Analysis of partially observed recursive tile systems

Sébastien Chédor, Christophe Morvan, Sophie Pinchinat, Hervé Marchand

Preprints of WODES 2012
October 3-5, 2012
Guadalajara, México

Analysis of partially observed recursive tile systems

Sébastien Chédor * **Christophe Morvan** ** **Sophie Pinchinat** *
Hervé Marchand ***

* *Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France*
** *Université de Paris Est, Marne-La-Vallée, France*
*** *Inria Rennes - Bretagne Atlantique, Campus de Beaulieu, 35042 Rennes, France*
e-mail: sebastien.chedor@inria.fr, christophe.morvan@univ-paris-est.fr, sophie.pinchinat@irisa.fr, herve.marchand@inria.fr

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. We consider recursive tile systems, which are infinite systems generated by a finite collection of finite *tiles*, a simplified variant of deterministic graph grammars. Recursive tile systems are expressive enough to capture classical models of recursive systems, such as the pushdown systems and the recursive state machines. They are infinite-state in general and therefore standard 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 opacity and diagnosability.

Keywords: Diagnosability, Opacity, Discrete event systems, Recursive systems

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* Badouel et al. (2007); Bryans et al. (2008); Cassez (2009) and of *diagnosis* Sampath et al. (1995); Yoo and Lafortune (2002); Jéron et al. (2006); Tripakis (2002); Bouyer et al. (2005).

On the one hand, information flow detection relates to computer security, based on the notion of *opacity*. 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.). 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 runtime. 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* Sampath et al. (1995); Yoo and Lafortune (2002), is tightly coupled with the observation capabilities and has to be established on its own.

Original works in the literature address these issues on finite discrete event systems. Regarding the opacity problem, Bryans et al. (2008); Badouel et al. (2007); Dubreil et al. (2009) show its PSPACE complexity. For infinite systems however, Cassez (2009) shows its undecidability for timed systems. Regarding diagnosis, the diagnosability property is now well understood Sampath et al. (1995); Cassandras and Lafortune (1999); Jéron et al. (2006), 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, Tripakis (2002) investigates timed systems and proves that diagnosability property is equivalent to non-zenoness. Bouyer et al. (2005) expand this work, and exhibit 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. Ushio et al. (1998) consider Petri nets and propose an effective construction of the diagnoser and show the undecidability of the diagnosability property. Baldan et al. (2010) generalize 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, Hélouet et al. (2006) consider a distributed observation of a distributed system modeled by a High Level Message Sequence Chart. Recently, Morvan and Pinchinat (2009) consider pushdown systems and show that the diagnosability property is decidable for a subclass of visibly pushdown systems. As exemplified in the literature, opacity and

diagnosis problems can be solved by using common techniques: the ability to compute an ϵ-*closure* (a projection), to perform a standard *determinization* preserving state properties, to compute a *self-product* or simulate it as a game, to detect *infinite executions*, and to support *reachability analysis*.

In this paper, we aim at solving the two aforementioned problems in the setting of *recursive tile systems* (RTS). RTS are equivalent to *deterministic graph grammars* of Courcelle (1990), and they represent infinite-state systems, in general. They form a natural extension pushdown systems (see *e.g.*, Caucal (2007)) as well as the recursive state machines Alur et al. (2001). Furthermore, they cannot be compared with Petri nets in general, nor with timed automata whose control structure is finite-state.

*Example 1. Let us motivate the use of the RTS 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*.*

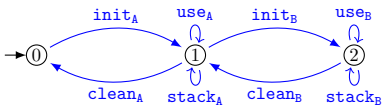

Figure 1. A two-resource system.

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 resource* A *(resp.* B*). The transitions* $\texttt{stack}_\texttt{A}$ *and* $\texttt{stack}_\texttt{B}$ *models the reception of resources* A *and* B*, respectively. Similarly, the transitions* $\texttt{use}_\texttt{A}$ *and* $\texttt{use}_\texttt{B}$ *models the handling of such 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 automaton: whenever transition* $\texttt{init}_\texttt{B}$ *is used, resources* A *that where already in the system remain, and whenever transition* $\texttt{clean}_\texttt{B}$ *is used, there should not be any resources* B *in the system. Its 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, RTS may be employed whenever counters are hierarchically organised, while still preserving interesting decidability results.*

We exhibit the class of *weighted RTS* which capture visibly pushdown systems, as the right notion to address effectiveness of the needed constructions for analysis. Weighted RTS are generated by *weighted deterministic graph grammars* Caucal and Hassen (2008). We identify decidable hypothesis under which the opacity and the diagnosis problems can be solved. The present work strictly extends the results of Morvan and Pinchinat (2009), and relies on a new approach with non-trivial constructions and transformation on graph grammars. In this presentation, we define RTS as objects generated by a finite set of *tiles*, a formalism equivalent to deterministic graph grammars introduced in Chédor et al. (2012). The monitor needed both for analyzing of information flow 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 RTS may produce objects which are not RTS. However, we propose a way to effectively compute the closure so that the result remains an RTS, but which is not weighted in general. Therefore, we consider the class of $\mathcal{C}$*wRTS* composed of the RTS whose closure is weighted. The class $\mathcal{C}$wRTS is decidable. We then show that the typing machine of every RTS in $\mathcal{C}$wRTS is an RTS, which makes possible the reachability analysis needed to verify the opacity property. Also, the self-product techniques useful to check the diagnosability property turns out to be effective for elements of the class $\mathcal{C}$wRTS.

This paper is organized in three main parts: in Section 2, we define DES 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 RTS, based on tiling systems, and we investigate respectively the detection of infinite paths, the computations of the closure, the determinization and the self-product. Finally, in Section 4, we apply the techniques to solve the opacity and diagnosis problems for infinite systems described by RTS in the class $\mathcal{C}$wRTS.

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 $\wp$ is a Σ'-*labelled path* $a_k \in \Sigma'$, for all $k \in I_\wp$.

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 $\mathcal{A}$. Given a finite path $\wp$, its *type* $Ty(\wp)$ is the type of its last state. The notation is extended 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.

The notion of type extends to traces: $t \in \mathcal{L}(\mathcal{A})$, by letting $Ty(t) = \bigcup_{q\in\mathcal{A}(t)} Ty(q)$, 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.

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 $\forall t \in \Sigma^*$, $a \in \Sigma$, $\pi(ta) = \pi(t)a$ if $a \in \Sigma_o$, and $\pi(t)$ 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 it there exists $t \in \Sigma^*$ such that $\mu = \pi(t)a$ and $p \xrightarrow{ta} q$.

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 2. 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.

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

Definition 3. 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 $\forall \mu \in obs(\mathcal{A})$, $Ty'(\mu) = T(\mu)$.

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

Definition 4. 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 Cassandras and Lafortune (1999) 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 DES can be determinized using a well-known powerset construction. We note it $\mathcal{D}(\mathcal{A})$. 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 *diagnosis problem* and the *opacity 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*.

The opacity problem. Opacity was introduced by Bryans et al. (2008) 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 5. 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 $Q_\lambda = \{q_2, q_5\}$.

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.

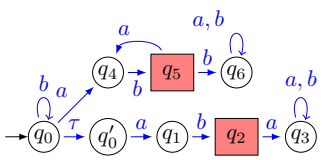

Figure 2. Non-opaque system

The opacity problem is shown reducible to the language universality problem (Dubreil et al. (2009)), hence, for arbitrary DES, it is undecidable (see Bryans et al. (2008) for more detailed results). However, in the finite-state case, 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 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. Following Dubreil et al. (2009), this monitor is based on the typing machine $\mathcal{D}(\mathcal{A}_o)$.

Example 3. Back to Example 2, the corresponding monitor (a part of the typed machine) is depicted in Figure 3. State $\{q_2, q_5\}$ is reachable from the initial state, so that the secret is not opaque.

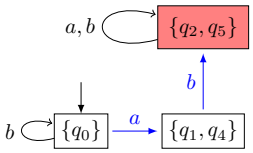

Figure 3. The corresponding Typing machine

The diagnosis problem. The diagnosis problem is a notion introduced by Sampath et al. (1995) which consists of detecting the satisfaction of a persistent property in a partial observation setting (in Sampath et al. (1995), the property encodes the fact that some fault has occurred in the system). As for opacity, 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 $Diag : Obs(\mathcal{A}) \to \{yes, no, ?\}$ 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. (Morvan and Pinchinat (2009)) 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 6. 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 (Morvan and Pinchinat (2009)): 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 DES. The standard procedure consists in contructing the self-product of the closure of the system and in seeking an "equivocal" loop in this structure (Jéron et al. (2006)). We recall the classic notion of self-product.

Definition 7. The *self-product* of a DES $\mathcal{B} = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ is the DES $\mathcal{B}^2 = (\Sigma, Q \times Q, \Lambda, (q_0, q_0), \Delta', Ty')$ where $((p_1, p_2), a, (q_1, q_2)) \in \Delta'$ if and only if $(p_1, a, q_1) \in \Delta$ and $(p_2, a, q_2) \in \Delta$ and $Ty'(p_1, p_2) = Ty(p_1) \cup Ty(p_2)$.

In the next section, we focus on a particular class of infinite-state systems, where the opacity and diagnosis problems can be approached.

3. RECURSIVE TILE SYSTEMS AND THEIR PROPERTIES

In this section, we define the *Recursive Tile Systems* (RTS), a model to define infinite states DES based on the regular graphs of Courcelle (1990). We present some key properties of these systems relative to closure (suppression of internal 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 and unobservable ones,
- Λ is a finite set of *colours* with $\texttt{init} \in \Lambda$.
- $\mathcal{T}$ is a set of tiles $\tau = ((\Sigma, \Lambda), Q, \to, \mathcal{C}, F)$ defined on (Σ, Λ) where
  - $Q \subseteq \mathbb{N}$ is the set of *vertices*,
  - $\to \subseteq Q \times \Sigma \times Q$ is a finite set of *transitions*,
  - $\mathcal{C} \subseteq Q \times \Lambda$ is a finite set of *coloured vertices*,
  - $F \subseteq \mathcal{T} \times 2^{\mathbb{N} \times \mathbb{N}}$, the *frontier*, relates to some *tile*, τ', a partial function (often denoted f') over $\mathbb{N}$, associating to vertices of Q', vertices of Q.
- $\tau_0 \in \mathcal{T}$ is an initial tile (the axiom).

We will provide a formal definition of *tiling* later on, however, the frontier F of a tile τ is used to append other tiles to τ. F identifies tiles and how some of their vertices are merged with those of τ. From a tile τ, with $(q_0, \texttt{init}) \in \mathcal{C}$, one can derive

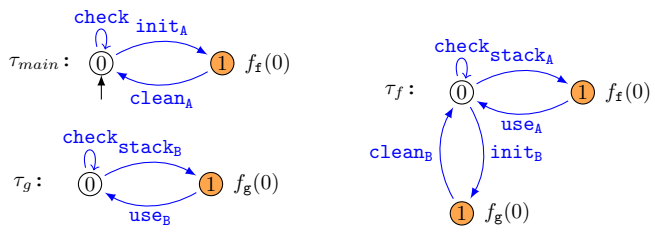

Figure 4. An RTS

a DES $[\tau] = (\Sigma, Q, \Lambda, q_0, \Delta, Ty)$ with $\forall q \in Q, Ty(q) = \{c \mid (q, c) \in \mathcal{C}\}$. 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 $\tau_{\texttt{main}}$ tile ensures that the system is initialised for resource A *and that it is cleaned at the end. The $\tau_{\texttt{f}}$ tile ensures that each resource* A *stacked is used before the* $\texttt{clean}_{\texttt{A}}$*. It also ensures that the urgency mode can be activated at every moment. The $\tau_{\texttt{g}}$ 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 occurres at any time. Formally, the corresponding RTS is defined by: $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_{\texttt{main}})$ with $\Sigma_o = \{\texttt{check}, \texttt{stack}_{\texttt{A}}, \texttt{stack}_{\texttt{B}}, \texttt{use}_{\texttt{A}}, \texttt{use}_{\texttt{B}}\}$, $\Sigma_{uo} = \{\texttt{init}_{\texttt{A}}, \texttt{init}_{\texttt{B}}, \texttt{clean}_{\texttt{A}}, \texttt{clean}_{\texttt{B}}\}$, $\Lambda = \{\texttt{init}\}$, a set of tiles $\mathcal{T} = \{\tau_{\texttt{main}}, \tau_{\texttt{f}}, \tau_{\texttt{g}}\}$, and $\tau_{\texttt{main}}$ the initial tile.*

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

For the frontier, e.g., in the tile $\tau_{\texttt{main}}$, 1 $f_t(0)$ *means that $(\texttt{f}, \{0 \rightarrow 1\})$ belongs to $F_{\texttt{main}}$, i.e. the vertex 0 of $t_{\texttt{f}}$ is associated to the vertex 1 of $\tau_{\texttt{main}}$.*

The semantics of an RTS is formally defined by a DES by a *tiling* operation that appends tiles to another tile (initially, the axiom), inductively defining a DES. Formally, given a set of tiles $\mathcal{T}$ and a tile $\tau = ((\Sigma, \Lambda), Q, \rightarrow, \mathcal{C}, F)$ with F defined on $\mathcal{T}$, the tiling of τ by $\mathcal{T}$, denoted by $\mathcal{T}(\tau)$, is the tile $\tau' = ((\Sigma, \Lambda), Q', \rightarrow', \mathcal{C}', F')$ iteratively defined according to the elements of the frontier F, as follows:

(1) Initially, $Q' = Q$, $\rightarrow' = \rightarrow$, $\mathcal{C}' = \mathcal{C}$ $F' = \emptyset$;
(2) for each pair $(\tau_k, f_k) \in F$, with $\tau_k = ((\Sigma, \Lambda), Q_k, \rightarrow_k, \mathcal{C}_k, F_k) \in \mathcal{T}_k$,
let $\varphi_k : Q_k \rightarrow \mathbb{N}$ be the injection mapping vertices of Q_k to new vertices of Q' with $\varphi_k(n) := f_k(n)$ whenever $n \in dom(f_k)$, $n + \max(Q') + 1$ otherwise, where $\max(Q')$ is the vertex with greatest value in Q'. The tile τ' is then defined by:
  - $Q' = Q' \cup Im(\varphi_k)$,
  - $\rightarrow' = \rightarrow' \cup \{(\varphi_k(n), a, \varphi_k(n')) \mid (n, a, n') \in \rightarrow_k\}$,
  - $\mathcal{C}' = \mathcal{C}' \cup \{(\varphi_k(n), \lambda) \mid (n, \lambda) \in \mathcal{C}_k\}$,
  - $F' = F' \cup \{(t_{k'}, F_{k'}) \mid (\tau_{k'}, f_{k'}) \in F_k\}$
    $F_{k'} = \{(\varphi_k(j), f_{k'}(j)) \mid j \in dom(f_{k'})\}$. The update of F' expresses that the frontier of the new tile τ' is composed from those of the tiles that have been added.

Remark 2. In a tiling, the order chosen to append a copy of the tiles that belong to the frontier is not important. Two different orders would produce isomorphic tiles (up to a renaming of vertices).

Example 5. We illustrate the principle of tiling using the RTS defined in Example 4. Consider that $\tau_{\texttt{main}}$ is the initial tile. Its tiling $\mathcal{T}(\tau_{\texttt{main}})$, is performed as follows: there is a single element in its frontier; we add a copy of $\tau_{\texttt{f}}$ (with new vertices), identifying vertex 1 of $\tau_{\texttt{main}}$ to vertex 0 of $\tau_{\texttt{f}}$. This new tile may be in turn extended by adding a copy of $\tau_{\texttt{f}}$, identifying 3 to 0 of $\tau_{\texttt{f}}$ and 4 to 0 of $\tau_{\texttt{g}}$. We illustrate the resulting tile in Fig. 5 (observe that our definition of $\varphi_{\texttt{comp}}$ induces that some elements of $\mathbb{N}$ are left out).

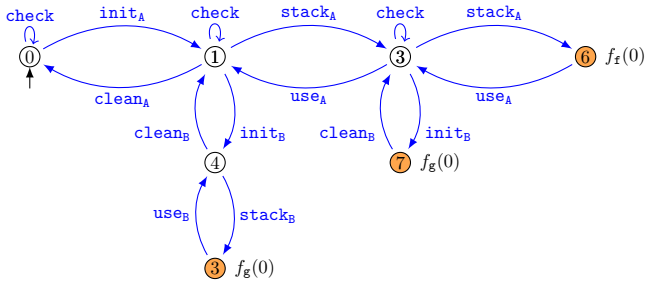

Figure 5. The DES defined by the tile $\mathcal{T}^2(\tau_{\texttt{main}})$

A DES is finally obtained from an RTS as the union of the DES of tiles resulting from the iterated tilings from the axiom. Formally,

Definition 9. Let $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ be an RTS. $\mathcal{R}$ defines a DES $[\mathcal{R}] = (\Sigma, Q_{\mathcal{R}}, \Lambda, Q_{\texttt{init}}, \Delta_{\mathcal{R}}, Ty_{\mathcal{R}})$ given by $\bigcup_k [\mathcal{T}^k(\tau_0)]$.

The infinite union of Definition 9 is valid because, by construction, for all $k \geq 0$: $[\mathcal{T}^k(\tau_0)] \subseteq [\mathcal{T}^{k+1}(\tau_0)]$, where $\subseteq$ is understood as the inclusion of substructures (inclusion of states, transitions and colourings).

For an RTS $\mathcal{R}$ with axiom τ_0, and a state q in $[\mathcal{R}]$, $\ell(q)$ 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 of $\mathcal{R}$, $[v]$ denotes the set of states in $[\mathcal{R}]$ corresponding to v.

Remark 3. The DES obtained from RTS correspond to the equational, or regular graphs of Courcelle (1990) and Caucal (2007) (derived from an axiom using deterministic HR-grammars). In fact RTS have been introduced in Chédor et al. (2012) and aims at a greater simplicity.

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

Proposition 4. (Adapted from Caucal (2007)). 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 $[\mathcal{R}']$ is isomorphic to $[\mathcal{R}]$ 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: $Q_{r_\lambda} = reach_{[\mathcal{R}']}(Q_\lambda, \Sigma')$.

Detecting infinite paths is central for the computation of the closure of regular DES which 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 being 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 $[\mathcal{R}]$ is decidable.

Proposition 5 derives from Lemma 1 in Chédor et al. (2012) (it enables the detection of infinite paths). Slightly adapting it, enable the detection of infinite paths crossing, eventually, only states of a given colour.

Observable behaviour of RTS: Abstracting away internal transitions is important for partial observation questions. The following proposition (from Chédor et al. (2012)) computes the closure of RTS.

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

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

Definition 10. An RTS $\mathcal{R}$ is *weighted for* λ if in its DES $[\mathcal{R}] = (\Sigma, Q_{\mathcal{R}}, \Lambda, Q_{\mathcal{R}_{\texttt{init}}}, \Delta_{\mathcal{R}}, Ty_{\mathcal{R}})$, if $Q_{\mathcal{R}_\lambda}$ is a singleton $\{q_0\}$, and for any $u \in \Sigma^*$ and any states $q, q' \in Q_{\mathcal{R}}$, $q_0 \xrightarrow{u} q$ and $q_0 \xrightarrow{u} q'$ implies $\ell(q) = \ell(q')$ (same level).

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

Note that determining if an RTS is weighted for any given colour is decidable, using an algorithm from Caucal and Hassen (2008).

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 $[\mathcal{R}]$ is deterministic. This is decidable from the set of tiles defining it.

Proposition 7. (Caucal and Hassen (2008)). 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 next theorem provides sufficient conditions under which the determinization of a weighted RTS $\mathcal{R}$ yields a typing machine of $[\mathcal{R}]$.

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 $[\mathcal{R}]$.

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 $[Clo(\mathcal{R})]$ from $Q_{\mathcal{R}_{\texttt{init}}}$ (and thus of $[\mathcal{R}]$) as well as their types. Hence $[\mathcal{D}(Clo(\mathcal{R})]$ is a typing machine of $[\mathcal{R}]$. □

Self-Product. As for the determinization case, the class of RTS is not closed by self-product Morvan and Pinchinat (2009). However, for weighted RTS, the following result holds, provided the colour `init` is defined in the self-product only for pair vertices which both have colour `init`.

Proposition 9. The self-product of an RTS weighted for `init`, is an RTS weighted for `init`.

Proof.(Sketch) To prove this proposition it is sufficient to consider the weighted RTS $\mathcal{R} = ((\Sigma, \Lambda), \mathcal{T}, \tau_0)$ as well as the RTS $\mathcal{R}_2 = ((\Sigma, \Lambda), \mathcal{T}_2, \tau_{02})$ where $\mathcal{T}_2$ is the set of products of tiles defined as follows: Given two tiles $\tau_1 = ((\Sigma, \Lambda), Q_1, \rightarrow_1, \mathcal{C}_1, F_1)$ and $\tau_2 = ((\Sigma, \Lambda), Q_2, \rightarrow_2, \mathcal{C}_2, F_2)$, we denote their product by $\tau_{1\times 2} = ((\Sigma, \Lambda), Q_{1\times 2}, \rightarrow_{1\times 2}, \mathcal{C}_{1\times 2}, F_{1\times 2})$. As for a traditional synchronous product, $Q_{1\times 2} = Q_1 \times Q_2$, for (q_1, q'_1) and $(q_2, q'_2) \in Q_1 \times Q_2$ and $a \in \Sigma$, $((q_1, q'_1), a, (q_2, q'_2)) \in \rightarrow_{1\times 2}$ if $(q_1, a, q_2) \in \rightarrow_1$ and $(q'_1, a, q'_2) \in \rightarrow_2$. For any two pairs, $(\tau_i, f_i) \in F_1$, and $(\tau_j, f_j) \in F_2$, an element of $F_{1\times 2}$ is produced: $(\tau_{i\times j}, f_i \times f_j)$, where the product of functions is the function over idependent product $(f_i \times f_j(a, b) = (f_i(a), f_j(b)))$. Then, 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 $(q_2, c) \in \mathcal{C}_2$. Finally, `init` is given to pairs where each vertex has colour `init`.

Finally, observe that such a computation may be performed for any RTS. But, unless $\mathcal{R}$ is weighted [1], there is no guarantee to have $[\mathcal{R}_2] = [\mathcal{R}]^2$. More precisely, the weighted property ensures that two paths in $[\mathcal{R}]$, with identical labels, reach the same level, hence may be carried out in $[\mathcal{R}_2]$.Thus, a simple induction proves that any path, from `init`, in $[\mathcal{R}]^2$ may be performed in $[\mathcal{R}_2]$ (the converse is always true). □

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 RTS for the opacity problem and diagnosability problems to become decidable, as it is not the case in general.

Proposition 10. The opacity problem is undecidable for RTS.

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 $\Sigma \cup \{\#\}$). The languages $\mathcal{L}\#$ and $\mathcal{L}'\#$ can be represented [2] respectively by $\mathcal{A} = ((\Sigma_\#, \Lambda), \mathcal{T}, \tau_0)$ and $\mathcal{A}' = ((\Sigma_\#, \Lambda), \mathcal{T}', \tau'_0)$ two RTS each having a single vertex labelled by `init` (vertices 0 of tiles τ_0 and τ'_0) and such that λ holds in the accepting states of $[\mathcal{A}]$. $\overline{\lambda}$ holds in all the other states of $[\mathcal{A}]$ and $[\mathcal{A}']$. Let us now consider the RTS $\mathcal{A}$" $= ((\Sigma_\# \cup \{s\}, \Lambda), \mathcal{T} \cup \mathcal{T}' \cup \{\tau''_0\}, \tau''_0)$ such that $\tau''_0 = ((\Sigma_\# \cup \{s\}, \Lambda), \{0, 1\}, \{0 \xrightarrow{s} 1\}, \{(0, \texttt{init})\}, \{(\tau_0, 0, 0), (\tau_0, 1, 0)\})$. In the RTS $\mathcal{A}$", 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. $\mathcal{A}$" and $\Sigma_\#$ ($s \in \Sigma_{uo}$). □

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

[2] This simple construction of a tiling system with an initial vertex is presented in Caucal (2007), Section 5 in the contetxt of deterministic graph grammar.

We can mimic the classic desicion procedure to decide 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 RTS whose closure [3] is weighted (recall it is already an RTS).

Theorem 1. The opacity problem is decidable over the class $\mathcal{C}$wRTS.

Proof. We give the decision procedure: let $\mathcal{R} \in \mathcal{C}$wRTS, as $Clo(\mathcal{R})$ is weighted, we apply Proposition 8 and obtain an RTS $\mathcal{D}(Clo(\mathcal{R}))$ such that $[\mathcal{D}(Clo(\mathcal{R}))]$ is a typing machine. It is then sufficient to solve the reachability of $\{\lambda\}$-states in $[\mathcal{D}(Clo(\mathcal{R}))]$, which is decidable by Proposition 4. □

Turning to diagnosabilty, we have the following.

Proposition 11. Diagnosability problem for RTS is undecidable.

This result is a consequence of Morvan and Pinchinat (2009) since visibly pushdown systems are a strict subclass of weighted RTS. However, we have a result similar to Theorem 1.

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

Proof. Let $\mathcal{R} \in \mathcal{C}$wRTS, since $Clo(\mathcal{R})$ is weighted, by Proposition 9, their exists an RTS $\mathcal{R}_2$ such that $[\mathcal{R}_2] = [Clo(\mathcal{R})]^2$ (self-product of $[Clo(\mathcal{R})]$) (Proposition 9). Now, by Lemma 1, non-diagnosability is equivalent to finding an infinite path of $\{\lambda, \overline{\lambda}\}$-typed states in $[Clo(\mathcal{R})]^2$. This can be decided according to Proposition 5. □

Note that the RTS $\mathcal{D}(Clo(\mathcal{R}))$ is a valuable object. Indeed, whenever the system is not opaque (following Dubreil et al. (2009) for finite-state systems), $\mathcal{D}(Clo(\mathcal{R}))$ can be exploited to monitor the system $[\mathcal{R}]$ and detect effective information flow. Similarly, when diagnosability holds, the RTS $\mathcal{D}(Clo(\mathcal{R}))$ provides the desired diagnoser.

5. CONCLUSION

The present paper demonstrates a way to extend detection of information flow and property diagnosis for infinite DES. We could overcome undecidability by exhibiting the subclass of weighted RTS for which determinization, self-product and detection of infinite path are effective. Note that the model 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 RTS 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*.

Finally, we believe that connected topics such as prognosis and supervisory control may also be approachable for RTS.

[3] according to Proposition 6, see also Chédor et al. (2012).

REFERENCES

Alur, R., Etessami, K., and Yannakakis, M. (2001). Analysis of recursive state machines. In *CAV*, volume 2102 of *LNCS*, 207–220.

Badouel, E., Bednarczyk, M., Borzyszkowski, A., Caillaud, B., and Darondeau, P. (2007). Concurrent secrets. *Discrete Event Dynamic Systems*, 17, 425–446.

Baldan, P., Chatain, T., Haar, S., and König, B. (2010). Unfolding-based diagnosis of systems with an evolving topology. *Information and Computation*, 208(10), 1169–1192. doi:10.1016/j.ic.2009.11.009.

Bouyer, P., Chevalier, F., and D'Souza, D. (2005). Fault diagnosis using timed automata. In *FoSSaCS'05)*, volume 3441 of *LNCS*, 219–233. Edinburgh, U.K.

Bryans, J., Koutny, M., Mazaré, L., and Ryan, P.Y.A. (2008). Opacity generalised to transition systems. *Int. J. Inf. Sec.*, 7(6), 421–435.

Cassandras, C.G. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Springer.

Cassez, F. (2009). The Dark Side of Timed Opacity. In *Proc. of the 3rd International Conference on Information Security and Assurance (ISA'09)*, volume 5576 of *LNCS*, 21–30. Seoul, Korea.

Caucal, D. (2007). Deterministic graph grammars. In *Texts in logics and games 2*, 169–250.

Caucal, D. and Hassen, S. (2008). Synchronization of grammars. In *CSR*, volume 5010 of *LNCS*, 110–121.

Chédor, S., Jéron, T., and Morvan, C. (2012). Test generation from recursive tiles systems. In A. Brucker and J. Julliand (eds.), *6th International Conference on Tests and Proofs*, volume 7305 of *LNCS*, 99–114. Prague, Czech Republic.

Courcelle, B. (1990). *Handbook of Theoretical Computer Science*, chapter Graph rewriting: an algebraic and logic approach. Elsevier.

Dubreil, J., Jéron, T., and Marchand, H. (2009). Monitoring confidentiality by diagnosis techniques. In *ECC*, 2584–2590. Budapest, Hungary.

Hélouet, L., Gazagnaire, T., and Genest, B. (2006). Diagnosis from scenarios. In *proc. of the 8th Int. Workshop on Discrete Events Systems, WODES'06*, 307–312.

Jéron, T., Marchand, H., Pinchinat, S., and Cordier, M.O. (2006). Supervision patterns in discrete event systems diagnosis. In *WODES'06*, 262–268.

Morvan, C. and Pinchinat, S. (2009). Diagnosability of pushdown systems. In *HVC2009, Haifa Verification Conference*, volume 6405 of *LNCS*, 21–33. Haifa, Israel.

Sampath, M., Sengupta, R., Lafortune, S., Sinaamohideen, K., and Teneketzis, D. (1995). Diagnosability of discrete event systems. *IEEE Trans. on Automatic Control*, 40(9), 1555–1575.

Tripakis, S. (2002). Fault diagnosis for timed automata. In W. Damm and E.R. Olderog (eds.), *FTRTFT*, volume 2469 of *LNCS*, 205–224. Springer.

Ushio, T., Onishi, I., and Okuda, K. (1998). Fault detection based on petri net models with faulty behaviors. *IEEE Int. Conf. on Systems, Man, and Cybernetics.*, 1, 113–118 vol.1.

Yoo, T.S.. and Lafortune, S. (2002). Polynomial-time verification of diagnosability of partially-observed discrete event systems. *IEEE Trans. on Automatic Control*, 47(3), 1491–1495.