Reactive Embedded Device Driver Synthesis using Logical Timed Models

Julien Tanguy[1,2], Jean-Luc Béchennec[2], Mikaël Briday[2] and Olivier-H Roux[2]
[1]*See4sys Technologie, Espace Performance La Fleuriaye, 44481 Carquefou CEDEX, France*
[2]*LUNAM Université, Ecole Centrale de Nantes, IRCCyN UMR CNRS 6597 1 rue de la Noë, 44321 Nantes, France*

Keywords: Real-Time Systems, Formal Modeling, Control, Logical Time, Software Synthesis.

Abstract: The critical nature of hard real-time embedded systems leads to an increased usage of Model Based Design to generate a correct-by-construction code from a formal specification. If Model Based Design is widely used at application level, most of the low level code, like the device drivers, remains written by hand. Timed Automata are an appropriate formalism to model real time embedded systems but are not easy to use in practice for two reasons i) both hardware and software timings are difficult to obtain, ii) a complex infrastructure is needed for their implementation. This paper introduces an extension of untimed automata with logical time. The new semantics introduces two new types of actions: *delayed action* which are possibly avoidable, and *ineluctable action* which will happen eventually. The controller synthesis problem is adapted to this new semantics. This paper focuses specifically on the reachability problem and gives an algorithm to generate a controller.

1 MOTIVATION

Nowadays embedded systems realize many critical functions. From avionics to automotive electronics control systems, the software has grown in size and its complexity has become more important.

The increasing number of functions and complexity of such systems make software development very difficult due to a high level of concurrency and to the hard real-time context. Many safety-critical systems now cooperate in real-time on several ECUs across many communication links.

Due to the critical nature of these systems, software vendors have to prove the functional safety of their systems. Unit and functional tests are common requirements, but they do not guarantee the software is bug free. The automotive community is now pushing for a more formal verification of the systems' behavior.

Model Based Design (MBD) methodologies are now widely used in the industry and are a way to address the complexity of these systems. Instead of writing the code by hand the engineer models the system to control and builds a model of the application. The models can be tested and simulated. They can also be verified if their spatial and temporal complexity and their size remain practicable. At last the code is generated automatically from the model.

In the automotive industry the AUTOSAR standard (Kirschke-Biller, 2011) proposes a framework for MBD. It specifies an architecture and a methodology for the design of such systems, based on current development methods, design best practices and applicable international standards (such as the ISO26262 norm (The International Organization for Standardization, 2011)).

However if MBD is used at the application level, basic software especially device drivers remain coded by hand and are more prone to have bugs. For example in the AUTOSAR standard the basic software is a set of modules. These modules are usually defined as a core of basic functionalities which can do everything and some configuration code which selects or refines the previously defined behaviors and wrapper code to encapsulate the module functionalities in APIs — see Figure 1. The configuration code is usually generated at compile-time and compiled along the core code, but the specification allows a post-compilation configuration which is passed to the core code by pointers.

This high level of configurability at every level increases greatly the complexity of such systems; they usually require multiple modules and abstraction levels. It can also result in a lot of dead code and if

Tanguy J., Béchennec J., Briday M. and Roux O..
Reactive Embedded Device Driver Synthesis using Logical Timed Models.
DOI: 10.5220/0005040101630169
In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2014), pages 163-169
ISBN: 978-989-758-038-3
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

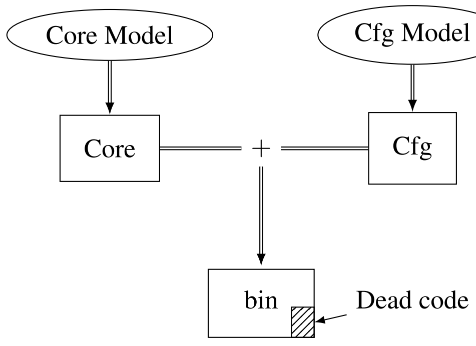

Figure 1: Current development methodology.

the configuration is not perfectly tuned to the application demands, so that unnecessary behaviors make it into the code and may be executed. This comes at the cost of decreased performance and greater memory footprint, in terms of stack size, ROM and RAM usage. The consistency of the configuration must also be checked in order to be sure that the driver cannot behave in an unspecified way.

The proposed approach. Instead of doing complex verification on existing systems, we propose to use an MBD approach to generate the device driver. However, to reduce the flaws of the current methodology, we combine the use of formal models, the lifting of configuration at the model level and well-known controller synthesis techniques (Ramadge and Wonham, 1989) to generate correct-by-construction software.

Given the real-time nature of these systems, the model of timed automata (Alur and Dill, 1994) and timed games (De Alfaro et al., 2003) is an appropriate formalism to express and model the required timed properties. In a timed game, we model two players, the controller and the environment, playing moves at a certain time on a shared model called the board. The board represents all possible states of the two players and the possible actions available at all times. In this case, the controller is the device driver, and the environment is the hardware device along with its environment (communication bus, analog signals, etc.). Moves played by the device driver are also known as controllable actions and moves played by the hardware device and its environment are known as uncontrollable actions.

Their level of expressiveness and well-known controller synthesis techniques and tools (Behrmann et al., 2007) allow the modeling of complex systems, while providing a formal proof on the behavior of the systems.

However, these timed models require a good understanding of all the components, including the knowledge of the timings of both players. These timings are rarely known: the hardware timings are not always described exhaustively, and the software execution times are rarely known precisely.

In addition, embedded systems hardware resources (computing power, memory, etc.) are limited. The basis of timed controlled, namely explicit clocks, are very expensive to use in a critical embedded environment. Implementation wise, the infrastructure needed to implement these clocks in a timed controller is not acceptable for implementing real-time systems at low level such as device drivers. In order to function properly, a timed driver may need an exact timing. These exact timings do translate into low-level software to hardware clocks triggering interrupts each time a controller needs to wait during an arbitrary amount of time. Even with this, a controller cannot guarantee all the timing constraints, because interrupts might have been masked for a certain computation.

Our contribution. In this paper, we explore another way of modeling embedded systems, starting from peripheral devices. We would like to derive a controller — a device driver — without explicit timed models. However, the untimed automata framework does not have the necessary level of expressiveness in order to generate useful drivers.

Beside the controllable and uncontrollable actions used in untimed games, device drivers rely on additional behaviors of the device in order to work. These behaviors can be reduced into two types of uncontrollable actions:

- Delayed actions, that 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 come usually with some kind of abortion mechanism, so they are *avoidable* in a certain point of view. They are modeled in an explicit timed context by guards with lower bounds on clocks as constraints.
- Ineluctable actions, that 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. These can be modeled using invariants on states, however in the explicit timed context we need to know an upper bound on the delay, which is not always possible with a quantitative approach.

We propose to extend the semantics of untimed games with two properties of uncontrollable actions:

avoidability — an avoidable action cannot disrupt the behavior of the driver — and ineluctability — the driver can rely on these actions to happen.

We will derive the traditional controller synthesis problem for reachability games in this context.

2 LOGICAL TIMED GAME AUTOMATA

In this section we propose a variant of the traditional untimed game automata with new logical-timed semantics. The modified semantics of actions let us express two important notions of timed automata: delay and urgency. Without these notions, it is impossible to write an untimed controller while keeping the *element of surprise*. The element of surprise is necessary in this context: a device driver should be able to react to any hardware interrupt when it happens.

Without delays, we cannot express the fact that an hardware action (such as analog conversions, or emitting a message on a communication bus) takes times, and as such the device driver can perform actions, even aborting the current operation (see Figure 3). Without urgency, we cannot model any predicted behavior of the device. In these kind of games, the device is expected to play every move at its disposal to make the controller fail, including choosing not to play, even if the controller cannot play any move in the current state. As such, it would not make sense to wait for something to happen (see Figure 4).

As we do not want real-valued clocks, we define a logical time semantics for game automata. It is based on the classical definition $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ 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 $(q, a, q') \in E$.

In addition to this definition, we also define $A_U^a \subseteq A_U$ and $A_U^{\diamond} \subseteq A_U$ the subsets of *avoidable* and *ineluctable* actions, respectively. Note that these subsets are independent, and may or may not intersect.

These subsets reflect a finer model of the environment: avoidable actions are actions which cannot happen instantaneously, such that we are able to avoid or abort them before they have a chance to happen. Ineluctable actions, on the other hand, can reasonably expected to happen, unless a major failure occurs.

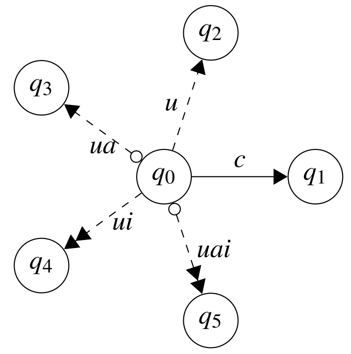

Figure 2: Graphical notation example: Here q_0 is the initial state, and $c \in A_C, u \in A_U^{\overline{a}\overline{\diamond}}, ua \in A_U^{a\overline{\diamond}}, ui \in A_U^{\overline{a}\diamond}$ and $uai \in A_U^{a\diamond}$.

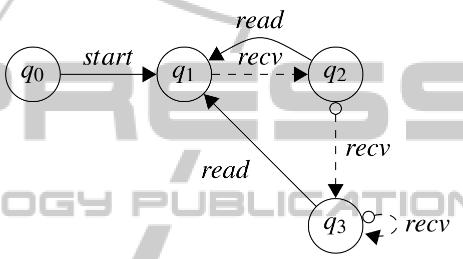

Figure 3: Rx part of a communication device: the uncontrollable actions *recv* represent the reception of a message. In q_1 the data register is empty, in q_2 it is full and in q_3 it is overwritten. Because the transmission of a message takes time, two immediate and consecutive receptions cannot happen. In this situation, we are able to express the ability for the driver to perform a *read* action between two *recv*, and avoid q_3.

For the rest of this paper, we will consider an arbitrary game $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$. For the following figures, we will use the following notations:

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

2.1 Definitions

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

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

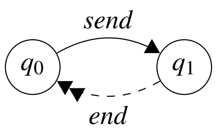

Figure 4: Tx part of a communication device: the uncontrollable action *end* represent the end of transmission interrupt. Because of the way the bus may be structured, or saturated, we do not know an upper bound on the reaction time, only that it will happen eventually.

Σ and $q' \in X$, s.t. $q' \xrightarrow{a} q$. If $\Sigma = A_C \cup A_U$, we note $\mathrm{pre}(X)$ and $\mathrm{suc}(X)$

For $i = C, U$, we define $\Gamma_i : Q \to 2^{A_i} \cup \{\perp\}$, with $\perp \notin A_i$ the enabling conditions. For $q \in Q$, $\Gamma_i(q)$ is the set of available moves for player i. The special action $\perp$ represents the act of *choosing not to play* in this state.

We denote by Δ the set $\{\mathbf{0}, \bullet\}$. It represents the logical time at which an action is played. It can be instantaneous ($\mathbf{0}$), or unknown ($\bullet$).

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

For $r \in \mathcal{R}$, 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 $\mathcal{R}^s$ the set of reliable runs, which do not depend on uncontrollable actions:

$$r \in \mathcal{R}^s \Leftrightarrow \mathrm{Act}(r) \cap A_U^{\diamond} = \emptyset$$

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

3 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.

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

Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game structure, and s_C a strategy for the controller. We define the *outcome* $\mathrm{Outcome}(q, s_C)$ of a strategy 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, $r' = r \xrightarrow{a} q' \in \mathrm{Outcome}(q, s_C)$ if $r' \in \overline{\mathcal{R}}$ and
  - $a \in A_U^{\overline{a}}$;
  - $a \in A_U^{a}$ and $\nexists (r \xrightarrow{a'} q''$ s.t. $\langle a', 0 \rangle \in s_C(r))$.
  - $a \in s_C(r)$.
- An infinite run belongs to $\mathrm{Outcome}(q, s_C)$ if all its finite prefixes also belong to $\mathrm{Outcome}(q, s_c)$

If $q = q_0$, we simply write $\mathrm{Outcome}(s_C)$. The control synthesis problem can be declined into objectives, or winning conditions. For a given game $\mathcal{G}$, a winning condition $C_{\mathcal{W}}$ is a set of allowed runs. A strategy s for the controller is winning if $\mathrm{Outcome}(s) \subseteq C_{\mathcal{W}}$.

The result of applying a strategy on a game is also a game whose set of runs is exactly the outcome of the strategy. We will use both definitions indifferently.

Relation to timed Games It is possible to express some of our semantics using timed games, as in (De Alfaro et al., 2003; Behrmann et al., 2007). The avoidable actions for example, translate directly into guards with a lower bound on clocks and vice versa. However, the ineluctability cannot be translated *as-is* into and from timed automata. We can use invariants on states to force the environment to play, but as the model uses explicit clocks, we have to express an upper bound on the invariants. Our extension here removes this need for explicits values or parameters. It restricts only the behavior of the environment, not of the controller, as it is in timed automata where invariants apply to all players including the controller.

4 REACHABILITY GAMES

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

Definition 1 (Reachability objective). *Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game, and Goal $\subseteq Q$ a set of goal states. A run $r \in \mathcal{R}$ is winning if it has a finite prefix r' such that* $\mathrm{Last}(r') \in Goal$. *The set of winning runs is denoted* $\mathrm{Reach}(Goal)$.

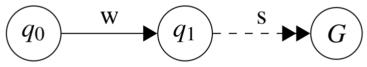

Figure 5: Example reachability game. The objective is to reach the state G.

4.1 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. 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);
- all other actions of the environment cannot prevent the game to reach a state in X.

Definition 2 (Controllable predecessors)**.** *Let $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ be a game, and $X \subseteq Q$ a set of states. The controllable predecessors $\pi(X)$ of X is the subset of Q defined by:*

$$\begin{aligned}\pi(X) = \mathrm{pre}_C(X) \setminus \mathrm{pre}_{U^{\overline{a}}}(\overline{X}) \\ \cup\, \mathrm{pre}_{U^{\diamond}}(X) \setminus \mathrm{pre}_U(\overline{X})\end{aligned} \tag{1}$$

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.

The set of winning states is computed using a backwards fixed-point algorithm.

Algorithm 3 Reachability computation algorithm

```
Require: $Goal$
Ensure: $\mathcal{W}$
  $\mathcal{W} \leftarrow Goal$
  while $\pi(\mathcal{W}) \neq \mathcal{W}$ do
      $\mathcal{W} \leftarrow \mathcal{W} \cup \pi(\mathcal{W})$
  end while
  if $initial \in \mathcal{W}$ then return Success
  else return Failure
  end if
```

From the set of winning and goal states, we can derive a non-deterministic, memoryless strategy for the controller. The canonical memoryless strategy $s_c^m : \mathcal{W} \to (2^{A_C} \cup \bot, \Delta)$ is defined by:

$$s_c^m(q) = \begin{cases} \bot \text{if } \nexists a \in A_C, q \xrightarrow{a} q', q' \in \mathcal{W}, \\ \{\langle a, d\rangle | a \in A_C, q \xrightarrow{a} q', q' \in \mathcal{W}\}, \end{cases} \tag{2}$$

where

$$d = \begin{cases} \mathbf{0} & \text{if } \exists a' \in A_U^a, q'' \in Q \setminus \mathcal{W} \\ & \text{such that } q \xrightarrow{a'} q'', \\ \bullet & \text{otherwise.} \end{cases}$$

4.2 Correctness of the computed strategy

The partial definition of s_c^m on $\mathcal{W}$ makes sense because the strategy does not allow leaving $\mathcal{W}$ (Lemma 5). Note that in the first case, the controller waits for an uncontrollable ineluctable action to occur, which is bound to happen by definition of ineluctable actions, and because the current state is in $\mathcal{W}$. The second case just cuts off transitions which would lead to loosing states.

The following lemmas and theorems hold if Algorithm 3 returns successfully. To simplify the formulation of the results we will always assume that it returns successfully. For our game $\mathcal{G}$, let us consider a reachability condition in the form of $Goal \subseteq Q$. We will assume that the algorithm returns successfully and computes the set $\mathcal{W}$ of winning states.

Lemma 4 (Winning states)**.** *From all states in $\mathcal{W}$, there is a* sure *run to a state in Goal, i.e. $\forall s \in \mathcal{W}, \exists r \in \mathcal{R}^s$ such that* $\mathrm{First}(r) = s$ *and* $\mathrm{Last}(r) \in Goal$.

Proof. We will proceed by induction. We define the following sequence:

$$\mathcal{W}_k = \begin{cases} Goal & \text{if } i = 0, \\ \mathcal{W}_{i-1} \cup \pi \mathcal{W}_{i-1} & \text{otherwise.} \end{cases}$$

It is trivial to see that for all states in $\mathcal{W}_0$, there is a *sure* run for a state in $Goal$. If the property holds for $n \in \mathbb{N}$, we can show that it holds for $n+1$ by finding a run from $\mathcal{W}_{n+1}$ to a state in $\mathcal{W}_n$, from which we have a *sure* winning run. From the definition of π, we can see that, for all states $q \in \mathcal{W}_{n+1}$:

$$\begin{aligned}\exists a \in A_C, q' \in \mathcal{W}_n \text{ such that } \exists q \xrightarrow{a} q' \text{and} \\ \nexists a' \in A_U^{\overline{a}}, q'' \in \overline{\mathcal{W}_n}, q \xrightarrow{a'} q'',\end{aligned} \tag{3}$$

or

$$\begin{aligned}\exists a \in A_U^{\diamond}, q' \in \mathcal{W}_n \text{ such that } \exists q \xrightarrow{a} q' \text{and} \\ \nexists a' \in A_U, q'' \in \overline{\mathcal{W}_n}, q \xrightarrow{a'} q''.\end{aligned} \tag{4}$$

Because the sequence is monotonic and the set of states is finite, it necessarily converges to a limit, $\mathcal{W}$. □

Lemma 5. *Given s_c^m defined by Equation 2, we have* $\text{Outcome}(s_c^m|_{\mathcal{W}}) = \text{Outcome}(s_c^m)$. *That is, the application of the strategy does not leave $\mathcal{W}$.*

Proof. Let us procede by contradiction. Let's suppose that there is a run $r \in \text{Outcome}(s_c^m) \setminus \text{Outcome}(s_c^m)|_{\mathcal{W}}$. Assuming our hypotheses, that means that there are one or several intermediate states in r which are not an element of $\mathcal{W}$. It cannot be first because $q_0 \subseteq \mathcal{W}$ by definition. Let's take q the first of these states, and denote by p the state just before q, and a the action such that $p \xrightarrow{a} q$ is an infix of r. We have $q \in \overline{\mathcal{W}}$ and $p \in \mathcal{W}$. By definition, we also have $a \notin A_U^{\diamond}$. By definition of the outcome, we have one of the following cases:

$$a \in A_U^{\overline{a}} \quad (5)$$

$$a \in A_U^{a} \text{ and } \nexists (p \xrightarrow{a'} q' \text{ s.t. } \langle a', 0\rangle \in s_c^m(p)) \quad (6)$$

$$a \in s_c^m(r). \quad (7)$$

The case of 7 is not possible by definition of s_c^m, and because $q \notin \mathcal{W}$. This would lead to a contradiction of the hypothesis.

If $a \in A_U^{\overline{a}}$ (5), then by definition we have $p \in \text{pre}_{U^{\overline{a}}}(\overline{\mathcal{W}})$, and $p \in \text{pre}_U(\overline{\mathcal{W}})$. By definition of π, we have $p \notin \pi(\mathcal{W})$, thus $p \notin \mathcal{W}$ by definition of $\mathcal{W}$. We would have a contradiction too.

Let's consider the case (6). The fact that there is no possible move $\langle a', 0\rangle \in s_c^m(p)$ implies that $s_c^m(p) = \perp$. Thus, we have $p \notin \text{pre}_C(\mathcal{W})$. Because $a \in A_U$, we have $p \notin \pi(\mathcal{W})$, thus $p \notin \mathcal{W}$ by definition of $\mathcal{W}$.

Since all possible cases lead to a contradiction, the hymothesis is false and thus the lemma holds. □

Lemma 4, proves that the game is possibly winning for the controller, with the right strategy.

We cannot prove that our generated controller includes all and only the winning runs, since it allows infinite runs (see Figure 6).

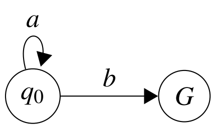

Figure 6: Possibly infinite game: the canonical strategy allows to do a infinitely many times before b.

We must however assure that the strategy is safe and sound, in that it does not capture loosing runs, and all winning runs are captured.

Theorem 6. *For all* $r \in \text{Outcome}(s_c^m), r \in \overline{\mathcal{R}} \implies \exists \rho \in \overline{\mathcal{R}^s}$ *s.t.* $r\rho \in \text{Reach}(\mathit{Goal})$ *and* $r\rho \in \text{Outcome}(s_c^m)$

Proof. If any finite run r does not leave $\mathcal{W}$, we have $\text{Last}(r) \in \mathcal{W}$. Thus, by Lemmas 4 and 5 there is a finite winning run $r' \in \overline{\mathcal{R}^s}$ such that $r' \in \text{Reach}(\mathit{Goal})$, and $rr' \in \text{Outcome}(s_c^m)$. Formally, $\forall r \in \text{Outcome}(s_c^m), \text{States}(r) \cap \overline{\mathcal{W}} = \emptyset$. □

Lemma 7. *For all* $r \in (\text{Reach}(\mathit{Goal}) \cap \mathcal{R}^s)|_{\mathcal{W}}$, *we have either*

$$\begin{cases} r \in \text{Outcome}(s_c^m), \textit{ or} \\ \exists r' \in \text{Outcome}(s_c^m), \\ \quad \textit{such that } r \in (\text{Reach}(\mathit{Goal}) \cap \mathcal{R}^s)|_{\mathcal{W}}. \end{cases}$$

Sketch of proof. From Theorem 6, we know that $\text{Outcome}(s_c^m) \subseteq \mathcal{R}^s|_{\mathcal{W}}$. For a given run $r \in (\text{Reach}(\mathit{Goal}) \cap \mathcal{R}^s)|_{\mathcal{W}}$, it can either be in $\text{Outcome}(s_c^m)$ or not. In the latter case, the run has been cut at a certain point because a different action from the environment could have spoiled the outcome of the game (see Figure 7).

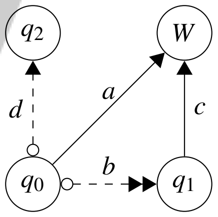

Figure 7: Example of a winning run cut by our strategy. The winning run bc is cut because the controller takes a immediately in order to prevent the environment to take d.

Because Algorithm 3 returns successfully, we can construct a *sure* winning run from every state in $\mathcal{W}$, so there is another winning run. □

As a consequence, we can state the following theorem about the existence of solutions and our ability to capture them.

Theorem 8. *If* $\text{Reach}(\mathit{Goal}) \neq \emptyset$, *then* $\text{Outcome}(s_c^m) \neq \emptyset$.

5 CONCLUSION

In this paper, we have presented a semantic extension of untimed automata to introduce a model based design methodology in the conception of low level software for embedded systems. This extension introduces two uncontrollable actions' properties that extend the model of the environment:

- the *delayed action* cannot happen instantaneously so that the device driver can perform another action if needed.
- the *ineluctable* action is guaranteed to happen eventually, and on which the driver can rely.

This model combines some of the expressiveness of timed games, with the simplicity of untimed automata. It allows an easier implementation of these models, more suitable to embedded real-time systems.

We have adapted the notion of control and reachability games for this extension and defined and proved an algorithm to solve these problems in the general case.

However, the generated strategy can be non-deterministic, and allows infinite runs (one can switch infinitely many times between two states before reaching the goal). In practice, we need to find a deterministic implementation of the strategy that finds the shortest path and eliminating loops.

The goal of this work is to provide the complete toolchain to model, configure and generate the device driver code for any given system. In order to do so, we must extend the controller synthesis to safety games, which deals with avoiding *bad* states. We will also propose a generic implementation of the resulting controller to complete the methodology.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2007). Uppaal-tiga: Time for playing games! In *Computer Aided Verification*, pages 121–125. Springer.

De Alfaro, L., Faella, M., Henzinger, T. A., Majumdar, R., and Stoelinga, M. (2003). The element of surprise in timed games. In *CONCUR 2003-Concurrency Theory*, pages 144–158. Springer.

Kirschke-Biller, F. (2011). Autosar – A worldwide standard current developments, roll-out and outlook. www.autosar.org.

Ramadge, P. J. and Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98.

The International Organization for Standardization (2011). ISO/DIS 26262 - Road vehicles - Functional safety.