Enforcement and Validation (at runtime) of Various Notions of Opacity

Yliès Falcone, Hervé Marchand

Discrete Event Dynamic Systems manuscript No.
(will be inserted by the editor)

Enforcement and Validation (at runtime) of Various Notions of Opacity

Yliès Falcone · Hervé Marchand

Received: date / Accepted: date

Abstract We are interested in the validation of *opacity*. Opacity models the impossibility for an attacker to retrieve the value of a *secret* in a system of interest. Roughly speaking, ensuring opacity provides confidentiality of a secret on the system that must not leak to an attacker. More specifically, we study how we can *model-check*, *verify and enforce* at system *runtime*, several levels of opacity. Besides existing notions of opacity, we also introduce K-step strong opacity, a more practical notion of opacity that provides a stronger level of confidentiality.

Keywords opacity · K-step opacity · runtime verification · runtime enforcement

1 Introduction

Security is a major concern in nowadays information systems. While infrastructures are becoming more complex, it becomes harder to ensure desired security properties. Usually three dimensions in security are distinguished: *confidentiality, integrity, availability*. Confidentiality is concerned with ensuring that the information in a system is accessible only to appropriate users.

Y. Falcone
UJF-Grenoble 1, Grenoble, France
Laboratoire d'Informatique de Grenoble (LIG), UMR 5217, F-38041, Grenoble, France
Tel.: +33 4 76 82 72 14
Fax: +33 4 76 51 49 85
E-mail: Ylies.Falcone@ujf-grenoble.fr

H. Marchand
Inria, Rennes - Bretagne Atlantique, Rennes, France
Tel.: +33 2 99 84 75 09
Fax: +33 2 99 84 71 71
E-mail: Herve.Marchand@inria.fr

Integrity aims to avoid alteration of data (e.g., during their treatment, transmission,...). Availability aims to maintain system resources operational when they are needed. Among existing security notions, *opacity* (see e.g., Badouel et al (2007); Bryans et al (2008)) is a generic and general notion used to express several existing confidentiality concerns such as trace-based non-interference and anonymity (see Bryans et al (2008)) and even secrecy (see Alur and Zdancewic (2006)). Opacity aims at preserving unwanted retrievals of a system secret (e.g., values of confidential variables) by untrusted users while observing the system. Roughly speaking, when examining the opacity of a secret on a given system, we check whether there are some executions of the system which can lead an external attacker to know the secret; in that case the secret is said to be *leaking*. While usual opacity is concerned by the *current* disclosure of a secret, K-opacity, introduced in Saboori and Hadjicostis (2007), additionally models secret retrieval in the past (e.g., K execution steps before).

Ensuring opacity and some of its variant on a system is usually performed using *supervisory control* (see Cassandras and Lafortune (2006) for an introduction), as for example in Dubreil et al (2010), Takai and Oka (2008), Takai and Kumar (2009) and Saboori and Hadjicostis (2012). In an abstract way, supervisory control consists in using a so-called controller to disable undesired behaviors of the system, e.g., those leading to reveal the secret. The technique of *dynamic observability* was proposed in Cassez et al (2009) to ensure the opacity of secrets. This technique consists in dynamically restraining, at each execution step of the system, the set of observable events to ensure opacity. Finally, Wu and Lafortune (2012) enforces opacity by statically inserting additional events in the output behavior of the system while preserving its observable behavior.

Motivations - the limitations of existing validation methods for opacity. Whilst there exist established techniques to ensure opacity, these techniques suffer from practical limitations preventing their applicability in some situations. First, supervisory control is an *intrusive technique*. Indeed, the internal behavior of the underlying system has to be modifiable before implementing a supervisory control approach. Second, to ensure opacity, supervisory control entails to *disable some internal behaviors* of the underlying system. As a consequence, opacity preservation comes often at the price of not achieving intended service fulfillment. Moreover, when a controller disables an action in the system, it also disables all its subsequent behaviors. That is to say, supervisory control does not take into account the situations where opacity leakage might be temporary: every opacity leakage is definitive. Third, ensuring opacity via dynamic observability comes at the price of *destroying observable behavior* of the system.

Those limitations motivate for investigating the use of other validation techniques, not suffering from the aforementioned limitations, in order to ensure opacity.

Runtime validation techniques. In this paper we are interested in runtime validation techniques, namely *runtime verification* and *runtime enforcement*, so as to validate several levels of opacity on a system. Runtime verification (see Pnueli and Zaks (2006); Havelund and Goldberg (2008); Falcone et al (2009b)) consists in checking during the execution of a system whether a desired property holds or not. Generally, one uses a special decision procedure, a *monitor*, grabbing relevant information in the run of an executing system and acting as an oracle to decide property validation or violation. Runtime enforcement (see Schneider (2000); Hamlen et al (2006); Falcone et al (2009a)) is an extension of runtime verification aiming to circumvent property violations. Within this technique the monitor not only observes the current program execution, but also modifies it. It uses an internal memorization mechanism in order to ensure that the expected property is fulfilled: it still reads an input sequence but now produces a new sequence of events so that the property is enforced.

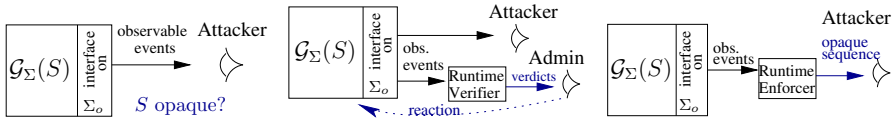
Paper organization. The remainder of this paper is organized as follows. Section 2 presents at an abstract level the three analysis that we propose to validate opacity. In Section 3, we introduce some preliminaries and notations. Then, Section 4 presents the various notions of opacity we consider. In Section 5 and Section 6, we verify and enforce opacity at runtime, respectively. Section 7 is dedicated to the description of related work. Finally, Section 8 gives some concluding remarks and opened perspectives. To facilitate the reading of this article, some technical proofs are given in Appendix A.

This article extends an article that appeared in the 52nd Conference on Decision and Control (see Falcone and Marchand (2013)). More specifically, this article brings the following additional contributions:

- to introduce the notion of K-step strong opacity that provides a stronger level of confidentiality;
- to introduce the notion of K-delay trajectory estimator that extends the notion of K-delay state estimator introduced in Saboori and Hadjicostis (2007) for the verification of what we will refer to as K-step weak opacity;
- to propose runtime verification and enforcement for the verification of K-step strong opacity (in addition to the usual notion of opacity of Badouel et al (2007) and K-step weak opacity of Saboori and Hadjicostis (2007));
- to prove the correctness of the runtime verification and enforcement monitors that we synthesize;
- to propose a more complete presentation of the concepts that appeared in the conference version.

2 The proposed approach

Context. The considered problem can be depicted in Fig. 1a. A system $\mathcal{G}$ produces sequences of events belonging to an alphabet Σ. Among the possible executions of the system, some of these are said to be *secret*. Some events of



(a) System vs Opacity (b) Opacity Verification (c) Opacity Enforcement

Fig. 1 Several ways to validate the opacity of a secret on a system

the system, in a sub-alphabet $\Sigma_o \subseteq \Sigma$, are observable by an external attacker through the system interface. We assume that the attacker does not interfere with the system (i.e., he performs only observations through the interface) and has a perfect knowledge of the structure (even internal) of the system. We are interested in the opacity of the secret executions on the considered system. That is, from a sequence of observable events, the attacker should not be able to deduce whether the current execution of the system (corresponding to this observation) is secret or not. In this case, the secret S is said to be *opaque*[1] w.r.t. the considered system and its interface.

Three techniques to validate opacity. We sketch the techniques that we propose to analyze and validate the opacity of a system. When model-checking (Fig. 1a) the opacity of a secret on the considered system, we take its specification to perform an analysis that provides the executions leading to a security leakage when they are observed through the interface of the system. This indicates existing flaws to the system designer so as to correct them. When verifying opacity at runtime (Fig. 1b), we introduce a runtime verifier which observes the same sequence of observable events as the attacker. The runtime verifier is a special kind of monitor that is in charge of producing verdicts related to the preservation or violation of the considered notion of opacity. With such a mechanism, the system administrator may react and (manually) take appropriate measures. When enforcing opacity at runtime (Fig. 1c), we introduce a runtime enforcer between the system and the attacker. The sequence of observable events from the system is directly fed to the enforcer. The new system now consists in the initial system augmented with the runtime enforcer. The attacker now observes the outputs produced by the runtime enforcer. The runtime enforcer modifies its input sequence and produces a new one in such a way that, on the output execution sequence seen by the attacker, the desired notion of opacity is preserved w.r.t. the actual execution on the initial system. Thus, the runtime enforcement approach proposed in this article automatically prevents opacity violation.

Originality. Opacity cannot be expressed as a property that can be decided on a single execution of the system in isolation. It obviously depends on the

[1] Several notions of opacity will be considered in this paper. Here we present only the simplest one informally.

current execution but also on the alternative, observationally equivalent, behaviors of the system. Runtime techniques (see Havelund and Goldberg (2008) for an overview) are thus not directly applicable in order to validate the opacity of systems. Then, one originality of this paper is to consider opacity preservation by the observable sequences of the system. It will allow us to apply runtime techniques operating on the observable behavior of a system. Moreover, to the best of our knowledge, the only runtime technique dedicated to opacity was the one proposed in Dubreil et al (2009). The authors proposed a runtime verification framework for the simplest variant of opacity proposed in this paper. Thus, this paper addresses the runtime verification for more evolved notions of opacity. Moreover, no runtime enforcement approach was proposed to validate any notion of opacity. Note also that, in some sense, compared to traditional runtime verification and enforcement approaches, this paper also studies how more evolved kinds of properties, such as opacity, can be validated using standard runtime-based techniques.

Advantages. Compared to previously depicted validation methods for opacity, the runtime validation techniques proposed in this article have several advantages. First, these techniques are *not intrusive*. Indeed, a runtime verification or runtime enforcement framework does not suppose being able to modify the internal behavior of the monitored system. It might be particularly useful when dealing with legacy code, leading model-checking to become obsolete since the internal system behavior cannot be modified. Moreover, the proposed runtime-based approaches do not *distort* the internal nor the observable behavior of the system. Furthermore, checking the opacity at runtime opens the way to react to misbehaviors; as it is shown with runtime enforcement in this paper. Ensuring opacity with runtime enforcement also has the advantage to modify only the observable behavior of the system *in a minimal way*: runtime enforcers, as defined in this article, delay the initial execution sequence at a minimum. This affords better confidence to the system provider.

3 Preliminaries and notation

Unless otherwise specified, considered functions are total. $\mathbb{N}$ denotes the set of non-negative integers. $\mathbb{B}$ denotes the set of Boolean values (true, false). Considering a finite set of elements E, a *sequence* s over E is formally defined by a total function $s : I \to E$ where I is the integer interval $[0, n]$ for some $n \in \mathbb{N}$. A *language* over E is a set of sequences over E. We denote by E^* the universal language over E (i.e., the set of all finite sequences over E), by E^+ the set of non-empty finite sequences over E. Furthermore, for $n \in \mathbb{N} \setminus \{0, 1\}$, the generalized Cartesian product of E is $E^n \stackrel{\text{def}}{=} E \times E \times \cdots \times E$, i.e., the Cartesian product of E of dimension n. The empty sequence of E^* is denoted by ϵ_E or ϵ when clear from the context. The length of a finite sequence $s \in E^*$ is noted $|s|$. For a sequence $s \in E^+$ and $i < |s|$, the $(i+1)$-th element of s is denoted by s^i, and the subsequence containing the $i+1$ first elements of s is denoted $s^{\cdots i}$.

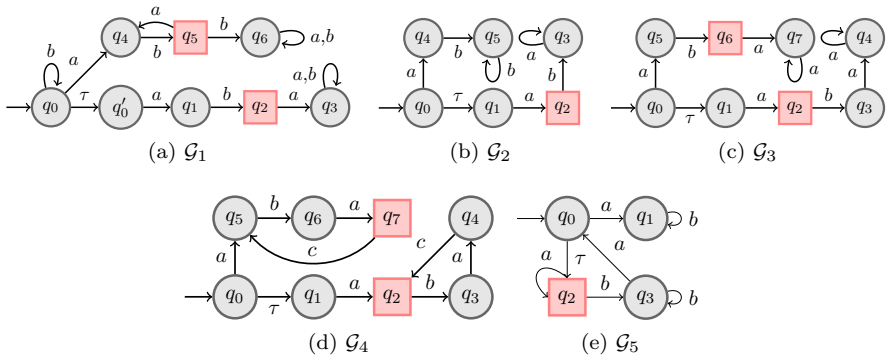
For $s, s' \in E^*$, we denote by $s \cdot s'$ the concatenation of s and s', and by $s \preceq s'$ the fact that s is a prefix of s'. The sequence $s \in E^*$ is said to be a prefix of $s' \in E^*$ when $\forall i \in [0, |s| - 1] : s^i = s'^i$ and $|s| \leq |s'|$. The *prefix-closure* of a language L wrt. E^* is defined as $\text{Pref}(L) \stackrel{\text{def}}{=} \{s \in E^* \mid \exists s' \in E^* : s \cdot s' \in L\}$. Given $s' \preceq s$, $|s - s'| \stackrel{\text{def}}{=} |s| - |s'|$.

We assume that the behaviors of systems are modeled by Labelled Transitions Systems (LTS for short) which actions belong to a finite set Σ. Sequences of actions are named *execution sequences*. The formal definition of an LTS is as follows:

Definition 1 (LTS) *A deterministic LTS is a 4-tuple $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ where $Q^{\mathcal{G}}$ is a finite set of states, $q^{\mathcal{G}}_{\text{init}} \in Q^{\mathcal{G}}$ is the initial state, Σ is the alphabet of* actions*, and $\delta_{\mathcal{G}} : Q^{\mathcal{G}} \times \Sigma \to Q^{\mathcal{G}}$ is* the partial transition function.

We consider a given LTS $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$. For $q \in Q^{\mathcal{G}}$, the new LTS $\mathcal{G}(q)$ is the LTS $\mathcal{G}$ initialised in q, i.e., $\mathcal{G}(q) \stackrel{\text{def}}{=} (Q^{\mathcal{G}}, q, \Sigma, \delta_{\mathcal{G}})$. We write $q \xrightarrow{a}_{\mathcal{G}} q'$ for $\delta_{\mathcal{G}}(q, a) = q'$ and $q \xrightarrow{a}_{\mathcal{G}}$ for $\exists q' \in Q^{\mathcal{G}} : q \xrightarrow{a}_{\mathcal{G}} q'$. We extend $\to_{\mathcal{G}}$ to arbitrary execution sequences by setting: $q \xrightarrow{\epsilon}_{\mathcal{G}} q$ for every state q, and $q \xrightarrow{s\sigma}_{\mathcal{G}} q'$ whenever $q \xrightarrow{s}_{\mathcal{G}} q''$ and $q'' \xrightarrow{\sigma}_{\mathcal{G}} q'$, for some $q'' \in Q^{\mathcal{G}}$. Given $\Sigma' \subseteq \Sigma$, $\mathcal{G}$ is said to be *Σ'-complete* whenever $\forall q \in Q^{\mathcal{G}}, \forall a \in \Sigma' : q \xrightarrow{a}_{\mathcal{G}}$. It is complete if it is Σ-complete. We set for any language $L \subseteq \Sigma^*$ and any set of states $X \subseteq Q^{\mathcal{G}}$, $\Delta_{\mathcal{G}}(X, L) \stackrel{\text{def}}{=} \{q \in Q^{\mathcal{G}} \mid \exists s \in L, \exists q' \in X : q' \xrightarrow{s}_{\mathcal{G}} q\}$. $\mathcal{L}(\mathcal{G}) \stackrel{\text{def}}{=} \{s \in \Sigma^* \mid q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}}\}$ denotes the set of execution sequences of the system $\mathcal{G}$. Given a set of marked states $F_{\mathcal{G}} \subseteq Q^{\mathcal{G}}$, the *marked (generated) language* of $\mathcal{G}$ is defined as $\mathcal{L}_{F_{\mathcal{G}}}(\mathcal{G}) \stackrel{\text{def}}{=} \{s \in \Sigma^* \mid \exists q \in F_{\mathcal{G}} : q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}} q\}$, i.e., the set of execution sequences that end in $F_{\mathcal{G}}$. Previous notations apply to deterministic finite-state machines (FSM) which are LTSs with an output function.

Observable Behavior. The observation interface between a user and the system is specified by a sub-alphabet of events $\Sigma_{\text{o}} \subseteq \Sigma$. The user observation through an interface is then defined by a *projection*, denoted by $P_{\Sigma_{\text{o}}}$, from Σ^* to Σ^*_{o} that erases in an execution sequence of Σ^* all events not in Σ_{o}. Formally, $P_{\Sigma_{\text{o}}}(\epsilon_\Sigma) \stackrel{\text{def}}{=} \epsilon_\Sigma$ and $P_{\Sigma_{\text{o}}}(s \cdot \sigma) \stackrel{\text{def}}{=} P_{\Sigma_{\text{o}}}(s) \cdot \sigma$ if $\sigma \in \Sigma_{\text{o}}$ and $P_{\Sigma_{\text{o}}}(s)$ otherwise. This definition extends to any language $L \subseteq \Sigma^*$: $P_{\Sigma_{\text{o}}}(L) \stackrel{\text{def}}{=} \{\mu \in \Sigma^*_{\text{o}} \mid \exists s \in L : \mu = P_{\Sigma_{\text{o}}}(s)\}$. In particular, given an LTS $\mathcal{G}$ over Σ and a set of observable actions $\Sigma_{\text{o}} \subseteq \Sigma$, the set of *observed traces* of $\mathcal{G}$ is $\mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G}) \stackrel{\text{def}}{=} P_{\Sigma_{\text{o}}}(\mathcal{L}(\mathcal{G}))$. Given two execution sequences $s, s' \in \Sigma^*$, they are equivalent w.r.t. $P_{\Sigma_{\text{o}}}$, noted $s \approx_{\Sigma_{\text{o}}} s'$ whenever $P_{\Sigma_{\text{o}}}(s) = P_{\Sigma_{\text{o}}}(s')$. Given two execution sequences s, s' such that $s' \preceq s$, $s \setminus s'$ is the suffix of s that permits to extend s' to s, and $|s - s'|_{\Sigma_{\text{o}}} \stackrel{\text{def}}{=} |P_{\Sigma_{\text{o}}}(s)| - |P_{\Sigma_{\text{o}}}(s')|$ corresponds to the number of observable events that are necessary to extend s' into s. Conversely, given $L \subseteq \Sigma^*_{\text{o}}$, the *inverse projection* of L is $P^{-1}_{\Sigma_{\text{o}}}(L) \stackrel{\text{def}}{=} \{s \in \Sigma^* \mid P_{\Sigma_{\text{o}}}(s) \in L\}$. Given $\mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$, $[\![\mu]\!]^{\mathcal{G}}_{\Sigma_{\text{o}}} \stackrel{\text{def}}{=} P^{-1}_{\Sigma_{\text{o}}}(\mu) \cap \mathcal{L}(\mathcal{G})$ (noted $[\![\mu]\!]_{\Sigma_{\text{o}}}$ when clear from context) is the set of observation traces of $\mathcal{G}$ *compatible* with μ, i.e., execution sequences of $\mathcal{G}$ having trace μ. Given $\mu' \preceq \mu$, we note $[\![\mu'/\mu]\!]_{\Sigma_{\text{o}}} \stackrel{\text{def}}{=} [\![\mu']\!]_{\Sigma_{\text{o}}} \cap \text{Pref}([\![\mu]\!]_{\Sigma_{\text{o}}})$ the set of



(a) $\mathcal{G}_1$ (b) $\mathcal{G}_2$ (c) $\mathcal{G}_3$

(d) $\mathcal{G}_4$ (e) $\mathcal{G}_5$

Fig. 2 Several systems with secret states (red squares)

traces of $\mathcal{G}$ that are still compatible with μ' knowing that μ' is the prefix of μ that occurred in the system.

4 Several notions of opacity

In this section, we formalize three different kinds of opacity. Each of them provides a different level of confidentiality for the secret. Opacity is defined on the observable and unobservable behaviors of the system. In the rest of the article, we let $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ be an LTS over Σ and $\Sigma_{\text{o}} \subseteq \Sigma$. We shall consider that the confidential information is directly encoded in the system by means of a set of states[2] $S \subseteq Q^{\mathcal{G}}$.

4.1 Simple opacity

If the current execution sequence of the system is $t \in \mathcal{L}(\mathcal{G})$, the attacker should not be able to deduce, from the knowledge of $P_{\Sigma_{\text{o}}}(t)$ and the structure of $\mathcal{G}$, that the current state of the system is in S. This is captured by the notion of opacity introduced by Bryans et al (2008) for transition systems:

Definition 2 (Simple opacity) *On $\mathcal{G}$, the secret S is opaque under the projection $P_{\Sigma_{\text{o}}}$ or $(\mathcal{G}, P_{\Sigma_{\text{o}}})$-opaque if*

$$\forall t \in \mathcal{L}_S(\mathcal{G}), \exists s \in \mathcal{L}(\mathcal{G}) : s \approx_{\Sigma_{\text{o}}} t \wedge s \notin \mathcal{L}_S(\mathcal{G}).$$

Example 1 *Consider the LTS $\mathcal{G}_1$ of Fig. 2a, with $\Sigma_{\text{o}} = \{a, b\}$. The secret is given by $S = \{q_2, q_5\}$. S is not $(\mathcal{G}_1, P_{\Sigma_{\text{o}}})$-opaque, as after the observation of a trace in $b^* \cdot a \cdot b$, the attacker knows that the system is currently in a secret state. Note that he does not know whether it is q_2 or q_5 but he knows that the state of the system is in S.*

[2] Equivalently, the secret can be given by a regular language over Σ^*, see Cassez et al (2009) for more details.

Another characterization of opacity was given by Dubreil (2009) in terms of observed traces: S is opaque w.r.t. $\mathcal{G}$ and P_{Σ_o} whenever $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : [\![\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. Hence, the set of traces for which the opacity of the secret S is leaking is defined by:

$$\text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0) \stackrel{\text{def}}{=} \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid [\![\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})\}.$$

Furthermore, S is $(\mathcal{G}, P_{\Sigma_o})$-opaque if and only if $\text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0) = \emptyset$.

4.2 K-step based opacity

Simple opacity requires that the secret should not be revealed only when the system is *currently* in a secret state. However, one can argue that confidentiality requirements may also prohibit inferring that the system went through a secret state in the past.

Example 2 *Consider the LTS $\mathcal{G}_2$ of Fig. 2b, with $\Sigma_o = \{a, b\}$. The secret given by $S = \{q_2\}$ is opaque, since after the observation of a, the attacker does not know whether the system is in state q_4 or q_2 and the secret is not leaked. However, after the observation of $a \cdot b \cdot a$, the unique execution sequence corresponding to this observation is $\tau \cdot a \cdot b \cdot a$, and the attacker can deduce that after the observation of the last a, the system was actually in state q_2 two steps ago.*

To take into account this particularity, we now introduce two notions of opacity: K-step weak opacity defined in Saboori and Hadjicostis (2007)[3], and K-step strong opacity. Intuitively, these notions take into account the opacity of the secret in the past and also allow to say that the knowledge of the secret becomes worthless after the observation of a given number of actions.

Definition 3 (K-step weak opacity) *For $K \in \mathbb{N}$, the secret S is K-step weakly opaque on $\mathcal{G}$ under the projection P_{Σ_o} or $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if*

$$\begin{aligned}\forall t \in \mathcal{L}(\mathcal{G}), \forall t' \preceq t : |t - t'|_{\Sigma_o} \leq K \wedge t' \in \mathcal{L}_S(\mathcal{G}) \\ \Rightarrow \exists s \in \mathcal{L}(\mathcal{G}), \exists s' \preceq s : s \approx_{\Sigma_o} t \wedge s' \approx_{\Sigma_o} t' \wedge s' \notin \mathcal{L}_S(\mathcal{G}).\end{aligned}$$

The secret S is K-step weakly opaque on $\mathcal{G}$ if for every execution t of $\mathcal{G}$, for every secret execution t' prefix of t with an observable difference inferior to K, there exist two executions s and s' observationally equivalent respectively to t and t' s.t. s' is not a secret execution.

Remark 1 *If S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque then S is $(\mathcal{G}, P_{\Sigma_o}, K')$-weakly opaque for $K' \leq K$. Moreover, 0-step weak opacity corresponds to simple opacity.*

[3] Compared with Saboori and Hadjicostis (2007), for simplicity, we only consider a unique initial state and deterministic LTSs.

Example 3 *For the system $\mathcal{G}_2$ in Fig. 2b, we can easily check that the secret is $(\mathcal{G}_2, P_{\Sigma_o}, 1)$-weakly opaque. However, the secret is not $(\mathcal{G}_2, P_{\Sigma_o}, 2)$-weakly opaque as after the observation of $a{\cdot}b{\cdot}a$, the only compatible execution sequence corresponding to this observation is $\tau \cdot a \cdot b \cdot a$, and the attacker can deduce, after the occurrence of the last a, that the system was actually in state q_2 two steps ago.*

To validate opacity with runtime techniques, we will need to characterize K-step weak opacity in terms of observable traces of the system.

Proposition 1 *$S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque iff*

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall \mu' \preceq \mu : |\mu - \mu'| \leq K \Rightarrow [\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G}).$$

S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if for each observable trace μ of the system, and each of its prefixes μ' with less than K observations less than μ, if there exists an execution compatible with μ' ending in a secret state, then there exists another compatible execution of the system that does not end in a secret state.

The proof of this proposition is given in Appendix A.1. In the sequel, the set of traces, for which the K-step weak opacity of the secret is revealed, is formally defined by:

$$\begin{aligned} &\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{W}}) \\ &\qquad \stackrel{\text{def}}{=} \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid \exists \mu' \preceq \mu : |\mu - \mu'| \leq K \wedge [\![\mu'/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})\} \end{aligned} \tag{1}$$

where $\mathsf{OP}_{\mathsf{K}}^{\mathsf{W}}$ indicates that predicate leak is parameterized with K-weak opacity.

Corollary 1 *S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque iff* $\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{W}}) = \emptyset$.

In some cases, it is interesting to characterize the set of traces that reveal the secret at *exactly* k steps with $k \leq K$. This set is defined as follows:

$$\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{W}}, k) \stackrel{\text{def}}{=} \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid (3) \wedge (4)\}, \tag{2}$$

with

$$\exists \mu' \preceq \mu : |\mu - \mu'| = k \wedge [\![\mu'/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G}), \tag{3}$$

$$\forall \mu' \preceq \mu : |\mu - \mu'| < k \Rightarrow [\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G}). \tag{4}$$

That is, there exists an observation trace that reveals the opacity of the secret k steps ago (3) and every observation trace which is produced strictly less than k steps ago does not reveal the opacity (4). Furthermore, one may notice that $\bigcup_{0 \leq k \leq K} \mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{W}}, k) = \mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{W}})$.

Example 4 (Limits of K-step weak opacity) *Let us consider the LTS $\mathcal{G}_3$ in Fig. 2c, with $\Sigma_o = \{a, b\}$. The secret is given by $S = \{q_2, q_6\}$. According to Definition 3, S is $(\mathcal{G}_3, P_{\Sigma_o}, K)$-weakly opaque for every $K \in \mathbb{N}$. However, after the observation $a \cdot b$, the attacker can deduce that $\mathcal{G}_3$ is either in state*

$q_6 \in S$ (if the actual execution sequence is $a \cdot b$) or was in state $q_2 \in S$ after the observation of a (if the actual execution sequence is $\tau \cdot a \cdot b$). In all cases, the attacker knows that at most one observation before the occurrence of b, $\mathcal{G}_3$ was in a secret state or is currently in a secret state.

The previous example leads us to introduce *K-step strong opacity*. This notion takes into account the limitation of K-step weak opacity and prevents the attacker to be sure that the secret occurred during the last K observable steps.

Definition 4 (K-step strong opacity) *For $K \in \mathbb{N}$, S is K-step strongly opaque on $\mathcal{G}$ under the projection P_{Σ_o} (or $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque) if*

$$\forall t \in \mathcal{L}(\mathcal{G}), \exists s \in \mathcal{L}(\mathcal{G}) : \Big(s \approx_{\Sigma_o} t \wedge \big(\forall s' \preceq s : |s - s'|_{\Sigma_o} \leq K \Rightarrow s' \notin \mathcal{L}_S(\mathcal{G})\big)\Big).$$

Intuitively, a secret is K-step strongly opaque if for each execution sequence of the system t, there exists another execution sequence s, equivalent to t, that never crossed a secret state during the last K observations. In other words, for each observable trace μ of the system, there exists at least one execution sequence compatible with μ that did not cross a secret state during the last K observations. Using the set

$$\mathrm{Free}_K^S(\mathcal{G}) \stackrel{\text{def}}{=} \{t \in \mathcal{L}(\mathcal{G}) \mid \forall t' \preceq t : \ |t - t'|_{\Sigma_o} \leq K \Rightarrow t' \notin \mathcal{L}_S(\mathcal{G})\},$$

corresponding to the execution sequences of $\mathcal{G}$ that did not cross a secret state during the last past K observations, we can give another characterization of strong opacity.

Proposition 2 *On $\mathcal{G}$, $S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque if and only if*

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : [\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_K^S(\mathcal{G}) \neq \emptyset.$$

The proof of this proposition is given in Appendix A.1.

Remark 2 *If S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque then it is also $(\mathcal{G}, P_{\Sigma_o}, K')$-strongly opaque for $K' \leq K$. And if S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque, then it is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque.*

We note $\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{S}}) \stackrel{\text{def}}{=} \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid [\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_K^S(\mathcal{G}) = \emptyset\}$ the set of traces for which there is an information flow w.r.t. K-step strong opacity where $\mathsf{OP}_{\mathsf{K}}^{\mathsf{S}}$ indicates that predicate leak is parameterized with K-strong opacity. As for K-step weak opacity, it is useful to know when exactly at most k steps ago the system went through a secret state. Then, the set $\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{S}})$ can be decomposed as

$$\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{S}}, k) \stackrel{\text{def}}{=} \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid [\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_k^S(\mathcal{G}) = \emptyset \\ \wedge \ \forall k' < k : [\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_{k'}^S(\mathcal{G}) \neq \emptyset\}.$$

We get that $\mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{S}}) = \cup_{k \leq K} \mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_{\mathsf{K}}^{\mathsf{S}}, k)$. Intuitively, the strong opacity of the secret leaks at k steps via the observed trace if it does not have a compatible execution sequence not free of secret states during the last k steps.

Example 5 *Back to $\mathcal{G}_3$ (Fig. 2c), following the reasons developed in Example 4, the secret is not $(\mathcal{G}_3, P_{\Sigma_o}, 1)$-strongly opaque. Considering the LTS $\mathcal{G}_4$ (Fig. 2d), the secret is $(\mathcal{G}_4, P_{\Sigma_o}, K)$-weakly opaque for every $K \in \mathbb{N}$ and $(\mathcal{G}_4, P_{\Sigma_o}, 1)$-strongly opaque. However the secret is not $(\mathcal{G}_4, P_{\Sigma_o}, 2)$-strongly opaque since after the observed trace $a \cdot b \cdot a$, we know that either the system is in q_7 (which is a secret state) or is in q_4. In both cases, we know that the system was in a secret state at most 2 steps ago.*

5 Verification of opacity at runtime

We are now interested in verifying the opacity of a secret on a given system modelled by an LTS. The device that we build, operates at runtime, but it can also be used to *model-check* opacity (see Section 5.3).

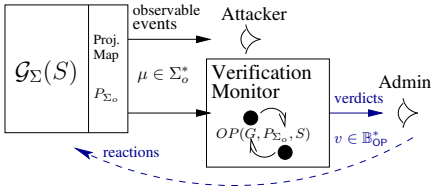


Fig. 3 Opacity verification at runtime

We aim to build a runtime verifier (see Fig. 3), i.e., a monitor capturing, for each observation $\mu \in \Sigma_o^*$ what a user can infer about the current execution of the system and the possible leakage of the secret w.r.t. the considered opacity. This monitor can be used by an administrator to discover opacity leakages on the system and take appropriate reactions.

In this section, we shall focus on K-weak/strong opacity. We refer to Dubreil et al (2009) and Falcone and Marchand (2010b) for details regarding the runtime verification of simple opacity.

Definition 5 (Runtime verifier) *A* runtime verifier *(R-Verifier) $\mathcal{V}$ is a finite state machine $(Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{V}}, \mathbb{D}_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ where $\Gamma^{\mathcal{V}} : Q^{\mathcal{V}} \to \mathbb{D}_{\mathsf{OP}}$ is the output function. $\mathbb{D}_{\mathsf{OP}}$ is a truth domain dedicated to the considered notion of opacity. For K-step based opacity $\mathbb{D}_{\mathsf{OP}} = \mathbb{D}^K_{\mathsf{OP}} \stackrel{\text{def}}{=} \{\text{leak}_0, \ldots, \text{leak}_K, \text{noleak}\}$.*

We now state the properties that an R-Verifier should satisfy:

Definition 6 (R-Verifier soundness and completeness) *An R-Verifier $\mathcal{V}$ is sound and complete w.r.t. $\mathcal{G}, P_{\Sigma_o}, S$ and $\mathsf{OP} \in \{\mathsf{OP}^W_K, \mathsf{OP}^S_K\}$ whenever $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall l \in [0, K]$:*

$$\begin{aligned} &\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_l \Leftrightarrow \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}, l) \\ \wedge\ &\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{noleak} \Leftrightarrow \mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}). \end{aligned}$$

An R-Verifier is sound ($\Rightarrow$ direction) if it never gives a false verdict. It is complete ($\Leftarrow$ direction) if all observations corresponding to the current leakage of opacity raise an appropriate "leak" verdict: it raises a noleak verdict when the opacity is preserved, an it raises a leak$_l$ verdict when the opacity leaks at l observable steps on the system.

5.1 K-delay trajectory estimators

When generating R-Verifiers of K-step weak and strong opacity, we will need the notion of K-delay trajectory estimator[4] which is an extension to the mechanism proposed in Saboori and Hadjicostis (2007). Intuitively, a K-delay trajectory estimator, according to the observation interface of a system, indicates the estimated states of the system during the K previous steps along with extra information about opacity leakage on transitions.

First, we need to introduce additional notations. Recall that the set of l-tuples of states, for $l \geq 2$, of $\mathcal{G}$ is $Q^l \stackrel{\text{def}}{=} Q^{\mathcal{G}} \times Q^{\mathcal{G}} \times \cdots \times Q^{\mathcal{G}} \stackrel{\text{def}}{=} \{(q_1, \ldots, q_l) \mid \forall i \in [1, l] : q_i \in Q^{\mathcal{G}}\}$. Intuitively elements of Q^l correspond to partial sequences of states. A set $m \in 2^{Q^l \times \mathbb{B}^{l-1}}$ is called an l-dimensional trajectory mapping. We denote by $m(i)$ the set of the $(l-i)^{\text{th}}$ trajectory of elements of m. Intuitively, $m(0)$ will correspond to the current state estimate whereas $m(i)$ will correspond to the trajectory estimate knowing that i observations have been made: it contains the state estimate knowing that i observations have been made and the secrecy of partial runs between state estimates i steps and $i-1$ step(s) ago. We also need to define:

– the *shift operator* $\blacktriangleleft$: $2^{Q^l \times \mathbb{B}^{l-1}} \times 2^{Q^2 \times \mathbb{B}} \to 2^{Q^l \times \mathbb{B}^{l-1}}$ s.t.

$$m \blacktriangleleft m_2 \stackrel{\text{def}}{=} \{((q_2, \ldots, q_{l+1}), (b_2, \ldots, b_l)) \in Q^l \times \mathbb{B}^{l-1} \mid ((q_1, \ldots, q_l), (b_1, \ldots, b_{l-1})) \in m \wedge ((q_l, q_{l+1}), b_l) \in m_2\},$$

– the observation mapping $\text{Obs} : \Sigma_{\text{o}} \to 2^{Q^2 \times \mathbb{B}}$ s.t.

$$\text{Obs}(\sigma) \stackrel{\text{def}}{=} \{((q_1, q_2), b) \in Q^2 \times \mathbb{B} \mid q_1, q_2, \in Q^{\mathcal{G}}, \exists s \in \Sigma^+ : P_{\Sigma_{\text{o}}}(s) = \sigma \wedge q_1 \stackrel{s}{\to}_{\mathcal{G}} q_2 \wedge b = s \in \text{Free}_1^S(\mathcal{G}(q_1))\},$$

– the function $\odot_l : 2^E \to 2^{E^l}$ s.t. $\odot_l(E) \stackrel{\text{def}}{=} \{(e, \ldots, e) \mid e \in E\}$, for a set E.

Based on the previous operations, K-delay trajectory estimators are defined as follows:

Definition 7 (K-Delay Trajectory Estimator) *For $\mathcal{G}$, a secret S, a projection $\mathcal{P}_{\Sigma_{\text{o}}}$, the* K-delay trajectory estimator *is an LTS $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$ s.t.:*

– *M^D is the smallest subset of $2^{Q^{K+1} \times \mathbb{B}^K}$ reachable from m^D_{init} with δ_D,*
– *$m^D_{\text{init}} \stackrel{\text{def}}{=} \{(t_q, t_{b_q}) \mid (1) \wedge (2)\}$ with:*

[4] We will also use it in Section 6 in order to enforce the various notions of opacity.

  - $(1) = t_q \in \odot_{K+1}(\{q\}) \wedge t_{b_q} \in \odot_K(\{b_q\})$,
  - $(2) = q \in \Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_o}) \wedge b_q \stackrel{\text{def}}{=} q \notin S$,
- $\delta_D : M^D \times \Sigma_o \to M^D$ *defined by* $\delta_D(m, \sigma) \stackrel{\text{def}}{=} m \blacktriangleleft \text{Obs}(\sigma)$, *for any* $m \in M^D$ *and* $\sigma \in \Sigma_o$.

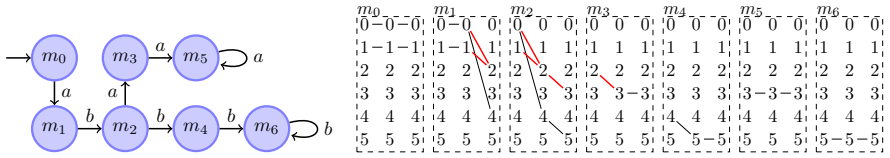
A K-delay trajectory estimator, for $\mathcal{G}$, is an LTS whose states contain suffixes of length K of "observable runs" that are compatible with the current observation on $\mathcal{G}$. Moreover, it also contains information regarding the secret states traversed between two consecutive states of $(q_0, \ldots, q_K)$, i.e., b_i is true whenever there exists a partial execution s between q_{i-1} and q_i that does not cross a secret state. On each transition fired by $\sigma \in \Sigma_o$, possibly visited states more than K steps ago are forgotten, and the current state estimate is updated: for a transition, the arriving state is obtained using the shift operator ($\blacktriangleleft$) and putting in front (at the location of the current state estimate) compatible current states according to the state estimate at the previous step (i.e., $\text{Obs}(\sigma)$ "filtered" by $\blacktriangleleft$).

Remark 3 (About initial state m^D_{init}) *Note that for the particular case of the initial state* m^D_{init}, *we do not use the set* $\text{Free}^S_l(\mathcal{G})$ *of free trajectories because it is used only for trajectories with at least one observation. Furthermore, the considered notions of opacity depend on observations and our* K*-delay trajectory estimators should remain consistent with the observation* ϵ_{Σ_o}. *That is, we should consider all trajectories compatible with the empty observation and thus all possible states reached by such trajectories. Thus, for the initial case, we consider only the reachable states with no observable event because the attacker does not know whether the system has started before observing an event: the states reachable through non-observable events are thus considered. Note that this is consistent with Saboori and Hadjicostis (2007) since in their definition the initial set of states was assumed to be either unknown (i.e., the whole state space) or reduced to a strict subset of the state space. In our case, we assume that the initial knowledge of the attacker, i.e., the set of possible system states when the attacker begins to monitor, is given by the set of states reachable from the initial state by a sequence of unobservable events. Note that we could have chosen, without any particular difficulty, a more general definition as the one in Saboori and Hadjicostis (2007).*

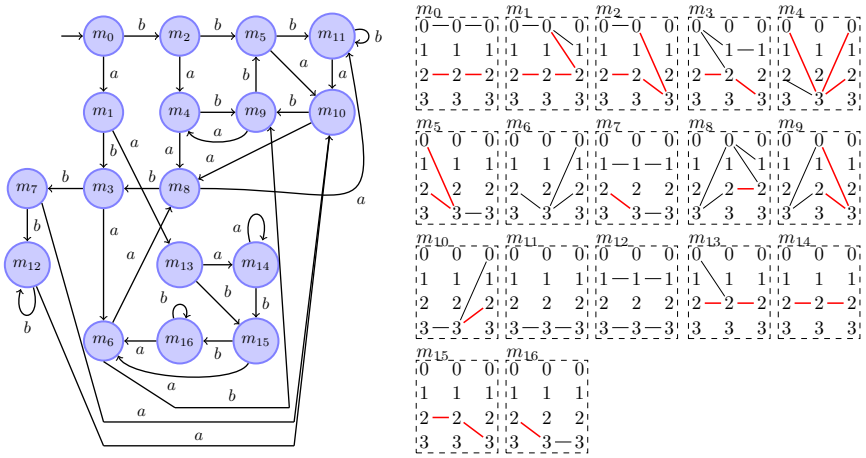
For K-step weak opacity analysis, we will only use information about traversed states. In this particular case, K-delay trajectory estimators reduce to the mechanism proposed in Saboori and Hadjicostis (2007). For K-step strong opacity analysis, we will need to use both traversed states and Booleans recording partial trajectory's opacity preservation.

Example 6 (2-delay trajectory estimators) *The 2-delay trajectory estimators*[5] *of* $\mathcal{G}_2$ *and* $\mathcal{G}_5$ *are respectively represented in Fig. 4a and Fig. 4b, with* $\Sigma_o = \{a, b\}$.

[5] For clarity, in the states of the graphic representation of K-delay trajectory estimators, a state q_i is noted i.



(a) 2-delay trajectory estimator for $\mathcal{G}_2$ of Fig. 2b



(b) 2-delay trajectory estimator for $\mathcal{G}_5$ of Fig. 2e

Fig. 4 Some trajectory estimators

– *For the 2-delay trajectory estimator of $\mathcal{G}_2$, examining m_2 gives us the following information: the current state estimate is $\{q_3, q_5\}$, i.e., $\mathcal{G}_2$ is either in q_3 or q_5. It was at the previous step in either state q_2 or q_4. It was two steps ago in either state q_0 or q_1. It followed one of the partial runs $q_0 \xrightarrow{\tau \cdot a}_{\mathcal{G}_2} q_2 \xrightarrow{b}_{\mathcal{G}_2} q_3$, $q_0 \xrightarrow{a}_{\mathcal{G}_2} q_4 \xrightarrow{b}_{\mathcal{G}_2} q_5$, $q_1 \xrightarrow{a}_{\mathcal{G}_2} q_2 \xrightarrow{b}_{\mathcal{G}_2} q_3$ represented by straight lines in the dashed box representing m_2. Moreover, every partial run is coloured in red whenever the actual compatible execution sequences surely crossed a secret state (i.e., when the Boolean is false). For instance in m_2, the partial runs $q_0 \xrightarrow{\tau \cdot a}_{\mathcal{G}_2} q_2 \xrightarrow{b}_{\mathcal{G}_2} q_3$ and $q_1 \xrightarrow{a}_{\mathcal{G}_2} q_2 \xrightarrow{b}_{\mathcal{G}_2} q_3$ are associated to execution sequences that surely crossed a secret state.*

– *For the 2-delay trajectory estimator of $\mathcal{G}_5$, examining m_9 gives us the following information: the current state estimate is $\{q_3\}$, i.e., $\mathcal{G}_5$ is surely in q_3. It was at the previous step either in q_0 or in q_2 and it followed one of the partial runs $q_3 \xrightarrow{a \cdot \tau}_{\mathcal{G}_5} q_2 \xrightarrow{b}_{\mathcal{G}_5} q_3$, $q_3 \xrightarrow{a}_{\mathcal{G}_5} q_0 \xrightarrow{\tau \cdot b}_{\mathcal{G}_5} q_3$ represented by straight lines in the dashed box representing m_9. The partial runs $q_0 \xrightarrow{\tau \cdot b}_{\mathcal{G}_5} q_3$ and $q_2 \xrightarrow{b}_{\mathcal{G}_5} q_3$ are associated to execution sequences that surely crossed a secret state (in these cases q_2).*

Suitability of K-delay trajectory estimators. K-delay trajectory estimators indeed exactly capture state estimation (with states tuples) and opacity preservation (with Boolean tuples) in the past. It is formally expressed by the two following lemmas[6].

Lemma 1 *Given a system $\mathcal{G}$ modelled by an LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$, then*

- $\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$ *s.t.* $|\mu| \geq K \wedge \mu = \mu' \cdot \sigma_1 \cdots \sigma_K$,
  $\forall s \in [\![\mu]\!]_{\Sigma_{\text{o}}} : (s = s' \cdot s_1 \cdots s_K \wedge \forall i \in [1, K] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i)$
  $\exists (q_0, \ldots, q_K) \in \delta_D(m^D_{\text{init}}, \mu) : q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$,

- $\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$ *s.t.* $n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n$,
  $\forall s \in [\![\mu]\!]_{\Sigma_{\text{o}}} : (s = s' \cdot s_1 \cdots s_n \wedge \forall i \in [1, n] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i)$,
  $\exists (q_0, \ldots, q_K) \in \delta_D(m^D_{\text{init}}, \mu) : q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots \xrightarrow{s_{n-1}}_{\mathcal{G}} q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K$,
  $\wedge\ q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n}$.

Lemma 2 *Given a system $\mathcal{G}$ modelled by an LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$, then $\forall m \in M^D$, $\forall (q_0, \ldots, q_K) \in m$, $\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m$:*

- $|\mu| = 0 \Rightarrow \forall (q, \ldots, q) \in m = m^D_{\text{init}}, \exists s \in [\![\mu]\!]_{\Sigma_{\text{o}}} : q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}} q \wedge P_{\Sigma_{\text{o}}}(s) = \epsilon$,

- $n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n \Rightarrow$
  $\exists s' \cdot s_1 \cdots s_n \in [\![\mu]\!]_{\Sigma_{\text{o}}} :$
  $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K \wedge \forall i \in [1, n] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i$

- $|\mu| \geq K \wedge \mu = \mu' \cdot \sigma_1 \cdots \sigma_K \Rightarrow$
  $\exists s' \cdot s_1 \cdots s_K \in [\![\mu]\!]_{\Sigma_{\text{o}}} :$
  $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots \xrightarrow{s_K}_{\mathcal{G}} q_K \wedge \forall i \in [1, K] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i$.

Technical proofs can be found in Appendix A.2. We prefer to give here the intuition on the suitability of K-delay trajectory estimators. Lemmas 1 and 2 state respectively the completeness and soundness of K-delay trajectory estimators:

- Lemma 1 states that for all observation traces that the system can produce, according to its length ($|\mu| \geq K$ or $|\mu| < K$) there exists a trajectory in one state of the trajectory estimator that records the appropriate last observation steps of μ (the last K ones if $|\mu| \geq K$ or the last ones if $|\mu| < K$).
- Lemma 2 states that all trajectories contained in all states of K-delay trajectory estimators correspond to actual observation traces on the system.

[6] As in Lemmas 1 and 2 we are only interested by the capture of state estimates by K-delay trajectory estimators, we focus on state tuples and ignore Booleans. That is, for the sake of readability, states of a K-delay trajectory estimators are noted $(q_0, \ldots, q_K)$ instead of $((q_0, \ldots, q_K), (b_0, \ldots, b_{K-1}))$.

Using the two previous lemmas, we can directly show the following proposition:

Proposition 3 *For $\mathcal{G}$ modelled by an LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, and $\Sigma_{\text{o}} \subseteq \Sigma$, the K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$ of $\mathcal{G}$ is s.t.:*

$$\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m \\ \Rightarrow \forall i \in [0, \min\{|\mu|, K\}] : m(i) = \delta_{\mathcal{G}}(q^{\mathcal{G}}_{\text{init}}, [\![\mu^{\cdots|\mu|-i-1}/\mu]\!]_{\Sigma_{\text{o}}}).$$

Even though differently presented, a similar result can be found in Saboori and Hadjicostis (2011).

5.2 R-Verifier synthesis

We tackle now the R-Verifier synthesis problem for K-step based opacity.

Synthesis for K-weak opacity. For K-weak opacity, R-Verifiers are synthesized directly from K-delay trajectory estimators.

Proposition 4 *For $\mathcal{G}$, $S \subseteq Q^{\mathcal{G}}$, the R-Verifier $\mathcal{V} = (Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_{\text{o}}, \delta_{\mathcal{V}}, \mathbb{D}^K_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ built from the K-delay trajectory estimator[7] $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$ of $\mathcal{G}$ where $Q^{\mathcal{V}} = M^D, q^{\mathcal{V}}_{\text{init}} = m^D_{\text{init}}, \delta_{\mathcal{V}} = \delta_D$, and $\Gamma^{\mathcal{V}} : Q^{\mathcal{V}} \to \mathbb{D}^K_{\mathsf{OP}}$ defined by*

- $\Gamma^{\mathcal{V}}(m) = \text{noleak}$ *if* $\forall k \in [0, K] : m(k) \notin 2^S$,
- $\Gamma^{\mathcal{V}}(m) = \text{leak}_{\text{l}}$ *where* $l = \min\{k \in [0, K] \mid m(k) \in 2^S\}$ *otherwise,*

is sound and complete w.r.t. $\mathcal{G}, P_{\Sigma_{\text{o}}}, S$ and the K-step weak opacity.

For K-weak opacity and an observation trace μ, an R-Verifier produces verdicts as follows. It produces the verdict noleak if for each of the K last prefixes of μ, there exists a compatible sequence that is not ending in a secret state. It produces the verdict leak_{l} if l is the minimum distance with an observation among the K last ones s.t. the system was in a secret state, $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, l)$. The proof of this proposition is given in Appendix A.2.3.

Synthesis for K-strong opacity. Synthesis of R-Verifiers for K-strong opacity is similarly based on the trajectories of K-delay estimators. As one expects, in a state of an estimator, the opacity of a secret will rely on whether all trajectories surely traverse a secret state. However, the set of considered trajectories requires to be be slightly adapted before being used to state verdicts. Intuitively, for some systems, some trajectory (represented by the traversed states) can be redundant with another in the sense that it contains overlapping actual pathes on the system that are observationnally equivalent for some suffix. Yet, given a number of observations, one trajectory could be free of secret state and not the other one. To see how this can pose an issue for the evaluation of strong-opacity leakage, consider the system in the example below.

[7] For clarity, for K-step weak opacity, only information about traversed state is considered in trajectory estimators since it is the sole information needed.

Example 7 (System with overlapping trajectory) *Consider $\mathcal{G}_p$ depicted in Fig. 5a where $\Sigma_o = \{a, b\}$ and $S_p = \{q_1, q_3\}$. The sequences a and $\tau \cdot \tau \cdot a$ are observed as a. The system $\mathcal{G}_p$ necessarily traverses a secret state when executing these sequences. Indeed, we have $a \in \mathcal{L}_{S_p}(\mathcal{G}_p)$ and consequently $a \notin \text{Free}_1^{S_p}(\mathcal{G}_p)$. Moreover, since $\tau \in \mathcal{L}(\mathcal{G}_p)$, $\tau \preceq \tau \cdot \tau \cdot a$, and $|\tau \cdot \tau \cdot a - \tau|_{\Sigma_o} \leq 1$, we have $\tau \cdot \tau \cdot a \notin \text{Free}_1^{S_p}(\mathcal{G}_p)$. By Proposition 2, S_p is not $(\mathcal{G}_p, P_{\Sigma_o}, 1)$-strong opaque. However, examining the trajectories contained in state m_1, the partial trajectory $q_2 \xrightarrow{a}_{\mathcal{G}_p} q_4$ could mislead us in the sense that it does not traverse a secret state. But, to enter state q_2 the system must traverse q_1, and, going from q_1 to q_2 does not require any further observation. Roughly speaking, the state m_1 contains redundant information w.r.t. strong opacity and the aforementioned trajectory should be projected.*

We now formalize this notion of projection using the notion of *redundant trajectory*. Given a system $\mathcal{G}$ and its associated trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ and $m \in M^D$, we define the predicate redundant : $M^D \times (Q^K \times \mathbb{B}^{K-1}) \times (Q^K \times \mathbb{B}^{K-1}) \to \{true, false\}$ indicating whether or not, given $m \in M^D$, a trajectory $t = \big((q_1, \ldots, q_K), (b_1, \ldots, b_{K-1})\big) \in m$ is redundant with a trajectory $t' = \big((q'_1, \ldots, q'_K), (b'_1, \ldots, b'_{K-1})\big) \in m$:

$$\begin{aligned}&\text{redundant}(m, t, t') = \exists j \in [1, K-1] : \\ &\quad b_j \neq b'_j \wedge q_j \neq q'_j \wedge \forall i \in [j+1, K] : q_i = q'_i \wedge \forall i \in [j+1, K-1] : b_i = b'_i \\ &\wedge \forall s'' \in \Sigma^* : |s''|_{\Sigma_o} = 1 \wedge q'_j \xrightarrow{s''}_{\mathcal{G}} q'_{j+1} \Rightarrow q_j \in \Delta_{\mathcal{G}}\big(\{q'_j\}, \text{Pref}(\{s''\})\big).\end{aligned}$$

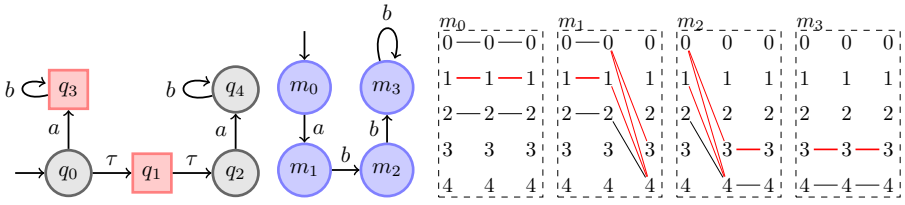
Within a state m of a trajectory estimator, $t = \big((q_1, \ldots, q_K), (b_1, \ldots, b_{K-1})\big)$ is redundant with $t' = \big((q'_1, \ldots, q'_K), (b'_1, \ldots, b'_{K-1})\big)$ if one can find an index $j \in [1, K-1]$ s.t. t and t' agree on trajectory elements with index greater than j (i.e., from $j+1$ to K states are equal, and from $j+1$ to $K-1$ Booleans are equal) and differ at position j. This means that the trajectories agree from the current state to the $(K-j)^{\text{th}}$ state in the past. Moreover, all paths of observable length 1 from q'_j to q'_{j+1} go through q_j. Note, the interval $[j+1, K-1]$ is possibly denoting an empty set (when $j = K-1$).

Observe that in the trajectory estimator of Example 7, some trajectories are redundant. In state m_1, the trajectories $\big((2,2,4), (true, true)\big)$ and $\big((1,1,4), (false, false)\big)$ are redundant with the trajectory $\big((0,0,4), (false, false)\big)$: the trajectories agree on the current state (state 4) and differ on index 1. Similarly, in state m_2, the trajectories $\big((2,4,4), (true, true)\big)$ and $\big((1,4,4), (false, false)\big)$ are redundant with the trajectory $\big((0,4,4), (false, false)\big)$.

Filtering out the redundant trajectories of a state m is noted $m\downarrow$ and defined as: $m\downarrow \stackrel{\text{def}}{=} \{t \in m \mid \nexists t' \in m : \text{redundant}(m, t, t')\}$.

Proposition 5 *For $\mathcal{G}$, $S \subseteq Q^{\mathcal{G}}$, the R-Verifier $\mathcal{V} = (Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{V}}, \mathbb{D}^K_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ built from the K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ where $Q^{\mathcal{V}} = M^D, q^{\mathcal{V}}_{\text{init}} = m^D_{\text{init}}, \delta_{\mathcal{V}} = \delta_D$, and $\Gamma^{\mathcal{V}} : Q^{\mathcal{V}} \to \mathbb{D}^K_{\mathsf{OP}}$ defined by:*

- $\Gamma^{\mathcal{V}}(m) = \text{noleak}$ *if* $\exists((q_i)_{0 \leq i \leq K}, (b_i)_{0 \leq i \leq K-1}) \in m\downarrow, \forall i \in [0, K]$: $q_i \notin S \wedge \forall i \in [0, K-1] : b_i = true$



(a) System $\mathcal{G}_p$ (b) 2-delay trajectory estimator of $\mathcal{G}_p$

Fig. 5 The trajectories in states m_1 and m_2 of the K-delay estimator need to be "projected"

– $\Gamma^{\mathcal{V}}(m) = \text{leak}_l$ *otherwise, with*

$$l = \min\Big\{l' \in [0,K] \mid \forall((q_i)_{0\le i\le K},(b_i)_{0\le i\le K-1}) \in m\downarrow, \\ \exists i \le l' : \big(l' \neq 0 \Rightarrow (q_{K-i} \in S \ \vee \ b_{K-i} = \mathit{false}) \wedge l' = 0 \Rightarrow q_K \in S\big)\Big\}$$

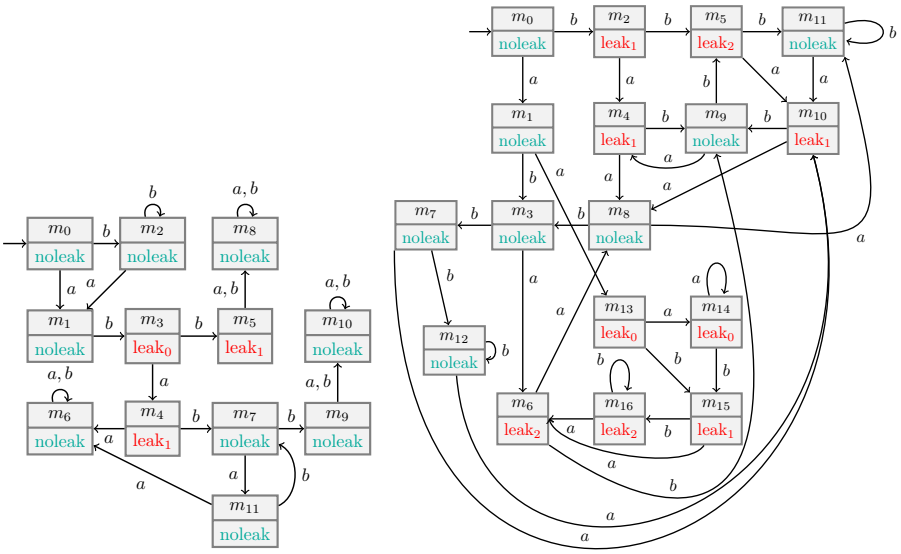
is sound and complete w.r.t. $\mathcal{G}, P_{\Sigma_o}, S$ *and* K*-step strong opacity.*

For K-strong opacity and an observation trace μ, an R-Verifier produces verdicts as follows. In each state of the state estimator, trajectories are filtered if redundant. The R-Verifier produces the verdict noleak if there exists a trajectory of the system, compatible with μ, s.t. during the last K observations, the system has not visited any secret state. It produces the verdict leak_l if l is the minimum number of observation steps for which the secret has occurred for sure, i.e., l is the minimum number of observations in the past after which all trajectories recorded in the estimator either ended in a secret state or went through a secret state. That is, $l = \min\{l' \in [0,K] \mid \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}, l')\}$. The proof of this proposition is given in Appendix A.2.4.

Example 8 (R-Verifiers) *We present some R-Verifiers: for the* 1*-step opacity (weak and strong) of* $\mathcal{G}_1$ *in Fig. 6a, and for the* 2*-step strong opacity of* $\mathcal{G}_5$ *in Fig. 6b. These R-Verifiers give us the runtime information about opacity leakage on their respective systems. For instance, for the R-Verifier of* $\mathcal{G}_5$*, if* $\mathcal{G}_5$ *produces the observation trace* $a \cdot b \cdot a$*, this R-Verifier gives the information about the opacity in a lock-step manner with the execution produced by* $\mathcal{G}_5$*. When the system produces* a *and then a subsequent* b*, these sequences do not reveal the opacity, the R-Verifier produces* noleak*. Then, when* $\mathcal{G}_5$ *produces another* a*, the sequence* $a \cdot b \cdot a$ *reveals the strong opacity of the secret 2 steps in the past, which is indicated by the R-Verifier producing* leak_2*.*

5.3 Model-checking the previous notions of opacity

While presented for the *runtime* verification of opacity, the mechanism of R-Verifier can be used to *model-check* the opacity of the system. In that case, one has just to build the appropriate K-delay trajectory estimator and R-Verifier, and check the reachability of states indicating an opacity leakage



(a) For the 1-step weak and strong opacity of $\mathcal{G}_1$

(b) For 2-strong opacity of $\mathcal{G}_5$

Fig. 6 R-Verifiers for K-step opacity

in the R-Verifier. Counter examples are just paths in the R-Verifier leading to leaking states. Moreover, if one wants to generate counter examples for a precise opacity leakage leak_l, one just has to compute the language accepted by R-Verifiers seen as LTSs for which marked states are those in which the output function produces leak_l.

Example 9 (Model-checking) *Using the R-Verifier of* 1*-step opacity (weak and strong) of* $\mathcal{G}_1$*, we can see that the secret of* $\mathcal{G}_1$ *is not* 1*-step weak nor strong opaque. The observable language of all counter examples is* $b^* \cdot a \cdot b \cdot (a+b+\epsilon)$*. The language of counter examples that leak the opacity of the secret exactly* 1 *step ago is* $b^* \cdot a \cdot b \cdot (a+b)$*.*

5.4 Summary and discussion

In this section, we have studied how we can verify at runtime, using the notion of R-Verifier, the previously introduced notions of opacity. To do so, we have introduced the mechanism of K-delay trajectory estimators from which R-Verifiers are directly generated. Depending on the notion of opacity of interest, we use the information about traversed states and secrecy of trajectories recorded in the states of K-delay trajectory estimators.

In this section, we have presented the runtime verification of opacity in which runtime verifiers are generated *a priori* and then combined with the

system at runtime. This technique has the advantage of minimizing the runtime computation (thus reducing the overhead on the original system). It is also possible to generate R-Verifiers *on-the-fly*. Indeed, while being in any state of the R-Verifier, the sole information that is needed to compute the verdict corresponding to a trace continuation with a new event is already present in the current state. On-the-fly generation of R-Verifiers has the advantage to minimize the space needed to perform runtime verification. In that case, it is needed to memorize only the current state. This is the usual trade-off between time and space consumptions. The proposed approach is compatible with both strategies.

Complexity issues for a mechanism similar to our K-delay trajectory estimators were studied in Saboori and Hadjicostis (2011). The complexity of the mechanisms proposed in this section follows from the complexity given in Saboori and Hadjicostis (2011). Indeed, our K-delay trajectory estimators just record a supplementary Boolean information. The complexity of computing this information representing whether or not a partial trace is free of secret states amounts to a reachability analysis in the initial system which is linear. Indeed, from an implementation point of view, the computation of this Boolean information is achieved first by considering the set of states of the system minus the secret states and second by performing a reachability check on this new state space. Moreover, R-Verifiers are naturally of the same complexity of the K-delay trajectory estimators they are generated from.

The verification of opacity is just concerned in detecting bad behaviors on a system. While verification might be a sufficient assurance, for some kinds of systems, opacity violation might be unacceptable. A further step in opacity validation is to automatically prevent opacity leakage. This is the problem addressed in the next section.

6 Enforcement of opacity at runtime

In this section, we consider a system $\mathcal{G}$, and we aim to build runtime enforcers for the previously introduced notions of opacity. Moreover, an underlying hypothesis is that the system is *live*, i.e., not deadlocked and always produces events, e.g., a reactive system

6.1 Informal principle and problem statement

Roughly speaking, the purpose of a runtime enforcer is to read some (unsafe) execution sequence produced by $\mathcal{G}$ (input to the enforcer) and to transform it into an output sequence that is safe regarding opacity (see Fig. 7).

A runtime enforcer acts as a delayer on an input sequence μ, using its internal memory to memorize some of the events produced by $\mathcal{G}$. The runtime enforcer releases a prefix o of μ containing some stored events, when the system has produced enough events so that the opacity is ensured. In other words,

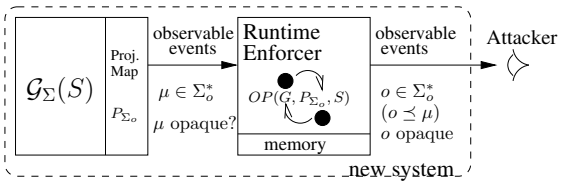
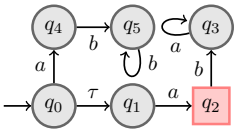


Fig. 7 Opacity enforcement at runtime

when the enforcer releases an output o (the only sequence seen by the attacker), then either this sequence does not reveal the opacity of the secret or if it does reveal it, the system has already produced a sequence $\mu \succeq o$, making the knowledge of o obsolete to the attacker. For instance, for a K-step based notion of opacity, if the enforcer releases a sequence o leaking the secret at $k \leq K$ steps, it has already received a sequence μ from the system s.t. $|\mu| - |o| > K - k$.

Let us illustrate informally how we expect to enforce opacity on an example.

Example 10 (Principle of enforcing opacity) *Let us go back on the system $\mathcal{G}_2$ introduced in Example 2. As we have seen previously, the secret is simply opaque but not $(\mathcal{G}_2, P_{\Sigma_o}, 2)$-weakly opaque. Indeed, after the observation of $a \cdot b \cdot a$, the only compatible trajectory corresponding to this observation is $\tau \cdot a \cdot b \cdot a$. Then, the attacker can deduce that, the system was actually in state q_2 two steps ago.*



Sequence of $\mathcal{G}_2$	Observable sequence	Memory	Output
τ	ϵ	ϵ	ϵ
$\tau \cdot a$	a	ϵ	a
$\tau \cdot a \cdot b$	$a \cdot b$	ϵ	$a \cdot b$
$\tau \cdot a \cdot b \cdot a$	$a \cdot b \cdot a$	a	$a \cdot b$
$\tau \cdot a \cdot b \cdot a \cdot a$	$a \cdot b \cdot a \cdot a$	ϵ	$a \cdot b \cdot a \cdot a$
$\tau \cdot a \cdot b \cdot a^+$	$a \cdot b \cdot a^+$	ϵ	$a \cdot b \cdot a^+$

Fig. 8 Enforcement of opacity on a trace of $\mathcal{G}_2$

With a runtime enforcer, we will delay this sequence in such a way that, when the attacker determines that the system was in a secret secret, it is always more than $K = 2$ steps ago on the real system. That is, some of the events produced by the system will be retained inside the enforcer memory. Intuitively, for the aforementioned sequence, the expected behavior of a runtime enforcer is as follows (see Fig. 8). When the system produces the sequence $\tau \cdot a$, the enforcer should not modify the observation trace a which is safe regarding opacity. When the system produces the sequence $\tau \cdot a \cdot b$, the enforcer observes $a \cdot b$ and lets the system execute normally (we expect the system execution to be minimally modified). Then, when the system produces a new a, the enforcer memorizes this event (the attacker still sees $a \cdot b$). Next, when the system produces another a, the system was in a secret state 3 steps ago. Thus, the enforcer can release the first stored a. Indeed, when the attacker observes $a \cdot b \cdot a$, the system has produced $a \cdot b \cdot a \cdot a$, and was in the secret state q_2 three steps ago: $(\mathcal{G}_2, P_{\Sigma_o}, 2)$-

weak opacity of S is thus preserved. At last, the last received a and subsequent ones can be freely output by the enforcer since they will not lead to a 2-weak opacity leakage.

Following this principle, the runtime enforcer for the 2-weak opacity of $\mathcal{G}_2$ is informally described in the following example.

Example 11 (Runtime enforcer) *The runtime enforcer for the 2-weak opacity of $\mathcal{G}_2$ is given in Fig. 9. Informally, the runtime enforcer is a finite-state machine that reads observable events from the system and execute an enforcement operation upon receiving each event. On receiving $a \cdot b$ it lets the system execute normally by executing twice the* dump *operation that releases directly the events in output. Then, when receiving an a event, it executes the* store_1 *operation to memorize the event in its internal memory for one unit of time. The* off *operation is executed when the runtime enforcer is not needed anymore. Similarly to the* dump *operation, it releases events in output*

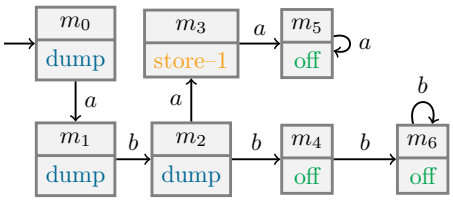


Fig. 9 Runtime enforcer for the 2-weak opacity of $\mathcal{G}_2$

6.2 Defining Runtime Enforcers

We formally define a notion of runtime enforcers which are special finite-state machines. By reading events, they produce enforcement operations that delay the input trace or release some already stored events so as to ensure the desired opacity. The introduced notion of enforcers is generic. In Section 6.4, we will specialize runtime enforcers for opacity but they can be similarly specialized for other notions of opacity or for regular properties (as seen for instance in Falcone et al (2011)).

Definition 8 (Enforcement operations *Ops* and memory) *The memory of runtime enforcers is a list whose elements are pairs consisting of an observable event and an integer. The set of possible configurations of the memory is thus $\mathcal{M}(T) = \bigcup_{i=0}^{T}(\Sigma_o \times \mathbb{N})^i$. When an element $(\sigma, d) \in \Sigma_o \times \mathbb{N}$ is inside the memory, it means that the event σ has to be retained d units of time before being released by the enforcer to preserve opacity. Enforcement operations take as inputs an observable event and a memory content (i.e., a special sequence of events, detailed later) to produce in output an observable sequence and a new memory content: $Ops \subseteq 2^{(\Sigma_o \times \mathcal{M}(T)) \to (\Sigma_o^* \times \mathcal{M}(T))}$.*

As seen in Example 11, enforcement operations consist of e.g., memorizing input events for some time units or halting the underlying system. In Section 6.4, we will formally define suitable enforcement operations for opacity.

Definition 9 (Generic R-Enforcer (R-Enforcer(Ops))) *An R-Enforcer $\mathcal{E}$ is a 6-tuple $(Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_{\text{o}}, \delta_{\mathcal{E}}, Ops, \Gamma^{\mathcal{E}}, \mathcal{M}(T))$ defined relatively to a set of observable events Σ_{o} and parameterized by a set of enforcement operations Ops. The finite set $Q^{\mathcal{E}}$ denotes the control states, $q^{\mathcal{E}}_{\text{init}} \in Q^{\mathcal{E}}$ is the initial state. $\delta_{\mathcal{E}} : Q^{\mathcal{E}} \times \Sigma_{\text{o}} \to Q^{\mathcal{E}}$ is the transition function. The function $\Gamma^{\mathcal{E}} : Q^{\mathcal{E}} \to Ops$ associates an enforcement operation to each state of the enforcer.*

Informally an R-Enforcer performs a transition by reading an event produced by the system. The arriving state of the transition is associated to an enforcement operation which is applied to the current event and the memory content.

The notion of *run* is naturally transposed from its definition for LTSs: for a trace $\mu = \sigma_0 \cdots \sigma_{n-1}$ of length n $run(\mu, \mathcal{E}) = (q_0, \sigma_0/\alpha_0, q_1) \cdot (q_1, \sigma_1/\alpha_1, q_2) \cdots (q_{n-1}, \sigma_{n-1}/\ \alpha_{n-1}, q_n)$, with $\forall i \in [0, n-1] : \Gamma^{\mathcal{E}}(q_{i+1}) = \alpha_i$. In the remainder of this section, $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_{\text{o}}, \delta_{\mathcal{E}}, Ops, \Gamma^{\mathcal{E}}, \mathcal{M}(T))$ designates an R-Enforcer with a set of enforcement operations *Ops*, and $\mu \in \Sigma^*_{\text{o}}$ designates the current observation trace of the system input to the R-Enforcer.

We formalize how R-Enforcers(Ops) react to input traces provided by a target system through the standard notions of *configuration* and *derivation*.

Definition 10 (Semantics of R-Enforcer(Ops)) *A* configuration *is a 3-tuple $(q, \mu, c) \in Q^{\mathcal{E}} \times \Sigma^*_{\text{o}} \times \mathcal{M}(T)$ where q denotes the current control state, μ the remaining trace to read, and c the current memory configuration.*

- *A configuration (q', μ', c') is* derivable in one step *from the configuration (q, μ, c) and* produces the output[8] *$o \in \Sigma^*_{\text{o}}$, and we note $(q, \mu, c) \overset{o}{\hookrightarrow} (q', \mu', c')$ if and only if $\mu = \sigma \cdot \mu' \wedge q \xrightarrow{\sigma}_{\mathcal{E}} q' \wedge \Gamma^{\mathcal{E}}(q') = \alpha \wedge \alpha(\sigma, c) = (o, c')$.*
- *A configuration C' is* derivable in several steps *from a configuration C and* produces the output *$o \in \Sigma^*_{\text{o}}$, and we note $C \overset{o}{\Longrightarrow}_{\mathcal{E}} C'$, if and only if*

$$\begin{aligned} \exists k \geq 0, \exists C_0, C_1, \ldots, C_k : C = C_0 \wedge C' = C_k \\ \wedge \forall i \in [0, k[, \exists o_i \in \Sigma^*_{\text{o}} : C_i \overset{o_i}{\hookrightarrow} C_{i+1} \\ \wedge\ o = o_0 \cdots o_{k-1}. \end{aligned}$$

We define the transformation performed by an R-Enforcer, with a set of enforcement operations Ops.

Definition 11 (Trace transformation) *$\mathcal{E}$ transforms μ into the output trace $o \preceq \mu$ as defined by the relation $\Downarrow_{\mathcal{E}} \subseteq \Sigma^*_{\text{o}} \times \Sigma^*_{\text{o}}$, where ϵ refers to $\epsilon_{\Sigma_{\text{o}}}$:*

- $\epsilon \Downarrow_{\mathcal{E}} \epsilon$,
- *$\mu \Downarrow_{\mathcal{E}} o$ if $\exists q \in Q^{\mathcal{E}}, \exists c \in \mathcal{M}(T) : (q^{\mathcal{E}}_{\text{init}}, \mu, \epsilon_{\mathcal{M}}) \overset{o}{\Longrightarrow}_{\mathcal{E}} (q, \epsilon, c)$.*

[8] Here note that o can be ϵ if the enforcer chooses to not produce an output.

The empty sequence ϵ is not modified by $\mathcal{E}$ (i.e., when the system does not produce any event). The observation trace $\mu \in \Sigma_o^*$ is transformed by $\mathcal{E}$ into the trace $o \in \Sigma_o^*$, when the trace is transformed from the initial state of $\mathcal{E}$, starting with an empty memory. Note that the resulting memory configuration c depends on the sequence $\mu \setminus o$ of events read by the R-Enforcer but not produced in output yet as we shall see in the remainder of this section.

Remark 4 *The proposed semantics is generic. Indeed, in addition of being used with enforcement operations for opacity, the semantics can be used with similar enforcement operations in order to enforce other notions of opacity or even regular temporal properties. To enforce regular safety properties (stating that "nothing bad should ever happen") one can use enforcement operations such as the dump and halt operations to obtain enforcement monitors as proposed in Schneider (2000). As far as the concatenation of the new received event and the already dumped sequence satisfies the safety property, the event is immediately dumped. Otherwise, the event is removed and the system feeding the enforcer is halted. To enforce regular co-safety properties (stating that "something good will happen within a finite amount of time"), one can use a store (with an unbounded delay) and dump operations to obtain enforcement monitors as proposed in Ligatti et al (2005). The store operation is used to memorize events while the input sequence does not satisfy (yet) the property, while the the dump operation releases all events in the memory as soon as the property is satisfied. To enforce any regular property such as a response property (liveness properties), one can combine the previously mentioned operations to obtain enforcement monitors as proposed in Falcone et al (2012).*

6.3 Enforcing the opacity at runtime

Before defining this enforcement notion more formally, we first formalize, for a given trace of $\mathcal{G}$, which of its prefixes can be safely output.

Definition 12 (Prefixes that are safe to output)

- *For simple opacity* OP_0*, a trace* $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$*, we say that it is safe to output* $\mu' \preceq \mu$*, noted* $\text{safe}_{\mathsf{OP}_0}(\mu, \mu')$*, if* $\big(\mu = \mu' \wedge \mu' \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)\big) \vee \mu' \prec \mu$*.*
- *For a* K*-step based notion of opacity* $\mathsf{OP}_\mathsf{K} \in \{\mathsf{OP}^\mathsf{K}_\mathsf{W}, \mathsf{OP}^\mathsf{K}_\mathsf{S}\}$*, a trace* $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$*, we say that it is safe to output* $\mu' \preceq \mu$*, noted* $\text{safe}_{\mathsf{OP}_\mathsf{K}}(\mu, \mu')$*, if*

$$\begin{array}{l} \mu' \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_\mathsf{K}) \\ \vee\ (\exists k \leq K : \mu' \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_\mathsf{K}, k) \wedge |\mu| - |\mu'| > K - k). \end{array}$$

That is, it is safe to produce $\mu' \preceq \mu$ if

- for simple opacity, either μ' does not reveal the opacity or that it reveals the opacity but it was produced on the system at least one step ago;
- for K-step based opacity, either μ' does not reveal the opacity or it reveals the opacity at k steps but it was produced on the system more than k steps ago.

Note that when it is safe to produce a given trace, then all its prefixes are safe to produce. That is, for $\mathsf{OP} \in \{\mathsf{OP}_0, \mathsf{OP}^{\mathsf{K}}_{\mathsf{W}}, \mathsf{OP}^{\mathsf{K}}_{\mathsf{S}}\}$:

$$\forall \mu, \mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \mathrm{safe}_{\mathsf{OP}}(\mu, \mu') \Rightarrow \forall \mu'' \prec \mu' : \mathrm{safe}_{\mathsf{OP}}(\mu, \mu'').$$

Furthermore, by convention, we will only consider systems for which it is safe to produce ϵ_{Σ_o}, i.e., when some sequences of $[\![\epsilon]\!]_{\Sigma_o}$ are not secret. Under the assumption that the system is alive, for a given trace, there always exists one of its extension traces which is safe to output, i.e.,

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \exists \mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \mu \preceq \mu' \wedge \mathrm{safe}_{\mathsf{OP}}(\mu', \mu)$$

e.g., any μ' s.t. $|\mu' - \mu| > K$. Moreover, the set of traces that lead a given sequence to be safe is extension-closed, i.e.,

$$\begin{aligned}\forall \mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : (\exists \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \mu' \preceq \mu \wedge \mathrm{safe}_{\mathsf{OP}}(\mu, \mu')) \\ \Rightarrow (\forall \mu'' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \mu \preceq \mu'' \Rightarrow \mathrm{safe}_{\mathsf{OP}}(\mu'', \mu')).\end{aligned}$$

Remark 5 *The previous properties have interesting practical consequences for runtime enforcers. First, since the set* $\{\mathrm{safe}_{\mathsf{OP}}(\mu, \mu') \mid \mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G})\}$ *is prefix-closed for any* $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$*, it means that, by outputting events, enforcers do not reduce the set of sequences that could be released in the future. Second, since the set* $\{\mathrm{safe}_{\mathsf{OP}}(\mu', \mu) \mid \mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G})\}$ *is extension-closed for any* $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$*, it means that, after releasing events in output, there is no possible future events that could be released and question the previously sequence of output events.*

Expected properties for runtime enforcers. We now explain what we mean exactly by *opacity enforcement*, and what are the consequences of this definition on the systems, secrets, and projections we shall consider. The following constraints are expected to hold for the enforcers we aim to synthesize.

soundness: the output trace should preserve the opacity of the system;
transparency: the input trace should be modified *in a minimal way*, namely if it already preserves opacity it should remain unchanged, otherwise its *longest prefix* preserving opacity should be issued.

On Example 10, soundness entails a runtime enforcer to e.g., output $a \cdot b$ (instead of $a \cdot b \cdot a$) when $\mathcal{G}_2$ produces $\tau \cdot a \cdot b \cdot a$. Transparency entails a runtime enforcer to e.g., output $a \cdot b \cdot a \cdot a$ (instead of any prefix) when $\mathcal{G}_2$ produces $\tau \cdot a \cdot b \cdot a \cdot a$.

Remark 6 *There always exists a trivial, sound but generally non transparent, enforcer delaying every event by* K *units of time for* K*-step based opacity.*

The formal definition of opacity-enforcement by an R-Enforcer(Ops) relates the input sequence produced by the program fed to the R-Enforcer(Ops) and the output sequence allowed by the R-Enforcer(Ops) so that the R-Enforcer is sound and transparent.

Definition 13 (Enforcement of opacity by an enforcer) *The R-Enforcer $\mathcal{E}$ enforces the opacity $\mathsf{OP} \in \{\mathsf{OP}_0, \mathsf{OP}^{\mathsf{K}}_{\mathsf{W}}, \mathsf{OP}^{\mathsf{K}}_{\mathsf{S}}\}$ of S w.r.t. P_{Σ_o} on a system $\mathcal{G}$ if $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \exists o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow$*

$$\mu \notin \mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \Rightarrow o = \mu \tag{5}$$

$$\mu \in \mathrm{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \Rightarrow o = \max_{\preceq}\{\mu' \preceq \mu \mid \mathrm{safe}_{\mathsf{OP}}(\mu, \mu')\} \tag{6}$$

A sound and transparent R-Enforcer always produces maximal safe sequences:

Property 1 Given a sound and transparent runtime enforcer $\mathcal{E}$:

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow \big(\mathrm{safe}_{\mathsf{OP}}(\mu, o) \wedge \forall o \prec o' \preceq \mu : \neg\, \mathrm{safe}_{\mathsf{OP}}(\mu, o')\big),$$

for $\mathsf{OP} \in \{\mathsf{OP}_0, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}\}$.

Most of the previous approaches (e.g., Ligatti et al (2005); Falcone et al (2009b)) on property enforcement used enforcement mechanisms with a finite but unbounded memory under the soundness and transparency constraints. Since we are setting our approach in a general security context, we go one step further on the practical constraints expected for a desired enforcement mechanism dedicated to opacity. Here we consider that the memory allocated to the enforcer has a given allocated size[9]. Besides the soundness and transparency constraints, we add the following one:

do-not-overflow: to enforce opacity, the size of the partial trace memorized by the enforcer does not exceed the allocated memory size.

When is the opacity of a secret enforceable on a system? After stating the constraints on runtime enforcement for opacity, we need to delineate the systems, interfaces and secrets s.t. opacity is enforceable using runtime enforcers. The existence of sound and transparent R-Enforcers with unbounded memory for opacity relies first on the provided characterization of opacity preservation on the observable behavior as finitary properties (Section 4) and second on existing results in enforcement monitoring of finitary properties (see e.g., Ligatti et al (2005); Falcone et al (2009b)). Now, the existence of an R-Enforcer for K-step based opacity[10] relies only on the *do-not-overflow* constraint of a memory of size T, which is satisfied iff

$$\max_{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})} \big\{ \min\{|\mu \setminus o| \mid o \preceq \mu \wedge \mathrm{safe}_{\mathsf{OP}_{\mathsf{K}}}(\mu, o)\}\big\} \leq T.$$

The previous enforcement criterion is not usable in practice and is not computable generally. Thus, we will give a more practical and decidable enforcement criterion using K-delay trajectory estimators. To each state of the K-delay trajectory estimator, we have seen that it is possible to determine the opacity leakage. Intuitively, the reasoning is as follows. If we are considering a

[9] Besides memory size limitation, this constraint can represent the desired quality of service, e.g., maximal allowed delay.

[10] The existence of an R-Enforcer for simple opacity relies on the trivial condition $T \geq 1$.

K-step based notion of opacity and we reach a state in the K-delay trajectory estimator s.t. it reveals the opacity of the secret 2 steps ago (e.g., the attacker knows that the system was in a secret state 2 steps ago). Then for $K \geq 2$, the enforcer has to delay the last event produced by the system by $K-1$ units of time. Indeed, after that, the attacker will know that the system was in a secret state $K+1$ steps ago. This knowledge is safe w.r.t. K-step based opacity.

The criterion on K-delay trajectory estimators uses the following lemma, which is a direct consequence of Lemmas 1 and 2.

Lemma 3 (States of K-delay trajectory estimators and opacity leakage) *Given a system $\mathcal{G}$, a K-step based notion of opacity $\mathsf{OP}_\mathsf{K} \in \{\mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}\}$ w.r.t. a projection map P_{Σ_o} and a secret S, the states of the K-delay trajectory estimator $D = (M^D, q^D_{\mathrm{init}}, \Sigma_\mathrm{o}, \delta_D)$ are s.t.:*

$$\begin{aligned}\forall \mu_1, \mu_2 \in \mathcal{T}_{\Sigma_\mathrm{o}}(\mathcal{G}) : \delta_D(m^D_{\mathrm{init}}, \mu_1) = \delta_D(m^D_{\mathrm{init}}, \mu_2) \\ \Rightarrow \exists k \in [0,K] : \mu_1, \mu_2 \in \mathrm{leak}(\mathcal{G}, P_{\Sigma_\mathrm{o}}, S, \mathsf{OP}_\mathsf{K}, k) \\ \vee \mu_1, \mu_2 \notin \mathrm{leak}(\mathcal{G}, P_{\Sigma_\mathrm{o}}, S, \mathsf{OP}_\mathsf{K}).\end{aligned}$$

For a given state in the trajectory estimator, all traces ending in this state reveal or preserve opacity in the same way. Thus, in a K-delay trajectory estimator $D = (M^D, m^D_{\mathrm{init}}, \Sigma_\mathrm{o}, \delta_D)$, to each state $m \in M^D$, we can associate the delay to hold (i.e., after which it is safe to "release") the last received event of the trace leading to this state in order to preserve $\mathsf{OP}_\mathsf{K} \in \{\mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}\}$:

$$\forall m \in M^D : \mathrm{hold}_{\mathsf{OP}_\mathsf{K}}(m) \stackrel{\text{def}}{=} \begin{cases} K+1-k & \text{when (7)} \\ 0 & \text{otherwise (i.e., when (8))} \end{cases}$$

with:

$$\exists k \in [0,K] : \forall \mu \in \mathcal{T}_{\Sigma_\mathrm{o}}(\mathcal{G}) : \delta_D(m^D_{\mathrm{init}}, \mu) = m \Rightarrow \mu \in \mathrm{leak}(\mathcal{G}, P_{\Sigma_\mathrm{o}}, S, \mathsf{OP}_\mathsf{K}, k) \quad (7)$$

$$\forall \mu \in \mathcal{T}_{\Sigma_\mathrm{o}}(\mathcal{G}) : \delta_D(m^D_{\mathrm{init}}, \mu) = m \Rightarrow \mu \notin \mathrm{leak}(\mathcal{G}, P_{\Sigma_\mathrm{o}}, S, \mathsf{OP}_\mathsf{K}) \quad (8)$$

Equivalently, using an R-Verifier for OP_K, $\mathcal{V} = (Q^\mathcal{V}, q^\mathcal{V}_{\mathrm{init}}, \Sigma_\mathrm{o}, \delta_\mathcal{V}, \mathbb{D}^K_{\mathsf{OP}}, \Gamma^\mathcal{V})$ for $\mathcal{G}$ and K-step based opacity, and synthesized from D (thus $M^D = Q^\mathcal{V}$ and $q^D_{\mathrm{init}} = q^\mathcal{V}_{\mathrm{init}}$), $\forall \mu \in \mathcal{T}_{\Sigma_\mathrm{o}}(\mathcal{G}) : \mathrm{hold}_{\mathsf{OP}_\mathsf{K}}(\delta_D(m^D_{\mathrm{init}}, \mu)) = K+1-k$ when $\Gamma^\mathcal{V}(\delta_\mathcal{V}(q^\mathcal{V}_{\mathrm{init}}, \mu)) = \mathrm{leak}_k$. Thus, synthesis of R-Enforcers will rely on the synthesis of R-Verifiers.

Proposition 6 (Enforcement criterion using K-delay trajectory estimators) Let $D = (M^D, m^D_{\mathrm{init}}, \Sigma_\mathrm{o}, \delta_D)$ be the K-delay trajectory estimator associated to $\mathcal{G}$. The K-step based opacity OP_K of the secret S is enforceable by an R-Enforcer with a memory of size T iff:

$$\max\{\mathrm{hold}_{\mathsf{OP}_\mathsf{K}}(m) \mid m \in M^D\} \leq T \quad (9)$$

The proof of this proposition is given in Appendix A.3.1. Consequently, enforcement of a K-step based opacity with a memory of a given size is decidable.

6.4 Synthesis of runtime enforcers

We now address the synthesis of runtime enforcers for opacity. We first start by defining the enforcement primitives endowed to the enforcers we consider.

Enforcement operations. We define enforcement operations specifically used by our R-Enforcers dedicated to opacity. First, we need to define some auxiliary operations. In the following, we will use the following notations for the R-Enforcers' memory. For two memory configurations c and c' s.t. $c = (\sigma_1, d_1)\cdots(\sigma_t, d_t)$, $c' = (\sigma_1, d'_1)\cdots(\sigma_{t'}, d'_{t'})$ with $t' \leq t$:

- $c_{\downarrow\Sigma_o} \stackrel{\text{def}}{=} \sigma_1 \cdots \sigma_t$,
- $(c \setminus c')_{\downarrow\Sigma_o}$ is ϵ_{Σ_o} if $c = c'$ and the sequence of events $\sigma_{t'+1}\cdots\sigma_t$ otherwise.

Definition 14 (Auxiliary operations) *For a memory $\mathcal{M}$ of size T, given $t \leq T$ and $c = (\sigma_1, d_1)\cdots(\sigma_t, d_t) \in \mathcal{M}(T)$,* free *and* delay *$(\mathcal{M}(T) \to \mathcal{M}(T))$ are defined as follows:*

- $\text{delay}(c) \stackrel{\text{def}}{=} (\sigma_1, d_1 - 1)\cdots(\sigma_t, d_t - 1)$*, with $t \leq T$;*
- $\text{free}(c) \stackrel{\text{def}}{=} (\sigma_i, d_i)\cdots(\sigma_t, d_t)$*, with $1 \leq i \leq t$ and*

$$\forall j \in [1, i-1] : d_j \leq 0 \ \wedge \ d_i > 0.$$

The operation delay consists in decrementing the delay of each element inside the memory. Intuitively, this operation is used when one step has been performed on the system, and thus the stored events revealing the opacity have to be retained for one unit of time less. The operation free consists in outputting the events that currently do not leak opacity, i.e., the events with a negative or null delay, starting from the first event in memory and until the first event with a positive delay.

The following operations are those actually used by the runtime enforcers.

Definition 15 (Enforcement Operations) *The actual enforcement operations are defined as follows where $\sigma \in \Sigma_o$, $c = (\sigma_1, d_1)\cdots(\sigma_t, d_t) \in \mathcal{M}(T)$.*

- $\text{store}_d(\sigma, c) \stackrel{\text{def}}{=} (o, c' \cdot (\sigma, d))$*, with $c' = \text{free} \circ \text{delay}(c), o = (c \setminus c')_{\downarrow\Sigma_o}$;*
- $\text{dump}(\sigma, c) \stackrel{\text{def}}{=} (o, c'')$ *with*
  - $c' \stackrel{\text{def}}{=} \text{free} \circ \text{delay}(c)$,
  - $o \stackrel{\text{def}}{=} c_{\downarrow\Sigma_o} \cdot \sigma$ *if $c' = \epsilon_{\mathcal{M}}$ and $(c \setminus c')_{\downarrow\Sigma_o}$ otherwise,*
  - $c'' \stackrel{\text{def}}{=} c' \cdot (\sigma, 0)$ *if $c' \neq \epsilon_{\mathcal{M}}$ and $\epsilon_{\mathcal{M}}$ otherwise;*
- $\text{off}(\sigma, c) \stackrel{\text{def}}{=} \text{dump}(\sigma, c)$*;*
- $\text{halt}(\sigma, c) \stackrel{\text{def}}{=} (\epsilon_{\Sigma_o}, \epsilon_{\mathcal{M}})$.

For $d \in [1, K]$, the store_d operation is issued when the event submitted to the R-Enforcer should be delayed by d unit(s) of time in order to preserve opacity. This operation consists in first releasing the events preserving the opacity (using $\text{free} \circ \text{delay}$) and appending the event with the needed delay to the memory. The dump operation is issued when the submitted event does

not reveal the opacity. The event is submitted but not necessarily produced in output. The R-Enforcer first releases the events preserving the opacity. After this step, if the memory is empty, then the event is appended to the output sequence. Otherwise, the event is appended in the memory with delay 0 so as to first be released in the future and preserve the order of the input trace. The off operation is issued by an R-Enforcer when the opacity will not be revealed whatever are the future observable events produced by the system. Thus, the R-Enforcer can be switched off. Though the off has the same definition as the dump operation, such an enforcement operation is useful in practice since it reduces the overhead induced by the R-Enforcer. The halt operation is issued when the considered notion of opacity is irremediably revealed. This operation consists in ignoring the submitted event, erasing the memory, and stopping the underlying system.

Synthesis of R-Enforcers. We propose now to address the synthesis of R-Enforcers relying on K-delay trajectory estimators.

For simple opacity the synthesis is achieved by the construction proposed in Proposition 7. In the states of the underlying 1-delay trajectory estimator, only the information about traversed states is considered for the simplicity of notations by forgetting the Boolean information.

Proposition 7 (Synthesis of R-Enforcers for simple opacity) Given $\mathcal{G}$, S, and $\Sigma_o \subseteq \Sigma$, the R-Enforcer $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, \{\text{store}_1, \text{dump}, \text{off}\}, \Gamma^{\mathcal{E}}, \mathcal{M}(T))$ built from the 1-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ where:

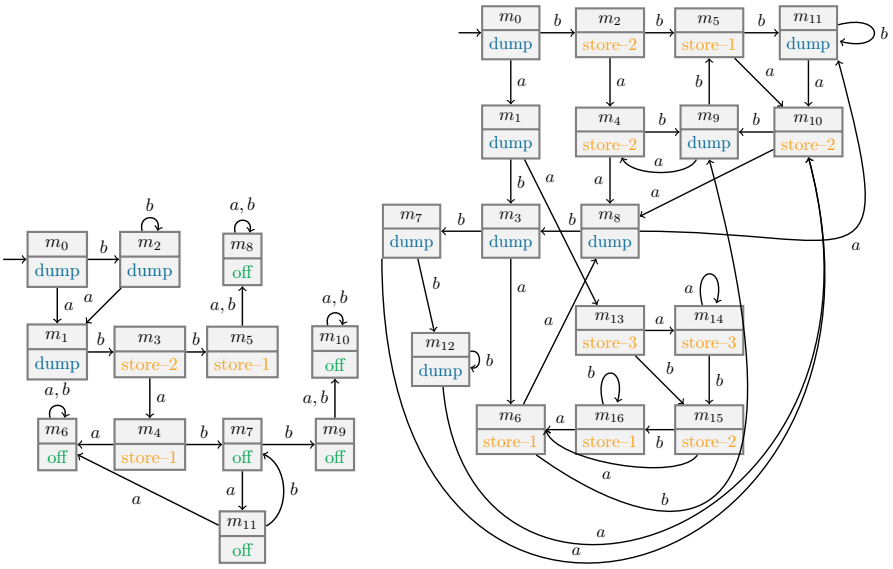
- $Q^{\mathcal{E}} = M^D, q^{\mathcal{E}}_{\text{init}} = m^D_{\text{init}}, \delta_{\mathcal{E}} = \delta_D$,
- $\Gamma^{\mathcal{E}} : Q^{\mathcal{E}} \to \{\text{store}_1, \text{dump}, \text{off}\}$ defined by
  - $\Gamma^{\mathcal{E}}(m) = \text{store}_1$ if $m(0) \in 2^S$,
  - $\Gamma^{\mathcal{E}}(m) = \text{off}$ if $m(0) \notin 2^S \wedge \forall m' \in \Delta_D(\{m\}, \Sigma_o^*) : m'(0) \notin 2^S$,
  - $\Gamma^{\mathcal{E}}(m) = \text{dump}$ if $m(0) \notin 2^S \wedge \exists m' \in \Delta_D(\{m\}, \Sigma_o^*) : m'(0) \in 2^S$,

enforces the simple opacity of S w.r.t. P_{Σ_o} on $\mathcal{G}$.

This proposition is proven correct in Appendix A.3.2. An R-Enforcer built following this construction processes an observation trace of the underlying system and enforces the simple opacity of the secret. It switches off when the current read observation trace and all its possible continuations on the system do not leak the simple opacity of the current secret. It dumps the event of the last observed trace when this trace does not leak the opacity but there is a possibility that the secret may simply leak in the future. It stores the last event of the observed trace in memory for one unit of time when this trace leaks the simple opacity of the secret.

Synthesis of R-Enforcers for K-step weak and strong opacity is achieved using K-delay trajectory estimators and the function $\text{hold}_{\mathsf{OP_K}}$ to uniformly address weak and strong versions of opacity.

Proposition 8 (Synthesis of R-Enforcers for K-step opacity) Given $\mathcal{G}$, S and $\Sigma_o \subseteq \Sigma$, the R-Enforcer $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, \{\text{halt}, \text{store}_d, \text{dump}, \text{off} \mid$



(a) For the 1-weak opacity of $\mathcal{G}_1$ (b) R-Enforcer of 2-strong opacity of $\mathcal{G}_5$

Fig. 10 R-Enforcers for K-step opacity

$d \in [1,T]\}, \Gamma^{\mathcal{E}}, \mathcal{M}(T))$, built from the K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ where $Q^{\mathcal{E}} = M^D, q^{\mathcal{E}}_{\text{init}} = m^D_{\text{init}}, \delta_{\mathcal{E}} = \delta_D$, and $\Gamma^{\mathcal{E}} : Q^{\mathcal{E}} \to \{\text{off}, \text{dump}, \text{store}_d, \text{halt} \mid d \in [1,T]\}$ defined by:

- $\Gamma^{\mathcal{E}}(m) = \text{off}$ if $\text{hold}_{\mathsf{OP_K}}(m) = 0 \wedge \forall m' \in \Delta_D(\{m\}, \Sigma_o^*) : \text{hold}_{\mathsf{OP_K}}(m') = 0$,
- $\Gamma^{\mathcal{E}}(m) = \text{dump}$ if $\text{hold}_{\mathsf{OP_K}}(m) = 0 \wedge \exists m' \in \Delta_D(\{m\}, \Sigma_o^*) : \text{hold}_{\mathsf{OP_K}}(m') \neq 0$,
- $\Gamma^{\mathcal{E}}(m) = \text{store}_d$ if $\exists d \in [1,T] : \text{hold}_{\mathsf{OP_K}}(m) = d$,
- $\Gamma^{\mathcal{E}}(m) = \text{halt}$ if $\exists d > T : \text{hold}_{\mathsf{OP_K}}(m) = d$.

enforces the K-step opacity $\mathsf{OP_K} \in \{\mathsf{OP^W_K}, \mathsf{OP^S_K}\}$ of S under P_{Σ_o} on $\mathcal{G}$.

This proposition is proven correct in Appendix A.3.3. The R-Enforcers built from this construction enforce the K-step based opacity which is weak or strong depending on the used function $\text{hold}_{\mathsf{OP_K}}$. They operate a dump or an off enforcement operation relatively to K-step opacity in a similar fashion to R-Enforcers for simple opacity. They store the last event of the current observation trace for d unit(s) of time, with $d \leq T$, when the current sequence leads the K-step opacity to be revealed $K+1-d$ step(s) ago ($\text{hold}_{\mathsf{OP_K}}(m) = d$). Consequently, when the attacker will observe this event, this will reveal the opacity of the secret at stricly more than K steps. When the current sequence leaks the K opacity of the secret d steps ago with $d > T$, the R-Enforcer halts the underlying system since the last event of this sequence cannot be memorized with the allocated memory of size T.

Example 12 (R-Enforcers of K-step opacity) *In Fig. 10a is represented the R-Enforcer of $\mathcal{G}_1$ for 1-step weak and strong opacity. In Fig. 10b is repre-*

sented the R-Enforcer of K-strong opacity for $\mathcal{G}_5$. These R-Enforcers are built from their K-delay trajectory estimators and with a sufficient memory size, i.e., in both cases $T > K$ for K-step opacity.

For instance, let us consider the observation trace $a \cdot a \cdot b$ on $\mathcal{G}_5$ and the enforcement of opacity on it. When the R-Enforcer reads a, this trace is safe regarding the considered opacity. The R-Enforcer goes to state m_1 and produces the operation dump*, i.e., the event is directly produced in output and visible to the attacker. Now, when the system produces the next a, it is read by the R-Enforcer, which ends in m_{13} and produces the enforcement operation* store_3*. That is, the last a will be retained in memory for 3 units of time. Indeed, the sequence $a \cdot a \in \text{leak}(\mathcal{G}_5, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}, 0)$. Then, when the system produces a subsequent b, this event will be retained in memory for 2 units of time. Note that at this moment, the previous a has now to be retained for 2 units of time. These two events will be released from the memory to the attacker at the same moment and in the order they were produced on the system. For instance, if the system produces the subsequent partial observation trace $b \cdot a$, we have $\text{safe}_{\mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}}(a \cdot a \cdot b \cdot b \cdot a, a \cdot a)$ and $\text{safe}_{\mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}}(a \cdot a \cdot b \cdot b \cdot a, a \cdot a \cdot b)$.*

Remark 7 (R-Enforcer optimization) *In R-Enforcers, we can reduce the states in which the* off *operation is produced, into a unique state. This is a straightforward adaptation of the transformation that is not modifying their correctness.*

Remark 8 (Delaying events and deadlocks) *Given some* enforceable *K-step opacity notion (i.e., with a with a sufficient memory), no deadlock can be introduced as we do not consider time. In the worst case scenario an enforcement monitor for K-step opacity delays the sequence by $K + 1$ time units.*

7 Related work

In this section, we propose a comparison with related work.

7.1 Model-checking of K-weak opacity

In Saboori and Hadjicostis (2011), the authors addressed the model-checking of K-weak opacity, using a mechanism similar to K-delay trajectory estimators. Expressing the authors' result in the context of K-delay trajectory estimators in which the Boolean information is omitted, a system $\mathcal{G}$ a secret $S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if and only if there does not exist a state m reachable in D, the K-delay trajectory estimator of $\mathcal{G}$, such that $\exists k \in [0, K] : m(k) \in 2^S$.

7.2 Using runtime techniques

Validating simple opacity via testing. In Marchand et al (2009), the authors are interested in testing the simple opacity of a system. In the context of

ensuring opacity via access control mechanisms, the authors show how to extend the classical theory of conformance testing in order to derive tests that detect violation of the conformance by an access control implementation to its specification. Thus, the authors study another kind of runtime-based technique, namely testing, in order to validate opacity. The authors only address the current opacity of the secret modelled by a finitary language. Validation of the K-step based opacity (weak and strong) through testing remains to be studied.

Runtime verification and runtime enforcement for linear-time properties. There exists a large amount of runtime verification and runtime enforcement frameworks designed for linear-time properties (see Havelund and Goldberg (2008); Leucker and Schallhart (2008); Falcone (2010) for short surveys). The major part of the research endeavor in runtime verification was done on the monitoring of safety properties (stating that something bad should not happen), as seen for example in Havelund and Rosu (2002). Runtime enforcement was initiated by the work of Schneider (2000) on what has been called *security automata*; i.e., monitors watching the execution and halting the program whenever it deviates from the desired property. Later, Ligatti et al (2009) proposed a more powerful enforcement mechanism called *edit-automata*. This mechanism featured the idea of "suppressing" (i.e., freezing) and "inserting" (frozen) actions in the current execution of a system.

To the best of the authors' knowledge, the only runtime-based approach to validate opacity was the one proposed in Dubreil et al (2009). The authors propose a runtime verification (monitoring) approach of simple opacity. Thus, this article is the first to address runtime verification for K-step based notions of opacity and to introduce runtime enforcement as a validation technique for opacity. Note that the notion of runtime enforcer proposed in this paper is inspired from and extends the variant used in Falcone et al (2008, 2009a) (and summarized in Falcone et al (2011)) where runtime enforcers were defined in order to enforce linear temporal properties.

7.3 Comparison with supervisory control

In this particular setting used to ensure opacity at runtime, thanks to the halt operation, runtime enforcement is similar to supervisory control. Indeed, blocking the system or letting its execution going through are the only primitives endowed to controllers. The difference between supervisory control and runtime enforcement is as follows. In supervisory control, the underlying system is put in parallel with a controller. When a controlled system tries to perform an illegal action, this action is disabled by the supervisory controller. In runtime enforcement, actions of the systems are directly fed to the enforcer, that delays or suppresses illegal actions. Illegal actions are thus actually executed in the system but not produced in output. In some sense, the effect of illegal actions is not visible from the outside. This illustrates that enforcement

monitoring is particularly appropriate to ensure a desired behavior on the system outputs, while supervisory control is particularly appropriate to ensure a desired behavior on the internal behavior of a system. Finally, note that, the system to be considered in a runtime enforcement approach is the initial system along with its runtime enforcer. In supervisory control, it is the product between the initial system and the controller which is to be considered.

8 Conclusion and future work

Conclusion. In this paper we are interested in the use of runtime techniques so as to ensure several levels of opacity. Proposed runtime techniques are complementary to supervisory control, which is usually used to validate opacity on systems. We take into account two existing levels of opacity (simple and K-weak), and introduce K-strong opacity circumventing some limitations of the opacity notions proposed so far. With runtime verification, we are able to detect leakages for the various levels of opacity. With runtime enforcement, opacity leakages are prevented, and this technique guarantees opacity preservation for the system of interest.

The techniques proposed in this paper have several advantages compared to existing validation approaches for opacity. With the aim of ensuring the opacity of system, runtime enforcement is a non intrusive technique that is not damaging the internal nor the observable behavior of the underlying system.

All results of this paper are implemented in a prototype toolbox, named TAKOS: a Java Toolbox for the Analysis of K-Opacity of Systems, which is freely available at the following address:

`http://toolboxopacity.gforge.inria.fr`.

TAKOS is able to model-check, synthesize R-Verifiers and R-Enforcers for the notions of opacity considered in this article. A complete description along with a user manual and usage of TAKOS on various examples are available in Falcone and Marchand (2010a).

Future work. Other kinds of opacity conditions might be interesting to handle in this framework like *initial opacity* (proposed in Saboori and Hadjicostis (2013)) or *infinite-step opacity* (proposed in Saboori and Hadjicostis (2009)). This would necessitate to introduce new kinds of trajectory estimators, as shown in Saboori and Hadjicostis (2009, 2013) for verification purposes.

As the proposed runtime techniques are complementary to supervisory control, we plan to study how we can combine those techniques to obtain the best of both worlds. For instance, when runtime enforcement with a given memory size is not possible, one may be interested in synthesizing controllers in order to restrict the system so as to ensure the existence of runtime enforcers.

Another current working direction is to address two practical implementation issues. The first one is the retrieval of a suitable model of the analyzed system from either its source or binary code, e.g., an LTS as considered in this

article, in order to apply the proposed techniques. The second one, is about integrating the synthesized high-level description in concrete programs. That will entail us to design a translation of R-Verifiers and R-Enforcers in the target programming language.

Acknowledgements The authors would like to gratefully thank the anonymous reviewers for their helpful remarks.

References

Alur R, Zdancewic S (2006) Preserving secrecy under refinement. In: Proc. of the 33rd Internat. Colloq. on Automata, Languages and Programming (ICALP 06), volume 4052 of Lecture Notes in Computer Science, Springer-Verlag, pp 107–118

Badouel E, Bednarczyk M, Borzyszkowski A, Caillaud B, Darondeau P (2007) Concurrent secrets. Discrete Event Dynamic Systems 17(4):425–446, DOI http://dx.doi.org/10.1007/s10626-007-0020-5

Bryans J, Koutny M, Mazaré L, Ryan PYA (2008) Opacity generalised to transition systems. International Journal of Information Security 7(6):421–435

Cassandras CG, Lafortune S (2006) Introduction to Discrete Event Systems. Springer-Verlag, Secaucus, NJ, USA

Cassez F, Dubreil J, Marchand H (2009) Dynamic observers for the synthesis of opaque systems. In: ATVA'09: 7^{th} International Symposium on Automated Technology for Verification and Analysis, pp 352–367

Dubreil J (2009) Monitoring and supervisory control for opacity properties. PhD thesis, Université de Rennes 1

Dubreil J, Jéron T, Marchand H (2009) Monitoring confidentiality by diagnosis techniques. In: European Control Conference, Budapest, Hungary, pp 2584–2590

Dubreil J, Darondeau P, Marchand H (2010) Supervisory control for opacity. IEEE Transactions on Automatic Control 55(5):1089–1100

Falcone Y (2010) You should better enforce than verify. In: Barringer H, Falcone Y, Finkbeiner B, Havelund K, Lee I, Pace GJ, Rosu G, Sokolsky O, Tillmann N (eds) RV, Springer, Lecture Notes in Computer Science, vol 6418, pp 89–105

Falcone Y, Marchand H (2010a) TAKOS: a Java Toolbox for the Analysis of K-Opacity of Systems. Available at http://toolboxopacity.gforge.inria.fr

Falcone Y, Marchand H (2010b) Various notions of opacity verified and enforced at runtime. Tech. Rep. 7349, INRIA

Falcone Y, Marchand H (2013) Runtime enforcement of k-step opacity. In: Proceedings of the 52nd Conference on Decision and Control, IEEE

Falcone Y, Fernandez JC, Mounier L (2008) Synthesizing enforcement monitors wrt. the safety-progress classification of properties. In: Sekar R, Pujari

AK (eds) ICISS, Springer, Lecture Notes in Computer Science, vol 5352, pp 41–55

Falcone Y, Fernandez JC, Mounier L (2009a) Enforcement monitoring wrt. the safety-progress classification of properties. In: SAC'09: Proceedings of the 2009 ACM symposium on Applied Computing, ACM, pp 593–600

Falcone Y, Fernandez JC, Mounier L (2009b) Runtime verification of safety-progress properties. In: RV'09: Proceedings of the 9^{th} Workshop on Runtime Verification. Revised selected Papers, pp 40–59

Falcone Y, Mounier L, Fernandez JC, Richier JL (2011) Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods in System Design 38(3):223–262

Falcone Y, Fernandez JC, Mounier L (2012) What can you verify and enforce at runtime? STTT 14(3):349–382

Hamlen KW, Morrisett G, Schneider FB (2006) Computability classes for enforcement mechanisms. ACM Trans Programming Lang and Syst 28(1):175–205, DOI http://doi.acm.org/10.1145/1111596.1111601

Havelund K, Goldberg A (2008) Verify your runs. In: VSTTE'05: Verified Software: Theories, Tools, Experiments: First IFIP TC 2/WG 2.3 Conference, Revised Selected Papers and Discussions, pp 374–383

Havelund K, Rosu G (2002) Efficient monitoring of safety properties. Software Tools and Technology Transfer

Leucker M, Schallhart C (2008) A brief account of runtime verification. Journal of Logic and Algebraic Programming 78(5):293–303

Ligatti J, Bauer L, Walker D (2005) Enforcing Non-safety Security Policies with Program Monitors. In: ESORICS, pp 355–373

Ligatti J, Bauer L, Walker D (2009) Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security 12(3):1–41, URL `http://doi.acm.org/10.1145/1455526.1455532`

Marchand H, Dubreil J, Jéron T (2009) Automatic testing of access control for security properties. In: TestCom'09, Springer-Verlag, LNCS, vol 5826, pp 113–128

Pnueli A, Zaks A (2006) PSL model checking and run-time verification via testers. In: FM'06: Proceedings of Formal Methods, pp 573–586

Saboori A, Hadjicostis CN (2007) Notions of security and opacity in discrete event systems. In: CDC'07: 46^{th} IEEE Conf. Decision and Control, pp 5056–5061

Saboori A, Hadjicostis CN (2009) Verification of infinite-step opacity and analysis of its complexity. In: Dependable Control of Discrete Systems

Saboori A, Hadjicostis CN (2011) Verification of k-step opacity and analysis of its complexity. IEEE Trans Automation Science and Engineering 8(3):549–559

Saboori A, Hadjicostis CN (2012) Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Trans Automat Contr 57(5):1155–1165

Saboori A, Hadjicostis CN (2013) Verification of initial-state opacity in security applications of discrete event systems. Inf Sci 246:115–132

Schneider FB (2000) Enforceable security policies. ACM Transaction of Information System Security 3(1):30–50

Takai S, Kumar R (2009) Verification and synthesis for secrecy in discrete-event systems. In: ACC'09: Proceedings of the 2009 conference on American Control Conference, IEEE Press, Piscataway, NJ, USA, pp 4741–4746

Takai S, Oka Y (2008) A formula for the supremal controllable and opaque sublanguage arising in supervisory control. SICE Journal of Control, Measurement, and System Integration 1(4):307–312

Wu Y, Lafortune S (2012) Enforcement of opacity properties using insertion functions. In: 51st IEEE Conf. on Decision and Contr., pp 6722–6728

A Proofs

A.1 Proofs of Section 4

Proposition 1 (p. 8). We shall prove that a secret $S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if and only if

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall \mu' \preceq \mu : |\mu - \mu'| \leq K \Rightarrow [\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G}).$$

Proof We prove the equivalence by showing the implication in both ways.

($\Rightarrow$) Let $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$, $\mu' \preceq \mu$ such that $|\mu - \mu'| \leq K$. Let $t \in [\![\mu]\!]_{\Sigma_o}$ and $t' \preceq t$ such that $t' \in [\![\mu']\!]_{\Sigma_o}$. Then, we have $|t - t'|_{\Sigma_o} \leq K$ and $t' \in [\![\mu'/\mu]\!]_{\Sigma_o}$. If $t' \notin \mathcal{L}_S(\mathcal{G})$, then $[\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. Otherwise $t' \in \mathcal{L}_S(\mathcal{G})$ and as S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque, there exist $s, s' \in \mathcal{L}(\mathcal{G})$ such that $s \approx_{\Sigma_o} t$, $s' \approx_{\Sigma_o} t'$, and $s' \notin \mathcal{L}_S(\mathcal{G})$. We thus have that $s' \in [\![\mu'/\mu]\!]_{\Sigma_o}$ and finally $[\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$.

($\Leftarrow$) Reciprocally, let $t \in \mathcal{L}(\mathcal{G})$, $t' \preceq t$ such that $|t - t'|_{\Sigma_o} \leq K$ and $t' \in \mathcal{L}_S(\mathcal{G})$. Let $\mu = P_{\Sigma_o}(t)$, $\mu' = P_{\Sigma_o}(t')$. By definition, we have $|\mu - \mu'| \leq K$. Now, by hypothesis we know that $[\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. So there exist $s \in [\![\mu]\!]$, $s' \preceq s$, such that $s' \in [\![\mu'/\mu]\!]_{\Sigma_o}$ and $s' \notin \mathcal{L}_S(\mathcal{G})$. Finally, we have found $s \in \mathcal{L}(\mathcal{G})$, $s' \preceq s$ such that $s \approx_{\Sigma_o} t \wedge s' \approx_{\Sigma_o} t' \wedge s' \notin \mathcal{L}_S(\mathcal{G})$. Thus, S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque. □

Proposition 2 (p. 10). We shall prove that, on $\mathcal{G}$, $S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque if and only if

$$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : [\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_K^S(\mathcal{G}) \neq \emptyset.$$

Proof We prove the equivalence by showing the implication in both ways.

($\Leftarrow$) Let $t \in \mathcal{L}(\mathcal{G})$ and $\mu = P_{\Sigma_o}(t)$, by hypothesis we have $[\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_K^S(\mathcal{G}) \neq \emptyset$. In particular, there exists $s \in [\![\mu]\!]_{\Sigma_o}$ (thus $s \approx_{\Sigma_o} t$) such that $\forall s' \preceq s : |s - s'|_{\Sigma_o} \leq K \wedge s' \notin \mathcal{L}_S(\mathcal{G})$, which entails that S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque.

($\Rightarrow$) Let $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ and $t \in [\![\mu]\!]_{\Sigma_o}$. As S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque, there exists $s \in \mathcal{L}(\mathcal{G})$ such that $s \approx_{\Sigma_o} t$ and $\forall s' \preceq s : |s - s'|_{\Sigma_o} \leq K \Rightarrow s' \notin \mathcal{L}_S(\mathcal{G})$. Obviously $s \in \mathrm{Free}_K^S(\mathcal{G})$, and, as $s \in [\![\mu]\!]_{\Sigma_o}$, $[\![\mu]\!]_{\Sigma_o} \cap \mathrm{Free}_K^S(\mathcal{G}) \neq \emptyset$. □

A.2 Proofs of Section 5

In the two following lemmas, let us recall that only the information about traversed states is considered.

A.2.1 Lemma 1 (p. 14)

Given a system $\mathcal{G}$ modeled by an LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay trajectory estimator $D = (M^D, q^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$, then

- $\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$ s.t. $|\mu| \geq K \wedge \mu = \mu' \cdot \sigma_1 \cdots \sigma_K$,
  $\forall s \in [\![\mu]\!]_{\Sigma_{\text{o}}} : (s = s' \cdot s_1 \cdots s_K \wedge \forall i \in [1, K] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i)$
  $\exists (q_0, \ldots, q_K) \in \delta_D(m^D_{\text{init}}, \mu) : q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$,

- $\forall \mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$ s.t. $n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n$,
  $\forall s \in [\![\mu]\!]_{\Sigma_{\text{o}}} : (s = s' \cdot s_1 \cdots s_n \wedge \forall i \in [1, n] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i)$,
  $\exists (q_0, \ldots, q_K) \in \delta_D(m^D_{\text{init}}, \mu) : q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots \xrightarrow{s_{n-1}}_{\mathcal{G}} q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K$,
  $\wedge\ q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n}$.

Proof The proof is done by induction on $|\mu|$. We consider only observation traces μ of length $|\mu| \geq 1$ since for the empty observation trace, the information provided by a K-delay trajectory estimator reduces to the state estimate reachable with unobservable events from the system. This information is given in the initial state of the trajectory estimator and is trivially correct by definition.

- For $|\mu| = 1$, $\mu = \sigma_1$, by definition $m^D_{\text{init}} = \odot_{K+1} \Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_{\text{o}}})$. In particular $(q, \ldots, q) \in m^D_{\text{init}}$ for every $q \in \Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_{\text{o}}})$. Let $s' \cdot s_1 \in [\![\sigma_1]\!]_{\Sigma_{\text{o}}}$ (with $P_{\Sigma_{\text{o}}}(s') = \epsilon$ and $P_{\Sigma_{\text{o}}}(s_1) = \sigma_1$) such that $q \xrightarrow{s_1}_{\mathcal{G}} q_1$ for some $q \in \Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_{\text{o}}})$. By definition $(q, q_1) \in \text{Obs}(\sigma_1)$ and thus $(q, \ldots, q, q_1) \in \delta_D(m^D_{\text{init}}, \sigma_1)$.
- Assume now that the property holds for any trace of $\mathcal{G}$ of length strictly less than K. Let $\mu \in \mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G})$ s.t. $n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n$ and $s \in [\![\mu]\!]_{\Sigma_{\text{o}}}$ s.t. $s = s' \cdot s_1 \cdots s_n \wedge \forall i \in [1, n] : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i \wedge P_{\Sigma_{\text{o}}}(s') = \epsilon$. We have $s' \cdot s_1 \cdots s_{n-1} \in [\![\mu']\!]_{\Sigma_{\text{o}}}$ with $\mu' = \sigma_1 \cdots \sigma_{n-1}$. By induction hypothesis, $\exists (q, q_0, \ldots, q_{K-1}) \in m' = \delta_D(m^D_{\text{init}}, \mu') : q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots \xrightarrow{s_{n-2}}_{\mathcal{G}} q_{K-2} \xrightarrow{s_{n-1}}_{\mathcal{G}} q_{K-1}$. Now, consider $m = m' \| \text{Obs}(\sigma_n)$. As $s \in [\![\mu]\!]_{\Sigma_{\text{o}}}$, we get that there exists $q_K \in Q^{\mathcal{G}}$ such that $q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots \xrightarrow{s_{n-2}}_{\mathcal{G}} q_{K-2} \xrightarrow{s_{n-1}}_{\mathcal{G}} q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K$ with $P_{\Sigma_{\text{o}}}(s_n) = \sigma_n$. Now by definition of the function Obs, we have $(q_{K-1}, q_K) \in \text{Obs}(\sigma_n)$ and finally by definition of $\|$, we have $(q_0, \ldots, q_n) \in m$.
- The case where $|\mu| \geq K$ follows exactly the same pattern. □

A.2.2 Lemma 2 (p. 15)

Given a system $\mathcal{G}$ modelled by an LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$, then $\forall m \in M^D$,

$\forall (q_0, \ldots, q_K) \in m, \forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m,$

- $|\mu| = 0 \Rightarrow \forall (q, \ldots, q) \in m = m^D_{\text{init}}, \exists s \in [\![\mu]\!]_{\Sigma_o} : q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}} q \wedge P_{\Sigma_o}(s) = \epsilon,$
- $n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n \Rightarrow$
  $\exists s' \cdot s_1 \cdots s_n \in [\![\mu]\!]_{\Sigma_o} :$
  $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K \wedge \forall i \in [1, n] : P_{\Sigma_o}(s_i) = \sigma_i$
- $|\mu| \geq K \wedge \mu = \mu' \cdot \sigma_1 \cdots \sigma_K \Rightarrow$
  $\exists s' \cdot s_1 \cdots s_K \in [\![\mu]\!]_{\Sigma_o} :$
  $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots \xrightarrow{s_K}_{\mathcal{G}} q_K \wedge \forall i \in [1, K] : P_{\Sigma_o}(s_i) = \sigma_i.$

Proof Let us consider $m \in M^D$, and $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m$. The proof is done by induction on $|\sigma|$.

- If $|\mu| = 0$, then $\mu = \epsilon_{\Sigma_o}$ and $m = m^D_{\text{init}} = \odot_{K+1}(\Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_o}))$. Then all state estimates $(q, \ldots, q) \in m^D_{\text{init}}$ are s.t. $q \in \Delta_{\mathcal{G}}(\{q^{\mathcal{G}}_{\text{init}}\}, [\![\epsilon]\!]_{\Sigma_o})$. Then, there exists $s \in L(\mathcal{G})$ s.t. $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}} q$ and $P_{\Sigma_o}(s) = \epsilon$.
- Assume that the property holds for all $\mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ such that $|\mu'| < n$ and consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ such that $|\mu| = n$
  - If $|\mu| = n < K$, μ can be written $\mu = \sigma_1 \cdots \sigma_n$. Consider now an element of m. It is of the form $(q_{K-n}, \ldots, q_{K-n}, q_{K-n+1}, \ldots, q_{K-1}, q_K)$. There exists $m_1 \in M^D$ s.t. $\delta_D(m_1, \sigma_n) = m = m_1 \| \operatorname{Obs}(\sigma_n)$ such that $(q_{K-n}, \ldots, q_{K-n}, q_{K-n+1}, \ldots, q_{K-1}) \in m_1$ and $(q_{K-1}, q_K) \in \operatorname{Obs}(\sigma_n)$. Let $\mu' = \sigma_1 \cdots \sigma_{n-1}$, by induction hypothesis on m_1 and μ', there exists $s'' = s' \cdot s_1 \cdots s_{n-1}$ with $P_{\Sigma_o}(s') = \epsilon$ and $\forall i \in [1, n-1] : P_{\Sigma_o}(s_i) = \sigma_i$ such that $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \xrightarrow{s_2}_{\mathcal{G}} \cdots \xrightarrow{s_{n-1}}_{\mathcal{G}} q_{K-1}$. Now by definition of Obs, there exists $s_n \in \Sigma^*$ with $P_{\Sigma_o}(s_n) = \sigma_n$ such that $q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K$. Finally $s = s'' \cdot s_n$ is such that $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K$ and $s \in [\![\mu]\!]_{\Sigma_o}$.
  - If $|\mu| \geq K$, μ can be written $\mu = \mu' \cdot \sigma_1 \cdots \sigma_K$. There exists $m_1 \in M^D$ s.t. $\delta_D(m_1, \sigma_K) = m = m_1 \| \operatorname{Obs}(\sigma_K)$. Furthermore, there exists $q \in Q^{\mathcal{G}}$ s.t. $(q, q_0, \ldots, q_{K-1}) \in m_1$ and $(q_{K-1}, q_K) \in \operatorname{Obs}(\sigma_K)$. By induction hypothesis applied on $(q, q_0, \ldots, q_{K-1}) \in m_1$, there exists $s" = s' \cdot s_0 \cdots s_{K-1} \in [\![\mu' \cdot \sigma_0 \cdots \sigma_{K-1}]\!]_{\Sigma_o}$ with $\forall i \in [0, K-1] : P_{\Sigma_o}(s_i) = \sigma_i$ such that $q \xrightarrow{s_0}_{\mathcal{G}} q_0 \xrightarrow{s_1}_{\mathcal{G}} \cdots \xrightarrow{s_{K-1}}_{\mathcal{G}} q_{K-1}$. Finally since $(q_{K-1}, q_K) \in \operatorname{Obs}(\sigma_K)$, there exists $s_K \in \Sigma^*$ with $P_{\Sigma_o}(s_K) = \sigma_K$ such that $q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$. Overall $s = s" \cdot s_K$ is such that $q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$ and $s \in [\![\mu]\!]_{\Sigma_o}$. □

A.2.3 Proposition 4 (p. 16)

We shall prove the soundness and completeness of the synthesized R-Verifiers for K-weak opacity, as exposed in Definition 6. That is, we prove that $\forall \mu \in$

$\mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall l \in [0, K]$:

$$\begin{array}{c} \Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_\text{l} \Leftrightarrow \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, l) \\ \wedge\ \Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{noleak} \Leftrightarrow \mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}). \end{array}$$

Proof We consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ s.t. $\delta_D(m^D_{\text{init}}, \mu) = m$.
($\Rightarrow$)

- If $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{noleak}$, we have $\Gamma^{\mathcal{V}}(m) = \text{noleak}$, that is $\forall i \in [0, K] : m(i) \notin 2^S$. Using Proposition 3, we have $\forall i \in [0, min\{|\mu|, K\}] : m(i) = \delta_{\mathcal{G}}(q^{\mathcal{G}}_{\text{init}}, [\![\mu^{\cdots|\mu|-i-1}/\mu]\!]_{\Sigma_o}) \notin 2^S$. That is, $\forall \mu' \preceq \mu : |\mu - \mu'|_{\Sigma_o} \leq K \Rightarrow [\![\mu/\mu']\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. According to Equation (1) (p. 9), it means that $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}})$.
- If $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_\text{l}$, we have $\Gamma^{D}(m) = \text{leak}_\text{l}$, that is $m(l) \in 2^S$ and $\forall i < l : m(i) \notin 2^S$. Similarly, using Proposition 3 we have $[\![\mu^{\cdots|\mu|-l-1}/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})$ and $\forall i < l : [\![\mu^{\cdots|\mu|-i-1}/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$, i.e., according to Equation (2) (p. 9) $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{W}_{K}, l)$.

($\Leftarrow$)

- If $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}})$, that is $\forall \mu' \preceq \mu : |\mu - \mu'| \leq K \Rightarrow [\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. Then using Proposition 3, we have $\forall i \in [0, min\{|\mu|, K\}] : m(i) \notin 2^S$. Following the definition of R-Verifiers construction, we have $\Gamma^{\mathcal{V}}(m) = \text{noleak}$.
- If $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{W}_{K}, l)$, then according to Equation (1), we have $[\![\mu^{\cdots|\mu|-l-1}/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})$ and $\forall i < l : [\![\mu^{\cdots|\mu|-i-1}/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. Now, using Proposition 3, $\forall i \in [0, l[: m(i) \notin 2^S$ and $m(l) \in 2^S$. According to the definition of the construction of R-Verifiers for K-weak opacity (definitions of $Q^{\mathcal{V}}$ and $\Gamma^{\mathcal{V}}$), we deduce that $\Gamma^{\mathcal{V}}(m) = \text{leak}_\text{l}$. □

A.2.4 Proposition 5 (p. 17)

We shall prove the soundness and completeness of the synthesized R-Verifiers for K-strong opacity, as exposed in Definition 6. That is, we prove that $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall l \in [0, K]$:

$$\begin{array}{c} \Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_\text{l} \Leftrightarrow \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}, l) \\ \wedge\ \Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{noleak} \Leftrightarrow \mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}). \end{array}$$

Proof We consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ s.t. $\delta_D(m^D_{\text{init}}, \mu) = m$.
($\Rightarrow$)

- If $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{noleak}$, i.e., $\exists((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m\downarrow$ s.t. $\forall i \in [0, K] : q_i \notin S$, and $\forall i \in [0, K-1] : b_i = \mathit{true}$. Let us consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$, s.t. $\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu) = m$. If $|\mu| \geq K$ and $\mu = \mu' \cdot \sigma_0 \cdots \sigma_{K-1}$, then according to Lemma 2, $\exists s = s' \cdot s_0 \cdots s_{K-1} \in [\![\mu]\!]_{\Sigma_o}$ with $\forall i \leq K-1 : P_{\Sigma_o}(s_i) = \sigma_i \wedge q_0 \xrightarrow{s_0}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_{K-1}}_{\mathcal{G}} q_K$. Moreover, as $\forall i \in [0, K-1] : b_i = \mathit{true}$, we can choose s such that $\forall i \in [0, K-1] : s_i \in \text{Free}^S_1(\mathcal{G}(q_{i-1}))$. Consequently $s_0 \cdots s_{K-1} \in \text{Free}^S_K(\mathcal{G}(q_0))$. Let s'' be the smallest prefix of

s' s.t. $|s''|_{\Sigma_o} = |s'|_{\Sigma_o}$. Necessarily we have $s' = s''$. Otherwise, because of the suitability of K-delay trajectory estimators, in m we would have $((q'_i)_{0\leq i\leq K}, (b'_i)_{0\leq i\leq K-1})$ where

$$\text{redundant}\left(((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}), ((q'_i)_{0\leq i\leq K}, (b'_i)_{0\leq i\leq K-1})\right)$$

would hold, which is a contradiction with $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m\downarrow$. Overall $s \in \text{Free}_K^S(\mathcal{G})$ and $s \in [\![\mu]\!]_{\Sigma_o} \cap \text{Free}_K^S(\mathcal{G})$ which means that $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S)$ (the case where $|\mu| < K$ is similar).

- If $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_l$ for some $l \in [1, K]$, i.e., $l = \min\{l' \in [1, K] \mid \forall((q_i)_{0\leq i\leq K},\ (b_i)_{0\leq i\leq K-1}) \in m, \exists i \leq l' : q_{K-i} \in S \vee b_{K-i} = \mathit{false}\}$. Let us suppose that $|\mu| \geq K$ (the case where $|\mu| < K$ is similar), $\mu = \mu' \cdot \sigma_0 \cdots \sigma_{K-1}$. Now, let us consider $s \in [\![\mu]\!]_{\Sigma_o}, s = s' \cdot s_0 \cdots s_{K-1}$ with $\forall i \leq K-1 : P_{\Sigma_o}(s_i) = \sigma_i$. By definition, there exists $(q_i)_{0\leq i\leq K}$ s.t. $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_0 \xrightarrow{s_0}_{\mathcal{G}} q_1 \cdots \xrightarrow{s_{K-1}}_{\mathcal{G}} q_K$ and according to Lemma 1, there exists $(b_i)_{0\leq i\leq K-1}$ s.t. $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m$. By hypothesis, there exists $i \leq l$ s.t. $q_{K-i} \in S$ or $b_{K-i} = \mathit{false}$.
  - If $q_{K-i} \in S$ then $s' \cdot s_0 \cdots s_{K-i-1} \in \mathcal{L}_S(\mathcal{G})$. Moreover, we have $|s - s' \cdot s_0 \cdots s_{K-i-1}|_{\Sigma_o} \leq l$, which gives us the expected result.
  - If $b_{K-i} = \mathit{false}$, meaning that $s_{K-i} \notin \text{Free}_1^S(\mathcal{G}(q_{K-i}))$, then there exists a prefix s'_{K-i} of s_{K-i} s.t. $s" = s' \cdot s_0 \cdots s'_{K-i} \in \mathcal{L}_S(\mathcal{G})$. Moreover, we have either $P_{\Sigma_o}(s'_{K-i}) = \sigma_{K-i}$ or $P_{\Sigma_o}(s'_{K-i}) = \epsilon$. In both cases, we have $|s - s"|_{\Sigma_o} \leq l$, which gives us again the expected result.

  Consider now $l' < l$, then $\exists((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m, \forall i \leq l' : q_{K-i} \notin S \wedge b_{K-i} = \mathit{true}$, which entails that all the sequences that match the elements of m belong to $\text{Free}_{l'}^S(\mathcal{G})$ and thus $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S, l)$.
- If $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \text{leak}_0$, then $\forall((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m, q_K \in S$, which entails that $[\![\mu]\!]_{\Sigma_o} \subset \mathcal{L}_S(\mathcal{G})$ and thus $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S, 0)$.

($\Leftarrow$)

- If $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S)$. It means that there exists $s \in [\![\mu]\!]_{\Sigma_o} \cap \text{Free}_K^S(\mathcal{G})$. Let $m = \delta_D(m^D_{\text{init}}, \mu)$. According to Lemma 1, there exist $s', s_1, \ldots, s_K \in \Sigma^*$ s.t.:
  - $s = s' \cdot s_1 \cdots s_K$,
  - $\forall i \leq K : P_{\Sigma_o}(s_i) = \sigma_i$,

  Each trajectory $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1})$ in $\delta_D(m^D_{\text{init}}, \mu)$ are s.t. $q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$. At least one trajectory in $\delta_D(m^D_{\text{init}}, \mu)$ is not redundant with the others. We have $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in \delta_D(m^D_{\text{init}}, \mu)\downarrow$. Let us note $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1})$ this trajectory. Now as $s \in \text{Free}_K^S(\mathcal{G})$, it is easy to see that $\forall i \in [0, K-1] : b_i = \mathit{true}$. Finally $\Gamma^{\mathcal{V}}(\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu)) = \Gamma^{\mathcal{V}}(m) = \text{noleak}$.
- If $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S, l)$ for some $l \in [1, K]$. By hypothesis, we have $[\![\mu]\!]_{\Sigma_o} \cap \text{Free}_l^S(\mathcal{G}) = \emptyset$ and $\forall l' < l : [\![\mu]\!]_{\Sigma_o} \cap \text{Free}_l^S(\mathcal{G}) \neq \emptyset$. Let $\delta_D(m^D_{\text{init}}, \mu) = m$ and $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m$. According to Lemma 2, $\exists s_1, \ldots, s_K \in \Sigma^* : s = s' \cdot s_1 \cdots s_K$ such that $\forall i \leq K : P_{\Sigma_o}(s_i) = \sigma_i$, $s \in [\![\mu]\!]_{\Sigma_o}$ and

$q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_K}_{\mathcal{G}} q_K$. As $s \notin \text{Free}_l^S(\mathcal{G})$, there exists $i \leq l$ such that either $q_{K-i} \in S$ or $s_{K-i} \notin \text{Free}_1^S(\mathcal{G}(q_{K-i}))$, which entails, by construction that $b_{K-i} = \mathit{false}$. Now for $l' < l$, there exists $s \in [\![\mu]\!]_{\Sigma_o} \cap \text{Free}_{l'}^S(\mathcal{G})$. To this s, we can associate an element $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m\downarrow$ (among the non-redundant trajectories of m) s.t. $\forall i \in [0, l']$: $q_{K-i} \notin S \wedge \forall i \in [1, l'] : b_{K-i} = \mathit{true}$, which entails that l is the smallest number s.t. $\forall((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m$, $\exists i \leq l : q_{K-i} \in S$ or $b_{K-i} = \mathit{false}$.

– If $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_K^S, 0)$, then by definition $[\![\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})$ and thus $\forall((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m : q_K \in S$, which concludes the proof. □

A.3 Proofs of Section 6

A.3.1 Proposition 6 (p. 27)

For a K-delay trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ associated to a system $\mathcal{G}$, we prove that the K-step based opacity $\mathsf{OP_K} \in \{\mathsf{OP_K^W}, \mathsf{OP_K^S}\}$ of the secret S is enforceable by an R-Enforcer with memory size T if and only if (9), i.e., if and only if

$$\max\{\text{hold}_{\mathsf{OP_K}}(m) \mid m \in M^D\} \leq T.$$

Proof This is a direct consequence of Proposition 3 and the definition of $\text{safe}_{\mathsf{OP}}(\mu, \mu')$ (Definition 12). Indeed, (9) $\Leftrightarrow \max\{K + 1 - l_m \mid m \in M^D\} \leq T$, with l_m s.t. $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m \Rightarrow \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l_m)$. Furthermore, using Lemma 3, one can notice that the previous proposition is equivalent to

$$\max_{\mu\in\mathcal{T}_{\Sigma_o}(\mathcal{G})} \left\{K + 1 - l_m \mid \delta_D(q^D_{\text{init}}, \mu) = m \wedge \mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l_m)\right\} \leq T.$$

Moreover, from the definition of safe, for a trace $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l)$, one can notice that $K + 1 - l = \min\{|\mu'| - |\mu| \mid \mu \preceq \mu' \wedge \text{safe}_{\mathsf{OP_K}}(\mu', \mu)\}$ with the convention that $l = K + 1$ when $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$. Then (9) $\Leftrightarrow \max_{\mu\in\mathcal{T}_{\Sigma_o}(\mathcal{G})} \left\{ \min\{|\mu'| - |\mu| \mid \mu \preceq \mu' \wedge \text{safe}(\mu', \mu)\}\right\} \leq T$. □

A.3.2 Proposition 7 (p. 29)

Proof We shall prove that: $\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \exists o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow$

$$\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0}) \Rightarrow o = \mu \quad (5)$$

$$\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0}) \Rightarrow o = \max\{\mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \text{safe}_{\mathsf{OP_0}}(\mu, \mu')\} \quad (6)$$

Let us consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$, the proof is conducted by induction on $|\mu|$.
If $|\mu| = 1$, then $\exists \sigma \in \Sigma_o : \mu = \sigma$. The run of μ on $\mathcal{E}$ can be expressed $\text{run}(\mu, \mathcal{E}) = (q^{\mathcal{E}}_{\text{init}}, \sigma/\alpha, q_1)$ with $q_1 \in Q^{\mathcal{E}}, \Gamma^{\mathcal{E}}(q_1) = \alpha$. The R-Enforcer's evolution of configurations is $(q^{\mathcal{E}}_{\text{init}}, \sigma, \epsilon_{\mathcal{M}}) \overset{o}{\hookrightarrow} (q, \epsilon_{\Sigma_o}, m)$ with $\alpha(\sigma, \epsilon_{\mathcal{M}}) = (o, m)$. Let us distinguish according to whether $\sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$ or not.

- If $\sigma \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$, then we use the correctness of R-Verifiers synthesized from K-delay trajectory estimators (Proposition 4). The state m_1 corresponding to q_1 in the corresponding K-delay trajectory estimator is s.t. $m_1(0) \notin 2^S$. Then, using the definition of R-Enforcers synthesis from K-delay trajectory estimators, we have $\alpha \in \{\text{dump}, \text{off}\}$. Using the definition of enforcement operations, we have: $\text{free} \circ \text{delay}(\epsilon_{\mathcal{M}}) = \epsilon_{\mathcal{M}}$, $o = \sigma \cdot (\epsilon_{\mathcal{M}})_{\downarrow \Sigma_o} = \sigma$, $m = \epsilon_{\mathcal{M}}$. Thus, we find (5).
- If $\sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$, then, similarly following from the correctness of R-Verifier synthesized from K-delay trajectory estimators (Proposition 4), we have $\alpha = \text{store}_1$. Similarly, we can find that $o = \epsilon_{\Sigma_o}$ and $m = (\sigma, 1)$. Furthermore, as $\text{safe}_{\mathsf{OP}_0}(\sigma, \epsilon_{\Sigma_o})$, we have (6).

Let us consider $\mu \in \Sigma_o^*$ s.t. $|\mu| = n$ s.t. (5) and (6) hold. Let us note $\mu = \sigma_0 \cdots \sigma_{n-1}$, and consider $\mu \cdot \sigma$. The run of $\mu \cdot \sigma$ on $\mathcal{E}$ can be expressed

$$\text{run}(\mu \cdot \sigma, \mathcal{E}) = (q^{\mathcal{E}}_{\text{init}}, \sigma_0/\alpha_0, q_1) \cdots (q_{n-1}, \sigma_{n-1}/\alpha_{n-1}, q_n) \cdot (q_n, \sigma/\alpha, q_{n+1})$$

with $\forall i \in [1, n+1] : q_i \in Q^{\mathcal{E}}, \alpha \in \{\text{store}_1, \text{dump}, \text{off}\}$, and $\forall i \in [0, n-1] : \alpha_i \in \{\text{store}_1, \text{dump}, \text{off}\}$. Let us distinguish again according to whether $\mu \cdot \sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$ or not.

- If $\mu\cdot\sigma \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$, then following the reasoning for the induction basis, we know that $\alpha \in \{\text{off}, \text{dump}\}$. Using the induction hypothesis, there exists $o \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$ s.t. $\mu \Downarrow_{\mathcal{E}} o$ and the constraints (5) and (6) hold.
  Now we distinguish two cases according to whether $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$ or not.
  - If $\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$, from (5), we know that $o = \mu$. Then, μ induces the following evolution of configurations for $\mathcal{E}$:

    $$(q^{\mathcal{E}}_{\text{init}}, \sigma_0 \cdots \sigma_{n-1} \cdot \sigma, \epsilon_{\mathcal{M}}) \overset{o_0}{\hookrightarrow} (q_1, \sigma_1 \cdots \sigma_{n-1} \cdot \sigma, m_1) \overset{o_1}{\hookrightarrow} \cdots \overset{o_{n-1}}{\hookrightarrow} (q_n, \sigma, \epsilon_{\mathcal{M}})$$

    with $o_0 \cdots o_{n-1} = o = \sigma_0 \cdots \sigma_{n-1}$. Since $\alpha \in \{\text{off}, \text{dump}\}$, $\alpha(\sigma, \epsilon_{\mathcal{M}}) = (\sigma, \epsilon_{\mathcal{M}})$. Then, we deduce the following evolution of configurations:

    $$(q^{\mathcal{E}}_{\text{init}}, \mu \cdot \sigma, \epsilon_{\mathcal{M}}) \cdots \overset{o_{n-1}}{\hookrightarrow} (q_n, \sigma, \epsilon_{\mathcal{M}}) \overset{\sigma}{\hookrightarrow} (q_{n+1}, \epsilon_{\Sigma_o}, \epsilon_{\mathcal{M}}).$$

    Then, we deduce $\mu \cdot \sigma \Downarrow_{\mathcal{E}} \mu \cdot \sigma$, which gives us (5).
  - Else ($\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$), from (6), we know that $o = \max\{\mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \text{safe}_{\mathsf{OP}_0}(\mu, \mu')\}$, i.e., using the definition of $\text{safe}_{\mathsf{OP}_0}$, $o = \sigma_0 \cdots \sigma_{n-2}$. Then, μ induces the following evolution of configurations for $\mathcal{E}$:

    $$(q^{\mathcal{E}}_{\text{init}}, \mu \cdot \sigma, \epsilon_{\mathcal{M}}) \overset{o_0}{\hookrightarrow} \cdots \overset{o_{n-1}}{\hookrightarrow} (q_n, \sigma, (\sigma_{n-1}, 1))$$

    with $o_0 \cdots o_{n-1} = o = \sigma_0 \cdots \sigma_{n-2}$, and $o_{n-1} = \epsilon_{\Sigma_o}$. Since $\alpha \in \{\text{off}, \text{dump}\}$, $\alpha(\sigma, (\sigma_{n-1}, 1)) = (\sigma_{n-1} \cdot \sigma, \epsilon_{\mathcal{M}})$. Then, we deduce the following evolution of configurations:

    $$(q^{\mathcal{E}}_{\text{init}}, \mu \cdot \sigma, \epsilon_{\mathcal{M}}) \cdots \overset{o_{n-1}}{\hookrightarrow} (q_n, \sigma, (\sigma_{n-1}, 1)) \overset{\sigma_{n-1}\cdot\sigma}{\hookrightarrow} (q_{n+1}, \epsilon_{\Sigma_o}, \epsilon_{\mathcal{M}}).$$

    Then, we deduce $\mu \cdot \sigma \Downarrow_{\mathcal{E}} \mu \cdot \sigma$, i.e., (5).

– Else ($\mu \cdot \sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$), the same reasoning can be followed: we distinguish according to whether $\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$ or not, apply the induction hypothesis, and use the definition of enforcement operations. □

A.3.3 Proposition 8 (p. 29)

Proof We shall prove that, for $\mathsf{OP_K} \in \{\mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}\}$:
$\forall \mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}), \exists o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow (5) \wedge (6)$, where:

$$\mu \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}) \Rightarrow o = \mu \quad (5)$$

$$\mu \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}) \Rightarrow o = \max\{\mu' \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \text{safe}_{\mathsf{OP_K}}(\mu, \mu')\}. \quad (6)$$

Let us consider $\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G})$, the proof is conducted by induction on $|\mu|$. Moreover, the proof is done for $\mathsf{OP_K} \in \{\mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}\}$, a K-step based notion of opacity (independently from whether it is weak or strong), since we will use the function $\text{hold}_{\mathsf{OP_K}}()$ for the state of the underlying trajectory estimator and the traces of the system.

If $|\mu| = 1$, then $\exists \sigma \in \Sigma_o : \mu = \sigma$. The run of μ on $\mathcal{E}$ can be expressed $run(\mu, \mathcal{E}) = (q^{\mathcal{E}}_{\text{init}}, \sigma/\alpha, q_1)$ with $q_1 \in Q^{\mathcal{E}}, \Gamma^{\mathcal{E}}(q_1) = \alpha$. The R-Enforcer's evolution of configurations is $(q^{\mathcal{E}}_{\text{init}}, \sigma, \epsilon_{\mathcal{M}}) \overset{o}{\hookrightarrow} (q_1, \epsilon_{\Sigma_o}, m)$ with $\alpha(\sigma, \epsilon_{\mathcal{M}}) = (o, m)$. Let us distinguish two cases according to whether $\sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$ or not.

– If $\sigma \notin \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$, then we use the correctness of R-Verifiers synthesized from K-delay trajectory estimators (Proposition 4). Using the definition and the properties of the function hold (Section 6.3, paragraph "When is the opacity of a secret enforceable on a system?"), the state m_1 corresponding to q_1 in the corresponding K-delay estimator is s.t. $\text{hold}_{\mathsf{OP_K}}(\delta_D(m^D_{\text{init}}, \sigma)) = \text{hold}_{\mathsf{OP_K}}(m_1) = 0$. Then, using the definition of R-Enforcers synthesis, we have $\alpha \in \{\text{dump}, \text{off}\}$. Using the definition of enforcement operations, we have: $\text{free} \circ \text{delay}(\epsilon_{\mathcal{M}}) = \epsilon_{\mathcal{M}}$, $o = (\epsilon_{\mathcal{M}})_{\downarrow \Sigma_o} \cdot \sigma = \sigma$, $m = \epsilon_{\mathcal{M}}$. Thus, we find (5).
– If $\exists k \in [0, K] : \sigma \in \text{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, k)$, then necessarily $k \in \{0, 1\}$. Similarly, following from the correctness of R-Verifiers synthesized from K-delay trajectory estimators (Propositions 4 and 5) and the definition of $\text{hold}_{\mathsf{OP_K}}$, we have $\text{hold}_{\mathsf{OP_K}}(\delta_D(m^D_{\text{init}}, \sigma)) = \text{hold}_{\mathsf{OP_K}}(m_1) = K + 1 - k$. From the definition of R-Enforcer synthesis, it follows that $\alpha = \text{store}_d$ with $d = K + 1 - k$. Similarly, we can find that $o = \epsilon_{\Sigma_o}$ and $m = (\sigma, d)$. Furthermore, as $\text{safe}_{\mathsf{OP_K}}(\sigma, \epsilon_{\Sigma_o})$, we have (6).

The induction case is performed again by distinguishing according to the opacity leakage of $\mu \cdot \sigma$. Similarly to the induction basis, we use the links between $\text{hold}_{\mathsf{OP_K}}$ applied to the states of the underlying trajectory estimator and the correctness of R-Verifiers. Then, one can easily show, using the definitions of enforcement operations, that the synthesized R-Enforcer is sound and transparent. Furthermore, one has to notice that, when an R-Enforcer produces a halt operation while reading a (partial) trace μ, no extension μ' of μ s.t. $|\mu'| - |\mu| \leq T$ can lead μ to be safely produced (i.e., μ' s.t. $\text{safe}_{\mathsf{OP_K}}(\mu', \mu)$). □