Diagnosis and Opacity Problems for Infinite State Systems Modeled by Recursive Tile Systems

Sébastien Chédor, Christophe Morvan, Sophie Pinchinat, Hervé Marchand

Discrete Event Dyn Syst

Diagnosis and Opacity Problems for Infinite State Systems Modeled by Recursive Tile Systems

Sébastien Chédor · Christophe Morvan ·
Sophie Pinchinat · Hervé Marchand

Abstract The analysis of discrete event systems under partial observation is an important topic, with major applications such as the detection of information flow and the diagnosis of faulty behaviors. These questions have, mostly, not been addressed for classical models of recursive systems, such as pushdown systems and recursive state machines. In this paper, we consider *recursive tile systems*, which are recursive infinite systems generated by a finite collection of finite *tiles*, a simplified variant of deterministic graph grammars (slightly more general than pushdown systems). Since these systems are infinite-state in general powerset constructions for monitoring do not always apply. We exhibit computable conditions on recursive tile systems and present non-trivial constructions that yield effective computation of the monitors. We apply these results to the classic problems of state-based opacity and diagnosability (off-line verification of opacity and diagnosability, and also run-time monitoring of these properties). For a decidable subclass of recursive tile systems, we also establish the decidability of the problems of state-based opacity and diagnosability.

Keywords Diagnosability · Opacity · Discrete event systems · Recursive systems

S. Chédor
Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France
E-mail: sebastien.chedor@inria.fr

C. Morvan
Université Paris-Est, UPEMLV, F-77454, Marne-La-Vallée, France
E-mail: christophe.morvan@univ-paris-est.fr

S. Pinchinat
Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France
E-mail: sophie.pinchinat@irisa.fr

H. Marchand
Inria Rennes - Bretagne Atlantique, Campus de Beaulieu, 35042 Rennes, France
E-mail: herve.marchand@inria.fr

1 Introduction

The automated analysis and generation of programs is a very active area. It aims at supporting the development of devices that monitor either computing systems or even physical ones while they evolve. Additionally, the communication between the device and the system under consideration may be limited, hence the actual executions of the system may be only partially observed. Even under partial observation, some hidden information may be computationally reconstructible from some specification of the system. Typical fields that address such issues are those of *information flow detection* [3,6,8], *diagnosis* [24,29,18,26,5], and combination of both [15].

On the one hand, information flow detection relates to computer security, based on the notion of *opacity* [6,3,15]. The *opacity problem* consists in determining whether an observer, who knows the system's behavior but who imperfectly observes it, is able to reconstruct critical information (*e.g.*, a password stored in a file, the value of some hidden variables, etc.). More precisely, the setting of state-based opacity [9] considers a discrete event system having a subset of unobservable events, and a subset of secret states. Then the problem amounts to determining whether it is possible to infer, from observation of the system, if it has reached a secret state. This can be performed a priori or at run-time for a given observed sequence of the system. There are other variants of this problem, like language-based, *initial*, *k-step*, infinite, initial-and-final opacity [22,23,9,28]. On the other hand, the field of diagnosis concerns systems that may have faulty behaviors. A diagnoser is a monitor which reveals the faults at run-time. In an ideal situation, we expect the device to be sound (when a fault is declared, it is indeed the case) and complete (every faulty behavior is eventually revealed within a finite delay). Soundness is often guaranteed by equipping the device with standard state estimate techniques, whereas completeness, also known as the *diagnosability property* [24,29], is tightly coupled with the observation capabilities and has to be established on its own.

Regarding the state-based opacity problem, [9] has shown the PSPACE-completeness of the opacity verification of a system. Off-line computation of a monitor in charge of the detection at run-time of an information has the same complexity. Moreover, as initial and language-based opacity is polynomially reducible to state-based-opacity, it has the same complexity. We thus focus in this paper to the state-based opacity problem. For the verification and monitoring of the k-step opacity or infinite opacity, the approaches [22,23] differ and are not considered in this paper. For infinite systems however, [8,6] show its undecidability for timed systems and general Petri nets. Regarding diagnosis, the diagnosability property is now well understood [24, 7,18,19], and the most efficient checking procedure seeks for particular cycles in a self-product construction, hence it yields a quadratic-time algorithm. On the contrary, for infinite systems, this approach may not be effective in general. For example, [26] investigates timed systems and proves that diagnosability property is equivalent to non-zenoness. [5] expands this work, and exhibits complexity bounds for the diagnosability problem: a 2EXPTIME complexity for the class of deterministic timed automata, and a PSPACE complexity for event-recording timed automata. Note that instead of building a self-product, they use game-based techniques. They also provide a construction for the diagnoser. [27] considers Petri nets and proposes an effective construction of the diagnoser and shows the undecidability of the diagnosability property. [4] generalizes the setting by considering graph transformation systems which allow to capture systems with mobility and variable topologies. The main contribution is to compute the set of executions of the system that characterizes a given observation. With the same objective, [16, 17] consider a distributed observation of a distributed system modeled by a High Level Message Sequence Chart. Recently, [21] considered pushdown systems and shows that the diagnosability property is decidable for a subclass of visibly pushdown systems. Following a similar trend, [20] considers both opacity and diagnosability for pushdown systems (PDSs). More precisely the authors consider opacity for PDSs whose closure is a visibly pushdown system. For diagnosability, they consider approximations by finite-state systems. The present paper is more general since the class of *weighted RTSs*, described later on, contains all visibly pushdown systems. However, we do not consider, here, approximations of such systems by finite-state ones.

As exemplified in the literature, opacity and diagnosis problems have been solved by using common techniques: (1) computation of an *ε-closure* (a projection), (2) *determinization* preserving state properties, (3) computation of a *self-product* or its simulation as a game, (4) detection of *infinite executions*, and (5) *reachability analysis*.

Whereas these techniques are well established and effective for finite-state systems, their transfer to infinite-state systems requires further investigations. In this paper, we study the opacity and diagnosability problems in the setting of *recursive tile systems* (RTSs). RTSs represent infinite-state systems, in general, and are equivalent to *deterministic graph grammars* of [14], these tile systems have been introduced in [13]. They form a natural extension of pushdown systems (see *e.g.*, [10]) and of the recursive state machines [1]. Furthermore, they cannot be compared with Petri nets in general, nor with timed automata whose state-space may be uncountable.

Example 1 Let us motivate the use of the RTSs by the following small example modeling a production system: it handles two resources (A and B) with an urgency feature depicted by the automaton on Figure 1. Resource B has higher priority than A.

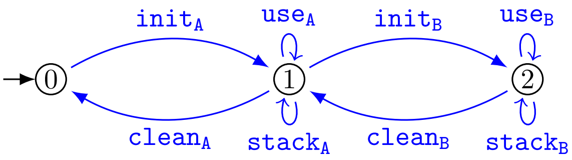

Fig. 1 A two-resource system.

In the initial state 0, the system contains no resource and can not handle any resource. In State 1 (resp. state 2), the system is configured to process resources A (resp. B). The transitions $\mathtt{stack_A}$ and $\mathtt{stack_B}$ model the reception of resources A and B, respectively. Similarly, the transitions $\mathtt{use_A}$ and $\mathtt{use_B}$ model the handling of the resources (implicitly implying the existence of such resource in the system). This system is rather simple, however two main features are not (and cannot) be depicted by this finite-state structure: whenever transition $\texttt{init}_\texttt{B}$ is performed, resources $\texttt{A}$ that was already in the system remains so, and whenever transition $\texttt{clean}_\texttt{B}$ occurs, resources $\texttt{B}$ should not be available anymore. It is worthwhile noticing that such properties can be easily modelled by a machine with two counters. Nevertheless, since two counters machines are Turing complete, this would lead to undecidability of very elementary problems such as reachability. However, as we shall see, RTSs may be exploited whenever counters are hierarchically organised, while still preserving interesting decidability results. ◇

As shown in this paper, both opacity and diagnosability are undecidable problems for the class of RTSs. We thus exhibit the class of *weighted RTSs* [11] which captures visibly pushdown systems, as the right notion to address effectiveness of the needed constructions for analysis. We identify decidable properties of RTSs ensuring decidability of the opacity and the diagnosability problems. The present work strictly extends the results of [21], and relies on a new approach with non-trivial constructions and transformation on tiling systems. The monitor needed both for analyzing information flows and fault occurrences is represented by a *typing machine*, a DES which preserves the set of observations of the system and the set of states reached by a given observation. It relies on an (observational) *closure* of the original system, followed by a *determinization* procedure. Straightforward application of these operations to RTSs may produce objects which are not RTSs. However, we propose an effective way to compute the closure as an RTS, but which may be not weighted in general (even if the initial system is). Therefore, we exhibit the class of $\mathcal{C}$*wRTS* , for "*closure-weighted RTS*" composed of the RTSs whose closure is weighted; moreover, checking whether an RTS is a $\mathcal{C}$wRTS is decidable. We further show that the typing machine of every $\mathcal{C}$wRTS is an RTS, yielding an effctive way to perform the reachability analysis needed to verify the opacity property. Also, the self-product techniques useful to check the diagnosability property can be conducted over $\mathcal{C}$wRTS. Furthermore, we assess complexity classes for each problem and compare them against those of finite state systems. Our findings are summarized in Table 1.

Problem	Finite-state	RTSs	$\mathcal{C}$wRTS
Opacity	PSPACE-complete [9]	Undecidable	2-EXPTIME
Diagnosability	PTIME [25]	Undecidable [21]	EXPTIME

Table 1 Complexities

This paper is organized in three main parts: in Section 2, we define DESs under partial observation we introduce the diagnosability and the opacity problems. We present a mathematical setting to solve these problems, abstracting from effectiveness aspects. In Section 3, we introduce the RTSs, and establish all technical results needed to prove the main results. In Section 4, we solve the opacity and diagnosability problems for infinite systems described by RTSs in the class $\mathcal{C}$wRTS. We also investigate the diagnosis of opacity. The complexity class of each decidable result is established as well.

2 Discrete event systems under partial observation

2.1 Discrete event systems

The model of DES is commonly used to represent the behavior of systems at a very high level of abstraction. It is composed of a (possibly infinite) set of states (or configurations) and transitions between those states, labeled by actions representing the atomic evolution of the system.

Definition 1 *A* discrete event system *(DES) is a tuple* $\mathcal{A} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ *where* Σ *is the alphabet of events,* Q *is the (infinite) set of* states*,* $q_0 \in Q$ *is the* initial state*,* $\Delta \subseteq Q \times \Sigma \times Q$ *is the set of transitions,* Λ *is a set of* propositions *and* $Ty : Q \to 2^{\Lambda}$ *is a* typing function *which allocates to each state* $q \in Q$ *the set of propositions* $Ty(q)$ *that hold in* q*.* $Ty(q)$ *is called the* type *of* q*.*

For the remainder of Section 2, we fix a typed DES $\mathcal{A} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$. A transition (p, a, q) in Δ is written $p \xrightarrow{a} q$. State p is the *source* and state q the *target*. The *in-degree* (*resp. out-degree*) of a state q is the number of transitions having q as target (*resp.* source); the *degree* of a state is the sum of its in and out-degrees. A typed DES $\mathcal{A}$ is *deterministic* if Δ is a function from $Q \times \Sigma$ to Q. We extend the relation $\to$ to an arbitrary word by setting: $q \xrightarrow{\varepsilon} q$ for every state q and $q \xrightarrow{ua} p$ whenever $q \xrightarrow{u} q'$ and $q' \xrightarrow{a} p$ for $u \in \Sigma^*$, $a \in \Sigma$ and for some $q' \in Q$. We note $p \to^* q$ whenever $p \xrightarrow{u} q$ for some $u \in \Sigma^*$ and we say that q is *reachable* from state p. We denote $Reach_{\mathcal{A}}(P, \Sigma')$, the set of states that can be reached from the set of states P only triggering events of Σ'. A proposition λ naturally denotes the set of states $Q_\lambda = \{q \in Q | \lambda \in Ty(q)\}$ in $\mathcal{A}$.

A *path* $\wp$ is given by an alternating sequence of events and states, indexed by an interval $I_\wp$ of $\mathbb{Z}$, such that, for each pair $(k, k+1)$ in $I_\wp$, there is a transition $q_k \xrightarrow{a_k} q_{k+1}$. Observe that we allow infinite paths (with intervals ending with $-\infty$ or $+\infty$). For a finite path $\wp$, we define its *type* $Ty(\wp)$ as the type of its last state. Given $\Sigma' \subseteq \Sigma$, we say that a path $\wp$ is Σ'*-labeled* whenever for all $k \in I_\wp$, $a_k \in \Sigma'$.

The (possibly infinite) word formed by the sequence of labels $(a_k)_{k \in I_\wp}$ is the *label* of $\wp$. A path from the initial state q_0 of $\mathcal{A}$ is called a *run* and its label is a *trace* of $\mathcal{A}$. We define by $\mathcal{L}(\mathcal{A}) = \{t \in \Sigma^* \mid q_o \xrightarrow{t} q \text{ for some } q \in Q\}$ the set of traces of $\mathcal{A}$ and by $\mathcal{A}(t) = \{q \in Q | q_0 \xrightarrow{t} q\}$ the set of states reachable from q_0 by a trace t of $\mathcal{A}$. Given $P \subseteq Q$, $\mathcal{L}_P(\mathcal{A}) = \{t \in \mathcal{L}(\mathcal{A}) \mid \mathcal{A}(t) \subseteq P\}$ denotes the set of traces that can end in the set of states P. Given a trace $t \in \mathcal{L}(\mathcal{A})$, we write $\mathcal{L}(\mathcal{A})/t = \{u \in \Sigma^* \mid tu \in \mathcal{L}(\mathcal{A})\}$ for the set of traces that extend t in $\mathcal{A}$. The notation for types is extended from paths to traces $t \in \mathcal{L}(\mathcal{A})$, $Ty(t)$ is $\bigcup_{q \in \mathcal{A}(t)} Ty(q)$, corresponding to the set propositions that hold in at least one of the states that could be reached in $\mathcal{A}$ by the trace t.

We recall the classic notion of synchronous product between two DES.

Definition 2 *The* synchronous product *of a DES* $\mathcal{A}_i = (\Sigma, Q_i, \Lambda_i, q_0^i, \Delta_i, Ty_i)$ *(*$i = 1, 2$*) is the DES* $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma, Q_1 \times Q_2, \Lambda, (q_0^1, q_0^2), \Delta, Ty)$ *where* $((p_1, p_2), a, (q_1, q_2)) \in \Delta$ *if and only if* $(p_1, a, q_1) \in \Delta_1$ *and* $(p_2, a, q_2) \in \Delta_2$ *and* $Ty(p_1, p_2) = Ty_1(p_1) \cup Ty_2(p_2)$*.*

Given $F_1 \in Q_1$ and $F_2 \in Q_2$, we have $\mathcal{L}_{F_1 \times F_2}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}_{F_1}(\mathcal{A}_1) \cap \mathcal{L}_{F_2}(\mathcal{A}_2)$. The product between a DES $\mathcal{A}$ and itself is called a *self-product* and the resulting DES is denoted by $\mathcal{A}^2$.

Partial Observation. The key point of our approach concerns the ability for a device to deduce information from a system by observing only a subset of its events. For this purpose, the set of events Σ of a DES $\mathcal{A}$ is partitioned into two subsets Σ_o and Σ_{uo} containing respectively *observable* events and *unobservable* ones. The canonical projection π of Σ onto Σ_o is standard: $\pi(\varepsilon) = \varepsilon$, and $\forall t \in \Sigma^*, a \in \Sigma, \pi(ta) = \pi(t)a$ whenever $a \in \Sigma_o$, and $\pi(ta) = \pi(t)$ otherwise. π naturally extends to languages. The inverse projection of L is defined by $\pi^{-1}(L) = \{t \in \Sigma^* \mid \pi(t) \in L\}$. For every $\mu \in \Sigma_o^*$, we write $p \overset{\mu}{\rightsquigarrow} q$ if there exists $t \in \Sigma^*$ and $a \in \Sigma_o$ such that $\mu = \pi(t)a$ and $p \overset{ta}{\rightarrow} q$.

An *observation* of $\mathcal{A}$ is a sequence $\mu \in \Sigma_o^*$ such that $\mu = \pi(t)$ for some $t \in \mathcal{L}(\mathcal{A}) \cap \Sigma^* \Sigma_o$; in that case, t is *associated* to observation μ. We denote by $obs(\mathcal{A})$ the set of observations of $\mathcal{A}$ and by $obs_P(\mathcal{A})$ the set of observations that can end in the set of states P. Following previous notations, we define $\mathcal{A}(\mu) = \{q \in Q \mid q_0 \overset{\mu}{\rightsquigarrow} q\}$ as the set of states reached by a run with observation μ. Two runs $\wp$ and $\wp'$ are *equivalent* if their traces are associated to the same observation.

In our setting, the information is encoded by the set of propositions that holds in the states of the system. In order to be able to infer information based on the observations, we first need to define the type of an observation. To do so, we naturally extend the typing function Ty of $\mathcal{A}$ to observations in the following way.

Definition 3 *Given an observation $\mu \in obs(\mathcal{A})$, its* type $Ty(\mu)$ *is* $\bigcup_{q \in \mathcal{A}(\mu)} Ty(q)$.

Notice that the type of traces and observations interact since $Ty(\mu)$ can alternatively be defined as $\bigcup_{t \in \pi^{-1}(\mu) \cap \Sigma^* \Sigma_o} Ty(t)$. In particular, $Ty(u) \subseteq Ty(\pi(u))$. Intuitively, the type of an observation μ corresponds to the set propositions that hold in at least one of the states that could be reached in $\mathcal{A}$ by a trace t that is compatible with the observation μ.

Remark 1 *In the literature, type of states, runs and observations in DESs are usually defined directly as subsets of the set of states. Our notion of type is slightly different, but aims at a greater simplicity in the following sections where DESs have a finite presentation, but have infinitely many states*

To compute the type of observations, we may use a *typing machine* :

Definition 4 *A* typing machine *of $\mathcal{A}$, is a deterministic DES $\mathcal{T}_{\Sigma_o}(\mathcal{A}) = (\Sigma_o, Q', \Lambda, q_0', \Delta', Ty')$ such that $\forall \mu \in obs(\mathcal{A})$, $Ty'(\mu) = Ty(\mu)$.*

This definition of a typing machine is not constructive and not surprisingly, building a typing machine cannot be achieved for arbitrary DESs, but it is very standard in the finite case setting. The construction highly relies on the notion of closure, given here for arbitrary DESs.

Definition 5 *We define the Σ_{uo}-*closure *as the DES $\mathcal{A}_o = (\Sigma_o, Q, \Lambda, q_0, \Delta', Ty')$ such that $\Delta' = \{(p, a, q) \mid a \in \Sigma_o \wedge p \overset{a}{\rightsquigarrow} q\}$ and $Ty'(q) = Ty(q)$ for all $q \in Q$.*

In other words, every unobservable transition of $\mathcal{A}$ is removed in the closure, while preserving the set of observations ($obs(\mathcal{A}_o) = obs(\mathcal{A})$) together with their types ($Ty'(\mu) = Ty(\mu), \forall \mu \in obs(\mathcal{A})$).

Finally, the classical approach to effectively compute typing machines in the *finite case* consists in two operations: (1) compute the Σ_{uo}-*closure* of the DES, and (2) compute its *determinization* (see [7] for example). Clearly, for a finite DES $\mathcal{A}$, the closure $\mathcal{A}_o$ of $\mathcal{A}$ is computable. Now, notice that $\mathcal{A}_o$ may not be deterministic in general, finite DESs can be determinized using a well-known powerset construction. We denote it by $\mathcal{D}(\mathcal{A})$. It is easy to see that given a finite DES $\mathcal{A}$ and $\mathcal{A}_o$ its Σ_{uo}-closure, $\mathcal{D}(\mathcal{A}_o)$ is a typing machine of $\mathcal{A}$.

2.2 Opacity and diagnosis problems

We examine two problems arising in the partial observation setting: the *opacity problem* and the *diagnosis problem*. We first fix some notations that will be used throughout this section. We consider a DES $\mathcal{A} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ with $\Lambda = \{\lambda, \overline{\lambda}\}$. We assume a given partition of Σ: Σ_{uo}, Σ_o. We refer to states with type $\{\lambda\}$ (resp. $\{\overline{\lambda}\}$) as *positive* (*resp. negative*) states. We assume that the set of positive and negative states form a partition of Q (i.e. $Q = Q_\lambda \cup Q_{\overline{\lambda}}$). Due to partial observation, an observation $\mu \in \Sigma_o^*$ may correspond to runs of the system reaching both positive and negative states. Such an observation will have the type $\{\lambda, \overline{\lambda}\}$. To have a uniform terminology, an observation with type $\{\lambda\}$ (*resp.* $\{\overline{\lambda}\}$) is called positive (*resp.* negative). An observation with type $\{\lambda, \overline{\lambda}\}$ is called *equivocal*.

Remark 2 *Notice that we have considered systems with labels on states (state-based framework). This framework is equivalent to the one where the system and the property are given apart and modeled by two distinct DESs or languages. In the finite case, this equivalence is an effective polynomial reduction. We will thus present the problems of opacity and diagnosis with respect to the state-based framework, keeping in mind that it applies to other contexts, for example the language-based framework.*

2.2.1 The opacity problem.

Opacity was introduced by [6] as a security property. The problem of opacity consists of determining whether an attacker, that knows the system and having only a partial observations of the system, is able or not to discover some secret behaviors occurring during execution. We here assume that the system is given by a DES $\mathcal{A}$ with $\Lambda = \{\lambda, \overline{\lambda}\}$ and that the secret is modeled by the state space Q_λ. Intuitively, there is no information flow (i.e. the system is opaque) as far as the attacker cannot surely infer, based on the observations, that after an execution the system is in Q_λ.

More formally the secret set of states Q_λ is opaque w.r.t. $\mathcal{A}$ and Σ_o, if

$$\forall \mu \in \Sigma_o^*, Ty(\mu) \neq \{\lambda\} \tag{1}$$

Definition 6 *Given a DES $\mathcal{A}$ with $\Sigma_o \subseteq \Sigma$ and $\Lambda = \{\lambda, \overline{\lambda}\}$ the opacity problem consists in checking whether (1) holds.*

Example 2 Consider the DES $\mathcal{A}$ on Figure 2, with $\Sigma_o = \{a, b\}$. The secret is given by the set of states $Q_\lambda = \{q_2, q_5\}$.

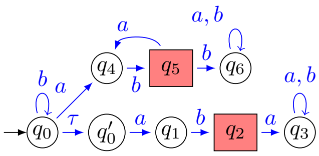

Fig. 2 Non-opaque system

Q_λ is not opaque, as after the observation of a trace in b^*ab, the attacker knows that the system is in a secret state. Note that he does not know whether it is q_2 or q_5 but he knows that the state of the system is in Q_λ. ◇

The language universality problem is shown reducible to the opacity problem ([9]), hence, for arbitrary DESs, it is undecidable (see [6] for more detailed results). However, in the finite-state case, opacity is decidable.

Proposition 1 *([9]) The opacity problem is* PSPACE*-complete for non-deterministic finite DESs.*

Checking the opacity of a secret in a system $\mathcal{A}$ and Σ_o relies on the computation the Σ_{uo}-closure of $\mathcal{A}$ and the corresponding deterministic DES $\mathcal{D}(\mathcal{A}_o)$. It is then sufficient to search for a state with type $\{\lambda\}$ in this new DES. If the system is not opaque, the next step is to build a monitor detecting any information flow. Assuming that this monitor also observes $\mathcal{A}$ though Σ_o, following [15], it is based on the typing machine $\mathcal{D}(\mathcal{A}_o)$. Intuitively, whenever a state of type $\{\lambda\}$ is reached after an observation in $\mathcal{D}(\mathcal{A}_o)$, then an information flow just occurred and the monitor can raise an alarm. Note that the problem of the detection of information flow becomes more intricate whenever the monitor observes $\mathcal{A}$ through a set of events that differs from Σ_o as in [15]. We shall come back to this point in Section 2.2.3.

Example 3 Back to Example 2, the typing machine is depicted in Figure 3. State $\{q_2, q_5\}$ is reachable from the initial state, hence the secret is not opaque. When verifying opacity, the construction of the typing machine may be stopped as soon as state $\{q_2, q_5\}$ is reached. Furthermore, in order to monitor at run-time the information flow this machine may be trimmed to states $\{q_0\}, \{q_1, q_4\}, \{q_2, q_5\}$ and an alarm corresponding to an information may be raised as soon as the state $\{q_2, q_5\}$ is reached. ◇

2.2.2 The diagnosis problem.

The diagnosis problem is a notion introduced by [24] which consists of detecting the satisfaction of a persistent property in a partial observation setting (in [24], the property encodes the fact that some fault has occurred in the system). As for opacity,

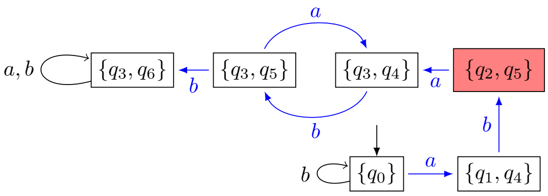

Fig. 3 The typing machine of Figure 2

we attach to each states of $\mathcal{A}$ either the type $\{\lambda\}$ or $\{\overline{\lambda}\}$ with the hypothesis that for all positive states (i.e. typed by $\{\lambda\}$), only positive states can be reached. Hence, a negative state means that the property has not been satisfied so far when the system reaches this state, and a positive state means that the property has been satisfied in the past.

We define the *Diagnosis Problem* as the problem of synthesising a diagnoser, that is, a function $Diag : Obs(\mathcal{A}) \rightarrow \{yes, no, ?\}$ on observations whose output value answers the question whether all traces corresponding to the observation have reached Q_λ. The function should verify:

$$Diag(\mu) = \begin{cases} yes & \text{if } Ty(\mu) = \{\lambda\} \\ no & \text{if } Ty(\mu) = \{\overline{\lambda}\} \\ ? & \text{otherwise.} \end{cases}$$

Clearly, as for the monitoring of opacity, the diagnoser of a DES $\mathcal{A}$ can be derived from its typing machine $\mathcal{D}(\mathcal{A}_o)$ with the convention that the verdict yes is attached to each state of type $\{\lambda\}$, no to each state of type $\{\overline{\lambda}\}$ and ? to each state of type $\{\lambda, \overline{\lambda}\}$. For the diagnoser to have practical interest, when a $\{\lambda\}$-typed trace takes place, we expect Function $Diag$ to eventually output the yes verdict. This can be formally captured by the notion of diagnosability: given a DES $\mathcal{A} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ and $\Sigma_o \subseteq \Sigma$, a proposition $\lambda \in \Lambda$ is *diagnosable in* $\mathcal{A}$ *w.r.t.* Σ_o whenever for all $u \in \mathcal{L}_{Q_\lambda}(\mathcal{A})$, there exists $n \in \mathbb{N}$ such that for every $t \in \mathcal{L}(\mathcal{A})/u$ with $\|\pi(t)\| \geq n$,

$$\pi^{-1}(u.t) \cap \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_{Q_\lambda}(\mathcal{A})$$

In other words, λ is diagnosable in $\mathcal{A}$ w.r.t. Σ_o whenever it is possible to detect, within a finite delay and based on the observation, that the current run of the system reached a positive state.

This can be equivalently formulated as follows.

Lemma 1 *([21]) A proposition λ is not diagnosable in $\mathcal{A}$ w.r.t. Σ_o if, and only if, there exist two infinite runs having the same observation, such that one reaches Q_λ and the other not.*

Definition 7 *The* diagnosability problem *consists in the following: Given a DES $\mathcal{A}$ with $\Sigma_o \subseteq \Sigma$ and $\Lambda = \{\lambda, \overline{\lambda}\}$, is λ diagnosable in $\mathcal{A}$ w.r.t. Σ_o ?*

In the general case, the diagnosability problem is undecidable ([21]): it is proved by an easy reduction of the emptiness of the intersection of context-free languages. On the contrary, the problem is decidable for finite-state DESs.

Proposition 2 *([25]) The problem of diagnosability for finite DESs is in* PTIME.

The standard procedure consists in contructing the self-product of the closure of the system and in seeking an "equivocal" loop in this structure ([18]).

2.2.3 Detection of information flows

Given a DES $\mathcal{A} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$, $\Lambda = \{\lambda, \overline{\lambda}\}$ and $\Sigma_o \subseteq \Sigma$, using the techniques described in the preceding sections, it is possible to check whether Q_λ is opaque w.r.t. $\mathcal{A}$ and Σ_o. When Q_λ is not opaque, it can be important for an administrator to supervise the system on-line and raise an alarm as soon as possible whenever an information flow occurs. We assume that the administrator observes $\mathcal{A}$ through the set of events $\Sigma_m \subseteq \Sigma$. The question is critical when Σ_m differs from Σ_o, otherwise the typing machine $\mathcal{D}(\mathcal{A}_o)$ implements the monitor (see Section 2.2.1).

The attacker knows that the system is in the secret set of states Q_λ after a trace $t \in \mathcal{L}(\mathcal{A})$ if and only if $\pi_o(t) \in \mathcal{L}_F(\mathcal{D}(\mathcal{A}_o))$ where F is the set of states of $\mathcal{D}(\mathcal{A}_o)$ with type $\{\lambda\}$. In general, it will not be possible to detect immediately the information flow. However, under some conditions, it will possible to detect that it occurred in the past. We thus investigate how to diagnose the property *"An information flow occurred in the past."* which is characterized by the following extension-closed language[1]

$$\mathcal{L}_\lambda = \pi_o^{-1}(\mathcal{L}_F(\mathcal{D}(\mathcal{A}_o))).\Sigma^*$$

The language $\mathcal{L}_\lambda$ can be recognized by a DES $\Omega = (\Sigma, Q^\Omega, \Lambda^\Omega, q_0^\Omega, \Delta^\Omega, Ty^\Omega)$, with $\Lambda^\Omega = \{\lambda_\Omega, \overline{\lambda}_\Omega\}$ such that $\mathcal{L}_{Q^\Omega_{\lambda_\Omega}}(\Omega) = \mathcal{L}_\lambda$ and $\mathcal{L}_{Q^\Omega_{\overline{\lambda}_\Omega}}(\Omega) = \Sigma^* \setminus \mathcal{L}_\lambda$. Note that, as $\mathcal{L}_\lambda$ is extension-closed, Λ^Ω fulfills the hypothesis of Section 2.2.2, i.e., for all states of Ω with type $\{\lambda_\Omega\}$, only states of type $\{\lambda_\Omega\}$ can be reached.

The idea is then to diagnose the fact that a sequence of $\mathcal{L}_\lambda$ has occurred (or not) in $\mathcal{A}$ based on an observation in $Obs_{\Sigma_m}(\mathcal{A})$. To do so, we first build the DES $\mathcal{A}_\Omega = \mathcal{A} \times \Omega$ (for simplicity, we assume that the type of a state (q, q_Ω) in $\mathcal{A}_\Omega$ is given by the type of q_Ω). Further, our aim is to diagnose the fact that, based on an observation $\mu \in Obs_{\Sigma_m}(\mathcal{A}_\Omega)$, all traces corresponding to the observation μ have reached a state of type $\{\lambda_\Omega\}$. Following the methodology of Section 2.2.2, it is possible, based on the typing machine of $\mathcal{A}_\Omega$, to build a diagnoser $Diag_{\mathcal{A}_\Omega} : Obs_{\Sigma_m}(\mathcal{A}) \to \{yes, no, ?\}$ such that

$$Diag_{\mathcal{A}_\Omega}(\mu) = \begin{cases} yes & \text{if } Ty^{\mathcal{A}_\Omega}(\mu) = \{\lambda_\Omega\} \\ no & \text{if } Ty^{\mathcal{A}_\Omega}(\mu) = \{\overline{\lambda}_\Omega\} \\ ? & \text{otherwise.} \end{cases}$$

From $Diag_{\mathcal{A}_\Omega}$, we can deduce the following information:

- $Diag_{\mathcal{A}_\Omega}(\mu) = yes$ means that all traces compatible with μ have type $\{\lambda_\Omega\}$ and thus belong to $\mathcal{L}_\lambda$, which entails that the attacker knows the secret.
- $Diag_{\mathcal{A}_\Omega}(\mu) = no$ means that all traces compatible with μ have type $\{\overline{\lambda}_\Omega\}$ and thus do not belong to $\mathcal{L}_\lambda$, which entails that the attacker does not know the secret.

[1] Once the attacker manage to acquire the secret, then it knows it forever.

- $Diag_{\mathcal{A}_\Omega}(\mu) = ?$ means that the administrator can not deduce anything about the knowledge of the attacker. He may (or may not) know the secret.

In other words, the function $Diag_{\mathcal{A}_\Omega}$ is sound but not complete (i.e., there might exist traces of $\mathcal{A}$ corresponding to an information flow, but based on its own observation (w.r.t. Σ_m), the administrator is not aware of it). A necessary and sufficient condition of the detection of every information flow is given by the following proposition:

Proposition 3 *[15] With the preceding notations, an information flow from $\mathcal{A}$ to the attacker w.r.t. Σ_o is certainly detected by the diagnoser if and only if λ_Ω is diagnosable in $\mathcal{A}_\Omega$ w.r.t. Σ_m.*

Remark 3 *It is easy to remark that in instead of considering $Diag_{\mathcal{A}_\Omega}$, we may have equivalently considered $Diag_\Omega$ (it is not necessary to build the product between $\mathcal{A}$ and Ω). However, Proposition 3 would not hold anymore as we shall only consider traces of $\mathcal{A}$ that lead to states of type λ_Ω in order to check diagnosability. Therefore, in that case, the diagnoser is sound but not complete. We shall come back to this point in section 4.2.*

3 Recursive Tile Systems and their properties

In this section, we define the *Recursive Tile Systems* (RTS), a model to define infinite-state DESs based on the regular graphs of [14]. We develop some key aspects of these systems mostly regarding to closure properties (deletion of unobservable events), product and determinization that will be useful for partial observation problems.

Definition 8 *A* recursive tile system *(RTS) is a tuple $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ where*

- *$\Sigma = \Sigma_o \cup \Sigma_{uo}$ is a finite alphabet of* events *partitioned into observable (Σ_o) and unobservable ones (Σ_{uo}),*
- *Λ is a finite set of* colours *with* `init` $\in \Lambda$*.*
- *$\mathcal{T}$ is a set of tiles; a* tile on (Σ, Λ) *is structure $\tau = ((\Sigma, \Lambda), Q_\tau, \rightarrow_\tau, \mathcal{C}_\tau, S_\tau, F_\tau)$ with*
  - *$Q_\tau \subseteq \mathbb{N}$ is a finite set of* vertices*,*
  - *$\rightarrow_\tau \subseteq Q_\tau \times \Sigma \times Q_\tau$ is a finite set of* transitions*,*
  - *$\mathcal{C}_\tau \subseteq Q_\tau \times \Lambda$ is a finite set of* coloured vertices*,*
  - *$S_\tau \subseteq Q_\tau$ is the* support *of τ (it serves to append tile τ to other tiles of $\mathcal{T}$).*
  - *$F_\tau \subseteq \mathcal{T} \times 2^{\mathbb{N} \times Q_\tau}$, the* frontier *of τ. A pair $(\tau', f') \in F_\tau$ tells how tile τ' can be appended to τ by glueing (identifying) the vertices of τ''s support S' with vertices of τ (see Definition 10); hence, $f' : S_{\tau'} \rightarrow Q_\tau$.*
- *$\tau_0 \in \mathcal{T}$ is an initial tile (the axiom).*

In the following we will need to interpret a tile as a DES.

Definition 9 *Given a tile $\tau = ((\Sigma, \Lambda), Q_\tau, \rightarrow_\tau, \mathcal{C}_\tau, S_\tau, F_\tau)$ (of some RTS), a vertex $q_0 \in Q_\tau$ with $(q_0, \texttt{init}) \in \mathcal{C}_\tau$, we define the* underlying DES of τ *by:*

$$[\tau]_{q_0} = (\Sigma, Q_\tau, \Lambda, q_0, \delta, Ty)$$

where $\delta = \rightarrow_\tau$, and for each $q \in Q_\tau$, $Ty(q) = \{\lambda \mid (q, \lambda) \in C_\tau\}$.

As the basic operation for performing tiling (see further), we formalize how tiles can be appended.

Definition 10 *Given two tiles $\tau = ((\Sigma, \Lambda), Q, \rightarrow, \mathcal{C}, S, F)$ and $\tau' = ((\Sigma, \Lambda), Q', \rightarrow'$ $, \mathcal{C}', S', F')$, we say that tile τ' is f'-*appendable to τ *whenever $(\tau', f') \in F$. In that case, and without loss of generality, even if we have to rename all vertices of $Q_{\tau'}$, we may assume that $Q \cap Q' = \emptyset$.*

*Furthermore, whenever τ' is f'-*appendable to τ *(recall $Q \cap Q' = \emptyset$), the f'-*appending of τ' to τ *is the tile*

$$\tau \bullet_{f'} \tau' = ((\Sigma, \Lambda), Q", \rightarrow ", \mathcal{C}", S", F")$$

where

- $Q" = Q \cup P'$, *where* $P' := Q' \setminus S'$,
- $\rightarrow " = \begin{cases} \rightarrow \cup (\rightarrow')_{|(P' \times P')} \\ \cup\{(q, a, q') | q \in Q, q' \in P', \exists s' \in S', f'(s') = q \text{ and } (s', a, q') \in \rightarrow'\} \\ \cup\{(q', a, q) | q' \in P', q \in Q, \exists s' \in S', f'(s') = q \text{ and } (q', a, s') \in \rightarrow'\} \\ \cup\{(q_1, a, q_2) | \exists s_1, s_2 \in S_{\tau'}, f'(s_i) = q_i (i = 1, 2) \text{ and } (s_1, a, s_2) \in \rightarrow'\} \end{cases}$
- $\mathcal{C}" = \mathcal{C}_{|(Q \times \Lambda)} \cup \mathcal{C}'_{|P'} \cup \{(f'(s'), \lambda) \mid (s', \lambda) \in \mathcal{C}'\}$
- $S" = S$
- $F" = (F \setminus \{(\tau', f')\}) \cup F'$

We may extend the append operation even if tile τ' is not appendable to τ: we can enlarge F_τ by adding all elements of the form $(\tau', \emptyset)$ for all τ' not appendable to τ; in this case we let $\tau \bullet_\emptyset \tau' = \tau$.

Example 4 provides three tiles which may be used to monitor the production system presented in Example 1.

Example 4 The tiles depicted in Figure 4 model the property we want to monitor on the production system defined in Example 1. The $\tau_{\mathtt{main}}$ tile ensures that the system is initialised for resource A and that it is cleaned at the end. The $\tau_{\mathtt{f}}$ tile ensures that each resource A stacked is used before the $\mathtt{clean_A}$. It also ensures that the urgency mode can be activated at every moment. The τ_{g} tile ensures that each resource B will be treated before returning to the normal mode. The check self loop represents an inspection routine that can occur at any time. Formally, the corresponding RTS is defined by: $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_{\mathtt{main}})$ with $\Sigma_o = \{\mathtt{check}, \mathtt{stack_A}, \mathtt{stack_B}, \mathtt{use_A}, \mathtt{use_B}\}$, $\Sigma_{uo} = \{\mathtt{init_A}, \mathtt{init_B}, \mathtt{clean_A}, \mathtt{clean_B}\}$, $\Lambda = \{\mathtt{init}\}$, $\mathcal{T} = \{\tau_{\mathtt{main}}, \tau_{\mathtt{f}}, \tau_{\mathrm{g}}\}$, and $\tau_{\mathtt{main}}$ the initial tile.

- $\tau_{\mathtt{main}} = ((\Sigma, \Lambda), Q_{\tau_{\mathtt{main}}}, \rightarrow_{\tau_{\mathtt{main}}}, \mathcal{C}_{\tau_{\mathtt{main}}}, S_{\tau_{\mathtt{main}}}, F_{\tau_{\mathtt{main}}})$ with
  $Q_{\tau_{\mathtt{main}}} = \{0, 1\}$, $\mathcal{C}_{\tau_{\mathtt{main}}} = \{(0, \mathtt{init})\}$ (init depicted by $\rightarrow\bigcirc$)
  $S_{\tau_{\mathtt{main}}} = \emptyset$, $F_{\tau_{\mathtt{main}}} = \{(\tau_{\mathtt{f}}, \{0 \rightarrow 1\})\}$, and $\rightarrow_{\tau_{\mathtt{main}}}$ depicted in Figure 4,
- $\tau_{\mathtt{f}} = ((\Sigma, \Lambda), Q_{\tau_{\mathtt{f}}}, \rightarrow_{\tau_{\mathtt{f}}}, \mathcal{C}_{\tau_{\mathtt{f}}}, S_{\tau_{\mathtt{f}}}, F_{\tau_{\mathtt{f}}})$ with
  $Q_{\tau_{\mathtt{f}}} = \{0, 1, 2\}$, $\rightarrow_{\tau_{\mathtt{f}}}$ $\mathcal{C}_{\tau_{\mathtt{f}}} = \emptyset$
  $S_{\tau_{\mathtt{f}}} = \{0\}$, $F_{\tau_{\mathtt{f}}} = \{(\tau_{\mathtt{f}}, \{0 \rightarrow 1\}), (\tau_{\mathrm{g}}, \{0 \rightarrow 2\})\}$ and $\rightarrow_{\tau_{\mathtt{f}}}$ depicted in Figure 4.
- $\tau_{\mathrm{g}} = ((\Sigma, \Lambda), Q_{\tau_{\mathrm{g}}}, \rightarrow_{\tau_{\mathrm{g}}}, \mathcal{C}_{\tau_{\mathrm{g}}}, S_{\tau_{\mathrm{g}}}, F_{\tau_{\mathrm{g}}})$ with
  $Q_{\tau_{\mathtt{f}}} = \{0, 1\}$, $\rightarrow_{\tau_{\mathrm{g}}}$ $\mathcal{C}_{\tau_{\mathrm{g}}} = \emptyset$
  $S_{\tau_{\mathrm{g}}} = \{0\}$, $F_{\tau_{\mathrm{g}}} = \{(\tau_{\mathrm{g}}, \{0 \rightarrow 1\})\}$ and $\rightarrow_{\tau_{\mathrm{g}}}$ depicted in Figure 4.

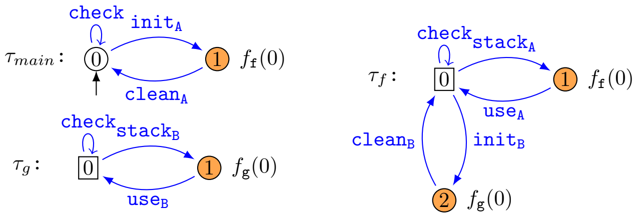

Fig. 4 An RTS.

For the frontier, e.g., in the tile $\tau_{\mathtt{main}}$, ① $f_{\tau_\mathtt{f}}(0)$ means that $(\tau_\mathtt{f}, \{0 \to 1\})$ belongs to $F_{\tau_{\mathtt{main}}}$, i.e. the vertex 0 of $\tau_\mathtt{f}$ is associated to the vertex 1 of $\tau_{\mathtt{main}}$. The square vertices are vertices of the support. Note that the axiom has no support. ◇

One easily verifies the following "parallel append" property:

Lemma 1 *Let τ be a tile and let $(\tau_1, f_1), (\tau_2, f_2) \in F_\tau$, with τ_1 and τ_2 appendable to τ. We have $(\tau \bullet_{f_1} \tau_1) \bullet_{f_2} \tau_2 = (\tau \bullet_{f_2} \tau_2) \bullet_{f_1} \tau_1$, modulo the renaming of vertices.*

Given an RTS $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ and $(q_0, \mathtt{init}) \in \mathcal{C}_{\tau_0}$, one can derive a DES $[\tau_0] = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ with $\forall q \in Q, Ty(q) = \{c \mid (q, c) \in \mathcal{C}\}$. The construction needs first to introduce notion of *tiling* of a tile by a set of tiles.

Definition 11 *Given a set of tiles $\mathcal{T}$ and a tile $\tau = ((\Sigma, \Lambda), Q, \to, \mathcal{C}, S, F)$ in $\mathcal{T}$, the* tiling of τ by $\mathcal{T}$ *is the tile $\mathcal{T}(\tau)$ obtained by appending to τ any $\tau' \in \mathcal{T}$, whenever possible:*

$$\mathcal{T}(\tau) = \tau\, \{\bullet_{f'} \tau'\}_{(\tau', f') \in F}$$

(which is well defined by Lemma 1).

Example 5 We illustrate the principle of tiling using the RTS defined in Example 4. Consider that $\tau_{\mathtt{main}}$ is the initial tile. Its tiling $\mathcal{T}(\tau_{\mathtt{main}})$, is performed as follows: there is a single element in its frontier; we add a copy of $\tau_\mathtt{f}$ (with new vertices), identifying vertex 1 of $\tau_{\mathtt{main}}$ to vertex 0 of $\tau_\mathtt{f}$. This new tile may be in turn extended by adding a copy of $\tau_\mathtt{f}$, identifying 3 to 0 of $\tau_\mathtt{f}$ and 4 to 0 of $\tau_\mathtt{g}$. We illustrate the resulting tile in Fig. 5. Observe that this tile has no support since it is a tiling of the axiom.

A tiling of some tile τ by a set of tiles $\mathcal{T}$ may be iterated: after having built $\mathcal{T}(\tau)$, we may keep on building $\mathcal{T}^2(\tau)$ (that is the tile $\mathcal{T}(\mathcal{T}(\tau))$), then $\mathcal{T}^3(\tau)$, etc.[2]

Now, the DES obtained from an RTS is defined as the infinite limit of the DESs of tiles resulting from iterating tilings starting from the axiom tile. Formally,

Definition 12 *Let $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ be an RTS and let $q_0 \in Q_{\tau_0}$ with $(q_0, \mathtt{init}) \in C_{\tau_0}$.*

The DES associated to of $\mathcal{R}$ starting from q_0 *is*

$$\llbracket \mathcal{R} \rrbracket_{q_0} \overset{\textbf{def}}{=} lim_{k \to \infty} [\mathcal{T}^k(\tau_0)]_{q_0}$$

By writing $\llbracket \mathcal{R} \rrbracket_{q_0} = (\Sigma, Q_\mathcal{R}, \Lambda, q_0, \Delta_\mathcal{R}, Ty_\mathcal{R})$, assuming $Q_{\mathcal{R}\,\mathtt{init}} = \{q_0\}$.

[2] while taking care of the preliminar renaming of vertices at each tiling step, as explained in Definition 10.

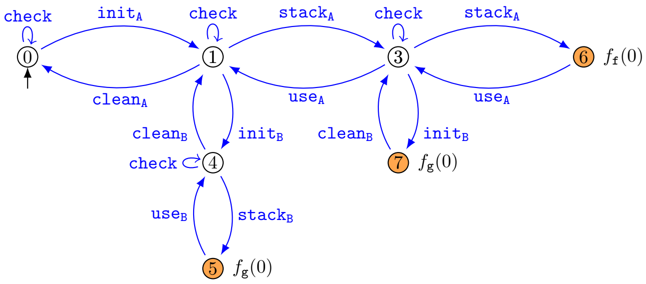

Fig. 5 The DES defined by the tile $\mathcal{T}^2(\tau_{\mathtt{main}})$

For an RTS $\mathcal{R}$ with axiom τ_0, and a state q in $[\![\mathcal{R}]\!]_{q_0}$, $\ell(q)$ denotes the *level* of q, *i.e.* the least $k \in \mathbb{N}$ such that q is a state of $[\mathcal{T}^k(\tau_0)]$, and $\tau(q)$ denotes the tile in $\mathcal{T}$ that generate q. For a vertex v of a tile in $\mathcal{R}$, $[\![v]\!]$ denotes the set of states in $[\![\mathcal{R}]\!]_{q_0}$ corresponding to v.

Remark 4 *The DESs obtained from RTSs correspond to the equational, or regular graphs of [14] and [10] (derived from an axiom using deterministic* HR*-grammars). In fact RTSs have been introduced in [13] and aims at a greater simplicity. More details relating RTSs with regular graphs are found in [12].*

Remark 5 (Complexity measure) *In this paper we will provide (or recall) space or time complexities for the algorithms we devise. The purpose is to be able to compare the performances of these algorithms with respect to those of finite DESs.*

The size of a DESs defined by some RTS is given by the total number *of vertices and transitions in the whole set of tiles. Observe that a finite DES is defined by a single tile RTS (with empty support and empty frontier); such a system has equal size measures as an RTS or as a finite DES.*

Next, we go through some properties on RTSs that shall be employed for analysing opacity and diagnosability of RTS.

Reachability: Computation of reachability sets, central for verification problems, is effective for RTSs:

Proposition 4 (Adapted from [10]) *Given an RTS $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$, a sub-alphabet $\Sigma' \subseteq \Sigma$, a colour $\lambda \in \Lambda$, and a new colour $r_\lambda \notin \Lambda$, an RTS $\mathcal{R}' = ((\Sigma, \Lambda \cup \{r_\lambda\}), \mathcal{T}', \tau_0')$ can be effectively computed, such that $[\![\mathcal{R}']\!]_{q_0}$ is isomorphic to $[\![\mathcal{R}]\!]_{q_0}$ with respect to the transitions and the colouring by Λ, and states reachable from a state coloured λ by events in Σ' are coloured by r_λ: $Q_{r_\lambda} = reach_{[\![\mathcal{R}']\!]_{q_0}}(Q_\lambda, \Sigma')$.*

As stated in [12], Proposition 4 produces its result (system $\mathcal{R}'$) in polynomial time with respect to the size of the initial RTS $\mathcal{R}$. Itself, $\mathcal{R}'$ has a polynomial size, and its semantic is isomorphic to $[\![\mathcal{R}]\!]_{q_0}$: there is a bijection φ between the set of states of $[\![\mathcal{R}']\!]_{q_0}$ and the one of $[\![\mathcal{R}]\!]_{q_0}$, furthermore, transitions and colours are compatible with φ. Hence these systems are identical with respect to the behaviour they model but $\mathcal{R}'$ contains extra information: states reachable from colour λ have colour r_λ.

Detecting infinite paths is central for the computation of the closure of DESs defined by RTSs, this closure is the backbone of the diagnosability and opacity problems. Infinite paths can be either *divergent*, or composed of cycles. A *divergent* path contains infinitely many distinct states, its existence impacts on the finite representability of the closure. Moreover, arbitrary infinite paths need to be considered for verifying diagnosability.

Proposition 5 *Given an RTS $\mathcal{R}$ and colour λ, the existence of an infinite path such that after some index $i_0 \in \mathbb{Z}$ every state of index i ($i \geq i_0$) has type $\lambda \in \Lambda$ in $[\![\mathcal{R}]\!]_{q_0}$ is decidable in polynomial time.*

Proposition 5 derives from Lemma 1 in [13, 12]. Thus we provide here the statement, and refer to [12] for a proof.

Lemma 2 *For an RTS $\mathcal{R}$, there exists a loop or a divergent path in $[\![\mathcal{R}]\!]_{q_0}$ if and only if there exists a vertex v and two states $q_1, q_2 \in [\![v]\!]$ with $\ell(q_1) \leq \ell(q_2)$ such that $q_1 \xrightarrow{w} q_2$ for some $w \in (\Sigma_{uo})^+$ and for all states q on this path, $\ell(q_1) \leq \ell(q)$.*

This lemma enables to prove Proposition 5.

Proof (*of Proposition 5*) Without loss of generality, from Proposition 4, we may assume that each vertex in $\mathcal{R}$ is reachable from `init` (up to a polynomial cost).

Now, the existence of a path satisfying Proposition 5 is equivalent to the existence of a reachable infinite path crossing only λ-labelled states.

Let $\mathcal{R}'$ be the restriction of $\mathcal{R}$ to vertices coloured by λ. Lemma 2 enables to assert the existence an infinite path in $\mathcal{R}'$ whenever there are self-reachable vertices. Hence, detecting an infinite path amounts to detecting self-reachable states in $\mathcal{R}'$. From each vertex of $\mathcal{R}'$ we use Proposition 4 (in polynomial time) to check if it reaches itself. If it is the case for any vertex of $\mathcal{R}$ then the system possesses an infinite λ-labelled path, otherwise it does not. □

Observable behaviour of RTSs: Abstracting away unobservable transitions is important for partial observation questions. The following proposition (from [13, 12][3]) computes the closure of RTSs.

Proposition 6 *Let $\mathcal{R}$ be an RTS with observable events $\Sigma_o \subseteq \Sigma$, and $[\![\mathcal{R}]\!]_{q_0} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ its DES. One can effectively compute an RTS $Clo(\mathcal{R})$ of exponential size whose associated DES $[\![Clo(\mathcal{R})]\!]_{q_0'} = (\Sigma_o, Q', \Lambda, q_0', \Delta', Ty')$ has no unobservable event, is of finite* out-*degree, and for any colour $\lambda \in \Lambda$, $obs_{Q_\lambda}([\![\mathcal{R}]\!]_{q_0}) = \mathcal{L}_{Q'_\lambda}([\![Clo(\mathcal{R})]\!]_{q_0'})$.*

Determinization of RTSs: In the following we consider weighted RTSs. This class possesses the property of being determinizable and closed by self-product.

Definition 13 *An RTS $\mathcal{R}$ is* weighted for λ *if in its DES $[\![\mathcal{R}]\!]_{q_0} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$, if Q_λ is a singleton $\{q_\lambda\}$, and for any $u \in \Sigma^*$ and any states $q, q' \in Q$, $q_\lambda \xrightarrow{u} q$ and $q_\lambda \xrightarrow{u} q'$ implies $\ell(q) = \ell(q')$ (same level).*

[3] Section 3.3 in [12] provides a detailled proof of this proposition.

Since we use RTSs to represent DESs we simply say that a RTS is weighted, whenever it is weighted for `init`.

Remark 6 *Determining if an RTS is weighted for any given colour is decidable, using an algorithm from [11] and the time complexity is polynomial (see [12], p17).*

The construction of typing machines used to check the opacity of a system and build the corresponding monitor highly relies on the determinization operation. An RTS $\mathcal{R}$ is *deterministic* if its underlying DES $[\![\mathcal{R}]\!]_{q_0}$ is deterministic. This is decidable from the set of tiles defining it.

Proposition 7 ([11]) *Any weighted RTS $\mathcal{R}$ can be transformed into a* deterministic *one $D(\mathcal{R})$ with same set of traces and, for any colour, same traces accepted in this colour. The resulting RTS is exponential in the size of $\mathcal{R}$.*

The next theorem provides sufficient conditions under which the determinization of a weighted RTS $\mathcal{R}$ yields a typing machine of $[\![\mathcal{R}]\!]_{q_0}$.

Proposition 8 *Let $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ be an RTS such that $Clo(\mathcal{R})$ is a weighted RTS, then determinizing the $Clo(\mathcal{R})$ produces a typing machine of $[\![\mathcal{R}]\!]_{q_0}$.*

Proof According to Proposition 6, $Clo(\mathcal{R})$ is an RTS and the types of traces are preserved. Finally, as $Clo(\mathcal{R})$ is a weighted RTS, applying Proposition 7, produces a deterministic RTS $\mathcal{D}(Clo(\mathcal{R}))$. This RTS, in turn, preserves the traces of $[\![Clo(\mathcal{R})]\!]_{q'_0}$ from q_0 (the single `init` labeled state in $[\![\mathcal{R}]\!]_{q_0}$), and thus preseving the traces of $[\![\mathcal{R}]\!]_{q_0}$. The types of these traces are also preserved. Hence $[\![\mathcal{D}(Clo(\mathcal{R})]\!]_{q"_0}$ is a typing machine of $[\![\mathcal{R}]\!]_{q_0}$. □

*Synchronous product:*The class of RTS is not closed under synchronous product. Indeed, the intersection of two context-free languages can be obtained by a product of two RTSs, if such a product was recursive the intersection of two context-free languages would be a context-free language ($\{a^n b^n c^k \mid n, k \in \mathbb{N}\} \cap \{a^n b^k c^k \mid n, k \in \mathbb{N}\}$ is not context-free). However, we can prove that the product of an RTS with a finite DES is an RTS. More precisely, given any RTS $\mathcal{R}$ with DES $[\![\mathcal{R}]\!]_{q_0}$, and a finite state DES $\mathcal{A} = (\Sigma, Q_\mathcal{A}, \Lambda_\mathcal{A}, q_{\mathcal{A}0}, \Delta_\mathcal{A}, Ty_\mathcal{A})$, one can compute an RTS denoted by $\mathcal{R} \times \mathcal{A}$ such that $[\![\mathcal{R} \times \mathcal{A}]\!]_{(q_0, q_{\mathcal{A}0})} = [\![\mathcal{R}]\!]_{q_0} \times \mathcal{A}$ (the × on the right-hand side of the equality is the product for DESs). The set of tile $\mathcal{T}_{\mathcal{R}\times\mathcal{A}}$ is formed by the synchronous products of each tile of $\mathcal{T}$ with $\mathcal{A}$. Formally, for a given tile $t_\mathcal{B} \in \mathcal{T}$, with $t_\mathcal{B} = ((\Sigma_\mathcal{A}, \Lambda_\mathcal{M}), Q_\mathcal{B}, \rightarrow_\mathcal{B}, \mathcal{C}_\mathcal{B}, S_\mathcal{B}, F_\mathcal{B})$, the product tile, denoted by $t_\mathcal{B} \times \mathcal{A}$, is the following: $t_\mathcal{B} \times \mathcal{A} = ((\Sigma_\mathcal{A}, \Lambda_\mathcal{M} \times \Lambda_\mathcal{A}), Q_\mathcal{B} \times Q_\mathcal{A}, \rightarrow_{\mathcal{B}\times\mathcal{A}}, \mathcal{C}_{\mathcal{B}\times\mathcal{A}}, S_{\mathcal{B}\times\mathcal{A}}, F_{\mathcal{B}\times\mathcal{A}})$, with the transitions and colours defined like for products of DES in Section 2, the support is simply: $S_\mathcal{B} \times Q_\mathcal{A}$, and, for each $(t_\mathcal{C}, f_\mathcal{C}) \in F_\mathcal{B}$ ($f_\mathcal{C} : S_\mathcal{C} \rightarrow Q_\mathcal{B}$), there is a $(t_{\mathcal{C}\times\mathcal{A}}, f_{\mathcal{C}\times\mathcal{A}})$ with $t_{\mathcal{C}\times\mathcal{A}}$ another tile of the product, and $f_{\mathcal{C}\times\mathcal{A}}$ a function between $S_\mathcal{C} \times Q_\mathcal{A}$ and $Q_\mathcal{B} \times Q_\mathcal{A}$ that associates to any pair $(q_\mathcal{C}, q_\mathcal{A})$ the pair: $(f_\mathcal{C}(q_\mathcal{C}), q_\mathcal{A})$. Any coloured trace of the product may be projected (with respect to colours) on either one of the systems and is a coloured trace of this system.

Nevertheless, even-though the product between two RTSs may not be an RTS for weighted RTSs the self-product (defined in Section 2) is always an RTS. We first provide a syntactic definition of this self-product then Proposition 9 assert the correctness of this construction. This result assumes that the colour `init` is defined in the self-product only for pairs which both have colour `init`)

Given a wheighted RTS $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ we denote its self-product by $\mathcal{R}_2 = ((\Sigma, \Lambda), \mathcal{T}_2, \tau_{02})$ where $\mathcal{T}_2$ is the set of products of tiles. For any two tiles $\tau_1 = ((\Sigma, \Lambda), Q_1, \rightarrow_1, \mathcal{C}_1, S_1, F_1)$ and $\tau_2 = ((\Sigma, \Lambda), Q_2, \rightarrow_2, \mathcal{C}_2, S_2, F_2)$, we denote their product as follows:

$$\tau_{1\times 2} = ((\Sigma, \Lambda), Q_{1\times 2}, \rightarrow_{1\times 2}, \mathcal{C}_{1\times 2}, S_{1\times 2}, F_{1\times 2})$$

Where these sets are defined as follows:

- $Q_{1\times 2} = Q_1 \times Q_2$,
- For each $(q_1, a, q_2) \in \rightarrow_1$ and $(q_1', a, q_2') \in \rightarrow_2$:
$$((q_1, q_1'), a, (q_2, q_2')) \in \rightarrow_{1\times 2}$$
- For every $c \in \Lambda \setminus \{\texttt{init}\}$, $((q_1, q_2), c) \in \mathcal{C}_{1\times 2}$ whenever $(q_1, c) \in \mathcal{C}_1$ or $(q_2, c) \in \mathcal{C}_2$.
- `init` is given to pairs where each vertex has colour `init`.
- $S_{1\times 2} = S_1 \times S_2$,
- For each $(\tau_i, f_i) \in F_1$, and $(\tau_j, f_j) \in F_2$: $(\tau_{i\times j}, f_i \times f_j) \in F_{1\times 2}$, where the product of functions is the function over independent product $(f_i \times f_j(a, b) = (f_i(a), f_j(b)))$.

Proposition 9 *The self-product of an RTS weighted for* `init`*, is an RTS weighted for* `init`*. This object has quadratic size.*

Proof To prove this proposition it is sufficient to consider the weighted RTS $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$, and its self-product $\mathcal{R}_2$, as defined above.

Observe that the computation of $\mathcal{R}_2$ may be performed for any RTS. But, unless $\mathcal{R}$ is weighted[4], there is no guarantee to have $[\![\mathcal{R}_2]\!]_{(q_0,q_0)} = [\![\mathcal{R}]\!]^2_{q_0}$. More precisely, the weighted property ensures that two paths in $[\![\mathcal{R}]\!]_{q_0}$, with identical labels, reach the same level, hence may be carried out in $[\![\mathcal{R}_2]\!]_{(q_0,q_0)}$.Thus, a simple induction proves that any path, from `init`, in $[\![\mathcal{R}]\!]^2_{q_0}$ may be performed in $[\![\mathcal{R}_2]\!]_{(q_0,q_0)}$ (the converse is always true).

Our definition of the frontier of $\mathcal{R}_2$ implies that no mixed pairs (containing a vertex in one frontier and the other which is not) belongs to the domain of any function, hence if the RTS is not weighted some paths may be shortened (the weighted condition forbids paths leaving one tile in one copy while staying in the same tile in the other). □

Example 6 *Figure 6 depicts an RTS $\mathcal{R}$ composed of two tiles, A and B. The shaded areas in each tile represents the frontier. The right-hand side illustrates the tile AB which is one of the 4 tiles of $\mathcal{R}_2$, the self-product (these tiles are AA, AB, BA and BB). In its frontier, each of the other product-tile appears.*

[4] This is a sufficient condition. There might be others.

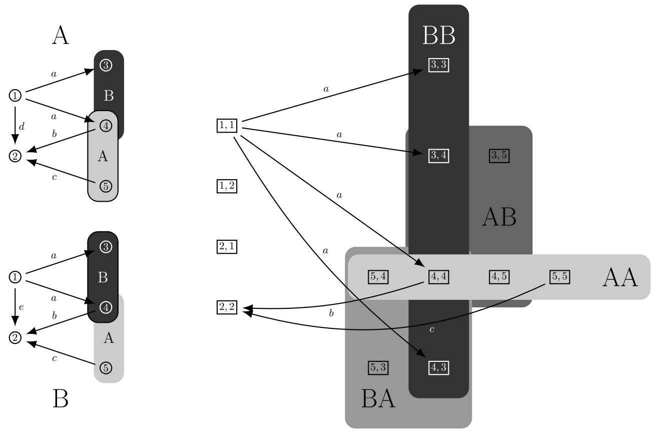

Fig. 6 An RTS formed by tiles A and B, and tile AB of its self-product

4 Opacity and diagnosis problems for RTS

We come back to opacity and diagnosability problems, but focusing on systems definable by RTS. We characterize sufficient conditions over RTSs for the opacity problem and diagnosability problems to become decidable, as it is not the case in general.

As already observed in Remark 2, our state-based framework is equivalent to the one where the system and the property are given apart. Since RTSs are not closed under synchronous product (see Section 3), but closed under product with a finite DES, in the context of diagnosability and opacity problems, it makes sense to assume that a given system results from the product of a finite DES with some RTS. This enables us to consider either regular properties with infinite-state systems, or symmetrically finite-state systems with non-regular properties.

4.1 Diagnosis and opacity for RTS

Proposition 10 *The opacity problem is undecidable for RTSs.*

Proof We reduce the inclusion problem for context-free languages: Let $\mathcal{L}$ and $\mathcal{L}'$ be two context-free languages over an alphabet Σ and two new symbols s and $\#$ not in Σ (we denote by $\Sigma_\#$ the set $\Sigma \cup \{\#\}$). We consider the languages formed by the concatenation of the $\#$ symbol at the end of each element of the languages: $\mathcal{L}\#$ and $\mathcal{L}'\#$ (we use these languages rather than the original ones in order to have a reduction of the inclusion of the languages rather than their sets of prefixes). These languages can be represented[5] respectively by $\mathcal{R} = ((\Sigma_\#, \Lambda), \mathcal{T}, \tau_0)$ and $\mathcal{R}' = ((\Sigma_\#, \Lambda), \mathcal{T}', \tau_0')$ two RTSs each having a single vertex labelled by `init` (vertices q_0 and q_0' in tiles τ_0 and τ_0' respectively) and such that the accepting states[6] of $[\![\mathcal{R}']\!]_{q_0'}$ have colour λ. $\overline{\lambda}$

[5] This simple construction of a tiling system with an initial vertex is presented in [10], Section 5 in the context of deterministic graph grammar.

[6] By construction, those reached by a path in $\mathcal{L}'\#$

holds in all the other states of $\llbracket\mathcal{R}\rrbracket_{q_0}$ and $\llbracket\mathcal{R}'\rrbracket_{q_0'}$. Let us now consider the RTS $\mathcal{R}" = ((\Sigma_{\#} \cup \{s\}, \Lambda), \mathcal{T} \cup \mathcal{T}' \cup \{\tau_0''\}, \tau_0'')$ such that $\tau_0'' = ((\Sigma_{\#} \cup \{s\}, \Lambda), \{0, 1\}, \{0 \xrightarrow{s} 1\}, \{(0, \texttt{init})\}, \emptyset, \{(\tau_0, 0, 0), (\tau_0', 0, 1)\})$, and $(s \in \Sigma_{uo})$. In the RTS $\mathcal{R}"$, every path leading to a secret state corresponds to a word in $\mathcal{L}'\#$, and each path finishing with the symbol $\#$ leading to non-secret state corresponds to a word of $\mathcal{L}\#$, thus, $\mathcal{L}' \subseteq \mathcal{L}$ if, and only if, the set of secret states is opaque w.r.t. $\mathcal{R}"$ and $\Sigma_{\#}$. □

We can mimic the classic decision procedure to solve the opacity problem on finite-state systems, which (as explained in Section 2.2) relies on the construction of a typing machine. Such a machine can be built whenever the closure of the RTS is also weighted. Let us denote by $\mathcal{C}wRTS$ the class of RTSs whose closure[7] is weighted (recall it is already an RTS).

Theorem 1 *The opacity problem is in* 2-EXPTIME *over the class $\mathcal{C}$wRTS. Furthermore, this problem is* EXPTIME*-hard.*

Proof We give the decision procedure: let $\mathcal{R} \in \mathcal{C}$wRTS and q_0 be an initial vertex, as $Clo(\mathcal{R})$ is weighted, we apply Proposition 8 and obtain an RTS $\mathcal{D}(Clo(\mathcal{R}))$ such that $\llbracket\mathcal{D}(Clo(\mathcal{R}))\rrbracket_{q"_0}$ is a typing machine. It is then sufficient to solve the reachability of $\{\lambda\}$-states in $\llbracket\mathcal{D}(Clo(\mathcal{R}))\rrbracket_{q"_0}$, which is decidable by Proposition 4. This may be performed in doubly-exponential time (since the size of $\mathcal{D}(Clo(\mathcal{R}))$ is doubly exponential, and the reachability polynomial). For EXPTIME-hardness, universality is EXPTIME-complete for visibly pushdown languages [2], which may be generated by weighted RTSs. But universality may be reduced to opacity (assume the universal language with secrete states on the one side and the language tested for universality on the other side, the system is opaque, if and only if the tested language is universal), hence opacity is EXPTIME-hard. □

Currently our 2-EXPTIME upper-bound seems like a huge step-back with respect to the PSPACE-completeness of the problem for finite-state systems. However, this upper-bound results from the exponential upper-bound of the closure which in most classical case would not be reached. In particular if there is a bound on the length of sequences of unobservable transitions, then the closure is polynomial. Resulting in an EXPTIME upper-bound for checking opacity.

Turning to diagnosability, we have the following.

Proposition 11 *Diagnosability problem for RTSs is undecidable.*

This result is a consequence of [21] since visibly pushdown systems are a strict subclass of weighted RTSs. However, we have a result similar to Theorem 1.

Theorem 2 *The diagnosability problem over the class $\mathcal{C}wRTS$ is in* EXPTIME.

Proof Let $\mathcal{R} \in \mathcal{C}$wRTS, since $Clo(\mathcal{R})$ is weighted, by Proposition 9, there exists an RTS $\mathcal{R}_2$ such that $\llbracket\mathcal{R}_2\rrbracket_{(q_0,q_0)} = \llbracket Clo(\mathcal{R})\rrbracket^2_{q_0}$ (self-product of $\llbracket Clo(\mathcal{R})\rrbracket_{q_0'}$) (Proposition 9). Now, by Lemma 1, non-diagnosability is equivalent to finding an infinite path of $\{\lambda, \overline{\lambda}\}$-typed states in $\llbracket Clo(\mathcal{R})\rrbracket^2_{q_0}$. Which can be decided according to Proposition 5. Turning to the complexity, only polynomial algorithms are used in this proof, and the closure produces an object of exponential size. □

[7] according to Proposition 6, see also [13].

Observe that this EXPTIME upper-bound is much larger than the polynomial bound for finite state systems. However, as noted earlier for the opacity problem, this bound results from the computation of the closure. For every restriction such that the closure of any RTS is of polynomial size (*e.g.*, sequences of unobservable transitions of bounded length), the problem of diagnosability becomes polynomial too.

Note that the RTS $\mathcal{D}(Clo(\mathcal{R}))$ is a valuable object. Indeed, whenever the system is not opaque (following [15] for finite-state systems), $\mathcal{D}(Clo(\mathcal{R}))$ can be exploited to monitor the system $[\![\mathcal{R}]\!]_{q_0}$ and detect effective information flow. Similarly, when diagnosability holds, the RTS $\mathcal{D}(Clo(\mathcal{R}))$ provides the desired diagnoser. In the next section we will develop several aspects related to this question.

4.2 Diagnosability of information flows

As discussed in Section 2.2.3, whenever a system has been identified as non-opaque, it may be useful to monitor its execution and determine whether the secret has actually been revealed. As we have already seen, it amounts to checking the diagnosability w.r.t. Σ_m of the language $\mathcal{L}_\lambda$, restricted to the actual executions of the system, where $\mathcal{L}_\lambda$ is given by:

$$\mathcal{L}_\lambda = \pi_o^{-1}(\mathcal{L}_F([\![\mathcal{D}(Clo(\mathcal{R}))]\!]_{q"_0}).\Sigma^*$$

where F is the set of states of $[\![\mathcal{D}(Clo(\mathcal{R}))]\!]_{q"_0}$ of type $\{\lambda\}$. More precisely, if $\mathcal{R}$ is in $\mathcal{C}$wRTS with respect to Σ_o, then an RTS $\mathcal{R}_\lambda$ representing $\mathcal{L}_\lambda$ is obtained from the three following steps:

1. Determinization produces an RTS modeling $\mathcal{D}(Clo(\mathcal{R})) = \mathcal{R}_1$.
2. In each tile of $\mathcal{R}_1$, each vertex v labeled by $\{\lambda\}$ (representing only states of the DES $[\![\mathcal{R}]\!]_{q_0}$ labeled by $\{\lambda\}$), add a transition to a (new) sink state in the same tile. This new state is labeled $\{\lambda\}$, and has a Σ self-loop. And remove every other out-going edges from v. This produces $\mathcal{R}_2$.
3. In each tile of $\mathcal{R}_2$, add a Σ_{uo}-labeled self-loop at each vertex in order to obtain $\mathcal{R}_\lambda$.

We then have $\mathcal{L}_\lambda = \mathcal{L}_F([\![\mathcal{R}_\lambda]\!]_{q_0})$, where F corresponds to the set of states of $[\![\mathcal{R}_\lambda]\!]_{q_0}$ having type $\{\lambda\}$.

Further, as explained in Section 2.2.3, we aim at diagnosing the occurrence of a sequence of $\mathcal{L}_\lambda$ in the system, based on an observation of Σ_m^*, which entails to perform a product between $\mathcal{R}_\lambda$ and the system. Knowing that, in general, the product of two $RTSs$ is not an RTS, several cases have to be considered. These cases depend on the fact that the original system is modeled by an RTS or by a finite DES.

1. If we assume that system $\mathcal{R}$ is the product of a finite system (say $\mathcal{A}_{fin}$ with initial state q_{fin}) with a secret modeled by some RTS, the synchronous product between $\mathcal{R}_\lambda$ and $\mathcal{A}_{fin}$ is then computable. Further, if $\mathcal{R}_\lambda \times \mathcal{A}_{fin}$ is in $\mathcal{C}$wRTS w.r.t. Σ_m, then according to Theorem 2, one can (1) check for the diagnosability of λ w.r.t. $[\![\mathcal{R}_\lambda \times \mathcal{A}_{fin}]\!]_{(q_0,q_{fin})}$ and Σ_m and (2) build the typing machine (i.e., the diagnoser) $\mathcal{D}(Clo(\mathcal{R}_\lambda \times \mathcal{A}_{fin}))$ in charge of the diagnosis of the occurrence of a sequence of $\mathcal{L}_\lambda$. Thus, whenever λ is diagnosable w.r.t. to $[\![\mathcal{R}_\lambda \times \mathcal{A}_{fin}]\!]_{(q_0,q_{fin})}$

and Σ_m, then, according to Proposition 3, the diagnoser is sound and complete and every information flow will be eventually detected. Otherwise the diagnoser may still be used, but without any guaranty of completeness.

2. If the system is modeled by an RTS, since its execution only outputs actual events of the DES (as noted in Remark 3), it is possible to use $\mathcal{D}(Clo(\mathcal{R}_\lambda))$ to monitor the information flows. Again two situations may occur:
   - $\mathcal{R}_\lambda$ is in $\mathcal{C}$wRTS w.r.t. Σ_m, then the computation of $\mathcal{D}(Clo(\mathcal{R}_\lambda))$ is effective and gives access to the diagnoser.
   - Otherwise, the computation of a deterministic RTS whose behavior corresponds to the observable behavior of $\mathcal{R}_\lambda$ is not possible. However, it is still possible to compute "on the fly" the diagnoser function by computing the possible states in which the DES $[\![\mathcal{R}_\lambda]\!]_{q_0}$ can be after the observation of every new event of Σ_m[8]. Details of such on-line computation can be found in [12].

   In both cases the diagnoser is not complete. But it is always sound since it will only emit alarms when the secret has actually been leaked.

5 Conclusion

The present paper demonstrates a way to extend detection of information flow and property diagnosis for infinite DESs. We could overcome undecidability by exhibiting the subclass of weighted RTSs for which determinization, self-product and detection of infinite path are effective. For these decidable cases, we have established complexity upper bounds for diagnosability and opacity problems. We however do not know yet whether our constructions are optimal or not: we have not been able to reduce classical problems that would yield tight bounds.

Also, note that the model of RTS subsumes visibly pushdown automata and height deterministic pushdown automata which also support these transformations. Furthermore, the conjoint description of the system and the property (the typed RTS) enables us to consider either regular properties with infinite-state systems, or symmetrically finite-state system with a non-regular properties. Indeed the product between an RTS and a finite-state machine is still an RTS.

Note that the effectiveness of the transformations on weighted RTSs does not imply the decidability of the diagnosability and opacity problems: for example, although computable, the closure of a weighted RTS is not weighted in general. This prevents us from going further in computing its determinization and self-product. A track to alleviate these drawbacks would be to investigate structural conditions on the initial RTS so that the applied transformations preserve the *property of being weighted*.

Another extension of this paper could be to consider k-opacity (*resp.* infinite opacity) which asks whether an attacker might determine that a secrete state *has been crossed* within k observations (*resp.* sometime in the past), see [23]. This extension could probably be solved for RTSs.

[8] Note that the same situation occurs in the first case, if $\mathcal{R}_\lambda \times \mathcal{A}_{fin}$ is not in $\mathcal{C}$wRTS.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Analysis of recursive state machines. In: CAV, *LNCS*, vol. 2102, pp. 207–220 (2001)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC 04, pp. 202–211. ACM (2004)
3. Badouel, E., Bednarczyk, M., Borzyszkowski, A., Caillaud, B., Darondeau, P.: Concurrent secrets. Discrete Event Dynamic Systems **17**, 425–446 (2007)
4. Baldan, P., Chatain, T., Haar, S., König, B.: Unfolding-based diagnosis of systems with an evolving topology. Information and Computation **208**(10), 1169–1192 (2010). DOI 10.1016/j.ic.2009.11.009
5. Bouyer, P., Chevalier, F., D'Souza, D.: Fault diagnosis using timed automata. In: FoSSaCS'05), *LNCS*, vol. 3441, pp. 219–233. Edinburgh, U.K. (2005)
6. Bryans, J., Koutny, M., Mazaré, L., Ryan, P.Y.A.: Opacity generalised to transition systems. Int. J. Inf. Sec. **7**(6), 421–435 (2008)
7. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer (1999)
8. Cassez, F.: The Dark Side of Timed Opacity. In: Proc. of the 3rd International Conference on Information Security and Assurance (ISA'09), *LNCS*, vol. 5576, pp. 21–30. Seoul, Korea (2009)
9. Cassez, F., Dubreil, J., Marchand, H.: Synthesis of opaque systems with static and dynamic masks. Formal Methods in System Design **40**(1), 88–115 (2012)
10. Caucal, D.: Deterministic graph grammars. In: Texts in logics and games 2, pp. 169–250 (2007)
11. Caucal, D., Hassen, S.: Synchronization of grammars. In: E. Hirsch, A. Razborov, A. Semenov, A. Slissenko (eds.) CSR, *LNCS*, vol. 5010, pp. 110–121. Springer (2008)
12. Chédor, S., Jéron, T., Morvan, C.: Test Generation from Recursive Tile Systems. Rapport de recherche RR-8206, INRIA (2013). URL http://hal.inria.fr/hal-00778134
13. ChÃ©dor, S., JÃ©ron, T., Morvan, C.: Test generation from recursive tiles systems. In: A. Brucker, J. Julliand (eds.) 6th International Conference on Tests and Proofs, *LNCS*, vol. 7305, pp. 99–114. Prague, Czech Republic (2012). http://www.irisa.fr/vertecs/Publis /Annee/2012.english.html
14. Courcelle, B.: Handbook of Theoretical Computer Science, chap. Graph rewriting: an algebraic and logic approach. Elsevier (1990)
15. Dubreil, J., JÃ©ron, T., Marchand, H.: Monitoring confidentiality by diagnosis techniques. In: ECC, pp. 2584–2590. Budapest, Hungary (2009)
16. Hélouet, L., Gazagnaire, T., Genest, B.: Diagnosis from scenarios. In: proc. of the 8th Int. Workshop on Discrete Events Systems, WODES'06, pp. 307–312 (2006)
17. HÃ©louet, L., Marchand, H., Genest, B., Gazagnaire, T.: Diagnosis from scenarios, and applications. Discrete Event Dynamic Systems : Theory and Applications (2013)
18. Jéron, T., Marchand, H., Pinchinat, S., Cordier, M.O.: Supervision patterns in discrete event systems diagnosis. In: WODES'06, pp. 262–268 (2006)
19. Jiang, S., Huang, Z., Chandra, V., Kumar, R.: A polynomial algorithm for testing diagnosability of discrete event systems. IEEE Transactions on Automatic Control **46**, 1318–1321 (2000)
20. Kobayashi, K., Hiraishi, K.: Verification of opacity and diagnosability for pushdown systems. Journal of Applied Mathematics (2013)
21. Morvan, C., Pinchinat, S.: Diagnosability of pushdown systems. In: HVC2009, Haifa Verification Conference, *LNCS*, vol. 6405, pp. 21–33. Haifa, Israel (2009)
22. Saboori, A., Hadjicostis, C.: Verification of initial-state opacity in security applications of des. In: 9th International Workshop on Discrete Event Systems, 2008. WODES 2008., pp. 328–333 (2008)
23. Saboori, A., Hadjicostis, C.N.: Verification of infinite-step opacity and complexity considerations. IEEE Trans. Automat. Contr. **57**(5), 1265–1269 (2012)
24. Sampath, M., Sengupta, R., Lafortune, S., Sinaamohideen, K., Teneketzis, D.: Diagnosability of discrete event systems. IEEE Trans. on Automatic Control **40**(9), 1555–1575 (1995)
25. Sampath, M., Sengupta, R., Lafortune, S., Sinaamohideen, K., Teneketzis, D.: Failure diagnosis using discrete event models. IEEE Transactions on Control Systems Technology **4**(2), 105–124 (1996)
26. Tripakis, S.: Fault diagnosis for timed automata. In: W. Damm, E.R. Olderog (eds.) FTRTFT, *LNCS*, vol. 2469, pp. 205–224. Springer (2002)
27. Ushio, T., Onishi, I., Okuda, K.: Fault detection based on petri net models with faulty behaviors. IEEE Int. Conf. on Systems, Man, and Cybernetics. **1**, 113–118 vol.1 (1998)
28. Wu, Y.C., Lafortune, S.: Comparative analysis of related notions of opacity in centralized and coordinated architectures. Discrete Event Dynamical Systems **23**, 307–339 (2013)
29. Yoo, T.S.., Lafortune, S.: Polynomial-time verification of diagnosability of partially-observed discrete event systems. IEEE Trans. on Automatic Control **47**(3), 1491–1495 (2002)