrete transition systems and their behaviours. Finite-state automata (FAs) are automata with finitely many states. They can be deterministic or non-deterministic.

Definition 1. – *A deterministic finite-state automaton (DFA) is a 5-tuple $A_d = (S, \Sigma, \delta, s_0, F)$, where*
  - *S is a finite set of states;*
  - *Σ is a finite alphabet;*
  - *$\delta : S \times \Sigma \to S$ is the transition relation, a partial function on $S \times \Sigma$;*
  - *$s_0 \in S$ is the initial state; and*
  - *$F \subseteq S$ is the set of accepting states.*

– *A non-deterministic finite-state automaton (NFA) is a 5-tuple $\mathcal{A}_n = (S, \Sigma \cup \{\varepsilon\}, \delta, Init, F)$, where*
  - *S is a finite set of states;*
  - *Σ is a finite alphabet;*
  - *$\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;*
  - *$Init \subseteq S$ is the set of initial states; and*
  - *$F \subseteq S$ 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 $\delta(s, a)$, and no ε-transition.

For an NFA $\mathcal{A}$, if $s_2 \in \delta(s_1, \sigma)$, (s_1, σ, s_2) is called a σ-transition, written as $s_1 \xrightarrow{\sigma} s_2$. A *run* of $\mathcal{A}$ is either a single state s_0, where $s_0 \in Init$, or a sequence $s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \cdots s_{n-1} \xrightarrow{\sigma_n} s_n$, where $n > 0$, $s_0 \in Init$, $\sigma_i \in \Sigma \cup \{\varepsilon\}$ and $s_i \in \delta(s_{i-1}, \sigma_{i-1})$ for $1 \leq i \leq n$. The *trace* of the run s_0 is ε, and the *trace* of the sequence from s_0 to s_n is the finite word obtained by projecting $\sigma_1\sigma_2 \dots \sigma_n$ onto

Σ^*, that is, the string $a_1a_2\dots a_m$ obtained by removing each ε from $\sigma_1\sigma_2\dots\sigma_n$; hence the length of the trace is m, less than or equal to n. An *accepting run* is a run ending in a state $s_n \in F$. The language generated by $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of traces of runs of $\mathcal{A}$; 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.

Two automata are called *language-equivalent*, or *equivalent* for short, if they generate and accept the same languages. An NFA $\mathcal{A}_n = (S, \Sigma, \delta, \mathit{Init}, F)$ can be transformed into an equivalent DFA $\mathcal{A}_d = (S', \Sigma, \delta', \mathit{Init}', F')$ 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 $\varepsilon R(s,a)$ the set of states which are reachable from state s via one a-transition together with ε-transitions before and after it. Then in $\mathcal{A}_d$, $S' = 2^S$; $\delta'(S_1, a) = \bigcup_{s_1 \in S_1} \varepsilon R(s_1, a)$; $\mathit{Init}' = \varepsilon R(s_0, \varepsilon)$; $F' = \{S_1 \mid S_1 \cap F \neq \emptyset\}$.

Regular expressions are another way to describe regular languages.

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

1. *(Base Clause): $\emptyset$, ε, $a \in \Sigma$ are regular expressions, where $\emptyset$ denotes the empty set, ε denotes the set $\{\varepsilon\}$, and a denotes the set $\{a\}$ for $a \in E$.*
2. *(Inductive Clause): If r, r_1, r_2 are regular expressions, then $r_1 \cdot r_2$, $r_1 + r_2$, r^* are regular expressions. $r_1 \cdot r_2$ denotes the concatenation of language denoted 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 denotes by r.*
3. *(External Clause): Regular expressions can only be constructed by applying 1 and 2.*

Theorem 1 (Kleene's Theorem). *Any regular language is accepted by a finite automaton; any language accepted by a finite automaton is regular.*

Complement and product operations on DFAs Consider a DFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$. The complement automaton $\mathcal{A}^{comp}$ 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 $s_{new} \notin S$;
2. Augment δ such that it becomes a total function, denoted as δ^{comp}. For all $(s,a) \in S \times \Sigma$, if $\delta(s,a)$ is defined, let $\delta^{comp}(s,a) = \delta(s,a)$; if $\delta(s,a)$ is not defined, let $\delta^{comp}(s,a) = s_{new}$. Also $\delta(s_{new}, a) = s_{new}$ for each $a \in \Sigma$. After that Σ^* becomes the language generated, while the language accepted keeps unchanged;
3. Let the accepting set of states be $(S \setminus F) \cup \{s_{new}\}$.

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

Given two DFAs $\mathcal{A}_1 = (S_1, \Sigma_1, \delta_1, s_{0,1}, F_1)$ 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: $S^p = S_1 \times S_2$; $\Sigma^p = \Sigma_1 \cap \Sigma_2$; $\delta^p((s_1, s_2), a) = (s_1', s_2')$ if

$\delta(s_1, a) = s'_1$ and $\delta(s_2, a) = s'_2$, and is not defined otherwise; $s^p_0 = (s_{0,1}, s_{0,2})$; $F^p = F_1 \times F_2$.

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

2.2 Real-time automata

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

Definition 3. *A real-time automaton is a 6-tuple $\mathcal{A} = (S, \Sigma, \Delta, \mathit{Init}, F, \mu)$, where*

- *S is a finite set of states;*
- *Σ is a finite alphabet;*
- *$\Delta \subseteq S \times \Sigma \times S$ is the transition relation;*
- *$\mathit{Init} \subseteq S$ is the set of initial states;*
- *$F \subseteq S$ is the set of accepting states; and*
- *$\mu : \Delta \to 2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\}$ is the time labelling function, whose range, $\mu(\Delta)$, is usually a set of intervals whose endpoints are in $\mathbb{N} \cup \{+\infty\}$ or $\mathbb{Q}_{\geq 0} \cup \{+\infty\}$.*

A *transition* $(s_1, a, s_2) \in \Delta$ starts in s_1, ends in s_2 and is labelled by a. Transitions of the form (s_1, a, s_2) are called a-transitions. Δ_a denotes the set of all a-transitions. Pre_a and Post_a denotes the set of states from which and to which are a-transitions respectively, i.e., $\mathit{Pre}_a = \{s_1 \mid \exists(s_1, a, s_2) \in \Delta\}$ and $\mathit{Post}_a = \{s_2 \mid \exists(s_1, a, s_2) \in \Delta\}$. A *run* of $\mathcal{A}$ is either a single initial state s_0 from Init or a finite sequence $\rho = s_0 \xrightarrow[\lambda_1]{a_1} s_1 \xrightarrow[\lambda_2]{a_2} \cdots s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$, where $n > 0$, $s_0 \in \mathit{Init}$, $(s_{i-1}, a_i, s_i) \in \Delta$, and $\lambda_i \in \mu(s_{i-1}, a_i, s_i)$ for $i \geq 1$. The *trace* of a run ρ, denoted by $\mathit{trace}(\rho)$, is defined as follows: if $\rho = s_0$, $\mathit{trace}(\rho) = \varepsilon_t$, where subscript "$t$" is used to emphasize the time factor; if ρ is of the form $s_0 \xrightarrow[\lambda_1]{a_1} s_1 \xrightarrow[\lambda_2]{a_2} \ldots s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$, $\mathit{trace}(\rho)$ is the timed word $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ where $t_i = \sum_{j=1}^{i} \lambda_j$, for $i = 1, \ldots, n$. Let $\mathit{Tr}(s_0)$ be the set of traces of runs from state s_0, and $\mathit{Tr}(S_0)$ be the set of traces of runs from any state $s_0 \in S_0$, i.e., $\mathit{Tr}(S_0) = \bigcup_{s_0 \in S_0} \mathit{Tr}(s_0)$. $L(\mathcal{A}) = \mathit{Tr}(\mathit{Init}) = \bigcup_{s_0 \in \mathit{Init}} \mathit{Tr}(s_0)$, is called the timed language generated by $\mathcal{A}$, and $L_f(\mathcal{A}) = \{\mathit{trace}(\rho) \mid \rho$ starts from $s_0 \in S_0$ and ends in $s_n \in F\}$ is the set of traces accepted by $\mathcal{A}$.

Example 1. In Fig.1, transitions are depicted as arrows, with their labels from the alphabet $\{a, b\}$ above and time-labels below. For the real-time automaton $\mathcal{A}_1$, $\mathit{Tr}(s_0) = \{\varepsilon_t\} \cup \{(a, t_a) \mid t_a \in [1, 2]\} \cup \{(a, t_a)(b, t_b) \mid t_a \in [1, 2], t_b \in [2, 3]\}$, and $\mathit{Tr}(s_3) = \{\varepsilon_t\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$.

$\mathcal{A}_1$ generates $L(\mathcal{A}_1) = \{\varepsilon_t\} \cup \{(a, t_a) \mid t_a \in [1, 2]\} \cup \{(a, t_a)(b, t_b) \mid t_a \in [1, 2], t_b \in [2, 3]\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$ and accepts $L_f(\mathcal{A}_1) = \{(a, t_a)(b, t_b) \mid t_a \in [1, 2], t_b \in [2, 3]\} \cup \{(b, t_b) \mid t_b \in [3, 4]\}$. $\square$

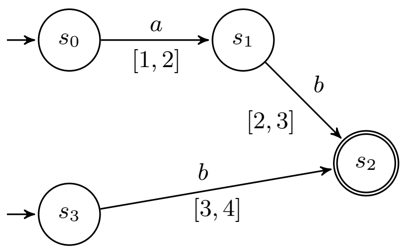


Fig. 1: A real-time automaton $\mathcal{A}_1$

2.3 Initial-state opacity of real-time automata

Given a real-time automaton $\mathcal{A}$ with an alphabet Σ and an observable alphabet $\Sigma_o \subseteq \Sigma$, intruders can only observe timed words in $P_{\Sigma_o,t}(L(\mathcal{A}))$. Suppose we have a set of secret states S_{secret}. In this case, can intruders detect whether the current run of the system starts from the secret set S_{secret} according to what they have observed? This is considered in the initial-state opacity problem. Formally,

Definition 4. *Given a real-time automaton $\mathcal{A} = (S, \Sigma, \Delta, Init, F, \mu)$, an observable alphabet $\Sigma_o \subseteq \Sigma$ and a secret set of states $S_{secret} \subseteq S$, $\mathcal{A}$ is* initial-state opaque *w.r.t. S_{secret} and Σ_o iff for all $s_0 \in Init \cap S_{secret}$ and all $w \in Tr(s_0)$, $\exists s'_0 \in Init \setminus S_{secret}$, $\exists w' \in Tr(s'_0)$ s.t.*

$$P_{\Sigma_o,t}(w) = P_{\Sigma_o,t}(w'),$$

or equivalently,

$$P_{\Sigma_o,t}(Tr(Init \cap S_{secret}))) \subseteq P_{\Sigma_o,t}(Tr(Init \setminus S_{secret})).$$

The *initial-state opacity problem* of real-time automata is thus expressed as follows: Is a real-time automaton $\mathcal{A} = (S, \Sigma, \Delta, Init, F, \mu)$ initial-state opaque w.r.t. some given secret set $S_{secret} \subseteq S$ and $\Sigma_o \subseteq \Sigma$?

In the following, we would like to use L_s and L_{ns} instead of $P_{\Sigma_o,t}(Tr(Init \cap S_{secret})))$ and $P_{\Sigma_o,t}(Tr(Init \setminus S_{secret}))$ respectively for sake of convenience.

Note that in the initial-state opacity problem, the set of accepting states of the real-time automaton $\mathcal{A}$ does not play any role. Therefore, without loss of generality, each state of the real-time automaton under study can be regarded as an accepting state such that $L_f(\mathcal{A}) = L(\mathcal{A})$.

Example 2 (Ctd.). We still consider the automata $\mathcal{A}_1$ shown in Fig.1. Let $\Sigma_o = \{b\}$ and $S_{secret} = \{s_0\}$. If $P_{\Sigma_o,t}(w_t) = (b, 3.5)$, possible runs include $\rho_1 = s_0 \xrightarrow[1]{a} s_1 \xrightarrow[2.5]{b} s_2$ and $\rho_2 = s_3 \xrightarrow[3.5]{b} s_2$. In this case the intruders are incapable of ascertaining the secret. If $P_{\Sigma_o,t}(w_t) = (b, 5)$, we can easily know that $w_t = (a, 2)(b, 5)$ and the unique run is $\rho = s_0 \xrightarrow[2]{a} s_1 \xrightarrow[3]{b} s_2$. Thus, the secret is exposed in this case. From the above, $\mathcal{A}_1$ is not initial-state opaque w.r.t. $\{s_0\}$ and $\{b\}$.

However, $\mathcal{A}_1$ is initial-state opaque w.r.t. $\{s_3\}$ and $\{b\}$. This is because there always exists a run $s_0 \xrightarrow[1]{a} s_1 \xrightarrow[t-1]{b} s_2$ with the same projection as $s_3 \xrightarrow[t]{b} s_2$ with $3 \leq t \leq 4$.

From the perspective of set theory, $P_{\{b\},t}(Tr(s_3)) = \{(b,t) \mid t \in [3,4]\} \subsetneq P_{\{b\},t}(Tr(s_0)) = \{(b,t) \mid t \in [3,5]\}$, so $\mathcal{A}_1$ is not initial-state opaque w.r.t. $\{s_0\}$ and $\{b\}$, and is initial-state opaque w.r.t. $\{s_3\}$ and $\{b\}$. $\square$

If there exist two real-time automata which accept the two languages L_s and L_{ns}, respectively, then we can solve the initial-state opacity problem by checking whether the accepting language of the former is included in that of the latter. This checking can be achieved by utilizing trace-equivalence relation given in the next section. The basic idea is to translate the two real-time automata into their trace-equivalent finite-state automata, and construct another finite-state automaton $\mathcal{A}^p_{dfa}$ according to the two resulting finite-state automata. The fact that $\mathcal{A}^p_{dfa}$ accepts nothing or ε implies that $L_s \subseteq L_{ns}$, i.e., the original real-time automaton is initial-state opaque.

3 Correspondence between NFAs and real-time automata

As real-time automata and non-deterministic automata have very similar structures, a real-time automaton with alphabet Σ can be translated into an NFA with alphabet $\Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\})$.

Given a real-time automaton $\mathcal{A} = (S, \Sigma, \Delta, Init, F, \mu)$, let μ_a denote the set of time information of all a-transitions, that is, $\mu_a = \{\mu(s_1, a, s_2) \mid (s_1, a, s_2) \in \Delta\}$, which is finite since Δ is finite. Each element of μ_a is a non-empty subset Λ_a of $\mathbb{R}_{\geq 0}$, such as a point, an interval, or a more complicated set. For each μ_a, a partition of $\mathbb{R}_{\geq 0}$ (i.e., a set of $\mathbb{R}_{\geq 0}$'s non-empty subsets satisfying each real number $x \geq 0$ is in one and only one of those subsets) should be constructed such that any $\Lambda_a \in \mu_a$ is the union of some elements from the partition. Here we define a function I to compute a partition of $\mathbb{R}_{\geq 0}$ based on a finite $C \in 2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\}$. I is defined by induction: if $|C| = 1$, say $\mu_a = \{\Lambda\}$, the partition is $\mathcal{I}(C) = \{\Lambda, \mathbb{R}_{\geq 0} \setminus \Lambda\} \setminus \{\emptyset\}$; if $|C| = k$ with $\mathcal{I}(C) = \{I_1, \ldots, I_{m_C}\}$ and $\Lambda \notin C$, $\mathcal{I}(C \cup \{\Lambda\}) = \{I_1 \cap \Lambda, I_1 \setminus \Lambda, \ldots, I_{m_C} \cap \Lambda, I_{m_C} \setminus \Lambda,\} \setminus \{\emptyset\}$. Additionally, it can be easily proved from the definition that $|\mathcal{I}(C)|$ is no more than $2^{|C|}$. So we can obtain the partition $\mathcal{I}(\mu_a)$ satisfying the aforementioned constraint. For instance, if $\mu_a = \{[2,5], [3,6]\}$, we can construct $\{[3,5], [2,3), (5,6], [0,2) \cup [6,+\infty)\}$ as one partition based on Λ_a. Then a non-deterministic finite-state automaton $\mathcal{A}_{nfa} = (S_{nfa}, \Sigma_{nfa}, \delta_{nfa}, Init_{nfa}, F_{nfa})$ can be constructed.

Definition 5. *Given $\mathcal{A} = (S, \Sigma, \Delta, Init, F, \mu)$, the corresponding NFA $\mathcal{A}_{nfa} = (S_{nfa}, \Sigma_{nfa}, \delta_{nfa}, Init_{nfa}, F_{nfa})$ can be constructed as follows.*

- *$S_{nfa} = S$;*
- *$\Sigma_{nfa} = \bigcup_{a \in \Sigma} \big(\{a\} \times \mathcal{I}(\mu_a)\big)$;*
- *$\delta_{nfa}(s_1, (a, I)) = \{s_2 \mid (s_1, a, s_2) \in \Delta \wedge I \subseteq I',$ for some $I' \in \mu(s_1, a, s_2)\}$;*

- $Init_{nfa} = Init$;
- $F_{nfa} = F$.

Example 3. The real-time automaton in Fig.1 can be translated into the finite-state automaton in Fig.2. Note that $\mathcal{I}(\mu_a) = \{[1,2],[0,1)\cup(2,+\infty)\}$ and $\mathcal{I}(\mu_b) = \{[3,3],[2,3),(3,4],[0,2)\cup(4,+\infty)\}$ and that the alphabet of the finite-state automaton is $\{a\}\times I(\mu_a)\cup\{b\}\times I(\mu_b)$. □

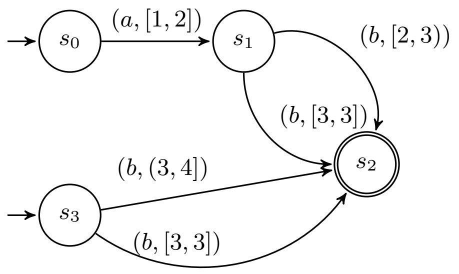


Fig. 2: $\mathcal{A}_2$, the corresponding FA of $\mathcal{A}_1$

Nevertheless, what needs special attention is that languages generated from the two kinds of automata are different. Obviously, the language generated by a real-time automaton $\mathcal{A}$ is a subset of $TW^*(\Sigma)$. By contrast, the language generated by the corresponding finite-state automaton $\mathcal{A}_{nfa}$ is a subset of $(\Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\}))^*$. The relationship between $L(\mathcal{A})$ and $L(\mathcal{A}_{nfa})$ can be described using the trace-equivalence relation defined below.

Definition 6. *Given L_1 a timed language over Σ and L_2 a language over $\Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\})$, L_2 is said to be trace-equivalent to L_1, denoted by $L_2 \approx_{tr} L_1$, if*

0. *$\varepsilon_t \in L_1$ iff $\varepsilon \in L_2$;*
1. *If any timed word $w_t = (a_1,t_1)(a_2,t_2)\ldots(a_n,t_n) \in L_1$, then there exists some $w = (a_1,\Lambda_1)(a_2,\Lambda_2)\ldots(a_n,\Lambda_n) \in L_2$ such that $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \leq n$;*
2. *If $w = (a_1,\Lambda_1)(a_2,\Lambda_2)\ldots(a_n,\Lambda_n) \in L_2$, then all timed words of the form $w_t = (a_1,t_1)(a_2,t_2)\ldots(a_n,t_n)$ with $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \leq n$ are in L_1.*

Lemma 1. *For a given real-time automaton $\mathcal{A} = (S,\Sigma,\Delta,Init,F,\mu)$ and its corresponding NFA $\mathcal{A}_{nfa} = (S_{nfa},\Sigma_{nfa},\delta_{nfa},Init_{nfa},F_{nfa})$ as defined in Def.5, $L_f(\mathcal{A}_{nfa}) \approx_{tr} L_f(\mathcal{A})$.*

Proof. $L_f(\mathcal{A}) = \emptyset$ iff $L_f(\mathcal{A}_{nfa}) = \emptyset$. This is because $L_f(\mathcal{A}) = \emptyset$ iff no accepting states are reachable in $\mathcal{A}$, iff no accepting states are reachable in $\mathcal{A}_{nfa}$, iff $L_f(\mathcal{A}_{nfa}) = \emptyset$. So, in this case, trivially $L_f(\mathcal{A}_{nfa}) \approx_{tr} L_f(\mathcal{A})$.

If $\varepsilon_t \in L_f(\mathcal{A})$, a possible run is s_0, where $s_0 \in \mathit{Init} \cap F$. Thus, $s_0 \in \mathit{Init}_{nfa} \cap F_{nfa}$ according to Def.5. Hence, $\mathcal{A}_{nfa}$ has a run s_0 whose trace is ε, i.e., $\varepsilon \in L_f(\mathcal{A}_{nfa})$. On the contrary, suppose $\varepsilon \in L_f(\mathcal{A}_{nfa})$, which implies there exists an initial state $s_0 \in \mathit{Init}_{nfa} \cap F_{nfa}$. It follows that $s_0 \in \mathit{Init} \cap F$ according to Def.5. Hence, $\varepsilon_t \in L_f(\mathcal{A})$.

Suppose $w_t = (a_1, t_1) \dots (a_n, t_n) \in L_f(\mathcal{A})$, where $n \geq 1$, then there exists a run of $\mathcal{A}$, say $\rho = s_0 \xrightarrow[\lambda_1]{a_1} s_1 \xrightarrow[\lambda_2]{a_2} \cdots s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$ such that $s_0 \in \mathit{Init}$, $s_n \in F$, $(s_{i-1}, a_i, s_i) \in \Delta$, and $\lambda_i \in \mu(s_{i-1}, a_i, s_i)$ for $i \geq 1$. So there exists a run of $\mathcal{A}_{nfa}$, that is, $\rho' = s_0 \xrightarrow{(a_1, \Lambda_1)} s_1 \xrightarrow{(a_2, \Lambda_2)} \cdots s_{n-1} \xrightarrow{(a_n, \Lambda_n)} s_n$, where $s_0 \in \mathit{Init}_{nfa}$ and $s_n \in F_{nfa}$, according to Def.5. Thus, there exists a $w = (a_1, \Lambda_1)(a_2, \Lambda_2) \dots (a_n, \Lambda_n) \in L_f(\mathcal{A}_{nfa})$ such that $t_1 \in \Lambda_1$ and $(t_i - t_{i-1}) \in \Lambda_i$ for $1 < i \leq n$, where $\Lambda_i \subseteq I_i$, for some $I_i \in \mu(s_{i-1}, a_i, s_i)$ and $1 \leq i \leq n$.

Given a word $w = (a_1, \Lambda_1)(a_2, \Lambda_2) \dots (a_n, \Lambda_n) \in L_f(\mathcal{A}_{nfa})$, there must be a run of the form $s_0 \xrightarrow{(a_1, \Lambda_1)} s_1 \xrightarrow{(a_2, \Lambda_2)} \cdots s_{n-1} \xrightarrow{(a_n, \Lambda_n)} s_n$, according to Def.5. Hence $(s_{i-1}, a_i, s_i) \in \Delta$ with $\Lambda_i \subseteq I_i$, for some $I_i \in \mu(s_{i-1}, a_i, s_i)$ and $1 \leq i \leq n$. It follows that there exists a run $s_0 \xrightarrow[\lambda_1]{a_1} s_1 \xrightarrow[\lambda_2]{a_2} \cdots s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$ and a timed word $w_t = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n) \in L_f(\mathcal{A})$ such that $t_1 \in I_1$ and $(t_i - t_{i-1}) \in I_i$ for $1 < i \leq n$. $\square$

In order to consider complement and intersection over trace-equivalent languages, we should put some restrictions over these languages over $\Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\})$, for example, the *partitioned* language defined below.

Definition 7. *A language L over an alphabet $E \subseteq \Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\})$, where Σ is finite, is called to be partitioned, if for any $a \in \Sigma$, $\mathfrak{P}_a = \{I_a : (a, I_a) \in E\}$ is a partition of $\mathbb{R}_{\geq 0}$.*

The accepting language of the NFA defined in Def.5 is partitioned, since $I(\mu_a)$ is a partition of $\mathbb{R}_{\geq 0}$ for each $a \in \Sigma$.

Lemma 2. *If L_1 is a timed language over Σ, L_2 is a partitioned language over $E = \bigcup_{a \in \Sigma} (\{a\} \times \mathfrak{P}_a)$ where each $\mathfrak{P}_a$ is a partition of $\mathbb{R}_{\geq 0}$, and $L_2 \approx_{tr} L_1$ as defined in Def.6, then it also holds that $(E^* \setminus L_2) \approx_{tr} (TW^*(\Sigma) \setminus L_1)$.*

Proof. $\varepsilon_t \in \Sigma \setminus L_1 \Leftrightarrow \varepsilon_t \notin L_1 \Leftrightarrow \varepsilon \notin L_2 \Leftrightarrow \varepsilon \in E^* \setminus L_2$.

Suppose $w_t = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n) \in TW^*(\Sigma) \setminus L_1$, then there must be $I_{a_i} \ni t_i - t_{i-1}$ for each i (t_0 is set to 0 here). Thus, it follows (a_1, I_{a_1}) (a_2, I_{a_2}) $\dots$ $(a_n, I_{a_n}) \in E^*$. $(a_1, I_{a_1})(a_2, I_{a_2}) \dots (a_n, I_{a_n})$ is not in L_2, otherwise w_t would be in L_1. So $(a_1, I_{a_1})(a_2, I_{a_2}) \dots (a_n, I_{a_n})$ is in $E^* \setminus L_2$.

Suppose $w = (a_1, \Lambda_1)(a_2, \Lambda_2) \dots (a_n, \Lambda_n) \in E^* \setminus L_2$, let $w_t = (a_1, t_1)$ (a_2, t_2) $\dots$ (a_n, t_n) be any timed word with $t_1 \in \Lambda_1$ and $t_i - t_{i-1} \in \Lambda_i$ for $i = 2, \dots, n$. It holds that $w_t \notin L_1$, otherwise there would exist some $w' = (a_1, \Lambda'_1)$ (a_2, Λ'_2) $\dots$ (a_n, Λ'_n) such that $t_1 \in \Lambda'_1$ and $t_i - t_{i-1} \in \Lambda'_i$ for $1 < i \leq n$, and therefore $\Lambda_i = \Lambda'_i$ and $w = w'$, which is a contradiction. So w_t is in $TW^*(\Sigma) \setminus L_1$. $\square$

Lemma 3. *If L_1, L_3 are timed languages over Σ, L_2, L_4 are partitioned languages over the same alphabet $E = \bigcup_{a\in\Sigma}(\{a\} \times \mathfrak{P}_a)$ where $\mathfrak{P}_a$ is a partition of $\mathbb{R}_{\geq 0}$, and $L_2 \approx_{tr} L_1$ and $L_4 \approx_{tr} L_3$ as defined in Def.6, it also holds that $(L_2 \cap L_4) \approx_{tr} (L_1 \cap L_3)$.*

Proof. $\varepsilon_t \in L_1 \cap L_3 \Leftrightarrow \varepsilon_t \in L_1 \wedge \varepsilon_t \in L_3 \Leftrightarrow \varepsilon \in L_2 \wedge \varepsilon \in L_4 \Leftrightarrow \varepsilon \in L_2 \cap L_4$.

If $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n) \in L_1 \cap L_3$, then $w_t \in L_1 \wedge w_t \in L_3$. There exists a $w^2 = (a_1, \Lambda_1^2)(a_2, \Lambda_2^2)\dots(a_n, \Lambda_n^2) \in L_2$ such that $t_i - t_{i-1} \in \Lambda_i^2$ for each i (here $t_0 = 0$), and there exists a $w^4 = (a_1, \Lambda_1^4)(a_2, \Lambda_2^4)\dots(a_n, \Lambda_n^4) \in L_4$ such that $t_i - t_{i-1} \in \Lambda_i^4$ for each i (also $t_0 = 0$). Since L_2 and L_4 are partitioned language over a common alphabet E, Λ_i^2 and Λ_i^4 are both in the partition $\mathfrak{P}_{a_i}$. $\Lambda_i^2 \cap \Lambda_i^4 \neq \emptyset$ means that $\Lambda_i^2 = \Lambda_i^4$ for $i = 1, \dots, n$ and $w^2 = w^4$. So $w^2 \in L_2 \wedge w^2 \in L_4$. Then $w^2 \in L_2 \cap L_4$.

If $w = (a_1, \Lambda_1)(a_2, \Lambda_2)\dots(a_n, \Lambda_n) \in L_2 \cap L_4$, $w \in L_2$ and $w \in L_4$. For any $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ where $t_i - t_{i-1} \in \Lambda_i$ ($t_0 = 0$), $w_t \in L_1$ and $w_t \in L_3$, so $w_t \in L_1 \cap L_3$. □

4 Decidability

Time plays an important role in real-time automata, since all transitions take some time to execute no matter whether their labels are observable. When unobservable labels are deleted from a trace, their elapsed time cannot vanish. In an observed timed word $v_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$, each a_i-transition takes some time less or equal to $t_i - t_{i-1}$ ($t_0 = 0$) due to possible unobservable transitions. For example, in Fig.1, if $P_{\{b\},t}(w_t) = (b, 4.5)$, there must be an a occurring before b in w_t, that is, $w_t = (a, t_a)(b, 4.5)$ with $1.5 \leq t_a \leq 2$.

If there exists a real-time automaton accepting the observable language generated by the original one, then we can translate this real-time automaton into its trace-equivalent finite-state automata and then into an equivalent deterministic automaton. Thus, two DFAs can be constructed $\mathcal{A}_{dfa}^{s}$ and $\mathcal{A}_{dfa}^{ns}$, which accept languages that are trace-equivalent to L_s and L_{ns} respectively as defined in Sect. 2.3.

We will describe the constructions in details in the following, with $\mathcal{A} = (S, \Sigma, \Delta, \mathit{Init}, F, \mu)$ being the original real-time automaton, $S_{secret} \subseteq S$ being the secret, and $\Sigma_o \subseteq \Sigma$ being the observable alphabet. τ is used to denote all the unobservable events in the set $\Sigma \setminus \Sigma_o$.

4.1 Calculating time between observable events

Unobservable transitions are similar to ε-transitions of non-deterministic automata. The only difference is that unobservable transitions still take some time.

Consider a run of $\mathcal{A}$: $s_0 \xrightarrow[\lambda_1]{\tau} s_1 \xrightarrow[\lambda_2]{\tau} \cdots \xrightarrow[\lambda_{n-1}]{\tau} s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$, which consists of $n-1$ unobservable transitions and one observable transition in sequence. Then a_n occurs at $\sum_{i=1}^{n} \lambda_i$. Similarly, if we consider a segment of a run: $s_0' \cdots \xrightarrow[\lambda_0]{a_0}$

$s_0 \xrightarrow[\lambda_1]{\tau} s_1 \xrightarrow[\lambda_2]{\tau} \cdots \xrightarrow[\lambda_{n-1}]{\tau} s_{n-1} \xrightarrow[\lambda_n]{a_n} s_n$, the time difference between a_n and a_0 is also $\sum_{i=1}^{n} \lambda_i$. It includes two parts: the sum of time taken by unobservable ones, and the time taken by the final observable one.

Based on the analysis above, there are two things to be done for each pair of states (s, s'), where s is an initial state or the post-state of an observable transition, and s' is the pre-state of an observable one. The first is to calculate how much time it can probably take to transit from s to s' via unobservable transitions, and the result is denoted by $\Lambda_{uo}(s, s')$, which is a subset of $\mathbb{R}_{\geq 0}$. The second is to sum up $\Lambda_{uo}(s, s')$ with the time taken by each observable transition from s', say (s', a, s''), thus we can obtain new transitions of the form (s, a, s'') whose corresponding time is the sum of $\Lambda_{uo}(s, s')$ and $\mu(s', a, s'')$. And therefore we can build a new real-time automaton whose alphabet is Σ_o alone.

In order to calculate $\Lambda_{uo}(s, s')$ for each pair (s, s'), we construct the *timing automaton* $\mathcal{A}_t$, a finite-state automaton with the alphabet $\bigcup_{\tau \in \Sigma \setminus \Sigma_o} \mu_\tau$, where $\mu_\tau = \{\mu(s_1, \tau, s_2) \mid (s_1, \tau, s_2) \in \Delta\}$. Only unobservable transitions of $\mathcal{A}$ and time taken by them are considered in $\mathcal{A}_t$. Each event in the alphabet of $\mathcal{A}_t$ is actually a non-empty subset of $\mathbb{R}_{\geq 0}$. Formally,

Definition 8. *The timing automaton of a real-time automaton $\mathcal{A}$ is a finite-state automaton $\mathcal{A}_t = (S_t, \Sigma_t, \delta_t, Init_t, F_t)$, where*

- *$S_t = \mathcal{A}.S$;*
- *Σ_t is $\bigcup_{\tau \in \Sigma \setminus \Sigma_o} \mu_\tau = \bigcup_{\tau \in \Sigma \setminus \Sigma_o} \{\mu(s_1, \tau, s_2) \mid (s_1, \tau, s_2) \in \Delta\}$;*
- *$\delta_t(s_1, \Lambda) = \{s_2 \mid \exists \tau \in \Sigma_{uo}\big((s_1, \tau, s_2) \in \mathcal{A}.\Delta \wedge \mathcal{A}.\mu(s_1, \tau, s_2) = \Lambda\big)\}$;*
- *$Init_t = \mathcal{A}.Init \cup \bigcup_{a \in \Sigma_o} Post_a$, and*
- *$F_t = \bigcup_{a \in \Sigma_o} Pre_a$.*

Based on $\mathcal{A}_t$, we can calculate $\Lambda_{uo}(s, s')$ using regular expressions by following the proof methods for Kleene's theorem.

Suppose the set of states is $\{s_i\}_{i \in \{1,\ldots,n\}}$. Let $R(s_{i_1}, s_{i_2}, k)$ be the regular expression denoting all the traces of runs from s_{i_1} to s_{i_2} where there are no states in between for $k = 0$, and no states with subscripts larger than k in between, for $1 \leq k \leq n$.

For any (s_{i_1}, s_{i_2}), the set $\{\Lambda \mid s_{i_2} \in \delta_t(s_{i_1}, \Lambda)\}$ is the events of all transitions from s_{i_1} to s_{i_2}. Let $R_0(s_{i_1}, s_{i_2})$ be the regular expression which is the sum of all events in $\{\Lambda \mid s_{i_2} \in \delta_t(s_{i_1}, \Lambda)\}$. If there is no such Λ, $R_0(s_{i_1}, s_{i_2})$ is set to $\emptyset$.

Then $R(s_{i_1}, s_{i_2}, k)$ is computed inductively: $R(s_{i_1}, s_{i_1}, 0) = \varepsilon + R_0(s_{i_1}, s_{i_1})$, and $R(s_{i_1}, s_{i_2}, 0) = R_0(s_{i_1}, s_{i_2})$ if $i_1 \neq i_2$. And $R(s_{i_1}, s_{i_2}, k+1) = R(s_{i_1}, s_{i_2}, k) + R(s_{i_1}, s_{k+1}, k) \cdot R(s_{k+1}, s_{k+1}, k)^* \cdot R(s_{k+1}, s_{i_2}, k)$.

So we can finally obtain $R(s_{i_1}, s_{i_2}, n)$ for each pair of states $(s_{i_1}, s_{i_2}) \in Init_t \times F_t$, which denotes the traces of runs from s_{i_1} to s_{i_2}.

After regular expressions have been obtained, the next step is to translate them into subsets of $\mathbb{R}_{\geq 0}$. Here $\emptyset$ means an empty set, ε means the set $\{0\}$, and Λ means the set Λ. And if r, r_1, r_2 are regular expressions and Λ, Λ_1, Λ_2 are their corresponding sets, we can translate $r_1 \cdot r_2$ into $\Lambda_1 + \Lambda_2 := \{\lambda_1 + \lambda_2 \mid \lambda_1 \in \Lambda_1, \lambda_2 \in \Lambda_2\}$, $r_1 + r_2$ 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 $(k+1)\Lambda = k\Lambda + \Lambda$ for $k \geq 0$.

Following these steps, we can obtain $\Lambda_{uo}(s_{i_1}, s_{i_2})$ from $R(s_{i_1}, s_{i_2}, n)$ for each pair of states $(s_{i_1}, s_{i_2}) \in \mathit{Init}_t \times F_t$.

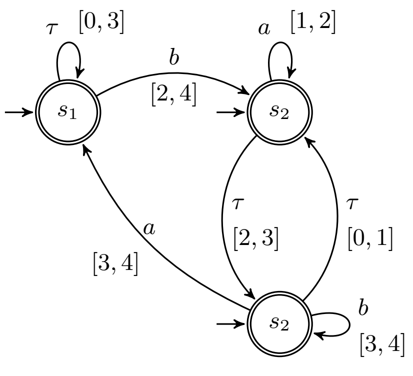


Fig. 3: Example: $\mathcal{A}$, the original real-time automaton under study

Example 4. In the real-time automaton $\mathcal{A}$ in Fig.3, Σ is divided into two sets, the observable $\{a, b\}$ and the unobservable $\{\tau\}$. The states s_1, s_2, s_3 are all initial and accepting. Transitions are arrows from one state to another with a label a, b, or τ, and timing information of each transition is the interval written near its label.

Its timing automaton, denoted as $\mathcal{A}_t$, is depicted in Fig.4.

$R(s_{i_1}, s_{i_2}, k)$ not equal to $\emptyset$ are listed below:

$k = 0$: $R(s_1, s_1, 0) = \varepsilon + [0, 3]$, $R(s_2, s_2, 0) = \varepsilon$, $R(s_2, s_3, 0) = [2, 3]$, $R(s_3, s_2, 0) = [0, 1]$, $R(s_3, s_3, 0) = \varepsilon$;

$k = 1$: $R(s_1, s_1, 1) = (\varepsilon + [0, 3]) + (\varepsilon + [0, 3])(\varepsilon + [0, 3])^*(\varepsilon + [0, 3]) = [0, 3]^*$, $R(s_2, s_2, 1) = \varepsilon$, $R(s_2, s_3, 1) = [2, 3]$, $R(s_3, s_2, 1) = [0, 1]$, $R(s_3, s_3, 1) = \varepsilon$;

$k = 2$: $R(s_1, s_1, 2) = [0, 3]^*$, $R(s_2, s_2, 2) = \varepsilon$, $R(s_2, s_3, 2) = [2, 3] + \varepsilon\varepsilon^*[2, 3] = [2, 3]$, $R(s_3, s_2, 2) = [0, 1] + [0, 1]\varepsilon^*\varepsilon = [0, 1]$, $R(s_3, s_3, 2) = \varepsilon + [0, 1]\varepsilon^*[2, 3) = \varepsilon + [0, 1] \cdot [2, 3]$;

$k = 3$: $R(s_1, s_1, 3) = [0, 3]^*$, $R(s_2, s_2, 3) = \varepsilon + [2, 3](\varepsilon + [0, 1] \cdot [2, 3])^*[0, 1] = ([2, 3] \cdot [0, 1])^*$, $R(s_2, s_3, 3) = [2, 3] + [2, 3](\varepsilon + [0, 1] \cdot [2, 3])^*(\varepsilon + [0, 1] \cdot [2, 3]) = [2, 3]([0, 1] \cdot [2, 3]))^*$, $R(s_3, s_2, 3) = [0, 1] + (\varepsilon + [0, 1] \cdot [2, 3])(\varepsilon + [0, 1] \cdot [2, 3])^*[0, 1] = [0, 1]([2, 3] \cdot [0, 1])^*$, $R(s_3, s_3, 3) = (\varepsilon + [0, 1] \cdot [2, 3]) + (\varepsilon + [0, 1] \cdot [2, 3])(\varepsilon + [0, 1] \cdot [2, 3])^*(\varepsilon + [0, 1] \cdot [2, 3]) = ([0, 1] \cdot [2, 3])^*$.

Finally, $\Lambda_{uo}(s_{i_1}, s_{i_2})$ can be obtained from $R(s_{i_1}, s_{i_2}, 3)$: $\Lambda_{uo}(s_1, s_1) = [0, +\infty)$, $\Lambda_{uo}(s_2, s_2) = \Lambda_{uo}(s_3, s_3) = \{0\} \cup [2, +\infty)$, $\Lambda_{uo}(s_3, s_2) = [0, 1] \cup [2, +\infty)$, $\Lambda_{uo}(s_2, s_3) = [2, +\infty)$, and $\Lambda_{uo}(s_1, s_2) = \Lambda_{uo}(s_1, s_3) = \Lambda_{uo}(s_2, s_1) = \Lambda_{uo}(s_3, s_1) = \emptyset$. □

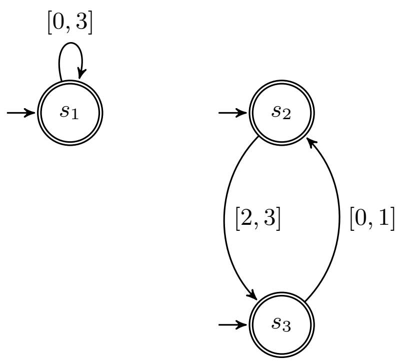


Fig. 4: Example: $\mathcal{A}_t$

4.2 Constructing real-time automata $\mathcal{A}_{obs}$, $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$

After that, we can define a real-time automaton $\mathcal{A}_{obs}$. Its alphabet is Σ_o, and each of its transition starts from an initial state or a post-state of an observable transition and ends in a post-state of an observable transition of $\mathcal{A}$. Formally,

Definition 9. *$\mathcal{A}_{obs} = (S_{obs}, \Sigma_{obs}, \Delta_{obs}, Init_{obs}, F_{obs}, \mu_{obs})$ can be constructed as follows:*

- $S_{obs} = \mathcal{A}.Init \cup \bigcup_{a\in\Sigma_o} Post_a$,
- $\Sigma_{obs} = \Sigma_o$,
- $\Delta_{obs} = \{(s_1, a, s_2) \mid \exists s_3 \in \mathcal{A}.S\big(\Lambda_{uo}(s_1, s_3) \neq \emptyset \wedge (s_3, a, s_2) \in \mathcal{A}.\Delta\big)\}$,
- $Init_{obs} = \mathcal{A}.Init$,
- $F_{obs} = \bigcup_{a\in\Sigma_o} Post_a$, *and*
- $\mu_{obs}(s_1, a, s_2) = \bigcup\{\Lambda_{uo}(s_1, s_3) + \mathcal{A}.\mu(s_3, a, s_2) \mid \exists s_3 \in \mathcal{A}.S\big(\Lambda_{uo}(s_1, s_3) \neq \emptyset \wedge (s_3, a, s_2) \in \mathcal{A}.\Delta\big)\}$.

Example 5 (Ctd.). Now we build $\mathcal{A}_{obs} = (S_{obs}, \Sigma_{obs}, \Delta_{obs}, Init_{obs}, F_{obs}, \mu_{obs})$ shown in Fig.5. $S_{obs} = Init_{obs} = F_{obs} = \{s_1, s_2, s_3\}$. $\Sigma_{obs} = \{a, b\}$. Transitions and their corresponding time labels are listed below:
(s_1, b, s_2): $\mu(s_1, b, s_2) = [2, +\infty)$;
(s_2, a, s_2): $\mu(s_2, a, s_2) = [1, 2] \cup [3, +\infty)$;
(s_3, a, s_2): $\mu(s_3, a, s_2) = [1, +\infty)$;
(s_2, b, s_3): $\mu(s_2, b, s_3) = [5, +\infty]$;
(s_3, b, s_3): $\mu(s_3, b, s_3) = [3, 4] \cup [5, +\infty)$;
(s_2, a, s_1): $\mu(s_2, a, s_1) = [5, +\infty)$;
(s_3, a, s_1): $\mu(s_3, a, s_1) = [3, 4] \cup [5, +\infty)$.
The time information is neglected in the figure for the sake of simplicity. □

Lemma 4. *Given w_t in $Tr(s_0)$ of $\mathcal{A}$, $P_{\Sigma,t}(w_t)$ is in $Tr(s_0)$ of $\mathcal{A}_{obs}$ if $P_{\Sigma,t}(w_t) \neq \varepsilon_t$.*

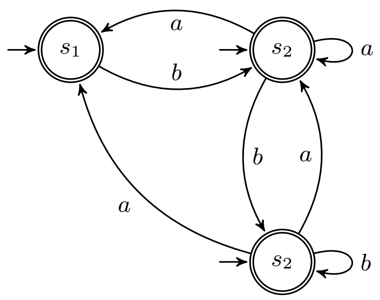


Fig. 5: Example: $\mathcal{A}_{obs}$

Proof. $P_{\Sigma,t}(w_t) \neq \varepsilon_t$ means that there is at least one observable label in w_t. Suppose observable labels occurring in w_t are a_1, a_2, ..., and a_n at t_1, t_2, ..., and t_n respectively, then $P_{\Sigma,t}(w_t) = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$. Let ρ be a run of $\mathcal{A}$ from s_0 whose trace is w_t. There must be transitions (s_1, a_1, s'_1), (s_2, a_2, s'_2), ..., and (s_n, a_n, s'_n) in ρ, and s_{i+1} is reachable from s'_i for $0 \leq i < n$. (Let $s'_0 = s_0$ and $t_0 = 0$ here.) Each $t_k - t_{k-1}$ is the sum of two parts: one is time taken by unobservable transition(s) from s'_{k-1} to s_k, and the other is time taken by transition (s_k, a_k, s'_k). In other words, there exists some $\lambda_{uo,k} \in \Lambda_{uo}(s'_{k-1}, s_k)$ and $\lambda_{o,k} \in \mathcal{A}.\mu(s_k, a_k, s'_k)$ such that $t_k - t_{k-1} = \lambda_{uo,k} + \lambda_{o,k}$. Based on the constructing steps of $\mathcal{A}_{obs}$, there exists transitions (s'_{k-1}, a_k, s'_k) in $\mathcal{A}_{obs}$ whose time labels include $t_k - t_{k-1}$ respectively, so that there exists a run $\rho_{obs} = s_0 \xrightarrow[t_1]{a_1} s'_1 \xrightarrow[t_2 - t_1]{a_2} s'_2 \cdots s'_{n-1} \xrightarrow[t_n - t_{n-1}]{a_n} s'_n$, whose trace is $P_{\Sigma,t}(w_t)$. □

Lemma 5. *Given $v_t \neq \varepsilon_t$ in $Tr(s_0)$ of $\mathcal{A}_{obs}$, there exists some w_t in $Tr(s_0)$ of $\mathcal{A}$ such that $P_{\Sigma,t}(w_t) = v_t$.*

Proof. Suppose there exists a run of $\mathcal{A}_{obs}$, say $\rho_{obs} = s_0 \xrightarrow[t_1]{a_1} s'_1 \xrightarrow[t_2 - t_1]{a_2} s'_2 \cdots s'_{n-1} \xrightarrow[t_n - t_{n-1}]{a_n} s'_n$, whose trace is $v_t = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$. There must be transitions (s_1, a_1, s'_1), (s_2, a_2, s'_2), ..., and (s_n, a_n, s'_n) in $\mathcal{A}$, and s_{i+1} is reachable from s'_i via only unobservable transitions for $0 \leq i < n$ ($s'_0 = s_0$ here). Each $t_k - t_{k-1}$ is the sum of some $\lambda_{uo,k} \in \Lambda_{uo}(s'_{k-1}, s_k)$ and $\lambda_{o,k} \in \mathcal{A}.\mu(s_k, a_k, s'_k)$ ($t_0 = 0$ here). Hence, there is a run of $\mathcal{A}$, $\rho = s_0 \cdots s_1 \xrightarrow[\lambda_{o,1}]{a_1} s'_1 \cdots s_2 \xrightarrow[\lambda_{o,2}]{a_2} s'_2 \cdots s_n \xrightarrow[\lambda_{o,n}]{a_n} s'_n$ with $P_{\Sigma_o,t}(trace(\rho)) = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n) = v_t$. □

Based on the above discussion, two real-time automata, $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$, can be constructed according to the given S_{secret} as follows:

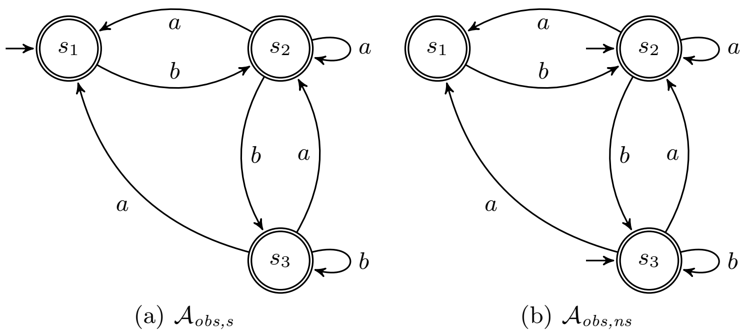
- If $(\mathcal{A}_{obs}.Init \cap \mathcal{A}_{obs}.F) \cap S_{secret}$ is not empty, let $\mathcal{A}_{obs,s}$ be the same as $\mathcal{A}_{obs}$ except that its initial states are $\mathcal{A}_{obs}.Init \cap S_{secret}$. Otherwise, we introduce a new state s_ε, having no transition starts from or ends in it, to ensure ε_t is also accepted. Here $\mathcal{A}_{obs,s}$ is the same as $\mathcal{A}_{obs}$ except that its initial states are $\{s_\varepsilon\} \cup \mathcal{A}_{obs}.Init \cap S_{secret}$, and that its accepting states are $\{s_\varepsilon\} \cup \mathcal{A}_{obs}.F$.

– If $(\mathcal{A}_{obs}.Init \cap \mathcal{A}_{obs}.F) \setminus S_{secret}$ is not empty, let $\mathcal{A}_{obs,ns}$ be the same as $\mathcal{A}_{obs}$ except that its initial states are $\mathcal{A}_{obs}.Init \setminus S_{secret}$. Otherwise, we introduce s_ε, then $\mathcal{A}_{obs,s}$ is the same as $\mathcal{A}_{obs}$ except that its initial and accepting states are $\{s_\varepsilon\} \cup \mathcal{A}_{obs}.Init \setminus S_{secret}$ and $\{s_\varepsilon\} \cup \mathcal{A}_{obs}.F$ respectively.

Theorem 2. *$\mathcal{A}_{obs,s}$ accepts language $L_s = P_{\Sigma_o,t}(Tr(\mathcal{A}.Init \cap S_{secret})))$, and $\mathcal{A}_{obs,ns}$ accepts language $L_{ns} = P_{\Sigma_o,t}(Tr(\mathcal{A}.Init \setminus S_{secret})))$.*

Proof. This is straightforward from Lemma 4 and Lemma 5 and the constructions of $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$. □

Example 6 (Ctd.). Let $S_{secret} = \{s_1\}$. Then $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$ are depicted in Fig.6. Their time information is also neglected in this figure, the same as the previous example's. □



(a) $\mathcal{A}_{obs,s}$ (b) $\mathcal{A}_{obs,ns}$

Fig. 6: Example: $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$

4.3 Building trace-equivalent NFAs

Since $\mathcal{A}_{obs,s}$ and $\mathcal{A}_{obs,ns}$ are built, they can be transformed into their corresponding NFAs $\mathcal{A}_{nfa,s}$ and $\mathcal{A}_{nfa,ns}$, which are trace-equivalent to them respectively. Thus can be built DFAs $\mathcal{A}_{dfa,s}$ and $\mathcal{A}_{dfa,ns}$ further, which are equivalent to $\mathcal{A}_{nfa,s}$ and $\mathcal{A}_{nfa,ns}$ respectively. By exploiting complement and product operations over DFAs, $\mathcal{A}^{p}_{dfa} = \mathcal{A}^{comp}_{dfa,ns} \times \mathcal{A}_{dfa,s}$ can be obtained. $\mathcal{A}^{p}_{dfa}$ accepts the language trace-equivalent to the intersection of $L_s = L_f(\mathcal{A}_{obs,s})$ and complement of $L_{ns} = L_f(\mathcal{A}_{obs,ns})$, based on Sect.3.

Theorem 3. *The initial-state opacity problem of real-time automata is decidable.*

Proof. Given a real-time automaton $\mathcal{A}$, Σ_o and S_{secret}, an automaton $\mathcal{A}^p_{dfa}$ can be constructed by following the steps discussed above. $\mathcal{A}^p_{dfa}$ accepts the language $L_f(\mathcal{A}_{dfa,ns})^c \cap L_f(\mathcal{A}_{dfa,s})$, which is trace-equivalent to the timed language $L^c_{ns} \cap L_s$.

The problem is to check whether $\mathcal{A}^p_{dfa}$ accepts any word $w \neq \varepsilon$. A word $w = (a_1, \Lambda_1) \dots (a_n, \Lambda_n)$ being accepted means that any timed word $w_t = (a_1, t_1) \dots (a_n, t_n)$ with $t_i - t_{i-1} \in \Lambda_i$ $(t_0 = 0)$ is in the set $L^c_{ns} \cap L_s$, that is, $\mathcal{A}$ is not initial-state opaque w.r.t S_{secret} and Σ_o. Otherwise, $\mathcal{A}$ is opaque w.r.t S_{secret} and Σ_o. $\square$

5 Conclusion

In this paper, we investigated the initial-state opacity problem of real-time automata. The original real-time automaton is first translated into a new one whose alphabet is the observable alphabet, and then into two real-time automata accepting the projection of secret and non-secret languages respectively. We introduce a relation between timed words over Σ and untimed words over $\Sigma \times (2^{\mathbb{R}_{\geq 0}} \setminus \{\emptyset\})$ called trace-equivalence, and transform real-time automata into finite-state automata. We also introduce the notion of partitioned languages, to guarantee the closure under complementation and product. Therefore results of finite-state automata can be applied. Finally, we come up with the conclusion that the initial-state opacity problem of real-time automata is decidable.

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. As an on-going and future work, it deserves to investigate the language opacity problem of RTA, which will be reported in another paper. In addition, it is quite interesting how to apply RTA to model security properties of communication protocols with time in the real-world.

References

1. Saboori, A., Hadjicostis, C.N.: Verification of initial-state opacity in security applications of discrete event systems. Information Sciences **246** (2013) 115 – 132
2. Bryans, J.W., Kounty, M., Ryan, P.Y.: Modelling opacity using petri nets. Electronic Notes in Theoretical Computer Science **121** (2005) 101 – 115 Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).
3. Tong, Y., Li, Z., Seatzu, C., Giua, A.: Verification of state-based opacity using Petri nets. IEEE Transactions on Automatic Control **62**(6) (2017) 2823–2837
4. Bryans, J.W., Kounty, M., Mazaré, L., Ryan, P.Y.A.: Opacity generalised to transition systems. International Journal of Information Security **7**(6) (Nov 2008) 421–435
5. Saboori, A., Hadjicostis, C.N.: Verification of k-step opacity and analysis of its complexity. In: Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference. (2009) 205–210

6. Saboori, A., Hadjicostis, C.N.: Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Transactions on Automatic Control **57**(5) (2012) 1155–1165
7. Saboori, A., Hadjicostis, C.N.: Verification of infinite-step opacity and complexity considerations. IEEE Transactions on Automatic Control **57**(5) (2012) 1265–1269
8. Keroglou, C., Hadjicostis, C.N.: Initial state opacity in stochastic des. 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA) (2013) 1–8
9. Bérard, B., Chatterjee, K., Sznajder, N.: Probabilistic opacity for Markov decision processes. Information Processing Letters **115**(1) (2015) 52 – 59
10. Bérard, B., Mullins, J., Sassolas, M.: Quantifying opacity. 2010 Seventh International Conference on the Quantitative Evaluation of Systems (2010) 263–272
11. Ibrahim, M., Chen, J., Kumar, R.: Secrecy in stochastic discrete event systems. In: Proceedings of the 11th IEEE International Conference on Networking, Sensing and Control. (2014) 48–53
12. Cassez, F.: The dark side of timed opacity. In Park, J.H., Chen, H.H., Atiquzzaman, M., Lee, C., Kim, T.h., Yeo, S.S., eds.: Advances in Information Security and Assurance. (2009)
13. Dima, C.: Real-time automata. Journal of Automata, Languages and Combinatorics **6**(1) (2001) 3–24