G. Sutre, A. Finkel, O. Roux and F. Cassez

Effective Recognizability and Model Checking of Reactive Fiffo Automata

Research Report LSV–98–10, Oct. 1998

Laboratoire
Spécification
et
Vérification

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens−cachan.fr/Publis/
Research Report LSV−98−10, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Oct. 1998
Short version published in *Proc. 7th Int. Conf. on Algebraic Methodology and Software Technology (AMAST'98), Amazonia − Manaus, Brazil, Jan. 1999 LNCS 1548, pp 106−123, Springer, 1999.*

Effective Recognizability and Model Checking of Reactive Fiffo Automata

G. Sutre[1], A. Finkel[1], O. Roux[2], F. Cassez[2]

[1] LSV, ENS Cachan & CNRS URA 2236, France, email: {sutre, finkel}@lsv.ens-cachan.fr
[2] IRCyN, EC Nantes & CNRS UMR 6597, France, email: {roux, cassez}@lan.ec-nantes.fr

Abstract. Our work intends to verify reactive systems with event memorization specified with the reactive language Electre. For this, we define a particular behavioral model for Electre programs, Reactive Fiffo Automata (RFAs), which is close to Fifo Automata. Intuitively, a RFA is the model of a reactive system which may store event occurrences that must not be immediately taken into account. We show that, contrarily to lossy systems where the reachability set is recognizable but not effectively computable, (1) the reachability set of a RFA is recognizable, and (2) it is effectively computable. Moreover, we also study the relationships between RFAs and Finite Automata and we prove that (3) from a trace language point of view, inclusions between RFAs and Finite Automata are undecidable and (4) the linear temporal logic LTL on states without the temporal operator next is decidable for RFAs, while LTL on transitions is undecidable.

1 Introduction

Objectives. The aim of this work is to verify reactive systems [MP92] with event memorization specified with the reactive language Electre [CR95]. A reactive program is supposed to react instantaneously to occurrences of events. A particular feature of the Electre language is that it is possible to store occurrences of events in order to process them later. The number of stored occurrences is unbounded. Consequently the behavioral model for an Electre program has an unbounded number of states and verification with standard model-checking techniques cannot be used on this model. Roux & Cassez have verified Electre programs by bounding the number of stored occurrences [CR97]. This paper deals with analysis of transition systems produced by compilation of Electre programs, *without any assumption on the boundedness of the number of stored occurrences.*

Related work. The behavioral model for Electre programs [CR95] is close to *Communicating Finite State Machines (CFSMs)* or *Fifo Automata*. However, this class has the power of Turing Machines since it's possible to simulate any Turing Machine by a system of two CFSMs [BZ83,FM97]. The reachability problem is decidable for systems with the recognizable channel property [Pac87], but this result cannot be easily used in general because this property is undecidable. Decidability results have been established for particular classes of Fifo Automata. The reachability problem is decidable for linear Fifo Automata, which can be simulated by colored Petri Nets [FC87,JJ93]. The reachability problem is decidable for lossy systems and the reachability set of a lossy system is recognizable [AJ93], but it is not effectively computable [CFP96]. Half-duplex systems and quasi-stable systems have a recognizable reachability set and it is effectively computable [CF97]. Semi-algorithms computing a symbolic representation for the reachability set of a Fifo Automaton have also been established [BG96,BGWW97,BH97,Que96,ABJ98].

Our contribution. Our work intends to establish similar results for the new class of *Reactive Fiffo Automata (RFAs)* [CR97] that models Electre programs. The three main results of the paper are:

1. the reachability set of a Reactive Fiffo Automaton is recognizable (section 4),
2. the reachability set of a Reactive Fiffo Automaton is effectively computable (section 4),
3. the linear temporal logic LTL without the temporal operator next (LTL\X) is decidable for Reactive Fiffo Automata (section 5). This result especially allows to check liveness and safety properties.

We also analyse the relationships between Reactive Fiffo Automata and Finite Automata and we prove that from a trace language point of view, inclusions between RFAs and Finite Automata are undecidable. Semantic models of other reactive languages are finite automata: in this sense, the expressiveness of Electre is strictly greater as the semantic model is a RFA.

Outline of the paper. Section 2 recalls several definitions we use throughout the paper. In section 3 we introduce the behavioral model for Electre programs which is a Reactive Fiffo Automaton. Section 4 is devoted to the proving that the reachability set of a RFA is recognizable and effectively computable. In section 5 we examine the relationships between Reactive Fiffo Automata and Finite Automata. Eventually we give in section 6 directions for future work.

Several proofs are not included in this paper, but they can be found in a longer version [1].

2 Preliminaries

Here are some basics on words and transition systems. Let Σ be an alphabet (a finite, non empty set). We write Σ^* for the set of all finite *words* $x_1x_2\cdots x_k$ with $x_i \in \Sigma$, and ε is the empty word. For two words $x, y \in \Sigma^*$, $x \sqcup\!\sqcup y$ is their shuffle: $x \sqcup\!\sqcup y = \{x_1y_1x_2y_2\cdots x_ny_n \ / \ x = x_1x_2\cdots x_n \text{ and } y = y_1y_2\cdots y_n \text{ with } x_i, y_i \in \Sigma^*\}$. If $x \in \Sigma^*$ is a word and $e \in \Sigma$ is a letter, we write $|x|_e$ for the number of occurrences of e in x. For two words $x, y \in \Sigma^*$, x is a *subword* of y iff $y \in x \sqcup\!\sqcup \Sigma^*$.

A *transition system* is a structure $TS = (S, s_0, A, \rightarrow)$ where S is a set of *states*, s_0 is the *initial state*, A is a finite set of *actions* and $\rightarrow \subseteq S \times A \times S$ is a set of *transitions*. We note $\xrightarrow{*}$ for the reflexive transitive closure of $\rightarrow$. An *execution* is a finite or infinite sequence of transitions $\left(s_i \xrightarrow{\alpha_i} s'_i\right)_{i\geq 1}$ such that for all $i \geq 1$, $s_{i+1} = s'_i$. Furthermore, we write $s_1 \xrightarrow{\alpha_1\alpha_2\cdots\alpha_n} s_{n+1}$ whenever we have $s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \cdots s_n \xrightarrow{\alpha_n} s_{n+1}$. A state s is said to be *reachable* in TS iff there exists an execution from the initial state $s_0 \xrightarrow{*} s$. The *reachability set* of TS, noted $\mathrm{RS}(TS)$, is the set of all reachable states in TS.

Let us also recall some decision problems for transition systems. The *Reachability Problem* is, given a transition system TS and a state s of TS, to determine whether s is reachable in TS. The *Reccurent Reachability Problem* is, given a transition system TS and a state s of TS, to determine whether there exists an execution in TS in which s appears infinitely often. The *Finite Reachability Set Problem* is, given a transition system TS, to determine whether the reachability set of TS is finite. The *Inclusion of Reachability Sets Problem* is, given two transition systems TS_1 and TS_2, to determine whether the reachability set of TS_1 contains the reachability set of TS_2. The *Termination Problem* is, given a transition system TS, to determine whether all executions in TS are finite.

[1] Available from the authors.

3 RFA: a model for reactive systems with event memorization

Electre is a reactive language aimed at specifying and programming real-time applications. Due to the types of these applications, we need to cope with events of different nature, for instance: a relevant classification concerns their memorization properties. To this extent, Electre provides for two sorts of events: fleeting or memorized.

These features are essential in the programming of some automated applications: real industrial experiments have been carried out in the field of embedded systems in cars and in the avionics (namely the SNECMA company [2] and the CERT/ONERA laboratory [BBnRR98]) but of course they are too big to be reported in this paper.

In order to understand the need for memorization of events, consider a conveyor which brings items to be manufactured at a rate which may differ from the rate of the manufacturing machine. To process all the incoming items, we have to memorize the pending items. Thus, the ability of the language to express memorization of events can ease the task of specification. This particular feature becomes crucial in the case there are many events the memorizations of which can be interleaved.

To deal with the memorization, we start with a finite model of Electre programs (Control Automaton, Definition 1) which does not take into account the memorization. Then, a list of stored occurrences of events is added to this finite model in order to deal with the ordered and multiple memorizations of the events occurrences. Thus, we obtain a Reactive Fiffo Automaton (RFA, Definition 2): a stored occurrence of an event is processed as soon as possible and priority is given to the oldest stored occurrence, hence the name *First In First Fireable Out (fiffo)*.

This memorization issue is completely defined in the semantics of the Electre language: this accounts for the semantic model of programs (RFA) which is the subject of this section.

The RFA model can be used for simulations or real executions (tools have been developed namely SILEX for simulations, EXILE for executions). A specific real-time executive based on the RFA model is run in EXILE and provides an efficient execution.

In this section, we first give a brief description of the language. Then, we define a behavioral model for Electre programs which is a Reactive Fiffo Automaton (RFA).

3.1 The Electre reactive language

Overview of the language. An Electre program describing the behavior of a process is made of three types of components:

modules: which are tasks of the process without blocking points: each instance of a module is a piece of executable code which can be either active, preempted or idle,

events: which can be software or hardware originated: each occurrence of an event is a signal which can be either memorized or not,

operators: combining the two previous components (for instance parallelism, sequence, preemption or launching (of a module by an event), repetition, and so on).

The term reactive means that the system controlling the process is to react *instantaneously* to any event occurring in the environment. As a running example, we will focus on an Electre program for describing the well-known readers/writers problem.

[2] Work partially supported under a three years grant number 765 358 L.

The readers/writers problem. The *readers/writers* problem was originally stated and solved in [CHP71]. There are several variations on this problem, all involving priorities. We specify our readers/writers problem here, with the following requirements:

– several readers can read the book simultaneously,
– when a writer writes the book, no other process (reader or writer) can access the book.

To specify the problem in the Electre language with two readers and two writers, we proceed as follows:

– the processes readers and writers are what we called *modules*
  - $READ_1$ (respectively $READ_2$) refers to the module for reader 1 (respectively reader 2) to read the book,
  - $WRITE$ refers to the module for both writer 1 and writer 2 to write the book,
– a request to read or write the book is an *event*
  - r_1 (respectively r_2) is a request for reading the book made by reader 1 (respectively reader 2),
  - w_1 (respectively w_2) is a request for writing the book made by writer 1 (respectively writer 2).

An Electre program that specifies the behavior of the system is presented in Figure 1.

```
PROGRAM Readers&Writers ;
     loop
        await
             { r₁ : READ₁ || r₂ : READ₂ }
           or
             #w₁  : WRITE
           or
             #w₂  : WRITE
      end loop ;
END Readers&Writers ;
```

Fig. 1. Readers/writers with no multiple memorization for reading requests

We shall not go into details about the syntax of the Electre language; the meaning of the above written program can be summed up as follows:

1. a request for reading (r_1 or r_2) is a *standard* event (no qualifier before them); this means that
   – if the request can not be taken into account at the time the event occurs then the request is:
     - ignored if it has already been stored;
     - stored otherwise;

     (it means that a standard event is memorized at most once).
   – on the contrary if the request can be taken into account, the corresponding module ($READ_1$ for request r_1) is launched (this is the meaning of the symbol ":"),

2. the activities of readers 1 and 2 may be run simultaneously (symbol "||") and when $READ_1$ is being run request r_2 can be taken into account (the converse when $READ_2$ is being run holds),
3. the writing activity $WRITE$ and the parallel activity $READ_1$ and $READ_2$ are in mutual exclusion (symbol "`or`"),
4. the requests for writing w_1 and w_2 can be satisfied one at a time (symbol "`or`") between the events w_1 and w_2;
5. the program consists in a cycle (structure "`loop` — `end loop`") of waiting until one of the events r_1, r_2, w_1 or w_2 occurs (structure "`await`"),
6. w_1 and w_2 are *multiple storage* events (prefixed by "`#`", Σ_M in Definition 1): occurrences of these events may be memorized an unbounded number of times.
   r_1 and r_2 are *single storage* events (not prefixed, Σ_S in Definition 1): only one occurrence of these events may be memorized at one time.
   Memorizable events are either multiple or single storage events. Other events are fleeting events.

3.2 From Electre programs to RFAs

The first step towards the behavioral model for Electre programs is a *Control Automaton*. Each transition of this automaton indicates what is to be done upon the occurrence of event. It is built according to the semantics of the language [CR95].

On the example of the readers and writers, we obtain the automaton depicted in Figure 2. It must be interpreted as follows:

- each `X` module completion (written $end_{\mathtt{X}}$) is a fleeting event,
- **immediate processing:** whenever the occurrence of an event `x` can be taken into account, the transition labeled `x` is triggered (e.g. $q_0 \xrightarrow{r_1} q_1$),

1.(a) **memorization/sending:** whenever the occurrence of a memorizable event `x` cannot be taken into account, it is stored, and the transition is labeled `!x` (e.g. $q_2 \xrightarrow{!w_1} q_2$); moreover, there is no state change in the Control Automaton (Definition 1, 1.(a)),

1.(b) **batch processing/reception:** whenever a stored occurrence of a memorizable event `x` is processed, the transition labeled ?`x` is triggered (e.g. $q_0 \xrightarrow{?w_2} q_3$). Batch processing an event has the same effect as the immediate processing of the same event (Definition 1, 1.(b)),

2. the automaton is complete w.r.t. memorizable events (Definition 1, 2.).

This does not state when the transitions are triggered. We will define the operational semantics of the Control Automaton in Definition 2.

In the sequel, we focus on the memorizable event: consequently, immediate processing transitions are abstracted in τ-transition (e.g. $q_0 \xrightarrow{r_1} q_1$ becomes $q_0 \xrightarrow{\tau} q_1$).

Now, we can give a formal definition of a Control Automaton:

Definition 1 (Control Automaton). *A* Control Automaton *is a finite transition system* $C = (Q, q_0, A, \rightarrow_C)$*, where:*

- Q *is a finite set of* control *states, and,*
- q_0 *is the initial state, and,*
- $A = (\{!, ?\} \times \Sigma) \cup \{\tau\}$ *is a finite set of actions such that* Σ *is an alphabet and* Σ_M*,* Σ_S *are subsets of* Σ *verifying:* $\Sigma_M \cap \Sigma_S = \emptyset$ *and* $\Sigma_M \cup \Sigma_S = \Sigma$*, and,*

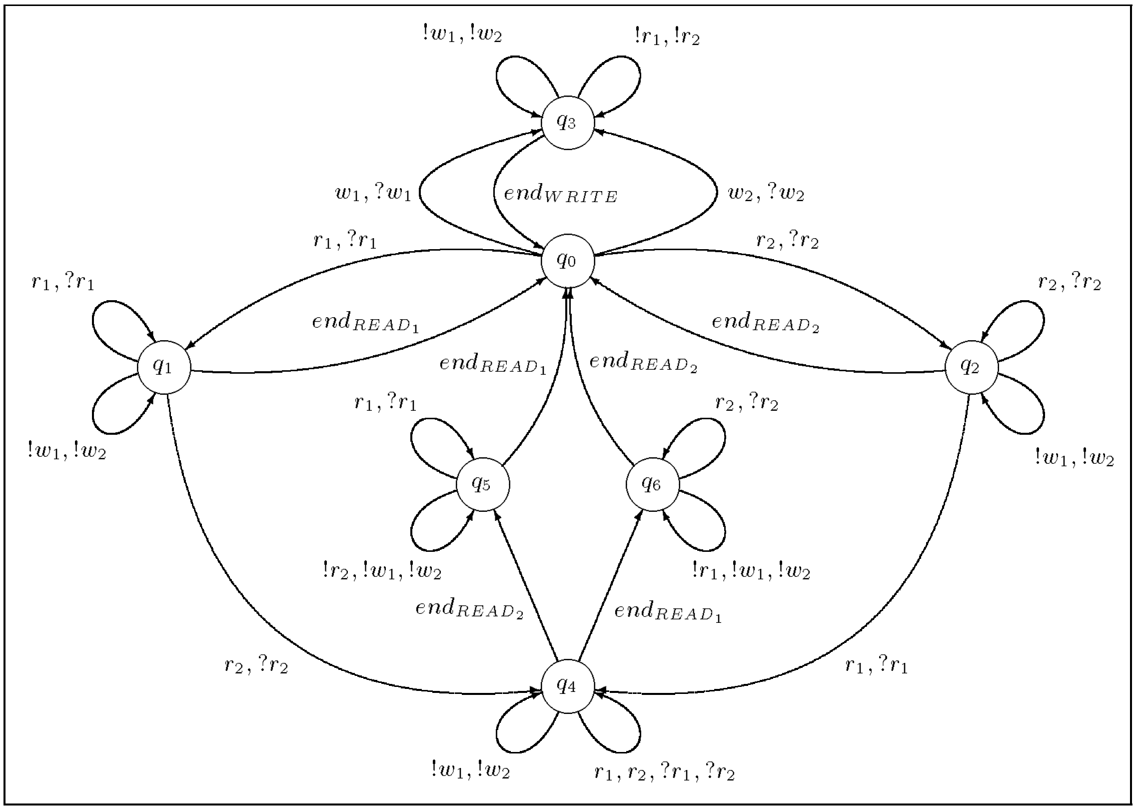

Fig. 2. The Reactive Fiffo Automaton for the readers/writers

- $\rightarrow_C$ *is any finite set of transitions, verifying the two following properties:*
  1. *for all* $q, q' \in Q$ *and* $e \in \Sigma$*:*
     (a) *if* $q \xrightarrow{!e}_C q'$*, then* $q = q'$*.*
     (b) *if* $q \xrightarrow{?e}_C q'$*, then* $q \xrightarrow{\tau}_C q'$*,*
  2. *for all* $q \in Q$ *and* $e \in \Sigma$*, we have either* $q \xrightarrow{!e}_C q$ *or there exists a state* q' *such that* $q \xrightarrow{?e}_C q'$*.*

  For every control state $q \in Q$*, we write* $\Sigma_q = \{e \in \Sigma \ / \ q \xrightarrow{!e}_C q\}$*.*

Remark 1. Every reachable state of a Control Automaton is reachable by an execution containing only τ-transitions.

A Control Automaton is built for every Electre program. A *fiffo queue* is then added to the Control Automaton to take the memorisation and batch processing of events into account. This is formally defined by the Reactive Fiffo Automaton (RFA).

Definition 2 (Reactive Fiffo Automaton). *The* Reactive Fiffo Automaton R *associated with a Control Automaton* $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow_C)$ *is the potentially infinite transition system* $R = (S, s_0, A, \rightarrow_R)$ *defined as follows:*

- $S = Q \times \Sigma^*$ *is the set of states, and,*
- $s_0 = (q_0, \varepsilon)$ *is the initial state, and,*

- $A = (\{!, ?\} \times \Sigma) \cup \{\tau\}$ *is the set of actions, and,*
- *the set of transitions* $\rightarrow_R$ *is the smallest subset of* $S \times A \times S$ *verifying:*
  1. *if* $q \xrightarrow{\tau}_C q'$ *then forall* $w \in {\Sigma_q}^*$, $(q, w) \xrightarrow{\tau}_R (q', w)$, *and*
  2. *if* $q \xrightarrow{!e}_C q'$ *then forall* $w \in {\Sigma_q}^*$, $(q, w) \xrightarrow{!e}_R (q', w')$, *where w' is defined by:*
     (a) $w' = w$ *if* $e \in \Sigma_S$ *and* $|w|_e \geq 1$,
     (b) $w' = we$ *otherwise, and*
  3. *if* $q \xrightarrow{?e}_C q'$ *then forall* $w_1 \in {\Sigma_q}^*, w_2 \in \Sigma^*$ $(q, w_1 e w_2) \xrightarrow{?e}_R (q', w_1 w_2)$,

Definition 3 (Stability). *A state* (q, w) *of a Reactive Fiffo Automaton is* stable *iff* $w \in {\Sigma_q}^*$. *Otherwise, it is* unstable.

The definition of a Reactive Fiffo Automaton corresponds to the informal semantics of the fiffo queue given in the beginning of this section:

- conditions 1 and 2 give priority to batch processings: stored occurrences of events are processed as soon as possible,
- condition 3 corresponds to the fiffo order: in a batch processing, priority is given to the oldest stored occurrence.

Example 1. Keeping our readers/writers example, let us consider the RFA R associated with the Control Automaton C described in Figure 2. An execution of R is for example:

$$(q_0, \varepsilon) \xrightarrow{w_1} (q_3, \varepsilon) \xrightarrow{!r_1} (q_3, r_1) \xrightarrow{!w_2} (q_3, r_1 w_2) \xrightarrow{!r_2} (q_3, r_1 w_2 r_2) \xrightarrow{end_{WRITE}} (q_0, r_1 w_2 r_2) \xrightarrow{?r_1} (q_1, w_2 r_2)$$
$$\xrightarrow{?r_2} (q_4, w_2) \xrightarrow{end_{READ_1}} (q_6, w_2) \xrightarrow{end_{READ_2}} (q_0, w_2) \xrightarrow{?w_2} (q_3, \varepsilon) \ldots$$

Three relevant observations can be done on this example :

- $(q_0, r_1 w_2 r_2)$ is an unstable state: priority is given to the processing of the first memorized occurrence (r_1),
- hence, even though $q_3 \xrightarrow{end_{WRITE}} q_0 \xrightarrow{w_1} q_3 \ldots$ is an execution of the Control Automaton C, $(q_3, r_1 w_2 r_2) \xrightarrow{end_{WRITE}} (q_0, r_1 w_2 r_2) \xrightarrow{w_1} (q_3, r_1 w_2 r_2) \ldots$ is not an execution of the RFA R, because the last transition is not a transition of R,
- in the transition $(q_1, w_2 r_2) \xrightarrow{?r_2} (q_1, w_2)$ above, it can be noticed that the processing of the memorized occurrences are done in the *First In First Fireable Out* order (which is not strictly the *fifo* order).

4 Computation of the recognizable reachability set of a RFA

We prove, in this section, that the reachability set of a Reactive Fiffo Automaton is recognizable and that it is effectively computable. This result especially allows us to decide the Reachability Problem, the Finite Reachability Set Problem and the Inclusion of Reachability Sets Problem for RFAs.

In the following, we consider a Reactive Fiffo Automaton R associated with a Control Automaton $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$.

Our first result states that the reachability set of a Reactive Fiffo Automaton is recognizable. This property comes essentially from condition 1.(a) of Definition 1. When an event e may be memorized, it is possible to memorize e^n for any $n \geq 0$.

Intuitively speaking, a RFA cannot count, but it takes the fiffo order (which is very close to the *fifo* order) into account. The fact that it cannot count allows the recognizability of its rechability set. Petri nets are orthogonal: they allow to count but not to retain the fifo ordering. So these two partially analysable models are based on different assumptions.

Hence, the fiffo queue of a RFA behaves like a fiffo queue capable of both lossiness and duplication errors [CFP96]. It follows that the reachability set of a RFA is recognizable.

Theorem 1. *The reachability set of R is recognizable.*

Proof. The proof is similar to the proof that the reachability set of a lossy system is recognizable [AJ93]. Let $\preceq$ be the well ordering over $Q \times \Sigma^*$ defined by $(q, w) \preceq (q', w')$ iff $q = q'$ and w is a subword of w'. Assume the reachability set $\mathrm{RS}(R)$ of R is downward closed. Then $\mathrm{Compl}(\mathrm{RS}(R))$ is upward closed. Since $\preceq$ is well ordering, $\mathrm{Compl}(\mathrm{RS}(R))$ has a finite set M of minimal elements, which gives a recognizable description of $\mathrm{Compl}(\mathrm{RS}(R))$:

$$\mathrm{Compl}(\mathrm{RS}(R)) = \bigcup_{(q,w)\in M} (q, w \sqcup\!\sqcup \Sigma^*)$$

As $\mathrm{Compl}(\mathrm{RS}(R))$ is recognizable, we obtain that $\mathrm{RS}(R)$ is recognizable. It remains to prove that $\mathrm{RS}(R)$ is downward closed. Let $(q, w_1 e w_2) \in \mathrm{RS}(R)$, with $w_1, w_2 \in \Sigma^*$ and $e \in \Sigma$. We show that $(q, w_1 w_2) \in \mathrm{RS}(R)$. Since $(q, w_1 e w_2) \in \mathrm{RS}(R)$, there exists an execution $\pi = (q_0, \varepsilon) \xrightarrow{*} (q, w_1 e w_2)$ in R, which may be decomposed as follows:

$$\pi = (q_0, \varepsilon) \xrightarrow{\sigma_1} (q, x) \xrightarrow{!e} (q, xe) \xrightarrow{\sigma_2} (q, w_1 e w_2)$$

where the event occurrence $\mathbf{e}$ memorized by the transition $t = (q, x) \xrightarrow{!\mathbf{e}} (q, x\mathbf{e})$ is the event occurrence $\mathbf{e}$ in $w_1 \mathbf{e} w_2$. Hence $|\sigma_2|_{?e} \leq |x|_e$ and π' defined below is still an execution of R, as it is possible to remove the transition t from π [3]:

$$\pi' = (q_0, \varepsilon) \xrightarrow{\sigma_1} (q, x) \xrightarrow{\sigma_2} (q, w_1 w_2)$$

□

However, this does not prove that the reachability set of a Reactive Fiffo Automaton is *effectively* recognizable. For instance in the case of lossy systems, it has been shown that the reachability set is recognizable, *but not computable* [CFP96].

The following of this section is devoted to the proving that the recognizable reachability set of a Reactive Fiffo Automaton is *computable.*

Definition 4. *For every $q \in Q$, we call* language of the fiffo queue in the control state q, *written $\mathcal{L}_R(q)$, the set:*

$$\mathcal{L}_R(q) = \{w \in \Sigma^* \ / \ (q_0, \varepsilon) \xrightarrow{*} (q, w)\}$$

[3] Intuitively, the memorized event occurrence $\mathbf{e}$ could have not occured, since memorization does not change the control state

It is clear, according to the definition of a Reactive Fiffo Automaton that if a control state $q \in Q$ is not reachable in the Control Automaton C then $\mathcal{L}_R(q) = \emptyset$. We will in the following deal with the control states $q \in \mathrm{RS}(C)$ reachable in C.

Notation. For every $F \subseteq \Sigma$, we write $\mathcal{S}(F^*)$ for the set of words over F containing at most one occurrence of each single storage event, $\mathcal{S}(F^*) = \{w \in F^* \;/\; \forall\, e \in \Sigma_S, |w|_e \leq 1\}$.

Let us notice that for any $F \subseteq \Sigma$, the language $\mathcal{S}(F^*)$ is regular. Moreover, according to the definition of a Reactive Fiffo Automaton, a single storage event can appear at most once in the fiffo queue part of a reachable state. More formally:

Remark 2. For all reachable state (q, w) of R, we have $w \in \mathcal{S}(\Sigma^*)$.

Lemma 1. *Let $q \in \mathrm{RS}(C)$ be a control state of R reachable in C. If $w \in \mathcal{S}(\Sigma_q^*)$ then (q, w) is reachable in R.*

Proof. According to Remark 1, there exists an execution $q_0 \xrightarrow{\tau} q_1 \xrightarrow{\tau} q_2 \cdots q_l \xrightarrow{\tau} q$ in C. In this way, $(q_0, \varepsilon) \xrightarrow{\tau} (q_1, \varepsilon) \xrightarrow{\tau} (q_2, \varepsilon) \cdots (q_l, \varepsilon) \xrightarrow{\tau} (q, \varepsilon)$ is an execution of R. Consequently, (q, ε) is reachable in R.

As $w \in \mathcal{S}(\Sigma_q^*)$, on one hand $\mathrm{alph}(w) \subseteq \Sigma_q$ and on the other hand for all $e \in \Sigma_S$, we have $|w|_e \leq 1$. Writing $w = e_1 \cdots e_k$, we infer that $(q, \varepsilon) \xrightarrow{!e_1 \cdots !e_k} (q, e_1 \cdots e_k)$ is an execution of R. Finally, (q, w) is reachable in R. □

Using Lemma 1 and Remark 2, it is easy to infer that the recognizable set of *stable* reachable states in R is computable, because it may be written as $\cup_{q \in \mathrm{RS}(C)} (q, \mathcal{S}(\Sigma_q^*))$.

We will now, in the following lemma, also deal with the *unstable* reachable states of R.

Lemma 2. *Let (q, w) be a stable reachable state of R and $(q, w) \xrightarrow{\tau} (q_1, w) \xrightarrow{?e_1?e_2 \cdots ?e_k} (q', w')$ be an execution of R. Then we have:*

1. *$e_1 e_2 \cdots e_k \in \mathcal{S}(\Sigma_q^*)$ and,*
2. *$w' \in \mathcal{S}(X^*)$, where $X = \Sigma_q \setminus (\Sigma_S \cap \{e_1, e_2, \cdots, e_k\})$ and $\{e_1, e_2, \cdots, e_k\}$ denotes the set associated with the multiset consisting of the elements $e_1, e_2, \cdots, e_k$.*

Proof. As (q, w) is a stable state, we have $w \in {\Sigma_q}^*$. On the other hand, since (q, w) is reachable in R, we get, from Remark 2, $w \in \mathcal{S}(\Sigma_q^*)$. Moreover, given that $?e_1?e_2 \cdots ?e_k$ is a sequence of receptions (i.e. batch processings in the Electre language), we obtain that $w \in w' \sqcup\!\sqcup \mathrm{Perm}(e_1 e_2 \cdots e_k)$. Hence, we infer:

- $e_1 e_2 \cdots e_k$ and w' are words over $\mathcal{S}(\Sigma_q^*)$, and,
- for every $e \in \Sigma_S$, $|w'|_e + |e_1 e_2 \cdots e_k|_e = |w|_e \leq 1$. Therefore, we have $w' \in \mathcal{S}(X^*)$, where $X = \Sigma_q \setminus (\Sigma_S \cap \{e_1, e_2, \cdots, e_k\})$. □

We now define, for all $(q, F) \in Q \times 2^{\Sigma}$, the set $\mathrm{CoReach}(q, F)$, which will allow us to prove that the reachability set of R is recognizable. Intuitively, we define $\mathrm{CoReach}(q, F)$ so that if a control state p is in $\mathrm{CoReach}(q, F)$, then the states in $(q, \mathcal{S}(F^*))$ are reachable from the set of stable states $(p, \mathcal{S}(\Sigma_p^*))$ and hence are reachable in R.

Definition 5 (CoReach). *Let $q \in Q$ be a control state of R and $F \subseteq \Sigma$ be a subset of Σ. The set $\mathrm{CoReach}(q, F) \subseteq Q$ is the set of control states $p \in Q$ such that there exists an execution $p \xrightarrow{\tau} q_1 \xrightarrow{?e_1?e_2 \cdots ?e_k} q$ in C verifying:*

1. $e_1 e_2 \cdots e_k \in \mathcal{S}(\Sigma_p^*)$ *and,*
2. $F = \Sigma_p \setminus (\Sigma_S \cap \{e_1, e_2, \cdots, e_k\})$.

Let us remark that for all $(q, F) \in Q \times 2^{\Sigma}$, the set $\{F \subseteq \Sigma \ / \ \mathrm{CoReach}(q, F) \neq \emptyset\}$ is finite. The following theorem gives a precise description of the reachability set of a Reactive Fiffo Automaton, which will allow us to prove that the recognizable reachability set of a Reactive Fiffo Automaton is computable.

Theorem 2. *For every control state* $q \in \mathrm{RS}(C)$ *reachable in* C*, we have:*

$$\mathcal{L}_R(q) = \mathcal{S}(\Sigma_q^*) \quad \bigcup \quad \left[\bigcup_{F \subseteq \Sigma \ / \ \mathrm{CoReach}(q,F) \neq \emptyset} \mathcal{S}(F^*) \right]$$

Proof. Let us prove the inclusion from left to right. Assume $q \in \mathrm{RS}(C)$ and $w \in \mathcal{L}_R(q)$. Two cases may arise:

- (q, w) is stable: then according to the definition of a stable state, we have $w \in {\Sigma_q}^*$. In this way, Remark 2 leads to $w \in \mathcal{S}(\Sigma_q^*)$.
- (q, w) is unstable: as $w \in \mathcal{L}_R(q)$, there exists an execution $\pi = (q_0, \varepsilon) \xrightarrow{*} (q, w)$ in R. Since (q_0, ε) is stable, π contains a stable state, and we call (q_s, w_s) the *last stable state* of π. As (q, w) is unstable, we come to $(q, w) \neq (q_s, w_s)$. Hence (q_s, w_s) is the source of a transition in π and we get that π may be written as:

$$\pi \quad = \quad (q_0, \varepsilon) \xrightarrow{*} (q_s, w_s) \xrightarrow{\alpha} (q_1, w_1) \xrightarrow{\sigma} (q, w)$$

  As (q_s, w_s) is the last stable state of π, we obtain:
  - on one hand $\alpha \notin \{?\} \times \Sigma$ and on the other hand $\alpha \notin \{!\} \times \Sigma$ because otherwise, (q_1, w_1) would be a stable state. Therefore $\alpha = \tau$ and $w_1 = w_s$.
  - every state in $(q_1, w_1) \xrightarrow{\sigma} (q, w)$ is unstable. Therefore, $\sigma \in (\{?\} \times \Sigma)^*$ and we assume in the following that σ is written $?e_1?e_2 \cdots ?e_k$.

  Let us assume that $F = \Sigma_{q_s} \setminus (\Sigma_S \cap \{e_1, e_2, \cdots, e_k\})$. We can apply Lemma 2 so that:
  - $e_1 e_2 \cdots e_k \in \mathcal{S}(\Sigma_{q_s}^*)$. Moreover $q_s \xrightarrow{\tau} q_1 \xrightarrow{?e_1?e_2\cdots?e_k} q$ is an execution of C. Therefore $q_s \in \mathrm{CoReach}(q, F)$ so that $\mathrm{CoReach}(q, F) \neq \emptyset$.
  - $w \in \mathcal{S}(F^*)$.

Finally:

$$w \quad \in \quad \mathcal{S}(\Sigma_q^*) \quad \bigcup \quad \left[\bigcup_{F \subseteq \Sigma \ / \ \mathrm{CoReach}(q,F) \neq \emptyset} \mathcal{S}(F^*) \right]$$

Let us prove the inclusion from right to left. Assume $q \in \mathrm{RS}(C)$. We notice that according to Lemma 1, for all $w \in \mathcal{S}(\Sigma_q^*)$, (q, w) is reachable in R. Now assume $\mathrm{CoReach}(q, F) \neq \emptyset$ and $w \in \mathcal{S}(F^*)$. Since $\mathrm{CoReach}(q, F) \neq \emptyset$, there exists a control state p and an execution $p \xrightarrow{\tau} q_1 \xrightarrow{?e_1?e_2\cdots?e_k} q$ in C such that:

- $e_1 e_2 \cdots e_k \in \mathcal{S}(\Sigma_p^*)$ and,
- $F = \Sigma_p \setminus (\Sigma_S \cap \{e_1, e_2, \cdots, e_k\})$.

As $w \in \mathcal{S}(F^*)$, we have for all $e \in \Sigma_S$:

– if $e \in \{e_1, \ldots, e_k\}$ then $e \notin F$, hence $e \notin \text{alph}(w)$. Therefore $|(e_1 \ldots e_k) \cdot w|_e = |e_1 \ldots e_k|_e \leq 1$.
– if $e \notin \{e_1, \ldots, e_k\}$ then $|(e_1 \ldots e_k) \cdot w|_e = |w|_e \leq 1$.

Consequently, in both cases, we have $|(e_1 \ldots e_k) \cdot w|_e \leq 1$. We remark that $(e_1 \ldots e_k) \cdot w \in {\Sigma_p}^*$, hence $(e_1 \ldots e_k) \cdot w \in \mathcal{S}(\Sigma_p^*)$. We can apply Lemma 1 so that $(p, (e_1 \ldots e_k) \cdot w)$ is reachable in R. Because $(p, (e_1 \ldots e_k) \cdot w) \xrightarrow{\tau} (q_1, (e_1 \ldots e_k) \cdot w) \xrightarrow{?e_1 \ldots ?e_k} (q, w)$ is an execution of R, we obtain that (q, w) is reachable in R. Finally:

$$w \in \mathcal{L}_R(q) \qquad \square$$

Now let us present the main result of this section which says that a regular expression for the recognizable reachability set of a Reactive Fiffo Automaton is effectively computable.

Theorem 3. *There exists an algorithm computing a regular expression for $\mathcal{L}_R(q)$, for every $q \in Q$.*

Proof. For every $(q, F) \in Q \times 2^{\Sigma}$, we define the set $\text{CoReach}'(q, F)$ in the same way that $\text{CoReach}(q)$, but enforcing the execution of C to be **simple** [4]. Let us notice that, for all $q \in Q$, the set $\{F \subseteq \Sigma \ / \ \text{CoReach}'(q, F) \neq \emptyset\}$ is finite and computable. Hence, it is enough to prove that for all $q \in Q$:

$$\bigcup_{F \subseteq \Sigma \ / \ \text{CoReach}(q,F) \neq \emptyset} \mathcal{S}(F^*) \quad = \quad \bigcup_{F \subseteq \Sigma \ / \ \text{CoReach}'(q,F) \neq \emptyset} \mathcal{S}(F^*)$$

The proof of $\supseteq$ is straightforward, when one remarks that for all $(q, F) \in Q \times 2^{\Sigma}$, $\text{CoReach}(q, F)$ contains $\text{CoReach}'(q, F)$. Now let us prove $\subseteq$. Assume $\text{CoReach}(q, F) \neq \emptyset$. By definition, there exists a control state p and an execution $p \xrightarrow{\tau} q_1 \xrightarrow{?e_1 ?e_2 \cdots ?e_k} q$ in C verifying the conditions 1 and 2 of Definition 5. If the execution $\pi = q_1 \xrightarrow{?e_1 ?e_2 \cdots ?e_k} q$ is not simple, we "simplify" π in the classical way: we remove the cycles untill we obtain an execution π' *possibly empty*. We remark that π' may be written as $\pi' = q_1 \xrightarrow{?e'_1 ?e'_2 \cdots ?e'_{k'}} q$ where $e'_1 e'_2 \cdots e'_{k'}$ is a subword of $e_1 e_2 \cdots e_k$. Assuming $F' = \Sigma_p \setminus (\Sigma_S \cap \{e'_1, e'_2, \cdots, e'_{k'}\})$, we get $p \in \text{CoReach}'(q, F')$ so that $\text{CoReach}'(q, F') \neq \emptyset$ and also $F \subseteq F'$. $\square$

From the previous theorem, one may easily deduce the following corollary.

Corollary 1. *The Reachability Problem, the Finite Reachability Set Problem, the Inclusion of Reachability Sets Problem, the Control State Reachability Problem and the Termination Problem are decidable for RFAs.*

While recognizability of the reachability set of a Reactive Fiffo Automaton comes essentially from condition 1.(a) of Definition 1, effectivity crucially depends on condition 2. of Definition 1. As a matter of fact, let us show that if we extend Control Automata in removing condition 2. of Definition 1 then the reachability set is still recognizable but it becomes not effectively computable. Indeed, for every machine capable of both lossiness and duplication errors, one may construct a generalized RFA having the same reachability set. Now it is known that machines capable of both lossiness and duplication errors have a non effective recognizable reachability set [CFP96].

[4] an execution $p_0 \to p_1 \to p_2 \cdots p_{l-1} \to p_l$ is *simple* iff on the one hand $p_0, p_1, \cdots, p_{l-1}$ are distinct and on the other hand $p_1, p_2, \cdots, p_l$ are distinct.

Example 2. Let us consider the RFA R modelling our readers/writers example. The reachability set of R is:

$$\bigcup_{i\in\{0,1,\cdots,6\}} (q_i, \mathcal{S}(F_i^*))$$

with $F_0 = F_3 = \{r_1, r_2, w_1, w_2\}$, $F_1 = F_5 = \{r_2, w_1, w_2\}$, $F_2 = F_6 = \{r_1, w_1, w_2\}$ and $F_4 = \{w_1, w_2\}$. The state $(q5, w_1r_2w_2w_2)$ is reachable while the state $(q5, r_1)$ is not reachable. All the control states of R are reachable. The reachability set of R is infinite and R does not terminate.

Remark 3. We have implemented an optimized algorithm which computes simultaneously for all control state q, a regular expression for $\mathcal{L}_R(q)$. This algorithm has a complexity of $O(K(|\Sigma|)\cdot |Q|\cdot|\rightarrow|)$, where $K(|\Sigma|) = O(2^{|\Sigma|})$ is the complexity of subset operations over Σ. Hence, the various decision problems of Corollary 1 are decidable with the same complexity.

The reachability set of a Reactive Fiffo Automaton with a non empty initial fiffo queue is still recognizable and effectively computable [Sut97]. This result especially allows us to decide the Recurrent Reachability Problem for RFAs.

The previous theorems are very useful when considering practical aspects: simulation and verification. Indeed, the compilation of an Electre program produces a RFA (given by its associated Control Automaton), which is used for simulation and verification purposes. This RFA leads to a C program, which is then compiled to produce an executable file. Clearly, the control states which are not reachable do not need to be included in the C program. This is also the case for the transitions of the Control Automaton which are not quasi-live (a transition t of a Control Automaton is *quasi-live* if there exists an execution containing t in the associated RFA). Fortunately, the Quasi-liveness Problem is decidable for RFAs.

Proposition 1. *The Quasi-liveness Problem is decidable for RFAs.*

Proof. If $t = (q \rightarrow q')$ is a τ-transition or if t is an emission transition of the Control Automaton C, then t is quasi-live if and only if q is a reachable control state of C, which is decidable. A reception transition $q \xrightarrow{?e} q'$ of C is quasi-live if and only if $\mathcal{L}_R(q) \cap {\Sigma_q}^* e\Sigma^* \neq \emptyset$, which is decidable because a regular expression for $\mathcal{L}_R(q)$ is computable. □

5 Relationships between Reactive Fiffo Automata and Finite Automata

In the previous section, we have precisely described the reachability set of a Reactive Fiffo Automaton. We now analyse the set of executions of a RFA, and we establish a comparison between RFAs and Finite Automata. We first study the general case of trace inclusion between a RFA and a Finite Automaton. We then analyse the model checking of LTL.

5.1 Trace inclusions

One may believe that Reactive Fiffo Automata are essentially equivalent to Finite Automata, because reception transitions are not blocking (as for each reception transition $q \xrightarrow{?e} q'$ of a Control Automaton, there exists a τ-transition $q \xrightarrow{\tau} q'$) and emission transitions can be repeated arbitrarily often. But the following undecidability results prove that this is not the case.

We define the *trace language* of a RFA in the usual way : we introduce a new alphabet of actions $\mathcal{A}$ and every transition of a Control Automaton is labelled by an *action* $a \in \mathcal{A}$ or by the empty word ε. The set of *finite (resp. infinite) traces* $\mathcal{T}^*(R)$ (resp. $\mathcal{T}^\omega(R)$) of a RFA R is the set of finite (resp. infinite) words on $\mathcal{A}$ corresponding to finite (resp. infinite) executions of R.

It is clear that for every regular language L (resp. ω-regular language L), there exists a RFA R such that $\mathcal{T}^*(R) = L$ (resp. $\mathcal{T}^\omega(R) = L$). The following theorem shows that from a trace language point of view, inclusions between RFAs and Finite Automata are undecidable.

Theorem 4. *The four following problems are undecidable :*

i) Given a RFA R and a regular language L, is $\mathcal{T}^*(R) \subseteq L$?
ii) Given a RFA R and a regular language L, is $\mathcal{T}^*(R) \supseteq L$?
iii) Given a RFA R and an ω-regular language L, is $\mathcal{T}^\omega(R) \subseteq L$?
iv) Given a RFA R and an ω-regular language L, is $\mathcal{T}^\omega(R) \supseteq L$?

Proof. (Sketch) We only proof i) and ii) as the two other cases are very similar, but on infinite words. Let Q'^*_2 denote the Anti-Dyck language [VFZ80]. The following problem is easily shown to be undecidable : given a regular language L, is $L \cap Q'^*_2 = \emptyset$? Now, there exist two RFAs R_1 and R_2 such that $\mathcal{T}^*(R_1) = Q'^*_2$ and $\mathcal{T}^*(R_2) = \mathrm{Compl}(Q'^*_2)$. Hence, for every regular language L, we have :

- $\mathcal{T}^*(R_1) \subseteq L$ if and only if $L \cap Q'^*_2 = \emptyset$, which is undecidable,
- $\mathcal{T}^*(R_2) \supseteq L$ if and only if $L \cap Q'^*_2 = \emptyset$, which is undecidable. □

As RFAs contain all Finite Automata, we obtain the following corollary.

Corollary 2. *The two following problems are undecidable :*

i) Given two RFAs R_1 and R_2, is $\mathcal{T}^*(R_1) \subseteq \mathcal{T}^*(R_2)$?
ii) Given two RFAs R_1 and R_2, is $\mathcal{T}^\omega(R_1) \subseteq \mathcal{T}^\omega(R_2)$?

5.2 Model checking with LTL

We prove, in this section, that the linear temporal logic LTL [MP92,Eme90] without the temporal operator next, which we denote by LTL$\backslash$X, is decidable for Reactive Fiffo Automata. This result especially allows to check liveness and safety properties, and also to decide the Reccurent State Problem for RFAs.

Notation. For every transition system $TS = (S, s_0, A, \rightarrow)$, we write $\mathcal{L}^\omega(TS)$ for the set of ω-sequences of states corresponding to infinite executions of TS:

$$\mathcal{L}^\omega(TS) = \{s_0 s_1 \cdots s_n \cdots \;/\; \forall\, i \in \mathbb{N}, s_i \rightarrow s_{i+1}\}$$

In the following, we consider a finite set AP of atomic propositions on which are based LTL formulas. Unless specified, we assume that the atomic propositions label control states. Two ω-sequences of states are *equivalent modulo stuttering* iff they display the same ω-sequence of states when two repeated consecutive states are seen as one state only.

Theorem 5 ([Lam83]). *Two ω-sequences of states equivalent modulo stuttering satisfy the same* LTL$\backslash$X *formulas.*

We will now prove that a Control Automaton C and the Reactive Fiffo Automaton R associated with C satisfy the same LTL$\setminus$X formulas. In this way, Model-checking of LTL$\setminus$X is decidable for Reactive Fiffo Automata. In order to demonstrate this result, we introduce a Finite Automaton, $\mathrm{Restr}(C)$, built from C. We will actually show that C, R and $\mathrm{Restr}(C)$ satisfy the same LTL$\setminus$X formulas.

Definition 6 (Restricted Control Automaton). *The* Restricted Control Automaton $\mathrm{Restr}(C)$ *associated with a Control Automaton* $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$ *is the finite transition system* $\mathrm{Restr}(C) = (S, s_0, A, \rightarrow_{\mathrm{Restr}(C)})$ *defined as follows:*

- $S = Q \cup \overline{Q}$, *with* $\overline{Q} = \{\overline{q} \ / \ q \in Q\}$ *a copy of* Q, *is the set of states, and,*
- $s_0 = q_0$ *is the initial state, and,*
- $A = (\{!\} \times \Sigma) \cup \{\tau\}$ *is the set of actions, and,*
- *the set of transitions* $\rightarrow_{\mathrm{Restr}(C)}$ *is the smallest subset of* $S \times A \times S$ *verifying for all* $q, q' \in Q$ *the two following properties:*
  1. *if* $q \xrightarrow{\tau} q'$ *then* $q \xrightarrow{\tau}_{\mathrm{Restr}(C)} q'$, *and,*
  2. *if* $q \xrightarrow{!e} q$ *then we have* $q \xrightarrow{!e}_{\mathrm{Restr}(C)} \overline{q}$ *and also* $\overline{q} \xrightarrow{!e}_{\mathrm{Restr}(C)} \overline{q}$.

We notice that, according to point 2. of Definition 1, each state of a Control Automaton is the source of a transition. Furthermore, this property holds for Reactive Fiffo Automata and Restricted automata too. In this way, model checking of LTL is well-defined for these transitions system.

In the following, in order to simplify the presentation, *we identify states* q *and* $\overline{q}$ *of a Restricted Control Automaton.* Moreover, if $Q \times \Sigma^*$ is the set of states of a Reactive Fiffo Automaton, we will write proj_Q for the projection on control states: proj_Q is the morphism $proj_Q : (Q \times \Sigma^*)^* \rightarrow Q^*$ defined by $\mathrm{proj}_Q(q, w) = q$.

The two following lemmas express close relations between Control Automata, Reactive Fiffo Automata and Restricted Control Automaton. Their proofs involve simple execution transformations, and the reader will find them in the appendix.

Lemma 3. *Let* R *be the Reactive Fiffo Automaton associated with a Control Automaton* $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$. *We have:*

$$\mathcal{L}^\omega(\mathrm{Restr}(C)) \ \subseteq \ \mathrm{proj}_Q(\mathcal{L}^\omega(R)) \ \subseteq \ \mathcal{L}^\omega(C)$$

Lemma 4. *Let* $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$ *be a Control Automaton. For every* ω*-sequence* $\nu \in \mathcal{L}^\omega(C)$, *there exists an* ω*-sequence* $\nu' \in \mathcal{L}^\omega(\mathrm{Restr}(C))$ *such that* ν *and* ν' *are equivalent modulo stuttering.*

We now present the main result of this section, which especially allow to check liveness and safety properties on RFAs.

Theorem 6. *Model-checking of* LTL$\setminus$X *is decidable for Reactive Fiffo Automata and it is* PSPACE-*complete.*

Proof. Let R be a RFA associated to a Control Automaton $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$. According to Lemma 3, to Lemma 4 and to Theorem 5, we obtain that C, R and $\mathrm{Restr}(C)$ satisfy the same LTL$\setminus$X formulas. Now, model-checking of LTL$\setminus$X is PSPACE-complete for Finite Automata [SC85], which concludes the proof. $\square$

Corollary 3. *The Reccurent Control State Problem is decidable for Reactive Fiffo Automata.*

We finally show that the model-checking of LTL with atomic propositions on transitions is undecidable. Let us remark that this result is stronger than Teorem 4 *iii*).

Theorem 7. *Model-checking of* LTL *with atomic propositions on transitions is undecidable for Reactive Fiffo Automata.*

Proof. (Sketch) We show that for every Fifo Automaton A, there exists a RFA R, a formula $\varphi \in$ LTL and a valuation on the set of control transitions of R such that the executions of A are exactly the executions of R satisfying the formula φ. In this way, we reduce the model checking of LTL for Fifo Automata (which is known to be undecidable) to the model checking of LTL for RFAs. □

6 Conclusion

In this work, we were interested in the verification of Reactive Fiffo Automata, an new class of infinite systems that models Electre programs.

We have shown in this paper that Reactive Fiffo Automata have a recognizable reachability set and that it is effectively computable. This result allows to decide several problems involved in verification of RFAs, for instance the Reachability Problem, the Finite Reachability Set Problem and the Inclusion of Reachability Sets Problem.

This work is a first step towards the assesment of response times of the system for taking into account memorized occurrences of events. This will be of a great significance for real-life systems.

We have also analysed the relationships between Reactive Fiffo Automata and Finite Automata. We have proved that from a trace language point of view, inclusions between RFAs and Finite Automata are undecidable. But fortunately, we obtained that the linear temporal logic LTL without the temporal operator next (LTL\X) is decidable for Reactive Fiffo Automata. This result especially allows to check liveness and safety properties.

The decidability of fragments of CTL has already been investigated in [Sut97]. The decidability of LTL (with atomic propositions on states) and the decidability of CTL remain open problems.

References

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc. of the* 10^{th} *Conference on Computer-Aided Verification (CAV)*, 1998.

[AJ93] P. Adulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. of the 8th IEEE Symposium on Logic in Computer Science*, 1993.

[BBnRR98] F. Boniol, A. Burgueño, O. Roux, and V. Rusu. Étude d'un modèle hybride discret - continu pour la spécification de systèmes temps-réel embarqués. Rapport de contrat CERT/ONERA - IRCyN N^0 DERI 3703-33, February 1998.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. of the* 8^{th} *Conference on Computer-Aided Verification (CAV)*, volume 1102, pages 1–12. LNCS, August 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of qdds. In *Proceedings of SAS'97*, September 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. In *Proc. of the 24th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1256, pages 560–570. LNCS, July 1997.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.

[CF97] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *Proc. of the* 9^{th} *Conference on Computer-Aided Verification (CAV)*, volume 1254, pages 304–315. LNCS, June 1997.

[CFP96] G. Cécé, A. Finkel, and I. S. Purushothaman. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[CHP71] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with "readers" and "writers". *Communications of the ACM*, 14(10):667–668, October 1971.

[CR95] F. Cassez and O. Roux. Compilation of the Electre reactive language into finite transition systems. *Theoretical Computer Science*, 146(1–2):109–143, July 1995.

[CR97] F. Cassez and O. Roux. Modelling and verifying reactive systems with event memorisation. Revised version submitted, 1997.

[Eme90] E. A. Emerson. *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier Science Publishers, 1990.

[FC87] A. Finkel and A. Choquet. Simulation of linear fifo nets by petri nets having a structured set of terminal markings. In *Proc. of the 8th European Workshop on Application and Theory of Petri Nets, Saragoza*, pages 95–112, 1987.

[FM97] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174:217–230, 1997.

[JJ93] T. Jéron and C. Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113:93–117, 1993.

[Lam83] L. Lamport. What good is temporal logic ? In *Information Processing'83. Proc. IFIP 9th World Computer Congress*, pages 657–668. North-Holland, September 1983.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[Pac87] J. K. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proc. of Protocol Specification, Testing and Verification, VII*, 1987.

[Que96] Y. M. Quemener. *Vérification de protocoles à espace d'états infini représentable par une grammaire de graphes.* PhD thesis, Université de Rennes 1 (FRANCE), 1996.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sut97] G. Sutre. Vérification de propriétés sur les automates à file réactifs produits par compilation de programmes Electre. Mémoire de DEA, Univ. Paris VII et Ecole Polytechnique, 1997.

[VFZ80] B. Vauquelin and P. Franchi-Zannettacci. Automates à file. *Theoretical Computer Science*, 11:221–225, 1980. Note.

A Proof of Lemma 3

Lemma 5. *Let R be the Reactive Fiffo Automaton associated with a Control Automaton $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$. We have:*

$$\mathcal{L}^\omega(\mathrm{Restr}(C)) \subseteq \mathrm{proj}_Q(\mathcal{L}^\omega(R)) \subseteq \mathcal{L}^\omega(C)$$

Proof. The inclusion $\mathrm{proj}_Q(\mathcal{L}^\omega(R)) \subseteq \mathcal{L}^\omega(C)$ is trivial according to the definition of the RFA R. Assume now that $q_0 q_1 \cdots q_n \cdots \in \mathcal{L}^\omega(\mathrm{Restr}(C))$. According to the definition of $\mathcal{L}^\omega(\mathrm{Restr}(C))$, there exists an infinite execution $\pi = q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \cdots q_n \xrightarrow{\alpha_n} q_{n+1} \cdots$ in $\mathrm{Restr}(C)$. Two cases may arise:

- for all $i \in \mathbb{N}, \alpha_i = \tau$. Then $(q_0, \varepsilon) \xrightarrow{\tau} (q_1, \varepsilon) \xrightarrow{\tau} (q_2, \varepsilon) \cdots (q_n, \varepsilon) \xrightarrow{\tau} (q_{n+1}, \varepsilon) \cdots$ is an infinite execution of R. In this way $(q_0, \varepsilon)(q_1, \varepsilon) \cdots (q_n, \varepsilon) \cdots \in \mathcal{L}^\omega(R)$, hence we have $q_0 q_1 \cdots q_n \cdots \in \mathrm{proj}_Q(\mathcal{L}^\omega(R))$.
- otherwise, there exists $i \in \mathbb{N}$ such that $\alpha_i \neq \tau$. Assume $m = \mathrm{Min}\{i \in \mathbb{N} \ / \ \alpha_i \neq \tau\}$. According to the definition of $\mathrm{Restr}(C)$, there exists $e \in \Sigma$ such that $\alpha_i = \tau$ for all $i \leq m-1$ and $\alpha_i = !e$ for all $i \geq m$. Let $(w_n)_{n \in \mathbb{N}}$ be the infinite sequence of finite words defined by:

$$\begin{cases} w_i = \varepsilon \text{ for all } 0 \leq i \leq m \\ \text{if } e \in \Sigma_S, \text{ then } w_i = e \text{ for all } i \geq m+1 \\ \text{if } e \in \Sigma_M, \text{ then } w_i = e^{i-m} \text{ for all } i \geq m+1 \end{cases}$$

  Let us remark that for all $i \in \mathbb{N}$, $(q_i, w_i) \xrightarrow{\alpha_i} (q_{i+1}, w_{i+1})$. Hence we have $(q_0, w_0)(q_1, w_1) \cdots (q_n, w_n) \cdots \in \mathcal{L}^\omega(R)$ and therefore $q_0 q_1 \cdots q_n \cdots \in \mathrm{proj}_Q(\mathcal{L}^\omega(R))$. ◻

B Proof of Lemma 4

Lemma 6. *Let $C = (Q, q_0, (\{!, ?\} \times \Sigma) \cup \{\tau\}, \rightarrow)$ be a Control Automaton. For every ω-sequence $\nu \in \mathcal{L}^\omega(C)$, there exists an ω-sequence $\nu' \in \mathcal{L}^\omega(\mathrm{Restr}(C))$ such that ν and ν' are equivalent modulo stuttering.*

Proof. Assume $\nu = q_0 q_1 \cdots q_n \cdots \in \mathcal{L}^\omega(C)$. According to the definition of $\mathcal{L}^\omega(C)$, there exists an infinite execution $\pi = q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \cdots q_n \xrightarrow{\alpha_n} q_{n+1} \cdots$ in C. We may assume that for all $i \in \mathbb{N}$, $\alpha_i \in (\{!\} \times \Sigma) \cup \{\tau\}$ as far as, from the definition of a Control Automaton, we may replace in π the labels in $\{?\} \times \Sigma$ by τ. Two cases may arise:

- for all $i \in \mathbb{N}$, there exists $j \geq i$ such that $\alpha_j = \tau$. Removing in π all the cycles of the form $q_i \xrightarrow{!e} q_{i+1}$, we obtain an infinite execution $\pi' = q_0 \xrightarrow{\alpha'_0} q'_1 \xrightarrow{\alpha'_1} q'_2 \cdots q'_n \xrightarrow{\alpha'_n} q'_{n+1} \cdots$ of C. As π' is an execution of C containing only τ-transitions, we get that π' is an execution of $\mathrm{Restr}(C)$.
- otherwise, there exists $i \in \mathbb{N}$ such that for all $j \geq i$, $\alpha_j \in \{!\} \times \Sigma$. Assume $m = \mathrm{Min}\{i \in \mathbb{N} \ / \ \forall j \geq i, \alpha_j \in \{!\} \times \Sigma\}$. Removing in π all the cycles of the form $q_i \xrightarrow{!e} q_{i+1}$ for $i \leq m-1$, we obtain an infinite execution $\pi' = q_0 \xrightarrow{\alpha'_0} q'_1 \xrightarrow{\alpha'_1} q'_2 \cdots q'_n \xrightarrow{\alpha'_n} q'_{n+1} \cdots$ of C. Now π' is an execution of C which may be written as $\pi' = \pi'_1 \cdot \pi'_2$ with:
  - π'_1 is an execution of C containing only τ-transitions ;
  - π'_2 is an execution of C containing only transitions which labels are in $\{!\} \times \Sigma$.

  Therefore, π' is an execution of $\mathrm{Restr}(C)$.

Hence, in both cases, $\nu' = q_0 q_1' \cdots q_n' \cdots \in \mathcal{L}^\omega(\mathrm{Restr}(C))$. Moreover, in both cases, π' is obtained from π by removing cycles. Consequently, ν and ν' are equivalent modulo stuttering. $\square$