Various Notions of Opacity Verified and Enforced at Runtime

Yliès Falcone, Hervé Marchand

Various Notions of Opacity Verified and Enforced at Runtime

Yliès Falcone — Hervé Marchand

N° 7349 — version 2

initial version July 2010 — revised version August 2010

Domaine 2

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7349--FR+ENG

Various Notions of Opacity Verified and Enforced at Runtime

Yliès Falcone, Hervé Marchand [*]

Domaine : Algorithmique, programmation, logiciels et architectures
Équipes-Projets Vertecs

Rapport de recherche n° 7349 — version 2[†] — initial version July 2010 — revised version August 2010 — 33 pages

Abstract: In this paper, we are interested in the validation of *opacity* where opacity means 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 *verify and enforce*, at system *runtime*, several levels of opacity. Besides already considered notions of opacity, we also introduce a new one that provides a stronger level of confidentiality.

Key-words: opacity, runtime techniques, verification, enforcement

[*] Email: FirstName.LastName@inria.fr. Homepage: `http://www.irisa.fr/prive/{yfalcone,hmarchand}`
[†] This updated version brings an overview of TAKOS, a toolbox implementing the results proposed by this report.

Résumé : Dans ce rapport, nous nous intéressons à la validation de l'*opacité*, où l'opacité signifie l'impossibilité pour un attaquant d'obtenir la connaissance d'un *secret* dans un système donné. D'un point de vue abstrait, assurer l'opacité garantie la confidentialité d'un secret qui ne doit fuir sur un système. Plus spécifiquement, nous étudions comment il est possible de *vérifier et d'assurer, à l'exécution* du système, plusieurs niveaux d'opacité. En plus de notions d'opacité déja considérées dans des travaux antérieurs, nous introduisons une nouvelle notion qui fournit un niveau de confidentialité plus élevé.

Mots-clés : opacité, technique à l'exécution, verification, enforcement

1 Introduction

Security is a major concern in nowadays information systems. While infrastructures are becoming more and more complex, it becomes harder to ensure desired security properties. Using formal methods that can be automated for ensuring security is a way to get confident in the system's behavior at a low cost. Three dimensions in security are usually distinguished: *confidentiality, integrity, availability.* Confidentiality is concerned with ensuring that the information in a system is accessible only to appropriate users. 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 these security concerns, opacity [BBB$^+$07, BKMR08] is a kind of confidentiality property aiming to preserve unwanted retrievals of a system secret by untrusted users. 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 (*e.g.*, the value of confidential variables). While usual opacity is concerned by the *current* disclosure of the secret, K-opacity [SH07] was introduced to also model secret retrieval in the past (*e.g.*, K execution steps before). Ensuring opacity on a system is usually performed using *supervisory control* [CL06], as for example in [DDM10]. 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. While supervisory control for opacity is now given a comprehensive theory, in this paper we investigate the use of practical techniques to validate the opacity of a system: runtime validation techniques.

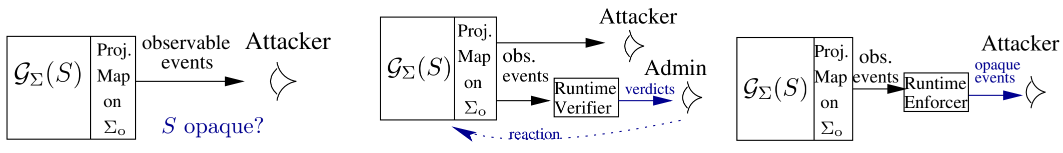
Runtime validation techniques. In this paper we are interested in runtime validation techniques, namely runtime verification and runtime enforcement so as to validate several levels of opacity on a system.

Runtime verification [Run10, PZ06, HG08, FFM09b] consists in checking during the execution of a system whether a desired property holds or not. There exist two main categories of approaches in runtime verification: verification of user-provided specifications and verification of absence of concurrency errors (*e.g.*, deadlock [BFHM06] and datarace [BH08]). In the general case, one uses a special decision procedure called a *monitor* grabbing relevant information in the run of an executing system and acting as an oracle to decide property validation or violation. The major part of research endeavor was done on the monitoring of safety properties, as seen for example in [RCB08]. Recently, a new definition of monitorability was given by Pnueli in [PZ06]. In [FFM09b], we have proposed a first exact characterization of the set of monitorable properties following Pnueli's definition. This characterization was done in the Safety-Progress classification of properties [CMP92b, CMP92a, MP90]. Furthermore, noticing the limitations of the previous definition of monitorability, we have proposed an alternative definition circumventing these limitations. From an abstract point of view, our definition of monitorability better takes into account the practical observation of a monitor and is based on its ability of distinguishing "good" and "bad" execution sequences of a system.

Runtime enforcement [Sch00, HMS06, FFM09a] is an extension of runtime verification aiming to circumvent property violations. It was initiated by the work of Schneider [Sch00] on what has been called *security automata.* In this work the monitors watch the current execution sequence and halt the underlying program whenever it deviates from the desired property. Such security automata are able to enforce the class of safety properties [HMS06] stating that *something bad can never happen.* More recently, Ligatti et al. [LBW09] showed that it is possible to enforce at runtime more than safety properties. Using a more powerful enforcement mechanism called *edit-automata*, it is possible to enforce the larger class of *infinite renewal properties.* In [FFM09a], we have proposed a generic notion of enforcement monitor that encompasses previous mechanisms. Moreover, we have characterized the set of enforceable properties independently of any enforcement mechanism.

Motivations. Whilst supervisory control is established as an effective technique to ensure opacity of properties, it suffers from two practical limitations. First, it is an intrusive technique. Thus, the behavior of the underlying system has to be modifiable before implementing a supervisory control approach. Second, in order to ensure opacity, supervisory control often entails to disable some behaviors of the underlying system. As a consequence, opacity preservation comes often at the price of not achieving intended service fulfilment. Let us note that ensuring opacity via dynamic observability has also been investigated in [CDM09]. This dynamic technique consists in restraining, in each state of the system, the set of observable events in order to ensure opacity. Though this technique achieves opacity preservation, it comes at the price of destroying observable behavior of the system.

Those limitations motivate for investigating the use of others validation techniques to ensure opacity. Furthermore, runtime techniques have proven to be now mature enough in order to address industrial challenges. At runtime, one is able to check non trivial properties on programs. And, runtime validation of opacity as proposed in this paper can be combined with others existing runtime frameworks for checking general properties. Moreover, acting with a runtime validation technique has the advantage of not being intrusive which appears to be better in term of confidence for the system provider. Finally, checking the opacity at runtime opens the way to react to misbehaviors; at it is shown with runtime enforcement in this paper.



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

Figure 1: Opacity and its validation on a system

The proposed approach. The considered problem can be depicted in Figure 11a. A system $\mathcal{G}$ produces sequences of events belonging to an alphabet Σ. Some of these events, in an alphabet $\Sigma_o \subseteq \Sigma$, are observable by an external attacker. Among the possible executions of the system, some of these are said to be secret. A projection map defines the observability of events. We are interested in the opacity of a secret on the considered system. That is, from the sequence of observable events, the attacker should not be able to deduce whether the current execution of the system is secret or not. In this case, the secret S is said to be opaque[1] w.r.t. the considered system and its projection map.

When verifying opacity at runtime (Figure 11b), we introduce a runtime verifier which observes the same sequence of observable events as the attacker. The runtime verifier 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 (Figure 11c), we introduce a runtime enforcer to which the sequence of observable events is directly fed. This mechanism modifies its input sequence and produces a new one in such a way that the desired notion of opacity is preserved by the output sequence. Runtime enforcement may also be seen as a way to automatically prevent opacity violation.

Originality. Opacity cannot be characterized directly as a class of properties since it depends also on the specification of the system. Runtime techniques 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 be able to apply runtime techniques that operates on the observable behaviors of a system. Moreover, to the best of our knowledge, no runtime approach was proposed on this topic. Thus, this paper also studies how more evolved kinds of properties such as opacity can be validated using standard runtime-based techniques.

Paper organization. The remainder of this paper is organized as follows. In Section 2, we introduce some needed notations and concepts. Then, Section 3 presents the various notions of opacity we consider. In Section 4 and Section 5, we respectively verify and enforce opacity at runtime. Finally, Section 6 gives some concluding remarks and opened perspectives.

2 Preliminaries and notations

Considering a finite set of elements E, we define notations about sequences of elements belonging to E. A *sequence* or *string* s containing elements of E is formally defined by a total function $s : I \rightarrow E$ where I is either the integer interval $[0, n]$ for some $n \in \mathbb{N}$, or $\mathbb{N}$ itself (the set of natural numbers). We

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

denote by E^* the set of finite sequences over E (partial function from $\mathbb{N}$), 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 (number of elements) of a finite sequence s is noted $|s|$ and the $(i+1)$-th element of s is denoted by s_i. For two sequences $s, s' \in E^*$, we denote by $s \cdot s'$ (or ss' when clear from context) the concatenation of s and s', and by $s \prec s'$ the fact that s is a strict prefix of s'. The sequence s is said to be a strict prefix of $s' \in E^*$ when $\forall i \in [0, |s| - 1] : s_i = s'_i$ and $|s| < |s'|$. Given $s' \preceq s'$, $|s - s'| \stackrel{\text{def}}{=} |s| - |s'|$. The sequence $s' \in \Sigma^*$ is said to be a strict suffix of $s \in \Sigma^*$, when $|s'| < |s|$ and $\forall i \in [0, |s'| - 1] : s'_i = s_{|s|-1-|s'|+i}$. When $s' \in E^*$, we note $s \preceq s' \stackrel{\text{def}}{=} s \prec s' \vee s = s'$ and $s \succeq s' \stackrel{\text{def}}{=} s \succ s' \vee s = s'$.

When E is the set of system events, we denote it Σ. Any subset of Σ^* is called a *language* over Σ. Let L be a language over Σ. L is said to be *extension-closed* when $L \cdot \Sigma^* = L$. The *prefix-closure* of L is defined as $\mathit{Pref}(L) \stackrel{\text{def}}{=} \{s \in \Sigma^* \mid \exists t \in \Sigma^* : s \cdot t \in L\}$. A language L is said *prefix-closed* whenever $L = \mathit{Pref}(L)$.

Labelled Transitions Systems. We assume that the behaviors of systems are modeled by Labelled Transitions Systems (LTS for short). The formal definition of a LTS is as follows:

Definition 2.1 (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 $\rightarrow_{\mathcal{G}}$ to arbitrary sequences by setting: $q \xrightarrow{\varepsilon}_{\mathcal{G}} q$ 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 $X \subseteq Q^{\mathcal{G}}$, $\Delta_{\mathcal{G}}(X, L) \stackrel{\text{def}}{=} \{q \in Q^{\mathcal{G}} \mid \exists s \in L, \exists q' \in X : q' \xrightarrow{s}_{\mathcal{G}} q\}$.
A *run* ρ from state $q^{\mathcal{G}}_{\text{init}}$ in $\mathcal{G}$ is a finite sequence of transitions

$$q^{\mathcal{G}}_{\text{init}} \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{i-1} \xrightarrow{a_i} q_i \cdots \; q_{n-1} \xrightarrow{a_n} q_n$$

s.t. $a_{i+1} \in \Sigma$ and $q_{i+1} \in \delta(q_i, a_{i+1})$ for $0 \leq i \leq n-1$. The *trajectory* corresponding to the run ρ is $\mathit{tr}(\rho) = a_1 \cdot a_2 \cdots a_n$. The length of ρ, denoted $|\rho|$, is n. The set of finite runs from $q^{\mathcal{G}}_{\text{init}}$ in $\mathcal{G}$ is denoted $\mathit{Runs}(\mathcal{G})$. $\mathcal{L}(\mathcal{G}) = \mathit{tr}(\mathit{Runs}(\mathcal{G})) = \{s \in \Sigma^*, q^{\mathcal{G}}_{\text{init}} \xrightarrow{s}_{\mathcal{G}}\}$ denotes the set of trajectories corresponding to some runs 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 $\mathcal{L}_{F_{\mathcal{G}}}(\mathcal{G}) \stackrel{\text{def}}{=} \{s \in \Sigma^* \mid \exists q \in F_{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}} \xrightarrow{l}_{\mathcal{G}} q\}$, *i.e.*, the set of trajectories that end in $F_{\mathcal{G}}$.

Previous notions apply to deterministic finite state automata (DFA) which are LTS with marked states.

Observable Behavior. The interface between a user and the system is specified by a sub-alphabet of events $\Sigma_{\text{o}} \subseteq \Sigma$. The behavior that is visible by a user, is then defined by its *projection* denoted by $P_{\Sigma_{\text{o}}}$ from Σ^* to Σ_{o}^* that erases in a sequence of Σ^* all events not in Σ_{o}. Formally,

$$\begin{array}{rccl} P_{\Sigma_{\text{o}}} : & \Sigma^* & \to & \Sigma_{\text{o}}^* \\ & \varepsilon_{\Sigma} & \mapsto & \varepsilon_{\Sigma_{\text{o}}} \\ & s \cdot \sigma & \mapsto & \left\{ \begin{array}{l} P_{\Sigma_{\text{o}}}(s) \cdot a \text{ if } a \in \Sigma_{\text{o}} \\ P_{\Sigma_{\text{o}}}(s) \text{ otherwise} \end{array} \right. \end{array}$$

This definition extends to any language $K \subseteq \Sigma^*$: $P_{\Sigma_{\text{o}}}(K) = \{\mu \in \Sigma_{\text{o}}^* \mid \exists s \in K : \mu = P_{\Sigma_{\text{o}}}(s)\}$. In particular, given a 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 $\mathcal{T}_{\Sigma_{\text{o}}}(\mathcal{G}) = P_{\Sigma_{\text{o}}}(L(\mathcal{G}))$. Given two sequences $s, s' \in \Sigma$, they are equivalent w.r.t. the observation induced by $P_{\Sigma_{\text{o}}}$, noted $s \approx_{\Sigma_{\text{o}}} s'$ whenever $P_{\Sigma_{\text{o}}}(s) = P_{\Sigma_{\text{o}}}(s')$. Moreover, given two trajectories $s' \preceq s$, $|s - s'|_{\Sigma_{\text{o}}} \stackrel{\text{def}}{=} |P_{\Sigma_{\text{o}}}(s)| - |P_{\Sigma_{\text{o}}}(s')|$ corresponds to the number of observable events that are necessary to extend s' into s. Conversely, given $K \subseteq \Sigma_{\text{o}}^*$, the *inverse projection* of K is $P^{-1}_{\Sigma_{\text{o}}}(K) = \{s \in \Sigma^* \mid P_{\Sigma_{\text{o}}}(s) \in K\}$.

Given an observation trace μ of $\mathcal{G}$, we define $[\![\mu]\!]^{\mathcal{G}}_{\Sigma_o}$ as the set of trajectories of $\mathcal{G}$ *compatible* with μ, *i.e.*, trajectories of $\mathcal{G}$ having trace μ. Formally:

$$[\![\mu]\!]^{\mathcal{G}}_{\Sigma_o} \stackrel{\text{def}}{=} P^{-1}_{\Sigma_o}(\mu) \cap \mathcal{L}(\mathcal{G})$$

In the remainder $[\![\mu]\!]^{\mathcal{G}}_{\Sigma_o}$ might be denoted $[\![\mu]\!]_{\Sigma_o}$ or $[\![\mu]\!]$ when clear from the context. Given $\mu' \preceq \mu$, we denote $[\![\mu'/\mu]\!]_{\Sigma_o} \stackrel{\text{def}}{=} [\![\mu']\!]_{\Sigma_o} \cap \mathit{Pref}([\![\mu]\!]_{\Sigma_o})$ the set of trajectories of $\mathcal{G}$ that are still compatible with μ' knowing that μ' is the prefix of the trace μ that occurred in the system.

A general notion of monitor. A monitor is a decision procedure consuming *observable events* (in Σ_o) and producing an appraisal on the sequence read so far for a property under interest, here the various notions of opacity that will be introduced in the next section. A general definition of monitor is the following:

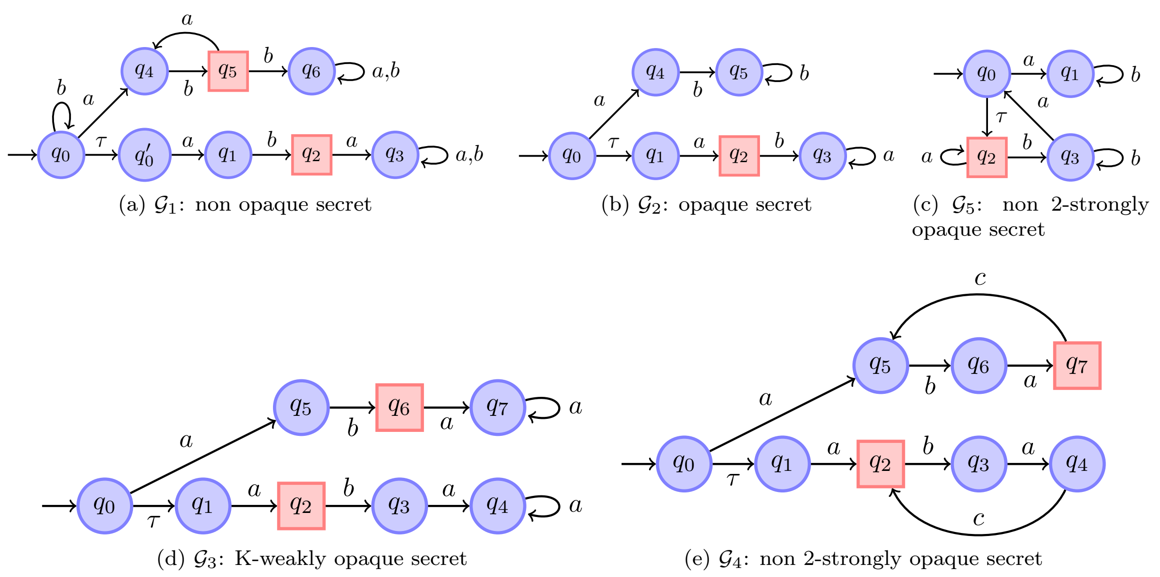
Definition 2.2 (Monitor) *A* monitor $\mathcal{M}$ *is a complete Moore Automaton* $(Q^{\mathcal{M}}, q^{\mathcal{M}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{M}}, X^{\mathcal{M}}, \Gamma^{\mathcal{M}})$*, where* $Q^{\mathcal{M}}$ *denotes the finite set of control states,* $q^{\mathcal{M}}_{\text{init}} \in Q^{\mathcal{M}}$ *is the initial state,* $\delta^{\mathcal{M}}$ *the complete transition function from* $Q^{\mathcal{M}} \times \Sigma_o \to Q^{\mathcal{M}}$ *and* $\Gamma^{\mathcal{M}} : Q^{\mathcal{M}} \to X^{\mathcal{M}}$ *the output function.*

The monitor that we will define in Sections 4 and 5 are finite-state machines producing an output in a relevant domain $X^{\mathcal{M}}$. This domain will be specified for verification and enforcement monitors. For monitors dedicated to verification, this output function gives a truth-value in a dedicated truth-domain for opacity. For monitors dedicated to enforcement, this function produces an enforcement operation inducing a modification on the input sequence so as to enforce the desired opacity.

3 Several notions of opacity

In this section, we formalize three different kinds of opacity. Opacity is defined on the behavior of the system (observable and unobservable). Since we are interested in runtime techniques, we will also characterize the various notions of opacity using the observable behavior of the system.

In the sequel, we let $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ be an LTS over Σ and $\Sigma_o \subseteq \Sigma$. The alphabet Σ_o defines the interface allowing a user to observe $\mathcal{G}$. In this paper, we shall consider that the confidential information is directly encoded in the system by means of a set of states[2] S. In Figure 2, several examples of systems are depicted; secret states are represented using red squares. These examples will be used to illustrate the concepts introduced in the remainder of this paper.



(a) $\mathcal{G}_1$: non opaque secret
(b) $\mathcal{G}_2$: opaque secret
(c) $\mathcal{G}_5$: non 2-strongly opaque secret
(d) $\mathcal{G}_3$: K-weakly opaque secret
(e) $\mathcal{G}_4$: non 2-strongly opaque secret

Figure 2: Several systems and their opacity

[2]Equivalently, the secret can be given by a regular language over Σ^*, see [CDM09] for more details.

3.1 Simple opacity

If the current trajectory of the plant is $t \in \mathcal{L}(\mathcal{G})$, the attacker should not be able to deduce, from the knowledge of $P_{\Sigma_o}(t)$ and the structure of $\mathcal{G}$, that the current state of the system is in S. Next we introduce the notion of opacity based on the one defined in [BKMR08]:

Definition 3.1 (Simple Opacity) *On the system $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, the secret $S \subseteq Q^{\mathcal{G}}$ is opaque under the projection map P_{Σ_o} or $(\mathcal{G}, P_{\Sigma_o})$-opaque if*

$$\forall t \in \mathcal{L}_S(\mathcal{G}), \exists s \in \mathcal{L}(\mathcal{G}) : s \approx_{\Sigma_o} t \wedge s' \notin \mathcal{L}_S(\mathcal{G})$$

We use OP_0 to refer to simple opacity, and we denote by $\mathsf{OP}_0(\mathcal{G}, P_{\Sigma_o}, S)$, the fact that the secret S is $(\mathcal{G}, P_{\Sigma_o})$-opaque.

Example 3.1 *Consider the LTS $\mathcal{G}_1$ of Figure 2a, with $\Sigma_o = \{a, b\}$. The secret is given by the set of states $S = \{q_2, q_5\}$. The secret S is not $(\mathcal{G}_1, P_{\Sigma_o})$-opaque, as after the observation of a trace in $b^* \cdot a \cdot b$, the attacker knows that the system is 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.* □

Another characterization of simple opacity can be given in terms of observed traces: S is opaque w.r.t. $\mathcal{G}$ and P_{Σ_o} whenever $\forall \mu \in \mathcal{T}(\mathcal{G}) : [\![\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$ [Dub09]. Thus, for a trace μ of the system, we say that μ has preserved the opacity of the secret of $\mathcal{G}$ or equivalently that the secret is simply opaque during the run of the system producing μ if $\forall \mu' \preceq \mu : [\![\mu']\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})$. Dually, the set of traces for which the simple opacity of the secret S is leaking is formally defined by:

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

Furthermore, S is $(\mathcal{G}, P_{\Sigma_o})$-opaque if and only if $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0) = \emptyset$ [Dub09].

Likewise, $\mathit{leaked}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0)$ denotes the set of traces for which the secret has leaked in the past:

$$\mathit{leaked}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0) = \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_0) \cdot \Sigma^*_{obs}$$

3.2 K-step based opacity

The above definition of opacity requires that the secret should not be revealed to the attacker only when the system is currently in a secret state. Once the system evolves out of this set of states, the secret cannot be revealed. However, one can argue that the secret could leak whenever the attacker is able to infer that the system went through a secret state in the past. We illustrate this remark through the following example:

Example 3.2 *Consider the LTS $\mathcal{G}_2$ of Figure 2b, with $\Sigma_o = \{a, b\}$. The secret is given by $S = \{q_2\}$. The secret S 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 only compatible trajectory corresponding to this observation is $\tau \cdot a \cdot b \cdot a$, and the attacker can deduce that after the observation of 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: the K-step weak opacity defined in [SH07] and the K-step strong opacity.

K-step weak opacity imposes the opacity of the secret even until K observations are performed on the system after the system was in a secret state. Note that when $K = \infty$, it entails that an attacker should never know that the system was in a secret state in the past.

K-step strong opacity imposes the opacity of the *occurrence of the secret* even until K observations are performed on the system. That is, the attacker must not be sure that the secret occurred on the system during the last K observations.

Intuitively, these notions also allow to say that the knowledge of the secret becomes worthless after the observation of a given number of actions.

Next, we formally introduce the notion of K-step weak opacity as defined in [SH07][3].

[3]Compared with [SH07], for simplicity, we only consider a unique initial state.

Definition 3.2 (K-step weak opacity) *On the system $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, the secret $S \subseteq Q^{\mathcal{G}}$ is K-step weakly opaque under the projection map P_{Σ_o} or $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if*

$$\forall t \in \mathcal{L}(\mathcal{G}), \forall t' \preceq t, |t - t'|_{\Sigma_o} \leq K : t' \in \mathcal{L}_S(\mathcal{G})$$

$$\Rightarrow \exists s \in \mathcal{L}(\mathcal{G}), \exists s' \preceq s : s \approx_{\Sigma_o} t \wedge s' \approx_{\Sigma_o} t' \wedge s' \notin \mathcal{L}_S(\mathcal{G})$$

Intuitively, S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if for each observable trace of the system, and each of its K longest prefixes, if there exists a compatible execution of the system ending in a secret state, then there exists another compatible execution of the system that does not end in a secret state. A consequence of this definition is that the "observable difference" between the length of a sequence $t \in \mathcal{L}(\mathcal{G})$ and its "hiding sequence" s' is always smaller than K.

Remark 3.1 *Note that if S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque then it is also $(\mathcal{G}, P_{\Sigma_o}, K')$-weakly opaque for $K' \leq K$. Especially, if S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque for $K \geq 1$, then S is $(\mathcal{G}, P_{\Sigma_o})$ opaque [SH07]. Furthermore, 0-step weak opacity corresponds to simple opacity.*

Example 3.3 (K-step weak opacity) *Back to Example 3.2 and Fig. 2b, we can easily check that the secret is $(\mathcal{G}_2, P_{\Sigma_o}, 1)$-weakly opaque. However, following the reasons developed in Example 3.2, the secret is not $(\mathcal{G}_2, P_{\Sigma_o}, 2)$-weakly opaque.*

We now characterize the above K-step weak opacity definition in terms of traces of the system.

proposition 3.1 *Let $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, and $S \subseteq Q^{\mathcal{G}}$, then S 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 $\mu \in \mathcal{T}(\mathcal{G})$, $\mu' \preceq \mu$ 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

In the sequel, the set of traces, for which the K-step weak opacity of the secret is revealed, is formally defined by:

$$\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}) = \{\mu \in \mathcal{T}_{\Sigma_o}(\mathcal{G}) \mid \exists \mu' \preceq \mu : |\mu - \mu'| \leq K \wedge [\![\mu'/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G})\} \quad (1)$$

Corollary 3.1 *S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if and only if $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}) = \emptyset$.*

As for the opacity, we define the set of traces for which the secret has leaked in the past:

$$\mathit{leaked}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}) = \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}) \cdot \Sigma^*_{obs}$$

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

$$\begin{aligned}\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, k) = \{\mu \in \mathcal{T}(\mathcal{G}) \mid\ & \exists \mu' \preceq \mu : |\mu - \mu'| = k \wedge [\![\mu'/\mu]\!]_{\Sigma_o} \subseteq \mathcal{L}_S(\mathcal{G}) \wedge \\ & \forall \mu' \preceq \mu : |\mu - \mu'| < k \Rightarrow [\![\mu'/\mu]\!]_{\Sigma_o} \not\subseteq \mathcal{L}_S(\mathcal{G})\}\end{aligned} \quad (2)$$

One may notice that $\bigcup_{0 \leq k \leq K} \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}}, k) = \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}})$.

Example 3.4 (Limits of K-step weak opacity) *Consider now the LTS $\mathcal{G}_3$ of Figure 2d, with $\Sigma_o = \{a, b\}$. The secret is given by $S = \{q_2, q_6\}$. According to Definition 3.2, S is $(\mathcal{G}_3, P_{\Sigma_o}, K)$-weakly opaque for every $K \in \mathbb{N}$. However, it is easy to see that after the observation $a \cdot b$ the attacker can deduce that the system is either in states $q_6 \in S$ (if the actual trajectory is $a \cdot b$) or was in state $q_2 \in S$ after the observation of a (if the actual trajectory is $\tau \cdot a \cdot b$). In all cases, the attacker knows that at most one observation before the occurrence of b, the system was in a secret state or currently is in S. The attacker does not know which state it is, but he knows that the secret occurred for sure.*

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 K previous observable steps of the system.

Definition 3.3 (K-step strong opacity) *On the system $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, the secret $S \subseteq Q^{\mathcal{G}}$ is K-step strongly opaque w.r.t. $\mathcal{G}$ and P_{Σ_o} (or $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque) if*

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

Intuitively, a secret is K-step strongly opaque if for each execution of the system t, there exists another execution s, that is equivalent to t and 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 sequence compatible with μ that did not cross a secret state during the last K observations. Using the set

$$\mathit{Free}^S_K(\mathcal{G}) = \{s \in \mathcal{L}(\mathcal{G}) \mid \forall s' \preceq s : \; |s - s'|_{\Sigma_o} \leq K \Rightarrow s' \notin \mathcal{L}_S(\mathcal{G})\}$$

which corresponds to the trajectories of the system $\mathcal{G}$ that did not cross a secret state during the last past K observations, we can give another characterization of the strong opacity

Property 3.1 *The secret $S \subseteq Q^{\mathcal{G}}$ is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque w.r.t. $\mathcal{G}$ and P_{Σ_o} if and only if*

$$\forall \mu \in \mathcal{T}(\mathcal{G}) : [\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_K(\mathcal{G}) \neq \emptyset \tag{3}$$

Proof
($\Leftarrow$) Let $t \in \mathcal{L}(\mathcal{G})$ and $\mu = P_{\Sigma_o}(t)$, by hypothesis we have $[\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_K(\mathcal{G}) \neq \emptyset$. 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 $\mu \in \mathcal{T}(\mathcal{G})$ 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 $s \in \mathit{Free}^S_K(\mathcal{G})$ and as $s \in [\![\mu]\!]_{\Sigma_o}$, we get $[\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_K(\mathcal{G}) \neq \emptyset$.

Remark 3.2 *Note that 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 that S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque then S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque.* $\square$

We note $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}})$ the set of traces for which there is an information flow w.r.t. the K-step strong opacity

$$\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}) = \{\mu \in \mathcal{T}(\mathcal{G}) \mid [\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_K(\mathcal{G}) = \emptyset\}$$

Corollary 3.2 *S is $(\mathcal{G}, P_{\Sigma_o}, K)$-strongly opaque if and only if $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}) = \emptyset$.*

As for the K-step weak opacity, we define the set of traces for which the secret has leaked in the past:

$$\mathit{leaked}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}) = \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}) \cdot \Sigma^*_{obs}$$

Remark 3.3 *Similarly to K-step weak opacity, it will be useful, when validating opacity with runtime techniques, to know that the opacity of the secret leaked exactly k steps ago. Then, one can also decompose the set $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}})$ as follows*

$$\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{S}}_{\mathsf{K}}, k) = \{\mu \in \mathcal{T}(\mathcal{G}) \mid [\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_k(\mathcal{G}) = \emptyset \text{ and } \forall k' < k : [\![\mu]\!]_{\Sigma_o} \cap \mathit{Free}^S_{k'}(\mathcal{G}) \neq \emptyset\}$$

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

Example 3.5 *Back to Example 3.4 and Figure 2d, following the reasons developed in Example 3.4, the secret is not $(\mathcal{G}_3, P_{\Sigma_o}, 1)$-opaque. Consider now the LTS $\mathcal{G}_4$ of Figure 2e. The set of secret states is $\{q_2, q_7\}$. This 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 it is not $(\mathcal{G}_4, P_{\Sigma_o}, 2)$-strongly opaque since after the observed sequence $a \cdot b \cdot a$, we know that either the system is in q_7 (which is a secret state) or is in q_4. In the later case, we know that the system was in q_2 exactly 2 observations ago.*

4 Verification of opacity at runtime

In this section, we are interested in verifying the opacity of a secret w.r.t. a given system. As we shall see, the device that we build, will perform the verification at runtime (as illustrated in Figure 3), but it can also be used to statically (*i.e.*, off-line) check the opacity.

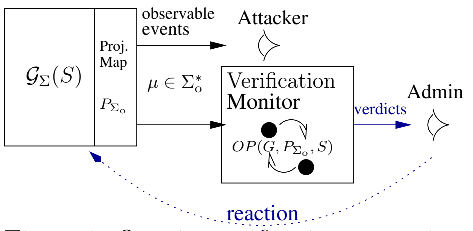


Figure 3: Opacity verification at runtime

We first present a general notion of verifier for the different notions of opacity presented in the previous section. We further introduce the notion of K-delay state estimator borrowed from [SH09] that will be the cornerstone of the construction of the monitors. We will also introduce the notion of K-delay trajectory estimator which is an extension of K-delay state estimator and is dedicated to K-strong opacity.

4.1 Notion of runtime verifier

The aim is to build a monitor for verification, *i.e.*, a function which captures, 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 relatively to the considered opacity.

Definition 4.1 (Runtime verifier) *A* runtime verifier *(R-Verifier) $\mathcal{V}$ is a monitor $(Q^\mathcal{V}, q_{\text{init}}^\mathcal{V}, \Sigma_o, \delta_\mathcal{V}, \mathbb{B}_{\mathsf{OP}}, \Gamma^\mathcal{V})$ where $\Gamma^\mathcal{V} : Q^\mathcal{V} \rightarrow \mathbb{B}_{\mathsf{OP}}$ is the output function. To a runtime verifier, we associate a* verification function *$[\![\mathcal{V}]\!] : \Sigma_o^* \rightarrow \mathbb{B}_{\mathsf{OP}}$ given by*

$$\forall \mu \in \Sigma_o^* : [\![\mathcal{V}]\!](\mu) = \Gamma^\mathcal{V}(\delta_\mathcal{V}(q_{\text{init}}^\mathcal{V}, \mu))$$

$\mathbb{B}_{\mathsf{OP}}$ is a truth domain dedicated to the considered notion of opacity: for simple opacity $\mathbb{B}_{\mathsf{OP}} = \mathbb{B}_{\mathsf{OP}}^0 = \{\text{leak}, \text{noleak}\}$, for K-step based opacity $\mathbb{B}_{\mathsf{OP}} = \mathbb{B}_{\mathsf{OP}}^K = \{\text{leak}_0, \ldots, \text{leak}_\text{K}, \text{noleak}\}$.

Considering the different notions of opacity, we now introduce the properties that a R-verifier should satisfy to be correct:

Definition 4.2 (R-Verifier soundness and completeness) *A R-Verifier $\mathcal{V}$ is sound and complete w.r.t. $\mathcal{G}, P_{\Sigma_o}, S$ and $\mathsf{OP} \in \{\mathsf{OP}_0, \mathsf{OP}_K^W, \mathsf{OP}_K^S\}$ whenever*

- *For simple opacity* $(\mathsf{OP} = \mathsf{OP}_0)$*:*

$$\forall \mu\ \mathcal{T}_{\Sigma_o}(\mathcal{G}) : \quad \begin{array}{lcl} [\![\mathcal{V}]\!](\mu) = \text{leak} & \Leftrightarrow & \mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \\ [\![\mathcal{V}]\!](\mu) = \text{noleak} & \Leftrightarrow & \mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \end{array}$$

- *For K-step based opacity* $(\mathsf{OP} \in \{\mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}\})$*:*

$$\forall \mu\ \mathcal{T}_{\Sigma_o}(\mathcal{G}), \forall l \in [0, K] : \quad \begin{array}{lcl} [\![\mathcal{V}]\!](\mu) = \text{leak}_\text{l} & \Leftrightarrow & \mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}, l) \\ [\![\mathcal{V}]\!](\mu) = \text{noleak} & \Leftrightarrow & \mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \end{array}$$

A R-Verifier is sound ($\Rightarrow$ direction) whenever it never gives a false verdict. It is complete ($\Leftarrow$ direction) if all the observations corresponding to the current leakage of opacity raise a "leak" verdict.

4.2 R-Verifier synthesis for simple opacity

First, we introduce the classical notion of determinization via subset construction adapted to our definition of opacity: $\mathit{Det}_{\Sigma_o}(\mathcal{G})$ denotes the deterministic automaton which is computed from $\mathcal{G}$. Formally, it can be obtained as follows:

Definition 4.3 *Let $\mathcal{G} = (Q^\mathcal{G}, q_{\text{init}}^\mathcal{G}, \Sigma, \delta_\mathcal{G})$ and $\Sigma_o \subseteq \Sigma$ then $\mathit{Det}_{\Sigma_o}(\mathcal{G}) = (\mathcal{X}, X_0, \Sigma_o, \Delta_{\Sigma_o})$ where:*

- *$\mathcal{X} \subseteq 2^{Q^\mathcal{G}} \setminus \emptyset$ and $X_0 = \Delta_\mathcal{G}(\{q_{\text{init}}^\mathcal{V}\}, (\Sigma \setminus \Sigma_o)^*)$*
- *given $a \in \Sigma_o$, if $X' = \Delta_\mathcal{G}(X, a \cdot (\Sigma \setminus \Sigma_o)^*) \neq \emptyset$ then $\Delta_{\Sigma_o}(X, a) = X'$.*

Based on this operation, we can build a R-Verifier w.r.t. the opacity property as follows:

PROPERTY 4.1 *Given a plant $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, a secret $S \subseteq Q^{\mathcal{G}}$ and $\Sigma_o \subseteq \Sigma$, the monitor $(\mathcal{X}, X_0, \Sigma_o, \Delta_{\mathcal{V}}, \mathbb{B}^0_{\mathsf{OP}}, \Gamma)$ built from $Det_{\Sigma_o}(\mathcal{G}) = (\mathcal{X}, X_0, \Sigma_o, \Delta_{\Sigma_o})$ is such that*

- $\forall X \in \mathcal{X}, \forall a \in \Sigma_o$ *s.t.*
  - $X \not\subseteq S$, *then* $\Delta_{\mathcal{V}}(X, a) = \Delta_{\Sigma_o}(X, a)$ *whenever it is defined,*
  - $X \subseteq S$, *then* $\Delta_{\mathcal{V}}(X, a) = X$;
- $\Gamma(X) = leak$ *iff* $X \subseteq S$;

is a sound R-Verifier w.r.t. $\mathcal{G}, P_{\Sigma_o}, S$ and the simple opacity property.

Proof This construction has been proved correct in [DJM09].

4.3 R-Verifier synthesis for K-step weak opacity

When generating runtime mechanisms for the verification of K-step weak opacity, we will need the notion of K-delay state estimator[4], they were introduced in [SH09] for studying initial opacity. Intuitively, a K-delay estimator of a system indicates, according to its observable interface, the estimated states of the system during the K previous steps. Here, we adapt the definition for the notions of opacity of interest in this paper.

4.3.1 K-delay state estimator

First we need to introduce some new notations. Given a LTS $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, for $l \geq 2$. The set of l-tuples of states of $\mathcal{G}$ $(Q^{\mathcal{G}})^l = Q^{\mathcal{G}} \times Q^{\mathcal{G}} \times \cdots \times Q^{\mathcal{G}} = \{(q_1, \ldots, q_l) | q_i \in Q^{\mathcal{G}}, 1 \leq i \leq l\}$ is denoted $Q^{\mathcal{G}}_l$. Intuitively elements of $Q^{\mathcal{G}}_l$ will correspond to partial sequences of states of the system. The set $m \in 2^{Q^{\mathcal{G}}_l}$ is called a l-dimensional state mapping. We denote by $m(i)$ the set of the $(l-i)^{\text{th}}$ states of elements of m. Intuitively, $m(0)$ will correspond to the current state estimate whereas $m(i)$ will correspond to the state estimate knowing that i observations have been made. We also need to define

- the *shift operator* $\| : 2^{Q^{\mathcal{G}}_l} \times 2^{Q^{\mathcal{G}}_2} \to 2^{Q^{\mathcal{G}}_l}$ such that

$$m \| m_2 \overset{\text{def}}{=} \{(q_2, \ldots, q_l, q_{l+1}) | (q_1, \cdots, q_l) \in m \wedge (q_l, q_{l+1}) \in m_2\} \quad (4)$$

- the observation mapping $M : \Sigma_o \to 2^{Q^{\mathcal{G}}_2}$ such that

$$M(\sigma) \overset{\text{def}}{=} \{(q_1, q_2) | q_1, q_2, \in Q^{\mathcal{G}} \wedge \exists s \in \Sigma^* : P(s) = \sigma \wedge q_1 \xrightarrow{s}_{\mathcal{G}} q_2\} \quad (5)$$

- the function $\odot_l : 2^{Q^{\mathcal{G}}} \to 2^{Q^{\mathcal{G}}_l}$ as $\odot_l(Q) = \{(q, \ldots, q) | q \in Q\}$

DEFINITION 4.4 (K-DELAY STATE ESTIMATOR [SH09]) *For a LTS $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, a projection map $\mathcal{P}_{\Sigma_o}$ w.r.t. Σ_o, the* K-delay state estimator *is a DFA $D = (M^D, q^D_{\text{init}}, \Sigma_o, \delta_D)$ s.t.:*

- M^D *is the smallest subset of* $2^{Q^{\mathcal{G}}_{K+1}}$ *reachable from* q^D_{init} *with* δ_D *(defined below),*
- $q^D_{\text{init}} = \odot_{K+1}(\delta_{\mathcal{G}}(q^{\mathcal{G}}_{\text{init}}, [\![\epsilon]\!]_{\Sigma_o}))$,
- $\delta_D : M^D \times \Sigma_o \to M^D$ *defined as* $\forall m \in M^D, \forall \sigma \in \Sigma_o : \delta_D(m, \sigma) = m \| M(\sigma)$.

Thus a K-delay state estimator, for a system $\mathcal{G}$ is a LTS whose states contain compatible sequences of visited states corresponding to each possible trace on $\mathcal{G}$. On each transition, possibly visited states K steps ago are forgotten, and the current state estimate is updated: for a transition, the arriving state is obtained using the shift operator ($\|$) and putting in front (the current state estimate) compatible current states according to the state estimate at the previous step.

[4]We will also use it in Section 5 in order to enforce the various notions of opacity.

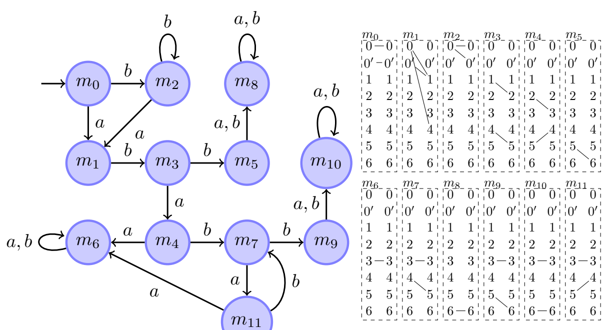
Example 4.1 (K-delay state estimator) *For the LTS $\mathcal{G}_1$ of Example 3.1, Figure 2a, the DFA represented in Figure 4a is the corresponding* 1*-delay state estimator*[5]*. For this* 1*-delay state estimator, examining m_7 gives us the following information: the current state estimate is $\{q_3, q_5\}$ ($\mathcal{G}_1$ is currently in either state q_3 or q_5), it was at the previous step in either q_3 or q_4, and it followed one of the partial runs: $q_3 \xrightarrow{b}_{\mathcal{G}_1} q_3$ or $q_4 \xrightarrow{b}_{\mathcal{G}_1} q_5$ represented by straight lines in the dashed box representing m_7.*

Similarly, the DFA represented in Figure 4b is the 2*-delay state estimator for the LTS $\mathcal{G}_2$ of Example 3.2, Figure 2b. Examining m_2 similarly indicates us that the current state estimate is $\{3, 5\}$, and that the partial runs of $\mathcal{G}_2$ are $q_0 \xrightarrow{a}_{\mathcal{G}_2} q_2 \xrightarrow{b}_{\mathcal{G}_2} q_3$ or $q_0 \xrightarrow{a}_{\mathcal{G}_2} q_4 \xrightarrow{b}_{\mathcal{G}_2} q_5$ or $q_1 \xrightarrow{a}_{\mathcal{G}_2} q_4 \xrightarrow{b}_{\mathcal{G}_2} q_5$.*

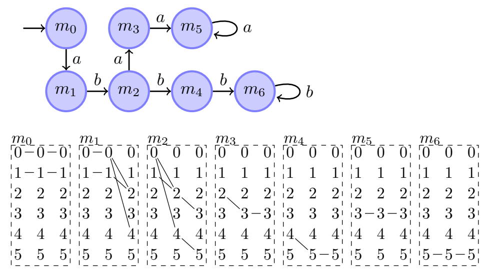
For the LTS $\mathcal{G}_3$ of Example 3.4, Figure 2d, the DFA represented in Figure 4c is the corresponding 2*-delay state estimator.*

For the LTS $\mathcal{G}_4$ of Example 3.5, Figure 2e, the DFA represented in Figure 4d is the corresponding 3*-delay state estimator.*

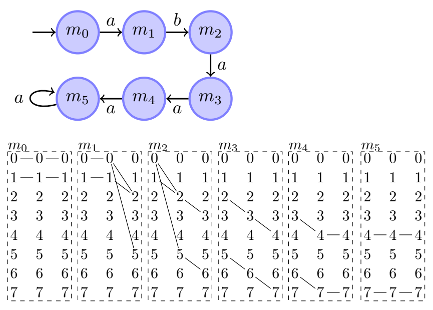
For the LTS $\mathcal{G}_5$ in Figure 2c, the DFA represented in Figure 4e is the corresponding 2*-delay state estimator.*



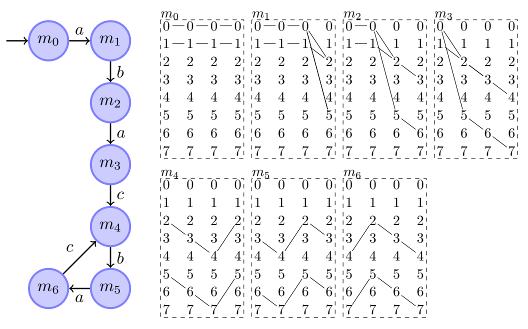
(a) 1-delay state estimator for $\mathcal{G}_1$ of Figure 2a



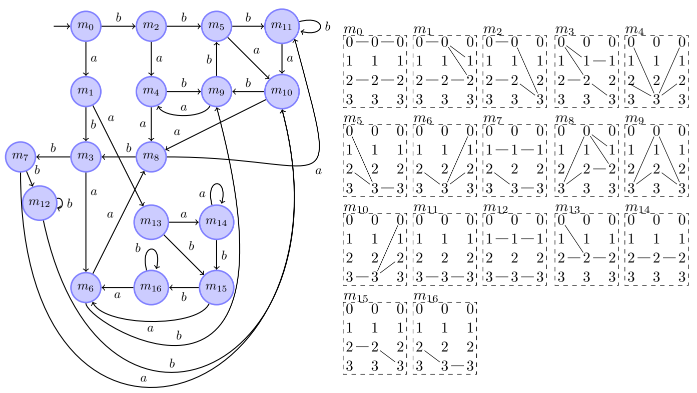
(b) 2-delay state estimator for $\mathcal{G}_2$ of Figure 2b



(c) 2-delay state estimator for $\mathcal{G}_3$ of Figure 2d



(d) 3-delay state estimator for $\mathcal{G}_4$ of Figure 2e



(e) 2-delay state estimator for $\mathcal{G}_5$ of Figure 2c

[5]To simplify notations, when representing K-delay state estimator, we note i instead of q_i for the state estimates.

Next, we introduce two technical lemmas showing that the tuples of states that belongs to a state m of a K-delay state estimator exactly correspond to the sequences of states that are crossed by the ending sequences that are compatible with the traces that reach m.

LEMMA 4.1 *Given a system $\mathcal{G}$ modelled by a LTS $(Q^{\mathcal{G}}, m^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay state estimator $D = (Q^D, q^D_{\text{init}}, \Sigma_o, \delta_D)$, then*

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

$$\forall \mu \in \mathcal{T}(\mathcal{G}) \text{ s.t. } n = |\mu| < K \wedge \mu = \sigma_1 \cdots \sigma_n, \forall s \in [\![\mu]\!]_{\Sigma_o}, : s = s_1 \cdots s_n \wedge \forall i \leq n : P_{\Sigma_o}(s_i) = \sigma_i$$
$$\exists (q_0, \ldots, q_K) \in \delta_D(m^D_{\text{init}}, \mu) : q_{K-n} \xrightarrow{s_1}_{\mathcal{G}} q_{K-n+1} \cdots \xrightarrow{s_{n-1}} q_{K-1} \xrightarrow{s_n}_{\mathcal{G}} q_K, \text{ with } q_{K-n} \in \delta(q^{\mathcal{G}}_{\text{init}}, [\![\epsilon]\!]_{\Sigma_o})$$

Proof The proof is done by induction on $|\mu|$.

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

LEMMA 4.2 *Given a system $\mathcal{G}$ modelled by a LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$ and the corresponding K-delay state estimator $D = (Q^D, q^D_{\text{init}}, \Sigma_o, \delta_D)$, then $\forall m \in Q^D$, $\forall (q_0, \ldots, q_K) \in m$, $\forall \mu \in \mathcal{T}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m$,*

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

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

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

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

PROPOSITION 4.1 *For a system $\mathcal{G}$ modelled by a LTS $(Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, and $\Sigma_o \subseteq \Sigma$, the K-delay state estimator $D = (Q^D, q^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ is such that:*

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

Even though differently presented, a similar result can be found in [SH09].

4.3.2 R-Verifier synthesis

We are now able to tackle the R-Verifier synthesis problem for K-weak opacity.

PROPOSITION 4.2 *Given a plant $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, a secret $S \subseteq Q^{\mathcal{G}}$ and $\Sigma_o \subseteq \Sigma$, the R-Verifier $\mathcal{V} = (Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{V}}, \mathbb{B}^K_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ built from the K-delay state estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ where:*

- $Q^{\mathcal{V}} = M^D, q^{\mathcal{V}}_{\text{init}} = q^D_{\text{init}}, \delta_{\mathcal{V}} = \delta_D$,
- $\Gamma^{\mathcal{V}} : Q^{\mathcal{V}} \to \mathbb{B}^K_{\mathsf{OP}}$ *defined by*
  - $\Gamma^{\mathcal{V}}(m) = \text{leak}_l$ *where* $l = min\{k \in [0, K] \mid m(k) \in 2^S\}$ *otherwise*
  - $\Gamma^{\mathcal{V}}(m) = \text{noleak}$ *if* $\forall k \in [0, K] : m(k) \notin 2^S$

is sound and complete w.r.t. $\mathcal{G}, P_{\Sigma_o}, S$ and the K-step weak opacity property[6].

Proof We prove the soundness and completeness of the synthesized R-Verifiers as exposed in Definition 4.2. We consider $\mu \in \mathcal{T}(\mathcal{G})$ s.t. $\delta_D(m^D_{\text{init}}, \mu) = m$.
$(\Rightarrow)$

- If $[\![\mathcal{V}]\!](\mu) = \text{noleak}$, we have that $\Gamma^D(m) = \text{noleak}$, that is $\forall i \in [0, K] : m(i) \notin 2^S$. Using Proposition 4.1, we have that $\forall i \in [max(K - |\mu|, 0), K] : m(i) = \delta_{\mathcal{G}}(q^{\mathcal{G}}_{\text{init}}, [\![\mu_{\cdots|\mu|-i}/\mu]\!]) \notin 2^S$. That is, $\forall \mu' \preceq \mu : |\mu - \mu'|_{\Sigma_o} \leq K \Rightarrow [\![\mu/\mu']\!] \not\subseteq \mathcal{L}_S(\mathcal{G})$. Which, according to Equation (1), means that $\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^{\mathsf{W}}_{\mathsf{K}})$.
- If $[\![\mathcal{V}]\!](\mu) = \text{leak}_l$, we have that $\Gamma^D(m) = \text{leak}_l$, that is $m(l) \in 2^S$ and $\forall i < l : m(i) \notin 2^S$. Similarly, using Proposition 4.1 we have that $[\![\mu_{\cdots|\mu|-l}/\mu]\!] \subseteq \mathcal{L}_S(\mathcal{G})$ and $\forall i < l : [\![\mu_{\cdots|\mu|-i}/\mu]\!] \not\subseteq \mathcal{L}_S(\mathcal{G})$, *i.e.*, according to Equation (2) $\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^W_K, l)$.

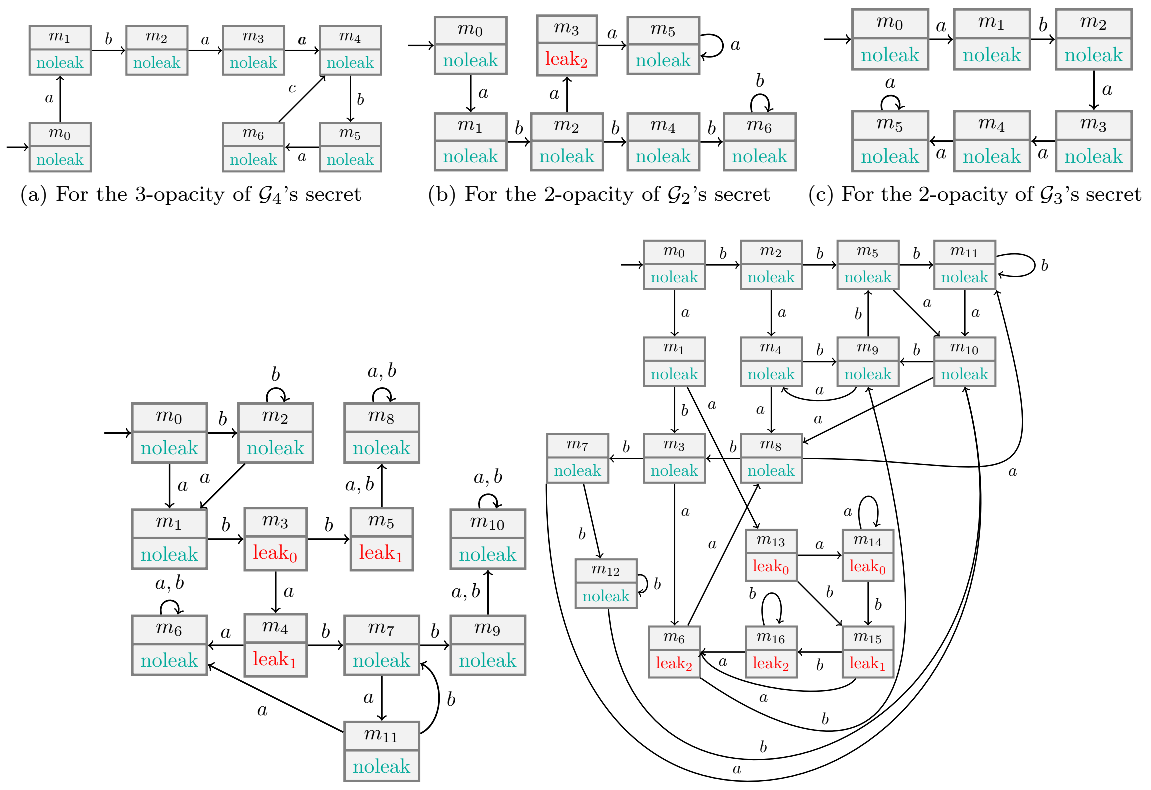
$(\Leftarrow)$

- If $\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^W_K)$, 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 4.1, we have that $\forall i \in [max(|\mu| - K, 0), K] : m(i) \notin 2^S$. Following the definition of R-Verifiers construction, we have that $\Gamma^{\mathcal{V}}(m) = \text{noleak}$.
- If $\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}^W_K, l)$, then according to Equation (1), we have $[\![\mu_{\cdots|\mu|-l}/\mu]\!] \subseteq \mathcal{L}_S(\mathcal{G})$ and $\forall i < l : [\![\mu_{\cdots|\mu|-i}/\mu]\!] \not\subseteq \mathcal{L}_S(\mathcal{G})$. Now, using Lemma 4.1, $\forall i \in [max(|\mu| - K, 0), l[: m(i) \notin 2^S$ 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 $\Gamma^{\mathcal{V}}(m) = \text{leak}_l$.

EXAMPLE 4.2 (R-VERIFIERS OF K-WEAK OPACITY) *In Figure 4 are represented R-verifiers of K-weak opacity for the systems depicted in Figure 2. These monitors are built from their respective K-delay state estimators.*

REMARK 4.1 *One can notice that, as simple opacity corresponds to 0-step opacity, a R-verifier for simple opacity can be obtained from a 1-delay state estimator in a straightforward manner by modifying the verdicts.*

[6] In [SH09] and its companion paper [SH08], it was shown that S is $(\mathcal{G}, P_{\Sigma_o}, K)$-weakly opaque if and only if there does not exist a state m reachable in D such that $\forall k \in [0, K] : m(k) \cap 2^S = \emptyset$. We here adapt the proof to fit with our definition of R-Verifier.



(a) For the 3-opacity of $\mathcal{G}_4$'s secret (b) For the 2-opacity of $\mathcal{G}_2$'s secret (c) For the 2-opacity of $\mathcal{G}_3$'s secret

(d) For the 1-opacity of $\mathcal{G}_1$'s secret (e) R-verifier of 2-step opacity of $\mathcal{G}_5$'s secret

Figure 4: R-verifiers of K-weak opacity

4.4 R-Verifier synthesis for the K-step strong opacity

When dealing with the K-step strong opacity, we are interested in preventing the attacker from being sure that the system went through a secret state during the last K observations. Similarly to K-weak opacity, we use a dedicated estimator to synthesize R-verifiers.

4.4.1 K-delay trajectory estimator

The notion of K-delay trajectory estimator consists in an extension of K-delay state estimator. We consider[7] a LTS $\mathcal{G} = (Q, q_{\text{init}}, \Sigma, \delta)$, and an integer l s.t. $2 \leq l \leq K$.

- the *shift operator* $\|_b : 2^{Q^l \times \mathbb{B}^{l-1}} \times 2^{Q^2 \times \mathbb{B}} \to 2^{Q^l \times \mathbb{B}^{l-1}}$ is such that

$$m\|_b m_2 \stackrel{\text{def}}{=} \{((q_2, \cdots, q_l, q_{l+1}), (b_2, \dots, b_l)) | ((q_1, \dots, q_l), (b_1, \dots, b_{l-1})) \in m_1 \wedge ((q_l, q_{l+1}), b_l) \in m_2\}$$

- the observation mapping $M_b : \Sigma_o \to 2^{Q^2}$ such that

$$M_b(\mu) \stackrel{\text{def}}{=} \{((q_1, q_2), b) | q_1, q_2, \in Q, \exists s \in \Sigma^* : \mathcal{P}_{\Sigma_o}(s) = \mu \wedge q_1 \xrightarrow{s}_{\mathcal{G}} q_2 \wedge b = s \in \mathit{Free}_l^S(\mathcal{G}_{(q_1)})\}$$

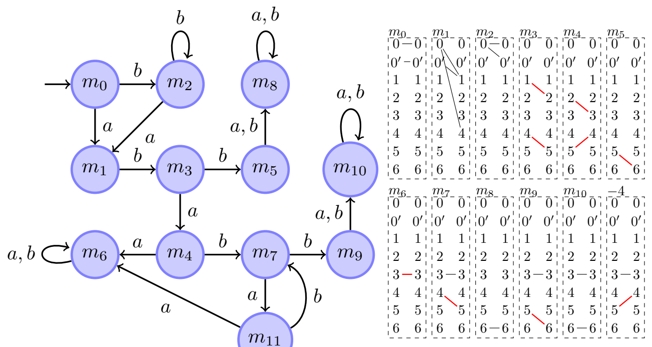
Definition 4.5 (K-Delay Trajectory Estimator) *For a plant $\mathcal{G} = (Q, q_{\text{init}}, \Sigma, \delta)$, a projection map $\mathcal{P}_{\Sigma_o}$ w.r.t. Σ_o, the K-*delay trajectory estimator *is a DFA $D = (M^D, q_{\text{init}}^D, \Sigma_o, \delta_D)$ s.t.:*

- *M^D is the smallest subset of $2^{Q^{K+1} \times \mathbb{B}^K}$ reachable from q_{init}^D with δ_D (defined below),*
- *$q_{\text{init}}^D = \odot_{K+1}(\delta(q_{\text{init}}, [\![\epsilon]\!]_{\Sigma_o}))$,*
- *$\delta_D : M^D \times \Sigma_o \to M_b^D$ defined by $\forall m \in M_b^D, \forall \sigma \in \Sigma_o : \delta_D(m, \sigma) = m\|_b M_b(\sigma)$.*

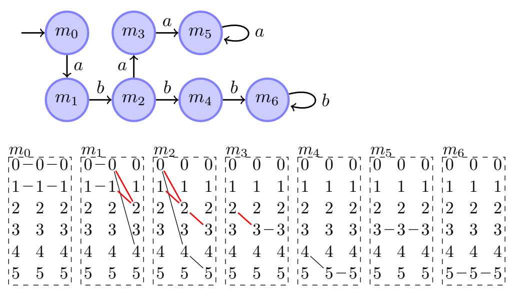
[7]To simplify notation, we may omit superscripts indicating the system under consideration, *e.g.*, we may note Q for $Q^{\mathcal{G}}$ etc.

Remark 4.2 *Similar to K-delay state estimators, K-delay trajectory estimators have the same number of states. The difference lies in the fact that it contains more information regarding the states traversed between two consecutive states of $(q_0, \ldots, q_K)$.*

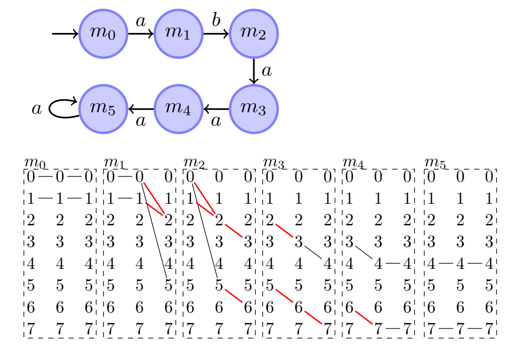
Example 4.3 (K-delay trajectory estimators) *K-delay trajectory estimator can be represented similarly to K-delay state estimator. The difference is that, in a state of the estimator, we "tag in red" links between two states in the system if this transition went through a secret state, i.e., if the associated Boolean is false. Trajectory estimators for the previously considered systems are represented in Figures 5a, 5b, 5c, 5d, 5e.*



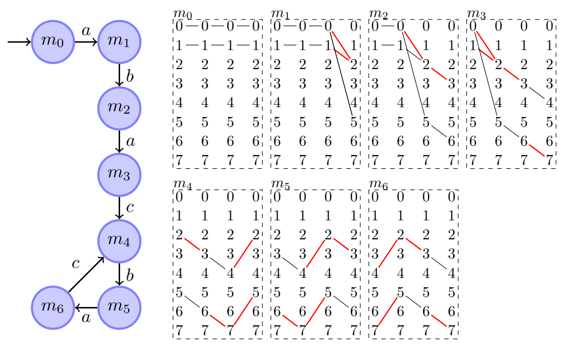
(a) 1-delay trajectory estimator for $\mathcal{G}_1$ of Figure 2a



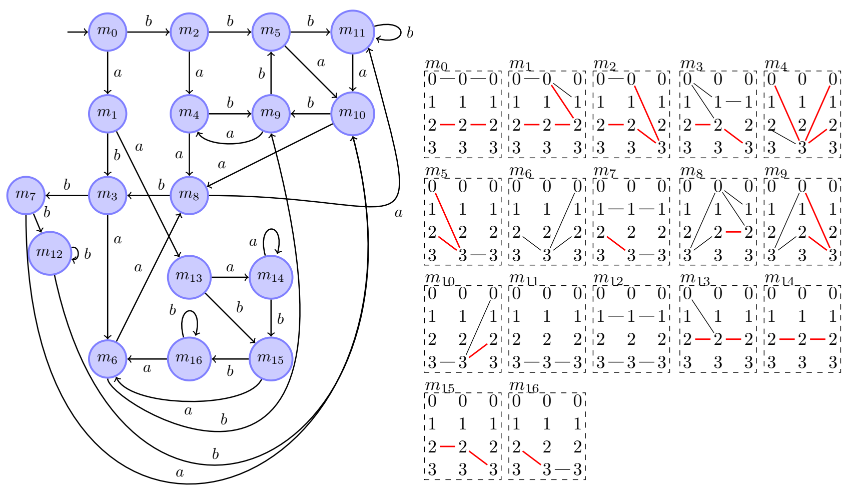
(b) 2-delay trajectory estimator for $\mathcal{G}_2$ of Figure 2b



(c) 2-delay trajectory estimator for $\mathcal{G}_3$ of Figure 2d



(d) 3-delay trajectory estimator for $\mathcal{G}_4$ of Figure 2e



(e) 2-delay trajectory estimator for $\mathcal{G}_5$ of Figure 2c

Figure 5: Trajectory estimators for the LTS of Figure 2

4.4.2 R-Verifier synthesis

Similarly to the synthesis of R-Verifiers for K-weak opacity from K-delay state estimators, we synthesize R-Verifiers for K-strong opacity from K-delay trajectory estimators.

PROPOSITION 4.3 *Given a plant $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, a secret $S \subseteq Q^{\mathcal{G}}$ and $\Sigma_{\text{o}} \subseteq \Sigma$, the R-Verifier $\mathcal{V} = (Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_{\text{o}}, \delta_{\mathcal{V}}, \mathbb{B}^K_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ built from the K-delay state estimator $(Q^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$ of $\mathcal{G}$ where:*

- $Q^{\mathcal{V}} = Q^D, q^{\mathcal{V}}_{\text{init}} = m^D_{\text{init}}, \delta_{\mathcal{V}} = \delta_D$,
- $\Gamma^{\mathcal{V}} : Q^{\mathcal{V}} \to \mathbb{B}^K_{\mathsf{OP}}$ *defined by:*
  - $\Gamma^{\mathcal{V}}(m) = \text{noleak}$ *if* $\exists((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m : \forall i \in [0, K]$*:* $q_i \notin S \wedge \forall i \in [0, K-1] : b_i = \mathit{true}$
  - $\Gamma^{\mathcal{V}}(m) = \text{leak}_{\text{l}}$ *where*

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

  otherwise

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

Proof We prove the soundness and completeness of the synthesized R-Verifiers as exposed in Definition 4.2. We consider $\mu \in \mathcal{T}(\mathcal{G})$ s.t. $\delta_D(m^D_{\text{init}}, \mu) = m$.
$(\Rightarrow)$

- If $[\![\mathcal{V}]\!](\mu) = \text{noleak}$, *i.e.*, $\exists((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m$ such that $\forall i \in [0, K] : q_i \notin S$, and $\forall i \in [0, K-1] : b_i = \mathit{true}$. Let $\mu \in \mathcal{T}(\mathcal{G})$, such that $\delta_{\mathcal{V}}(q^{\mathcal{V}}_{\text{init}}, \mu) = m$. If $|\mu| \geq K$ s.t. $\mu = \mu' \cdot \sigma_1 \cdots \sigma_K$, then according to Lemma 4.2, $\exists s = s' \cdot s_0 \cdots s_{K-1} \in [\![\mu]\!]_{\Sigma_{\text{o}}}$ with $\forall i \leq K-1 : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i \wedge q_0 \xrightarrow{s_0}_{\mathcal{G}} q_1 \cdots q_{K-1} \xrightarrow{s_{K-1}}_{\mathcal{G}} q_K$. Moreover, as $\forall i \in [0, K-1] : b_i = \mathit{true}$, we can choose s such that $\forall i \in [0, K-1] : s_i \in \mathit{Free}^S_K(\mathcal{G}(q_{i-1}))$. Overall $s \in \mathit{Free}^S_K(\mathcal{G})$ and $s \in [\![\mu]\!]_{\Sigma_{\text{o}}} \cap \mathit{Free}^S_K(\mathcal{G})$ which means that $\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}^K_S)$ (the case where $|\mu| < K$ is similar).
- If $[\![\mathcal{V}]\!](\mu) = \text{leak}_{\text{l}}$ for some $l \in [1, K]$, *i.e.*, $l = \mathit{min}\{l' \in [0, K] \mid \forall((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m, \exists i \leq l' : q_{K-i} \in S \vee b_{K-i} = \mathit{false}\}$. 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 $s \in [\![\mu]\!]_{\Sigma_{\text{o}}}, s = s' \cdot s_0 \cdots s_{K-1}$ with $\forall i \leq K-1 : P(s_i) = \sigma_i$. By definition, there exists $(q_i)_{0\leq i\leq K}$ such that $q^{\mathcal{G}}_{\text{init}} \xrightarrow{s'}_{\mathcal{G}} q_0 \xrightarrow{s_0}_{\mathcal{G}} q_1 \cdots \xrightarrow{s_{K-1}}_{\mathcal{G}} q_K$ and according to Lemma 4.1, there exists $(b_i)_{0\leq i\leq K-1}$ such that $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in m$. 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 that $|s - s' \cdot s_0 \cdots s_{K-i-1}| \leq l$, which gives us the expected result.
  - If $b_{K-i} = \mathit{false}$, meaning that $s_{K-i} \notin \mathit{Free}^S_1(\mathcal{G}(q_{K-i}))$, then $s" = s' \cdot s_0 . \cdots s_{K-i} \in \mathcal{L}_S(\mathcal{G})$ with $|s - s"|_{\Sigma_{\text{o}}} \leq l$, which gives us again the expected result.

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

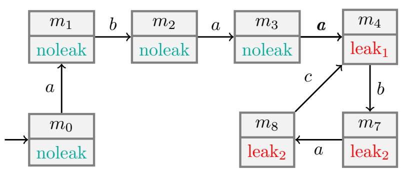
$(\Leftarrow)$

- If $\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}^K_S)$. It means that there exists $s \in [\![\mu]\!]_{\Sigma_{\text{o}}} \cap \mathit{Free}^S(\mathcal{G})$. Let $m = \delta_D(m^D_{\text{init}}, \mu)$. According to Lemma 4.1, $\exists s_1, \ldots, s_k \in \Sigma^* : s = s' \cdot s_1 \cdots s_k$ such that $\forall i \leq K : P_{\Sigma_{\text{o}}}(s_i) = \sigma_i$, there exists $((q_i)_{0\leq i\leq K}, (b_i)_{0\leq i\leq K-1}) \in \delta_D(m^D_{\text{init}}, \mu) : q_0 \xrightarrow{s_1}_{\mathcal{G}} q_1 \cdots q_{k-1} \xrightarrow{s_k}_{\mathcal{G}} q_k$. Now as $s \in \mathit{Free}^{\mathcal{G}}(S)$, it is easy to see that $\forall i \in [0, K-1], b_i = \mathit{true}$ and that $\forall i \in [0, K-1] : b_i = \mathit{true}$ and finally $[\![\mathcal{V}]\!](\mu) = \Gamma^{\mathcal{V}}(m) = \text{noleak}$.

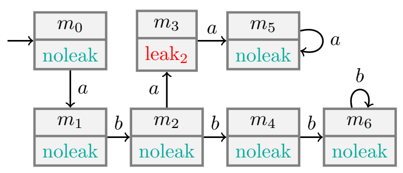
- If $\mu \in leak(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_S^K, l)$ for some $l \in [1, K]$. By hypothesis, we have that $[\![\mu]\!]_{\Sigma_o} \cap Free_l^S(\mathcal{G}) = \emptyset$ and $\forall l' < l : [\![\mu]\!]_{\Sigma_o} \cap Free_l^S(\mathcal{G}) \neq \emptyset$. Let $\delta_D(m_{\text{init}}^D, \mu) = m$ and $((q_i)_{0 \leq i \leq K}, (b_i)_{0 \leq i \leq K-1}) \in m$. According to Lemma 4.2, $\exists s_1, \ldots, s_k \in \Sigma^* : s = s' \cdot s_1 \cdots s_k$ 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 $s \notin Free_l^S(\mathcal{G})$, there exists $i \leq l$ such that either $q_{K-i} \in S$ or $s_{K-1} \in Free_1^S(\mathcal{G}(q_{k-i}))$, which would entail, by construction that $b_{K-i} = false$. Now for $l' < l$, there exists $s \in [\![\mu]\!]_{\Sigma_o} \cap Free_{l'}^S(\mathcal{G})$. For this s, we can make correspond an element $((q_i)_{0 \leq i \leq K}, (b_i)_{0 \leq i \leq K-1}) \in m$ such that $\forall i \in [0, l']$: $q_{K-i} \notin S \wedge \forall i \in [0, l'] : b_{K-i} = true$, which entails that l is the smallest number such that $\forall ((q_i)_{0 \leq i \leq K}, (b_i)_{0 \leq i \leq K-1}) \in m$, $\exists i \leq l : q_{K-i} \in S$ or $b_{K-i} = false$.

- If $\mu \in leak(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_S^K, 0)$, 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.

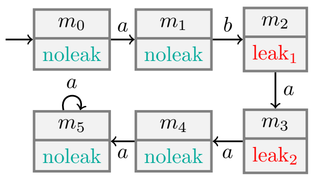
That is, the R-Verifier is producing noleak if there exists a trajectory of the system, compatible with the current observation, s.t. during the last K observations, the system has not visited any secret state. The R-Verifier produces leak_l if l is the minimum number of steps for which the secret has occurred for sure, that is if all the compatible execution sequences on the system have visited the secret, then $l \leq K$ is the minimum number of steps s.t. the secret occurred in the past.



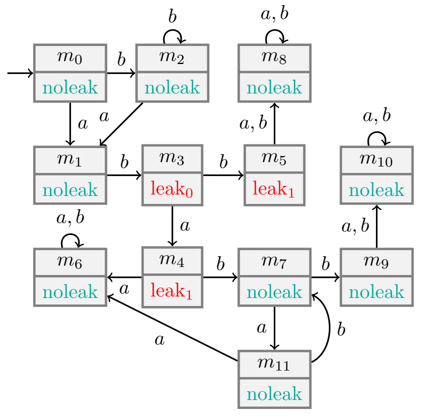
(a) For the 3-opacity of $\mathcal{G}_4$'s secret



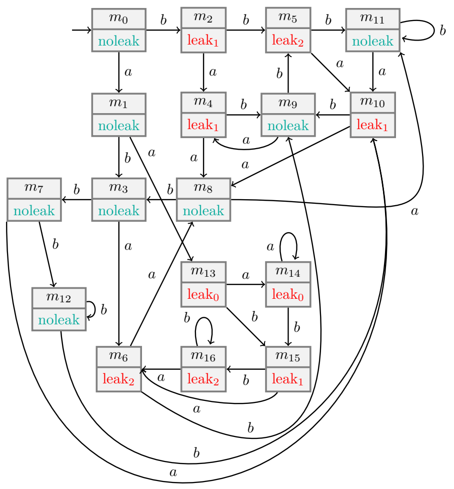
(b) For the 2-opacity of $\mathcal{G}_2$'s secret



(c) For the 2-opacity of $\mathcal{G}_3$'s secret



(d) For the 1-opacity of $\mathcal{G}_1$'s secret



(e) For the 2-opacity of $\mathcal{G}_5$'s secret

Figure 6: R-verifiers of K-strong opacity

EXAMPLE 4.4 (R-VERIFIERS OF K-STRONG OPACITY) *In Figure 6 are represented R-verifiers of K-strong opacity for the systems depicted in Figure 2. These monitors are built from their respective K-delay trajectory estimators.*

5 Enforcement of opacity at runtime

In this section, we consider a system $\mathcal{G}$[8], and we aim to build runtime enforcers, *i.e.*, a monitor dedicated to runtime enforcement, for the previously introduced notions of opacity.

[8]An underlying hypothesis in this section is that the system is live, *i.e.*, not deadlocked and always produces events, *e.g.*, a reactive system

5.1 Informal principle

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 Figure 7).

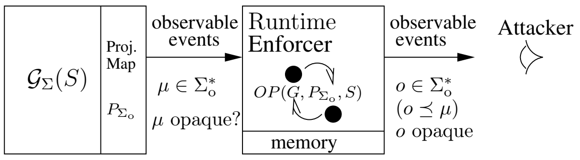
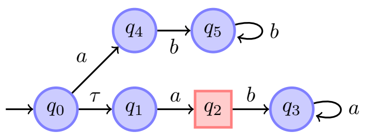


Figure 7: Enforcement of opacity at runtime

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

Example 5.1 (Principle of enforcing opacity) *Let us go back on the system $\mathcal{G}_2$ introduced in Example 3.2. 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
$\tau \cdot a$	$\epsilon \cdot a$	ϵ	a
$\tau \cdot a \cdot b$	$\epsilon \cdot a \cdot b$	ϵ	$a \cdot b$
$\tau \cdot a \cdot b \cdot a$	$\epsilon \cdot a \cdot b \cdot a$	a	$a \cdot b$
$\tau \cdot a \cdot b \cdot a \cdot a$	$\epsilon \cdot a \cdot b \cdot a \cdot a$	ϵ	$a \cdot b \cdot a \cdot a$
$\tau \cdot a \cdot b \cdot a^+$	$\epsilon \cdot a \cdot b \cdot a^+$	ϵ	$a \cdot b \cdot a^+$

Figure 8: Enforcement of opacity on a trace of $\mathcal{G}_2$

With a runtime enforcer, we will delay this sequence s.t., when the attacker determines that the system was in a secret secret, it is always more than K 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 Figure 8). When the system produces the sequence $\tau \cdot a$, the enforcer should not modify this sequence since a 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, 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 preserved. At last, the last received a and subsequent ones can be freely output by the monitor since they will not lead to a 2-weak opacity leakage.

5.2 Runtime Enforcers

We define generic notions of runtime enforcers which are special finite state transducers. The working principle of enforcement at runtime using monitors is depicted in Figure 7. Given an output from the system, the state of the underlying automaton evolves, and it produces an enforcement operation. The realization of enforcement operations on the input execution sequence modifies it so as to ensure the desired opacity as informally introduced in Example 5.1. In the opacity context, the modification consists in delaying the input sequence so as to delay enough the attacker observation to preserve the desired level of opacity on the system. In order to delay the input sequence, a runtime enforcer may save some of the input events together with some additional information inside an internal memory.

The following notion of runtime enforcers (R-Enforcer) is inspired from the variant introduced in [FFM09a] where R-Enforcers were defined in order to enforce linear temporal properties. Here we customize and adapt them for opacity.

Definition 5.1 (Enforcement operations *Ops* and memory) *Enforcement operations are aimed to operate a modification of the internal memory of the runtime enforcer and potentially produce an output. The internal memory is of size T and will be appointed to the monitor. Its set of configurations is denoted $\mathcal{M}(T)$, and will be specialized for enforcement of 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 $\subseteq 2^{(\Sigma_o \times \mathcal{M}(T)) \rightarrow (\Sigma_o^* \times \mathcal{M}(T))}$.*

Examples of enforcement operations consist of *e.g.*, memorizing input events or halting the input events. In Section 5.4, we will formally define the needed enforcement operations for opacity.

Definition 5.2 (Generic R-Enforcer (R-Enforcer(Ops))) *A* runtime enforcer *$\mathcal{E}$ is a special monitor equipped with a memory, i.e., a 6-tuple $(Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, Ops, \Gamma^{\mathcal{E}}, \mathcal{M}(T))$ defined relatively to a set of observable events Σ_o and parametrized 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. The complete function $\delta_{\mathcal{E}} : Q^{\mathcal{E}} \times \Sigma_o \rightarrow Q^{\mathcal{E}} \times Ops$ is the transition function.*

In the following we abbreviate $\delta_{\mathcal{E}}(q,a) = (q',\alpha)$ by $q \xrightarrow{a/\alpha}_{\mathcal{E}} q'$. Notions of *run* and *trace* are naturally transposed from their definitions for LTS: for a trace μ of length n $run(\mu,\mathcal{E}) = (q_0, \mu_0/\alpha_0, q_1) \cdot (q_1, \mu_1/\alpha_1, q_2) \cdots (q_{n-1}, \mu_{n-1}/\alpha_{n-1}, q_n)$. In the remainder of this section, $\mu \in \Sigma_o^*$ designates a (partial) trace of the system, and $(Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, Ops, \Gamma^{\mathcal{E}}, M(T))$ designates an R-Enforcer.

We formalize the way an R-Enforcer(Ops) reacts to an input sequence provided by a target system through the standard notions of *configuration* and *derivation*.

Definition 5.3 (Semantics of R-Enforcer(Ops)) *For an R-Enforcer(Ops) $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, Ops, \Gamma^{\mathcal{E}}, M(T))$, a* configuration *is a 3-tuple $(q,\mu,c) \in Q^{\mathcal{E}} \times \Sigma_o^* \times \mathcal{M}(T)$ where q denotes the current control state, μ is a (partial) trace produced by the system (input to be read by the enforcer), 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[9] *$o \in \Sigma_o^*$, and we note $(q,\mu,c) \overset{o}{\hookrightarrow} (q',\mu',c')$ if and only if $\mu = a \cdot \mu' \wedge q \xrightarrow{a/\alpha}_{\mathcal{E}} q' \wedge \Gamma^{\mathcal{E}}(q') = \alpha \wedge \alpha(a,c) = (o,c')$.*
- *A configuration C' is* derivable in several steps *from a configuration C and* produces the output *$o \in \Sigma_o^*$, and we note $C \overset{o}{\Longrightarrow}_{\mathcal{E}} C'$, if and only if there exists $k \geq 0$ and configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0$, $C' = C_k$, $C_i \overset{o_i}{\hookrightarrow} C_{i+1}$ for all $i \in [0,k[$, and $o = o_0 \cdot o_1 \cdots o_{k-1}$.*

Definition 5.4 (Sequence transformation) *We define the transformation performed by an R-Enforcer, with set of enforcement operations Ops, while reading an input sequence $\mu \in \Sigma_o^*$ (produced by a system $\mathcal{G}_\Sigma$) and producing an output sequence $o \in \Sigma_o^*$. The relation $\Downarrow_{\mathcal{E}} \subseteq \Sigma_o^* \times \Sigma_o^*$ is defined as follows where ϵ refers to ϵ_{Σ_o}:*

- *$\epsilon \Downarrow_{\mathcal{E}} \epsilon$,*
- *$(q,\mu) \Downarrow_{\mathcal{E}} o$, if $\exists q' \in Q^{\mathcal{E}}, \exists c,c' \in \mathcal{M}(T) : (q,\mu,c) \overset{o}{\Longrightarrow}_{\mathcal{E}} (q',\epsilon,c')$*
- *$\mu \Downarrow_{\mathcal{E}} o$ if $\exists q \in Q^{\mathcal{E}}, \exists c, \in \mathcal{M}(T) : (q^{\mathcal{E}}_{\text{init}}, \mu, \epsilon_{\mathcal{M}}) \overset{o}{\Longrightarrow}_{\mathcal{E}} (q, \epsilon_{\Sigma_o}, c)$.*

The empty sequence ϵ_{Σ_o} is not modified by $\mathcal{E}$, when the system does not produce any event. The sequence $\mu \in \Sigma_o^*$ is transformed by $\mathcal{E}$ from the state $q \in Q^{\mathcal{E}}$ and the memory configuration c into the sequence $o \in \Sigma_o^*$, if there exists a derivation starting from a configuration which state is q, and producing o. The trace $\mu \in \mathcal{T}(\mathcal{G})$ is transformed by $\mathcal{E}$ into the trace $o \in \mathcal{T}(\mathcal{G})$, when the trace is transformed from the initial state of the R-Enforcer(Ops), starting with an empty memory.

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

5.3 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 5.5 (Prefixes that are safe to output) *For simple opacity* $\mathsf{OP_0}$*, a trace* $\mu \in \mathcal{T}(\mathcal{G})$ *produced by the system, we say that it is safe to produce the output* $\mu' \preceq \mu$*, noted* $\mathit{safe}_{\mathsf{OP_0}}(\mu, \mu')$ *if* $\mu' \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0}) \vee \mu' \prec \mu$.

For a K*-step based notion of opacity* $\mathsf{OP_K} \in \{\mathsf{OP^K_W}, \mathsf{OP^K_S}\}$*, a trace* $\mu \in \mathcal{T}(\mathcal{G})$ *produced by the system, we say that it is safe to produce the output* $\mu' \preceq \mu$*, noted* $\mathit{safe}_{\mathsf{OP^K_S}}(\mu, \mu')$ *if* $\mu' \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}) \vee (\exists k \leq K : \mu' \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, k) \wedge |\mu| - |\mu'| \geq K - k)$.

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

Note that when it is safe to produce a given sequence, then all prefixes of this sequence are safe to produce. That is, for $\mathsf{OP} \in \{\mathsf{OP_0}, \mathsf{OP^K_W}, \mathsf{OP^K_S}\}$: $\forall \mu, \mu' \in \mathcal{T}(\mathcal{G}) : [\mathit{safe}_{\mathsf{OP}}(\mu, \mu') \Rightarrow \forall \mu'' \prec \mu' : \mathit{safe}(\mu, \mu'')]$.

Furthermore, by convention, we will only consider systems for which it is safe to produce ϵ (*i.e.*, nothing), *i.e.*, when all sequences of $[\![\epsilon]\!]_{\Sigma_o}$ are not in the secret. One may remark that for a given sequence, when the system is alive, there always exists one of its extensions s.t. this sequence is safe in output, *i.e.*, $\forall \mu \in \mathcal{T}(\mathcal{G}), \exists \mu' \in \mathcal{T}(\mathcal{G}) : \mu \preceq \mu' \wedge \mathit{safe}_{\mathsf{OP}}(\mu, \mu')$, *e.g.*, any μ' s.t. $|\mu' - \mu| > K$. Moreover, the set of sequences that lead a given sequence to be safe is extension-closed, *i.e.*, $\forall \mu' \in \mathcal{T}(\mathcal{G}), (\exists \mu \in \mathcal{T}(\mathcal{G}) : \mu' \preceq \mu \wedge \mathit{safe}_{\mathsf{OP}}(\mu, \mu')) \Rightarrow \forall \mu'' \in \mathcal{T}(\mathcal{G}) : \mu \preceq \mu'' \Rightarrow \mathit{safe}_{\mathsf{OP}}(\mu'', \mu)$.

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. Usually, property enforcement by an enforcer is defined as the conjunction of the two following constraints [LBW05, FFM09b] that we express here in the context of opacity enforcement. Those constraints are expected on the enforcers we aim to synthesize.

soundness: the output sequence should preserve the opacity of the system;

transparency[10]**:** the input sequence should be modified *in a minimal way*, namely if it already preserves opacity it should remain unchanged, otherwise its *longest prefix* preserving opacity (*i.e.*, that is safe to produce) should be issued.

On Example 5.1, 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 $a \cdot b \cdot a$ (instead of any prefix) when $\mathcal{G}_2$ produces $\tau \cdot a \cdot b \cdot a$.

Expected properties for runtime enforcers. We now give the formal definition of opacity-enforcement by an R-Enforcer(Ops). The notion of enforcement relates the input sequence produced by the program fed to the R-Enforcer(Ops) and the output sequence allowed by the R-Enforcer(Ops) (safe w.r.t. opacity).

Definition 5.6 (Enforcement of opacity by a monitor) *A runtime enforcer* $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, \mathit{Ops}, \Gamma^{\mathcal{E}}, M(T))$ *enforces the opacity* $\mathsf{OP} \in \{\mathsf{OP_0}, \mathsf{OP^K_W}, \mathsf{OP^K_S}\}$ *of a secret* S *w.r.t.* P_{Σ_o} *on a system* $\mathcal{G}$ *if* $\forall \mu \in \mathcal{T}(\mathcal{G}), \exists o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow$

$$\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \Rightarrow o = \mu \tag{6}$$

$$\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}) \Rightarrow o = \mathit{max}_{\preceq}\{\mu' \in \mathit{Pref}(\mu) \mid \mu' \in \mathit{safe}_{\mathsf{OP}}(\mu, \mu')\} \tag{7}$$

[10]This notion corresponds somehow to the notion of maximality in supervisory control theory [DDM10].

The constraints (6) and (7) ensure soundness and transparency of $\mathcal{E}$: (6) will ensure that if μ already preserved the opacity then it is not transformed, (7) will ensure that if μ reveals the opacity, the monitor outputs its longest prefix that is safe regarding opacity.

Soundness is due to the fact that the produced sequence o always preserves the opacity. Transparency is ensured by the fact that correct execution sequences (*i.e.*, preserving opacity) are not changed, and incorrect ones are restricted to their longest correct prefix. One may notice that enforcement is only suitable when it is always safe for a runtime enforcer to produce ϵ_{Σ_o}.

Using the formal definition of opacity enforcement and the definition of safe sequences, a sound and transparent runtime enforcer always produces maximal safe sequences, as stated in the following property.

Property 5.1 *For a runtime enforcer $\mathcal{E}$ enforcing the opacity $\mathsf{OP} \in \{\mathsf{OP}_0, \mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}\}$, verifying the soundness and transparency constraints of Definition 5.6, its input sequence μ and output sequence o are s.t.:*
$\forall \mu \in \mathcal{T}(\mathcal{G}), \forall o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow \big(\mathit{safe}_{\mathsf{OP}}(\mu, o) \wedge \forall o \prec o' \preceq \mu : \neg \mathit{safe}_{\mathsf{OP}}(\mu, o')\big)$

Most of the previous approaches (*e.g.*, [LBW05, FFM09b]) on property enforcement used runtime enforcers 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 monitor has a given size[11]. Besides the **soundness** and **transparency** constraints, we add the following one:

> **do-not-overflow:** to enforce opacity, the size of the sequence of events memorized by the monitor 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, projection maps and secrets s.t. opacity is enforceable using runtime monitors. We first state the existence of R-Enforcers for the enforcement of opacity relying on the characterization of opacity preservation as finitary properties (see Sections 3).

Corollary 5.1 (Existence of sound and transparent R-Enforcer(Ops)) *For a given system $\mathcal{G}$, a projection map P_{Σ_o}, if it is safe to produce ϵ (i.e., when $[\![\epsilon]\!] \not\subseteq \mathcal{L}_{S_\mathcal{G}}(\mathcal{G})$), then there exists a sound and transparent R-Enforcer(Ops) s.t. it enforces the opacity of S on $\mathcal{G}$ w.r.t. P_{Σ_o}. More specifically:*

- *with a memory of size* 1 *for simple opacity* (OP_0),
- *with a bounded memory for K-step based opacity* ($\mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}$).

Proof These are direct consequences of the following fact: any observation can be released by a runtime enforcer when it is followed by K observations. Moreover, opacity preservation by observable traces of the system can be expressed as finitary properties. Then, existence of sound and transparent runtime enforcers is a consequence of the results in [FFM09b].

Now, it turns out that the existence of a runtime enforcer for K-step based opacity relies on the satisfiability of the **do-not-overflow** constraint. Thus, we state an enforcement criterion for the **do-not-overflow** constraint. As the condition for the existence of an R-Enforcer for simple opacity is straightforward, we will focus on K-step based opacity in the remainder of this subsection.

proposition 5.1 (Enforcement criterion of K-step based opacity) *Given a system $\mathcal{G}$, a K-step based notion of opacity (i.e., $\mathsf{OP}_\mathsf{K}^\mathsf{W}$ or $\mathsf{OP}_\mathsf{K}^\mathsf{S}$) w.r.t. a projection map P_{Σ_o} and a secret S, which violation by a trace μ is given through the predicate $\mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_\mathsf{K})$, $\mathsf{OP}_\mathsf{K} \in \{\mathsf{OP}_\mathsf{K}^\mathsf{W}, \mathsf{OP}_\mathsf{K}^\mathsf{S}\}$, the opacity of the secret S is enforceable by a sound and transparent R-Enforcer(Ops) with memory size T if:*

$$\mathit{max}_{\mu \in \mathcal{T}(\mathcal{G})}\{\mathit{min}\{|\mu \setminus o| \mid o \preceq \mu \wedge \mathit{safe}_{\mathsf{OP}_\mathsf{K}}(\mu, o)\} \leq T$$

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

Proof This proposition simply states that the maximal number of elements to be memorized should be less than the allocated memory size, for all traces of the system. Indeed, for a trace $\mu \in \mathcal{T}(\mathcal{G})$, $\{o \preceq \mu \mid \mathit{safe}(\mu, o)\}$ is the set of sequences that can be produced by a sound runtime enforcer. For $\mu \in \mathcal{T}(\mathcal{G})$, a transparent enforcer will produce $\mathit{max}\{o \preceq \mu \mid \mathit{safe}_{\mathsf{OP_K}}(\mu, o)\}$, and will have $\mathit{min}\{|\mu \setminus o| \mid o \preceq \mu \wedge \mathit{safe}_{\mathsf{OP_K}}(\mu, o)\}$ elements in its memory. Naturally, the memory of such a runtime enforcer does not overflow if $\mathit{min}\{|\mu \setminus o| \mid o \preceq \mu \wedge \mathit{safe}_{\mathsf{OP_K}}(\mu, o)\} \leq T$.

Remark 5.1 *When we are trying to enforce K-step based opacity and the size of the allocated memory is greater than K, there always exists a trivial enforcer delaying every event by K units of time. Note that this enforcement monitor is not transparent in general.*

The previous enforcement criterion is not doable in practice, as it is not computable in the general case. Thus, we will give a more practical enforcement criterion suitable to determine whether the opacity of a secret is enforceable with an R-Enforcer with a given memory size. This criterion uses the K-delay state estimator[12] associated to a system. To each state of the K-delay state estimator, we have seen that it is possible to determine the opacity leakage. Intuitively, the reasoning is as follows. If we are considering a K-step based notion of opacity and we reach a state in the K-delay state 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 state estimators uses the following lemma, which is a direct consequence of Lemmas 4.1 and 4.2.

Lemma 5.1 (States of K-delay state estimators and opacity leakage) *Given a system $\mathcal{G}$, a K-step based notion of opacity $\mathsf{OP_K} \in \{\mathsf{OP^W_K}, \mathsf{OP^S_K}\}$ w.r.t. a projection map P_{Σ_o} and a secret S, the states of the K-delay state estimator $D = (Q^D, q^D_{\text{init}}, \Sigma_o, \delta_D)$ are s.t.:*

$\forall \mu_1, \mu_2 \in \mathcal{T}(\mathcal{G}) : \delta_D(\mu_1, m^D_{\text{init}}) = \delta_D(\mu_2, m^D_{\text{init}})$
$\Rightarrow (\exists k \in [0, K] : \mu_1, \mu_2 \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, k)) \vee \mu_1, \mu_2 \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$

This lemma states that, for a given state in the state estimator, all traces ending in this state reveal or preserve opacity in the same way.

For a system $\mathcal{G}$ on which we aim to enforce a K-step based opacity $\mathsf{OP_K} \in \{\mathsf{OP^W_K}, \mathsf{OP^S_K}\}$ of a secret, and its associated K-delay state estimator $D = (Q^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$, to each state $m \in Q^D$, 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. That is, $\forall m \in Q^D : \mathit{hold}_{\mathsf{OP_K}}(m) = K + 1 - l_m$ when there exists $l_m \in [0, K]$ is s.t. $\forall \mu \in \mathcal{T}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m \Rightarrow \mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l_m)$ and $\mathit{hold}_{\mathsf{OP_K}}(m) = 0$ otherwise ($\forall \mu \in \mathcal{T}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m \Rightarrow \mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$). Equivalently, using a R-Verifier for $\mathsf{OP_K}$, $\mathcal{V} = (Q^{\mathcal{V}}, q^{\mathcal{V}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{V}}, \mathbb{B}^K_{\mathsf{OP}}, \Gamma^{\mathcal{V}})$ for $\mathcal{G}$ and K-step based opacity, $\forall \mu \in \mathcal{T}(\mathcal{G}) : \mathit{hold}_{\mathsf{OP_K}}(\mu) = K + 1 - \Gamma(\delta(q^{\mathcal{V}}_{\text{init}}, \mu))$. Thus, synthesis of R-Enforcers will rely on the synthesis of R-Verifiers.

Property 5.2 (Enforcement criterion using K-delay state estimators) *Given a system $\mathcal{G}$, a K-step based notion of opacity $\mathsf{OP_K} \in \mathsf{OP^W_K}$ or $\mathsf{OP^S_K}$ w.r.t. a projection map P_{Σ_o}, and a secret S. Let $D = (Q^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ be the K-delay state estimator[13] associated to $\mathcal{G}$. The opacity of the secret S is enforceable by an R-Enforcer with memory size T iff:*

$$\mathit{max}\{\mathit{hold}_{\mathsf{OP_K}}(m) \mid m \in Q^D\} \leq T \tag{8}$$

Proof This is a direct consequence of Lemma 5.1 and the definition of $\mathit{safe}(\mu, \mu')$. Indeed, (8) $\Leftrightarrow$ $\mathit{max}\{K{+}1{-}l_m \mid m \in Q^D\} \leq T$, with l_m s.t. $\forall \mu \in \mathcal{T}(\mathcal{G}) : \delta_D(m^D_{\text{init}}, \mu) = m \Rightarrow \mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l_m)$. Furthermore, using Lemma 5.1, one can notice that the previous proposition is equivalent to

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

Moreover, from the definition of safe, for a trace $\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}, l)$, one can notice that $K + 1 - l = \mathit{min}\{|\mu'| - |\mu| \mid \mu \preceq \mu' \wedge \mathit{safe}(\mu', \mu)\}$ with the convention that $l = K + 1$ when $\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$. Then (8) $\Leftrightarrow \mathit{max}_{\mu \in \mathcal{T}(\mathcal{G})}\{\mathit{min}\{|\mu'| - |\mu| \mid \mu \preceq \mu' \wedge \mathit{safe}(\mu', \mu)\} \leq T$.

A consequence of the previous property is that the enforcement of a K-step notion of opacity with a memory of a given size is decidable.

[12] Equivalently, we can use K-delay trajectory estimator.
[13] A similar criterion can be defined on K-delay trajectory estimators.

5.4 Enforcement monitor synthesis

We now tackle the problem of synthesizing runtime enforcers so as to ensure opacity. We first start by defining the enforcement primitives endowed to the monitors we consider. Then, we show how we synthesize R-Enforcers from K-delay estimators (for states and trajectories).

The memory of runtime enforcers for opacity is a sorted list whose elements are pairs consisting of an observable event and an integer. The set of possible configurations of the memory is thus $\mathcal{M}(T) = (\Sigma_o \times \mathbb{N})^T$. When an element (σ, d) with $\sigma \in \Sigma_o, d \in \mathbb{N}$ is inside the memory, it means that the event σ has to be retained d units of time before being released by the monitor so as to ensure opacity.

Enforcement operations. We define enforcement operations specifically used by our runtime enforcers dedicated to opacity. First, we need to define some auxiliary operations. In the following, we will use the following notations for the runtime enforcer's memory. For a memory configuration $(\sigma, d) \in \mathcal{M}(T)$, $(\sigma, d).delay = d$. For two memory configurations c, c' s.t. $c = ((\sigma^1, d^1) \cdots (\sigma^t, d^t))$, $c' = ((\sigma^1, d^1) \cdots (\sigma^{t'}, d^{t'}))$ with $t' \leq t$, (*i.e.*, c' is a prefix of c):

- $c_{\downarrow\Sigma_o} = d^1 \cdots d^t$,
- $(c \setminus c')_{\downarrow\Sigma_o}$ is ϵ_{Σ_o} if $c = c'$ and the sequence of events $\sigma^{t'+1} \cdots \sigma^t$ otherwise.

Definition 5.7 (Auxiliary operations) *The enforcement operations that we consider use the two auxiliary operations* free : $\mathcal{M}(T) \to \mathcal{M}(T)$ *and* delay : $\mathcal{M}(T) \to \mathcal{M}(T)$. *Given* $c = (\sigma^1, d^1) \cdots (\sigma^t, d^t) \in \mathcal{M}(T)$, *with* $t \leq T$.

- $\text{delay}(c) = (\sigma^1, d^1 - 1) \cdots (\sigma^t, d^t - 1)$ *with* $t \leq T$;
- $\text{free}(c) = (\sigma^i, d^i) \cdots (\sigma^t, d^t)$, *with* $1 \leq i \leq t$ *and*
  - $\forall j \in [1, i[: c_j.delay \leq 0$,
  - $\forall j \in [i, t] : c_j.delay > 0$.

The auxiliary operation delay consists in decrementing the delays of each elements 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 auxiliary operation free consists in realizing the events that now do not leak opacity. Those events are those for which the associated delay is negative or null.

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

Definition 5.8 (Enforcement Operations) *The enforcement operations that we consider are defined as follows where* $\sigma \in \Sigma_o, c = (\sigma^1, d^1) \cdots (\sigma^t, d^t) \in \mathcal{M}(T)$.

- $\text{halt}(\sigma, c) = (\epsilon_{\Sigma_o}, \epsilon_{\mathcal{M}})$;
- *For* $d \in [1, K]$: $\text{store\_}d(\sigma, c) = (o, (\sigma, d) \cdot c')$, *with* $c' = \text{free} \circ \text{delay}(c)$ *and* $o = (c \setminus c')_{\downarrow\Sigma_o}$;
- $\text{dump}(\sigma, c) = (o, c'')$ *with*
  - $c' = \text{free} \circ \text{delay}(c)$,
  - $o = \sigma \cdot c_{\downarrow\Sigma_o}$ *if* $c' = \epsilon_{\mathcal{M}}$ *and* $(c \setminus c')_{\downarrow\Sigma_o}$ *else*,
  - $c'' = (\sigma, 0) \cdot c \setminus c'$ *if* $c' \neq \epsilon_{\mathcal{M}}$ *and* $\epsilon_{\mathcal{M}}$ *else;*
- $\text{off}(\sigma, c) = \text{dump}(\sigma, c)$.

The enforcement operation halt is issued by a runtime enforcer when the considered notion of opacity is irremediably revealed. Thus, the underlying program should be stopped. This operation consists in ignoring the submitted event, erasing the memory, and halting the underlying system.

For $d \in [1, K]$, the enforcement operation store_d is issued when the event submitted to the runtime enforcer should be delayed by d unit(s) of time in order to the opacity to be preserved. This operation consists in first releasing the events preserving the opacity (using free ∘ delay) and appending the event with the needed delay to the memory.

The enforcement operation dump is issued by a runtime enforcer when the event submitted to the runtime enforcer does not reveal the opacity. The event is submitted to the enforcement but is not inevitably produced in output. The runtime 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 sequence.

Then enforcement operation off is issued by a runtime enforcer when opacity will not be revealed whatever are the future observable events produced by the system. Thus, the runtime enforcer can be switched off. Though the off has the same definition than the dump operation, such an enforcement operation is useful in practice since it reduces the overhead induced by the runtime enforcer.

Synthesis of runtime enforcers. Synthesis of runtime enforcers is based on K-delay state estimators and K-delay trajectory estimators.

PROPERTY 5.3 *Given a plant* $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$*, a secret* $S \subseteq Q^{\mathcal{G}}$ *and* $\Sigma_{\text{o}} \subseteq \Sigma$*, the R-Enforcer* $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_{\text{o}}, \delta_{\mathcal{E}}, \{\text{store\_1}, \text{dump}, \text{off}\}, \Gamma^{\mathcal{E}}, M(T))$ *built from the* 1*-delay state estimator*[14] $D = (M^D, m^D_{\text{init}}, \Sigma_{\text{o}}, \delta_D)$ *of* $\mathcal{G}$ *where:*

- $Q^{\mathcal{E}} = M^D, q^{\mathcal{E}}_{\text{init}} = q^D_{\text{init}}, \delta_{\mathcal{E}} = \delta_D$,
- $\Gamma^{\mathcal{E}} : Q^{\mathcal{E}} \rightarrow \mathbb{B}^K_{\mathsf{OP}}$ *defined by*
  - $\Gamma^{\mathcal{E}}(m) = \text{off}$ *if* $m(0) \notin 2^S \wedge \forall m' \in \mathit{Reach}_D(m) : m'(0) \notin 2^S$
  - $\Gamma^{\mathcal{E}}(m) = \text{dump}$ *if* $m(0) \notin 2^S \wedge \exists m' \in \mathit{Reach}_D(m) : m'(0) \in 2^S$
  - $\Gamma^{\mathcal{E}}(m) = \text{store\_1}$ *if* $m(0) \in 2^S$

enforces the simple opacity of S *w.r.t.* $P_{\Sigma_{\text{o}}}$ *on* $\mathcal{G}$*.*

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

$$\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}_0) \Rightarrow o = \mu \tag{9}$$

$$\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}_0) \Rightarrow o = \mathit{max}\{\mu' \in \mathcal{T}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \mu' \in \mathit{safe}_{\mathsf{OP}_0}(\mu, \mu')\} \tag{10}$$

Let us consider $\mu \in \mathcal{T}(\mathcal{G})$, the proof is conducted by induction on $|\mu|$.
If $|\mu| = 1$, then $\exists \sigma \in \Sigma_{\text{o}} : \mu = \sigma$. The run of μ on $\mathcal{E}$ can be expressed $\mathit{run}(\mu, \mathcal{E}) = (q^{\mathcal{E}}_{\text{init}}, \sigma/\alpha, q_1)$ with $q_1 \in Q^{\mathcal{E}}, \Gamma^{\mathcal{E}}(q_1) = \alpha$. The R-Enforcer's evolution of configurations is $(q^{\mathcal{E}}_{\text{init}}, \sigma, \epsilon_{\mathcal{M}}) \overset{o}{\hookrightarrow} (q, \epsilon_{\Sigma_{\text{o}}}, m)$ with $\alpha(\sigma, \epsilon_{\mathcal{M}}) = (o, m)$. Let us distinguish according to whether $\sigma \in \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}_0)$ or not.

- If $\sigma \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}_0)$, then we use the correctness of R-Verifiers synthesized from K-delay state estimators (Proposition 4.2). The state m_1 corresponding to q_1 in the corresponding K-delay state estimator is s.t. $m_1(0) \notin 2^S$. Then, using the definition of R-Enforcers synthesis from K-delay state estimators, we have that $\alpha \in \{\text{dump}, \text{off}\}$. Using the definition of enforcement operations, we have: $\text{free} \circ \text{delay}(\epsilon_{\mathcal{M}}) = \epsilon_{\mathcal{M}}$, $o = \sigma \cdot (\epsilon_{\mathcal{M}})_{\downarrow \Sigma_{\text{o}}} = \sigma$, $m = \epsilon_{\mathcal{M}}$. Thus, we find (9).
- If $\sigma \in \mathit{leak}(\mathcal{G}, P_{\Sigma_{\text{o}}}, S, \mathsf{OP}_0)$, then similarly following from the correctness of R-Verifier synthesized from K-delay state estimators (Proposition 4.2), we have that $\alpha = \text{store\_1}$. Similarly, we can find that $o = \epsilon_{\Sigma_{\text{o}}}$ and $m = (\sigma, 1)$. Furthermore, as $\mathit{safe}_{\mathsf{OP}_0}(\sigma, \epsilon_{\Sigma_{\text{o}}})$, we have (10).

Let us consider $\mu \in \Sigma_{\text{o}}^*$ s.t. $|\mu| = n$ for which (9) and (10) 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

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

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

[14]The enforcer can be equivalently built from $\mathit{Det}_{\Sigma_{\text{o}}}(\mathcal{G})$ (see Remark 4.1). In the general case, $\mathit{Det}_{\Sigma_{\text{o}}}(\mathcal{G})$ is smaller than the corresponding 1-delay state estimator. Here we choose to present synthesis from state estimator for uniformity of notations.

- If $\mu \cdot \sigma \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$, then following the reasoning for the induction basis, we know that $\alpha \in \{\text{off}, \text{dump}\}$. Using the induction hypothesis, we have that there exists $o \in \mathcal{T}(\mathcal{G})$ s.t. $\sigma_0 \cdots \sigma_{n-1} \Downarrow_{\mathcal{E}} o$ and the constraints (9) and (10) hold.

  Now we distinguish according to whether $\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$ or not.

  – If $\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$, from (9), we know that $o = \mu$. Then, μ induces the following evolution of configurations for $\mathcal{E}$:

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

  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:

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

  Then, we deduce $\mu \cdot \sigma \Downarrow_{\mathcal{E}} \mu \cdot \sigma$, which gives us (9).

  – Else ($\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_0})$), from (9), we know that $o = \mathit{max}\{\mu' \in \mathcal{T}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \mathit{safe}_{\mathsf{OP_0}}(\mu, \mu')\}$, *i.e.*, using the definition of $\mathit{safe}_{\mathsf{OP_0}}$, $o = \sigma_0 \cdots \sigma_{n-2}$. Then, μ induces the following evolution of configurations for $\mathcal{E}$:

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

  with $o_0 \cdots o_{n-1} = o = \sigma_0 \cdots \sigma_{n-2}$, and $o_{n-1} = \epsilon_{\Sigma_o}$. Since $\alpha = \text{store\_1}$, $\alpha(\sigma, (\sigma_{n-1}, 1)) = (\sigma_{n-1}, \sigma)$. Then, we deduce the following evolution of configurations:

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

  Then, we deduce $\mu \cdot \sigma \Downarrow_{\mathcal{E}} \mu$. From $\mathit{safe}_{\mathsf{OP_0}}(\mu \cdot \sigma, \mu)$ and $\neg \mathit{safe}_{\mathsf{OP_0}}(\mu \cdot \sigma, \mu \cdot \sigma)$, *i.e.*, $\mu = \mathit{max}(\mu' \in \mathcal{T}(\mathcal{G}) \mid \mu' \preceq \mu \cdot \sigma \wedge \mathit{safe}_{\mathsf{OP_0}}(\mu \cdot \sigma, \mu')\}$, we deduce (10).

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

Now, we synthesize R-Enforcers for K-step based opacity from K-delay state estimators (for K-weak opacity) and trajectory estimators (for K-strong opacity).

Property 5.4 *Given a plant $\mathcal{G} = (Q^{\mathcal{G}}, q^{\mathcal{G}}_{\text{init}}, \Sigma, \delta_{\mathcal{G}})$, a secret $S \subseteq Q^{\mathcal{G}}$ and $\Sigma_o \subseteq \Sigma$, the R-Enforcer $\mathcal{E} = (Q^{\mathcal{E}}, q^{\mathcal{E}}_{\text{init}}, \Sigma_o, \delta_{\mathcal{E}}, \{\text{halt}, \text{store\_}d, \text{dump}, \text{off} \mid d \in [1, K]\}, \Gamma^{\mathcal{E}}, M(T))$, which memory is of size T, built from the K-delay state or trajectory estimator $D = (M^D, m^D_{\text{init}}, \Sigma_o, \delta_D)$ of $\mathcal{G}$ (along with the associated function $\mathit{hold}_{\mathsf{OP_K}}()$) where:*

- $Q^{\mathcal{E}} = M^D, q^{\mathcal{E}}_{\text{init}} = q^D_{\text{init}}, \delta_{\mathcal{E}} = \delta_D$,
- $\Gamma^{\mathcal{E}} : Q^{\mathcal{E}} \to \mathbb{B}^K_{\mathsf{OP}}$ *defined by*
  – $\Gamma^{\mathcal{E}}(m) = \text{off}$ *if* $\mathit{hold}_{\mathsf{OP_K}}(m) = 0 \wedge \forall m' \in \mathit{Reach}_D(m) : \mathit{hold}_{\mathsf{OP_K}}(m') = 0$,
  – $\Gamma^{\mathcal{E}}(m) = \text{dump}$ *if* $\mathit{hold}_{\mathsf{OP_K}}(m) = 0 \wedge \exists m' \in \mathit{Reach}_D(m) : \mathit{hold}_{\mathsf{OP_K}}(m') = 0$,
  – $\Gamma^{\mathcal{E}}(m) = \text{store\_}d$ *if* $\exists d \in [1, T] : \mathit{hold}_{\mathsf{OP_K}}(m) = d$,
  – $\Gamma^{\mathcal{E}}(m) = \mathit{halt}$ *if* $\exists d > T : \mathit{hold}_{\mathsf{OP_K}}(m) = d$,

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

Proof We have to prove that, for $\mathsf{OP_K} \in \{\mathsf{OP^W_K}, \mathsf{OP^S_K}\}$: $\forall \mu \in \mathcal{T}(\mathcal{G}), \exists o \preceq \mu : \mu \Downarrow_{\mathcal{E}} o \Rightarrow$

$$\mu \notin \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}) \Rightarrow o = \mu \tag{11}$$

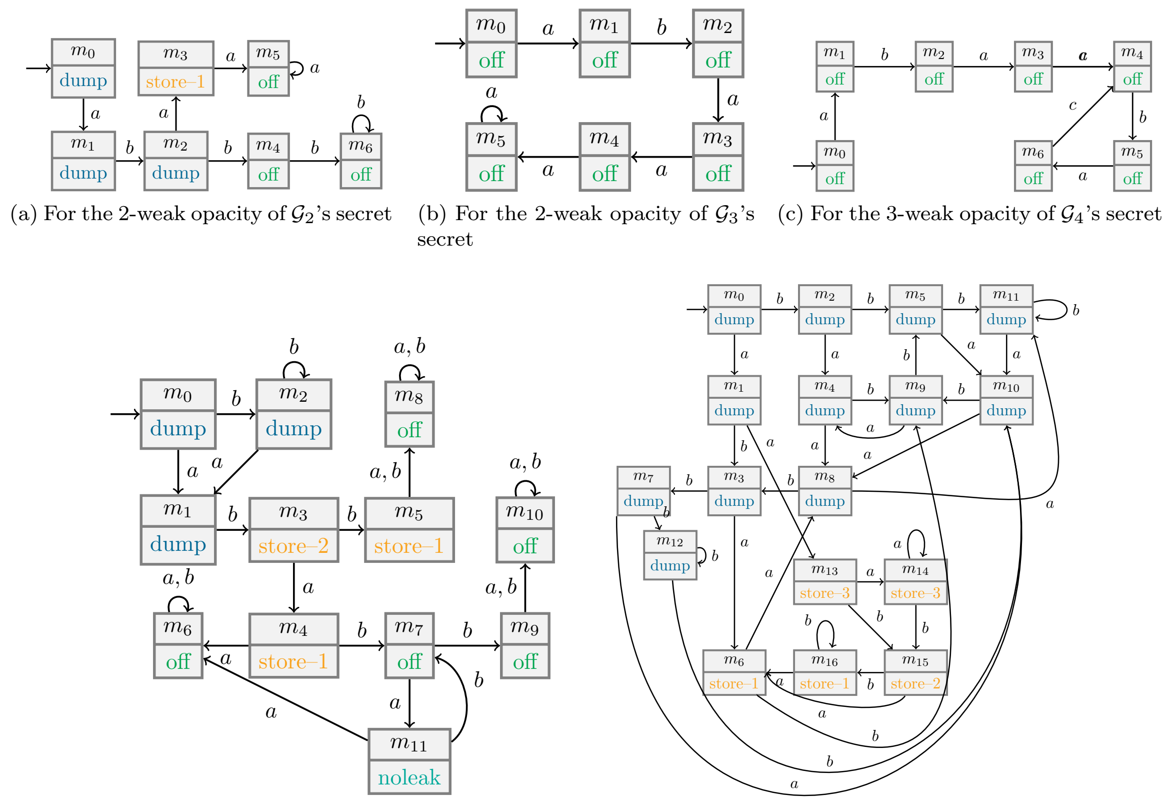
$$\mu \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K}) \Rightarrow o = \mathit{max}\{\mu' \in \mathcal{T}(\mathcal{G}) \mid \mu' \preceq \mu \wedge \mu' \in \mathit{safe}_{\mathsf{OP_K}}(\mu, \mu')\} \tag{12}$$

Let us consider $\mu \in \mathcal{T}(\mathcal{G})$, the proof is conducted by induction on $|\mu|$. Moreover, the proof is done for $\mathsf{OP_K}$, a K-step based notion of opacity (independently from whether it is weak or strong), since we will use the function $\mathit{hold}_{\mathsf{OP_K}}()$ for the state of the underlying estimator (for states and trajectories) 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 $\mathit{run}(\mu, \mathcal{E}) = (q^{\mathcal{E}}_{\text{init}}, \sigma/\alpha, q_1)$ 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 $\sigma \in \mathit{leak}(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP_K})$ or not.

- If $\sigma \notin leak(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_\mathsf{K})$, then we use the correctness of R-Verifiers synthesized from K-delay state and trajectory estimators (Proposition 4.2). Using the definition and the properties of the function *hold* (Section 5.3, "When is the opacity of a secret enforceable?"), the state m_1 corresponding to q_1 in the corresponding K-delay estimator is s.t. $hold_{\mathsf{OP}_\mathsf{K}}(m_1) = hold_{\mathsf{OP}_\mathsf{K}}(\sigma) = 0$. Then, using the definition of R-Enforcers synthesis, we have that $\alpha \in \{\text{dump}, \text{off}\}$. 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 (11).
- If $\exists k \in [1, K] : \sigma \in leak(\mathcal{G}, P_{\Sigma_o}, S, \mathsf{OP}_\mathsf{K}, k)$, then necessarily $k = 1$. Similarly, following from the correctness of R-Verifiers synthesized from K-delay state and trajectory estimators (Propositions 4.2 and 4.3) and the definition of $hold_{\mathsf{OP}_\mathsf{K}}$, we have that $hold_{\mathsf{OP}_\mathsf{K}}(\sigma) = hold_{\mathsf{OP}_\mathsf{K}}(m_1) = 1$. From the definition of R-Enforcer synthesis, it follows that $\alpha = \text{store\_1}$. Similarly, we can find that $o = \epsilon_{\Sigma_o}$ and $m = (\sigma, 1)$. Furthermore, as $safe_{\mathsf{OP}_\mathsf{K}}(\sigma, \epsilon_{\Sigma_o})$, we have (12).

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 $hold_{\mathsf{OP}_\mathsf{K}}$ applied to the states of the underlying estimator (state for weak opacity and trajectory for strong opacity), and the correctness of R-Verifier. Then, one can show easily, 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. $safe_{\mathsf{OP}_\mathsf{K}}(\mu', \mu)$).

Example 5.2 (R-Enforcers of K-weak opacity) *In Figure 9 are represented R-enforcers of K-weak opacity for the systems depicted in Figure 2. They are built from their respective K-delay state estimators.*



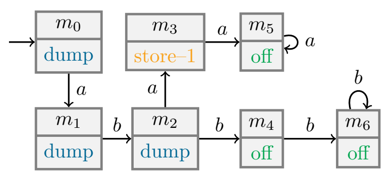
m0 dump, m1 dump, m2 dump, m3 store–1, m4 off, m5 off, m6 off
m0 off, m1 off, m2 off, m3 off, m4 off, m5 off
m0 off, m1 off, m2 off, m3 off, m4 off, m5 off, m6 off
m0 dump, m1 dump, m2 dump, m3 store–2, m4 store–1, m5 store–1, m6 off, m7 off, m8 off, m9 off, m10 off, m11 noleak
m0 dump, m1 dump, m2 dump, m3 dump, m4 dump, m5 dump, m6 store–1, m7 dump, m8 dump, m9 dump, m10 dump, m11 dump, m12 dump, m13 store–3, m14 store–3, m15 store–2, m16 store–1
a
b
c
a, b

(a) For the 2-weak opacity of $\mathcal{G}_2$'s secret (b) For the 2-weak opacity of $\mathcal{G}_3$'s secret (c) For the 3-weak opacity of $\mathcal{G}_4$'s secret

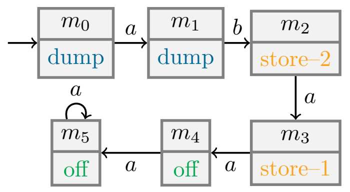
(d) For the 1-weak opacity of $\mathcal{G}_1$'s secret (e) R-enforcer of 2-weak opacity of $\mathcal{G}_5$'s secret

Figure 9: R-enforcers of K-weak opacity

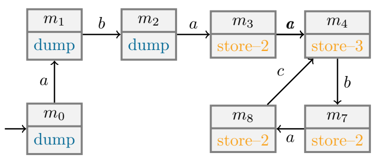
Example 5.3 (R-Enforcers of K-strong opacity) *In Figure 10 are represented R-enforcers of K-strong opacity for the systems depicted in Figure 2. They are built from their respective K-delay trajectory estimators.*



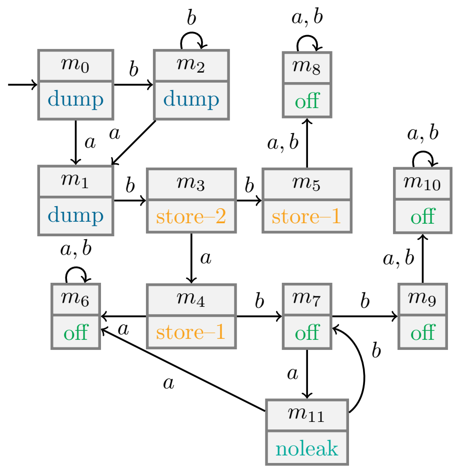
(a) For the 2-strong opacity of $\mathcal{G}_2$'s secret



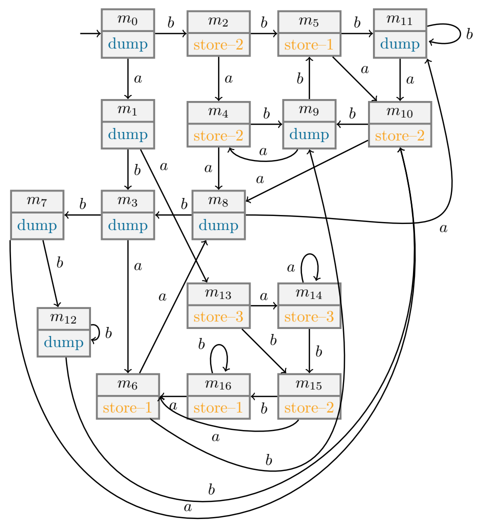
(b) For the 2-strong opacity of $\mathcal{G}_3$'s secret



(c) For the 3-strong opacity of $\mathcal{G}_4$'s secret



(d) For the 1-strong opacity of $\mathcal{G}_1$'s secret



(e) R-enforcer of 2-strong opacity of $\mathcal{G}_5$'s secret

Figure 10: R-enforcers of K-strong opacity

Remark 5.2 (About off states) *One may remark that, in runtime 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 5.3 (Comparison with supervisory control) *Note that, in this particular setting used to ensure opacity at runtime, thanks to the* halt *operation, runtime enforcement encompasses supervisory control. Indeed, blocking the underlying system or letting its execution going through, are the only primitives endowed to controllers.*

6 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 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.

All results of this paper are implemented in a prototype toolbox which is freely available [FM10]. A brief description of the tool is proposed in Appendix A. A complete description is available in [FM10].

Future works. 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 the existence of runtime enforcers is ensured.

Acknowledgement. The authors would like to thank Thierry Jéron for his remarks on a preliminary version of this paper.

A TAKOS: a Java Toolbox for Analyzing K-Opacity of Systems

TAKOS is the implementation of the results proposed in this report. Besides additional features, with TAKOS the user is able to:

(i) to check offline (*i.e.*, model-check) the opacity of a secret on a system,

(ii) to synthesize a runtime verification monitor in order to check the opacity at system runtime,

(iii) and to synthesize an enforcement monitor in order to ensure the opacity of a secret at system runtime.

In this section we briefly overview TAKOS its architecture, its functioning principle, and how to use it in order to validate (using the aforementioned techniques) the various levels of opacity presented in this report. A deeper presentation of TAKOS is available on its website [FM10].

Following the methodology presented in the introduction, the user can now put the validation into practice using TAKOS as depicted in Figure 11.

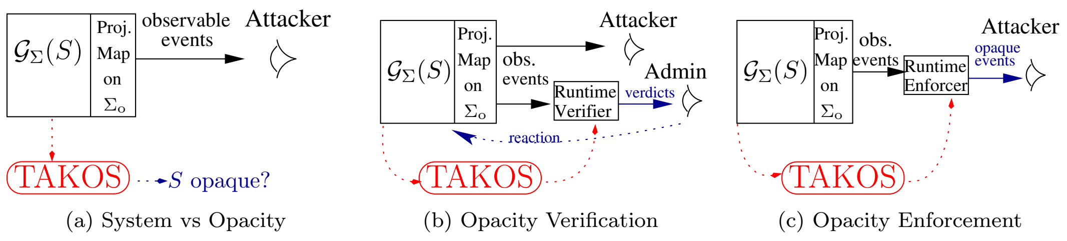
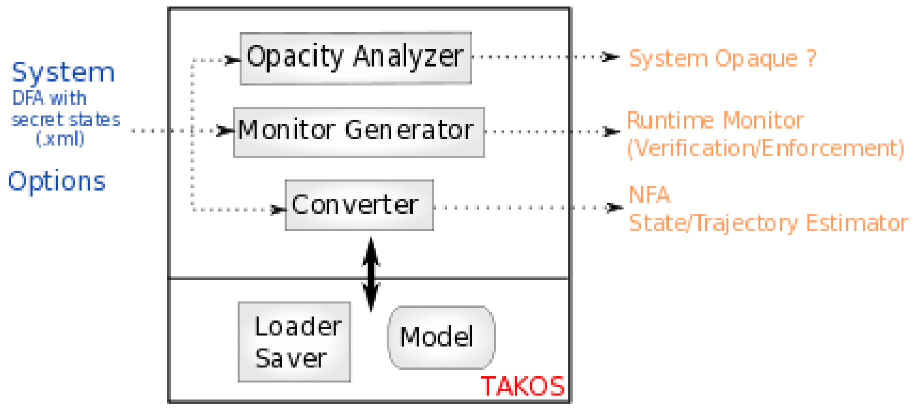


Figure 11: Three ways to validate opacity on a system with TAKOS

The architecture of TAKOS is depicted in Figure 12. As one can see in Figures 12 and 11, the user submits a system description as input to the tool, some options indicating the wished analysis. Then TAKOS produces the appropriate output (*e.g.*, a statement indicating whether or not the secret is opaque, an R-Verifier, etc...).



System
DFA with secret states (.xml)
Options
Opacity Analyzer
Monitor Generator
Converter
Loader Saver
Model
TAKOS
System Opaque ?
Runtime Monitor (Verification/Enforcement)
NFA State/Trajectory Estimator

Figure 12: Architecture of TAKOS

Let us take an example to briefly sketch how it is possible to validate opacity with TAKOS. Let us consider the system $\mathcal{G}_2$ introduced in Example 3.2 (p. 7) and depicted in Figure 2b (p. 6).

To generate the 2-delay trajectory estimator of $\mathcal{G}_2$, one can use the following command[15]:

```
java -jar ToolboxOpacity.jar -in path/to/plant2.xml -testimator 3
```

[15]In the tool, the size of an estimator is the number of state steps recorded.

The option `-in` and its parameter is used to indicate the path to the XML file describing $\mathcal{G}_2$. The option `-testimator` and its parameter is used to indicate that we want a trajectory estimator of size 3.

A possible description of $\mathcal{G}_2$ in the XML format supported by the tool is given in Listing 1.

Listing 1: Encoding of the LTS $\mathcal{G}_2$

```
<DFAutomaton>
  <Alphabet>
    <Event id="b" observable="true"/>
    <Event id="delta" observable="false"/>
    <Event id="a" observable="true"/>
  </Alphabet>
  <States>
    <State id="0" initial="true">
      <Transition event="a" nextState="4"/>
      <Transition event="delta" nextState="1"/>
    </State>
    <State id="1">
      <Transition event="a" nextState="2"/>
    </State>
    <State id="2" secret="true">
      <Transition event="b" nextState="3"/>
    </State>
    <State id="3">
      <Transition event="a" nextState="3"/>
    </State>
        <State id="4">
      <Transition event="b" nextState="5"/>
    </State>
    <State id="5">
      <Transition event="b" nextState="5"/>
    </State>
  </States>
  <AcceptingCondition>
    <AccState>1</AccState>
  </AcceptingCondition>
  <Description>DFA G2</Description>
</DFAutomaton>
```

Defining a plant in the XML format consists mainly in defining the alphabet, the list of states along with their transitions.

Launching the aforementioned command, one can obtain the trajectory estimator represented in Figure 13. One can remark that is indeed the trajectory estimator as expected (see Figure 5b).

```
Description: Trajectory Estimator of the DFA (size 3):DFA G2
Alphabet: [b, a]
State list:
m4:[2-*-3---3]
m5:[3---3---3]
m3:[4---5---5]
m6:[5---5---5]
m0:[1---1---1, 0---0---0]
m1:[1---1-*-2, 0---0---4, 0---0-*-2]
m2:[0-*-2-*-3, 0---4---5, 1-*-2-*-3]

Transition list:
m4:
   -a->m5
m5:
   -a->m5
m3:
   -b->m6
m6:
   -b->m6
m0:
   -a->m1
m1:
   -b->m2
m2:
   -b->m3
   -a->m4
```

Figure 13: 2-delay trajectory estimator for $\mathcal{G}_2$ produced by TAKOS

References

[BBB$^+$07] Eric Badouel, Marek Bednarczyk, Andrzej Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, 2007.

[BFHM06] Saddek Bensalem, Jean-Claude Fernandez, Klaus Havelund, and Laurent Mounier. Confirmation of deadlock potentials detected by runtime analysis. In Shmuel Ur and Eitan Farchi, editors, *PADTAD*, pages 41–50. ACM, 2006.

[BH08] Eric Bodden and Klaus Havelund. Racer: effective race detection using AspectJ. In Barbara G. Ryder and Andreas Zeller, editors, *ISSTA*, pages 155–166. ACM, 2008.

[BKMR08] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *Int. J. Inf. Secur.*, 7(6):421–435, 2008.

[CDM09] Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Dynamic observers for the synthesis of opaque systems. In Zhiming Liu and Anders P. Ravn, editors, *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2009.

[CL06] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[CMP92a] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *Automata, Languages and Programming*, pages 474–486, 1992.

[CMP92b] Edward Y. Chang, Zohar Manna, and Amir Pnueli. The safety-progress classification. Technical report, Stanford University, Dept. of Computer Science, 1992.

[DDM10] Jérémy Dubreil, Phillipe Darondeau, and Hervé Marchand. Supervisory control for opacity. *IEEE Transactions on Automatic Control*, 55(5):1089–1100, 2010.

[DJM09] Jérémy Dubreil, Thierry Jéron, and Hervé Marchand. Monitoring confidentiality by diagnosis techniques. In *European Control Conference*, pages 2584–2590, Budapest, Hungary, August 2009.

[Dub09] Jérémy Dubreil. *Monitoring and Supervisory Control for Opacity Properties.* PhD thesis, Université de Rennes 1, November 2009.

[FFM09a] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Enforcement monitoring wrt. the safety-progress classification of properties. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 593–600, New York, NY, USA, 2009. ACM.

[FFM09b] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Runtime Verification of Safety-Progress Properties. In Saddek Bensalem and Doron Peled, editors, *RV*, volume 5779 of *Lecture Notes in Computer Science*, pages 40–59. Springer, 2009.

[FM10] Yliès Falcone and Hervé Marchand. TAKOS: a Java Toolbox for Analyzing K-Opacity of Systems, July 2010. Available at http://toolboxopacity.gforge.inria.fr.

[HG08] Klaus Havelund and Allen Goldberg. Verify your runs. In *Verified Software: Theories, Tools, Experiments: First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions*, pages 374–383, Berlin, Heidelberg, 2008. Springer-Verlag.

[HMS06] Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.*, 28(1):175–205, 2006.

[LBW05] Jay Ligatti, Lujo Bauer, and David Walker. Enforcing Non-safety Security Policies with Program Monitors. In *ESORICS*, pages 355–373, 2005.

[LBW09] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security*, 12(3):1–41, January 2009.

[MP90] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *PODC '90: Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 377–410, New York, NY, USA, 1990. ACM.

[PZ06] Amir Pnueli and Aleksandr Zaks. PSL Model Checking and Run-Time Verification Via Testers. In *Formal Methods*, pages 573–586, 2006.

[RCB08] Grigore Roşu, Feng Chen, and Thomas Ball. Synthesizing monitors for safety properties – this time with calls and returns –. In *Workshop on Runtime Verification (RV'08)*, volume 5289 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2008.

[Run10] Runtime Verification. http://www.runtime-verification.org, 2001-2010.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[SH07] Anooshiravan Saboori and Christoforos N. Hadjicostis. Notions of security and opacity in discrete event systems. In *Proc. 46th IEEE Conf. Decision and Control, New Orleans, LA*, pages 5056–5061, 2007.

[SH08] A. Saboori and C. N. Hadjicostis. Delayed state estimation in discrete event systems and applications to security problems. Technical report, UIUC Coordinated Science Laboratory, February 2008.

[SH09] Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of k-step opacity and analysis of its complexity. In *Proc. 48th IEEE Conf. Decision and Control and 28th Chinese Control Conference, Shangai, P.R. China*, pages 5056–5061, 2009.

Contents

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399