HAL
open science

Dynamic Observers for the Synthesis of Opaque Systems

Franck Cassez, Jérémy Dubreil, Hervé Marchand

To cite this version:

Franck Cassez, Jérémy Dubreil, Hervé Marchand. Dynamic Observers for the Synthesis of Opaque Systems. [Research Report] PI 1930, 2009, pp.22. inria-00396442

HAL Id: inria-00396442

https://inria.hal.science/inria-00396442v1

Submitted on 18 Jun 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1930 – June 2009

Dynamic Observers for the Synthesis of Opaque Systems

Franck Cassez* , Jérémy Dubreil** , Hervé Marchand***

Abstract: In this paper, we address the problem of synthesizing *opaque* systems. A secret predicate S over the runs of a system G is *opaque* to an external user having partial observability over G, if s/he can never infer from the observation of a run of G that the run belongs to S. We first investigate the case of *static* partial observability where the set of events the user can observe is fixed a priori. In this context, we show that checking whether a system is opaque is PSPACE-complete, which implies that computing an optimal static observer ensuring opacity is also a PSPACE-complete problem. Next, we introduce *dynamic* partial observability where the set of events the user can observe changes over time. We show how to check that a system is opaque w.r.t. to a dynamic observer and also address the corresponding synthesis problem: given a system G and secret states S, compute the set of dynamic observers under which S is opaque. Our main result is that the set of such observers can be finitely represented and can be computed in EXPTIME.

Key-words: security, confidentiality property, opacity, synthesis, dynamic observation

---

Observateurs dynamiques pour la synthèse de systèmes opaques

Résumé : *Dans cette article, nous nous intéressons à la synthèse de systèmes opaques. Un prédicat S sur les exécutions d'un système G est opaque vis à vis d'un utilisateur, observant partiellement le système, si celui-ci ne peut jamais déduire des traces observées que l'exécution courante appartient à S. Dans un premier temps, nous nous intéressons au cas où le sous-ensemble des événements que l'attaquant peut observer est fixe. Dans ce cadre, nous montrons que la vérification de l'opacité est PSPACE complète. Dans un deuxième temps, nous introduisons la notion d'observation partielle dynamique qui traduit le fait que l'ensemble des événements que l'attaquant peut observer varie avec le temps. Nous montrons comment vérifier l'opacité sur un système vis à vis d'un observateur dynamique et considérons également le problème de synthèse associé: étant donné un système G et un secret S, calculer l'ensemble des observateurs dynamiques pour lesquels S est opaque. Nous montrons ainsi que cet ensemble peut être représenté de manière fini avec une complexité de calcul EXPTIME.*

Mots clés : *sécurité, propriété de confidentialité, opacité, synthèse, observation dynamique*

This work was partially supported by the Politess RNRT project.

* Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme. National ICT Australia & CNRS, Sydney, Australia
** VerTeCs, INRIA, centre Rennes - Bretagne Atlantique
*** VerTeCs, INRIA, centre Rennes - Bretagne Atlantique

©IRISA – Campus de Beaulieu – 35042 Rennes Cedex – France – +33 2 99 84 71 00 – www.irisa.fr

1 Introduction

Security is one of the most important and challenging aspects in designing services deployed on large open networks like Internet or mobile phones. Some services like medical information storage, e-banking or e-voting systems may deal with information that should not be corrupted or acquired by unauthorized people. For such services, naturally subject to malicious attacks, methods to certify their security are crucial. In this context there has been many research to develop formal methods for the design of secure systems and a growing interest in the formal verification of security properties [16, 3, 12] and in their model-based testing [20, 15, 6, 14, 10]. Security properties are generally divided into three categories: *integrity*, *availability* and *confidentiality*. We focus here on confidentiality and especially information flow properties. We use the notion of *opacity* introduced in [18] and later developed in [4] formalizing the absence of information flow, or more precisely, the impossibility for an attacker to infer the truth of a predicate (it could be the occurrence in the system of some particular sequences, or the fact that the system is in some particular configurations). Consider a predicate φ over the runs of a system G and an attacker observing only a subset of the events of G. We assume also that the attacker knows the model G. The attacker should not be able to infer that a run of G belongs to φ. The secret φ is opaque for G with respect to a given partial observation if for every run of G that belongs to φ, there exists a run, observationally equivalent from the attacker's point of view, that does not belong to φ. In such case, the attacker can never be sure that a run of G satisfying φ has occurred. In the sequel, we shall consider a secret φ corresponding to either a regular language or a set of secret states. Finally, note that the definition of opacity is general enough to model other notions of information flow like trace-based non-interference [11] and anonymity. The reductions are given in [4]. Note also that *secrecy* [1] can be handled as a particular case of opacity (see Section 3) and thus our framework applies to secrecy as well.

Related Work. Methods for the synthesis of opaque systems have already been investigated from the supervisory control point of view. In these frameworks, some of the events are uncontrollable and the set of events an external attacker can observe is fixed. If the system is G, the approach is then to *restrict* G (remove some of its behaviors) using a supervisor (or controller) C, in order to render a secret φ opaque in the supervised system G/C. In [2], the authors consider several secrets and attackers with different sets of observable events. They provide sufficient conditions to compute an optimal controller preserving all secrets assuming that the controller has complete knowledge of the system and full control on it. In [8, 9], the authors consider a control problem under partial observation and provide algorithms to compute the optimal controller ensuring the opacity of one secret against one attacker. Other works on the enforcement of opacity by means of controllers can be found in [22]. Note that these approaches are intrusive in the sense that the system G has to be modified.

Our Contribution. In this paper, instead of restricting the behavior of the system by means of a controller which enables/disables some actions, we consider *dynamic* observers that will dynamically change the set of observable events in order to ensure opacity. Compared to the approaches related to the supervisory control theory, this approach is not intrusive in the sense that it does not restrict system's behavior but only hide observable events. Indeed, one can think of a dynamic observer as a monitor or *filter* (See Figure 1) which is added on top of the system.

Figure 1: Architecture Filtering out Sequences of Events in G

The main contributions of this paper are two-fold. First, we extend the notion of opacity for static observers (i.e., the natural projection) to dynamic observers.[1] We show how to check opacity when the dynamic observer is given by a finite automaton. Second we give an algorithm to compute the set of all dynamic observers which can ensure opacity of a secret φ for a given system G. In particular, we show

[1] At this point, it should be mentioned that we assume the attacker has not only a perfect knowledge of the system but also of the observer.

that this problem can be reduced to a *safety 2-player game problem*. Intuitively, the first player will play the role of an observer and decide which subset of event should remain observable after a given trace and Player 2 will play the role of both the system and the attacker and will decide what will be the next observable event amongst the ones Player 1 has last chosen to render observable. The goal of Player 2 is thus to make the system evolve in states in which the attacker is sure that the secret is revealed, whereas the goal of Player 1 is opposite (he has to choose the successive sequences of observable events' sets so that the secret is never revealed). In particular, we show that the set of valid observers (the ones which ensure opacity) can be finitely represented. Finally we consider an optimization problem which is to compute a least expensive observer which ensures opacity. Notice also that *secrecy* (see section 3) can be handled as a particular case of opacity and thus our framework applies to secrecy as well. Finally, the notion of *dynamic observers* was already introduced in [5] for the fault diagnosis problem. Notice that the fault diagnosis problem and the opacity problems are not reducible one to the other and thus we have to design new algorithms to solve the opacity problems under dynamic observations.

Organization of the Paper. In Section 2 we introduce some notations for words, languages and finite automata. In Section 3 we define the notion of opacity and show that deciding opacity for finite automata is PSPACE-complete. We also consider an optimization problem of computing a largest set of observable events to ensure opacity and show that this problem is PSPACE-complete as well. Section 4 considers dynamic observers for ensuring opacity. We prove that the set of all observers that ensure opacity can be computed in EXPTIME. In Section 5 we define a notion of cost for observers and give an algorithm to compute a least expensive dynamic observer which ensures opacity.

2 Notation & preliminaries

Let Σ be a finite alphabet. Σ^* is the set of finite words over Σ and contains the *empty* word ε. A *language* L is any subset of Σ^*. Given two words $u \in \Sigma^*$ and $u' \in \Sigma^*$ we denote $u.u'$ the concatenation of u and u' which is defined in the usual way. $|u|$ stands for the length of the word u (the length of the empty word is zero). We let Σ^n with $n \in \mathbb{N}$ denote the words of length n over Σ.

Given $\Sigma_1 \subseteq \Sigma$, we define the *projection* operator on finite words, $P_{\Sigma_1} : \Sigma^* \to \Sigma_1^*$, that removes in a sequence of Σ^* all the events that do not belong to Σ_1. Formally, P_{Σ_1} is recursively defined as follows: $P_{\Sigma_1}(\varepsilon) = \varepsilon$ and for $\lambda \in \Sigma, u \in \Sigma^*$, $P_{\Sigma_1}(u.\lambda) = P_{\Sigma_1}(u).\lambda$ if $\lambda \in \Sigma_1$ and $P_{\Sigma_1}(u)$ otherwise. Let $K \subseteq \Sigma^*$ be a language. The definition of projection for words extends to languages: $P_{\Sigma_1}(K) = \{P_{\Sigma_1}(u) \mid u \in K\}$. Conversely, let $K \subseteq \Sigma_1^*$. The *inverse projection* of K is $P_{\Sigma_1}^{-1}(K) = \{u \in \Sigma^* \mid P_{\Sigma_1}(u) \in K\}$.

Definition 1 (Automaton) *An* automaton *G is a tuple $(Q, q_0, \Sigma, \delta, F)$ with Q a set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition relation and $F \subseteq Q$ is the set of* accepting *states. If Q is finite, G is a* finite automaton *(FA).* ■

For $q \in Q$, *Enabled*(q) is the set of actions *enabled* at q, i.e., the set of λ such that $\delta(q, \lambda) \neq \emptyset$. We write $q \xrightarrow{\lambda}$ whenever $\lambda \in$ *Enabled*(q). An automaton is *complete* if for each $\lambda \in \Sigma$ and each $q \in Q$, $q \xrightarrow{\lambda}$. The automaton A is deterministic if for all $q \in Q$ and all $\lambda \in \Sigma$, $|\delta(q, \lambda)| \leq 1$. A *run* ρ from state q_0 in G is a finite sequence of transitions

$$q_0 \xrightarrow{\lambda_1} q_1 \xrightarrow{\lambda_2} q_2 \ \cdots \ q_{i-1} \xrightarrow{\lambda_i} q_i \ \cdots \ q_{n-1} \xrightarrow{\lambda_n} q_n \tag{1}$$

s.t. $\lambda_{i+1} \in \Sigma$ and $q_{i+1} \in \delta(q_i, \lambda_{i+1})$ for $i \geq 0$. The *trace* of the run ρ is $tr(\rho) = \lambda_1.\lambda_2 \cdots \lambda_n$. We let $last(\rho) = q_n$, and the length of ρ, denoted $|\rho|$, is n. For $i \leq n$ we denote by $\rho(i)$ the prefix of the run ρ truncated at state q_i, i.e., $\rho(i) = q_0 \xrightarrow{\lambda_1} q_1 \cdots q_{i-1} \xrightarrow{\lambda_i} q_i$. The set of finite runs from q_0 in G is denoted *Runs*(G).
A word $u \in \Sigma^*$ is *generated* by G if $u = tr(\rho)$ for some $\rho \in$ *Runs*(G). Let $L(G)$ be the set of words generated by G. The word $u \in \Sigma^*$ is *accepted* by G if $u = tr(\rho)$ for some $\rho \in$ *Runs*(G) with $last(\rho) \in F$. The *language of (finite) words* $L_F(G)$ of G is the set of words accepted by G. If G is a FA such that $F = Q$ we simply omit F in the tuple that defines G.

In the sequel we will use the *Post* operator defined as follows: let $X \subseteq Q$, $Post(X, \varepsilon) = X$ and for $u \in \Sigma^*, \lambda \in \Sigma, Post(X, u.\lambda) = \cup_{q \in Post(X,u)} \delta(q, \lambda)$. We also let $Post(X, L) = \cup_{u \in L} Post(X, u)$ for a non empty language L.

Product of Automata. The product of automata is defined in the usual way: the product automaton represents the concurrent behavior of the automata with synchronization on the common events.

Definition 2 (Product of Automata) *Let* $A_1 = (Q_1, q_0^1, \Sigma_1, \delta_1, F_1)$ *and* $A_2 = (Q_2, q_0^2, \Sigma_2, \delta_2, F_2)$. *The* product *of* A_1 *and* A_2 *is the automaton* $A_1 \times A_2 = (Q, q_0, \Sigma, \delta, F)$ *where:*

- $Q = Q_1 \times Q_2$, $F = F_1 \times F_2$
- $q_0 = (q_0^1, q_0^2)$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $\delta : Q \times \Sigma \to 2^Q$ *is defined by:*

$$(q_1, q_2), \lambda \quad \mapsto \quad \begin{cases} \delta_1(q_1, \lambda) \times \delta_2(q_2, \lambda) \text{ if } \lambda \in \Sigma_1 \cap \Sigma_2 \\ \delta_1(q_1, \lambda) \times \{q_2\} \text{ if } \lambda \in \Sigma_1 \setminus \Sigma_2 \\ \{q_1\} \times \delta_2(q_2, \lambda) \text{ if } \lambda \in \Sigma_2 \setminus \Sigma_1 \end{cases}$$

■

3 Opacity with Static Projections

Enforcing opacity aims at preventing an attacker, denoted $\mathcal{U}$, from deducing confidential information on the execution of a system G from the observation of a subset of events $\Sigma_o \subseteq \Sigma$. Given a run of G with trace s, the observation of the attacker $\mathcal{U}$ is given by the static natural projection $P_{\Sigma_o}(s)$ following the architecture of Figure 1 with $D(u) = P_{\Sigma_o}(u)$.

In the sequel we let $G = (Q, q_0, \Sigma, \delta)$ be a non-deterministic finite automaton over Σ. The language $L_\varphi \subseteq \Sigma^*$ represents a confidential information on the execution of G, i.e. if the current trace of a run is $u \in L(G)$, the user should not be able to deduce, from the knowledge of $P_{\Sigma_o}(s)$ and the structure of G, that $u \in L_\varphi$. As stressed earlier, the attacker is armed with full information on the structure of G (he can perform computations using G like subset constructions) but has only partial observability at runtime upon its behaviors, namely the observed traces are in Σ_o^*.

Let $\Sigma_o \subseteq \Sigma$. The set of Σ_o-*traces* of G is $Tr_{\Sigma_o}(G) = P_{\Sigma_o}(L(G))$. We define the operator $[\![\cdot]\!]_{\Sigma_o}$ by:

$$[\![\varepsilon]\!]_{\Sigma_o} = \{\varepsilon\} \quad \text{and for } \mu \in \Sigma_o^* \text{ and } \lambda \in \Sigma_o,\ [\![\mu.\lambda]\!]_{\Sigma_o} = P_\Sigma^{-1}(\mu).\lambda \cap L(G)$$

i.e., $u \in [\![\mu.\lambda]\!]_{\Sigma_o}$ iff (1) the projection of u is $\mu.\lambda$ and (2) $u \in L(G)$ and (3) u ends with an observable "λ" event.

Remark 1 *The above semantics is consistent with an on-line observation performed by a user of the system for whom the system is only seen through the interface given by the observation mask* P_{Σ_o}. *We suppose that users are reacting faster than the system. Therefore, when an observable event occurs, a user can take a decision before the system proceeds with any unobservable event. This explains why we do not consider trajectories ending with unobservable events in the definition of the semantics.* □

Next we introduce the notion of opacity first defined in [18, 4]. Intuitively, a secret L_φ is said to be *opaque* with respect to a pair (G, Σ_o) if the attacker $\mathcal{U}$ can never be sure that the current trace of the run in G is in L_φ.

Definition 3 (Trace Based Opacity) *Let* $L_\varphi \subseteq \Sigma^*$. *The secret* L_φ *is* opaque *with respect to* (G, Σ_o) *if for all* $\mu \in Tr_{\Sigma_o}(G)$, $[\![\mu]\!]_{\Sigma_o} \not\subseteq L_\varphi$. ■

Remark 2 *Notice that* $[\![\mu]\!]_{\Sigma_o}$ *is never empty when* $\mu \in Tr_{\Sigma_o}(G)$.

Example 1 *Let* G *be the automaton depicted in Figure 2 with* $\Sigma = \{h, p, a, b\}$, $\Sigma_o = \{a, b\}$. *The secret under consideration is the occurrence of the event "h", and this can defined by* $L_\varphi = \Sigma^*.h.\Sigma^*$. *This should not be revealed to the users of the system, knowing that "h" is not observable.* L_φ *is not opaque w.r.t.* (G, Σ_o) *as by observing "b", the sole corresponding sequence in* $[\![b]\!]_{\Sigma_o}$ *is* $h.b$ *and thus it is in* L_φ. *Note that if the attacker observes only "a", then it cannot tell whether the current sequence of actions of the system belongs to* L_φ *as* $[\![a]\!]_{\Sigma_o} = \{p.a, h.a\}$ *and thus is not included in* L_φ. □

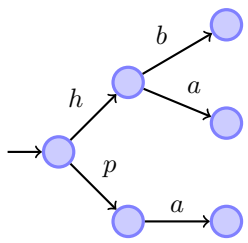


Figure 2: Trace based opacity illustration

An alternative definition of opacity is when the secret is a set of states of the system.

Definition 4 (State Based Opacity) *Let $F \subseteq Q$. The secret F is* opaque *with respect to (G, Σ_o) if for all $\mu \in Tr_{\Sigma_o}(G)$, $Post(\{q_0\}, [\![\mu]\!]_{\Sigma_o}) \not\subseteq F$.* ■

We can extend the definition of opacity to a (finite) family of sets $\mathcal{F} = \{F_1, F_2, \cdots, F_k\}$: the secret $\mathcal{F}$ is *opaque* with respect to (G, Σ_o) if for all $F \in \mathcal{F}$, for all $\mu \in Tr_{\Sigma_o}(G)$, $Post(\{q_0\}, [\![\mu]\!]_{\Sigma_o}) \not\subseteq F$. This can be used to express other kinds of confidentiality properties. For example, [1] introduced the notion of *secrecy* of a set of states F. Intuitively, F is not *secret* w.r.t. G and Σ_o whenever after an observation μ, the attacker either knows that the system is in F or knows that it is not in F. Secrecy can thus be handled considering the opacity w.r.t. a family $\{F, Q \setminus F\}$. In the sequel we consider only one set of states F and, when necessary, we point out what has to be done for solving the problems with family of sets.

Example 2 *Consider the automaton G of Figure 3, with $\Sigma_o = \Sigma = \{a, b\}$. The secret is given by the states represented by red squares) i.e., $F = \{q_2, q_5\}$.*

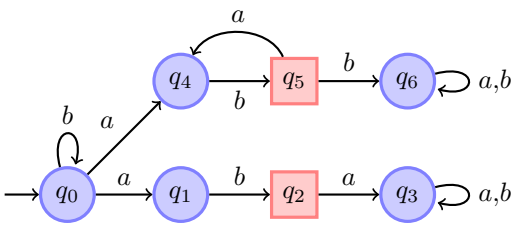


Figure 3: State based opacity illustration

The secret F is certainly not state-based opaque with respect to (G, Σ), as by observing a trace of $b^*.a.b$, an attacker $\mathcal{U}$ knows that the system is in a secret state. Notice that he does not know whether it is q_2 or q_5 but still he knows that the state of the system is in F. □

3.1 From Trace Based to State Based Opacity

Let L_φ be given by a finite and complete automaton A_φ with accepting states Q_φ and initial state q_0^φ. Define $G \times A_\varphi$ with accepting states $F_\varphi = Q \times Q_\varphi$.

Proposition 1 *If A_φ is deterministic then L_φ is trace based opaque w.r.t. (G, Σ_o) iff F_φ is State Based opaque w.r.t. $(G \times A_\varphi, \Sigma_o)$.*

Proof. The proof is a direct consequence of Definitions 3, 4 and 2. Let (q_0, q_0^φ) be the initial state of $G \times A_\varphi$. We have $s \in L(G) \cap L_\varphi \Leftrightarrow s \in L_{F_\varphi}(G \times A_\varphi)$ and Since A_φ is deterministic, this is equivalent to $Post(\{(q_0, q_0^\varphi)\}, [\![\mu]\!]_{\Sigma_o}) \subseteq F_\varphi$. Consequently for all $\mu \in Tr_{\Sigma_o}(G)$, $[\![\mu]\!]_{\Sigma_o} \subseteq L_\varphi \iff Post(\{(q_0, q_0^\varphi)\}, [\![\mu]\!]_{\Sigma_o}) \subseteq F_\varphi$. □

Hence in this case, Trace Based Opacity can be reduced in polynomial time to State Based Opacity. If A_φ is non-deterministic, prior to the previous construction we need to determinize A_φ and the product has size exponential in $|A_\varphi|$.
In the sequel we shall focus on variations of the State Based Opacity problem:

Problem 1 (Static State Based Opacity Problem)
INPUT: *A non-deterministic FA* $G = (Q, q_0, \Sigma, \delta, F)$ *and* $\Sigma_o \subseteq \Sigma$.
PROBLEM: *Is* F *opaque w.r.t.* (G, Σ_o) *?*

3.2 Checking State Based Opacity

Let $G = (Q, q_0, \Sigma, \delta, F)$ be a non-deterministic finite automaton with F the secret states. In order to check for the opacity of F w.r.t. (G, Σ_o), we first introduce the classical notion of determinization via subset construction adapted to our definition of opacity: $Det_o(G)$ denotes the deterministic automaton which is computed from G. Formally, it can be obtained as follows:

Definition 5 *Let* $G = (Q, q_0, \Sigma, \delta, F)$ *then* $Det_o(G) = (\mathcal{X}, X_0, \Sigma_o, \Delta, F_o)$ *where:*

- $\mathcal{X} \subseteq 2^Q \setminus \emptyset$ *and* $X_0 = \{q_0\}$ *and* $F_o = 2^F$,
- *given* $\lambda \in \Sigma_o$, *if* $X' = Post(X, (\Sigma \setminus \Sigma_o)^*.\lambda) \neq \emptyset$ *then* $\Delta(X, \lambda) = X'$. ■

Now, based on this operation, to check whether F is opaque w.r.t. (G, Σ_o) we can proceed as follows:

1. determinize G using the construction of Definition 5 and denote $Det_o(G)$ the obtained automaton with states in 2^Q. Note that the set of accepting states is $F_o = 2^F$;
2. check whether F_o is reachable in $Det_o(G)$
   - if yes, then F is not opaque w.r.t. (G, Σ_o),
   - otherwise, F is opaque w.r.t. (G, Σ_o).

Indeed, if there exists $\mu \in \Sigma_o^*$ such that $\mu \in L_{F_o}(Det_o(G))$, then according to Definition 5, it entails that $Post(\{q_0\}, [\![\mu]\!]_{\Sigma_o}) \subseteq F$, meaning that F is not opaque w.r.t. (G, Σ_o). Thus, according to this construction, the set of observed traces for which the attacker knows that the current execution discloses the secret is given by $L_{F_o}(Det_o(G))$ where $F_o = 2^F$.

To check opacity for a family $\{F_1, F_2, \cdots, F_k\}$, we define $\mathcal{F}$ to be the set $2^{F_1} \cup 2^{F_2} \cup \cdots \cup 2^{F_k}$ (as pointed out before, this enables us to handle secrecy).

The previous construction shows that State Based Opacity on non-deterministic FA can be checked in exponential time. Actually, checking state based opacity for (non-deterministic) FA is PSPACE-complete. First, from [21], checking language universality for a non-deterministic FA is PSPACE-complete. Given a FA G over Σ and F the set of accepting states, the (language) universality problem is to decide whether G accepts all possible sequences, namely if $L_F(G) = \Sigma^*$. If not, then G is not universal. From [21], checking language universality for a non-deterministic FA is PSPACE-complete. To show that the state based opacity problem is PSPACE-complete, we prove that state based opacity is equivalent to universality.

Theorem 1 *Problem 1 is PSPACE-complete for non-deterministic FA.*

Proof. Let $G = (Q, q_0, \Sigma, \delta, F)$ be a non-deterministic finite automaton with accepting states in F. We assume that G is complete i.e., $L(G) = \Sigma^*$. Note that in this case $[\![w]\!]_\Sigma = w$.

$$G \text{ is not language universal} \iff \exists w \in \Sigma^* s.t.\ Post(\{q_0\}, [\![w]\!]_\Sigma) \subseteq Q \setminus F$$

With the definition of state based opacity, taking $\Sigma_o = \Sigma$,

$$Q \setminus F \text{ is not opaque w.r.t. } (G, \Sigma) \iff \exists \mu \in \Sigma^* s.t.\ Post(\{q_0\}, [\![\mu]\!]_\Sigma) \subseteq Q \setminus F$$

Hence $L_F(G) = \Sigma^*$ iff $Q \setminus F$ is opaque w.r.t. (G, Σ) and opacity if PSPACE-complete. □

3.3 Maximum Cardinality Set for Static Projections

If a secret is opaque w.r.t. to a set of observable events Σ_o, it will still be opaque w.r.t. any subset of Σ_o (the less you observe, the less accurate you are). It might be of interest to hide as few events as possible from the attacker still preserving opacity of a secret. Indeed, hiding an event can be seen as energy consuming or as limiting the interactions or visibility for users of the system (and some of them are not malicious attackers) and thus should be avoided if unnecessary.

Assume the secret is opaque w.r.t. $\Sigma_o = \emptyset$. This suggests an optimization problem which can be formulated as follows: What is the maximum cardinality of the sets of observable events Σ_o such that the secret is opaque ? More precisely, we can state the following optimization problem:

Problem 2 (Maximum Number of Observable Events)

INPUT: *A non-deterministic FA $G = (Q, q_0, \Sigma, \delta, F)$, $n \in \mathbb{N}$ s.t. $n \leq |\Sigma|$.*

PROBLEMS:

(A) *Is there any $\Sigma_o \subseteq \Sigma$ with $|\Sigma_o| = n$, such that F is opaque w.r.t. (G, Σ_o) ?*

(B) *If the answer to (A) is "yes", find the maximum n_0 such that there exists $\Sigma_o \subseteq \Sigma$ with $|\Sigma_o| = n_0$ and F is opaque w.r.t. (G, Σ_o).*

Theorem 2 *Problem 2.(A) and Problem 2.(B) are PSPACE-complete.*

Proof. PSPACE-easiness follows directly as we can guess a set Σ_o with $|\Sigma_o| = n$ and check in PSPACE whether F is opaque w.r.t. (G, Σ_o). Thus Problem 2.(A) is in NPSPACE and thus in PSPACE.

PSPACE-hardness is also easy because taking $n = |\Sigma|$ amounts to checking that F is opaque w.r.t. (G, Σ) which has been shown equivalent to the universality problem (proof of Theorem 1).
To solve Problem 2.(B) it suffices to iterate a binary search and thus Problem 2.(B) is also in PSPACE. To see it is PSPACE-complete, to check whether F is opaque w.r.t. (G, Σ), it suffices to solve Problem 2.(B) and then check whether $n_0 = |\Sigma|$. □

4 Opacity with Dynamic Projection

So far, we have assumed that the observability of events is given a priori and this is why we used the term static projections/observers. We generalize this approach by considering the notion of *dynamic projections* encoded by means of *dynamic observers* as introduced in [5] for the fault diagnosis problem. In this section, we formulate the opacity problem using dynamic observers. Notice that the fault diagnosis problem and the opacity problems are not reducible one to the other and thus we have to design new ad hoc algorithms to solve the opacity problems under dynamic observations.

Next we introduce the notion of dynamic projection that permits to render unobservable some events after a given observed trace (for example, some outputs of the system). To illustrate the benefits of such projections, we consider the following example:

Example 3 *Consider again the automaton G of Example 2, Figure 3, where the set of secret states is $F = \{q_2, q_5\}$. With $\Sigma_o = \Sigma = \{a, b\}$, F is not opaque.*

If either $\Sigma_o = \{a\}$ or $\Sigma_o = \{b\}$, then the secret becomes opaque. Thus if we have to define static sets of observable events, at least one event will have to be permanently unobservable. However, the less you hide, the more important is the observable behavior of the system. Thus, we should try to reduce as much as possible the hiding of events. For this particular example, we can be more efficient by using a dynamic *observer that will render unobservable an event only when necessary. In this example, after observing b^*, the attacker still knows that the system is in the initial state. However, if a subsequent "a" follows, then the attacker should not be able to observe "b" as in this case it could know the system is in a secret state. We can then design a dynamic events's hider as follows: at the beginning, everything is observable; when an "a" occurs, the observer hides any subsequent "b" occurrences and permits only the observation of "a". Once an "a" has been observed, the observer releases its hiding by letting both "a" and "b" be observable again.* ◇

4.1 Opacity Generalized to Dynamic Projection

In this section, we define the notion of dynamic projection and its associated dynamic observer. We show how to extend the notion of opacity in order to take into account the dynamic aspect of events' observability.

4.1.1 Dynamic projections and dynamic observers

An (observation-based) dynamic projection is a function that will decide to let an event be observable or to hide it (see Figure 1), thus playing the role of a filter between the system and the attacker to prevent information flow. Such a projection can be defined as follows:

Definition 6 *A* dynamic observability choice *is a mapping* $T_D : \Sigma^* \to 2^{\Sigma}$. *The (observation-based)* dynamic projection *induced by* T_D *is the mapping* $D : \Sigma^* \to \Sigma^*$ *defined by:* $D(\varepsilon) = \varepsilon$ *and for all* $u \in \Sigma^*$, *and all* $\lambda \in \Sigma$,

$$D(u.\lambda) = D(u).\lambda \text{ if } \lambda \in T_D(D(u)) \text{ and } D(u.\lambda) = D(u) \text{ otherwise.} \tag{2}$$

Assuming that $u \in \Sigma^*$ occurred in the system and $\mu \in \Sigma^*$ has been observed so far by the attacker i.e., $\mu = D(u)$, then the events that are currently observable are the ones which belong to $T_D(\mu)$. Note that the choice of this set cannot change until an observable event occurs in the system. Given $\mu \in \Sigma^*$, $D^{-1}(\mu) = \{u \in \Sigma^* \mid D(u) = \mu\}$ i.e., the set of sequences that project onto μ.

Example 4 *A dynamic projection* $D : \Sigma^* \to \Sigma^*$ *corresponding to the one we introduced in Example 3 can induced by the dynamic observability choice* T_D *defined by* $\forall u \in b^*.a$, $T_D(u) = \{a\}$, *and* $T_D(u) = \{a, b\}$ *for all the other sequences* $u \in \Sigma^*$. ◇

For a model G as above and a dynamic projection D, we denote by $Tr_D(G) = D(L(G))$, the set of observed traces. Conversely, given $\mu \in Tr_D(G)$, the set of words $[\![\mu]\!]_D$ of G that are compatible with μ is defined by:

$$[\![\varepsilon]\!]_D = \{\varepsilon\} \quad \text{and for } \mu \in \Sigma^*,\ \lambda \in \Sigma : [\![\mu.\lambda]\!]_D = D^{-1}(\mu).\lambda \cap L(G)$$

Given two different dynamic projections D_1 and D_2 and a system G over Σ, we say that D_1 and D_2 are G-equivalent, denoted $D_1 \sim_G D_2$, whenever for all $u \in L(G)$, $D_1(u) = D_2(u)$. The relation $\sim_G$ identifies two dynamic projections when they agree on $L(G)$; they can disagree on other words in Σ^* but since they will not be generated by G, it will not make any difference from the attacker point of view. In the sequel we will be interested in computing the interesting part of dynamic projections given G, and thus will compute one dynamic projection in each class.

4.1.2 Opacity with dynamic projection.

As in the previous section, we assume that the user is considered as an attacker, $\mathcal{U}$, who is armed for this with full information on the structure of G and knows D. Based on these assumptions, we generalize Definition 4 by taking into account the new observation interface given by D.

Definition 7 *Given a FA* $G = (Q, q_0, \Sigma, \delta, F)$, F *is opaque with respect to* (G, D) *if*

$$\forall \mu \in Tr_D(G),\ Post(\{q_0\}, [\![\mu]\!]_D) \not\subseteq F \tag{3}$$

■

Again, this definition extends to family of sets. We say that D is a valid dynamic projection if (3) is satisfied (i.e., whenever F is opaque w.r.t. (G, D)) and we denote by $\mathcal{D}$ the set of valid dynamic projections. Obviously if $D_1 \sim_G D_2$, then D_1 is valid if and only if D_2 is valid. We denote by $\mathcal{D}_{\sim_G}$ the quotient set of $\mathcal{D}$ by $\sim_G$.

Remark 3 *Let* $\Sigma_o \subseteq \Sigma$ *be a fixed subset of actions, then if* D *is a dynamic projection that defines a constant mapping making actions in* Σ_o *always observable (and the others always unobservable), we have* $D(\mu) = P_{\Sigma_o}(\mu)$ *and we retrieve the original definition of state based opacity in case of static projection. Finally, note that we can also alternatively consider a trace-based opacity as the one defined in Definition 3, with dynamic projection instead of natural projection with a result similar to the one of Proposition 1. The property of secrecy can be extended as well using dynamic projection.* ◇

In the sequel, we will be interested in checking the opacity of F w.r.t. (G, D) or to synthesize such a dynamic projection D ensuring this property. In Section 3, the dynamic projection was merely the natural projection and computing the observational behavior of G was easy. Here, we need to find a characterization of these dynamic projections that can be used to check opacity or to enforce it. To do so, we introduce the notion of dynamic observer [5] that will encode a dynamic projection in terms of automata.

Definition 8 (Dynamic observer) *An* observer *is a complete deterministic labeled automaton* $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$ *where* X *is a (possibly infinite) set of states,* $x_0 \in X$ *is the initial state,* Σ *is the set of input events,* $\delta_o : X \times \Sigma \to X$ *is the transition function (a total function), and* $\Gamma : X \to 2^{\Sigma}$ *is a labeling function that specifies the set of events that the observer keeps observable at state* x*. We require that for all* $x \in X$ *and for all* $\lambda \in \Sigma$*, if* $\lambda \notin \Gamma(x)$*, then* $\delta_o(x, \lambda) = x$*, i.e., if the observer does not want an event to be observed, it does not change its state when such an event occurs.* ■

We extend δ_o to words of Σ^* by: $\delta_o(q, \varepsilon) = q$ and for $u \in \Sigma^*, \lambda \in \Sigma$, $\delta_o(q, u.\lambda) = \delta_o(\delta_o(q, u), \lambda)$. Assuming that the observer is at state x and an event λ occurs, it outputs λ whenever $\lambda \in \Gamma(x)$ or nothing (ε) if $\lambda \notin \Gamma(x)$ and moves to state $\delta_o(x, \lambda)$. An observer can be interpreted as a functional transducer taking a string $u \in \Sigma^*$ as input, and producing the output which corresponds to the successive events it has chosen to keep observable.

Example 5 *Examples of dynamic observers are given in Figure 4.* ◇

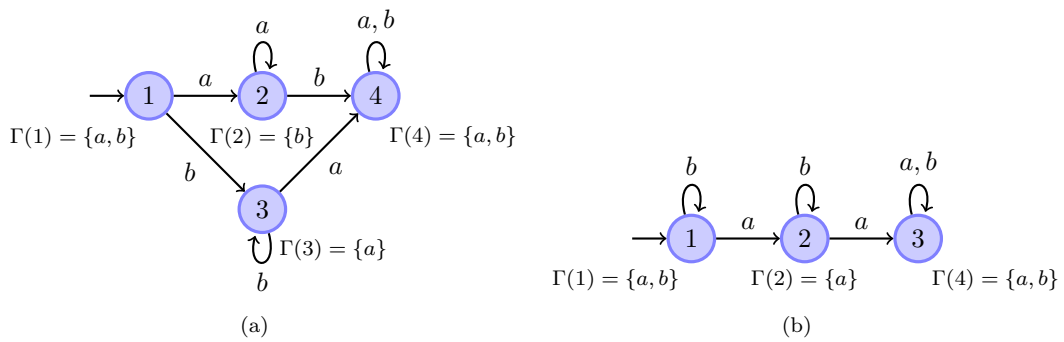


Figure 4: Examples of Dynamic Observers

We now relate the notion of dynamic observer to the notion of dynamic projection.

Proposition 2 *Let* $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$ *be an observer and define* $D_{\mathcal{O}}$ *as follows:* $D_{\mathcal{O}}(\varepsilon) = \varepsilon$*, and for all* $u \in \Sigma^*$*,* $D_{\mathcal{O}}(u.\lambda) = D_{\mathcal{O}}(u).\lambda$ *if* $\lambda \in \Gamma(\delta_o(x_0, u))$ *and* $D_{\mathcal{O}}(u)$ *otherwise. Then* $D_{\mathcal{O}}$ *is a dynamic projection.*

Proof. To prove that $D_{\mathcal{O}}$ defined above is a dynamic projection, it is sufficient to exhibit a dynamic observability choice $T : \Sigma^* \to 2^{\Sigma}$ and to show that (2) holds. Let $T(u) = \Gamma(\delta_o(x_o, D_{\mathcal{O}}(u)))$. It is easy to show by induction that $\delta_o(x_o, u) = \delta_o(x_o, D_{\mathcal{O}}(u))$ because $\delta_o(x, \lambda) = x$ when $\lambda \notin \Gamma(x)$. We can then define $T(u) = \Gamma(\delta_o(x_o, u))$ and the result follows from this remark. □

In the sequel, we shall write $[\![\mu]\!]_{\mathcal{O}}$ for $[\![\mu]\!]_{D_{\mathcal{O}}}$.

Proposition 3 *Given a dynamic projection* D *and* T_D *its dynamic observability choice, consider* $\mathcal{O}_D = (\Sigma^*, \varepsilon, \Sigma, \delta_D, T_D)$ *where* $\delta_D(u, \lambda) = D(u.\lambda)$*. Then* $\mathcal{O}_D$ *is a dynamic observer.*

Proof. $\mathcal{O}_D$ is complete and deterministic by construction and after a sequence s if $D(u.\lambda) = D(u)$ then $\delta_D(u, \lambda) = u$. □

Note that there might exists several equivalent observers that encode the same dynamic projection. For example, the observer depicted in Figure 4(b) is one observer that encodes the dynamic projection described in Example 4. But, one can consider other observers obtained by unfolding an arbitrary number of time the self-loops in states 1 or 3.Finally, to mimic the language theory terminology, we will

say that a dynamic projection D is *regular* whenever there exists a finite state dynamic observer $\mathcal{O}$ such that $D_{\mathcal{O}} = D$.

To summarize this part, we can state that with each dynamic projection D, we can associate a dynamic observer $\mathcal{O}_D$ such that $D = D_{\mathcal{O}_D}$. In other words, we can consider a dynamic projection or one of its associated dynamic observers whenever one representation is more convenient than the other. If the dynamic projection D derived from $\mathcal{O}$ is valid, we say that $\mathcal{O}$ is a *valid* dynamic observer. In that case, we will say that F is opaque w.r.t. $(G, \mathcal{O})$ and we denote by $\mathcal{OBS}(G)$ the set of all valid dynamic observers.

4.1.3 Checking opacity

The first problem we are going to address consists in checking whether a given dynamic projection ensures opacity. To do so, we assume given a dynamic observer which defines this projection map. The problem, we are interested in, is then the following:

Problem 3 (Dynamic State Based Opacity Problem)

INPUT: *A non-deterministic FA $G = (Q, q_0, \Sigma, \delta, F)$ and a dynamic observer $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$.*

PROBLEM: *Is F opaque w.r.t. $(G, \mathcal{O})$?*

We first construct an automaton which represents what an attacker will see under the dynamic choices of observable events made by $\mathcal{O}$ (i.e. by hiding in G the events the observer have chosen to hide after observing a given trace). To do so, we define the automaton $G \otimes \mathcal{O} = (Q \times X, (q_0, x_0), \Sigma \cup \{\tau\}, \delta, F \times X)$ where τ is a fresh letter not in Σ and δ is defined for each $\lambda \in \Sigma$, and $(q, x) \in Q \times X$ by:

- $\delta((q, x), \lambda) = \delta_G(q, \lambda) \times \{\delta_o(x, \lambda)\}$ if $\lambda \in \Gamma(x)$;
- $\delta((q, x), \tau) = \big(\cup_{\lambda \in \Sigma \setminus \Gamma(x)} \delta_G(q, \lambda)\big) \times \{x\}$.

Proposition 4 *F is opaque w.r.t. $(G, \mathcal{O})$ if and only if $F \times X$ is opaque w.r.t. to $(G \otimes \mathcal{O}, \Sigma)$.*

Proof. Let $\mu \in Tr_{\mathcal{O}}(G)$ be a trace observed by the attacker. We prove the following by induction on the length of μ:

$$q \in Post_G(\{q_0\}, [\![\mu]\!]_{\mathcal{O}}) \iff (q, x) \in Post_{G \otimes \mathcal{O}}(\{(q_0, x_0)\}, [\![\mu]\!]_{\Sigma}) \text{ for some } x \in X.$$

If $\mu = \varepsilon$, the result is immediate. Assume now that $\mu' = \mu.\lambda$. Let $q \in Post_G(\{q_0\}, [\![\mu']\!]_{\mathcal{O}})$. By definition of $[\![\mu']\!]_{\mathcal{O}}$ we have

$$q_0 \xrightarrow{u} q' \xrightarrow{v} q'' \xrightarrow{\lambda} q$$

with $u \in [\![\mu]\!]_{\mathcal{O}}$, $u.v.\lambda \in [\![\mu.\lambda]\!]_{\mathcal{O}}$. By induction hypothesis, it follows that the state $(q', \delta_o(x_0, u)) \in Post_{G \otimes \mathcal{O}}(\{(q_0, x_0)\}, [\![\mu]\!]_{\Sigma})$ where $\delta_o(x_0, u)$ is the (unique) state of $\mathcal{O}$ after reading u. Then, there exists a word $w \in (\Sigma \cup \{\tau\})^*$ such that $P_{\Sigma}(w) = \mu$ and $(q_0, x_0) \xrightarrow{w} (q', \delta_o(x_0, u))$ is a run of $G \otimes \mathcal{O}$. Assume $v = v_1.v_2.\cdots.v_k, k \geq 0$. As $\mathcal{O}(u.v) = \mathcal{O}(u)$, we must have $v_i \notin \Gamma(\delta_o(x_0, u.v_1.\cdots.v_i))$ when $1 \leq i \leq k$. Hence, by construction of $G \otimes \mathcal{O}$, there is a sequence of transitions in $G \otimes \mathcal{O}$ of the form

$$(q', \delta_o(x_0, u)) \xrightarrow{\tau} \delta_o(x_0, u.v_1) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (q'', \delta_o(x_0, u.v))$$

with $\lambda \in \Gamma(\delta_o(x_0, u.v))$. Thus, $(q_0, x_0) \xrightarrow{w} (q', \delta_o(x_0, u)) \xrightarrow{\tau^k.\lambda} (q, \delta_o(u.v.\lambda))$ is a run of $G \otimes \mathcal{O}$ with $P_{\Sigma}(w.\tau^k.\lambda) = \mu.\lambda = \mu'$. So, $(q, \delta_o(x_0, u.v.\lambda)) \in Post_{G \otimes \mathcal{O}}(\{(q_0, x_0)\}, [\![\mu']\!]_{\Sigma})$. The converse is also easy to establish: if we have a sequence of τ transitions in $G \otimes \mathcal{O}$, they come from actions in G which are not observable and this completes the proof. □

The previous result is general, and if $\mathcal{O}$ is a finite state observer we obtain the following theorem:

Theorem 3 *For finite state observers, Problem 3 is PSPACE-complete.*

Proof. As the size of the product $G \otimes \mathcal{O}$ is the product of the size of G and the size of $\mathcal{O}$ and State Based Opacity can be checked in PSPACE, PSPACE-easiness follows. Moreover, checking state based opacity with respect to (G, Σ) can de done using a simple observer with one state which always let Σ observable and PSPACE-hardness follows. □

As Proposition 4 reduces the problem of checking opacity with dynamic observers to the problem of checking opacity with static observers, Theorem 3 extends to family of sets (and thus to secrecy).

4.2 Enforcing opacity with dynamic projections

So far, we have assumed that the dynamic projection/observer was given. Next we will be interested in *synthesizing* one in such a way that the secret becomes opaque w.r.t. the system and this observer. Initially we assume that the attacker can observe all events in Σ. The problem we have in mind is to add an interface (a dynamic observer/projection) between the system and the attacker that will filter out some events so that the confidential information never leaks following the architecture of Figure 1. Thus the problem can be stated as follows:

Problem 4 (Dynamic Observer Synthesis Problem)
INPUT: *A non-deterministic FA* $G = (Q, q_0, \Sigma, \delta, F)$.
PROBLEM: *Compute the set of valid observers* $\mathcal{OBS}(G)^2$.

Deciding the existence of a valid observer is trivial: it is sufficient to check whether always hiding Σ is a solution. Moreover, note that $\mathcal{OBS}(G)$ can be infinite and that there might be an infinite number of different valid projections/observers ensuring the opacity of F with respect to G.
To solve Problem 4, we reduce it to a safety 2-player game. Player 1 will play the role of an observer and Player 2 what the attacker observes. Assume the automaton G can be in any of the states $s = \{q_1, q_2, \cdots, q_n\}$, after a sequence of actions occurred. A round of the game is: given s, Player 1 chooses which letters should be observable next i.e., a set $t \subseteq \Sigma$; then it hands it over to Player 2 who picks up an observable letter $\lambda \in t$; this determines a new set of states G can be in after λ, and the turn is back to Player 1. The goal of the Players are defined by:

- The goal of Player 2 is to pick up a sequence of letters such that the set of states that can be reached after this sequence is included in F. If Player 2 can do this, then it can infer the secret F. Player 2 thus plays a *reachability game* trying to enforce a particular set of states, say *Bad* (i.e., the states in which the secret is disclosed).
- The goal of Player 1 is opposite: it must keep the game in a safe set of states where the secret is not disclosed. Thus Player 1 plays a *safety game* trying to keep the game in the complement set of *Bad*.

As we are playing a (finite) turn-based game, Player 2 has a strategy to enforce *Bad* iff Player 1 has no strategy to keep the game in the complement set of *Bad* (turn-based finite games are *determined* [17]).
We now formally defines the 2-player game and show it allows us to obtain a finite representation of all the valid dynamic observers. Let $H = (S_1 \cup S_2, s_0, M_1 \cup M_2, \delta_H)$ be a deterministic game automaton given by:

- $S_1 = 2^Q$ is the set of Player 1 states and $S_2 = 2^Q \times 2^\Sigma$ the set of Player 2 states;
- the initial state of the game is the Player 1 state $s_0 = \{q_0\}$;
- Player 1 will choose a set of events to hide in Σ. Thus, Player 1 actions are in the alphabet $M_1 = 2^\Sigma$ and Player 2 actions in $M_2 = \Sigma$;
- the transition relation $\delta_H \subseteq (S_1 \times M_1 \times S_2) \cup (S_2 \times M_2 \times S_1)$ is given by:
  - Player 1 moves (choice of events to observe): if $s \in S_1$, $t \subseteq \Sigma$, then $\delta_H(s, t) = (s, t)$;
  - Player 2 moves (choice of next observable event): if $(s, t) \in S_2$, $\lambda \in t$ and $s' = Post(s, (\Sigma \setminus t)^*.\lambda) \neq \emptyset$, then $\delta_H((s, t), \lambda) = s'$.

We define the set of *Bad* states to be the set of Player 1 states s s.t. $s \subseteq F$. For family of sets $F_1, F_2, \cdots, F_k$, *Bad* is the set of states $2^{F_1} \cup 2^{F_2} \cup \cdots \cup 2^{F_k}$.

Remark 4 *If we want to exclude the possibility of hiding everything for Player 1, it suffices to build the game H with this constraint on Player 1 moves (i.e., $\forall s \in S_1$, $Enabled(s) \neq \emptyset$). Using a similar method, we can also consider other kinds of constraints: for example, a valid observer could choose to hide outputs whenever it is necessary to preserve the secret, however, this observer must keep observable (and thus accepted) all the requests sent by the attacker to the system. To do so, assuming that Σ is partitioned into $\Sigma_? \cup \Sigma_!$, where $\Sigma_?$ denotes the set of inputs of the system and $\Sigma_!$ the set of outputs events, we can force the observer to choose to hide only events of $\Sigma_!$, letting observable all the actions performed by the attacker by building the game H so that $\forall s \in S_1$, $2^{\Sigma_?} \subseteq Enabled(s)$.* □

Let $Runs_i(H), i = 1, 2$ be the set of runs of H that end in a Player i state. A *strategy* for Player i is a mapping $f_i : Runs_i(H) \to M_i$ that associates with each run that ends in a Player i state, the new choice of Player i. Given two strategies f_1, f_2, the game H generates the set of runs $Outcome(f_1, f_2, H)$ combining the choices of Players 1 and 2 w.r.t. f_1 and f_2. f_1 is a *winning strategy* for Playing 1 in H for avoiding Bad if for all Player 2 strategies f_2, no run of $Outcome(f_1, f_2, H)$ contains a Bad state. A winning strategy for Player 2 is a strategy f_2 s.t. for all strategy f_1 of Player 1, $Outcome(f_1, f_2, H)$ reaches a Bad state. As turn-based games are determined, either Player 1 has a winning strategy or Player 2 has a winning strategy.

We now relate the set of winning strategies for Player 1 in H to the set of valid dynamic projections. Let $P_{M_2}(\varrho) = P_\Sigma(tr(\varrho))$ for a run ϱ of H.

Definition 9 *Given a dynamic projection D, we define a strategy f_D such that for every $\varrho \in Runs_1(H)$, $f_D(\varrho) = T_D(P_{M_2}(\varrho))$.* ■

Let $Outcome_1(f_1, H) = (\cup_{f_2} Outcome(f_1, f_2, H)) \cap Runs_1(H)$ be the set of runs ending in a Player 1 state which can be generated in the game when Player 1 plays f_1 against all the possible strategies of Player 2. The set of runs $Outcome_2(f_2, H)$ is similarly defined.

Lemma 1 *Let D be a dynamic projection. If $\varrho \in Outcome_1(f_D, H)$ and $\mu = P_{M_2}(tr(\varrho))$, then $\mu \in Tr_D(G)$ and $last(\varrho) = Post_G(s_0, [\![\mu]\!]_D)$.*

Proof. The proof is by induction on the length of μ:

1. base case: for the run s_0 it is trivial since $s_0 = \{q_0\}$, $[\![\varepsilon]\!]_D = \{\varepsilon\}$ and $\varepsilon \in Tr_D(G)$.
2. induction step: assume it is true for the run

$$\varrho = s_0 \xrightarrow{t_0} (s_0, t_0) \xrightarrow{\lambda_0} s_1 \cdots \longrightarrow \cdots s_n \xrightarrow{t_n} (s_n, t_n) \xrightarrow{\lambda_n} s_{n+1} \in Outcome_1(f_D, H)$$

Let $\varrho' = \varrho \xrightarrow{t_{n+1}} (s_{n+1}, t_{n+1}) \xrightarrow{\lambda_{n+1}} s_{n+2}$ be a run in $Outcome_1(f_D, H)$ and $\mu = P_{M_2}(tr(\varrho)) = \lambda_0.\lambda_1.\cdots.\lambda_n$. By definition of f_D, $t_{n+1} = f_D(\varrho) = T_D(\mu)$. Hence $\lambda_{n+1} \in T_D(\mu)$ and

$$\begin{aligned} s_{n+2} &= Post_G(s_{n+1}, (\Sigma \setminus t_{n+1})^*.\lambda_{n+1}) \\ &= Post_G(Post_G(s_0, [\![\mu]\!]_D), (\Sigma \setminus t_{n+1})^*.\lambda_{n+1}) \\ &= Post_G(s_0, [\![\mu]\!]_D.(\Sigma \setminus t_{n+1})^*.\lambda_{n+1}) \\ &= Post_G(s_0, [\![\mu.\lambda_{n+1}]\!]_D) \end{aligned}$$

By construction of H, $s_{n+2} \neq \emptyset$ thus $Post_G(\{q_0\}, [\![\mu.\lambda_{n+1}]\!]_D) \neq \emptyset$ and $\mu.\lambda_{n+1} \in Tr_D(G)$.

□

Lemma 2 *Let D be a dynamic projection. For all $\mu \in Tr_D(G)$, there exists a unique run $\varrho \in Outcome_1(f_D, H)$ such that $\mu = P_{M_2}(tr(\varrho))$.*

Proof. We prove this by induction on the length of μ. Note that as the game H is deterministic, and strategies prescribes one move, it suffices to prove the existence of a run. If $\mu = \varepsilon$, then s_0 is a good candidate. Suppose that every word in $Tr_D(G) \cap \Sigma^n$ satisfies the property of the Lemma and let $\mu.\lambda \in Tr_D(G) \cap \Sigma^{n+1}$. Since $\mu \in \Sigma^n$, there exists a run $\varrho \in Outcome_1(f_D, H)$ such that $\mu = P_{M_2}(tr(\varrho))$. Let $s = last(\varrho)$ and $t = f_D(\varrho) = T_D(\mu)$ the (unique) action Player 1 can perform after the run ϱ. Then $\delta_H(s, t) = (s, t)$ and $\varrho \xrightarrow{t} (s, t) \in Runs_2(H)$. According to Lemma 1, $s = Post_G(s_0, [\![\mu]\!]_D)$. Since $\mu.\lambda \in Tr_D(G)$, $\lambda \in t$ and $s' = Post_G(s_0, [\![\mu.\lambda]\!]_D) \neq \emptyset$. So $s' = Post_G(s, (\Sigma \setminus t)^*.\lambda) = \delta_H((s, t), \lambda)$ and $\varrho' = \varrho \xrightarrow{t} (s, t) \xrightarrow{\lambda} s'$ is a run of H. Hence $\varrho' \in Outcome_1(f_D, H)$ and such that $P_{M_2}(tr(\varrho')) = \mu.\lambda$. □

Proposition 5 *Let D be a dynamic projection. D is valid if and only if f_D is a winning strategy for Player 1 in H.*

Proof. Assume that D is a valid dynamic projection and let $\varrho \in Outcome_1(f_D, H)$ with $s = last(\varrho)$ and $\mu = P_{M_2}(tr(\varrho))$. According to Lemma 1, $s = Post_G(s_0, [\![\mu]\!]_D)$. Since D is a valid dynamic projection, $s \not\subseteq F$, so $s \notin Bad$. This implies that f_D is a winning strategy.

For the other implication, assume that D is not a valid observer. This means that there is a trace $\mu \in Tr_D(G)$ such that $Post_G(s_0, [\![\mu]\!]_D) \subseteq F$. Since $\mu \in Tr_D(G)$, then according to Lemma 2, there exists a unique run $\varrho \in Runs_1(H)$ such that $P_{M_2}(tr(\varrho)) = \mu$. Then, $last(\varrho) = Post_G(s_0, [\![\mu]\!]_D) \in Bad$, so f_D is not a winning strategy. □

Given a strategy f for Player 1 in H, for all $\mu \in \Sigma^*$, there exists at most one $\varrho_\mu \in Outcome_1(f, H)$ such that $P_{M_2}(tr(\varrho_\mu)) = \mu$.

Definition 10 *Let f be s strategy for Player 1 in H. We define the dynamic projection D_f induced by the dynamic observability choice $T_f : \Sigma^* \to 2^\Sigma$ given by: $T_f(\mu) = f(\varrho_\mu)$ if ϱ_μ.* ■

Proposition 6 *If f is a winning strategy for Player 1 in H, then D_f is a valid dynamic observer.*

Proof. Applying the construction of the Lemma 1, we obtain the strategy $f_{D_f} = f$. Since f is a winning strategy, by Proposition 5, we get that D_f is a valid dynamic projection. □

Notice that we only generate a representative for each of the equivalence classes induced by $\sim_G$. However, an immediate consequence of the two previous propositions is that there is a bijection between the set of winning strategies of Player 1 and $\mathcal{D}_{\sim_G}$.

4.3 Most Permissive Dynamic Observer

We now define the notion of *most permissive* dynamic observers and show the existence of a most permissive dynamic observer for a system G.

For an observer $\mathcal{O} = (X, x_o, \Sigma_o, \delta_o, \Gamma)$ and $w \in \Sigma^*$, recall that $\Gamma(\delta_o(x_o, w))$ is the set of events that $\mathcal{O}$ chooses as observable after observing w.

Assume $w = a_0 a_1 \cdots a_k$. Let $\overline{w} = \Gamma(x_o).a_1.\Gamma(\delta_o(x_o, a_1)).a_2.\Gamma(\delta_o(x_o, a_2)) \cdots a_k.\Gamma(\delta_o(x_o, a_k)))$ i.e., $\overline{w}$ contains the history of what $\mathcal{O}$ has chosen to observe at each step and the events that occurred after each choice.

Definition 11 *Let $\mathcal{O}^* : (2^\Sigma.\Sigma)^* \to 2^{2^\Sigma}$. By definition, such a mapping $\mathcal{O}^*$ is the most permissive observer[3] for ensuring that F is opaque if the following holds:*

$$\mathcal{O} = (X, x_o, \Sigma_o, \delta_o, \Gamma) \text{ is a valid observer} \iff \forall w \in L(G),\ \Gamma(\delta_o(x_o, w)) \in \mathcal{O}^*(\overline{w})$$

The definition of the most permissive observer states that any valid observer $\mathcal{O}$ must choose a set of observable events in $\mathcal{O}^*(\overline{w})$ on input w; if an observer chooses its set of observable events in $\mathcal{O}^*(\overline{w})$ on input w, then it is a valid observer.

Theorem 4 *The most permissive dynamic observer $\mathcal{O}^*$ can be represented by a finite automaton.*

Proof. As H is turn-based 2-player game under full observation, there is a *most permissive winning strategy* which is state-based on H (see [23]). It is defined as follows: Let us first compute the set of winning states of the game for player 1 It is defined as follows: Let $Good = (S_1 \cup S_2) \setminus Bad$ be the set of safe states of H. To solve this 2-player game, we define the $Cpre$ operator:

$$CPre(S) = \quad \{s \in S_1 \mid \exists t \subseteq \Sigma \mid \delta_H(s, t) \in S\} \cup \{(s, t) \in S_2 \mid \forall \lambda \in t \mid \delta_H((s, t), \lambda) \in S\}$$

Then by iterating $Cpre$ and computing the fix-point $Cpre^*(Good) = \cap_i Cpre^i(Good)$, we obtain the set of winning states of the game for Player 1 [23], and as the set of states is finite, this computation terminates. If the initial state of the game belongs to $Cpre^*(Good)$, then there is a strategy for Player 1 to win. Consider now the following finite automaton derived from H: $\mathcal{F}_H = (Cpre^*(Good), s_0, \Sigma \cup 2^\Sigma, \delta_{\mathcal{F}_H})$, where $\delta_{\mathcal{F}_H}$ is the restriction of δ_H to the states $Cpre^*(Good)$. Now, f is a winning strategy for Player 1 w.r.t. H and Bad if and only if for any run $\rho \in Runs_1(H)$, $f(\rho) \in Enabled_{\mathcal{F}_\mathcal{H}}(last(\rho))$, namely, every

[3]Strictly speaking $\mathcal{O}^*$ is not an observer because it maps to sets of sets of events whereas observers map to sets of events. Still we use this term because it is the usual terminology in the literature.

move defined by f is a move of $\mathcal{F}_H$. In other words, any winning strategy in H is an *instance* of $\mathcal{F}_H$. Now from Proposition 6, given a winning strategy, we can define a valid dynamic projection from which we can derive a valid dynamic observer (Proposition 3). Conversely, with each valid dynamic observer D is associated a valid dynamic projection $D_{\mathcal{O}}$ (Proposition 2) and from Proposition 5, $f_{D_{\mathcal{O}}}$ is a winning strategy which thus can be generated by $\mathcal{F}_H$. □

The previous theorem states that $\mathcal{F}_H$ can be used to generate any observer. In particular, given a state-based winning strategy, the corresponding valid observer is finite and thus its associated dynamic projection is regular.

An immediate corollary of Theorem 4 is the following:

Corollary 1 *Problem 4 is in EXPTIME.*

Proof. Computing the winning states and $\mathcal{F}_H$ on turn-based games can be done in linear time in the size of the game. As H has size exponential in G and Σ the algorithm we provide to solve Problem 4 is EXPTIME. □

It remains open whether Problem 4 is EXPTIME-complete.

Example 6 *To illustrate this section, we consider the following small example. The system is depicted by the automaton in Figure 5(a). The set of secret states is reduced to the state (2). Figure 5(b)*

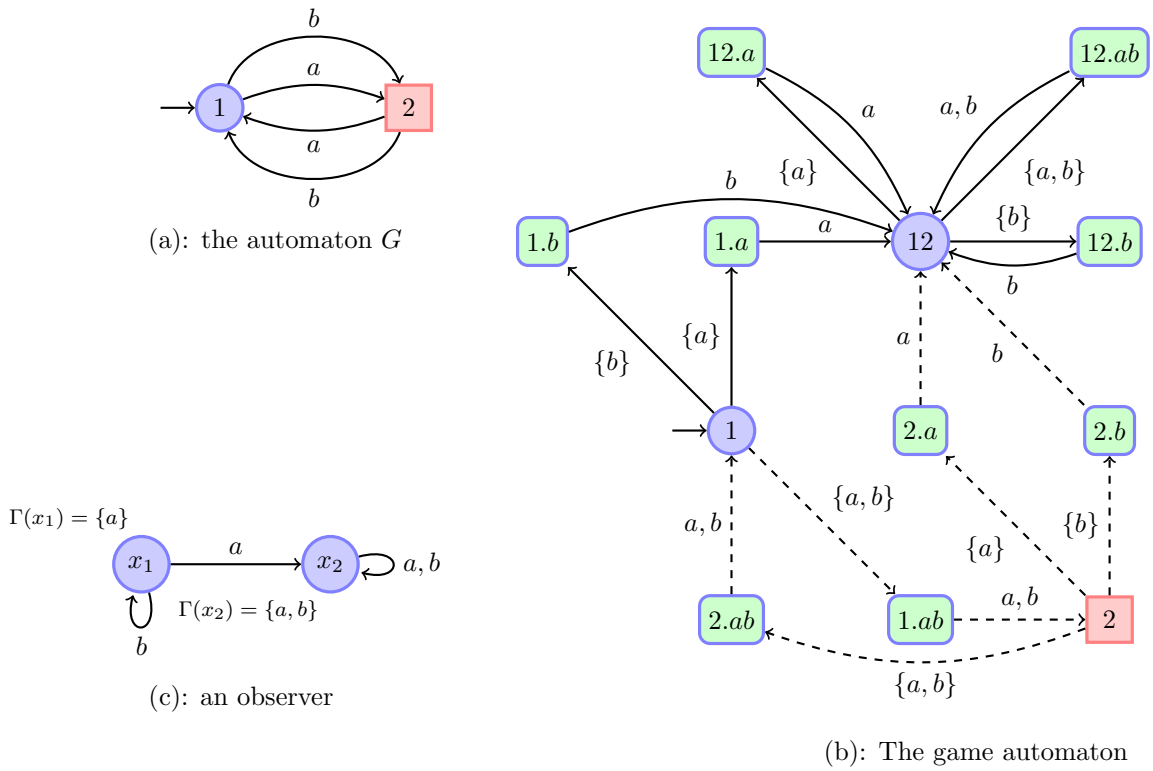


Figure 5: Example of a game automaton

represents the associated game automaton. The states of Player 1 are represented by circles whereas the ones of Player 2 are represented by squares. The only bad states is the state (2) (bottom right). The most permissive strategy is obtained when Player 1 does not allow transition $\{a, b\}$ to be triggered in state (1) (otherwise, Player 2 could chose to observe either event a or b and in this case the game will evolve into state (2) and the secret will be revealed). The dashed lines represents the transitions that are removed from the game automaton to obtain $\mathcal{F}_H$. Finally, Figure 5(c) represents a possible observer $\mathcal{O}$ generated from the game automaton. ◇

5 Optimal Dynamic Observer

Among all the possible observers that ensure the opacity of the secret, it is worthwhile noticing that some are better (in some sense) than other. Note that if $q_0 \notin F$, then there exists a trivial observer that always erases all events of Σ that ensures the opacity of F. The attacker will observe nothing and will thus not be able to disclose the secret. Obviously, this observer is not very interesting as the effect would be to deny services from the standpoint of a user. In this section, we would like to define a notion of cost of for observers which captures this intuitive notion.

We first introduce a general cost function and we show how to compute the cost of a given pair $(G, \mathcal{O})$ where G is a system and $\mathcal{O}$ a finite state observer. Second, we show that among all the valid observers (that ensure opacity), there is an optimal cost, and we can compute an observer which ensures this cost. The problems in this section and the solutions are closely related to the results in [5] and use the same tools: Karp algorithm [13] and a result of Zwick and Paterson [24].

5.1 Cost of an observer

We want to define a notion of cost which takes into account the set of events the observer chooses to hide and also how long it hides them. We assume we are given a system $G = (Q, q_0, \Sigma, \delta, F)$ and a finite observer $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$.

With each set of observable events $\Sigma' \in 2^\Sigma$, we associate a *cost of hiding* $\Sigma \setminus \Sigma'$ which is a positive integer. We denote $\mathit{Cost} : 2^\Sigma \to \mathbb{N}$ this function. Now, if the observer $\mathcal{O}$ is in state x, the current cost per time unit is $\mathit{Cost}(\Gamma(x))$. Examples of such cost mapping are given in the sequel. A measure of the cost incured by an observer $\mathcal{O}$ on a sequence of events w produced by G is just the average of the cost of hiding events.

Definition 12 (Cost of a run) *Let $\mathit{Runs}^n(G)$ the set of runs of length n in $\mathit{Runs}(G)$. Given a run $\rho = q_0 \xrightarrow{a_1} q_1 \cdots q_{n-1} \xrightarrow{a_n} q_n \in \mathit{Runs}^n(G)$, let $x_i = \delta_o(x_0, w_i)$ with $w_i = \mathit{tr}(\rho[i])$. The* cost *associated with $\rho \in \mathit{Runs}^n(G)$ is defined by:*

$$\mathit{Cost}(\rho, G, \mathcal{O}) = \frac{1}{n+1} \cdot \sum_{i=0}^{n} \mathit{Cost}(\Gamma(x_i)).$$

■

Notice that the time basis we take is the number of steps which occurred in G. Thus if the observer is in state x, and chooses to observe $\Gamma(x)$ at steps i and $i+1$ in a run of G, $\mathit{Cost}(\Gamma(x))$ will be counted at steps i and $i+1$. The definition of the cost of a run corresponds to the average cost per time unit, the time unit being the number of steps of the run in G.

Definition 13 (Cost of an observer) *Define the cost of the set of runs of length n that belongs to $\mathit{Runs}^n(G)$ by*

$$\mathit{Cost}(n, G, \mathcal{O}) = \max\{\mathit{Cost}(\rho, G, \mathcal{O}) \mid \rho \in \mathit{Runs}^n(G)\} \tag{4}$$

and the cost of an observer with respect to a system G is

$$\mathit{Cost}(G, \mathcal{O}) = \limsup_{n \to \infty} \mathit{Cost}(n, G, \mathcal{O}) \tag{5}$$

(the limit may not exist whereas the limit sup is always defined.) ■

Here are examples for the cost mapping *Cost* on 2^Σ:

- Suppose that we have a function $c : \Sigma \to \mathbb{N}$ that associates with each $a \in \Sigma$ the cost $c(a)$ corresponding to the penalty if this event is hidden. Hiding $\Sigma' \subseteq 2^\Sigma$ costs the sum of the cost of the hidden events. Hence $\mathit{Cost}(\Gamma(x)) = \sum_{a \in \Sigma \setminus \Gamma(x)} c(a)$ and we can set $\mathit{Cost}(\Sigma)$ to a particular value.
- Assume $\Sigma = \{a, b\}$ and $c(a) = c(b) = 2$; hiding a and b at the same time costs more than hiding a alone or b alone. This can be captured by $\mathit{Cost}(\emptyset) = 7$ and $\mathit{Cost}(\{a\}) = 2$ and $\mathit{Cost}(\{b\}) = 2$.

Remark 5 *Alternatively, one can define the cost of a run as follows: Given a run* $\rho = q_0 \xrightarrow{a_1} q_1 \cdots q_{n-1} \xrightarrow{a_n} q_n \in Runs^n(G)$*, let* $x_i = \delta_o(x_0, w_i)$ *with* $w_i = tr(\rho[i])$*. The* cost *of* $\rho \in Runs^n(G)$ *is defined by:*

$$Cost2(\rho, G, \mathcal{O}) = \frac{1}{n+1} \cdot \sum_{i=0}^{n} \begin{cases} c(a_i) & \text{if } a_i \notin L(\delta_o(x_0, w_i)) \\ 0 & \text{otherwise} \end{cases} .$$

Compared with $Cost(\rho, G, \mathcal{O})$*, we only take into account the costs of the events that are filtered by the observer when they actually occur in the run* ρ*. Further, the cost of an observer is given by Definition 13.*

5.2 Computing the cost of a Dynamic Observer

We show that we can compute the cost of a given finite state observer $\mathcal{O}$ for a finite state system G. We reduce the problem of computing this cost to computing the *maximum mean-weight* in a weighted graph.

5.2.1 Weighted Automata & Karp's Algorithm

Before introducing the optimal dynamic observer and opacity synthesis problem, we present a set of tools that we are going to use in that process. These deal with the notion of cost in a model of dynamic behaviors such as a finite automaton model. The notion of cost for automata has already been defined and algorithms to compute some optimal values related to this model are described in many papers. For our purposes, the model of *weighted automata* is appropriate. We recall here this model and the results of [13] which will be used later.

Definition 14 (Weighted Automaton) *A* weighted automaton *is a pair* (A, w) *s.t.* $A = (Q, q_0, \Sigma, \delta)$ *is a finite automaton and* $w : Q \to \mathbb{N}$ *associates a weight with each state.* ■

Definition 15 (Mean Cost) *Let* $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ *be a run of* A*. The* mean cost *of* ρ *is*

$$\mu(\rho) = \frac{1}{n+1} \times \sum_{i=0}^{n} w(q_i) .$$

■

We remind that the length of $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ is $|\rho| = n$. We assume that each run of length n of A can be extended to a run of length $n+1$. Let $Runs^n(A)$ be the set of runs of length n in $Runs(A)$. The *maximum mean-weight* of the runs of length n for A is $\nu(A, n) = \max\{\mu(\rho) \text{ for } \rho \in Runs^n(A)\}$. The *maximum mean weight* of A is $\nu(A) = \limsup_{n\to\infty} \nu(A, n)$. Actually the value $\nu(A)$ can be computed using Karp's maximum mean-weight cycle algorithm [13] on weighted graphs. If $c = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ is a cycle of A i.e., $q_0 = q_n$, the *mean weight* of the cycle c is $\mu(c) = \frac{1}{n+1} \cdot \sum_{i=0}^{n} w(q_i)$. The *maximum mean-weight cycle* of A is the value $\nu^*(A) = \max\{\mu(c) \text{ for } c \text{ a cycle of } A\}$. As stated in [24], for weighted automata, the mean-weight cycle value is the value that determines the mean-weight value (the transient behaviors of the system are not contributing to this value). It follows that $\nu(A) = \limsup_{n\to\infty} \nu(A, n) = \lim_{n\to\infty} \nu(A, n) = \nu^*(A)$.

The original Karp's maximum mean cycle algorithm [13] works for weighted automaton where the weights are on the edges. We give the version where weights are on vertices. Let $\nu^* = \max_c \mu(c)$ where c ranges over all cycles in A. A cycle c with $\mu(c) = \nu^*$ is a *maximum mean-weight cycle*. Let $D(q)$ be the weight of a most expensive path from q_0 to q and $D_k(q)$ be the weight of a most expensive path which has exactly k edges (if there is no such path $D_k(v) = -\infty$). Assume $|Q| = n$. Karp's algorithm is based on the fact that

$$\nu^* = \max_{q\in Q} \min_{0\le k\le n-1} \frac{D_n(q) - D_k(q)}{n-k}$$

The values $D_k(q)$ can be computed iteratively:

$$D_0(q_0) = w(q_0) \tag{6}$$

$$D_0(q) = -\infty \quad \text{for} \quad q \neq q_0 \tag{7}$$

$$D_{k+1}(q) = \max_{q\in\delta(q',a)} \{D_k(q') + w(q)\} \tag{8}$$

Thus for each state q we can compute $\min(q) = \min_{0 \leq k \leq n-1} \frac{D_n(q) - D_k(q)}{n-k}$ and then compute the value $\max_{q \in Q} \min(q)$ to obtain ν^*. This algorithm runs in $O(n.m)$ where $|Q| = n$ and $|\delta| = m$ (where $|\delta|$ denotes the number of transitions in δ). Improvements [7] can be made to this algorithm still the worst case run-time is $O(n.m)$.

5.2.2 Algorithm

We assume that G generates runs of arbitrary length for simplicity. We use the automaton $G \otimes \mathcal{O}$ defined in Section 4.1.3. Remind that $G \otimes \mathcal{O} = (Q \times X, (q_o, x_0), \Sigma \cup \{\tau\}, \delta, F \times X)$. We define the weight function for $G \otimes \mathcal{O}$ as: $w(q, x) = Cost(\Gamma(x))$. Thus $\mathcal{G} = (G \otimes \mathcal{O}, w)$ is a weighted automaton.
By Definition 12 and Definition 15, and the results related to the Karp's maximum mean cycle algorithm described in Section 5.2.1, it is easy to see that:

Theorem 5 *$Cost(G, \mathcal{O}) = \nu^*(\mathcal{G})$.*

Using Karps's algorithm it follows that:

Theorem 6 *Computing $Cost(G, \mathcal{O})$ is in PTIME.*

Proof. The size of $G \otimes \mathcal{O}$ is clearly polynomial in the size of G and $\mathcal{O}$. □

Example 7 *To illustrate the construction of the weighted automaton, let us consider the system G and the observer $\mathcal{O}_2$ of example 6. Assume that $c(a) = c(b) = 2$ and $c(c) = 0$. The automaton is given in Figure 6. The weight function is pictured near each state. The values $D_k(v)$ and $min(v)$ for each state*

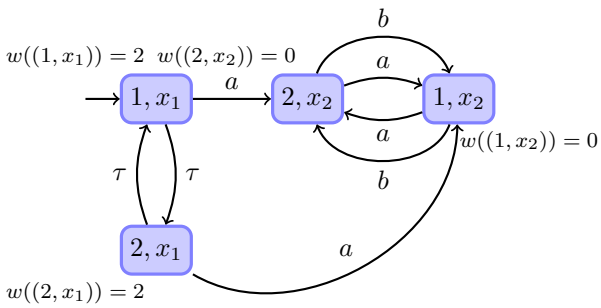


Figure 6: $\mathcal{G} = (G \otimes \mathcal{O}_2, w)$

v are given in Table 1. The maximum mean-weight value $\nu^*(\mathcal{G})$ is the maximum value $\max_v \min(v)$ for every state v of $G \times \mathcal{O}$. We thus obtain $Cost(G, \mathcal{O}) = 2$ □

	$1, x_1$	$2, x_2$	$1, x_2$	$2, x_1$
D_0	2	$-\infty$	$-\infty$	$-\infty$
D_1	$-\infty$	2	$-\infty$	4
D_2	6	$-\infty$	4	$-\infty$
D_3	$-\infty$	6	$-\infty$	8
D_4	10	$-\infty$	8	$-\infty$
min	**2**	$-\infty$	**2**	$-\infty$

with
$$\begin{cases} D_{k+1}(1, x_1) = D_k(2, x_1) + 2 \\ D_{k+1}(2, x_2) = max\{D_k(1, x_1), D_k(1, x_2)\} \\ D_{k+1}(1, x_2) = max\{D_k(2, x_2), D_k(2, x_1)\} \\ D_{k+1}(2, x_1) = D_k(1, x_1) + 2 \end{cases}$$

Table 1: Iterations for $G \otimes \mathcal{O}$.

5.3 Optimal Dynamic Observer

We now turn our attention to a more difficult decision problem:

Problem 5 (Bounded Cost Observer)

INPUTS: *an automaton* $G = (Q, q_0, \Sigma, \delta, F)$ *and an integer* $k \in \mathbb{N}$.

PROBLEM:

(A) *Is there an observer* $\mathcal{O} \in \mathcal{OBS}$ *s.t.* F *is opaque w.r.t.* G *and* $\mathcal{O}$ *and* $Cost(G, \mathcal{O}) \leq k$ *?*

(B) *If the answer to (A) is "yes", compute a witness observer* $\mathcal{O}$ *s.t.* $Cost(G, \mathcal{O}) \leq k$.

To solve this problem we use a result By Zwick and Paterson [24], which is an extension of Karp's algorithm for finite state games.

5.3.1 Zwick and Paterson's Algorithm

Definition 16 (Weighted Graph) *A* weighted directed graph *is a pair* (G, w) *s.t.* $G = (V, E)$ *is a directed graph and* $w : E \to \{-W, \cdots, 0, \cdots, W\}$ *assigns an integral weight to each edge of* G *with* $W \in \mathbb{N}$. *We assume that each vertex* $v \in V$ *is reachable from a unique* source *vertex* v_0 *and has at least one outgoing transition.* ■

Definition 17 (Weighted Graph Game) *A* weighted graph game $G = (V, E)$ *is a bipartite weighted graph with* $V = V_1 \cup V_2$ *and* $E = E_1 \cup E_2$, $E_1 \subseteq V_1 \times V_2$ *and* $E_2 \subseteq V_2 \times V_1$. *We assume the initial vertex* v_0 *of* G *belongs to* V_1. ■

Vertices V_i are Player i's vertices. A weighted graph game is a turn based game in which the turn alternates between Player 1 and Player 2. The game starts at a vertex $v_0 \in V_1$. Player 1 chooses an edge $e_1 = (v_0, v_1)$ and then Player 2 chooses an edge $e_2 = (v_1, v_2)$ and so on and they build an infinite sequence of edges. Player 2 wants to maximise $\liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=1}^{n} w(e_i)$ and Player 1 wants to minimize $\limsup_{n\to\infty} \frac{1}{n} \cdot \sum_{i=1}^{n} w(e_i)$.

One of the result of [24] is that there is a rational value $\nu \in \mathbb{Q}$ s.t. Player 2 has a strategy to ensure $\liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=1}^{n} w(e_i) \geq \nu$ and Player 1 has a strategy to ensure that $\limsup_{n\to\infty} \frac{1}{n} \cdot \sum_{i=1}^{n} w(e_i) \leq \nu$. ν is called the value of the game.

Let $n = |V|$. To compute ν, proceed as follows [24]:

1. Let $\nu_0(v) = 0$ for $v \in V$. For $v \in V$ and $k \geq 1$, $\nu_k(v)$ is defined by:

$$\nu_k(v) = \begin{cases} \min_{(v,w)\in E}\{w(v,w) + \nu_{k-1}(w)\} & \text{if } v \in V_1 \\ \max_{(v,w)\in E}\{w(v,w) + \nu_{k-1}(w)\} & \text{if } v \in V_2 \end{cases}$$

   This is the equivalent of the $D_k(v)$ values for Karp's algorithm using a min max strategy depending on which player is playing;

2. for each $v \in V$, compute $\nu'(v) = \nu_k(v)/k$ for $k = 4 \cdot n^3 \cdot W$.

3. for each vertex, the value of the game from v is the only rational number with a denominator at most n that lies in the interval $]\nu'(v) - \alpha, \nu'(v) + \alpha[$ with $\alpha = \frac{1}{2n(n-1)}$.

The value of the game is $\nu = \nu(v_0)$ where v_0 is the initial vertex. To compute an optimal strategy for Player 1, proceed as follows:

1. compute the values $\nu(v)$ for each $v \in V$;

2. if all the vertices of V_1 have outgoing degree 1, there is a unique strategy and it is positional and optimal;

3. otherwise, take a vertex $v \in V_1$ with outgoing degree $d \geq 2$. Remove $\lceil \frac{d}{2} \rceil$ edges from v leaving at least one. Recompute the value m_v for each v. If $m_v = \nu(v)$, there is an optimal positional strategy which uses the remaining edges from v. Otherwise there is a positional strategy that uses one of the removed edges.

We can iterate the previous scheme to find an optimal strategy for Player 1. In summary some of the results by Zwick and Paterson [24] we are going to use are:

- there is a value $\nu \in \mathbb{Q}$, called the *value of the game* s.t. Player 2 has a strategy to ensure that $\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} w(e_i) \geq \nu$ and Player 1 has a strategy to ensure that $\limsup_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} w(e_i) \leq \nu$; this value can be computed in $O(|V|^3 \times |E| \times W)$ where W is the range of the weight function (assuming the weights are in the interval $[-W..W]$). Note that deciding whether this value satisfies $\nu \bowtie c$ for $\bowtie \in \{=, <, >\}$ for $c \in \mathbb{Q}$ can be done in $O(|V|^2 \times |E| \times W)$.

- there are optimal memoryless strategies for both players that can be computed in $O(|V|^4 \times |E| \times \log(|E|/|V|) \times W)$.

5.3.2 Synthesis of an Optimal Observer

To solve the Problem 5, we use the most permissive strategy $\mathcal{F}_H$ we computed in Section 4.2. Given G and $\mathcal{F}_H$, we build a weighted graph game $WG(G, \mathcal{F}_H)$ s.t. the value of the game is the optimal cost for the set of all observers. Moreover an optimal observer can be obtained by taking an optimal memoryless strategy in $WG(G, \mathcal{F}_H)$.

To build $WG(G, \mathcal{F}_H)$ we first make $\mathcal{F}_H$ complete. This is to enable $\mathcal{F}_H$ to synchronize with G even if the event chosen by G from the current state is not among the one that $\mathcal{F}_H$ has chosen to observe. Let (x, L) be a state of $\mathcal{F}_H$ with L the set of events we have chosen to observe. For each event $\lambda \in \Sigma \setminus L$, we add a transition to a new state $x(L)$, and from $x(L)$ a transition to (x, L) with the choice of observable events L: this is as if we can observe λ but we keep the same choice of observable events when λ occurs. Doing this, we obtain the complete most permissive strategy $\mathcal{F}'_H$. We do this in order to take into account the current choice of the observer at each step taken by G which is the time basis.

Each transition of $\mathcal{F}'_H$ is assigned a cost: if it is a transition labeled by $\lambda \in \Sigma$, the weight is zero; if it is labeled by $\Sigma' \subseteq \Sigma$, the weight is $Cost(\Sigma')$. In the product $G \times \mathcal{F}'_H$, either G and $\mathcal{F}'_H$ make a joint synchronized transition (Σ) or only $\mathcal{F}'_H$ makes a transition (labeled in 2^{Σ}). We can assign a weight to a transition of $G \times \mathcal{F}'_H$ by taking the weight of the transition taken by $\mathcal{F}'_H$. This way we obtain the weighted automaton $WG(G, \mathcal{F}_H)$.

Using Zwick and Paterson's algorithm on $WG(G, \mathcal{F}_H)$, we can solve the following problem:

Problem 6 (Optimal Cost Observer)

INPUTS: *an automaton G, a secret F.*

PROBLEM: *Compute the least $r \in \mathbb{Q}$ s.t. there is a valid observer $\mathcal{O}$ s.t. F is opaque w.r.t. $(G, \mathcal{O})$ and $Cost(G, \mathcal{O}) \leq r$.*

By construction of $WG(G, \mathcal{F}_H)$ we have:

Theorem 7 *Let ν be the value of the game $WG(G, \mathcal{F}_H)$. Then $r = 2 \cdot \nu$.*

Proof. By construction of $WG(G, \mathcal{F}_H)$, one time unit of G is simulated by 2 steps of $WG(G, \mathcal{F}_H)$. Hence the result. □

Corollary 2 *Problem 5 and 6 can be solved in $O(2^{|G|})$.*

Proof. Checking whether $c \leq 2 \cdot \nu$ can be done in $O(|V|^2 \times |E| \times W)$ if the game has $|V|$ vertices, $|E|$ edges and the maximal constant is W. Computing ν can be done in $O(|V|^3 \times |E| \times W)$. Using our construction, $WG(G, \mathcal{F}_H)$ has at most $|Q| \cdot (2^{|Q|} + (2^{|Q|} \cdot 2^{|\Sigma|}))$ vertices. Computing the strategies can also be done in polynomial time in the size of the game. □

Example 8 *Coming back to Example 6, we first need to complete the game automaton depicted in Figure5(c). This leads to the new automaton $\mathcal{F}'_H$ given in (Figure 7). Further, we need to make the product between G and $\mathcal{F}'_H$ and to attach to each transition of the game automaton its weight (we here assume that $c(a) = c(b)$). The result is depicted in Figure 8.*

Using Zwick and Paterson's algorithm we obtain that $\nu = c(a)$ is 2 and the most permissive weight strategy is obtained by removing the dashed transitions in the game of Figure 8. This leads to two different observer $\mathcal{O}_1$ and $\mathcal{O}_2$ presented in Figure 9(a) and (b).

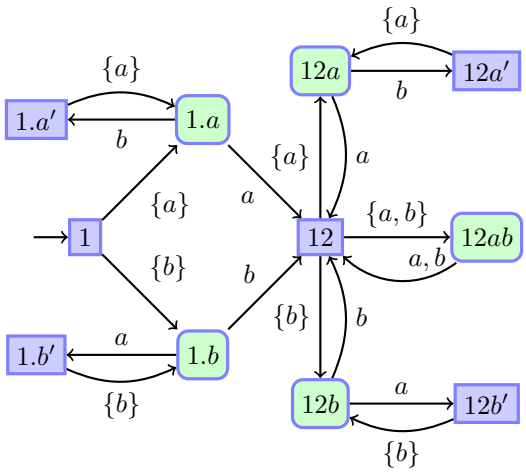


Figure 7: $\mathcal{F}'_H$

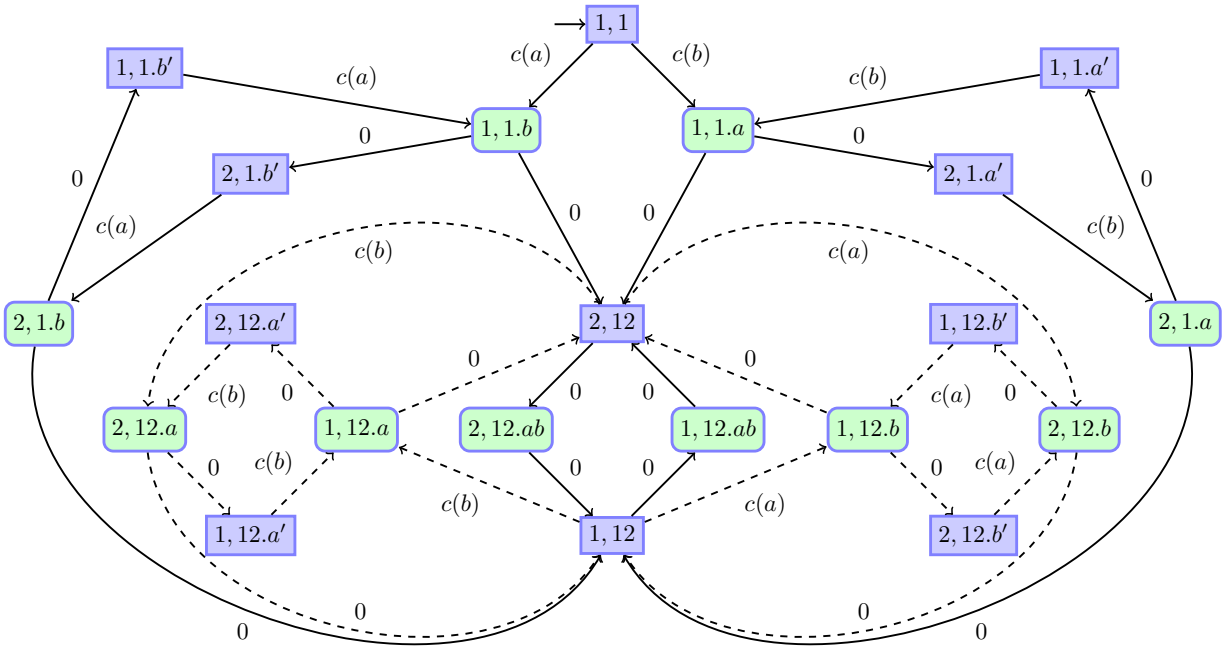
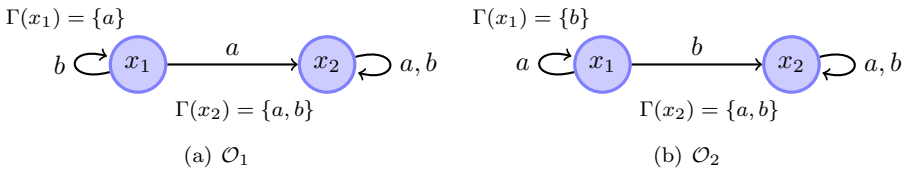


Figure 8: The weighted game automaton



(a) $\mathcal{O}_1$ (b) $\mathcal{O}_2$

Figure 9: The two optimal observers

6 Conclusion

In this article, we have investigated the synthesis of opaque system when the observability of events can change over time. In the context of static observers, where the observability of events is fixed a priori, we provided an algorithm (PSPACE-complete) to compute a maximal subalphabet of observable actions ensuring opacity. Then, we considered the case where the observability of events can be adapted during system's execution. We defined a model of dynamic observers determining whether an event is observable after a given observed trace. We formulated the notion of valid observer for an observer ensuring opacity and we proved that verifying the validity of a finite state observer is PSPACE-complete. We provided a method to compute the set of all valid observers by computing the set of winning strategies in a turn-based safety two-player game and demonstrated in that case that the set of all valid observers can be finitely represented. We have shown that the previous problem can be solved in exponential time. EXPTIME-completeness is left open.

In this article, we assumed that the dynamic observers can change the set observable events only after an observable event has occurred. This assumption should fit most applications since the knowledge of the attacker also depends on observed traces. It would be interesting to investigate also the case where this decision depends on the word executed by the system. The case where the observability depends on the state of the system should also be considered. Finally, the notion of semantics of an observed trace used throughout this article is based on the assumption that attacker can react, i.e. acquire knowledge, faster that system's evolution. It would be interesting to adapt this work to other notion of semantic, i.e. for other indistinguishably relation over system's execution.

References

[1] Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving secrecy under refinement. In *ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming*, pages 107–118. Springer, 2006.

[2] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17:425–446, December 2007.

[3] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340, Chicago, IL, June 2005. IEEE Computer Society.

[4] Jeremy W Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, May 2008.

[5] Franck Cassez and Stavros Tripakis. Fault diagnosis with static or dynamic diagnosers. *Fundamenta Informatica*, 88(4):497–540, November 2008.

[6] V. Darmaillacq, J.-C. Fernandez, R. Groz, L. Mounier, and J.-L. Richier. Test generation for network security rules. In *TestCom 2006*, volume 3964 of *LNCS*, 2006.

[7] Ali Dasdan, Sandy S. Irani, and Rajesh K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Annual ACM IEEE Design Automation Conference*, pages 37–42, New Orleans, Louisiana, United States, 1999. ACM Press New York, NY, USA. ISBN:1-58133-109-7.

[8] Jérémy Dubreil, Philippe Darondeau, and Hervé Marchand. Opacity enforcing control synthesis. In B. Lennartson, M. Fabian, K. Akesson, A. Giua, and R. Kumar, editors, *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08)*, pages 28–35, Göteborg, Sweden, May 2008. IEEE.

[9] Jérémy Dubreil, Philippe Darondeau, and Hervé Marchand. Opacity enforcing control synthesis. Technical Report 1921, IRISA, February 2009.

[10] Jérémy Dubreil, Thierry Jéron, and Hervé Marchand. Monitoring information flow by diagnosis techniques. Technical Report 1901, IRISA, August 2008.

[11] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part I: Information flow). In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396, Heidelberg, 2001. Springer-Verlag.

[12] N. Hadj-Alouane, S. Lafrance, F. Lin, J. Mullins, and M. Yeddes. On the verification of intransitive noninterference in mulitlevel security. *IEEE Transaction On Systems, Man, And Cybernetics—Part B: Cybernetics*, 35(5):948–957, Oct 2005.

[13] Richard M. Karp. A characterization of the minimum mean cycle in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[14] G. Le Guernic. Information flow testing - the third path towards confidentiality guarantee. In *Advances in Computer Science, ASIAN 2007. Computer and Network Security, LNCS No 4846*, pages 33–47, 2007.

[15] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2):2–16, 2005.

[16] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.

[17] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, September 1975.

[18] Laurent Mazaré. Using unification for opacity properties. In *Proceedings of the 4th IFIP WG1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 165–176, Barcelona (Spain), 2004.

[19] A. Saboori and C. Hadjicostis. Opacity-enforcing supervisory strategies for secure discrete event systems. In *47th IEEE Conference on Decision and Control*, 2008.

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

[21] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973.

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

[23] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 1–13. Springer, 1995. Invited talk.

[24] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.