Noname manuscript No.
(will be inserted by the editor)

Logical Time Control of concurrent DES

Jean-Luc Béchennec[1] · **Didier Lime**[2] · **Olivier (H.) Roux**[2]

the date of receipt and acceptance should be inserted later

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 occurrence 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 paper then extends these results to Game Petri Nets which can express concurrent timed behaviors and where an avoidable transition can lose its avoidability by the elapsing of time. The usefulness of this new model is illustrated by a real device driver synthesis example.

Keywords Finite automata · Game theory · Controller synthesis · Timed systems

1 Introduction

The theory of supervisory control has been well developed since about 30 years ago with the seminal works of [26,23,18]. It has become a basic paradigm for the control of discrete event systems (DES) modeled as finite state machines.

1
CNRS, LS2N, Nantes France
E-mail: jean-luc.bechennec@ls2n.fr

2
Ecole Centrale de Nantes, LS2N, Nantes France
E-mail: didier.lime@ec-nantes.fr
E-mail: olivier-h.roux@ec-nantes.fr

Since [23], different formalisms have been considered to model (un)controllable actions and control problems. Formulating control problems as two-player games have provided efficient solutions [24]. In this setting, the controller is modeled by a player and the environment by its opponent. Determining whether a controller exists amounts to determine if it can win and computing a winning strategy is equivalent to synthesizing a controller. However, these turn-based games [24] where one player chooses their action before the other chooses theirs, are sequential and do not allow to model concurrency. Therefore, concurrent games [14, 16, 1] 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.

Besides 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 fails, including choosing not to play.

The model of timed automata [3] and timed games [15] 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 [2, 6] allow the modeling of systems with complex interactions while providing a formal proof of 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 acknowledgment of the reception of a command. An ineluctable action is guaranteed to happen eventually if nothing is done to abort it, which differs from 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.

Finally, since an avoidable uncontrollable action is avoidable because it is assumed to take some time, a strategy for the controller wishing to avoid that action is to act urgently.

*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 time games, in which players can play their actions immediately (urgently) or not. As a consequence, moves by the players carry information both on the action played and the timing at which they are played.

We then extend these logical time games in order to express concurrent timed behaviors with game Petri Nets.

This paper is organised as follows:

We first give in Section 2, the basic definitions and notations for logical time games. By using these notations, we justify our new model in Section 3. Then, in Section 4, we solve the controller synthesis problem for logical time games. In Section 5, Section 6 and Section 7, we respectively focus on reachability games, safety games and safe reachability games. In Section 8 we extend these results to Game Petri Nets. We discuss the complexity of the winning state computation algorithm implemented in our tool ROMÉO in section 10. Finally, in Section 11 we illustrate our method on a case study based on a Microchip CAN controller.

This article is an extension of [4,5], with mainly the addition of the complete proofs, the safe reachability games, and the setting of Game Petri Nets. The case-study has also been updated to use Game Petri Nets.

2 Logical time games

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

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

Definition 1 (Game structure) A game structure is a tuple $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ 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.
- $A_U^{\odot} \subseteq A_U$ and $A_U^{\blacktriangle} \subseteq A_U$ are the subsets of *avoidable* and *ineluctable* actions, respectively. Note that these subsets are independent, and their intersection is not necessarily empty.
- $\delta : Q \times (A_C \cup A_U) \times Q$ 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.

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

2.1 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.

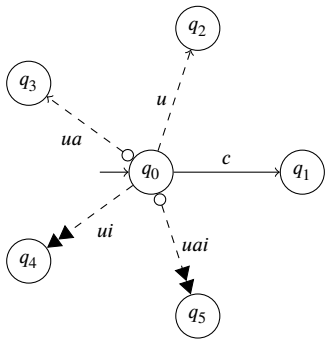


Fig. 1 Graphical notation example: Here q_0 is the initial state, and $c \in A_C, u \in A_U^{\overline{\odot}\,\overline{\blacktriangle}}, ua \in A_U^{\odot\overline{\blacktriangle}}, ui \in A_U^{\overline{\odot}\blacktriangle}$ and $uai \in A_U^{\odot\blacktriangle}$.

2.2 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 $\{\mathbf{0}, \overline{\mathbf{0}}\}$, which represents the logical time at which an action is played. It can be instantaneous ($\mathbf{0}$), or non-immediate ($\overline{\mathbf{0}}$). Semantically, $\langle a, \mathbf{0}\rangle$ means that the action a is performed *immediately*, whereas in $\langle a, \overline{\mathbf{0}}\rangle$, the action a is performed after a non-null time.

Avoidable actions

From a given state q an avoidable (and then non-immediate) action $\langle u, \overline{\mathbf{0}}\rangle$ (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 u will eventually happen if we do not do anything else from this state, that is to say, if we wait long enough. Note that it could still happen immediately.

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 $\langle c, \mathbf{0} \rangle$.

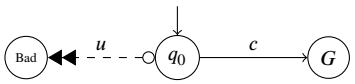


Fig. 2 Avoidable actions (even when ineluctable) can be prevented by the controller

Moreover, like any non-avoidable uncontrollable action, a non-avoidable ineluctable action cannot be prevented by a controllable action. The environment still has a choice of what it wants to play when there is an ineluctable action, however. The important thing is that it must play something, and to that extent, ineluctability could equally well be defined on states, though from an applicative point of view it makes sense to keep it on actions as we did.

This latter consideration also demonstrates how ineluctability is different from fairness. With the general notion of fairness, one assumes that all edges, or states, or some other features are considered infinitely often. Ineluctability does not imply anything like this: in Figure 3, the environment might very well decide to always play the loop and never the ineluctable action. In that sense it is not assumed to be fair, neither weakly nor strongly. It follows that the two leftmost reachability games in that figure are not winning.

Note that the time at which the loop is taken does not need to be 0. The rightmost game in Figure 3 is actually a timed automaton with a single clock x, which is reset to 0 on the loop and cannot exceed 2 when in q_0. Also, it must be greater than or equal to 2 to proceed to G. This is a model we want to abstract with the leftmost automaton: the invariant implies that the environment must play (but does not prescribe which action) and if the environment always chooses the loop at any time before x is 2 then the guard of the transition to G is actually never satisfied.

Note that we try here to show the conceptual differences between fairness and ineluctability but we do not claim that neither of them can simulate the other using more complex constructions. This is left as an open problem.

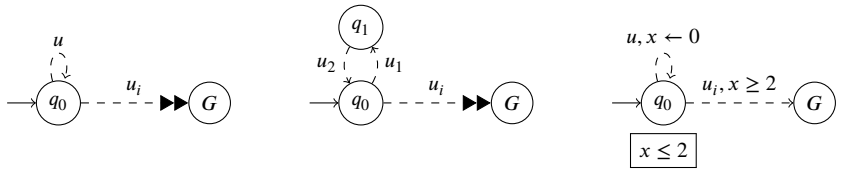


Fig. 3 Ineluctability is not fairness.

2.3 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. $q \xrightarrow{a} q'$. When $\Sigma = A_C \cup A_U$, we omit Σ and simply write $\mathrm{pre}(X)$ and $\mathrm{suc}(X)$.

A *run* of a game structure is a sequence $q_0 \langle a_1, d_1 \rangle q_1 \langle a_2, d_2 \rangle q_2 \ldots$ with $a_i \in A_C \cup A_U$, $d_i \in \Delta$, $q_i \in Q$, and such that $q_i \xrightarrow{a_i} q_{i+1}$ for all $i \geq 0$. 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 $\langle a_1, d_1 \rangle \langle a_2, d_2 \rangle \ldots$.

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

3 Justification for this new model

For the examples used in this section, we consider a reachability game (formally defined in Section 5) starting in q_0 where the goal is to reach a state denoted G.

3.1 Avoidable actions

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 [21,15] allow solving the controller synthesis problem for reachability or safety objectives. In [25], the authors consider an abstraction of timed automata [3] where a transition τ represents the fact that some time elapses. 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 [25], 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.

The concept of avoidable uncontrollable action addresses concerns similar to that of (controllable) forcible action from [11]. A forcible action can preempt the elapsing of time and therefore happen immediately, which is very much what the controller strategy to avoid an avoidable uncontrollable action will be. Yet, in practice, forcible actions are part of the framework called timed DES, in which time is modeled explicitly by *tick* events. In our formalism, time is modeled implicitly, and also much less precisely, allowing for models that are smaller and easier to analyze.

3.2 Ineluctable actions

In our games, by default, players have the option not to play. Ineluctable actions locally remove this possibility for the environment. They model things that are known to happen in a

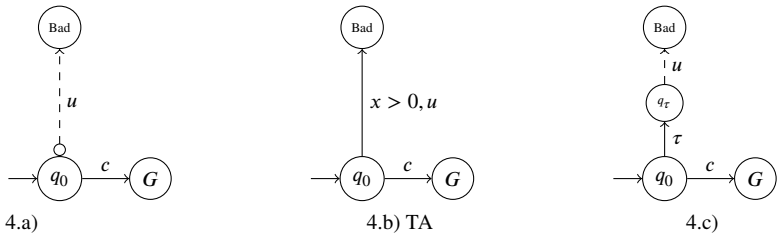


Fig. 4 Avoidable uncontrollable actions can be prevented by the controller

nominal context: the end of a transmission or a conversion, or more generally an acknowledgment of the reception of a command.

Ineluctable actions vs controllable actions. 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 belong to the controller who has to ensure that both states q_1 and q_2 are winning.

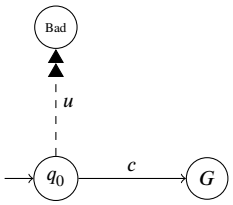


Fig. 5 Ineluctable actions cannot be prevented by the controller

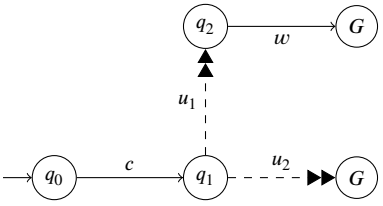


Fig. 6 Ineluctable actions are not selected by the controller

We could try to replace ineluctable actions with both a normal uncontrollable action and a controllable action with the same source and target as in Figure 7. The idea would be that the environment can still choose what action it wants to play but if it tries to do nothing the controller can choose instead, effectively forcing the environment to play. This trick works

Fig. 7 Ineluctable actions cannot be forced by the controller

when there are no avoidable actions but the games in Figure 7 are counter-examples for the general case: the original game on the left is actually losing because the environment can delay its move until it can play the avoidable action. But in the transformed game on the right, the controller can play action c immediately, effectively forcing the modeled ineluctable action to happen at time 0 and avoiding the losing action, and thus the game is winning.

Ineluctable actions vs timed actions. 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 behavior of the environment. In [13], Timed Games are based on Timed Automata with invariants that 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 this has not been done in the literature, 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 [10]. 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 behavior of the environment and not that of the controller.

4 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.

Let us recall C and U are the two players called controller and environment, respectively.

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 Last(r).

When both $\langle a, \mathbf{0} \rangle$ and $\langle a, \overline{\mathbf{0}} \rangle$ are in $s(r)$, we write $\langle a, \mathbf{0} + \overline{\mathbf{0}} \rangle \in s(r)$ for short.

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 $\text{Last}(r) = q$.

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

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

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 strategies. Since we are interested in the strategies for the controller to win whatever the (legal) strategy of the environment is, 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 $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ be a game structure, r one of its runs, and s_C a strategy for the controller. The *outcome* $\text{Outcome}(q, s_C)$ of s_C from state q is the subset of $\mathcal{R}$ defined inductively by:

- $q \in \text{Outcome}(q, s_C)$
- If $r \in \text{Outcome}(q, s_C)$ is finite, $r' = r \xrightarrow{\langle a,d \rangle} q' \in \text{Outcome}(q, s_C)$ if $r' \in \overline{\mathcal{R}}$ and one of the following holds true:
  - $a \in A_U^{\overline{\odot}}$ and if $\exists \langle a', \mathbf{0} \rangle \in s_C(r)$, then $d = \mathbf{0}$;
  - $a \in A_U^{\odot}$ and $\exists \langle a', \overline{\mathbf{0}} \rangle \in s_C(r)$.
  - $\langle a, d \rangle \in s_C(r)$.
- An infinite run belongs to $\text{Outcome}(q, s_C)$ if all its finite prefixes also belong to $\text{Outcome}(q, s_C)$

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 $a \in A_C \cup A_U^{\blacktriangle}$, 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 $\text{MaxOutcome}(q, s_C)$ the set of runs that are maximal in $\text{Outcome}(q, s_C)$.

The control synthesis problem can be stated using winning conditions, also called objectives. 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 $\text{MaxOutcome}(q, s) \subseteq C_{\mathcal{W}}$. A state q is winning if there exists a winning strategy from q. The game itself is winning if q_0 is winning.

5 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 $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ be a game structure, and $\text{Goal} \subseteq Q$ a set of goal states. The reachability winning condition (or objective) Reach(Goal) for Goal is the set of runs r that are maximal in $\mathcal{R}$ and such that $\text{States}(r) \cap \text{Goal} \neq \emptyset$.

For example, for the game of Figure 8, the objective is to reach the state G and we have Goal $= \{G\}$ and Reach(Goal) $= \{q_0 \langle c, d_1 \rangle q_1 \langle u, d_2 \rangle G \mid d_1, d_2 \in \Delta\}$.

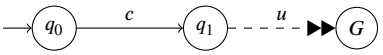


Fig. 8 The objective is to reach the state G.

5.1 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 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 backward 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 $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ 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:

$$\begin{aligned} \pi(X) = \mathrm{pre}_{A_C}(X) \setminus \mathrm{pre}_{A_U^{\overline{\odot}}}(\overline{X}) \\ \cup \mathrm{pre}_{A_U^{\blacktriangle}}(X) \setminus \mathrm{pre}_{A_U}(\overline{X}) \end{aligned} \tag{1}$$

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, for Goal $\subseteq Q$ the set of winning states for the winning condition Reach(Goal) is computed using the following classic backward fixed-point algorithm 1: $\mathcal{W}_0 =$ Goal and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n)$. When it exists, the final fixed-point set of winning states is noted $\mathcal{W}$.

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* Outcome(q_1, s_1) *such that* States$(r) \cap$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.

Algorithm 1 Winning states computation algorithm for reachability game

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$, Goal $\subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow$ Goal
  while $\pi(\mathcal{W}) \not\subseteq \mathcal{W}$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cup \pi(\mathcal{W})$
  end while
  return $\mathcal{W}$

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 MaxOutcome(q_2, s). If States$(r) \cap Q_1 = \emptyset$ then $r \in$ MaxOutcome(q_2, s_2) and therefore it eventually goes through Goal. Otherwise, we can write r as $r_2 r_1$, with $r_2 \in$ Outcome(q_2, s_2), Last$(r_2) \in Q_1$ and $r_1 \in$ Outcome(Last$(r_2), s_1)$. Since 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, $\mathcal{W}_0 =$ Goal, and $q \in$ 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 $q \in \text{pre}_{A_C}(\mathcal{W}_n) \backslash \text{pre}_{A_U^{\overline{\odot}}}(\overline{\mathcal{W}_n})$: 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 $q \notin \text{pre}_{A_U^{\overline{\odot}}}(\overline{\mathcal{W}_n})$. 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 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 $x \in A_U^{\overline{\odot}}$ 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 $q \in \mathrm{pre}_{A_U^{\blacktriangle}}(\mathcal{W}_n) \setminus \mathrm{pre}_{A_U}(\overline{\mathcal{W}_n})$. This case is fairly similar to the previous one: we know there exists some $a \in A_U^{\blacktriangle}$ 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 MaxOutcome(q, s). Note that the run consisting of only q cannot be maximal since $a \in A_U^{\blacktriangle}$. 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 MaxOutcome(q, s) reach Goal in at most $n + 1$ steps.

Consider the possible actions from q. If they are all 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 one 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 $q \notin \mathrm{pre}_{A_U^{\overline{\odot}}}(\overline{\mathcal{W}_n})$. 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, $q \in \mathrm{pre}_{A_U^{\blacktriangle}}(\mathcal{W}_n)$. 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 one controllable or one 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 $\overline{\mathbf{0}}$.

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 moving 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.

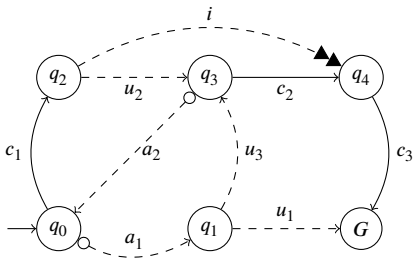


Fig. 9 A reachability game. The objective is to reach the state G.

5.2 Reachability game example

Let us consider the reachability game $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ of Figure 9 where the objective is to reach the state G: Goal $= \{G\}$. By applying the backward fixed-point algorithm 1: $\mathcal{W}_0 =$ 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 $s(q_0) = \{\langle c_1, \mathbf{0}\rangle\}$, $s(q_3) = \{\langle c_2, \mathbf{0}\rangle\}$, $s(q_4) = \{\langle c_3, \mathbf{0} + \overline{\mathbf{0}}\rangle\}$ and $s(q_1) = s(q_2) = s(G) = \emptyset$.

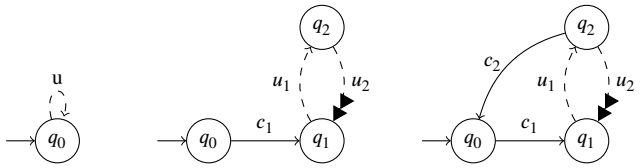
6 Safety game

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 $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ be a game structure and Safe $\subseteq Q$ a set of *safe* states.

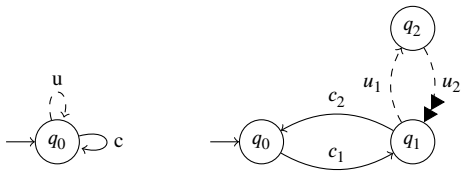
The safety objective for Safe is the set of all infinite maximal runs r of $\mathcal{G}$ such that $\mathrm{States}(r) \subseteq \mathrm{Safe}$.

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 not to play, the controller must be able to move. Hence, the safety games of Figure 10 where all the states are in the set of *safe* states, are losing. Indeed, for the game of Figure 10.a, we have $q_0 \notin \pi(\{q_0\})$ and for the games of Figures 10.b and 10.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 11 are winning.



10.a) 10.b) 10.c)

Fig. 10 All the states are safe but the games are not winning.



11.a) 11.b)

Fig. 11 All the states are safe and the games are winning.

6.1 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 backward fixed-point algorithm 2: $\mathcal{W}_0 = \mathrm{Safe}$ 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 5, we can prove the soundness and completeness of Algorithm 2, by proving the following two lemmas. The proofs are very similar to those of Section 5 and are therefore omitted.

Algorithm 2 Winning states computation algorithm for safety games

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$, Safe $\subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow$ Safe
  while $\mathcal{W} \nsubseteq \pi(\mathcal{W})$ **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* MaxOutcome(q, s)*, we have* States$(r) \subseteq$ 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* MaxOutcome(q, s)*, we have* States$(r') \subseteq$ Safe*, then* 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 MaxOutcome(q, s) are such that States$(r) \subseteq$ 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 $s^s : \mathcal{W} \to 2^{(A_C \times \Delta)}$ that does exactly this:

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 $\exists a' \in A_U^{\odot}, q'' \notin \mathcal{W}$ and $d = \mathbf{0} + \overline{\mathbf{0}}$ otherwise.

Permissive strategies are a key notion in supervisory control [23]. In reactive synthesis, permissiveness is measured in terms of the set of behaviors allowed by the strategy [7]. Hence most permissive strategies do not need to 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',* Outcome$(q_0, s') \subseteq$ 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 Outcome$(q_0, s') \setminus$ Outcome(q_0, s^s). Let r be the longest prefix of that run that is in Outcome(q_0, s^s). Let $q =$ Last(r). Then we have, for some action a, $q \xrightarrow{\langle a, d\rangle} q' \in$ Outcome(s', q) and $q \xrightarrow{\langle a, d\rangle} q' \notin$ 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 $a \in A_U$, $d = \mathbf{0} + \overline{\mathbf{0}}$, 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 $c \in A_U^{\odot}$ and a state q'' such that $q \xrightarrow{\langle c, \mathbf{0}+\overline{\mathbf{0}}\rangle} q''$ and $q'' \notin \mathcal{W}$. Since there is no immediate controllable action in $s'(r)$ then clearly $r \xrightarrow{\langle c, \mathbf{0}+\overline{\mathbf{0}}\rangle} q'' \in \text{Outcome}(q_0, s')$, which, by Theorem 3, contradicts the fact that s' is winning.

6.2 Safety game example

Let us consider the safety game $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ of Figure 12 where the objective is to avoid the state B. Hence $\text{Safe} = \{q_0, q_1, q_2\}$ is the set of *safe* states.

By applying the backward fixed-point algorithm 2: $\mathcal{W}_0 = \text{Safe}$ 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 $s(q_1) = \{\langle c, \mathbf{0}\rangle\}$.

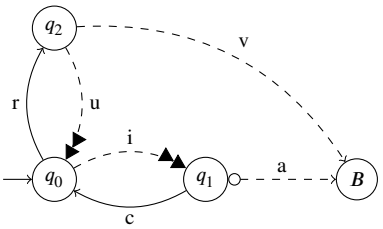


Fig. 12 A winning safety game. The objective is to avoid the state B.

7 Safe reachability

From a practical point of view, reachability and safety must often be carried out jointly. The goal is to reach an objective state while avoiding the states which are not safe.

Definition 8 (Safe reachability objective)
Let $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ be a game structure, and $\text{Goal}, \text{Safe} \subseteq Q$ respectively sets of goal and safe states. The safe reachability winning condition (or objective) for Goal and Safe is the set of runs r that are maximal in $\mathcal{R}$ and such that $\text{States}(r) \cap \text{Goal} \neq \emptyset$ and $\text{States}(r) \subseteq \text{Safe}$.

It is not enough to apply successively the computation of the winning states for the reachability game then for the safety game because if the strategy for reachability consists in going through states which are not sure, these states will be removed by the safety game and the objective state will no longer be reachable. Hence, for the game of Figure 13, without considering that B should be avoided, all the states are winning for the reachability game. So a strategy to reach the state G consists in making c_1 then c_2 then c_3 but the safety game then withdraws the state B and the system blocks in the state q_1.

Similarly, if we first apply the safety game, the strategy can consist in waiting for an ineluctable action from a state which was safe if it is left immediately as illustrated in Figure

13. If we successively apply the computation of the winning states for the safety game and then for the reachability game, we first obtain that the safety game removes the state B. Then, a reachability game strategy consists in playing c_4 and then waiting for the occurrence of u_2 while this wait can allow the occurrence of u_1 which would lead to B.

It is thus necessary to propose a fixed point dedicated to these kinds of properties.

The set of winning states for a safe reachability game can thus be calculated using the backward fixed-point algorithm 3:

$\mathcal{W}_0 = \text{Goal} \cap \text{Safe}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n) \cap \text{Safe}$.

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

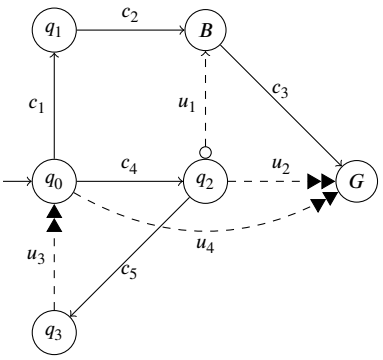


Fig. 13 The objective is to reach the state G while avoiding the state B.

Algorithm 3 Winning states computation algorithm for safe reachability game

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$, Goal $\subseteq Q$, Safe $\subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow \text{Goal} \cap \text{Safe}$
  while $\pi(\mathcal{W}) \cap \text{Safe} \not\subseteq \mathcal{W}$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cup \pi(\mathcal{W}) \cap \text{Safe}$
  end while
  return $\mathcal{W}$

The application of this algorithm on the game of Figure 13 will successively give: $W_0 = \{G\}$ et $W_1 = \{G, q_0\}$. $W_2 = \{G, q_0, q_3\}$. $\mathcal{W} = W_3 = \{G, q_0, q_3, q_2\}$. The strategy to reach the state G while avoiding the state B therefore consists in waiting in the state q_0 for the occurrence of the ineluctable action u_4.

8 Game Petri Nets

We now extend the previous results to Game Petri Nets, which can express concurrency between transitions and logical time and where an avoidable transition can lose its avoidability by the elapsing of time.

8.1 Petri Nets

Definition 9 (Petri Net) A Petri Net is a 4-tuple $\mathcal{N} = (P, T, Pre, Post, m_0)$ where P is a finite set of places, T is a finite set of transitions, Pre and $Post$ are matrices of $\mathbb{N}^{|P|\times|T|}$ called the backward and forward incidence matrices, such that $Pre(p,t) = n$ with $n > 0$ when there is an arc from place p to transition t with weight n and $Post(p,t) = n$ with $n > 0$ when there is an arc from transition t to place p with weight n, and the vector $m_0 \in \mathbb{N}^{|P|}$ is called the initial marking.

Given a Petri Net $\mathcal{N} = (P, T, Pre, Post, m_0)$, we denote $Pre(.,t)$ (also written $pre(t)$) as the vector $(Pre(p_1,t), Pre(p_2,t), ..., Pre(p_{|P|},t))$ *i.e.* the t^{th} column of the matrix Pre. The same notation is used for $Post(.,t)$ (or $post(t)$).

Definition 10 (Marking) A marking of a Petri Net $\mathcal{N} = (P, T, Pre, Post, m_0)$ is a vector $m \in \mathbb{N}^{|P|}$.

If $m \in \mathbb{N}^{|P|}$ is a marking, $m(p_i)$ is the number of tokens in place p_i and we have:

$$m \leq m' \Leftrightarrow \forall p \in P, m(p) \leq m'(p)$$

An example of *Petri Net* is given in Figure 14.

$$m_0 = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$$

m_0 is also noted by the set $\{p1, p1\}$

$$Pre = \begin{pmatrix} 1 & 1 \\ 0 & 2 \\ 0 & 0 \end{pmatrix} \qquad Post = \begin{pmatrix} 0 & 0 \\ 3 & 0 \\ 0 & 1 \end{pmatrix}$$

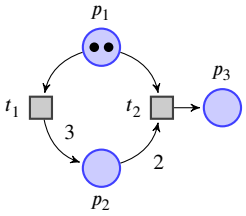


Fig. 14 A Petri Net

Operational Semantics: Given a Petri Net $\mathcal{N}$, a transition $t \in T$ is said *enabled by a marking* m when $m \geq pre(t)$.

Definition 11 (PN Semantics) The semantics of PN is a transition system $\mathcal{N}_\mathcal{T} = (Q, q_0, \rightarrow)$ where, $Q = \mathbb{N}^{|P|}$, $q_0 = m_0$, $\rightarrow \in Q \times T \times Q$ such that,

$$m \xrightarrow{t_i} m' \Leftrightarrow \begin{cases} m \geq pre(t_i) \\ m' = m - pre(t_i) + post(t_i) \end{cases}$$

This relation holds for sequences of transitions:

- $m \xrightarrow{w} m'$ if w is the empty word and $m = m'$
- $m \xrightarrow{wt} m'$ if $\exists m'', m \xrightarrow{w} m'' \wedge m'' \xrightarrow{t} m'$ where $w \in T^*$ and $t \in T$.

Given a marking m, reachability asks if it is reachable from m_0. Formally, $\mathcal{N}$ reaches m iff there exists a sequence w such that $m_0 \xrightarrow{w} m$. We denote $\mathcal{R}(\mathcal{N})$ the set (possibly infinite) of reachable markings of $\mathcal{N}$.

A place in a Petri net is called k-bounded if it does not contain more than k tokens in all reachable markings, including the initial marking. It is bounded if it is k-bounded for some k. A Petri net is called k-bounded or bounded when all of its places are.

8.2 Game Petri Nets with logical time

8.2.1 Definitions

We first extend Petri Nets with *avoidable* and *ineluctable* transitions (PNAE).

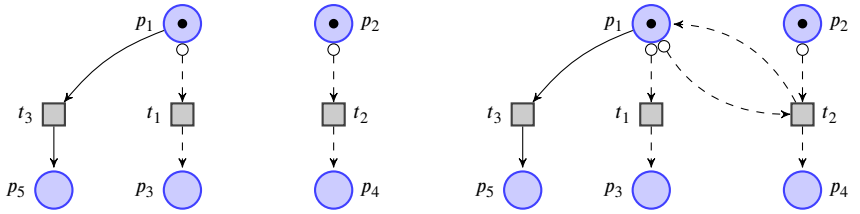
Definition 12 (PNAE) A Petri Net with *avoidable* and *ineluctable* transitions (PNAE) is a tuple $\mathcal{N}_{ae} = (P, T, T^{\odot}, T^{\blacktriangle}, Pre, Post, m_0)$ where $\mathcal{N} = (P, T, Pre, Post, m_0)$ is a Petri Net and $T^{\odot} \subseteq T$ and $T^{\blacktriangle} \subseteq T$ are the subsets of *avoidable* and *ineluctable* transitions, respectively.

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

Definition 13 (Game Petri Net) A Game Petri Net $\mathcal{G}_{\mathcal{N}}$ is a tuple $\mathcal{G}_{\mathcal{N}} = (P, T_C, T_U, T_U^{\odot}, T_U^{\blacktriangle}, Pre, Post, m_0)$ where $T = T_U \cup T_C$, $T_U \cap T_C = \emptyset$, $T_U^{\odot} \subseteq T_U$, $T_U^{\blacktriangle} \subseteq T_U$ and $\mathcal{N} = (P, T, T_U^{\odot}, T_U^{\blacktriangle}, Pre, Post, m_0)$ is a PNAE called the underlying Petri Net of $\mathcal{G}_{\mathcal{N}}$.

8.2.2 Some intuitions

We will define the semantics of Game Petri Net by an associated game structure. However, Petri nets are a model for concurrency and in order to capture its semantics in a sequential game structure, we have to take into account that an avoidable transition can lose its avoidability by the elapsing of time.



15.a) $T_U^{\odot} = \{t_1, t_2\}$ 15.b) $T_U^{\odot} = \{t_1, t_2\}$

Fig. 15 Time elapsing when firing avoidable transitions in Game Petri Nets

Consider the game Petri Net of Figure 15.a, t_1 and t_2 are avoidable, meaning that they are non-immediate. Hence the firing of t_2 implies that time has elapsed since the initial marking and the remaining transition t_1 can now fire immediately and is no more avoidable. On the contrary, in Figure 15.b, after the firing of t_2, the transition t_1 is newly enabled and is then avoidable in the marking $\{p_1, p_4\}$.

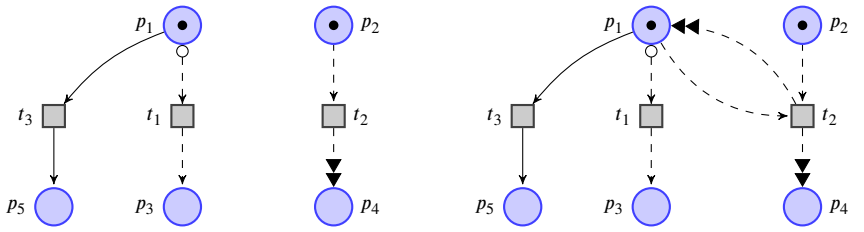


Fig. 16 Time elapsing when firing ineluctable transitions in Game Petri Nets

Time elapsing is less obvious when we consider the concurrency between an avoidable transition and an ineluctable transition. Let us consider the game Petri Net of Figure 16.a. The firing of the ineluctable transition t_2 can be immediate or not. Then after the firing of t_2, the transition t_1 may be still avoidable. However, consider that the strategy of the controller is to do nothing in the marking $\{p_1, p_2\}$ in order to wait for the firing of t_2, leading to the marking $\{p_1, p_4\}$. This waiting step can take some non-null time, and then it is consistent to consider that the transition t_1 is no more avoidable for the controller in the marking $\{p_1, p_4\}$.

On the contrary, for the Game Petri Net of Figure 16.b (as for Figure 15.b), the transition t_1 is newly enabled by the firing of t_2 and is then also avoidable in the marking $\{p_1, p_4\}$.

8.2.3 Semantics of PNAE

The set of enabled transitions for a marking M is $enabled(M) = \{t \mid M \geq pre(t)\}$

Newly enabled transitions: Classically and as defined for time Petri Nets [8], a transition t is *newly enabled* after firing t_i from marking M if it is enabled by $M' = M - pre(t_i) + post(t_i)$ but not by $M - pre(t_i)$. Also a transition is always newly enabled by its own firing.

Formally, the set of transitions that are newly enabled by the firing of t_i from marking M is:

$$\uparrow enabled(M, t_i) = \{t \mid \big(M - pre(t_i) + post(t_i) \geq pre(t_k)\big) \wedge \big((M - pre(t_i) < pre(t)) \vee (t = t_i)\big)\}$$

The set of avoidable newly enabled transitions is then:

$$\uparrow enabled^{\odot}(M, t_i) = \uparrow enabled(M, t_i) \cap T^{\odot}$$

State of a PNAE: Given a PNAE $\mathcal{N} = (P, T, T^{\odot}, T^{\blacktriangle}, Pre, Post, m_0)$, a state of $\mathcal{N}$ is a pair $(m, s_{\odot}) \in \mathbb{N}^{|P|} \times T^{\odot}$ where m is a marking and $s_{\odot}$ is the set of transitions which are avoidable from this state i.e. the avoidable transitions that have not lost their avoidability.

Definition 14 (PNAE Semantics) The semantics of a PNAE $\mathcal{N}$ is a transition system $\mathcal{N}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ where, $Q = \mathbb{N}^{|P|} \times T^{\odot}$, $q_0 = (m_0, \{t \in T^{\odot} | m_0 \geq pre(t)\}$ and $\rightarrow \in Q \times T \times Q$ such that,

$$(m, s_{\odot}) \xrightarrow{t_i} (m', s'_{\odot}) \Leftrightarrow \begin{cases} m \geq pre(t_i) \\ m' = m - pre(t_i) + post(t_i) \\ s'_{\odot} = \uparrow enabled^{\odot}(M, t_i) \text{ if } t_i \in T^{\odot} \cup T^{\blacktriangle} \text{ and} \\ \quad \big(s_{\odot} \cap enabled(M')\big) \cup \uparrow enabled^{\odot}(M, t_i) \text{ otherwise} \end{cases}$$

$\mathcal{R}(\mathcal{N})$ is the set of reachable marking of $\mathcal{N}$. Moreover, $\mathcal{N}$ reaches a state $q = (m, s^{\odot})$ iff there exists a sequence w such that $q_0 \xrightarrow{w} q$. We denote $\mathcal{R}_s(\mathcal{N})$ the set of reachable states of $\mathcal{N}$. Since the set $T^{\odot}$ is finite, there is a finite number of subsets $s^{\odot}$ and then $\mathcal{R}_s(\mathcal{N})$ is finite iff $\mathcal{R}(\mathcal{N})$ is finite i.e. $\mathcal{N}$ is bounded.

8.2.4 Game structure of Game Petri Net

From a Game Petri Net, we can derive an associated game structure defined as follows:

Definition 15 (Game structure of Game Petri Net)
Let $\mathcal{G}_{\mathcal{N}} = (P, T_C, T_U, T_U^{\odot}, T_U^{\blacktriangle}, Pre, Post, m_0)$, be a Game Petri Nets, $\mathcal{N}$ be its underlying Petri Net and $\mathcal{N}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ its semantics.

The game structure of $\mathcal{G}_{\mathcal{N}}$ is $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ such that:

- $Q = \mathcal{R}_s(\mathcal{N})$ is a set of reachable states of $\mathcal{N}$
- $A_C = T_C$
- $A_U^{\odot} = T_U^{\odot}$
- for each avoidable action $a \in A_U^{\odot}$, we create a copy $\overline{a}$. The set of those copies is $A_{U^{\overline{\odot}}} = \{\overline{a} \mid a \in A_U^{\odot}\}$. We further define the subset of those copies that should be ineluctable: $A^{\blacktriangle}_{U^{\overline{\odot}}} = \{\overline{a} \mid a \in A_U^{\odot} \cap A_U^{\blacktriangle}\}$;
- $A_U^{\blacktriangle} = T_U^{\blacktriangle} \cup A^{\blacktriangle}_{U^{\overline{\odot}}}$
- $A_U = A_U^{\blacktriangle} \cup A_U^{\odot} \cup A_{U^{\overline{\odot}}}$
- For all $q = (m, s_{\odot}) \in Q$ and $q' = (m', s'_{\odot}) \in Q$, we have $(q, t, q') \in \delta$ if
  - $t \in A_C \cup A_U^{\blacktriangle}$ and $q \xrightarrow{t} q'$
  - $t \in A_U^{\odot} \cap s_{\odot}$ and $q \xrightarrow{t} q'$
  - $t \in A_{U^{\overline{\odot}}}$ and $\exists t' \in A_U^{\odot}$ such that $t' \notin s_{\odot}$ and $q \xrightarrow{t'} q'$

If $\mathcal{N}$ is bounded, the game structure $\mathcal{G}$ associated with $\mathcal{G}_{\mathcal{N}}$ is finite and we can now define reachability objective, safety objective and compute strategies for $\mathcal{G}_{\mathcal{N}}$ by using $\mathcal{G}$ as in Sections 4, 5, 6 and 7.

8.2.5 Example

Let us consider the game Petri net of Figure 17. Its associated game structure is given in Figure 18. In state q_1, the transition t_2 is no longer avoidable after the firing of the transition t_1 and its non-avoidable copy $\overline{t_2}$ is possible. Then the only strategy to reach q_3 consists in firing t_4 immediately in the initial state q_0.

8.3 Game Petri Nets for unbounded nets

If the underlying Petri Net of a Game Petri Nets is not bounded, the associated game structure, as defined in Definition 15 is not finite. However, if the safety objective includes the k-boundedness of the net, we only need the k-bounded part of the game structure to compute the strategy. Actually, we do not need the successor of a state with a marking such that a place is not k-bounded for this marking.

Recall that $\mathcal{R}_s(\mathcal{N})$ is the set of reachable states of a PNAE $\mathcal{N}$. Let $\mathcal{R}_s^k(\mathcal{N}) = \{q = (m, s_{\odot}) \in \mathcal{R}_s(\mathcal{N}) \mid \forall p \in P, m(p) \leq k\}$ the subset of k-bounded reachable states of $\mathcal{N}$.

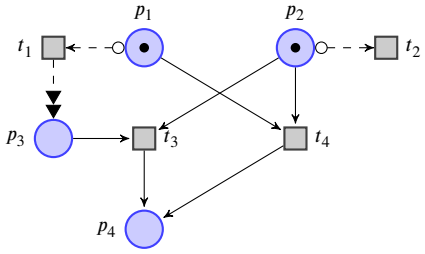


Fig. 17 A Game Petri Net Example. The goal is to reach a state with a token in p_4

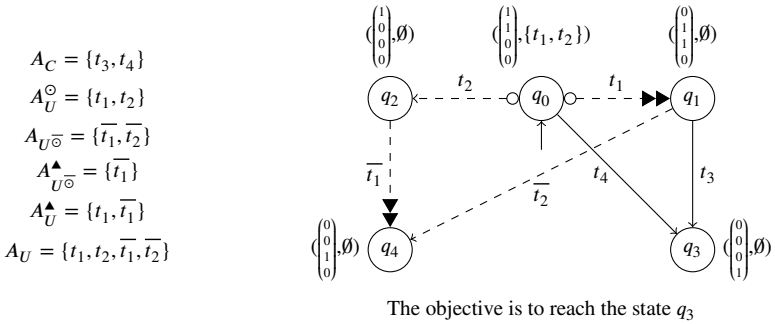


Fig. 18 Game structure associated with the Game PN of Figure 17

For a Game Petri Net with unbounded underlying net and a k-boundedness safety objective, we can define a k-Game structure by the projection of the infinite game structure over a finite set of markings Q as follows.

Definition 16 (k-Game structure of a Game Petri Net with unbounded underlying net and k-boundedness objective)

Let $\mathcal{G}_{\mathcal{N}} = (P, T_C, T_U, T_U^{\odot}, T_U^{\blacktriangle}, Pre, Post, m_0)$, be a Game Petri Net, $\mathcal{N}$ be its underlying Petri Net and $\mathcal{N}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ its semantics.

Let $\mathcal{G} = (Q, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta)$ the game structure of $\mathcal{G}_{\mathcal{N}}$ as defined in Definition 15.

For a k-boundedness safety objective, the k-Game structure of $\mathcal{G}_{\mathcal{N}}$ is $\mathcal{G}_k = (Q_k, q_0, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \delta_k)$ where:

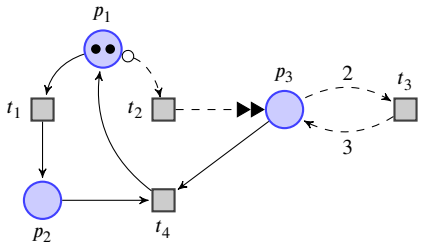
- $Q_k = \mathcal{R}_s^k(\mathcal{N}) \cup \{q' \mid q \in \mathcal{R}_s^k(\mathcal{N}), \exists t \in T_U \cup T_C \text{and } q \xrightarrow{t} q'\}$
- $\forall q \in A_U \cup A_C, \forall q \in Q_k, \forall q' \in Q_k, (q, t, q') \in \delta_k$ iff $(q, t, q') \in \delta$

Note that, for a Petri net $\mathcal{N}$ whose weights are equal to 1, we have $Q_k = \mathcal{R}_s^{k+1}(\mathcal{N})$.

8.4 Example

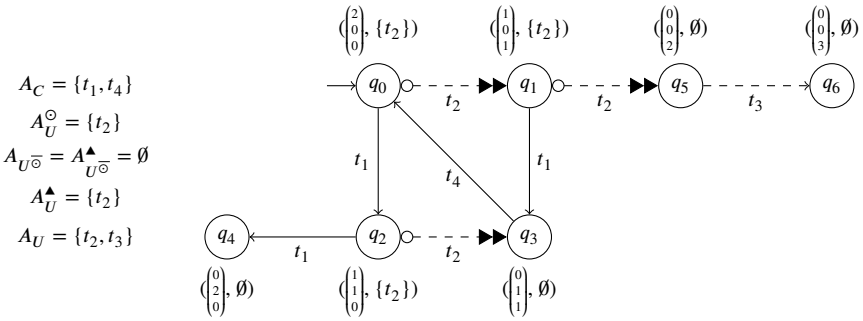
Let us consider the game Petri net of Figure 19 where the place p_3 is not bounded. Its associated k-Game structure for a 2-bounded safety objective is given in Figure 20. The successor of the state $q_6 = (\begin{pmatrix}0\\0\\3\end{pmatrix}, \emptyset)$ is not in this k-game structure since its marking is not 2-bounded. Note that the set $A_{U^{\overline{\odot}}}$ is empty since in state q_1, the transition t_2 is still avoidable because it is newly enabled by its own firing.

A first strategy consists in firing t_1 immediately and waiting in the state q_2 until the firing of t_2 and then firing t_4 in q_3. A second strategy consists in waiting in the state q_0 until the firing of t_2. In q_1, t_2 is avoidable then t_1 can be fired immediately.



$T_C = \{t_1, t_4\}$, $T_U = \{t_2, t_3\}$, $T_U^{\odot} = \{t_2\}$ and $T_U^{\blacktriangle} = \{t_2\}$

Fig. 19 A Game Petri Net with unbounded underlying PN (place P_3 is not bounded)



The safety objective is to avoid the state q_6.

Fig. 20 k-Game structure associated with the Game PN of Figure 19 for a 2-bounded safety objective.

9 Concurrent composition of game structures

9.1 Definition and semantics

It is convenient to describe a system as a parallel composition of automata. In section 2 only monolithic automata are considered. We will consider concurrent compositions of game structures as a particular case of game Petri nets.

It is well-known that an automaton is a particular case of ordinary Petri Net with one token in the input place and where every transition has exactly one input place and one output place. This Petri is either labeled or has equivalently a dedicated transition per occurrence of a given action of the initial automaton (the k^{th} occurrence of action a is, for example, called a^k). This class of Petri Nets is called state graph. When there is only one initial token, it leads to a Petri Net that contains only one token at any time.

To define the parallel composition of game structures, we use the classical composition notion based on a *synchronization function à la* Arnold-Nivat.

Definition 17 (synchronized composition of game structures) Let $G_1, \ldots, G_n$ be n game structure with $G_i = \left(Q^i, q_0^i, A_C, A_U, A_U^{\odot}, A_U^{\blacktriangle}, \rightarrow_i\right)$. A *synchronization function* f is a partial function from $(A_C \cup A_U \cup \{\bullet\})^n \rightarrow A_C \cup A_U$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(G_1| \ldots |G_n)_f$ the parallel composition of the G_i's w.r.t. f.

(G_i, a_i) is said to be not involved in the partial synchronization function f if $f(a_1, \ldots, a_i, \ldots, a_n) = a \Rightarrow \forall x \neq i, a_x = \bullet$ and $a = a_i$.

Definition 18 (Semantics of the synchronized composition of game structures) The semantics of the composition $(G_1| \ldots |G_n)_f$ is defined by its translation into a Game Petri Net[1] as follows:

- each state $q \in Q_1 \cup Q_2 \ldots \cup G_n$ is translated into a place with the same name $q \in P$. Hence a state of the composition is a marking of its corresponding GPN that we denote either by a vector or by the set of places with one token.
- If (G_i, a_i) is not involved in f, then for all occurence $o_k = (q, a_i, q') \in \delta_i$ in G_i, we add a transition a_i^k such that $Pre(q_i, a_i^k) = Post(q_i', a_i^k) = 1$. Moreover, if $a_i \in A_\chi$, with $\chi \in \{C, U\}$ then $a_i^k \in T_\chi$ and if $a_i \in A_U^{\chi'}$ with $\chi' \in \{\odot, \blacktriangle\}$ then $a_i^k \in T_U^{\chi'}$.
- If $f(a_1, \ldots, a_n) = a$, then for all $o_k = (q_1, a_1, q_1')(q_2, a_2, q_2') \ldots (q_n, a_n, q_n') \in \delta_1 \times \delta_2 \cdots \times \delta_n$ with $a_i = \bullet \Rightarrow q_i = q_i'$, we add a transition a^k in T such that $\forall i \in [1, n]$, $a_i \neq \bullet \Leftrightarrow Pre(q_i, a^k) = Post(q_i', a^k) = 1$. Moreover, if $a \in A_\chi$, with $\chi \in \{C, U\}$ then $a^k \in T_\chi$ and if $a \in A_U^{\chi'}$ with $\chi' \in \{\odot, \blacktriangle\}$ then $a^k \in T_U^{\chi'}$

We illustrate this translation with the example of Figure 21 showing the composition of two game structures G_1 and G_2 synchronized by the function f. The synchronization $f(u_1, u_2) = u_1$ means that actions u_1 and u_2 are synchronized and the result is action u_1 that is avoidable but non-ineluctable. We could have chosen that the result of the synchronization is the ineluctable action u_2 or another action u_3 that would be both avoidable and ineluctable or another action u_4 that would be neither avoidable nor ineluctable. The other actions are (implicitly and by default) not synchronized that can be explicitly specified for example for the action c_1 by $f(c_1, \bullet) = f(\bullet, c_1) = c_1$.

This kind of synchronization function is very powerful and allows us to model broadcast or point-to-point synchronization between any number of game structures. Figure 22 shows the synchronization of 3 game structures where actions with the same name are synchronized and keep their nature.

Because of non-determinism (in a single automaton or because of synchronizations), there can be several occurrences of an action in the result of a composition. Since we are considering unlabelled Petri nets, we have to explicitly distinguish each of the k occurrences of an action a. Hence k copies of each action a are created. For example, in Figure 21, action c_1 has two copies c_1^1 and c_1^2 and, in Figure 22, action u_2 has two copies u_2^1 and u_2^2.

[1] For this construction, $\forall p \in P$ and $\forall t \in T$, $Pre(p, t) = Post(p, t) = 0$ except if it is explicitly set to 1.

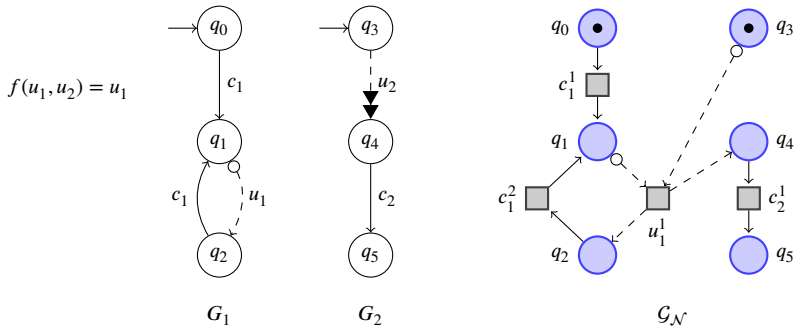


Fig. 21 Game structures G_1 and G_2 and the Game PN $\mathcal{G}_\mathcal{N}$ of the composition $(G_1|G_2)_f$

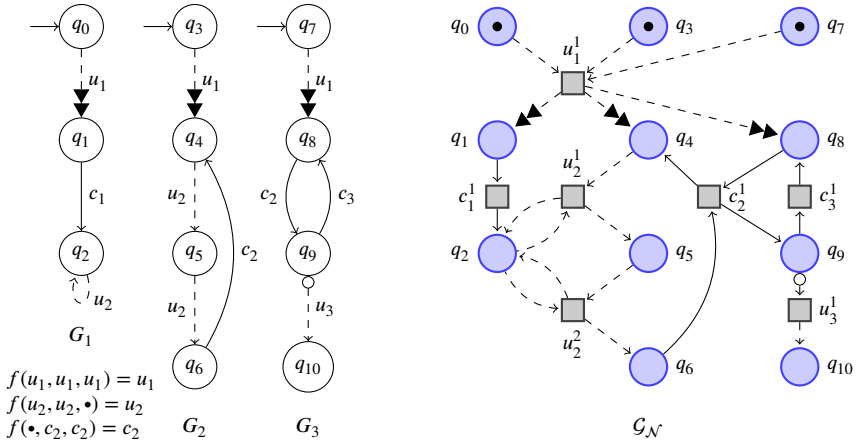


Fig. 22 Game structures G_1, G_2 and G_3 and the Game PN $\mathcal{G}_\mathcal{N}$ of the composition $(G_1|G_2|G_3)_f$

9.2 Computation of the strategy

The concurrent and synchronized composition of Game Structures is a Game PN whose semantics is given in section 8 as a monolithic game structure. Hence, the concurrent and synchronized composition of Game Structures is a game structure as illustrated in Figure 23 for the composition $(G_1|G_2|G_3)_f$ given in Figure 22. Since, by construction, the Game Petri Net of Figure 22 is safe, in order to avoid overloading Figure 23, we represent the marking by a column of marked places.

At this step we can equivalently remove the exponents of the action names. If we consider that the safety objective of the game consists in avoiding the state q_{10}, the most permissive memoryless strategy is:

$$s(\begin{pmatrix} q_1 \\ q_4 \\ q_8 \end{pmatrix}) = \{\langle c_1, \mathbf{0}+\overline{\mathbf{0}}\rangle\},\ s(\begin{pmatrix} q_2 \\ q_6 \\ q_8 \end{pmatrix}) = \{\langle c_2, \mathbf{0}+\overline{\mathbf{0}}\rangle\},\ s(\begin{pmatrix} q_2 \\ q_4 \\ q_9 \end{pmatrix}) = \{\langle c_3, \mathbf{0}\rangle\},$$
$$s(\begin{pmatrix} q_2 \\ q_5 \\ q_9 \end{pmatrix}) = s(\begin{pmatrix} q_2 \\ q_6 \\ q_9 \end{pmatrix}) = \{\langle c_3, \mathbf{0}+\overline{\mathbf{0}}\rangle\} \text{ and is empty for other states.}$$

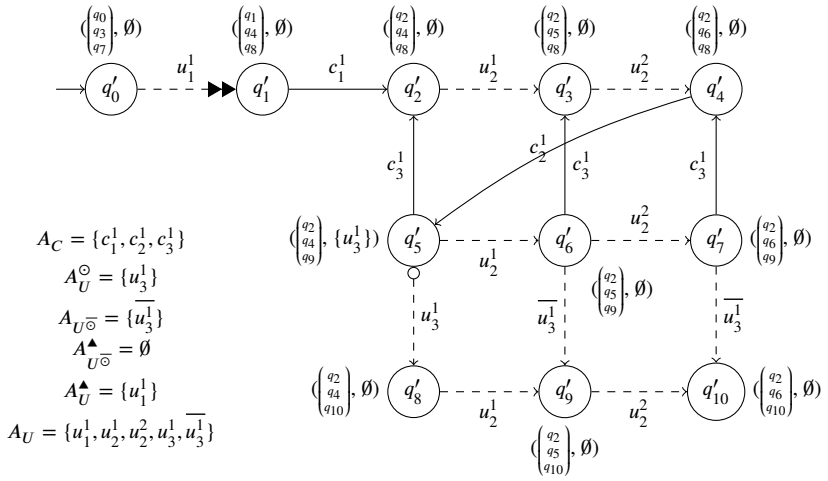


Fig. 23 Monolithic Game structure associated with the composition $(G_1|G_2|G_3)_f$ of Figure 22

.

10 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 [12], 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 [20]. 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 time games. The CTS can be generated from the ROMÉO GUI.

In order to compare the two approaches on a concrete case, a model using Time Petri Nets, more precisely a Time Petri Net control model, and a model using Game Petri Nets have been implemented and compared with respect to memory consumption and execution time. We use a classical case-study presented in [9]. It is a level crossing model with 2 to 4 tracks of independent trains in order to obtain problems of increasing complexity. We use the Time Petri Net models of the trains and the gate given in [9]. The only two controllable actions are *down* and *up* corresponding to the order from the controller to respectively lower and raise the gate. For the non-quantitative model (Game Petri Net), we consider that the lowering of the gate is instantaneous (i.e. merged with *down*) and the raising is ineluctable after the action *up*. Moreover, it is ineluctable that the train will be far after being on the crossing. Our goal is to synthesize the gate controller that ensures that there are no trains on the crossing without the barriers being closed.

Controller synthesis on Time Petri Net is computed by using the method proposed in [17] that extends to Time Petri Nets the timed game algorithm of [12]. It is implemented also in the tool ROMÉO. The results are presented in Table 1.

Table 1 Comparison of computation times and memory consumption between the controller synthesis on Time Petri Nets and Game Petri Nets for the example of a level crossing with 2 to 4 tracks of independent trains. The computer used for these measurements is a MacBook Pro with a 2.6 GHz Intel Core i7 processor and 16 GB of memory.

Number of trains	Time Petri Net		Game Petri Net	
	Execution time (s)	Memory (Mb)	Execution time (s)	Memory (Mb)
2	0.2	5.3	< 0.1	0.3
3	109.4	735.4	< 0.1	0.9
4	$>$ 2h (killed)	$>$ 4Gb	0.1	4.2

The strategy obtained with Game Petri Nets consists in lowering the gate as soon as a train approaches whereas the strategy for the quantitative Time model allows to wait a little before lowering the gate. In this sense this strategy is more precise. However, we can see in Table 1 that the computation times and memory consumption are much more important.

Finally, quantitative time data are often not available as for the following case study.

11 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 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 a case, the second action is avoidable.

11.1 CAN controller driver modeling

The device chosen for the case study is the Microchip CAN controller available in PIC18Cxx8 microcontroller family [22]. 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 two transmit buffers, which are called TXB0 and TXB1, 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 a transmit buffer the device sends it as soon as possible and asserts an interrupts to notify the corresponding TXB has just been emptied.

The model of the driver is presented in Figure 24. We have added two boolean variables: $PW0$ (Pending Write in TXB0) and $PW1$ (Pending Write in TXB1) 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 places, and the part that is executed in the interrupt handler represented by

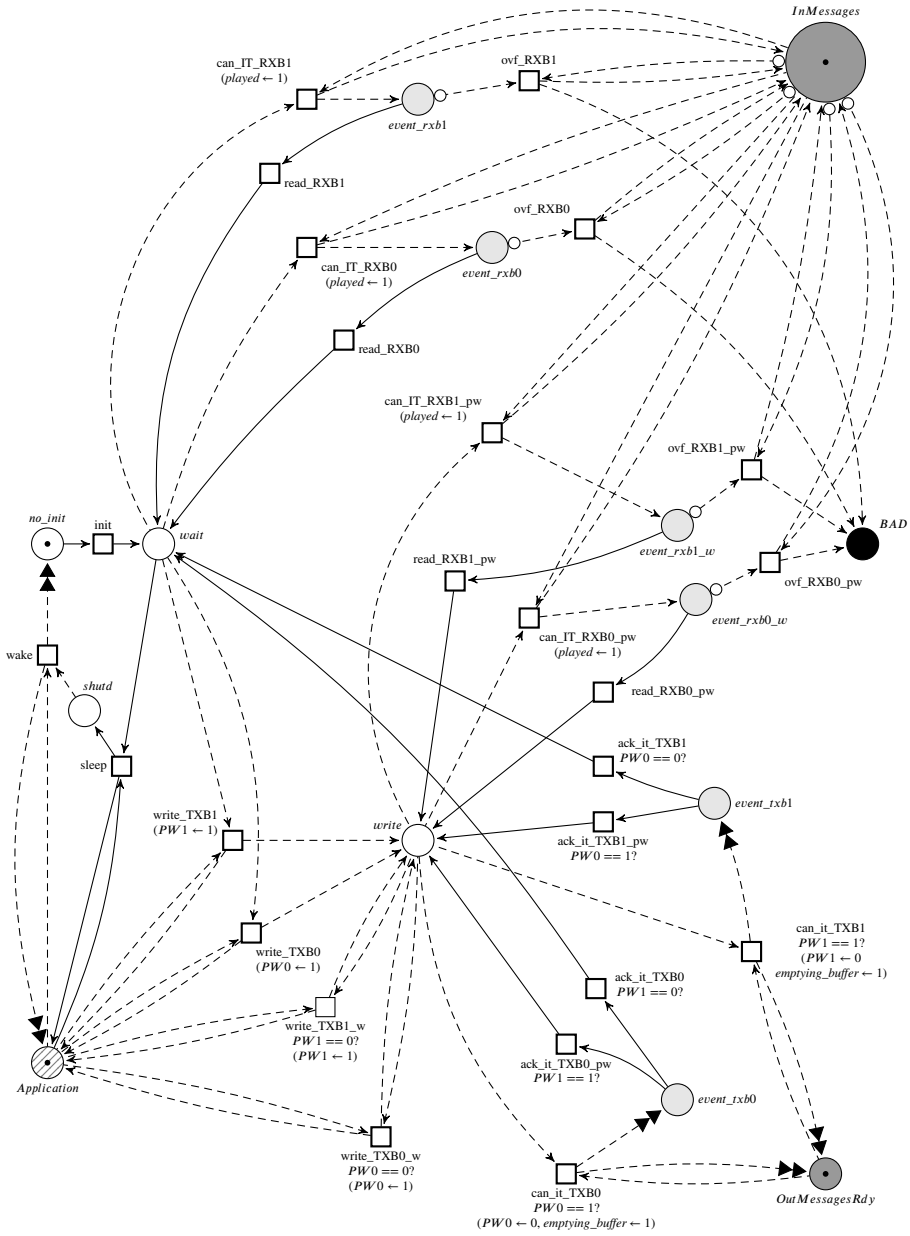


Fig. 24 PIC18Cxx8 CAN controller driver model. Guards are noted with a '?' and updates are noted within parenthesis with a ←.

light gray places. A black bad place that has to be avoided by the controller is added. In addition, the place corresponding to the application environment is hatched in grey and those corresponding to the hardware environment are filled with dark grey.

Starting from the *no_init* place the device can be configured as described above and the driver waits requests in the *wait* place. From there two uncontrollable transitions, corresponding to one of the write requests from the application (write_TXB0 or write_TXB1) may occur and the corresponding boolean variable is set accordingly. Two other uncontrollable transitions correspond to the arrival of a message in one of the receive buffer (can_IT_RXB0 or can_IT_RXB1). From the *write* place we find again the two uncontrollable transitions corresponding to the arrival of a message and also two ineluctable uncontrollable transitions which are fired by the device when TXB0 or TXB1 is emptied. Places whose name begins with *event* represent the entry point(s) of the device interrupt handler(s). From there the controller can fire the transitions corresponding to the processing of the event: read the receive buffer which has been filled (read_RBX0, read_RBX1, read_RBX0_pw and read_RBX1_pw) or acknowledge the emptying of one of the transmit buffers (ack_IT_RBX0, ack_IT_RBX1, ack_IT_RBX0_pw and ack_IT_RBX1_pw).

During the execution of the interrupt handler (light gray places) 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.

The receiving part of this system, whose name starts with *event_rxb* and which requires avoidable transitions, could be modeled with Time Petri Nets. It would indeed be enough to place on the transitions going to the *write* and *wait* places the time guards $[0,0]$ and on those leading to the BAD state, the time guards $]0,\infty[$. But transitions from *write* to places whose name starts with *event_txb* require the notion of ineluctability. Indeed, the date on which these transitions are fired is not known because it depends on the messages sent by the other nodes which transit on the CAN network. Here the use of a Time Petri Net would force to invent an arbitrary date in order to allow modeling.

11.2 Winning strategy

We used our tool ROMÉO [20] for the modeling of this case study and to compute the winning states and the synthesis of the strategy[2]. 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 6 and therefore computes a strategy for the environment to falsify the property. Of course, it does not find any. 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*, which is set when the environment fires a transition corresponding to a message received in RXB0 or RXB1, and *emptying_buffer* which is set when the environment fires a transition corresponding to the emptying of TXB0 or TXB1. That way staying in place *wait* or *write* and returning to one of these states after the environment has played can be distinguished. An additional place is also added, *shutd*. The *shutd* place models the fact that the system may be switched off. The wake transition is the switching on of the system and the sleep transition is the switching off of the system. If the environment decides not to fire any transition, *shutd* will be reachable eventually. The goal of the controller is to reach one of the following states:

- *shutd*;
- *wait* with *played* = true;
- (*write* or *wait*) with *emptying_buffer* = true.

[2] The model and the property to generate the strategy can be downloaded from `https://github.com/jlbirccyn/JDEDS-Model`

Table 2 Memoryless strategy

state	variables	play	next
shutd	–	–	*shutd*
wait	¬*played* & ¬*emptying_buffer*	$\langle \text{sleep}, \mathbf{0} + \overline{\mathbf{0}} \rangle$	*shutd*
no_init	–	$\langle \text{init}, \mathbf{0} + \overline{\mathbf{0}} \rangle$	*wait*
wait	*played* | *emptying_buffer*	–	*wait*
write	–	–	*write*
event_txb0	*emptying_buffer* & *PW1*	$\langle \text{ack\_it\_TXB0\_pw}, \mathbf{0} \rangle$	*write*
event_txb0	*emptying_buffer* & ¬*PW1*	$\langle \text{ack\_it\_TXB0}, \mathbf{0} \rangle$	*wait*
event_txb1	*emptying_buffer* & *PW0*	$\langle \text{ack\_it\_TXB1\_pw}, \mathbf{0} \rangle$	*write*
event_txb1	*emptying_buffer* & ¬*PW0*	$\langle \text{ack\_it\_TXB1}, \mathbf{0} \rangle$	*wait*
event_rxb0_w	–	$\langle \text{read\_RXB0\_pw}, \mathbf{0} \rangle$	*write*
event_rxb1_w	–	$\langle \text{read\_RXB1\_pw}, \mathbf{0} \rangle$	*write*
event_rxb0	–	$\langle \text{read\_RXB0}, \mathbf{0} \rangle$	*wait*
event_rxb1	–	$\langle \text{read\_RXB1}, \mathbf{0} \rangle$	*wait*

ROMÉO finds and computes the winning strategy. Table 2 summarises this strategy.

Starting from *no_init* the controller must play init to reach a winning state. In *wait*, if the environment plays can_IT_RXB0 or can_IT_RXB1, *played* is set and the controller has to play immediately read_RXB0 or read_RXB1, respectively, to go back to *wait*. The environment may play write_TXB0or write_TXB1to go into *write*. From *write*the environment may choose to play write_TXB0_w or write_TXB1_w to fill the second transmit buffer. Or it can play can_IT_RXB0_pw or can_IT_RXB1_pw. As a result *played* is set. In this case the controller has to play immediately read_RXB0_pw or read_RXB1_pw, respectively, to go back to *write*. If the environment decides not to play uncontrollable actions, inevitably, according to which transmit buffer is full, can_it_TXB0 or can_it_TXB1 happens, *emptying_buffer* is set, and the controller returns immediately to state *wait* or *write* by playing ack_it_TXB0 or ack_it_TXB0_pw or ack_it_TXB1 or ack_it_TXB1_pw according to the state of the transmit buffers.

12 Conclusion

We first 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. We have extended these results to Game Petri Nets which can express concurrent behaviors and where an avoidable transition can lose its avoidability by the elapsing of time.

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.

References

1. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theoretical Computer Science **386**(3), 188 – 217 (2007)
2. Altisen, K., Tripakis, S.: Tools for controller synthesis of timed systems. In: 2nd Workshop on Real-Time Tools (RT-TOOLS'2002) (2002)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
4. Bechennec, J.L., Lime, D., Roux, O.H.: Control of DES with urgency, avoidability and ineluctability. In: 19th International Conference on Application of Concurrency to System Design (ACSD'19) (2019)
5. Béchennec, J.L., Roux, O.H., Lime, D.: Contrôle des SED avec urgence, évitabilité et inéluctabilité. In: Modélisation des Systèmes Réactifs (MSR'19), Modélisation des Systèmes Réactifs (MSR'19). Angers, France (2019). URL `https://hal.archives-ouvertes.fr/hal-02415319`
6. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Computer Aided Verification, pp. 121–125. Springer (2007)
7. Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. RAIRO - Theoretical Informatics and Applications (RAIRO: ITA) **36**, 261–275 (2002)
8. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. Software Eng. **17**(3), 259–273 (1991)
9. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time petri nets. In: 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), *Lecture Notes in Computer Science*, vol. 2619, pp. 442–457. Springer Verlag, Warsaw, Poland (2003)
10. Bornot, S., Sifakis, J.: An algebraic framework for urgency. Inf. Comput. **163**(1), 172–202 (2000)
11. Brandin, B.A., Wonham, W.M.: Supervisory control of timed discrete-event systems. IEEE Transactions on Automatic Control **39**(2), 329–342 (1994)
12. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR 2005–Concurrency Theory, pp. 66–80. Springer (2005)
13. Chatain, Th., David, A., Larsen, K.G.: Playing games with timed games. In: A. Giua, M. Silva, J. Zaytoon (eds.) Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09). Zaragoza, Spain (2009)
14. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Strategy improvement for concurrent reachability and safety games. CoRR (2012)
15. De Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: CONCUR 2003-Concurrency Theory, pp. 144–158. Springer (2003)
16. De Alfaro, L., Henzinger, T.A., Majumdar, R.: Symbolic algorithms for infinite-state games. In: CONCUR 2001—Concurrency Theory, pp. 536–550. Springer (2001)
17. Gardey, G., Roux, O.F., Roux, O.H.: Safety control synthesis for time Petri nets. In: 8th International Workshop on Discrete Event Systems (WODES'06), pp. 222–228. IEEE Computer Society Press, Ann Arbor, USA (2006)
18. Golaszewski, C., Ramadge, P.: Control of discrete event processes with forced events. In: Proceedings of the 26th Conference on Decision and Control (1987)
19. Jard, C., Lime, D., Roux, O.H.: Clock Transition Systems. In: 21th international Workshop on Concurrency, Specification and Programming (CS&P 2012). Berlin, Germany (2012)
20. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.M.: Romeo: A parametric model-checker for Petri nets with stopwatches. In: TACAS 2009, *Lecture Notes in Computer Science*, vol. 5505, pp. 54–57. Springer, York, UK (2009)
21. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: STACS 95, pp. 229–242. Springer (1995)
22. Microchip: PIC18CXX8 Data Sheet (DS30475A). High-Performance Microcontrollers with CAN Module. http://ww1.microchip.com/downloads/en/DeviceDoc/30475a.pdf (2000)
23. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987)
24. Thomas, W.: On the synthesis of strategies in infinite games. In: STACS 95, pp. 1–13. Springer (1995)

25. Tripakis, S., Altisen, K.: On-the-fly controller synthesis for discrete and dense-time systems. In: In FM'99, volume 1708 of LNCS, pp. 233–252. Springer Verlag (1999)
26. Wonham, W.M., Ramadge, P.J.: On the supremal controllable sublanguage of a given language. In: Decision and Control, 1984. The 23rd IEEE Conference on, vol. 23, pp. 1073–1080 (1984)