Control of DES with Urgency, Avoidability and Ineluctability

Jean-Luc Béchennec, Didier Lime, Olivier Henri Roux

Control of DES with urgency, avoidability and ineluctability

1st Jean-Luc Béchennec
CNRS, LS2N
Nantes France
jean-luc.bechennec@ls2n.fr

2nd Didier Lime
Ecole Centrale de Nantes, LS2N
Nantes France
Didier.Lime@ec-nantes.fr

3rd Olivier (H.) Roux
Ecole Centrale de Nantes, LS2N
Nantes France
olivier-h.roux@ec-nantes.fr

***Abstract*— – The synthesis of controllers for reactive systems can be done by computing winning strategies in two-player games. Timed (game) Automata are an appropriate formalism to model real-time embedded systems but are not easy to use for controller synthesis for two reasons: i) timed models require the knowledge of the precise timings of the system (for example, if an action must occur in the future, the deadline of this occurence must be known) ii) in practice, the dense state space makes the computation of the controller often impossible for complex systems. This paper introduces an extension of untimed game automata with logical time. The new semantics introduces two new types of uncontrollable actions: *delayed actions* which are possibly avoidable, and *ineluctable actions* which will eventually happen if nothing is done to abort it. The controller synthesis problem is adapted to this new semantics. This paper focuses specifically on the reachability and safety objectives and gives algorithms to generate a controller. The usefulness of this new model is illustrated by a device driver synthesis example.**

***Index Terms*—Finite automata, Game theory, Controller synthesis, Timed systems**

I. Introduction

The theory of supervisory control has been well developed since about 30 years ago with the seminal works of [1]–[3]. It has become a basic paradigm for the control of discrete event systems (DES) modeled as finite state machines.

Since [2], different formalisms have been considered to model (un)controllable actions and control problems. Formulating control problems as two-player games has provided efficient solutions [4]. In this setting, the controller is modeled by a player and the environment by its opponent. Determining whether a controller exists amounts to determining if it can win and computing a winning strategy is equivalent to synthesizing a controller. However these turn-based games [4] where one player chooses their action before the other chooses theirs, are sequential and do not allow to model concurrency. Therefore, concurrent games [5]–[7] have been proposed, for which, at each round of the game, player 1 (the controller) and player 2 (the environment) independently and simultaneously choose moves, and both choices are used to determine the next state of the game.

Beside the controllable and uncontrollable actions used in untimed frameworks, controlled systems often rely on additional behavioral capabilities, based in particular on two important notions: delays and urgency. Without delays, we cannot express the fact that some actions (such as analog conversions, or emissions of messages on a communication bus) take time, and that the controller can perform actions during that time, even aborting the current environment operation. In that case the controller must make use of some kind of urgency. In addition, without urgency, we cannot model ineluctable behaviors (such as the eventual arrival of a product at the end of the conveyor belt on which it is placed) of the environment since, in untimed games, the environment is expected to play every move at its disposal to make the controller fail, including choosing not to play.

The model of timed automata [8] and timed games [9] is an appropriate formalism to express and model these timed properties. In a timed game, the time at which the two players (controller and environment), play their moves is taken explicitly into account. Their level of expressiveness and well-known controller synthesis techniques and tools [10], [11] allow the modeling of systems with complex interactions, while providing a formal proof on the behavior of the system. Yet, the computational complexity of the involved algorithms limits the size of the systems that can be addressed in practice.

Moreover, these timed formalisms require a good understanding of all the components of the system, including the knowledge of the timings of the actions of both players. These timings are rarely known precisely. Moreover, when these timings are known, or at least bounds on those timings, the complexity of the timed controller synthesis algorithms is still a problem.

Hence, it would be very interesting to derive a controller without explicit timed models (i.e. without precise timing quantification) while keeping the notion of urgency and delay. The behavior we would like to capture can be reduced into two types of uncontrollable actions:

- Delayed (avoidable) actions, which take time to complete or cannot happen immediately, such as writing to an external memory, sending a message on a bus, performing a specific computation on a hardware dedicated unit, etc. These actions usually come with some kind of abortion mechanism, so they are *avoidable* from a certain point of view. They are modeled in an explicit timed context by guard constraints with non-zero lower bounds on clocks.
- Ineluctable actions, which are known to happen in a nominal context: the end of a transmission or a conversion, or more generally an acknowledgement of the reception of a

command. An ineluctable action is guaranteed to happen eventually if nothing is done to abort it, which differs to the notion of fairness. In the untimed context, it is not sufficient to consider these actions as controllable. First, except if it is explicitly avoidable, an ineluctable action cannot be prevented by the controller, even if it leads to losing the game. Second, when there is a choice between two controllable actions, the controller chooses but when it is between two ineluctable actions, the environment chooses.

Our contribution.: We propose to extend the framework of untimed games with avoidability and ineluctability for uncontrollable actions and with urgency for controllable actions:

- an *avoidable (delayed) action* cannot happen immediately so that the controller can perform an *urgent action* to avoid it if needed.
- an *ineluctable* action is guaranteed to happen eventually if nothing else is done to abort it, and the controller may want to rely on it.

We revisit the controller synthesis problem for reachability and safety games in this context leading to what we call logical timed games.

This paper is organized as follows:

We first give in Section II, the basic definitions and notations for logical timed games. By using these notations, we justify our new model in Section III. Then, in Section IV, we solve the controller synthesis problem for logical timed games. In Section V and Section VI we respectively focus on reachability games and safety games. We discuss the complexity of the winning state computation algorithm implemented in our tool ROMÉO in section VII. Finally, in Section VIII we illustrate our method on a case study based on the Microchip CAN controller.

II. LOGICAL TIMED GAMES

In this section we propose a variant of the traditional untimed game automata with new logical-timed semantics capturing avoidability and ineluctability.

Let C and U be the two players respectively called controller and environment.

Definition 1 (Game structure)**.** *A game structure is a tuple* $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ *where*

- Q *is a set of states*
- $q_0 \in Q$ *is the initial state*
- A_C *and* A_U *are two disjoint sets of actions for the controller and the environment, respectively.*
- $\delta : Q \times (A_C \cup A_U) \times Q$ *a set of edges between states. We denote* $q \xrightarrow{a} q'$ *for* $(q, a, q') \in \delta$.

For the sake of simplicity, we assume the underlying finite automaton is deterministic.

In addition to this definition, we define $A_U^\star \subseteq A_U$ and $A_U^\diamond \subseteq A_U$ the subsets of *avoidable* and *ineluctable* actions, respectively. Note that these subsets are independent, and their intersection is not necessarily empty.

We also denote $A_U^{\overline{\star}\overline{\diamond}}$, $A_U^{\star\overline{\diamond}}$, $A_U^{\overline{\star}\diamond}$, $A_U^{\star\diamond}$, $A_U^{\overline{\star}}$ and $A_U^{\overline{\diamond}}$, the other subsets of A_U based on these two notions. As an example, $A_U^{\overline{\star}\overline{\diamond}}$ is the subset of actions of A_U which are not *avoidable* and not *ineluctable* and $A_U^{\overline{\diamond}}$ is the subset of actions of A_U which are not *ineluctable* but can be either *avoidable* or not.

A. Graphical notations

For the following figures, we will use the following notations illustrated in Figure 1:

- States are represented by circles, and the initial state is denoted q_0.
- Controllable transitions are represented by solid arrows.
- Uncontrollable transitions are represented by dashed arrows.
- Avoidable transitions start with a circle.
- Ineluctable transitions end with a double arrowhead.

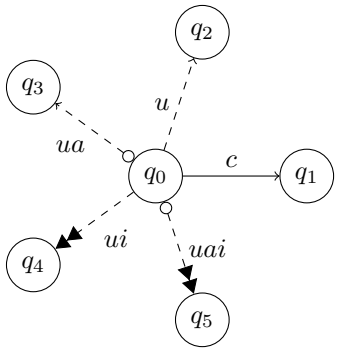

Figure 1. Graphical notation example: Here q_0 is the initial state, and $c \in A_C, u \in A_U^{\overline{\star}\overline{\diamond}}, ua \in A_U^{\star\overline{\diamond}}, ui \in A_U^{\overline{\star}\diamond}$ and $uai \in A_U^{\star\diamond}$.

B. Behaviors in game structures

The behaviors in game structures are timed behaviors, but only at a logical level, in which we distinguish immediate actions from others: we thus denote by Δ the set $\{\mathbf{0}, \bullet\}$, which represents the logical time at which an action is played. It can be instantaneous ($\mathbf{0}$), or unknown ($\bullet$). Semantically, $\langle a, \mathbf{0}\rangle$ means that the action a is performed *immediately*, whereas in $\langle a, \bullet\rangle$, the action is performed at an unknown time, possibly zero.

Avoidable actions: From a given state q an avoidable action u (as in Figure 4) can be prevented by any other action c from the same state q by the timed action $\langle c, \mathbf{0}\rangle$.

Ineluctable actions: From a given state, an ineluctable action $\langle u, \bullet\rangle$ will eventually happens if we do not do anything else from this state, that is to say if we wait long enough.

Avoidability and Ineluctability vs fairness: An ineluctable action can also be avoidable from a given state and the reachability game shown in Figure 2 is winning by doing $\langle c, \mathbf{0}\rangle$.

Moreover, like any non-avoidable uncontrollable action, an ineluctable (non-avoidable) action can not be prevented by a controllable action. On the other hand it can be prevented

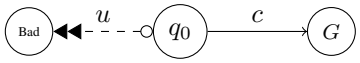

Figure 2. Avoidable action (even ineluctable) can be prevented by the controller

by an uncontrollable action that would perform a loop in null time. Hence the reachability games of Figure 3 are not winning showing the difference from the notion of fairness.

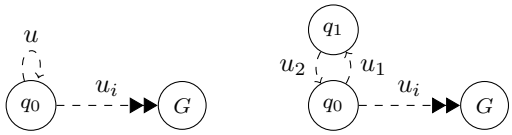

Figure 3. Ineluctability is not fairness.

C. Predecessor, successor and run

For $\Sigma \subseteq A_C \cup A_U$, we define the predecessor and successor functions $\mathrm{pre}_\Sigma : 2^Q \to 2^Q, \mathrm{suc}_\Sigma : 2^Q \to 2^Q$: Let $X \subseteq Q$, $\forall q \in Q, q \in \mathrm{pre}_\Sigma(X)$ iff $\exists a \in \Sigma$ and $q' \in X$, s.t. $q \xrightarrow{a} q'$, and $\forall q' \in Q, q' \in \mathrm{suc}_\Sigma(X)$ iff $\exists a \in \Sigma$ and $q \in X$, s.t. $q \xrightarrow{a} q'$. If $\Sigma = A_C \cup A_U$, we note $\mathrm{pre}(X)$ and $\mathrm{suc}(X)$

A *run* of a game structure is a sequence $q_0\ \langle a_1, t_1\rangle\ q_1\ \langle a_2, t_2\rangle\ q_2\ \ldots$ with $a_i \in A_C \cup A_U$, $t_i \in \Delta$, $q_i \in Q$, and such that $q_i \xrightarrow{a_i} q_{i+1}$ for all $i >= 0$. We denote by $\mathcal{R}$ the set of runs, and by $\overline{\mathcal{R}}$ the set of finite runs. Note that a finite run always ends with a state.

For a run $r \in \mathcal{R}$, we define $\mathrm{First}(r)$ the first state of r, $\mathrm{States}(r)$ the set of states which appear in r, and $\mathrm{Act}(r)$ the set of actions which appear in r. If $r \in \overline{\mathcal{R}}$, we define $\mathrm{Last}(r)$ the last state of r. We define the length $|r|$ of a run r as the size of the subsequence $\langle a_1, t_1\rangle\langle a_2, t_2\rangle \ldots$.

For $R \subseteq \mathcal{R}$ and $X \subseteq Q$, we denote by $R|_X$ the subset of R such that $\forall r \in R|_X, \mathrm{States}(r) \subseteq X$.

III. Justification for this new model

For the examples used in this section, we consider a reachability game (formally defined in Section V) starting in q_0 where the goal is to reach a state denoted G.

A. Avoidable action

The problem of avoidable (delayable) actions can be solved by using timed models such as timed automata. The avoidable actions can be translated directly into guards with a non-zero lower bound on clocks as depicted in Figures 4.a and 4.b. Hence, timed games [9], [12] allow to solve the controller synthesis problem for reachability or safety goal. In [13], the authors consider an abstraction of timed automata [8] where a transition τ represents the fact that some time elapse. The authors argue that the abstract timed transitions (τ) can be considered as controllable for the purposes of controller synthesis. This abstraction does not require explicit delays and if τ represents non-null elapsing of time, from a state q_0, an action τ followed by an uncontrollable action u is equivalent to an avoidable action u from q_0 as depicted in Figure 4.c. In [13], this abstraction is generated by a quotient of the timed game automata by a time-abstracting bisimulation and can be viewed as a game graph on which the complexity of the controller synthesis algorithm is quadratic in the size of the graph.

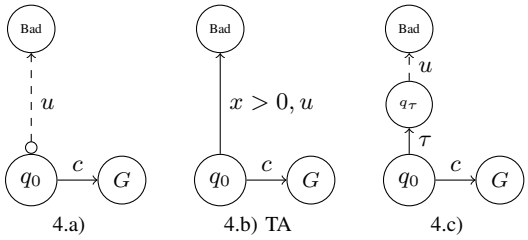

Figure 4. Avoidable uncontrollable action can be prevented by the controller

B. Ineluctable action

Ineluctable actions are known to happen in a nominal context: the end of a transmission or a conversion, or more generally an acknowledgement of the reception of a command.

Ineluctable action vs controllable action: In the untimed context, it is not sufficient to consider these actions as controllable. First, an ineluctable action cannot be prevented by the controller, even if it leads to losing the game (see Figure 5). Second, when there is a choice between two controllable actions, the controller chooses but when it is between two ineluctable actions, the environment chooses. For example, in Figure. 6, assume the emission of a message on a communication bus (action c). It can lead to an immediate success (action u_2) or it can first fail (action u_1) and can become a success later. It is ineluctable that either u_1 or u_2 occurs, but the choice between u_1 and u_2 does not depend from the controller which have to ensure that both states q_1 and q_2 are winning.

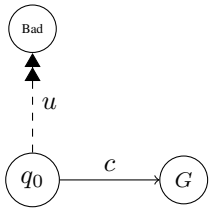

Figure 5. Ineluctable action cannot be prevented by the controller

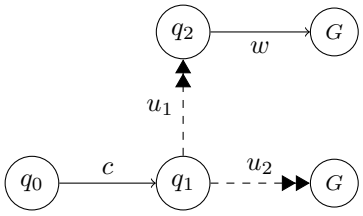

Figure 6. Ineluctable action is not selected by the controller

Ineluctable action vs timed action: In the timed context, ineluctability cannot be translated *as-is* into and from timed automata. We can use invariants on locations to force the environment to play, but this requires the knowledge of an upper bound on the delay, which is often not possible.

Moreover invariants apply to all players, including the controller, whereas ineluctable actions only restrict the behaviour of the environment. In [14], Timed Games are based on Timed Automata with invariants which are restricted to constraints of the form $x \leq k$ (where x is a clock and k is a constant). However, the environment can decide not to take action if an invariant requires to leave a state and the controller can do so.

Although the current work has not done so, it is possible to extend Timed Game Automata in order to take into account ineluctability for example by extending the notion of deadline or urgency [15]. However, reachability and safety timed games are decidable but are EXPTIME-complete and the symbolic states manipulated by the algorithms are *regions* or *zones* that are too powerful for untimed models and limit the size of the systems that can be addressed in practice.

Our model eliminates the need to put explicit values on time invariants and only restricts the behaviour of the environment and not that of the controller.

IV. Controller synthesis

In this section, we will solve the controller synthesis problem using our modified semantics. The goal is to derive a strategy for the controller to restrict the behavior of the game. Those strategies prescribe either a set of controllable moves that should be done either immediately, or with no timing restriction, or to wait and do nothing until some action happens, which is represented by an empty set.

Definition 2 (Strategy)**.** *A strategy s_i for player $i \in \{C, U\}$ is a function $s_i : \overline{\mathcal{R}} \to 2^{(A_i \times \Delta)}$. It is said to be* memoryless *if it only depends on the current state of the run, i.e. $s_i : Q \to 2^{(A_i \times \Delta)}$.*

We impose that if $\langle a, d\rangle \in s(r)$, then a is indeed possible from $\mathrm{Last}(r)$.

Definition 3 (Strategies with ineluctable and avoidable actions)**.** *Let $s_U : \overline{\mathcal{R}} \to 2^{(A_U \times \Delta)}$ be a strategy of the environment and let r be a run in the game, with $\mathrm{Last}(r) = q$.*

If there exists $a \in A_U^{\diamond}$, $d \in \Delta$, and a state q' such that $q \xrightarrow{\langle a, d\rangle} q'$ then $s_U(r) \neq \emptyset$.

If there exists $a \in A_U^{\star}$, $d \in \Delta$, and a state q' such that $q \xrightarrow{\langle a, d\rangle} q'$, and if $\langle a, d\rangle \in s_U(r)$, then $d \neq \mathbf{0}$.

Starting from a run consisting of some state (usually the initial state), both players inductively build a set of runs (because of non-determinism) by playing their strategy. Since we are interested in the strategies for the controller to win whatever the (legal) strategy of the environment, we directly define outcomes of a strategy of the controller, as the union over all strategies of the environment of all such sets of runs.

Definition 4 (Outcome)**.** *Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game structure, r one of its runs, and s_C a strategy for the controller. The* outcome $\mathrm{Outcome}(q, s_C)$ *of s_C from state q is the subset of $\mathcal{R}$ defined inductively by:*

- $q \in \mathrm{Outcome}(q, s_C)$
- *If $r \in \mathrm{Outcome}(q, s_C)$ is finite, $r' = r \xrightarrow{\langle a, d\rangle} q' \in \mathrm{Outcome}(q, s_C)$ if $r' \in \overline{\mathcal{R}}$ and one of the following holds true:*
  - *$a \in A_U^{\overline{\star}}$ and $d = \bullet$ if $\nexists (r \xrightarrow{a'} q''$ s.t. $\langle a', \mathbf{0}\rangle \in s_C(r))$, or $d = \mathbf{0}$ otherwise;*
  - *$a \in A_U^{\star}$, $d = \bullet$, and $\nexists (r \xrightarrow{a'} q''$ s.t. $\langle a', \mathbf{0}\rangle \in s_C(r))$.*
  - *$\langle a, d\rangle \in s_C(r)$.*
- *An infinite run belongs to $\mathrm{Outcome}(q, s_C)$ if all its finite prefixes also belong to $\mathrm{Outcome}(q, s_C)$*

Intuitively, we are interested in runs that are long enough to have a chance to fulfill the objective. Maximality distinguishes those runs that are the longest that the controller can produce, through its actions (possibly with diverting moves from the environment) or by relying on the ineluctable actions of the environment.

A run r is *maximal* in a set of runs R if either it is finite and there is no $a \in A_C \cup A_U^{\diamond}$, and no $q' \in Q$ such that $r \xrightarrow{a} q' \in R$, or it is infinite and none of its finite prefixes are maximal. We denote by $\mathrm{MaxOutcome}(q, s_C)$ the set of runs that are maximal in $\mathrm{Outcome}(q, s_C)$.

The control synthesis problem can be stated using objectives, or winning conditions. For a given game structure $\mathcal{G}$, a winning condition $C_{\mathcal{W}}$ is a set of allowed runs. We call the pair $(\mathcal{G}, C_{\mathcal{W}})$ a *game*.

In such a game, a strategy s for the controller is winning from state q if $\mathrm{MaxOutcome}(q, s) \subseteq C_{\mathcal{W}}$. A state q is winning if there exists a winning strategy from q. The game itself is winning if q_0 is winning.

V. Reachability games

A reachability objective of the controller is to force the game to reach a certain set of states. Formally:

Definition 5 (Reachability objective)**.**
Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game structure, and $\mathrm{Goal} \subseteq Q$ a set of goal states. The reachability objective $\mathrm{Reach}(\mathrm{Goal})$ for Goal is the set of runs r that are maximal in $\mathcal{R}$ and such that $\mathrm{States}(r) \cap \mathrm{Goal} \neq \emptyset$.

For example, for the game of Figure 7, the objective is to reach the state G and we have $\mathrm{Goal} = \{G\}$ and $\mathrm{Reach}(\mathrm{Goal}) = \{q_0\ \langle c, \bullet\rangle\ q_1\ \langle u, \bullet\rangle\ G\}$.

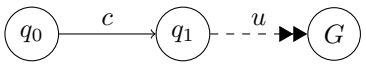

Figure 7. The objective is to reach the state G.

A. Computing the strategy

The computation of the strategy is obtained from the set of winning states. A state is winning for the controller if it is possible to reach a goal state from the strategy i.e. if the controller has a strategy to reach a goal state against all strategies of the environment. The main algorithm for computing winning strategies for reachability games is a backwards fixed-point algorithm over the *controllable predecessor* function.

Intuitively, a state s is a controllable predecessor of X if the following conditions are met:

- there is an action which is guaranteed to happen (either controllable or uncontrollable ineluctable) and leads to X;
- all other actions of the environment cannot prevent the game to reach a state in X.

Definition 6 (Controllable predecessors)**.**
Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game structure, and $X \subseteq Q$ a set of states. The controllable predecessors $\pi(X)$ of X is the subset of Q defined by:

$$\begin{aligned}\pi(X) = &\operatorname{pre}_{A_C}(X) \setminus \operatorname{pre}_{A_U^{\overline{\star}}}(\overline{X}) \\ &\cup \operatorname{pre}_{A_U^{\diamond}}(X) \setminus \operatorname{pre}_{A_U}(\overline{X})\end{aligned} \tag{1}$$

The two parts of the formula represent two different ways to win:

- if there is a controllable action from s to a state in X, all uncontrollable actions must either be avoidable, or also lead to states in X
- if there is an ineluctable uncontrollable action, all other uncontrollable actions must also lead to a state in X.

Given this new definition of π, the set of winning states is computed using the following classic backwards fixed-point algorithm: $\mathcal{W}_0 = \mathrm{Goal}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n)$. When it exists, the final fixed-point set is noted $\mathcal{W}$.

Algorithm 1 Winning states computation algorithm for reachability game

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow), \mathrm{Goal} \subseteq Q$
Output: $\mathcal{W}$

```
W ← Goal
while π(W) ⊈ W do
    W ← W ∪ π(W)
end while
return W
```

Lemma 1. *Let $(\mathcal{G}, C_{\mathcal{W}})$ be a reachability game. Let q_1 and q_2 be two states of $\mathcal{G}$. Let s_1 be a memoryless strategy that is winning from q_1 and s_2 be a memoryless strategy that is winning from q_2. Let Q_1 be the set of states of runs r in $\mathrm{Outcome}(q_1, s_1)$ such that $\mathrm{States}(r) \cap \mathrm{Goal} = \emptyset$ (i.e. the states that are traversed before reaching Goal).*

Let s be the memoryless strategy defined by: for all $q \in Q$, if $q \in Q_1$ then $s(q) = s_1(q)$, otherwise $s(q) = s_2(q)$. Then s is winning from both q_1 and q_2.

Proof. The fact that s is winning from q_1 is obvious. Now, from q_2 this is also quite straightforward. Let r be a run in $\mathrm{MaxOutcome}(q_2, s)$. If $\mathrm{States}(r) \cap Q_1 = \emptyset$ then $r \in \mathrm{MaxOutcome}(q_2, s_2)$ and therefore it eventually goes through Goal. Otherwise, we can write r as $r_2 r_1$, with $r_2 \in \mathrm{Outcome}(q_2, s_2)$, $\mathrm{Last}(r_2) \in Q_1$ and $r_1 \in \mathrm{Outcome}(\mathrm{Last}(r_2), s_1)$. Since $\mathrm{Last}(r_2) \in Q_1$, and since from there we follow the s_1, then for sure r_1 eventually goes through Goal. □

Lemma 2. *If $q \in \mathcal{W}_n$ (i.e. the value of $\mathcal{W}$ at the end of the n-th iteration of the while loop) then there exists a winning memoryless strategy from q that permits to win in n action steps or less.*

Proof. By induction on n.

Base case: before the first iteration of the while loop, $\mathcal{W}_0 = \mathrm{Goal}$, and $q \in \mathrm{Goal}$ implies that we have a strategy to win without doing anything. It is indeed equivalent to having a run with no action step from q to Goal.

Induction step: suppose the property holds for some $n \geq 0$. Let $q \in \mathcal{W}_{n+1}$. Then either $q \in \mathcal{W}_n$ or $q \in \pi(\mathcal{W}_n)$.

If $q \in \mathcal{W}_n$, then the induction hypothesis directly gives the result.

If $q \notin \mathcal{W}_n$ and therefore $q \in \pi(\mathcal{W}_n)$. Two more cases arise:

- either $q \in \operatorname{pre}_{A_C}(\mathcal{W}_n) \setminus \operatorname{pre}_{A_U^{\overline{\star}}}(\overline{\mathcal{W}_n})$: then there exists some $a \in A_C$ and $q_a \in \mathcal{W}_n$ such that $q \xrightarrow{a} q_a$. Let $\{b_1, \ldots, b_p\}$ be the set of uncontrollable, non-ineluctable actions possible in q and let q_i be the state such that $q \xrightarrow{b_i} q_i$ for all i. Then $q_i \in \mathcal{W}_n$, because $q \notin \operatorname{pre}_{A_U^{\overline{\star}}}(\overline{\mathcal{W}_n})$. By the induction hypothesis, we know that there are memoryless winning strategies s_a from q_a, and s_i for each of the q_i's. By Lemma 1, we can merge all those strategies in one memoryless strategy s'. Now, we exhibit a winning strategy: let s be the memoryless strategy such that $s(q) = \{\langle a, \mathbf{0}\rangle\}$ and $s(q') = s'(q')$ for all $q' \neq q$. Let us prove that s is indeed winning from q.
  Let r be a run in $\mathrm{MaxOutcome}(q, s)$. Note that the run consisting of only q cannot be maximal since $a \in A_C$. Therefore we have at least one action in r. Consider the first of those and call it x:
  - First suppose that $x \in A_C$. Then we must have $x = a$ because s says to play a in q. Now, remark that since $q \notin \mathcal{W}_n$, it is clear that it never appears in the outcomes of s' from q_a or any of the q_b's, so the outcomes of s and s' from those states are the same. Consequently, all maximal runs from q that start with a will eventually go through Goal because s' is winning.
  - Suppose now that $x \in A_U$. Then we must have $x \in A_U^{\overline{\star}}$ because s says to play immediately in q. Furthermore, the state reached by taking x is one of the q_i's defined above, from which s' is winning, and with the same argument as in the previous point, the maximal runs that start with x also eventually go through Goal.

- or $q \in \mathrm{pre}_{A_U^\diamond}(\mathcal{W}_n) \setminus \mathrm{pre}_{A_U}(\overline{\mathcal{W}_n})$. This case is fairly similar to the previous one: we know there exists some $a \in A_U^\diamond$ and $q_a \in \mathcal{W}_n$ such that $q \xrightarrow{a} q_a$. Let $\{b_1, \ldots, b_p\}$ be the set of uncontrollable actions possible in q and let q_i be the state such that $q \xrightarrow{b_i} q_i$ for all i. Then $q_i \in \mathcal{W}_n$, because $q \notin \mathrm{pre}_{A_U}(\overline{\mathcal{W}_n})$. By the induction hypothesis, we know that there are memoryless winning strategies s_a from q_a, and s_i for each of the q_i's. By Lemma 1, we can merge all those strategies in one memoryless strategy s'.
  Let s be the memoryless strategy such that $s(q) = \emptyset$ and $s(q') = s'(q')$ for all $q' \neq q$. We prove that s is winning from q.
  Let r be a run in $\mathrm{MaxOutcome}(q, s)$. Note that the run consisting of only q cannot be maximal since $a \in A_U^\diamond$. Therefore we have at least one action in r. Consider the first of those and call it x. Since the strategy says to wait, we cannot have $x \in A_C$. So $x \in A_U$, and the state reached by taking x is one of the q_i's above and we get the result with the same reasoning as before. □

Lemma 3. *If there exists a winning strategy from state q that permits to win in n action steps or less, then $q \in \mathcal{W}_n$.*

Proof. By induction on n.

Base case: If we can win without changing states, it must be the case that $q \in \mathrm{Goal} = \mathcal{W}_0$.

Induction step: suppose the property holds for some $n \geq 0$. Suppose that we have a winning strategy s from state q such that all runs in $\mathrm{MaxOutcome}(q, s)$ reach Goal in at most $n+1$ steps.

Consider the possible actions from q. If they all are uncontrollable and not ineluctable, or there is also controllable transitions but $s(q) = \emptyset$, then q is itself a maximal run and therefore $q \in \mathrm{Goal} = \mathcal{W}_0$, which implies that $q \in \mathcal{W}_{n+1}$. Otherwise:

- either there is at least a controllable action a in $s(q)$. Then it will be present in the outcome of s from q, leading to a state q_a, and then, since s is winning from q it is also from q_a, but in at most n steps. So we can apply the induction hypothesis and conclude that $q \in \mathrm{pre}_{A_C}(\mathcal{W}_n)$. By definition of the outcome, uncontrollable, non-ineluctable actions always appear in the outcome of s from q and, with the same reasoning, they all lead to states in $\mathcal{W}_n$. So $q \notin \mathrm{pre}_{A_U^\mp}(\overline{\mathcal{W}_n})$. And finally $q \in \mathcal{W}_{n+1}$.
- or there is no controllable action in $s(q)$ but there is at least an ineluctable uncontrollable action x possible from q. So x appears in the outcome of s from q and, as before, $q \in \mathrm{pre}_{A_U^\diamond}(\mathcal{W}_n)$. Similarly all possible uncontrollable actions appear in the outcome (since the strategy must be to wait) and, again as before, they therefore all lead to $\mathcal{W}_n$. Consequently, $q \in \mathcal{W}_{n+1}$. □

From Lemmas 2 and 3, we can deduce the following two results:

Theorem 1 (Completeness and Soundness)**.** *$q \in \mathcal{W}$ if and only if q is winning.*

Proof. If q is winning then there is a strategy from q that permits to win in a finite number of steps. So, by Lemma 3, $q \in \mathcal{W}_n$ for some n. Reciprocally, if $q \in \mathcal{W}$, it is in $\mathcal{W}_n$ for some n and, by Lemma 2, it is winning. □

Theorem 2 (Memoryless strategies)**.** *If the game is winning then it is winning with a memoryless strategy.*

Proof. If the game is winning then its initial state q_0 is winning with a strategy that permits to win in a finite number of steps then, by Lemma 3, q_0 is in $\mathcal{W}_n$ for some n and, by Lemma 2, there is therefore a winning *memoryless* strategy from q_0. □

The proof of Lemma 2 shows how one can effectively build a memoryless winning strategy when the game is winning: at each iteration, each new state added to $\mathcal{W}$ has either at least a controllable or an uncontrollable ineluctable transition to a state of $\mathcal{W}$ that was added in a previous iteration. The strategy can be the set (or any of its subsets) of those controllable actions. Those controllable actions are played at time $\mathbf{0}$ in the proof to keep it simple, but it is clear that if no delayable action to $\overline{\mathcal{W}}$ is possible, they can also be played at time $\bullet$.

It is clear that this strategy also ensures that the goal states are reached in the minimal number of steps possible.

Also note that, as always for reachability games, the canonical strategy that would always allow to move to any state in $\mathcal{W}$ is not winning in general since it might allow loops within $\mathcal{W}$, and thus infinite runs never reaching to goal states.

B. Reachability game example

Let us consider the reachability game $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ of Figure 8 where the objective is to reach the state G: $\mathrm{Goal} = \{G\}$. By applying the backward fixed-point algorithm 1: $\mathcal{W}_0 = \mathrm{Goal}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n)$, we obtain successively:

$\mathcal{W}_0 = \{G\}$, $\pi(\mathcal{W}_0) = \{q_4\}$, $\mathcal{W}_1 = \{G, q_4\}$, $\pi(\mathcal{W}_1) = \{q_3, q_4\}$, $\mathcal{W}_2 = \{G, q_3, q_4\}$, $\pi(\mathcal{W}_2) = \{q_2, q_3, q_4\}$, $\mathcal{W}_3 = \{G, q_2, q_3, q_4\}$, $\pi(\mathcal{W}_3) = \{q_0, q_2, q_3, q_4\}$, $\mathcal{W}_4 = \{G, q_0, q_2, q_3, q_4\}$, $\pi(\mathcal{W}_4) = \{q_0, q_2, q_3, q_4\}$

A winning memoryless strategy is $s(q_0) = \{\langle c_1, \mathbf{0}\rangle\}$, $s(q_3) = \{\langle c_2, \mathbf{0}\rangle\}$, $s(q_4) = \{\langle c_3, \bullet\rangle\}$ and $s(q_1) = s(q_2) = s(G) = \emptyset$.

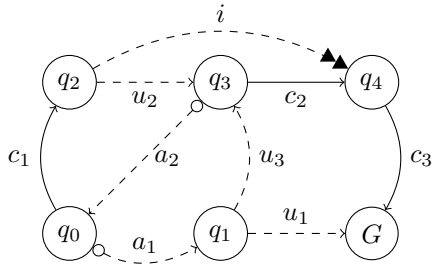

Figure 8. A reachability game. The objective is to reach the state G.

VI. Safety game

A safety objective for the controller is to force the game to stay in a specified set of states, or equivalently, to avoid a set of states.

Definition 7 (Safety objective)**.**
Let $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow)$ *be a game structure and* $\mathrm{Safe} \subseteq Q$ *a set of* safe *states. The safety objective for* Safe *is the set of all infinite maximal runs* r *of* $\mathcal{G}$ *such that* $\mathrm{States}(r) \subseteq \mathrm{Safe}$.

Note that we exclude finite maximal runs from the objective because we do not want the controller to win by deadlocking or by reaching an uncontrollable livelock i.e. a set of states with no outgoing controllable transition. It means that when the environment decides to not play, the controller must be able to move. Hence, the safety games of Figure 9 where all the states are in the set of *safe* states, are loosing. Indeed, for the game of Figure 9.a, we have $q_0 \notin \pi(\{q_0\})$ and for the games of Figures 9.b and 9.c, we have $q_1 \notin \pi(\{q_0, q_1, q_2\})$ meaning that the environment can block in q_1 (by not playing u_1 since it is not ineluctable) and to avoid q_1, the controller must block in q_0. A contrario, the games of Figure 10 are winning.

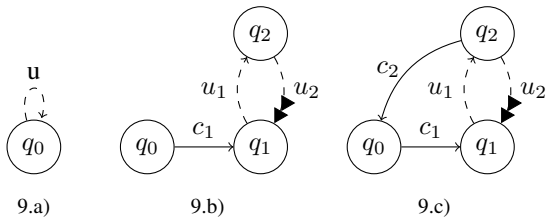

Figure 9. All the states are safe but the games are not winning.

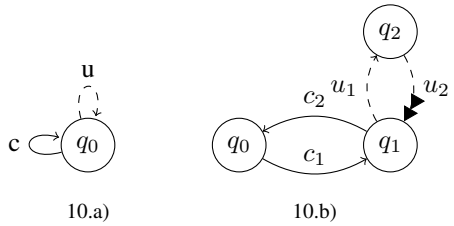

Figure 10. All the states are safe and the games are winning.

A. Computation of the strategy

The strategy is computed from the set of winning states. A state is winning for the controller if it is possible to force the game to stay in Safe.

Given our new definition of π, the set of winning states for the controller is computed using the following classic backwards fixed-point algorithm: $\mathcal{W}_0 = \mathrm{Safe}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cap \pi(\mathcal{W}_n)$.

When it exists, the final fixed-point set is noted $\mathcal{W}$.

Like in Section V, we can prove the soundness and completeness of Algorithm 2, by proving the following two lemmas. The proof are very similar to those of Section V and are therefore omitted.

Algorithm 2 Winning states computation algorithm for safety games

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow), \mathrm{Safe} \subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow \mathrm{Safe}$
  while $\mathcal{W} \not\subseteq \pi(\mathcal{W})$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cap \pi(\mathcal{W})$
  end while
  return $\mathcal{W}$

Lemma 4. *If* $q \in \mathcal{W}_n$ *then there exists a memoryless strategy* s *such that for any prefix* r *of length* n *of a run in* $\mathrm{MaxOutcome}(q, s)$*, we have* $\mathrm{States}(r) \subseteq \mathrm{Safe}$.

Lemma 5. *If there exists a strategy* s *and a run* r *such that for any prefix* r' *of length* n *of a run in* $\mathrm{MaxOutcome}(q, s)$*, we have* $\mathrm{States}(r') \subseteq \mathrm{Safe}$*, then* $\mathrm{Last}(r) \in \mathcal{W}_n$.

From those two lemmas, the main results follow:

Theorem 3 (Completeness and Soundness)**.** $q \in \mathcal{W}$ *if and only if* q *is winning.*

Proof. If q is winning then there is a strategy s from q such that prefixes r of any length of runs in $\mathrm{MaxOutcome}(q, s)$ are such that $\mathrm{States}(r) \subseteq \mathrm{Safe}$. So, by Lemma 3, $q \in \mathcal{W}_n$ for all n and, in particular, $q \in \mathcal{W}$. Reciprocally, if $q \in \mathcal{W}$, let n be such that $\mathcal{W} = \mathcal{W}_n$, then for all $m \geq n$, $q \in \mathcal{W}_n$. So for all $m \geq n$, there is a memoryless strategy from q that stays in Safe for at least m steps. Since there is only a finite number of states and of actions, there is only a finite number of memoryless strategies on the game structure. So there is one that is winning for an infinity of $m \geq n$, which implies that no prefix of the maximal runs in its outcome ever goes out of Safe, and therefore that strategy is winning. □

Theorem 4 (Memoryless strategies)**.** *If the game is winning then it is winning with a memoryless strategy.*

For safety games, and following the previous results, it is clear that moving to any winning state is always a winning strategy for the controller. We define a canonical memoryless strategy $s^s : \mathcal{W} \rightarrow 2^{(A_C \times \Delta)}$ that does exactly that:

Let $s^s(q) = \{\langle a, d\rangle | a \in A_C, q \xrightarrow{a} q' \Rightarrow q' \in \mathcal{W}\}$, with $d = \mathbf{0}$ if $\exists a' \in A_U^\star, q'' \notin \mathcal{W}$ and $d = \bullet$ otherwise.

Permissive strategies are a key notion in supervisory control [2]. In reactive synthesis, permissiveness is measured in terms of the set of behaviours allowed by the strategy [16]. Hence most permissive strategies need not exist, depending on the type of winning objectives.

Theorem 5. *Strategy* s^s *is the most permissive winning strategy for the safety objective* Safe*, i.e, for all winning strategies* s'*,* $\mathrm{Outcome}(q_0, s') \subseteq \mathrm{Outcome}(q_0, s^s)$.

Proof. Ab absurdo. Assume that s^s is not the most permissive winning strategy. Then there exists a winning strategy s' and a run in $\mathrm{Outcome}(q_0, s') \backslash \mathrm{Outcome}(q_0, s^s)$. Let r be the longest prefix of that run that is in $\mathrm{Outcome}(q_0, s^s)$. Let $q = \mathrm{Last}(r)$. Then we have, for some action a, $q \xrightarrow{\langle a, d \rangle} q' \in \mathrm{Outcome}(s', q)$ and $q \xrightarrow{\langle a, d \rangle} q' \notin \mathrm{Outcome}(s^s, q)$.

We must have $q' \in \mathcal{W}$ or s' cannot be winning because of Theorem 3. Then, by definition of s^s, it is not possible that $a \in A_C$, so it must be the case that $a \in A_U$. And, by definition of Outcome, the only possibility is that $a \in A_U$, $d = \bullet$, there is an action $b \in A_C$, such that $\langle b, \mathbf{0} \rangle \in s^s(r)$, and there is no such action in $s'(r)$. This in turn implies that there is a third action $c \in A_U^\star$ and a state q'' such that $q \xrightarrow{\langle c, \bullet \rangle} q''$ and $q'' \notin \mathcal{W}$. Since there is no immediate controllable action in $s'(r)$ then clearly $r \xrightarrow{\langle c, \bullet \rangle} q'' \in \mathrm{Outcome}(q_0, s')$, which, by Theorem 3, contradicts the fact that s' is winning. □

B. Safety game example

Let us consider the safety game $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow)$ of Figure 11 where the objective is to avoid the state B. Hence $\mathrm{Safe} = \{q_0, q_1, q_2\}$ is the set of *safe* states.

By applying the backward fixed-point algorithm 2: $\mathcal{W}_0 = \mathrm{Safe}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cap \pi(\mathcal{W}_n)$, we obtain successively:

$\mathcal{W}_0 = \{q_0, q_1, q_2\}$, $\pi(\mathcal{W}_0) = \{q_0, q_1\}$, $\mathcal{W}_1 = \{q_0, q_1\}$, $\pi(\mathcal{W}_1) = \{q_0, q_1\}$.

The most permissive memoryless strategy is $s(q_0) = \emptyset$ and $s(q_1) = \{\langle c, \mathbf{0} \rangle\}$.

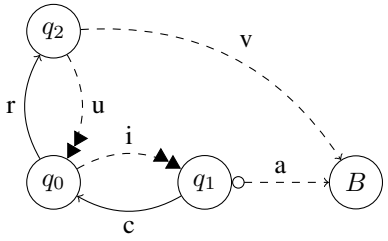

Figure 11. A winning safety game. The objective is to avoid the state B.

VII. Complexity and Implementation

While the algorithms we give are well-suited for pedagogical exposition and proofs, and possibly for an implementation using symbolic decision diagrams based representations of sets of states, they are not optimal for an explicit enumeration of states. Nonetheless, plugging our definition of the controllable predecessors operator π into the untimed algorithm of [17], we can compute the winning states for reachability, or their complement for safety, in time linear with respect to the number of edges in the automaton.

Based on this latter algorithm, we have implemented the computation of the winning states and the synthesis of the strategy in our tool Roméo [18]. With its textual input language, Roméo handles a model called Clock Transition Systems (CTS) [19] which encompasses both finite automata and Petri Nets. We have extended CTS with controllable, uncontrollable, avoidable and ineluctable actions in order to model logical timed games. The CTS can be generated from the Roméo GUI.

VIII. Case study

Device drivers synthesis is a good example of logical time game controllers synthesis. Here the environment is i) the hardware device along with its connections to external systems: communication networks, analog signals, etc and ii) the application using the driver. In the former case uncontrollable actions are interrupts that are triggered to signal, for instance, the availability of a data in a hardware buffer. In the latter case they are requests made by the application. In both cases exact timings are unknown since they depend on the actual hardware and on the execution time of the actual binary program which is not available yet. However some time related rules are known like the inter-arrival time of messages on a communication network or the time between two interrupts of a timer for instance. So, when reacting to an uncontrollable action the controller has time to perform its task before the arrival of the next same uncontrollable action. In such case the second action is avoidable.

A. CAN controller driver modelling

The device chosen for the case study is the Microchip CAN controller available in PIC18Cxx8 microcontroller family [20]. This CAN controller features two receive buffers, RXB0 and RXB1 and three transmit buffers, TXB0, TXB1 and TXB2. Each of these buffers can hold a complete CAN message. For the sake of simplicity, we consider only one transmit buffer, which is called TXB, in this case study. The device is configured so that i) when a message is received from the bus it is put in one of the receive buffers and an interrupt is asserted. ii) when a message is written to the transmit buffer the device sends it as soon as possible and asserts an interrupts to notify TXB has just been emptied.

The model of the driver is presented at Figure 12. We added boolean variables: PW (Pending Write), RXB0IF (RXB0 Interrupt flag), RXB1IF (RXB1 Interrupt Flag), TXBIF (TXB Interrupt Flag) to simplify the drawing of the model. The driver is cut into two parts: the part that is executed in user mode, represented by white states and the part that is executed in the interrupt handler represented by gray states. In addition a black bad state that has to be avoided by the controller is added. Starting from the no_init state the device can be configured as described above and the driver waits requests in the wait state. From there three uncontrollable actions, corresponding to a write request from the application (*write_TXB*) or the arrival of a message in one of the receive buffer (*can_it_0* or *can_it_1*), may occur and the corresponding boolean variable is set accordingly. From the write state we find again the two uncontrollable actions corresponding to the arrival of a message and also an ineluctable action which is done by the device when TXB is emptied. The state event represents the entry point of the device interrupt handler. From there the

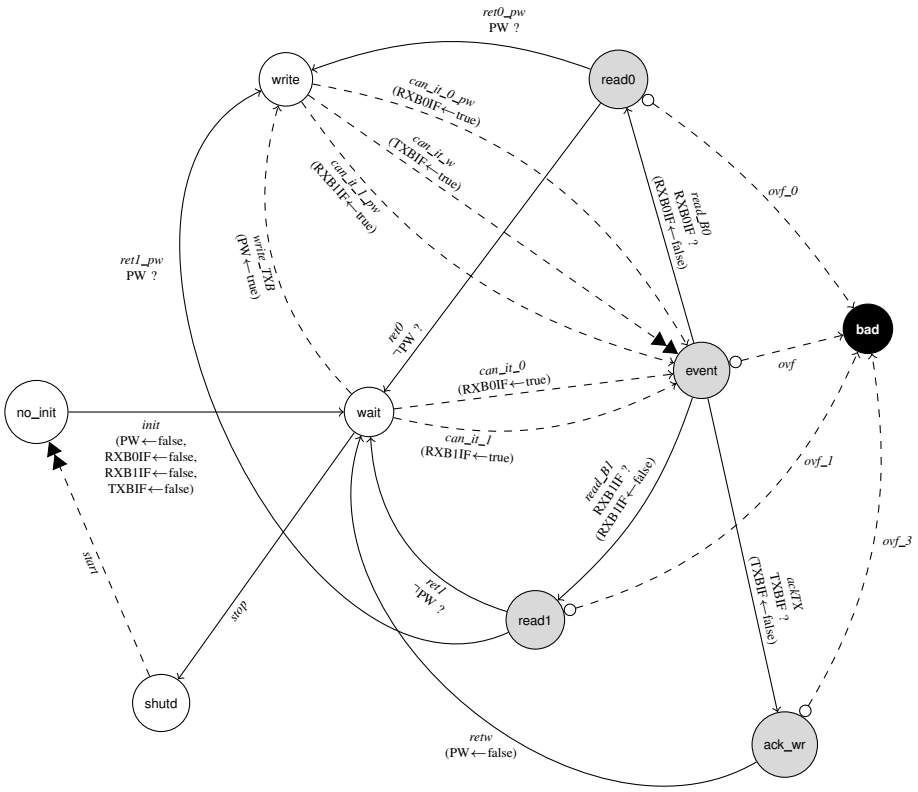

Figure 12. PIC18Cxx8 CAN controller driver model. Guards are noted with a '?', negations are noted with a ¬ and updates are noted inside parenthesis with a ←.

controller can play the actions corresponding to the processing of the event: read the receive buffer which has been filled (read0 or read1) or acknowledge the emptying of the transmit buffer (ack_write).

During the execution of the interrupt handler uncontrollable actions are avoidable because i) device interrupts are masked ii) the controller has enough time to play its actions before the occurence of a new interrupt.

B. Winning strategy

We used our tool ROMÉO [18] for the modelling of this case study and to compute the winning states and the synthesis of the strategy. We first verify that the safety property, where BAD is never reached, holds. But for safety ROMÉO actually computes the complement of the fixed-point given in section VI and therefore computes a strategy for the environment to falsify the property. Of course it finds none. So in order to get the strategy for the controller we also verify a reachability objective.

To express this objective, we need to add two boolean variables called PLAYED_wait, which is set when the environment plays an action to leave state wait, and PLAYED_write which is set when the environment plays an action to leave state write. That way staying in state x (PLAYED_x is false) and returning to this state after the environment has played (PLAYED_x is true) can be distinguished. An additional state is also added, shutd. The shutd state models the fact that the system may be off. The *start* transition is the switching on of the system and the *stop* transition is the switching off of the system. If the environment decides to not play any action, shutdown will be reachable eventually. The goal of the controller is to reach one of the following states:

- shutdown,
- wait with PLAYED_wait = true,
- write with PLAYED_write = true

Table I
MEMORYLESS STRATEGY

state	variables	play	next
no_init	–	$\langle init, \bullet \rangle$	wait
wait	–	–	wait
write	–	–	write
write	–	–	event
event	¬RXB0IF, ¬RXB1IF, TXBIF	$\langle ackTX, \mathbf{0} \rangle$	ack_wr
event	RXB0IF, ¬RXB1IF, ¬TXBIF	$\langle read\_B0, \mathbf{0} \rangle$	read0
event	¬RXB0IF, RXB1IF, ¬TXBIF	$\langle read\_B1, \mathbf{0} \rangle$	read1
ack_wr	–	$\langle retw, \mathbf{0} \rangle$	wait
read0	PW	$\langle ret0\_pw, \mathbf{0} \rangle$	write
read0	¬PW	$\langle ret0, \mathbf{0} \rangle$	wait
read1	PW	$\langle ret1\_pw, \mathbf{0} \rangle$	write
read1	¬PW	$\langle ret1, \mathbf{0} \rangle$	wait
shutd	–	–	shutd

The strategy is summarized in Table I. Starting from no_init the controller must play *init* to reach a winning state. In wait, if the environment plays *can_it_0* or *can_it_1*, PLAYED

is set and the controller has to play immediately *read_B0 – ret0* or *read_B1 – ret1*, respectively, to go back to wait. If the environment plays *write_TXB*, both PW and PLAYED are set. From write the environment may choose to play *can_it_0_pw* or *can_it_1_pw*. Then the controller has to play immediately *read_B0 – ret0_pw* or *read_B1 – ret1_pw*, respectively, to go back to write. If the environment decides to not play uncontrollable actions, inevitably *can_it_w* happens and the controller returns immediately to state wait by playing *ackTX – retw*.

IX. Conclusion

We have presented an extension of finite automata with logical time. This extension introduces two new properties of uncontrollable actions that extend the model of the environment:

- the *delayed action* cannot happen instantaneously so that the controller may preemptively perform another action if needed.
- the *ineluctable* action is guaranteed to happen eventually, and the controller can hence rely on it.

This model combines some of the expressiveness of timed games, with the simplicity of finite automata. It allows an easier implementation of these models, more suitable to embedded real-time systems. We have adapted the notion of control, reachability and safety games for this extension and defined and proved algorithms to solve these problems in the general case. Finally we have implemented the computation of the winning states and the synthesis of the strategy in our tool Roméo.

Further work includes extending the approach to more complex control objectives, such as Büchi conditions, and deal with concurrent behaviors modelled by networks of finite automata.

References

[1] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," in *Decision and Control, 1984. The 23rd IEEE Conference on*, vol. 23, Dec 1984, pp. 1073–1080.

[2] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[3] C. Golaszewski and P. Ramadge, "Control of discrete event processes with forced events," in *Proceedings of the 26th Conference on Decision and Control*, Dec. 1987.

[4] W. Thomas, "On the synthesis of strategies in infinite games," in *STACS 95*. Springer, 1995, pp. 1–13.

[5] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, "Strategy improvement for concurrent reachability and safety games," *CoRR*, 2012.

[6] L. De Alfaro, T. A. Henzinger, and R. Majumdar, "Symbolic algorithms for infinite-state games," in *CONCUR 2001—Concurrency Theory*. Springer, 2001, pp. 536–550.

[7] L. de Alfaro, T. A. Henzinger, and O. Kupferman, "Concurrent reachability games," *Theoretical Computer Science*, vol. 386, no. 3, pp. 188 – 217, 2007.

[8] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[9] L. De Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga, "The element of surprise in timed games," in *CONCUR 2003-Concurrency Theory*. Springer, 2003, pp. 144–158.

[10] K. Altisen and S. Tripakis, "Tools for controller synthesis of timed systems," in *2nd Workshop on Real-Time Tools (RT-TOOLS'2002)*, july 2002.

[11] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *Computer Aided Verification*. Springer, 2007, pp. 121–125.

[12] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *STACS 95*. Springer, 1995, pp. 229–242.

[13] S. Tripakis and K. Altisen, "On-the-fly controller synthesis for discrete and dense-time systems," in *In FM'99, volume 1708 of LNCS*. Springer Verlag, 1999, pp. 233–252.

[14] Th. Chatain, A. David, and K. G. Larsen, "Playing games with timed games," in *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, A. Giua, M. Silva, and J. Zaytoon, Eds., Zaragoza, Spain, Sep. 2009.

[15] S. Bornot and J. Sifakis, "An algebraic framework for urgency," *Inf. Comput.*, vol. 163, no. 1, pp. 172–202, 2000.

[16] J. Bernet, D. Janin, and I. Walukiewicz, "Permissive strategies: from parity games to safety games," *RAIRO - Theoretical Informatics and Applications (RAIRO: ITA)*, vol. 36, pp. 261–275, 2002.

[17] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Efficient on-the-fly algorithms for the analysis of timed games," in *CONCUR 2005–Concurrency Theory*. Springer, 2005, pp. 66–80.

[18] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez, "Romeo: A parametric model-checker for Petri nets with stopwatches," in *TACAS 2009*, ser. Lecture Notes in Computer Science, vol. 5505. York, UK: Springer, Mar. 2009, pp. 54–57.

[19] C. Jard, D. Lime, and O. H. Roux, "Clock Transition Systems," in *21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)*, Berlin, Germany, Sep. 2012.

[20] Microchip, *PIC18CXX8 Data Sheet (DS30475A). High-Performance Microcontrollers with CAN Module.*, http://ww1.microchip.com/downloads/en/DeviceDoc/30475a.pdf, 2000.