Dynamic Observers for the Synthesis of Opaque Systems

Franck Cassez, Jérémy Dubreil, Hervé Marchand

Dynamic Observers for the Synthesis of Opaque Systems

Franck Cassez[1,⋆], Jérémy Dubreil[2,⋆⋆], and Hervé Marchand[2,⋆⋆]

[1] National ICT Australia & CNRS, Sydney, Australia
[2] INRIA Rennes - Bretagne Atlantique, Campus de Beaulieu, Rennes, France

Abstract. In this paper, we address the problem of synthesizing *opaque* systems. 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 and computing an optimal static observer ensuring opacity are both PSPACE-complete problems. 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. 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 synthesis problem can be solved in EXPTIME.

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, e-voting systems etc. For such services, naturally subject to malicious attacks, methods to certify their security are crucial. In this context there has been a lot of research to develop formal methods for the design of secure systems and a growing interest in the formal verification of security properties [1–3] and in their model-based testing [4–8]. 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* defined in [9] 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 of events, 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 that the attacker knows the model G. In this context, the attacker should not be able to infer that a run of G satisfies φ. The secret φ is opaque for G with respect to a given partial observation if for every run of G that satisfies φ, there exists a run, observationally equivalent from the attacker's point of view, that does not satisfy φ. In such a 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 a set of secret states. Finally, note that the definition of opacity is general enough to define other notions of information flow like trace-based non-interference and anonymity (see [9]). Note also that *secrecy* [10] can be handled as a particular case of opacity (see Section 3) and thus our framework applies to secrecy as well.

⋆ Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.
⋆⋆ Authors partially supported by the Politess RNRT project.

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 $C(G)$. In [11], 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 [12, 13], 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 [14]. 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 C which disables some actions, we consider *dynamic* observers that will dynamically change the set of observable events in order to ensure opacity. Compared to the previous approaches related to the supervisory control theory, this approach is not intrusive in the sense that it does not restrict G but only hides some events at different points in the course of the execution of the system. Indeed, one can think of a dynamic observer as a *filter* (See Figure 1) which is added on top of G.

Fig. 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. Finally we consider an optimization problem which is to compute a least expensive dynamic observer.

The notion of *dynamic observers* was already introduced in [15] 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 with static observers and show that deciding opacity for finite automata is PSPACE-complete. We also consider the optimization problem of computing a largest set (cardinality-wise) of observable events to ensure opacity and we show that this problem is PSPACE-complete as well. Section 4 is the core of the paper and considers dynamic observers for ensuring opacity. We prove that the set of all observers that ensure opacity can be computed in

[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.

EXPTIME. In Section 5 we briefly discuss how to compute optimal dynamic observers. Omitted proofs and details are given in Appendix or available in the extended version of this paper [16].

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 $v \in \Sigma^*$, we denote $u.v$ the concatenation of u and v 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 set of 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 $\lambda \in \Sigma, s \in \Sigma^*$, $P_{\Sigma_1}(s.\lambda) = P_{\Sigma_1}(s).\lambda$ if $\lambda \in \Sigma_1$ and $P_{\Sigma_1}(s)$ otherwise. Let $K \subseteq \Sigma^*$ be a language. The definition of projection for words extends to languages: $P_{\Sigma_1}(K) = \{P_{\Sigma_1}(s) \mid s \in K\}$. Conversely, let $K \subseteq \Sigma_1^*$. The *inverse projection* of K is $P^{-1}_{\Sigma,\Sigma_1}(K) = \{s \in \Sigma^* \mid P_{\Sigma_1}(s) \in K\}$. We omit the subscript Σ_1 in the sequel when it is clear from the context.

We assume that the system is given by an *automaton* G which 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). We write $q \xrightarrow{\lambda}$ whenever $\delta(q, \lambda) \neq \emptyset$. An automaton is *complete* if for each $\lambda \in \Sigma$ and each $q \in Q$, $q \xrightarrow{\lambda}$. G is deterministic if for all $q \in Q, \lambda \in \Sigma$, $|\delta(q, \lambda)| \leq 1$.

A *run* ρ from state q_0 in G is a finite sequence of transitions $q_0 \xrightarrow{\lambda_1} q_1 \xrightarrow{\lambda_2} q_2 \cdots q_{i-1} \xrightarrow{\lambda_i} q_i \cdots q_{n-1} \xrightarrow{\lambda_n} q_n$ s.t. $\lambda_{i+1} \in \Sigma$ and $q_{i+1} \in \delta(q_i, \lambda_{i+1})$ for $0 \leq i \leq n-1$. 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 shall use the *Post* operator defined by: 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.

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. Given $G_1 = (Q_1, q_0^1, \Sigma_1, \delta_1, F_1)$ and $G_2 = (Q_2, q_0^2, \Sigma_2, \delta_2, F_2)$ we denote $G_1 \times G_2$ the product of G_1 and G_2.

3 Opacity with Static Projections

In the sequel, we let $G = (Q, q_0, \Sigma, \delta, F)$ be a non-deterministic automaton over Σ and $\Sigma_o \subseteq \Sigma$. Enforcing opacity aims at preventing an attacker $\mathcal{U}$, from deducing confidential information on the execution of a system from the observation of the events in Σ_o.

Given a run of G with trace u, the observation of the attacker $\mathcal{U}$ is given by the static natural projection $P_{\Sigma_o}(u)$ following the architecture of Figure 1 with $D(u) = P_{\Sigma_o}(u)$. In this paper, we shall consider that the confidential information is directly encoded in the system by means of a set of states F[2]. If the current trace of a run is $u \in L(G)$, the attacker should not be able to deduce, from the knowledge of $P_{\Sigma_o}(u)$ and the structure of G, that the current state of the system is in F. As stressed earlier, the attacker $\mathcal{U}$ has full knowledge of the structure of G (he can perform computations using G like subset constructions) but only has a partial observation upon its behaviors, namely the observed traces in Σ_o^*. 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: $[\![\varepsilon]\!]_{\Sigma_o} = \{\varepsilon\}$ and for $\mu \in \Sigma_o^*$ and $\lambda \in \Sigma_o$, $[\![\mu.\lambda]\!]_{\Sigma_o} = P_\Sigma^{-1}(\mu).\lambda \cap L(G)$. In other words, $u \in [\![\mu.\lambda]\!]_{\Sigma_o}$ iff (1) the projection of u is $\mu.\lambda$ and (2) the sequence u ends with an observable "λ" event and (3) $u \in L(G)$.

Remark 1. We suppose that $\mathcal{U}$ is reacting faster than the system. Therefore, when an observable event occurs, $\mathcal{U}$ can compute the possible set of states of G before G moves even if G can do an unobservable action. ◇

Next we introduce the notion of opacity defined in [9]. Intuitively, a set of states F 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 state of G is in the set F.

Definition 1 (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 Definition 1 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 each $F \in \mathcal{F}$, F is opaque w.r.t. (G, Σ_o). This can be used to express other kinds of confidentiality properties. For example, [10] 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.

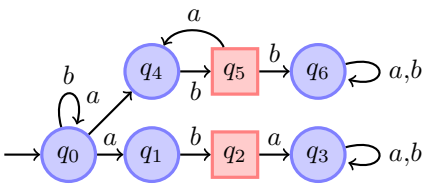


Fig. 2. State based opacity illustration

Example 1. Consider the automaton G of Figure 2, with $\Sigma_o = \Sigma = \{a, b\}$. The secret is given by the states $F = \{q_2, q_5\}$. The secret F is certainly not opaque with respect to (G, Σ), as by observing a trace $b^*.a.b$, the attacker $\mathcal{U}$ knows that the system is in a secret state. Note 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. □

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) ?

[2] Equivalently, the secret can be given by a regular language over Σ^*, see [16].

3.1 Checking State Based Opacity

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: $Det_o(G) = (\mathcal{X}, X_0, \Sigma_o, \Delta, F_o)$ denotes the deterministic automaton given by:

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

Checking whether F is opaque w.r.t. (G, Σ_o) amounts to checking whether a state in F_o is reachable. To check opacity for a family $\{F_1, F_2, \cdots, F_k\}$, we define F_o 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 opacity on non-deterministic FA can be checked in exponential time. Actually, checking state based opacity for (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 $L_F(G) = \Sigma^*$. If not, then G is not universal. Checking language universality for non-deterministic FA is PSPACE-complete [17] and Problem 1 is equivalent to universality.

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

Proof. We assume that G is complete i.e., $L(G) = \Sigma^*$. Note that $[\![u]\!]_\Sigma = u$. Now, G is not universal iff there exists $u \in \Sigma^*$ such that $\mathit{Post}(\{q_0\}, [\![u]\!]_\Sigma) \subseteq Q \setminus F$. With the definition of state based opacity, taking $\Sigma_o = \Sigma$, we have:

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

PSPACE-easiness was already known and follows from a result in [18]: the model-checking problem for a temporal logics which can specify security properties is proved to be PSPACE-complete.

3.2 Maximum Cardinality for Static Projections

If a secret is opaque w.r.t. a set of observable events Σ_o, it is worthwhile noticing that it will still be opaque w.r.t. any subset of Σ_o. 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. Given the set of events Σ of G, we can check whether the secret is opaque w.r.t. $\Sigma_o \subseteq \Sigma$. In that case, we may increase the number of visible letters and check again if the secret remains opaque. This suggests the following optimization problem:

Problem 2 (Maximum Cardinality of Observable Events).
INPUT: A non-deterministic FA $G = (Q, q_0, \Sigma, \delta, F)$ and $n \in \mathbb{N}$ 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. We generalize this approach by considering the notion of *dynamic projections* encoded by means of *dynamic observers* as introduced in [15]. Dynamic projection allows us 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 2. Consider again the automaton G of Example 1, Figure 2, where $F = \{q_2, q_5\}$. 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. We can be more efficient by using a dynamic *projection that will render unobservable an event only when necessary. Indeed, after observing b^*, the attacker 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

We now define the notion of dynamic projection and its associated dynamic observer.

Dynamic Projections and Observers. A dynamic projection is a function that will decide to let an event be observable or to hide it, thus playing the role of a filter between the system and the attacker to prevent information flow (see Figure 1).

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

$$D(u.\lambda) = D(u).\lambda \text{ if } \lambda \in T(D(u)) \text{ and } D(u.\lambda) = D(u) \text{ otherwise.} \quad (1)$$

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

Example 3. A dynamic projection $D : \Sigma^* \to \Sigma^*$ corresponding to the one of Example 2 can be induced by the dynamic observability choice T defined by $T(u) = \{a\}$ for all $u \in b^*.a$ and $T(u) = \{a, b\}$ for all the other sequences $u \in \Sigma^*$. $\diamond$

Given a FA G 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.

Opacity with Dynamic Projection. We generalize Definition 1 by taking into account the new observation interface given by D.

Definition 3. *Given a FA $G = (Q, q_0, \Sigma, \delta, F)$, F is opaque with respect to (G, D) if*

$$\forall \mu \in Tr_D(G),\ \mathit{Post}(\{q_0\}, [\![\mu]\!]_D) \not\subseteq F. \tag{2}$$

Again, this definition extends to family of sets. We say that D is a *valid* dynamic projection if (2) 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 $\sim_G$.

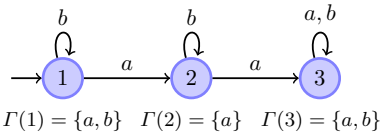


Fig. 3. Example of a Dynamic Observer

Remark 2. Let $\Sigma_o \subseteq \Sigma$, 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 $D(u) = P_{\Sigma_o}(u)$ and we retrieve Definition 1. The property of secrecy can be extended as well using dynamic projection. $\diamond$

In the sequel, we shall 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 [15] that will encode a dynamic projection in terms of automata.

Definition 4 (Dynamic observer). *A* dynamic 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. An example of dynamic observer is given in Figure 3. We now relate the notion of dynamic observer to the one of dynamic projection.

Proposition 1. *Let $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$ be an observer and define $D_\mathcal{O}$ by: $D_\mathcal{O}(\varepsilon) = \varepsilon$, 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. In the sequel, we write $[\![\mu]\!]_\mathcal{O}$ for $[\![\mu]\!]_{D_\mathcal{O}}$.*

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 (1) 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. □

Proposition 2. *Given a dynamic projection D induced by T, let $\mathcal{O}_D = (\Sigma^*, \varepsilon, \Sigma, \delta_D, T)$ where $\delta_D(w, \lambda) = D(w.\lambda)$. Then $\mathcal{O}_D$ is a dynamic observer.*

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

Note that there might exist several equivalent observers that encode the same dynamic projection. For example, the observer depicted in Figure 3 is one observer that encodes the dynamic projection described in Example 3. But, one can consider other observers obtained by unfolding an arbitrary number of times 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.

4.2 Checking Opacity with Dynamic Observers

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 $\mathcal{O}$. 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 $\lambda \in \Sigma \cup \{\tau\}$, 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 3. *F is opaque w.r.t. $(G, \mathcal{O})$ iff $F \times X$ is opaque w.r.t. $(G \otimes \mathcal{O}, \Sigma)$.*

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:

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

If $\mu = \varepsilon$, the result is immediate. Assume that $\mu' = \mu.\lambda$. Let $q \in \mathit{Post}_G(\{q_0\}, [\![\mu']\!]_{\mathcal{O}})$. 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 $(q', \delta_o(x_0, u)) \in \mathit{Post}_{G \otimes \mathcal{O}}(\{(q_0, x_0)\}, [\![\mu]\!]_{\Sigma})$ 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

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

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 $P_{\Sigma}(w.\tau^k.\lambda) = \mu.\lambda = \mu'$. This implies $(q, \delta_o(x_0, u.v.\lambda)) \in \mathit{Post}_{G \otimes \mathcal{O}}(\{(q_0, x_0)\}, [\![\mu']\!]_{\Sigma})$. For the converse if we have a sequence of τ transitions in $G \otimes \mathcal{O}$, they must originate from actions in G which are not observable. □

The previous result is general, and if $\mathcal{O}$ is a FA 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 opacity can be checked in PSPACE, PSPACE-easiness follows. Now, checking state based opacity with respect to (G, Σ) can be done using a simple observer with one state which always let Σ observable and PSPACE-hardness follows. □

As Proposition 3 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).

4.3 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.

Problem 4 (Dynamic Observer Synthesis Problem).
INPUT: A non-deterministic FA $G = (Q, q_0, \Sigma, \delta, F)$.
PROBLEM: Compute the set of valid dynamic observers $\mathcal{OBS}(G)$[3].

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. 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 (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* [19]).
We now formally define 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 the deterministic game automaton derived from G and 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, then Player 1 actions are in the alphabet $M_1 = 2^\Sigma$ and Player 2 actions are 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 (observable events): if $s \in S_1$, $t \subseteq \Sigma$, then $\delta_H(s, t) = (s, t)$;
  - Player 2 moves (observed events): if $(s, t) \in S_2$, $\lambda \in t$ and $s' = \mathit{Post}(s, (\Sigma \setminus t)^*.\lambda) \neq \emptyset$, then $\delta_H((s, t), \lambda) = s'$.

Remark 3. 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., $\forall s \in S_1$, and $t \neq \emptyset$, $\delta_H(s, t) = (s, t)$. ◇

We define the set of Bad states to be the set of Player 1 states s s.t. $s \subseteq F$. For a family of sets $F_1, F_2, \cdots, F_k$, Bad is the set of states $2^{F_1} \cup 2^{F_2} \cup \cdots \cup 2^{F_k}$. Let $\mathit{Runs}_i(H), i = 1, 2$ be the set of runs of H that end in a Player i state. A *strategy* for

[3] Our aim is actually to be able to generate at least one observer for each representative of $\mathcal{D}_{\sim_G}$, thus capturing all the interesting dynamical projections.

Player i is a mapping f_i : $\mathit{Runs}_i(H) \rightarrow M_i$ 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 $\mathit{Outcome}(f_1, f_2, H)$ 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 $\mathit{Outcome}(f_1, f_2, H)$ 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, $\mathit{Outcome}(f_1, f_2, H)$ 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 $P_{M_2}(\varrho) = P_\Sigma(\mathit{tr}(\varrho))$ for a run ϱ of H. The proof of the following Proposition 4 is given in Appendix.

Definition 5. *Given a dynamic projection D, we define the strategy f_D such that for every $\varrho \in \mathit{Runs}_1(H)$, $f_D(\varrho) = T_D(P_{M_2}(\varrho))$.*

Proposition 4. *Let D be a dynamic projection. D is valid if and only if f_D is a winning strategy for Player 1 in H.*

Given a strategy f for Player 1 in H, for all $\mu \in \Sigma^*$, there exists at most one run $\varrho_\mu \in \mathit{Outcome}_1(f, H)$ such that $P_{M_2}(\mathit{tr}(\varrho_\mu)) = \mu$.

Definition 6. *Let f be a strategy for Player 1 in H. We define the dynamic projection D_f induced by the dynamic observability choice $T_f : \Sigma^* \rightarrow 2^\Sigma$ given by: $T_f(\mu) = f(\varrho_\mu)$ if ϱ_μ is in $\mathit{Outcome}(f, H)$ and $T_f(\mu) = \Sigma$ otherwise.*

Notice that when ϱ_μ is not in $\mathit{Outcome}(f, H)$, it does not really matter how we define T_f because there is no word $w \in L(G)$ s.t. $\mu = D_f(w)$.

Proposition 5. *If f is a winning strategy for Player 1 in H, then D_f is a valid dynamic projection.*

Proof. Applying the construction of Definition 5 yields $f_{D_f} = f$. Since f is a winning strategy, by Proposition 4, 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 $\mathcal{D}_{\sim_G}$.

4.4 Most Permissive Dynamic Observer

We now define the notion of *most permissive* valid dynamic observers. For an observer $\mathcal{O} = (X, x_o, \Sigma, \delta_o, \Gamma)$ and $w \in \Sigma^*$, recall that $\Gamma(\delta_o(x_o, w))$ is the set of events that $\mathcal{O}$ chooses to render observable after observing w. Assume that $w = \lambda_1\lambda_2\cdots\lambda_k$. Let $\overline{w} = \Gamma(x_o).\lambda_1.\Gamma(\delta_o(x_o, w[1])).\lambda_2.\Gamma(\delta_o(x_o, w[2]))\cdots\lambda_k.\Gamma(\delta_o(x_o, w[k]))$ i.e., $\overline{w}$ contains the history of what $\mathcal{O}$ has chosen to observe at each step and the next observable event that occurred after each choice.

Definition 7. *Let $\mathcal{O}^* : (2^\Sigma.\Sigma)^* \rightarrow 2^{2^\Sigma}$. The mapping $\mathcal{O}^*$ is the* most permissive valid dynamic observer[4] *ensuring the opacity of F if the following holds:*

[4] 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.

$\mathcal{O} = (X, x_o, \Sigma, \delta_o, \Gamma)$ *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 valid 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.

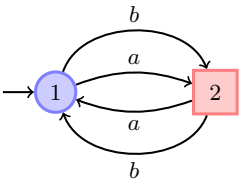
Theorem 4. *The most permissive valid observer $\mathcal{O}^*$ can be computed in EXPTIME.*

Proof. The detailed proof is given in Appendix. For a sketch, the most permissive valid dynamic observer is obtained using the most permissive winning strategy in the game H. It is well-known result [20] that for a finite game, if there is a winning strategy, there is a memoryless most permissive one. Moreover whether there is a winning strategy can be decided in linear time in the size of the game. As the size of H is exponential in the size of G and Σ the result follows. □

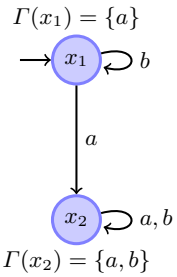
We let $\mathcal{F}_H$ be the automaton representing the most permissive observer. Theorem 4 states that $\mathcal{F}_H$ can be used to generate any valid observer. In particular, given a finite-memory winning strategy, the corresponding valid observer is finite and thus its associated dynamic projection is regular. An immediate corollary of Theorem 4 is the following:

Corollary 1. *Problem 4 is in EXPTIME.*

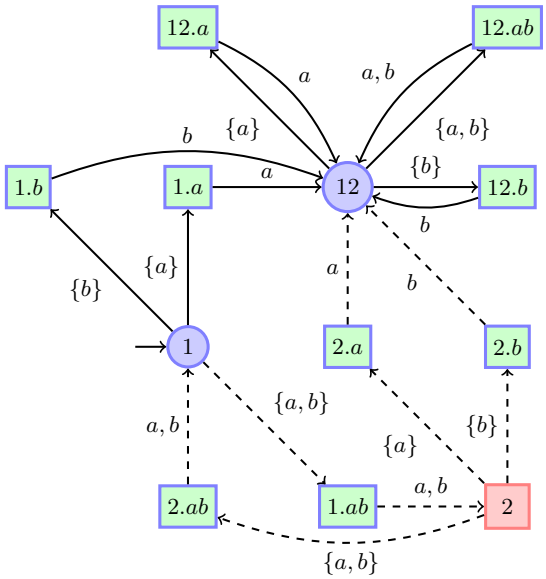
Example 4. To illustrate this section, we consider the following small example. The system is depicted by the automaton in Figure 4(a). The set of secret states is reduced to the state (2). Figure 4(b) represents the associated game automaton. The states of Player 1



(a): The Automaton G



(c): A Finite Observer



(b): The Game H

Fig. 4. Most Permissive Dynamic Observer

are represented by circles whereas the ones of Player 2 are represented by squares. The only bad states is the state (2). The most permissive valid dynamic observer is obtained when Player 1 does not allow transition $\{a, b\}$ to be triggered in state (1) (otherwise, Player 2 could choose 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 the most permissive observer. Finally, Figure 4(c) represents a valid observer $\mathcal{O}$ generated from the most permissive observer with the memoryless strategy $f(1) = \{a\}$ and $f(12) = \{a, b\}$. $\diamond$

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: they hide less events on *average*. We here define a notion of cost 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 [15] and use the same tools: Karp's mean-weight algorithm [21] and a result of Zwick and Paterson [22]. 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 that the observer is a finite automaton $\mathcal{O} = (X, x_0, \Sigma, \delta_o, \Gamma)$. 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 $\mathcal{O}$ is in state x, the current cost per time unit is $\mathit{Cost}(\Gamma(x))$. Let $Runs^n(G)$ be the set of runs of length n in $Runs(G)$. Given a run $\rho = q_0 \xrightarrow{\lambda_1} q_1 \cdots q_{n-1} \xrightarrow{\lambda_n} q_n \in \mathit{Runs}^n(G)$, 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 $i+1$, $\mathit{Cost}(\Gamma(x))$ will be counted twice: 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. Define the cost of the set of runs of length n that belongs to $Runs^n(G)$ by: $\mathit{Cost}(n, G, \mathcal{O}) = \max\{\mathit{Cost}(\rho, G, \mathcal{O}) \mid \rho \in Runs^n(G)\}$. The *cost of an observer* with respect to a system G is

$$\mathit{Cost}(G, \mathcal{O}) = \limsup_{n \to \infty} \mathit{Cost}(n, G, \mathcal{O}) \tag{3}$$

(notice that the limit may not exist whereas the limit sup is always defined.) To compute the cost of a given observer, we can use a similar algorithm as the one given in [15], and using Karps's maximum mean-weight cycle algorithm [21]:

Theorem 5. *Computing Cost*$(G, \mathcal{O})$ *is in PTIME.*

Proof. We can prove that the cost of an observer is equal to the maximum mean-weight cycle in $G \otimes \mathcal{O}$. The size of $G \otimes \mathcal{O}$ is polynomial in the size of G and $\mathcal{O}$. Computing the maximum mean-weight cycle can be done in linear time w.r.t. the size of $G \otimes \mathcal{O}$. □

Finally we can solve the following optimization problem:

Problem 5 (Bounded Cost Observer).
INPUTS: an automaton $G = (Q, q_0, \Sigma, \delta, F)$ and an integer $k \in \mathbb{N}$.
PROBLEMS:
(A) Is there any $\mathcal{O} \in \mathcal{OBS}(G)$ s.t. F is opaque w.r.t. $(G, \mathcal{O})$ 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 from Zwick and Paterson [22], which is an extension of Karp's algorithm for finite state games.

Theorem 6. *Problem 5 can be solved in EXPTIME.*

The solution to this problem is the same as the one given in [15], and the proof for the opacity problem is detailed in [16]. The key result is Theorem 4, which enables us to represent all the winning strategies in H as a finite automaton. Synchronizing G and the most permissive valid dynamic observer $\mathcal{F}_H$ produces a *weighted game*, the optimal value of which can be computed in PTIME (in the size of the product) using the algorithm in [22]. The optimal strategies can be computed in PTIME as well. As $G \times \mathcal{F}_H$ has size exponential in G and Σ, the result follows.

6 Conclusion

In this paper, we have investigated the synthesis of opaque systems. 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. We have also defined a model of dynamic observers determining whether an event is observable after a given observed trace. We proved that the synthesis of dynamic observers can be solved in EXPTIME, and EXPTIME-hardness is left open.

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 as it would be easy to implement in practice. Finally, the notion of semantics of an observed trace used throughout this article is based on the assumption that the attacker can react, i.e., acquire knowledge, faster than the system's evolution. It would be interesting to adapt this work to other types of semantics.

References

1. Lowe, G.: Towards a completeness result for model checking of security protocols. Journal of Computer Security **7**(2-3) (1999) 89–146.
2. Blanchet, B., Abadi, M., Fournet, C.: Automated Verification of Selected Equivalences for Security Protocols. In: 20th IEEE Symposium on Logic in Computer Science (LICS 2005), Chicago, IL, IEEE Computer Society (June 2005) 331–340.
3. Hadj-Alouane, N., Lafrance, S., Lin, F., Mullins, J., Yeddes, M.: On the verification of intransitive noninterference in mulitlevel security. IEEE Transaction On Systems, Man, And Cybernetics—Part B: Cybernetics **35**(5) (Oct 2005) 948–957.
4. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1) (2000) 30–50.
5. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Sec. **4**(1-2) (2005) 2–16.
6. Darmaillacq, V., Fernandez, J.C., Groz, R., Mounier, L., Richier, J.L.: Test generation for network security rules. In: TestCom 2006. Volume 3964 of LNCS. (2006).
7. Le Guernic, G.: Information flow testing - the third path towards confidentiality guarantee. In: Advances in Computer Science, ASIAN 2007. Computer and Network Security, Volume 4846 of LNCS. (2007) 33–47.
8. Dubreil, J., Jéron, T., Marchand, H.: Monitoring information flow by diagnosis techniques. Technical Report 1901, IRISA (August 2008).
9. Bryans, J., Koutny, M., Mazaré, L., Ryan, P.: Opacity generalised to transition systems. International Journal of Information Security **7**(6) (May 2008) 421–435.
10. Alur, R., Černý, P., Zdancewic, S.: Preserving secrecy under refinement. In: ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming, Volume 4051 of LNCS. Springer (2006) 107–118.
11. Badouel, E., Bednarczyk, M., Borzyszkowski, A., Caillaud, B., Darondeau, P.: Concurrent secrets. Discrete Event Dynamic Systems **17** (December 2007) 425–446
12. Dubreil, J., Darondeau, P., Marchand, H.: Opacity enforcing control synthesis. In: Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08), Göteborg, Sweden (May 2008) 28–35.
13. Dubreil, J., Darondeau, P., Marchand, H.: Opacity enforcing control synthesis. Technical Report 1921, IRISA (February 2009)
14. Takai, S., Oka, Y.: A formula for the supremal controllable and opaque sublanguage arising in supervisory control. SICE Journal of Control, Measurement, and System Integration **1**(4) (March 2008) 307–312.
15. Cassez, F., Tripakis, S.: Fault diagnosis with static or dynamic diagnosers. Fundamenta Informatica **88**(4) (November 2008) 497–540.
16. Cassez, F., Dubreil, J., Marchand, H.: Dynamic Observers for the Synthesis of Opaque Systems. Technical Report 1930, IRISA (May 2009).
17. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: STOC, ACM (1973) 1–9.
18. Alur, R., Cerný, P., Chaudhuri, S.: Model checking on trees with path equivalences. In Grumberg, O., Huth, M., eds.: TACAS. Volume 4424 of LNCS, Springer (2007) 664–678.
19. Martin, D.A.: Borel determinacy. Annals of Mathematics **102**(2) (1975) 363–371.
20. Thomas, W.: On the synthesis of strategies in infinite games. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95). Volume 900 of LNCS. Springer (1995) 1–13 Invited talk.
21. Karp, R.: A characterization of the minimum mean cycle in a digraph. Discrete Mathematics **23** (1978) 309–311.
22. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theoretical Computer Science **158**(1–2) (1996) 343–359.