Verification of Embedded Reactive Fiffo Systems

Frédéric Herbreteau[1,2], Franck Cassez[1], Alain Finkel[3], Olivier Roux[1] and Grégoire Sutre[2,4]

[1] IRCCyN (CNRS UMR 6597), 1, rue de la Noe, 44321 Nantes cedex 3, France {Franck.Cassez, Olivier.Roux}@irccyn.ec-nantes.fr

[2] LaBRI (CNRS UMR 5800), 351, cours de la Libération, 33405 Talence Cedex, France {Frederic.Herbreteau, Gregoire.Sutre}@labri.fr

[3] LSV (CNRS UMR 8643), 61, avenue du Président. Wilson, 94235 Cachan cedex, France finkel@lsv.ens-cachan.fr

[4] ERL, 253 Cory Hall, University of California, Berkeley, CA 94720, USA

Abstract. Reactive Fiffo Systems (RFS) are used to model reactive systems which are able to memorize the events that cannot be processed when they occur. In this paper we investigate the decidability of verification problems for Embedded RFS which are RFS running under some environmental constraints. We show that almost all the usual verification problems are undecidable for the class of Periodically Embedded RFS with two memorizing events, whereas they become decidable for Regularly Embedded RFS with a single memorizing event. We then focus on Embedded Lossy RFS and we show in particular that for Regularly Embedded Lossy RFS the set of predecessors $Pred^*$ is upward closed and effectively computable.

Keywords: Reactive Fiffo Systems, Embedded Systems, Verification, Real-Time Systems, Infinite-State Systems, Decidability.

1 Introduction

Context. Model-checking has become a very popular verification method, since it is fully automatic for finite state systems and it has been applied successfully in particular for VLSI circuits [7]. However, since most systems are intrinsically infinite state, there is a need to design an efficient verification methodology for infinite state systems.

We focus in this paper on *Embedded Reactive Fiffo Systems (Embedded RFS)*, a model for embedded (asynchronous) reactive systems with event memorization, close to SDL [9]. Any Embedded RFS is actually a *closed system*, obtained by synchronization of an environment defining the sequences of input events with a *Reactive Fiffo System* modeling the reactions to these events.

Reactive Fiffo Systems (RFS) were introduced in [8, 22] (and called *Reactive Fiffo Automata* there) to model (asynchronous) reactive systems with event memorizations that are specified in the Electre [8] reactive language. A major feature of this language is that it is possible to store occurrences of events in order to process them later. The way the events are processed is called the *Fiffo* order for *First In First Fireable Out* i.e. we take into account the stored events as soon as possible and in case of conflict the oldest stored occurrence is processed. The number of stored occurrences in the Fiffo queue is theoretically unbounded. Consequently the behavioral model of an Electre program is a RFS which has an infinite number of states [22].

Related work. In a previous work [22], we analyzed RFS *with an implicit environment* of the form Σ^*, that is an environment generating every sequence of events. We proved the following results for RFS: (1) the set $Post^*$ of the reachable states of a RFS is recognizable and effectively computable; (2) the linear temporal logic LTL without the next operator is decidable for RFS.

These results were surprising since RFS are close to *Communicating Finite State Machines (CFSMs)* or *Fifo Automata*, and it is well known that this class has the power of Turing Machines [6, 13]. This difference is mainly due to the fact that RFS (with an implicit environment) are open systems (whereas CFSMs are closed) on the one hand, and from the looping form of the memorizing transitions (which can thus be executed any number of times) which entails that the state space of RFS are downward closed and thus recognizable.

Our Contribution. The purpose of this paper is to analyze RFS *with an explicit environment*, the so-called *Embedded RFS*, which are RFS that run under some *environmental constraints*. Indeed, without an explicit environment, we are able to compute the set of reachable states $Post^*$ [22], but many of these states would not be reachable in a "realistic" environment that would constrain the system.

Embedded RFS are naturally defined as the synchronization between a RFS and an environment given as a labeled transition system. We in particular consider *regular* and *periodic* environments, where the languages of event sequences are regular or periodic respectively. These classes of environments are particularly relevant in the area of real-time systems as for instance, many applications are strongly periodic (e.g. *scheduling*) or mainly composed of *regular* cyclic processes (control of nuclear plants, avionic systems, ...).

The main results of the paper are two fold; we first show that:

(1) Periodically Embedded RFS have the power of Turing machines: most verification problems are undecidable for this class.

Since Embedded RFS are too powerful, even in the "simple" case of periodic environments, we then try to lower their expressiveness in order to obtain decidability results. We show that:

(2) When there is at most one memorizing event, most verification problems are decidable for Regularly Embedded RFS.
(3) Embedded Lossy RFS are well-structured transition systems (WSTS) with effective pred-basis (as defined in [14]), provided that the environment is a

WSTS with effective pred-basis itself. Hence the covering problem is decidable, and we can model-check safety properties on these systems.

(4) Moreover, we show that *Pred**, the set of predecessors, is effectively computable for Regularly Embedded Lossy RFS. Consequently model-checking the existential fragment of CTL is decidable for this class.

As emphasized in the conclusion these results give a foundation for the verification of Embedded RFS.

Outline of the paper. The next section recalls some basics and introduces the model of *Embedded RFS*. In Section 3 we prove one of our main results: Periodically Embedded RFS have the power of Turing machines. In Section 4, we focus on a *lossy* version of RFS and we prove our second main result: Embedded Lossy RFS are well structured transition systems. Finally in Section 5 we draw some conclusions and give some hints for future work.

2 Embedded Reactive Fiffo Systems

2.1 Preliminaries

Ordering and quasi-ordering. For a given set X, a binary relation $\leq$ is a *quasi-ordering* if it is transitive and reflexive. The *upward-closure* of $Y \subseteq X$ is $\uparrow\!Y = \{x' \in X \mid \exists x \in Y,\ x \leq x'\}$. A finite set X is a *finite basis* for an upward-closed set X' if $\uparrow\!X = X'$. A *well quasi-ordering (wqo)* $(X, \leq)$ is a quasi-ordering $\leq$ such that in every infinite sequence: $x_1, x_2, \ldots, x_i, \ldots$ of elements from X, there exist two integers $i < j$ such that $x_i \leq x_j$. Every pair $(X, \leq)$ such that X is finite is *de facto* a wqo. Furthermore, by definition, a well quasi-ordering $(X, \leq)$ has a finite number of minimal elements (it is well-founded).

Considering a finite alphabet $\Sigma = \{a_1, \ldots, a_n\}$, a finite (resp. infinite) word $u \in \Sigma^*$ is a finite (resp. infinite) sequence of symbols from Σ; u_i denotes the i^{th} letter of u and ε is the empty word. The *shuffle* of two finite words u and v is the set defined by: $u \sqcup\!\sqcup\, v = \{u_1v_1 \ldots u_nv_n \mid u_1, \ldots, v_n \in \Sigma^* \text{ and } u = u_1 \ldots u_n,\ v = v_1 \ldots v_n\}$. The *sub-word ordering* $\preccurlyeq$ is defined by: $\forall u, v \in \Sigma^*,\ u \preccurlyeq v$ iff u can be obtained from v by removing some letters. The *Parikh mapping* [21] Ψ defines the vector $\Psi(u) = (k_1 \quad \ldots \quad k_n)$ where k_i is the number of occurrences of a_i in the finite word u.

Labeled Transition Systems. A *labeled transition system* is a 4-tuple $S = (Q, q_0, L, \rightarrow)$ where Q is the set of states, $q_0 \in Q$ is the initial state, L is the set of labels and $\rightarrow \subseteq Q \times L \times Q$ is the transition relation. A *run* of S is a finite or infinite sequence of transitions $q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} \cdots \xrightarrow{l_n} q_{n+1} \cdots$. We denote $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$. A state $q' \in Q$ is *reachable* in S from state $q \in Q$ iff $q \rightarrow^* q'$. *Post*$^*(S)$ is the set of reachable states in S from q_0. For a given state $q \in Q$ and a given label $l \in L$, $\mathit{Pred}_l(q) = \{q' \in Q \mid q' \xrightarrow{l} q\}$ is the set of the predecessors of q by l. Let $in(q)$ (resp. $out(q)$) denotes the sets of *incoming labels* (resp. *outgoing labels*) from $q \in Q$. Finally, the sets of the *predecessors* of q is $\mathit{Pred} = \bigcup_{l \in in(q)} \mathit{Pred}_l(q)$ and *Pred** denotes its transitive and reflexive closure.

Problems of interest. For a given labeled transition system $S = (Q, q_0, L, \rightarrow)$ and a wqo $(Q, \leq)$, we can define the following problems :

- The *boundedness problem*: Is $Post^*(S)$ finite ?
- The *covering problem*: For any states $q \in Post^*(S)$ and $q' \in Q$, is there any $q'' \in Q$ such that $q' \leq q''$ and q'' is reachable from q ?
- The *termination problem*: Is every run of S finite ?
- The *(resp. effective) recognizability problem*: Is there a (resp. computable) finite representation for $Post^*(S)$?
- The *recurrent locality problem*: Is there a run of S that visits a given $q \in Q$ infinitely often ?
- The *LTL (resp. CTL, EG, EF) model-checking problem*: For a given LTL (resp. CTL, EG, EF) formula[1] ϕ, does every run of S satisfy ϕ ?

2.2 Reactive Fiffo Automata

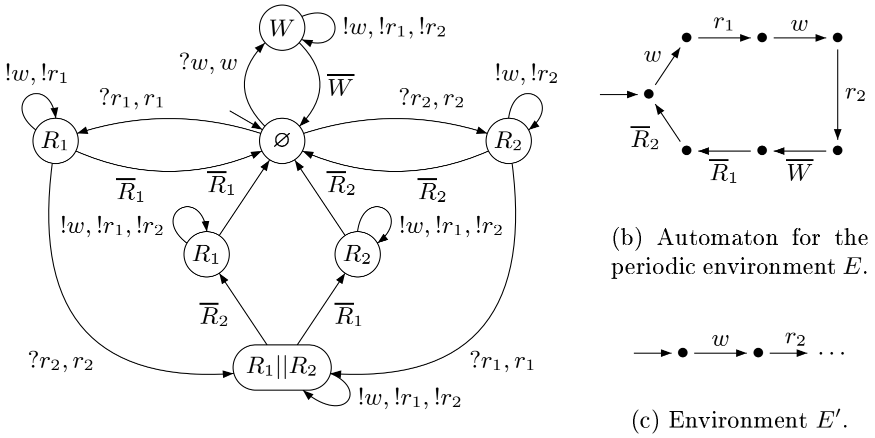
Reactive systems are usually composed of a set of tasks which are activated, preempted or ended when some events occur. Reactive Fiffo Automata (RFA) [8] aim at modeling such *asynchronous* systems, thus they follow two main guidelines: *memorization* and *instantaneous reaction*. Indeed, the asynchronous assumption implies that the tasks have a non null duration, thus they may prevent the processing of events, which may thus need to be memorized. For reactive systems, we have chosen to batch process memorized events *as soon as possible* and in case two memorized events can be processed, to give the priority to the older one. This guarantees fairness among the memorized occurrences. Moreover, batch processing has priority against processing of new incoming occurrences. This type of processing is called *Fiffo*[2]. Hence, RFA can serve as a semantic model for Electre programs [8], as well as for SDL [9] specifications, and for any formalism for reactive systems with memorization.

Figure 1(a) depicts a RFA modeling a readers/writers mutual exclusion protocol with two readers R_1 and R_2 and one writer W. In each location the running tasks are given ($\varnothing$ means no task is running, R_1 that reader 1 is reading and so on.) The events $\overline{W}, \overline{R}_1, \overline{R}_2$ correspond to the end of a writing or reading task. A transition of the form $\varnothing \xrightarrow{r_1} R_1$ means that event r_1 can be processed in location $\varnothing$ and leads to a state where R_1 is running; $\varnothing \xrightarrow{?r_1} R_1$ means that processing a memorized occurrence of event r_1 leads to R_1 and finally $R_1 \xrightarrow{!w} R_1$ means that if w occurs when the system is in location R_1 it has to be stored in the queue. (Notice that two states correspond to task R_1 running but they are not bisimilar.)

Definition 1 (Reactive Fiffo Automaton (RFA)). *A* Reactive Fiffo Automaton (RFA) *is a 4-tuple* $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ *where:* Q_R *is a finite set of*

[1] Recall that the EF (resp. EG) fragment of CTL uses predicates, boolean operators, the one-step next and the EF (resp. EG) operators.

[2] First In First Fireable Out.



(a) A RFA for a 2 readers/1 writer problem.

(b) Automaton for the periodic environment E.

(c) Environment E'.

Fig. 1. Some examples of RFA and environments.

locations *with the distinguished* initial location q_R^0, $A_R = \Sigma \cup (\{!, ?\} \times \Sigma_M)$ *is the finite set of* actions *(labels) where* Σ *is the finite set of* events, $\Sigma_M \subseteq \Sigma$ *is the finite set of* memorized events *and* $\Sigma_F = \Sigma \setminus \Sigma_M$ *is the set of* fleeting events *and finally,* $\rightarrow_R \subseteq Q_R \times A_R \times Q_R$ *is the deterministic transition relation such that* $\forall q \in Q_R$, $\forall e \in \Sigma_M$*:*

1. *either* $q \xrightarrow{!e}_R q$,
2. *or* $\exists q' \in Q_R$ *such that* $q \xrightarrow{e}_R q'$ *and* $q \xrightarrow{?e}_R q'$. □

The memorizing transitions are loops: the processing of the event is delayed and the system does not allow any state change. Notice that the memorizing capability does not apply to all the events of the RFA. Indeed, the events' set Σ is divided in two parts: the memorized events Σ_M and the fleeting events Σ_F, with $\Sigma_M \cap \Sigma_F = \emptyset$. One could also consider static priority between events, like "if e_1 is in the Fiffo list, batch process e_1 even if some other e_2 is before e_1 in the list and can be processed." This can already be specified by RFA, and thus our work includes this case.

Following points (1) and (2) the RFA never *looses* a memorized event occurrence, and as entailed by (2), the processing (e) and the batch processing ($?e$) of an event have the same effect w.r.t. to the state's change they bring about.

Definition 2 (Reactive Fiffo System (RFS)). *A* Reactive Fiffo System (RFS)*, giving the semantics of the RFA* $R = (Q_R, q_R^0, A_R, \rightarrow_R)$*, is the (infinite) labeled transition system* $S = (Q_S, q_S^0, A_S, \rightarrow_S)$ *defined by:* $Q_S = Q_R \times \Sigma_M^*$ *is the set of states where one distinguishes the initial state* $q_S^0 = (q_R^0, \varepsilon)$*,* $A_S = A_R$ *is the finite set of actions (labels) and the transition relation* $\rightarrow_S$ *is the smallest subset of* $Q_S \times A_S \times Q_S$ *such that:*

1. $(q, w) \xrightarrow{!e}_S (q, w.e)$ *if* $q \xrightarrow{!e}_R q$ *and* $w \in \left(\Sigma_q^!\right)^*$ *(memorizing)*
2. $(q, w) \xrightarrow{e}_S (q', w)$ *if* $q \xrightarrow{e}_R q'$ *and* $w \in \left(\Sigma_q^!\right)^*$ *(processing)*
3. $(q, w_1.e.w_2) \xrightarrow{?e}_S (q', w_1.w_2)$ *if* $q \xrightarrow{?e}_R q'$ *and* $w_1 \in \left(\Sigma_q^!\right)^*$ *(batch processing)*
4. $(q, w) \xrightarrow{e}_S (q, w)$ *if* $e \notin (out(q) \cup \Sigma_M)$ *and* $w \in \left(\Sigma_q^!\right)^*$

with $\Sigma_q^! = \left\{e \in \Sigma_M \mid q \xrightarrow{!e}_R q\right\}$. □

A state (q, w) is *stable* if $w \in \left(\Sigma_q^!\right)^*$ otherwise it is *unstable* (in the latter case an event is to be batch processed). Priority is given to the batch processings since the other transitions are only enabled in stable states. Moreover the batch processed occurrence (if any) is the oldest one as condition $w_1 \in \left(\Sigma_q^!\right)^*$ of (3) above implies. Finally, by point (4), the RFS is complete and thus reacts to every event (whereas the RFA may not be complete w.r.t fleeting events).

Example 1. The sequence below depicts a possible run of the readers/writers RFA in Figure 1(a).

$$(\varnothing, \varepsilon) \xrightarrow{w}_S (W, \varepsilon) \xrightarrow{!r1}_S (W, r_1) \xrightarrow{!w}_S (W, r_1 w) \xrightarrow{!r_2}_S (W, r_1 w r_2) \xrightarrow{\overline{W}}_S$$
$$(\varnothing, r_1 w r_2) \xrightarrow{?r_1}_S (R_1, w r_2) \xrightarrow{?r_2}_S (R_1 || R_2, w) \xrightarrow{\overline{R_1}}_S (R_2, w) \xrightarrow{\overline{R_2}}_S (\varnothing, w) \ldots$$

The transition $(R_1, w r_2) \xrightarrow{?r_2}_S (R_1 || R_2, w)$ clearly depicts the Fiffo (First In First Fireable Out) policy. Indeed, r_2 is batch processed because (i) the configuration $(R_1, w r_2)$ is unstable, and (ii) the events are batch processed as soon as possible and with priority to the oldest one.

2.3 Embedded Reactive Fiffo Systems

An *environment* $E = (Q_E, q_E^0, A_E, \rightarrow_E)$ for a RFS $S = (Q_S, q_S^0, A_S, \rightarrow_S)$ is a labeled transition system, with $A_E = \Sigma$, defining the input words of the system (i.e. sequences of events in Σ^*). The *Embedded Reactive Fiffo System* $S \parallel E$ is obtained by synchronization of S and E:

Definition 3 (Embedded Reactive Fiffo Systems). *An* Embedded Reactive Fiffo System (Embedded RFS) *is a (infinite) labeled transition system* $S \parallel E = (Q_{S\parallel E}, q_{S\parallel E}^0, A_{S\parallel E}, \rightarrow_{S\parallel E})$ *defined by:* $Q_{S\parallel E} = Q_R \times (\Sigma_M)^* \times Q_E$ *is the set of configurations and* $q_{S\parallel E}^0 = \langle q_R^0, \varepsilon, q_E^0 \rangle$ *is the initial configuration,* $A_{S\parallel E} = A_S$ *is the set of actions, and finally the transition relation* $\rightarrow_{S\parallel E}$ *is the smallest subset of* $Q_{S\parallel E} \times A_{S\parallel E} \times Q_{S\parallel E}$ *such that:*

1. $\langle q_R, w, q_E \rangle \xrightarrow{!e}_{S\parallel E} \langle q_R, we, q_E' \rangle$ *if* $(q_R, w) \xrightarrow{!e}_S (q_R, we)$ *and* $q_E \xrightarrow{e}_E q_E'$,
2. $\langle q_R, w, q_E \rangle \xrightarrow{e}_{S\parallel E} \langle q_R', w, q_E' \rangle$ *if* $(q_R, w) \xrightarrow{e}_S (q_R', w)$ *and* $q_E \xrightarrow{e}_E q_E'$,
3. $\langle q_R, w_1 e w_2, q_E \rangle \xrightarrow{?e}_{S\parallel E} \langle q_R', w_1 w_2, q_E \rangle$ *if* $(q_R, w_1 e w_2) \xrightarrow{?e}_S (q_R', w_1 w_2)$.

Notice that by definition 2, rule 3 has priority over rules 1 and 2. □

The runs of $S \parallel E$ are called the *constrained runs* of the system. The notions of stability and unstability naturally extend to Embedded RFS. In the sequel, Embedded RFS(n) denotes the class of Embedded RFS with n memorized events ($|\Sigma_M| = n$).

Considering the environment E in Figure 1(b), the execution of Example 1 is the prefix of a constrained run of the RFA in Figure 1(a). But this execution is discarded by E' in Figure 1(c) since the transition $(W, \varepsilon) \xrightarrow{!r_1}_S (W, r_1)$ is disabled by r_2 in E'.

Observe that, provided the set of events Σ is non empty, a RFS has no deadlock state, since in every state it is able to react to an event occurrence, either by memorizing it, by losing it (if this event is fleeting, by (4) in Definition 2), or by processing it. Hence any Embedded RFS with a non-terminating environment does not terminate. In the rest of the paper, we won't discuss the termination problem anymore, and by "problems of interest" we mean all the problems defined in Section 2.1 except the termination problem.

3 The Power of Embedded RFS

In this section, we first prove that RFS having two memorized events and with a periodic environment are deterministic counter automata. It follows that all the problems of interest are undecidable. Then, we focus on RFS with at most one memorized event constrained by regular environments and prove that their reachability set is effectively recognizable.

3.1 Undecidability of Periodically Embedded RFS(2)

A *Periodically Embedded RFS* is an Embedded RFS the environment of which recognizes a periodic sequence of events u^* where u is a finite word.

Definition 4 (n-counter automata [20]). *A* n-counter automaton C *is defined by the tuple* $(Q_C, q_C^0, \{c_1, \dots, c_n\}, \rightarrow_C)$ *where:* Q_C *is a finite set of states with initial state* q_C^0, $\{c_1, \dots, c_n\}$ *is a finite set of* counters *with values in* $\mathbb{N}$ *and* $\rightarrow_C \subseteq Q_C \times (\{c_1, \dots, c_n\} \times \{{+}{+}, {-}{-}, {=}0?\}) \times Q_C$ *is the transition relation where* c_{k++} *denotes the increasing of* c_k *whereas* c_{k--} *is its decreasing and* $c_{k=0?}$ *represents the zero testing operation. A* n*-counter automaton is* deterministic *provided for each state* q *there is:*

- *either an increasing transition:* $q \xrightarrow{c_{k++}}_C q'$,
- *or a decreasing and a zero testing transition dealing with the same counter:* $q \xrightarrow{c_{k--}}_C q''$ *and* $q \xrightarrow{c_{k=0?}}_C q'$. □

The semantics of a n-counter automaton C is the labeled transition system $S = (Q \times \mathbb{N}^2, \langle q_0, 0, \dots, 0\rangle, \rightarrow)$ where $\rightarrow$ is the smallest subset of $(Q \times \mathbb{N}^2) \times (Q \times \mathbb{N}^2)$ given by $(m_1, \dots, m_n \in \mathbb{N})$:

1. $\langle q, m_1, \dots, m_n\rangle \rightarrow \langle q', m_1 + 1, \dots, m_n\rangle$ if $q \xrightarrow{c_{1++}}_C q'$,

2. $\langle q, 0, m_2, \ldots, m_n\rangle \rightarrow \langle q', 0, m_2, \ldots, m_n\rangle$ if $q \xrightarrow{c_1=0?}_C q'$,
3. $\langle q, m_1, \ldots, m_2\rangle \rightarrow \langle q', m_1 - 1, \ldots, m_2\rangle$ if $q \xrightarrow{c_1--}_C q'$ and $m_1 > 0$.

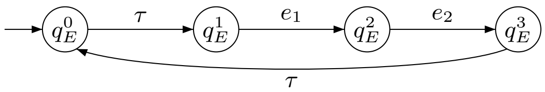
and the symmetric transitions for $c_2, \ldots, c_n$. A *run* of a n-counter automaton C is a run in S.

In the sequel we say that a transition system S *simulates* a transition system S' iff $Post^*(S') = f\big(Post^*(S)\big)$ with f a simple mapping (e.g. projection on a subset of the set of states in Q so that $Post^*(S')$ can be computed easily from $Post^*(S)$).

Theorem 1 ([20]). *For any Turing machine T, there is a deterministic 2-counter automaton that simulates T.* □

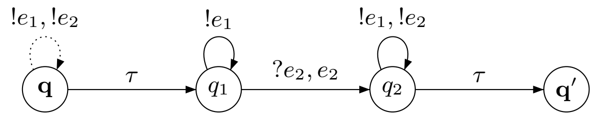
We now prove how to simulate a Deterministic 2-counter automaton by a Periodically Embedded RFS(2). Assume C is a deterministic 2-counter automaton with counters c_1 and c_2. Let $S \parallel E$ be the Periodically Embedded RFS built in the following way. The two counters c_1 and c_2 are respectively defined by the two memorized events e_1 and e_2 so that the number of memorized occurrences of e_1 (resp. e_2) is the value of c_1 (resp. c_2).

Every transition in C is translated in a RFA structure (a widget). Our construction runs as follows: to simulate one step of the 2-counter automaton, we constrain the RFA widgets with a word u. This way, by processing u^* we can simulate an execution of the counter machine. The key point here is to find a word u to be processed by a RFA widget in order to simulate a step. The environment we take here recognizes $(\tau e_1 e_2 \tau)^*$ where τ is a fleeting event and e_1 and e_2 are memorized events.

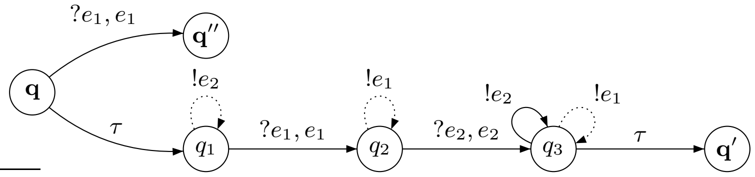


The widgets for the two types of transitions using counter c_1 in a 2-counter automaton are given by[3]:

1. widget for $q \xrightarrow{c_1++}_C q'$:



2. widget for $q \xrightarrow{c_1=0?}_C q'$ and $q \xrightarrow{c_1--}_C q''$:



[3] All the dotted loops are necessary to match Definition 1 point (1), but they are never executed during the simulation of C.

The fleeting event τ is introduced to allow zero testing (see widget (2)) and the two τ transitions: $q_2 \xrightarrow{\tau} q'$ (on widget (1)) and $q_3 \xrightarrow{\tau} q'$ (on widget (2)) are needed to put together the widgets translating the transitions of C.

Since increasing c_k is modeled by memorizing e_k, the environment must produce e_1 and e_2 sequentially since (i) it is periodic, and (ii) widget (1) needs to apply to both counters.

Thus, on widget (1), from q_1, either there is one memorized e_2: one of them is batch processed (leading from q_1 to q_2) and e_1 then e_2 are memorized in q_2, or e_1 is memorized in q_1 and e_2 is processed, leading to q_2. In both cases, the number of memorized e_1 has been increased by one, whereas the number of stored e_2 is invariant from q to q'. In widget (2), e_1 is immediately processed from q_1 to q_2 since q_1 is reachable iff there is no memorized e_1. Then, from q_2, e_2 is applied the same treatment as in widget (1).

It follows that the 2-counter automaton C can move from $\langle q, m_1, m_2 \rangle$ to $\langle q', m_1 + 1, m_2 \rangle$ iff widget (1) can move from $\langle q, w, q_E^0 \rangle$, with $\Psi(w) = (m_1 \quad m_2)$, to $\langle q', w', q_E^0 \rangle$, with $\Psi(w') = (m_1 + 1 \quad m_2)$ i.e. by processing exactly the word $\tau e_1 e_2 \tau$. The same remark applies for the other widget and the decreasing transition.

Theorem 2. *For every deterministic* 2*-counter automaton, there exists a Periodically Embedded RFS(2) that simulates C.* □

As a consequence, all the problems of interest are undecidable for Embedded RFS with at least two memorizing events and a periodic environment.

3.2 Regularly Embedded RFS(1) are Effectively Recognizable

In the previous section, we have seen that all the problems of interest are undecidable for Periodically Embedded RFS(2). This class of Embedded RFS includes the huge majority of practical systems. We now focus on Embedded RFS with only one memorizing event e_m in order to complete our study. Particularly, we show that this restriction heavily decreases the power of Embedded RFS as this is the case for Fifo automata. Indeed, note that when there is a unique memorizing event, the Fiffo queue of a RFS actually behaves as a counter, since only the number of events in the queue is relevant. Memorization (batch processing) of an occurrence of e_m thus corresponds to a increment (resp. decrement) of the counter. The processing of a fleeting event in a locality where e_m can be (batch) processed is possible iff the queue is empty, that is iff the counter is equal to zero.

Following the previous intuitive ideas, any RFS(1) can be seen as a one-counter automaton. Recall that one-counter automata are effectively closed under synchronization with finite automata. Hence we get that any Embedded RFS(1) can be seen as a one-counter automaton. Since one-counter automata form a subclass of pushdown automata which have effectively recognizable reachability sets and also decidable LTL and CTL model-checking [4, 15], we obtain the following result:

Theorem 3. *Regularly Embedded RFS with at most one memorizing event have effectively recognizable reachability sets. Moreover LTL and CTL model-checking are decidable for this class.* □

As a consequence, all the problems of interest are decidable for Embedded RFS with at most one memorizing event and a regular environment.

4 Embedded Lossy Reactive Fiffo Systems

In this section, we consider unreliable Embedded RFS that may loose some memorized events non deterministically. Embedded Reactive Fiffo Systems are equipped with the lossy capability by extending definition 3 with the following rule :

4. $\langle q_R, w, q_E\rangle \xrightarrow{l}_{S\|E} \langle q_R, w', q_E\rangle$ if $w' \preccurlyeq w$.

The lossy action l must also be added to $A_{S\|E}$ to obtain an *Embedded Lossy Reactive Fiffo System (Embedded Lossy RFS)*. Notice that the losing action only acts on the memorizing queue, while R and E are not affected by it. We prove that Lossy RFS with well-structured environments are well-structured themselves.

4.1 Well-Structured Transition Systems

Well-Structured Transition Systems (WSTS) [14, 12, 11, 18, 3] provide a general framework for the analysis of infinite state systems. Several definitions of WSTS exist, from the first ones of Finkel [11, 12] and Abdulla *et al.* [3] to the unifying framework of Finkel and Schnoebelen [14].

WSTS are transition systems equipped with a wqo $\leq$ on the states which is compatible with their transition relation $\rightarrow$: "for all $q_1 \leq q_1'$ and transition $q_1 \rightarrow q_2$ there exists a sequence $q_1' \xrightarrow{*} q_2'$ such that $q_2 \leq q_2'$."

$$\begin{tikzcd} \forall & q_1 \arrow[r, dash, "\leq"] \arrow[d] & q_1' \arrow[d, dashed, "*"] & \\ & q_2 \arrow[r, dash, dashed, "\leq"] & q_2' & \exists \end{tikzcd}$$

Depending on the sequence $\sigma = q_1' \xrightarrow{*} q_2'$ and following [14], the compatibility is *strong* if σ is of length 1. If the sequence $\sigma = q_1' \rightarrow q_{11}' \rightarrow \ldots \rightarrow q_{1n}' \rightarrow q_2'$ contains at least one transition and moreover $q_1 \leq q_{1i}',\ \forall i \in \{1, \ldots, n\}$ then it is a *stuttering* compatibility.

Taking into account the labels of the transition system, we obtain a (strongly compatible) *Well-Structured Labeled Transition System (WSLTS)* [14]

$$\begin{tikzcd} \forall & q_1 \arrow[r, dash, "\leq"] \arrow[d, "a"] & q_1' \arrow[d, dashed, "a"] & \\ & q_2 \arrow[r, dash, dashed, "\leq"] & q_2' & \exists \end{tikzcd}$$

A WS(L)TS has *effective pred-basis* [14] if there exists an algorithm computing a finite-basis of $\uparrow$*Pred*$=$ ($\uparrow q$) for a given state q (denoted $pb(q)$). Then, for a WS(L)TS with decidable $\leq$ and effective pred-basis, it is possible to compute a finite basis of $Pred^*(\uparrow q)$, and it follows that the covering problem is decidable [14]. Furthermore, if the WS(L)TS has stuttering compatibility the model-checking of the fragment EG of CTL is decidable [18, 14].

4.2 WS–Embedded Lossy RFS are well-structured

Definition 5 (WS–Embedded Lossy RFS). *A* WS–Embedded Lossy RFS *is an Embedded Lossy RFS where the environment is a WSLTS.* □

Let $S \parallel E$ be an WS–Embedded Lossy RFS with a well-structured environment $(E, \sqsubseteq)$. We denote $\leq$ the binary relation on the configurations in $Q_{S\parallel E}$ defined by: $\langle q_R, w, q_E\rangle \leq \langle q'_R, w', q'_E\rangle \;\Leftrightarrow\; q_R = q'_R$ and $w \preccurlyeq w'$ and $q_E \sqsubseteq q'_E$.

Notice that $(Q_{S\parallel E}, \leq)$ is a wqo since $=$ is a wqo on finite sets and Q_R is finite, $(\Sigma^*, \preccurlyeq)$ is a wqo by Higman's lemma [16] and $\sqsubseteq$ is a wqo as we assume that the environment is well-structured.

Theorem 4. *Every WS–Embedded Lossy RFS $(S \parallel E, \leq)$ is a WSTS with stuttering compatibility.* □

Now, deciding if a given configuration $\langle q_R, w, q_E\rangle$ is covered from $\langle q'_R, w', q'_E\rangle$ consists in checking whether $\langle q'_R, w', q'_E\rangle \in Pred^*(\uparrow\langle q_R, w, q_E\rangle)$, where *Pred* is given by:

$$
\begin{aligned}
Pred(\langle q_R, w, q_E\rangle) = &\left\{\langle q'_R, w, q'_E\rangle \mid q'_R \xrightarrow{e}_R q_R \;\wedge\; q'_E \xrightarrow{e}_E q_E \;\wedge\; w \in (\Sigma^!_{q'_R})^*\right\}\\
&\cup \left\{\langle q_R, w', q'_E\rangle \mid q_R \xrightarrow{!e}_R q_R \;\wedge\; q'_E \xrightarrow{e}_E q_E \;\wedge\; w = w'e \;\wedge\; w' \in (\Sigma^!_{q'_R})^*\right\}\\
&\cup \left\{\langle q'_R, w', q_E\rangle \mid q'_R \xrightarrow{?e}_R q_R \;\wedge\; \forall w'_1 \in (\Sigma^!_{q'_R})^*,\, w'_2 \in \Sigma_M^* \text{ s.t. } w = w'_1 w'_2,\right.\\
&\left. w' = w'_1 e w'_2\right\}\\
&\cup \{\langle q_R, w', q_E\rangle \mid w' \in (\Sigma_M^* \sqcup\!\sqcup\, w)\}
\end{aligned}
$$

This is decidable for WS–Embedded Lossy RFS since from the following theorem, one can compute a finite basis for $Pred^*(\uparrow\langle q_R, w, q_E\rangle)$.

Theorem 5. *A WS–Embedded Lossy RFS has effective pred-basis if its environment has effective pred-basis.* □

Notice that computing a finite representation for $Post^*(S)$ (the effective recognizability problem) and computing a finite representation for $Pred^*(S)$ are two distinct problems since RFS (and thus (Lossy) Embedded RFS) are not effectively invertible. As a result, the $Post^*$ sets of RFS are recognizable but not computable whereas the $Pred^*$ sets are effectively recognizable.

Example 2. Assume that the environment E is finite-state. Clearly, E equipped with the equality quasi-ordering is a WSLTS with decidable wqo and with effective pred-basis.

Recalling that the loosing action l is not visible by the environment, we obtain that $\uparrow\langle q_R, w, q_E\rangle \subseteq Pred^*(\langle q_R, w, q_E\rangle)$ for any configuration $\langle q_R, w, q_E\rangle$ of $S \parallel E$. Therefore, we can compute a finite basis for $Pred^*(\langle q_R, w, q_E\rangle)$ by applying the principle of [1] for Lossy communicating machines.

Theorem 6. *For any configuration $\langle q_R, w, q_E\rangle$ of a WS–Embedded Lossy RFS, the upward closed set $Pred^*(\langle q_R, w, q_E\rangle)$ is effectively computable.* □

Thus it follows that the coverability problem is decidable for WS–Embedded Lossy RFS and as a consequence the reachability problem and the EF model-checking problem are decidable. The EG model-checking problem is also decidable since WS–Embedded Lossy RFS have stuttering compatibility.

4.3 Undecidability Results

The strong connection between deterministic two counters automata and Periodically Embedded RFS(2) (see Section 3.1) leads us to focus on the link between *Lossy Counters Automata* [5] and Periodically Embedded Lossy RFS.

Lossy Counters Automata (Lossy CA) are defined from counter automata (see Definition 4) by extending their semantics with the loosing rule:

4. $(q, m_1, \ldots, m_n) \xrightarrow{l} (q, m'_1, \ldots, m'_n)$ if $(\forall i,\ m'_i \leq m_i)$

We first prove the following result, which is easily derived from [10].

Theorem 7. *The boundedness problem and the recurrent locality problem are undecidable for Lossy 3–Counter Automata.* □

Now, the construction given in Section 3.1 to simulate deterministic 2-counter automata with Periodically Embedded RFS(2) can be adapted for deterministic 3-counter automata and Periodically Embedded RFS(3). Observe that a loss in a Periodically Embedded Lossy RFS corresponds (w.r.t. Parikh's mapping) to a loss in the counters automaton. Moreover, correctness of the construction is preserved by losiness.

The undecidability of the recurrent locality problem stated in the following theorem is proved in [19], using the idea in [2] for communicating finite state machines. The proof uses a machine that guesses its initial configuration in order to have an infinite execution. The existence of such a configuration is itself undecidable since the termination problem is. One needs a non-deterministic counter automaton for that "initial guess", but periodic environments constrain RFS in a strong deterministic way : our construction in Section 3.1 would not apply. The solution consists in considering *ultimately periodic environments* which first allow this "initial guess", and then constrain the considered RFS in the same periodic way that the environment we used in Section 3.1, thus preserving our construction.

An *Ultimately Periodic Embedded Lossy RFS* is an Embedded RFS with an ultimately periodic environment : it recognizes a language $L.(u)^\omega$ where L is a regular language and u a finite word. We get the following theorem.

Theorem 8. *The recurrent locality problem, the LTL model-checking problem and the boundedness problem are undecidable for Ultimately Periodic Embedded Lossy RFS(3). The CTL model-checking problem is undecidable for Ultimately Periodic Embedded Lossy RFS(4).* □

The difference between the LTL and the CTL model-checking problems lies in the impossibility to model the recurrent locality problem in CTL. As far as we know, the (un)decidability of CTL model-checking problem for Lossy 3–Counter Automata is still an open problem.

Notice that ultimately periodic environments are well-structured with effective pred-basis, since they can be modeled by finite state transition systems, and $=$ is a wqo on finite states. Thus, decidability results for WS–Embedded Lossy RFS apply to Ultimately Periodic Embedded Lossy RFS.

Theorem 9. *The covering problem and the model-checking problems for the fragments EG and EF of CTL are decidable for Ultimately Periodic Embedded Lossy RFS.* □

And the undecidability results for Ultimately Periodic Embedded Lossy RFS extend to WS–Embedded Lossy RFS.

Theorem 10. *The boundedness problem, the recurrent locality problem and the LTL and CTL model-checking problems are undecidable for WS–Embedded Lossy RFS.* □

5 Conclusion

Our results are summarized in table 1. In this table, U stands for "undecidable", D for "decidable", Y and N indicates if the state-space is recognizable, and its effectiveness between parentheses.

	Regularly Embedded RFS(1)	WS–Embedded Lossy RFS	Periodically Embedded RFS(2)
Boundedness	D	U*	U
Covering	D	D	U
Recognizability(effective)	Y(Y)	Y(N)	N
Recurrent locality	D	U*	U
LTL Model-Checking	D	U*	U
CTL/EG/EF Model-checking	D/D/D	U*/D/D	U/U/U

Table 1. Summary of the results.

For Embedded Lossy RFS, the star * denotes that these problems are still open when the RFS has two memorized events (and also for 3 memorized events in the case of the CTL model-checking problem).

Notice that the termination problem for Embedded RFS reduces to the termination problem for its environment. The decidability of the covering problem for WS–Embedded Lossy RFS allows us to verify safety properties on this class. Since the lossy semantics yields an upper approximation of an Embedded RFS, we get an (of course incomplete) method to verify safety properties on (non-lossy) Embedded RFS.

Our future work is concerned with the boundedness problem. Since it is undecidable for Embedded RFS and WS–Embedded Lossy RFS, we intend to design a semi-algorithm based on a test close to the one defined for Fifo Automata [17] in order to detect unboundedness of Embedded RFS.

References

1. P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 160–170. IEEE Computer Society Press, 1993.
2. P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. In S. Abiteboul and E. Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327, Jerusalem, Israel, 11–14 July 1994. Springer-Verlag.
3. P. A. Abdulla, K. Čerāns, B. Jonsson, and Y-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *INFCTRL: Information and Computation (formerly Information and Control)*, 160, 2000.
4. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
5. A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. 16th Ann. Symp. Theoretical Aspects of Computer Science (STACS'99), Trier, Germany, Mar. 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333. Springer, 1999.
6. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, April 1983.
7. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
8. F. Cassez and O. Roux. Compilation of the ELECTRE reactive language into finite transition systems. *Theoretical Computer Science*, 146(1–2):109–143, 24 July 1995.
9. CCITT. *Recommendation Z.100: Specification and Description Language SDL*, blue book, volume x.1 edition, 1988.
10. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T nets. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310. Springer, 1999.
11. A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In Thomas Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages, and Programming*, volume 267 of *LNCS*, pages 499–508, Karlsruhe, FRG, July 1987. Berlin: Springer.

12. A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, December 1990.
13. A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1–2):217–230, 15 March 1997.
14. A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
15. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97), Bologna, Italy, July 1997*, volume 9 of *Electronic Notes in Theor. Comp. Sci.*, pages 30–40. Elsevier Science, 1997.
16. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7):326–336, September 1952.
17. T. Jéron and C. Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113(1):93–117, 1993.
18. O. Kushnarenko and Ph. Schnoebelen. A formal framework for the analysis of recursive-parallel programs. *Lecture Notes in Computer Science*, 1277:45–??, 1997.
19. R. Mayr. Undecidable problems in unreliable computations. In *International Symposium on Latin American Theoretical Informatics (LATIN'2000)*, volume 1776 of *Lecture Notes in Computer Science*, Punta del Este, Uruguay, 2000. Springer-Verlag.
20. M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, London, 1 edition, 1967.
21. R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, October 1966.
22. G. Sutre, A. Finkel, O. Roux, and F. Cassez. Effective recognizability and model checking of reactive fiffo automata. In *Proc. 7th Int. Conf. Algebraic Methodology and Software Technology (AMAST'98), Amazonia, Brazil, Jan. 1999*, volume 1548 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 1999.