A State Class Based Controller Synthesis Approach for Time Petri Nets

Loriane Leclercq, Didier Lime, Olivier H Roux

A state class based controller synthesis approach for Time Petri Nets*

Loriane Leclercq[1][0000−0002−6254−8691], Didier Lime[1][0000−0001−9429−7586], and Olivier H. Roux[1][0000−0003−1665−0481]

École Centrale de Nantes, CNRS, LS2N, Nantes, France
{Loriane.Leclercq, Didier.Lime, Olivier-h.Roux}@ec-nantes.fr

Abstract. We propose a new algorithm for reachability controller synthesis with time Petri nets (TPN). We consider an unusual semantics of time Petri nets in which the firing date of a transition is chosen in its static firing interval when it becomes enabled. This semantics is motivated i) *by a practical concern:* it aims at approaching the implementation of the controller on a real-time target ; ii) *by a theoretical concern:* it ensures that in the classical state class graph [6], every state in each state class is an actual reachable state from the TPN, which is not the case with the usual interval-based semantics. We define a new kind of two-player timed game over the state class graph and we show how to efficiently and symbolically compute the winning states using state classes. The approach is implemented in the tool Roméo [24]. We illustrate it on various examples including a case-study from [2].

Keywords: Time Petri nets, state classes, timed games, controller synthesis

1 Introduction

Reactive systems allow multiple components to work and interact together and with the environment. In order to ensure the correctness of such systems, we can use controllers to restrict their behavior. Unlike the model-checking problem in which systems are mainly represented as standalone (open-loop systems), the control framework models the interaction between a controller and its environment by using controllable and non-controllable actions. The problem is to design this controller (or a strategy for the controller) that ensures a given specification is valid whatever the environment does (closed-loop system).

The theory of control was first defined over discrete event systems in [26] and then extended to various models. The idea is to model the system and the properties we are interested in, and to implement a controller that makes sure the model behaves correctly w.r.t the properties. Some of the basic properties are reachability and safety. Given a state of the system, the *reachability problem* consists in deciding if this good state is reachable in every execution. Dually the

* This work has been partially funded by ANR project ProMiS ANR-19-CE25-0015

safety problem consists in deciding if we can stay in good states forever for every execution. These control problems usually use controllable events, for example transitions in our case, that will be handled by the controller.

Timed games and time Petri nets for control Games on graphs [27] are a classical and successful framework for controller synthesis in reactive systems. Many such systems however have strong timing requirements and must therefore be modelled using timed formalisms such as timed automata [1]. This leads to the notion of timed games [25], that have been much studied theoretically, and for which efficient clock based algorithms using the so-called zones [16] have been devised [15] and implemented, e. g., in the state-of-the art tool Uppaal-Tiga [3]. The clock-zone based algorithm faces a termination problem that can be solved with an extrapolation/approximation operation. This operation makes things more complex by adding states that are actually not reachable but, in general, do not interfere with the properties we are interested in. In [15], for instance, the authors just assume the clocks to be bounded to avoid dealing with it. And in some cases the added states do interfere with the property of interest [13, 14].

Another classical formalism for timed models is time Petri nets (TPNs) [6]. It is possible to apply to TPNs a semantics very close to the one of timed automata by making clocks appear that measure the duration for which transitions have been enabled and then a state-space computation can be done in a manner similar to timed automata [19]. It thus no surprise that the algorithm of [15] can be lifted to TPNs [18]. It is indeed implemented in the tool Roméo [24], and even extended to account for timing parameters [23].

The classical semantics for TPNs is not the ń explicit clocks ż one of [19] however, but rather the interval-based semantics of [6]. The latter semantics leads naturally to a different kind of abstraction called state classes, which has some advantages: in particular it does not require the use of an extrapolation/approximation operation to ensure termination.

The control problem for time Petri nets has been studied in [20, 2] with forward approaches computing the reachable states over a modified state class graph (SCG) in order to synthesize new constraints to reach winning states. In [2], no constraints are back propagated but the time constraints of a transition that remains enabled contain all its past and have the size of the path during which it remains enabled. In [20], the new constraints are back-propagated to previous classes, until the events when transitions were first newly enabled. Only rectangular constraints are propagated without splitting the state classes, making it impossible to synthesize a controller with a state where a controllable action should be done in disjoint intervals.

Controller implementation Implementing a timed controller on a hardware target such as a microcontroller is not a trivial operation. If a controllable action is to be performed after waiting for a duration within a time interval $[a, b]$, it is necessary to first choose a duration d within this interval and then to go into a non-active wait which is usually achieved by using a timer of duration d that will trigger an interrupt. The program associated with this interruption then

executes the controllable action. If an action of the environment occurs in the meantime that requires a change of the d duration, then the controller must have the ability to change the timer value. This is implicitly considered in most of the works on timed controller synthesis for time Petri nets and timed automata whose computation is based on a backward clock-based algorithm [25, 15, 18]. This implicit re-evaluation of waiting durations makes the implementation of the controller difficult (unless active waiting, i. e. polling, is used, which is not acceptable in a real-time context) because it is not known a priori which actions should cause the re-evaluation of durations.

Contribution We propose an unusual semantics of time Petri nets in which the firing date of a transition is chosen in its static firing interval when it becomes enabled. This semantics is motivated by a practical concern: it aims at approaching the implementation of the controller on a real-time target. The choice of the timer value must be made as soon as the controllable transition is enabled. If this value is to be re-evaluated then the Petri net must model it explicitly.

It is also motivated by a theoretical concern: there is a tight correspondence between this semantics and the construction of the state class graph. It ensures that in the classical state class graph [6], every state in each state class is an actual reachable state from the TPN, which is not the case with the usual interval-based semantics. This semantics was already used in [8], motivated by the use of dynamic firing dates, that can be chosen again at every firing event.

We then leverage the state class abstraction to solve timed games. We define a new kind of timed games based on TPNs and we show how to efficiently and symbolically compute the winning states using state classes. Our method computes backward winning states on the SCG using predecessor operators to split the state classes. The approach, implemented in the tool Roméo [24] is applied on two examples including a case-study from [2].

The rest of this article is organized as follows: Section 2 introduces our new semantics nets and provides the necessary basic definitions of time Petri nets, state classes and the two-player game over this graph, Section 3 describes the computations of winning states for the controller leading to the strategy. Section 4 applies our approach to two case studies. We conclude in Section 5.

2 Definitions

2.1 Preliminaries

We denote the set of natural numbers (including 0) by $\mathbb{N}$ and the set of real numbers by $\mathbb{R}$. We note $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers. For $n \in \mathbb{N}$, we let $[\![0, n]\!]$ denote the set $\{i \in \mathbb{N} \mid i \leq n\}$. For a finite set X, we denote its size by $|X|$.

Given a set X, we denote by $\mathcal{I}(X)$, the set of non-empty, non necessarily bounded, real intervals that have their finite end-points in X. We say that an interval I is non-negative if $I \subseteq \mathbb{R}_{\geq 0}$.

Given sets V and X, a V-valuation (or simply valuation when V is clear from the context) of X is a mapping from X to V. We denote by V^X the set of V-valuations of X. When X is finite, given an arbitrary fixed order on X, we often equivalently consider V-valuations as vectors of $V^{|X|}$.

2.2 Time Petri nets

A time Petri net is a Petri net with time intervals associated with each transition. We propose a slightly different semantics than the one commonly used, in which firing dates are decided at the moment transitions are newly enabled. We consider that input tokens are consumed before the firing of a transition and produced after, so transitions using one of these input tokens have their firing date chosen again.

Definition 1 (Time Petri net). *A* time Petri net *(TPN) is a tuple* $\mathcal{N} = (P, T, F, I_s)$ *where:*

- P *is a finite non-empty set of* places*,*
- T *is a finite set of* transitions *such that* $T \cap P = \emptyset$*,*
- $F : (P \times T) \cup (T \times P)$ *is the* flow function*,*
- $I_s : T \to \mathcal{I}(\mathbb{N})$ *is the* static firing interval *function,*

We assume T *contains at least one transition* t_{init} *and* P *contains at least a place* p_0 *such that* $(p_0, t_{\mathsf{init}}) \in F$*,* $I_s(t_{\mathsf{init}}) = [0, 0]$*, for all* $p \in P \backslash \{p_0\}$*,* $(p, t_{\mathsf{init}}) \notin F$ *and for all* $t \in T$*,* $(p_0, t) \notin F$*. We also assume that for all* $t \in T$ *there exists* $p \in P$ *such that* $(p, t) \in F$*.*

Places of a Petri net can contain *tokens*. A *marking* is then usually an $\mathbb{N}$-valuation of P giving the number of tokens in each place.

Remark 1. For the sake of simplicity, we consider only *safe* nets, i.e., nets in which there is always at most 1 token in each place and where all arcs have weight 1. All subsequent developments can be generalised without any difficulty to more complex discrete dynamics provided the net remains *bounded*, i.e., there is a constant K such that all places never contain more than K tokens. Boundedness is an appropriate restriction since the control problem is undecidable for unbounded TPN. The proof of [22] extends directly to our semantics.

We therefore define a marking as the set of the places of P containing a token. We say those places are *marked*.

Usually we define an initial marking for the net. Without loss of generality, we consider here that all places are initially empty except p_0 which is marked. An immediate transition t_{init} sets the initial marking by firing. By construction, while it has not fired, no other transition can fire.

Given a transition t, we define the sets of its input places $\mathsf{Pre}(t) = \{p \mid (p, t) \in F\}$ and of its output places $\mathsf{Post}(t) = \{p \mid (t, p) \in F\}$.

Definition 2 (Enabled and persistent transitions). *A transition t is said to be* enabled *by marking m if all its input places are marked: $\mathsf{Pre}(t) \subseteq m$. A transition t is said to be* persistent *by firing transition t' from marking m if it is not fired and still enabled when removing tokens from the input places of t': $t \neq t'$ and $\mathsf{Pre}(t) \subseteq m \setminus \mathsf{Pre}(t')$. We say that t is* newly enabled *by firing t' from m, if t is enabled before and after the firing of t' but not persistent. We denote by $\mathsf{en}(m)$, $\mathsf{pers}(m,t)$ and $\mathsf{newen}(m,t)$ respectively the sets of enabled, persistent and newly enabled transitions.*

Remark 2. The definition of newly enabled transitions uses a reset policy in which every transition that is disabled by a token taken by $\mathsf{Pre}(t)$ is considered newly enabled even if it is enabled again after putting back tokens from $\mathsf{Post}(t)$. And in particular the fired transition itself is always considered newly enabled. This is the usual memory policy called intermediate semantics, see [5] for details and comparison with other semantics

Definition 3 (States and semantics of a TPN). *A* state *of a TPN is a pair $s = (m, \theta)$ with $m \subseteq P$ a marking and $\theta : T \to \mathbb{R}_{\geq 0} \cup \{\bot\}$ a function that associates a firing date with every transition t enabled at marking m ($t \in \mathsf{en}(m)$) and $\bot$ to all other transitions. For any valuation on transitions θ, we denote by $\mathsf{tr}(\theta)$ the set of transitions t such that $\theta(t) \neq \bot$. We will use θ_i to denote $\theta(t_i)$.*

The semantics of a TPN is a Timed Transition System *$(S, s_0, \Sigma, \to)$ with:*

- *S the set of all possible states,*
- *an initial state $s_0 = (\{p_0\}, \theta_0) \in S$ with $\theta_0(t_{\mathsf{init}}) = 0$, and $\forall t \neq t_{\mathsf{init}}, \theta_0(t) = \bot$,*
- *a labelling alphabet Σ divided between two types of letters: $t_f \in T$ and $d \in \mathbb{R}_{\geq 0}$,*
- *the transition relation between states $\to \subseteq S \times \Sigma \times S$ and, $(s, a, s') \in \to$, denoted by $s \xrightarrow{a} s'$:*
  - *either $(m, \theta) \xrightarrow{t_f} (m', \theta')$ for $t_f \in T$ when:*
    1. *$t_f \in \mathsf{en}(m)$ and $\theta_f = 0$*
    2. *$m' = (m \setminus \mathsf{Pre}(t_f)) \cup \mathsf{Post}(t_f)$*
    3. *$\forall t_k \in T, \theta'_k \in I_s(t_k)$ if $t_k \in \mathsf{newen}(m, t_f)$, $\theta'_k = \theta_k$ if $t_k \in \mathsf{pers}(m, t_f)$, and $\theta'_k = \bot$ otherwise*
  - *or $(m, \theta) \xrightarrow{d} (m, \theta')$ when: $d \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t_k \notin \mathsf{en}(m), \theta_k = \bot$, and $\forall t_k \in \mathsf{en}(m), \theta_k - d \geq 0$ and $\theta'_k = \theta_k - d$.*

Remark 3. Had we not assumed a unique transition t_{init} enabled, with a time interval reduced to $[0, 0]$, we would have in general an infinity of initial states corresponding to all possible choices of function θ_0 with values in the static firing intervals of enabled transitions, which is not a problem but is a small inconvenience. Otherwise for the two-player game construction to follow, we would have needed a first half turn to reach a correct state before even starting.

A *run* in the semantics of a TPN is a possibly infinite sequence $s_0 a_0 s_1 a_1 s_2 a_2 \cdots$ such for all i, $s_i \xrightarrow{a_i} s_{i+1}$. We denote by $\mathsf{seq}(\rho)$ the subsequence of ρ containing exactly the transitions $a_0 a_1 a_2 \cdots$.

In this semantics the choice of firing date occurs directly when the transition is newly enabled whereas in the classical semantics of [6] this choice is postponed to the moment the transition fires. This is already used in [21] but with probabilistic choices instead of non-deterministic one.

We have chosen this semantics because it will allow us to more precisely relate states and state classes as defined in the next section. Such a close relation has never been achieved with the semantics of [6], leading to further refinements into so-called atomic state classes [10].

2.3 State classes

The number of states from a TPN is not finite in general because of the density of the static intervals. There are several finite representations abstracting the state space of a TPN using various methods and one of them is the state class graph. One of its benefits is to be finite as long as the TPN is bounded, i.e. the number of tokens in each place is bounded (by 1 in the case of safe nets).

Definition 4 (State class). *Let $\sigma = t_1...t_n$ be a sequence of transitions. The* state class *K_σ is the set of all states obtained by firing σ in order, with all possible delays* before *each fired transition. Clearly, all states in K_σ share the same marking m, and so we write $K_\sigma = (m, D)$ where D, called the* firing domain*, is the union of all possible firing date functions for those states.*

The firing domain D is a set of valuations of transitions. With an arbitrary order on transitions, and ignoring $\perp$ values, such a valuation can be seen as a point in $\mathbb{R}_{\geq 0}^{|\mathsf{en}(m)|}$. We will therefore consider such sets of valuations as subsets of $\mathbb{R}_{\geq 0}^{|\mathsf{en}(m)|}$. And as we will see, firing domains are actually a special kind of convex polyhedra in that space.

As a direct consequence of Definition 4, we have the following lemma:

Lemma 1. *Let $K_\sigma = (m, D)$. Let $s = (m, \theta)$; then $\theta \in D$ if and only if there exists a run ρ from the initial state s_0 to s, such that $\mathsf{seq}(\rho) = \sigma$ and either σ is empty or ρ ends with a transition firing.*

Remark 4. With the usual semantics of [6], only the *if* part holds [7], because the timing part of states in that semantics assigns intervals to enabled transition and an arbitrary interval taken from D does not necessarily correspond to a reachable state. A state can contain an interval overlapping two adjacent intervals grouped in a class, but that is not reachable.

We can naturally now extend the notions of enabled, persistent, and newly enabled transitions to state classes: $\mathsf{en}((m, D)) = \mathsf{en}(m)$, $\mathsf{newen}((m, D), t) = \mathsf{newen}(m, t)$, and $\mathsf{pers}((m, D), t) = \mathsf{pers}(m, t)$.

We have the following lemma:

Lemma 2. *Let $K_\sigma = (m, D)$ and $K_{\sigma.t_f} = (m', D')$, with $t_f \in \mathsf{en}(m)$. We have:*

$$\theta' \in D' \text{ iff } \exists \theta \in D \text{ s. t. } \begin{cases} \forall i \in \mathsf{en}(m), \theta_i - \theta_f \geq 0 \\ \forall i \in \mathsf{pers}(m, t_f), \theta'_i = \theta_i - \theta_f \\ \forall i \in \mathsf{newen}(m, t_f), \theta'_i \in I_s(i) \end{cases}$$

Proof. By Lemma 1, for all states $s' = (m', \theta') \in K_{\sigma.t_f}$ there exists a run ρ' that goes from the initial state s_0 to s' such that $\mathsf{seq}(\rho') = \sigma.t_f$. Also ρ ends with the firing of t_f.

Let $s = (m, \theta)$ and $s'' = (m, \theta'')$ be the states in ρ' such that $\exists d. s \xrightarrow{d} s'' \xrightarrow{t_f} s'$. Possibly, we have $s = s''$. Let ρ be the prefix of ρ' ending in s, then $\mathsf{seq}(\rho) = \sigma$ and ρ does not end with a delay, so $s \in K_\sigma$ by Lemma 1. We thus have $\theta \in D$ and Definition 4 directly implies the three expected conditions because, from top to bottom, t_f is firable, we must delay until θ_f is 0, and the firing dates for newly enabled transitions are chosen in their static firing intervals. □

From Lemma 2, D' is not empty if and only if there exists θ in D such that for all $i \in \mathsf{en}(m)$, $\theta_i \geq \theta_f$. In that case we say that t_f is *firable* from (m, D).

Algorithm 1 then follows straightforwardly from Lemma 1 to compute $K' = (m', D')$ from $K = (m, D)$ by firing firable transition t_f.

Algorithm 1 Successor (m', D') of (m, D) by firing firable transition t_f

1: $m' \leftarrow (m \setminus \mathsf{Pre}(t_f)) \cup \mathsf{Post}(t_f)$
2: $D' \leftarrow D \wedge \bigwedge_{i \neq f, i \in \mathsf{en}(m)} \theta_f \leq \theta_i$
3: for all $i \in \mathsf{en}(m \setminus \mathsf{Pre}(t_f)), i \neq f$, add variable θ'_i to D', constrained by $\theta'_i = \theta_i - \theta_f$
4: eliminate (by existential projection) variables θ_i for all i from D'
5: for all $i \in \mathsf{newen}(m, t_f)$, add variable θ''_i to D', constrained by $\theta''_i \in I_s(i)$

The state class associated with the empty sequence ϵ contains the set of initial states, here reduced to a singleton: $K_\epsilon = (m_0, \{\theta_0\})$.

Algorithm 1 corresponds to the classical state class computation from [6]. The initial class is also what we would obtain with that construction. It is well-known that those state classes can be represented and computed using a special kind of convex polyhedra encoded in the efficient data structure called *difference bound matrix* (DBM) [6, 17]. An efficient way to directly compute successor classes is given in [11, 12].

Definition 5. *Starting from K_ϵ, we can construct an infinite directed tree (labeled by fired transitions) by inductively computing successors by firable transitions. The* State class graph (SCG) *$\mathcal{G}$ is the graph obtained by quotienting this tree with the equality relation on state classes (same marking, and same firing domain).*

2.4 Two-player game on the State Class Graph

Since we are interested in controller synthesis, from now on, the set of transitions is partitioned between two sets T_c and T_u which contain respectively controllable and uncontrollable transitions. The controllable transitions are controlled by a controller, in the sense that it can choose their firing dates, and the order of firing but uncontrollable transitions can be fired in between.

For short, we define $\mathsf{newen_u}(m,t) = T_u \cap \mathsf{newen}(m,t)$, $\mathsf{newen_c}(m,t) = T_c \cap \mathsf{newen}(m,t)$, $\mathsf{en_u}(m) = T_u \cap \mathsf{en}(m)$ and $\mathsf{en_c}(m) = T_c \cap \mathsf{en}(m)$. We also extend these notations to state classes as before.

We now define a game over the TPN $\mathcal{N}$ that simulates the behavior of controllable and uncontrollable transitions in order to decide whether a set of states is always reachable by choosing the right controllable firing dates or not. And if this is the case, a strategy for the controller will be constructed.

A round in the game is in three steps:

1. the controller chooses a firable transition $t_c \in T_c$ that he wants to fire first;
2. the environment chooses either to fire a firable transition $t_u \in T_u$ or to let the controller fire t_c;
3. both choose independently the firing dates of their newly enabled transitions.

Definition 6. *Let $\mathcal{N} = (P, T, F, I_s)$, a time Petri net with $T = T_c \cup T_u$ and $T_c \cap T_u = \emptyset$ and $(S, s_0, \Sigma, \to)$, its semantics, an* arena *is a tuple $\mathcal{A} = (S, \to, Pl, (\mathsf{Movt}_i)_{i\in Pl}, (\mathsf{Movf}_i)_{i\in Pl}, \mathsf{Trans})$ with:*

- *$Pl = (Pl_u, Pl_c)$ the two players of the game: the environment (Pl_u) and the controller (Pl_c). The controller plays over controllable transitions, whereas the environment plays over uncontrollable transitions.*
- *$\mathsf{Movt}_u : S \times T_c \to 2^T$ and $\mathsf{Movt}_c : S \to 2^T$ rule the choices of transitions:*

$$\mathsf{Movt}_c(m,\theta) = \{t_i \mid t_i \in \mathsf{en_c}(m) \wedge \theta_i = \min_{t_k \in \mathsf{en}(m)} \theta_k\}$$
$$\mathsf{Movt}_u((m,\theta),t_c) = \{t_i \mid t_i \in \mathsf{en_u}(m) \wedge \theta_i = \min_{t_k \in \mathsf{en}(m)} \theta_k\} \cup \{t_c\}$$

- *$\mathsf{Movf}_u : S \times T \to 2^{\mathbb{R}^{T_u}_{\geq 0}}$ and $\mathsf{Movf}_c : S \times T \to 2^{\mathbb{R}^{T_c}_{\geq 0}}$ rule the choices of firing dates:*

$$\mathsf{Movf}_c((m,\theta),t_i) = \left\{ \theta^c \in \mathbb{R}^{T_c}_{\geq 0} \;\middle|\; \begin{array}{l} \theta^c_k \in I_s(t_k) \text{ if } t_k \in \mathsf{newen}(m,t_i) \\ \theta^c_k = \theta_k - \theta_i \text{ if } t_k \in \mathsf{pers}(m,t_i) \\ \theta^c_k = \bot \text{ otherwise} \end{array} \right\}$$
$$\mathsf{Movf}_u((m,\theta),t_i) = \left\{ \theta^u \in \mathbb{R}^{T_u}_{\geq 0} \;\middle|\; \begin{array}{l} \theta^u_k \in I_s(t_k) \text{ if } t_k \in \mathsf{newen}(m,t_i) \\ \theta^u_k = \theta_k - \theta_i \text{ if } t_k \in \mathsf{pers}(m,t_i) \\ \theta^u_k = \bot \text{ otherwise} \end{array} \right\}$$

– *finally,* $\mathsf{Trans} : S \times T \times T \times \mathbb{R}_{\geq 0}^{T_c}, \mathbb{R}_{\geq 0}^{T_u} \to S$ *combines all the choices of the players and gives the resulting state:*

$$\mathsf{Trans}(s, t_c, t_u, \theta^c, \theta^u) = ((m \setminus \mathsf{Pre}(t_u)) \cup \mathsf{Post}(t_u), \theta^c \cup \theta^u)$$

when $t_c \in \mathsf{Movt}_c(s)$, $t_u \in \mathsf{Movt}_u(s, t_c)$, $\theta(t_u) = \min_k(\theta(t_k))$, $t_u \in T_u \vee t_u = t_c$, $\theta^c \in \mathsf{Movf}_c(s, t_u)$ *and* $\theta^u \in \mathsf{Movf}_u(s, t_u)$.
Note that $\theta^u \cup \theta^c$ *is a disjoint union and is in* $\mathbb{R}_{\geq 0}^{T}$ *because* $\mathbb{R}_{\geq 0}^{T_u}$ *and* $\mathbb{R}_{\geq 0}^{T_c}$ *are disjoints and their union is* $\mathbb{R}_{\geq 0}^{T}$.

A reachability game $\mathcal{R} = (\mathcal{A}, \mathsf{Goal})$ *consists of an arena and a set* $\mathsf{Goal} \subseteq S$ *of goal states. The objective of* Pl_c*, the controller, is to reach a state in* Goal *and the objective of* Pl_u*, the environment, is to avoid these states.*

Definition 7. *A* play *in an arena is a finite or infinite word* $s_0 s_1 ... s_n$ *over the alphabet* S *such that*

$$s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 .... \xrightarrow{t_n} s_n$$

with $\forall i, \exists \theta_i^u, \theta_i^c, t_{ci}, t_{ui}.\mathsf{Trans}(s_i, t_{ci}, t_{ui}, \theta_i^c, \theta_i^u) = s_{i+1}$ *and* $t_{ci} \in \mathsf{Movt}_c(s_i)$, $t_{ui} \in \mathsf{Movt}_u(s_i, t_{ci})$, $\theta_i^u \in \mathsf{Movf}_u(s_i, t_{ui})$ *and* $\theta_i^c \in \mathsf{Movf}_c(s_i, t_{ui})$.

Definition 8. *A* strategy *for the environment* Pl_u *(resp. the controller* Pl_c*) is a function* $\sigma_u : S \times T_c \to T \times \mathbb{R}_{\geq 0}^{T_u}$ *(resp.* $\sigma_c : S \to T_c \times \mathbb{R}_{\geq 0}^{T_c}$*).*[1]

Definition 9. *A play* $s_0 s_1 ...$ conforms to *strategy* σ_c *(resp.* σ_u*) if at each position* i *(except the last in case of a finite play):* $\exists t_{ci}, t_{ui}, \theta_i^c, \theta_i^u$ *s. t.* $\sigma_c(s_i) = (t_{ci}, \theta_i^c)$ *(resp.* $\sigma_u(s_i, t_{ci}) = (t_{ui}, \theta_i^u)$*) and* $Trans(s_i, t_{ci}, t_{ui}, \theta^c, \theta^u) = s_{i+1}$.

Definition 10. *A* maximal play *is a play that is either infinite or finite and such that from the last state no new marking is reachable.*

Definition 11. *A maximal play is* winning *for the controller if there is a position* n *such that* $s_n \in \mathsf{Goal}$*. Otherwise, the play is winning for the environment.*

A strategy is winning *for a player if and only if all plays conforming to this strategy are winning for that player.*

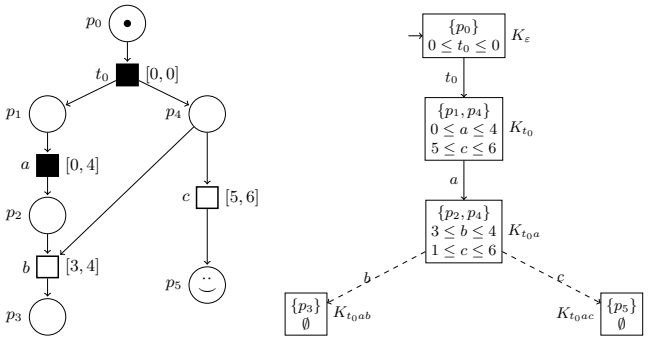
Remark 5. This game is a *two-player determined concurrent game* and we always have either Pl_c wins or Pl_u wins, but not both.

3 Computing the winning states

The construction of a strategy for the controller is based on the state class graph $\mathcal{G}$. To construct such a winning strategy over this graph, we will use a backward process to recursively compute the controllable predecessors of the target states until a fixed point is reached.

[1] Usualy, strategies are defined with the whole trace as memory but we will see in subsection 3.3 that by construction we only need memoryless strategies.

A TPN $\mathcal{N}$ and its state class graph $\mathcal{G}$ are shown in figure 1a and 1b. We use black squares to depict controllable transitions and white ones for uncontrollable transitions. In $\mathcal{G}$, dashed arrows are used for uncontrollable transitions. Goal states are those with a token in place p_5. To reach such a state from states in $ClassG_{t_0a}$, we must fire transition c before b. This is the condition that we will have to propagate during the backward process.



(1a) Time Petri net $\mathcal{N}$ (1b) State class graph $\mathcal{G}$

Definition 12. *Let $C \xrightarrow{t_f} C'$ be a transition in $\mathcal{G}$ and let B be a subset of the class C'.*

We define the set of predecessors $\mathsf{Pred}_{C \xrightarrow{t_f} C'}(B)$ *of $B \subseteq C'$ in C by transition t_f:* $\mathsf{Pred}_{C \xrightarrow{t_f} C'}(B) = \{s \in C \mid \exists s'.s \xrightarrow{t_f} s' \in B\}$.

We further define two sets $\mathsf{cPred}_{C \xrightarrow{t_f} C'}(B)$ and $\mathsf{uPred}_{C \xrightarrow{t_f} C'}(B)$ for the *controllable* and *uncontrollable predecessors* of a subset B of C' in class C and by firing transition t_f. The controllable predecessors correspond to states from which the controller can force to reach B and the uncontrollable predecessors correspond to states from which the controller can not force to avoid B.

Definition 13. *Without loss of generality we suppose $\{t_1, ..., t_n\} = \mathsf{newen_c}(C, t_f)$ and $\{t_{n+1}, ..., t_{n+k}\} = \mathsf{newen_u}(C, t_f)$. We define:*

$$\mathsf{cPred}_{C \xrightarrow{t_f} C'}(B) = \left\{ (m,\theta) \in C \;\middle|\; \begin{array}{l} \forall t_i \in \mathsf{newen_c}(C,t_f), \exists \theta'_i \in I_s(t_i) \text{ s. t.} \\ \forall t_{n+j} \in \mathsf{newen_u}(C,t_f), \forall \theta'_{n+j} \in I_s(t_{n+j}), \\ s \xrightarrow{t_f} s' = (m',\theta') \in B \\ \textit{where } \forall i \in [\![1, n+k]\!], \theta'(t_i) = \theta'_i \\ \textit{ and } \forall i \in [\![1, l]\!], \theta'(t_{n+k+i}) = \theta(t_{n+k+i}) - \theta(t_f) \end{array} \right\}$$

And:

$$\mathsf{uPred}_{C \xrightarrow{t_f} C'}(B) = \left\{ (m,\theta) \in C \middle| \begin{array}{l} \forall t_i \in \mathsf{newen_c}(C,t_f), \forall \theta'_i \in I_s(t_i) \text{ s. t.} \\ \forall t_{n+j} \in \mathsf{newen_u}(C,t_f), \exists \theta'_{n+j} \in I_s(t_{n+j}), \\ s \xrightarrow{t_f} s' = (m',\theta') \in B \\ \textit{where } \forall i \in [\![1, n+k]\!], \theta'(t_i) = \theta'_i \\ \quad \textit{and } \forall i \in [\![1,l]\!], \theta'(t_{n+k+i}) = \theta(t_{n+k+i}) - \theta(t_f) \end{array} \right\}$$

In order to symbolically compute these sets of states, we will need a few operators on sets of valuations.

In the following, D and D' are sets of valuations and we denote valuations by the sequences of their non-$\perp$ values.

We first define the classical *existential projection*:

Definition 14. *For any set of valuations D s. t. $\forall \theta \in D, \mathsf{tr}(\theta) = \{t_1, \ldots, t_{n+k}\}$,*

$$\pi^{\exists}_{\{t_1,\ldots,t_n\}}(D) = \{(\theta_1 ... \theta_n) \mid \exists \theta_{n+1}, ..., \theta_{n+k}, (\theta_1 ... \theta_{n+k}) \in D\}$$

We also define a less usual *universal projection* of D' inside D:

Definition 15. *For any two sets of valuations D and D' such that $D' \subseteq D$ and $\forall \theta \in D, \mathsf{tr}(\theta) = \{t_1, \ldots, t_{n+k}\}$,*

$$\pi^{\forall}_{\{t_1,\ldots,t_n\}}(D, D') = \left\{ (\theta_1 ... \theta_n) \middle| \begin{array}{l} \exists \theta_{n+1}, ..., \theta_{n+k}, (\theta_1 ... \theta_{n+k}) \in D \\ \wedge \forall \theta_{n+1}, ..., \theta_{n+k}, (\theta_1 ... \theta_{n+k}) \in D \\ \qquad \implies (\theta_1 ... \theta_{n+k}) \in D' \end{array} \right\}$$

We also need an *extension operation*:

Definition 16. *For any set of valuations D s. t. $\forall \theta \in D, \mathsf{tr}(\theta) = \{t_1, \ldots, t_n\}$,*

$$\pi^{-1}_{\{t_1,\ldots,t_{n+k}\}}(D) = \{(\theta_1 ... \theta_{n+k}) \mid (\theta_1 ... \theta_n) \in D \textit{ and } \forall i, \theta_{n+i} \geq 0\}$$

Finally, we define a *backward in time* operator:

Definition 17. *For any set of valuations D s. t. $\forall \theta \in D, \mathsf{tr}(\theta) = \{t_1, \ldots, t_n\}$ and for $t_f \neq t_i$ for all $i \in [\![1, n]\!]$,*

$$D + t_f = \{(\theta'_1 \ldots \theta'_n \theta'_f) \mid (\theta_1 \ldots \theta_n) \in D, \theta'_f \geq 0 \textit{ and } \forall i, \theta'_i = \theta_i + \theta'_f\}$$

Remark 6. The universal projection is parameterized by two sets of valuations unlike the existential projection. The reason for this choice is that we only want states that after extension are correct regarding the semantics of the TPN. And since in our construction we will use this projection with $\{t_{n+1}, ...t_{n+k}\} \in \mathsf{newen}(A,t)$ we can easily justify this choice

because choosing a firing date θ_{n+i} outside of $I_s(t_{n+i})$ is not relevant for a newly enabled transition. So it is natural to restrict projections inside a set of valuations to those that can possibly be extended in some correct and reachable states, namely states that are part of a class in the SCG.

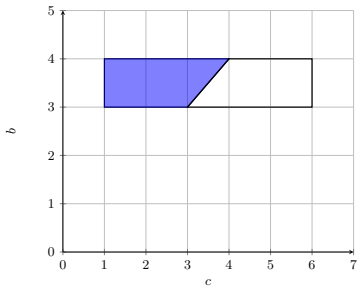
Note that for now the extension operation gives us this kind of irrelevant valuations, therefore we will need to intersect them with the domain of a state class from the graph beforehand.

The universal projection is expressible with set complements and existential projections only, as stated in the following proposition. We denote by $\overline{D}$ the complement of D, i. e., $\overline{D} = \{s \mid s \notin D\}$.

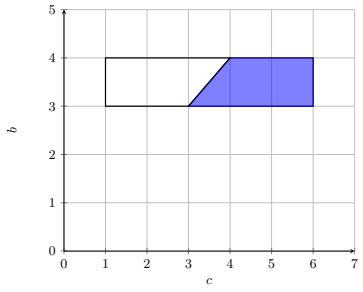
Proposition 1. *Let* $\tau = \{t_1, ..., t_n\}$ $\forall \tau \subseteq T, \pi^{\forall}_{\tau}(D, D') = \pi^{\exists}_{\tau}(D) \cap \overline{\pi^{\exists}_{\tau}(\overline{D'} \cap D)}$.

Due to the lack of space we ommit the proof.

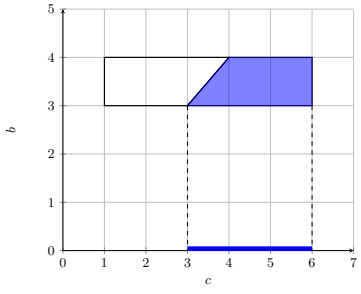
Example 1. A graphical way to see the intuition behind the universal projection in two dimensions is given in figure 2. We use for this example the TPN in figure 1a, with D' being the part of the domain of the state class K_σ for the firing sequence $\sigma = t_0.a$, that allows to put a token in p_5.



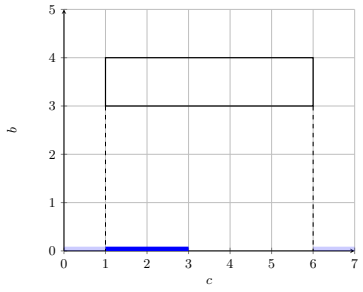
(a) Set D' in blue in the rectangle of $D_{t_0 a}$



(b) $D_{t_0 a} \cap \overline{D'}$



(c) $\pi^{\exists}_{\{c\}}(D_{t_0 a} \cap \overline{D'})$



(d) $\pi^{\exists}_{\{c\}}(D_{t_0 a}) \cap \overline{\pi^{\exists}_{\{c\}}(D_{t_0 a} \cap \overline{D'})} = \pi^{\forall}_{\{c\}}(D_{t_0 a}, D')$

Fig. 2: Example of universal projection

3.1 Symbolic computation for Pred ()

We now have the necessary operations on sets of valuations to be able to symbolically compute the predecessors of a subset of a class. In the following we always assume that $B \subseteq C'$ and in particular we thus have $\mathsf{en}(C') = \mathsf{en}(B)$.

We will use the projection operators as building blocks for the $\mathsf{cPred}()$ and $\mathsf{uPred}()$ predecessor operators.

Proposition 2 ($\mathsf{cPred}_{C \xrightarrow{t_f} C'}(B)$ **computation).** *Let* $C = (m, D)$ *and* $C' = (m', D')$. *Consider* $B = (m', D'') \subseteq C'$, *and let* $\mathsf{cPred}_{C \xrightarrow{t_f} C'}(B) = (m, D_p)$. *Then:*

$$D_p = D \cap \pi^{-1}_{\mathsf{en}(C)}\Big(\pi^{\exists}_{\mathsf{pers}(C,t_f)}\big(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')\big) + t_f\Big)$$

Proof. As in the definition of $\mathsf{cPred}()$, we have chosen without loss of generality that $\mathsf{newen_c}(C, t_f) = \{t_1, ..., t_n\}$, $\mathsf{newen_u}(C, t_f) = \{t_{n+1}, ..., t_{n+k}\}$ and $\mathsf{pers}(C, t_f) = \{t_{n+k+1}, ..., t_{n+k+l}\}$.

$\rightarrow$: First suppose $s = (m, \theta) \in \mathsf{cPred}_{C \xrightarrow{t_f} C'}(B)$. By Definition 13, we have $s \in C$ and $\forall t_i \in \mathsf{newen_c}(C, t_f), \exists \theta'_i \in I_s(t_i)$ s. t. $\forall t_{n+j} \in \mathsf{newen_u}(C, t_f), \forall \theta'_{n+j} \in I_s(t_{n+j}), s \xrightarrow{t_f} s' = (m', \theta') \in B$ where $\forall i \in [\![1, n+k]\!], \theta'(t_i) = \theta'_i$ and $\forall i \in [\![1, l]\!], \theta'_{n+k+i} = \theta_{n+k+i} - \theta_f$.

Since $B \subseteq C'$, we also have $s' \in C'$. Let θ^1 be the valuation such that $\forall i \in [\![1, n]\!], \theta^1_i = \theta'_i$ and $\forall i \in [\![1, l]\!], \theta^1_{n+k+i} = \theta_{n+k+i} - \theta_f$ and $\forall i \in [\![1, k]\!], \theta^1_{n+i} = \bot$, i. e., we have removed uncontrollable newly enabled transitions from θ''. Then from Definition 15, $\theta^1 \in \pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')$ because the only way to assign values to newly enabled transitions within D' is to take them in their static firing interval, and because those intervals are all non-empty. Further define θ^2 from θ^1 by removing controllable newly enabled transitions: $\forall t_i \in \mathsf{newen_c}(C, t_f), \theta^2_i = \bot$ and $\forall t_i \notin \mathsf{newen_c}(C, t_f), \theta^2_i = \theta^1_i$. Then by construction, $\theta^2 \in \pi^{\exists}_{\mathsf{pers}(C,t_f)}(\{\theta^1\})$, and hence $\theta^2 \in \pi^{\exists}_{\mathsf{pers}(C,t_f)}(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D''))$. In θ^2, exactly all persistent transitions have a value different from $\bot$, so if we add θ_f to all of those, we obtain a new valuation θ^3, with $\theta^3 = \theta^2_i = (\theta_i - \theta_f) + \theta_f = \theta_i$ for all persistent transition t_i, and $\theta^3_i = \bot$ for all other transitions. By construction, $\theta^3 \in \pi^{\exists}_{\mathsf{pers}(C,t_f)}(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')) + t_f$. Finally, we can extend θ^3 with values for the transitions in $\mathsf{en}(m) \setminus \mathsf{pers}(C, t_f)$ in the following manner: let θ^4 defined by $\theta^4_i = \theta^3_i$ for all persistent transitions t_i, $\theta^4_i = \theta_i$ for all $t_i \in \mathsf{en}(m) \setminus \mathsf{pers}(C, t_f)$ and $\theta^4_i = \bot$ for all other transitions. Then clearly, $\theta^4 \in \pi^{-1}_{\mathsf{en}(C)}\Big(\pi^{\exists}_{\mathsf{pers}(C,t_f)}\big(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')\big) + t_f\Big)$. But since θ^4 has exactly all the same values for transitions as θ, we have the expected result.

$\leftarrow$: consider $\theta \in D \cap \pi^{-1}_{\mathsf{en}(C)}\Big(\pi^{\exists}_{\mathsf{pers}(C,t_f)}\big(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')\big) + t_f\Big)$. Let $s = (m, \theta)$. By definition of π^{-1}, there exists a θ^1 in $\pi^{\exists}_{\mathsf{pers}(C,t_f)}(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')) +$

t_f such that for all persistent transitions t_i, $\theta_i^1 = \theta_i$, and $\theta_f \neq \bot$, and for all other transitions t_i, $\theta_i^1 = \bot$. By definition of the extension operator, there exists a valuation $\theta^2 \in \pi^{\exists}_{\mathsf{pers}(C,t_f)}(\pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D''))$, such that for all persistent transitions t_i, $\theta_i^2 + \theta_f = \theta_i$, and for all other transitions t_i, $\theta_i^2 = \bot$.

By definition of $\pi^{\exists}$, there exists a valuation $\theta^3 \in \pi^{\forall}_{\substack{\mathsf{newen_c}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D', D'')$ such that for all persistent transitions t_i, we still have $\theta_i^3 = \theta_i^2 = \theta_i - \theta_f$, and for all newly enabled controllable transitions t_i we have $\theta_i^3 \neq \bot$.

By definition of $\pi^{\forall}$, there exists a valuation $\theta^4 \in D''$ such that for all persistent transitions t_i, we still have $\theta_i^4 = \theta_i - \theta_f$, and for all newly enabled (controllable and uncontrollable) transitions t_i we have $\theta_i^4 \neq \bot$. In addition, we know that for any other valuation $\theta^5 \in D'$ that equals θ^4 on all but the newly enabled uncontrollable transitions, we also have $\theta^5 \in D''$.

By construction, we have $\mathsf{tr}(\theta^4) = \mathsf{en}(m')$ (persistent plus all newly enabled transitions) and since $\theta^4 \in D'' \subseteq D'$, we have for all newly enabled transitions t_i, $\theta_i^4 \in I_s(t_i)$, and $s = (m, \theta) \xrightarrow{t_f} (m', \theta^4)$ and we have the same properties for all θ^5 as defined above. This, with $\theta^5 \in D''$ implies that $s = (m, \theta) \in \mathsf{cPred}_{C \xrightarrow{t_f} C'}(B)$. □

Proposition 3 is similar to Proposition 2 and its proof follows the same steps so we omit it.

Proposition 3 ($\mathsf{uPred}_{C \xrightarrow{t_f} C'}(B)$ computation). *Let $C = (m, D)$ and $C' = (m', D')$. Consider $B = (m', D'') \subseteq C'$, and let $\mathsf{uPred}_{C \xrightarrow{t_f} C'}(B) = (m, D_p)$. Then:*

$$D_p = D \cap \pi^{-1}_{\mathsf{en}(C)}\left(\pi^{\forall}_{\mathsf{pers}(C,t_f)}\left(\pi^{\exists}_{\substack{\mathsf{newen_u}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D'), \pi^{\exists}_{\substack{\mathsf{newen_u}(C,t_f)\\ \cup \mathsf{pers}(C,t_f)}}(D'')\right) + t_f\right)$$

3.2 Predecessor computations with DBMs

First recall that a DBM is a matrix in which coefficient $(d_{ij}, \prec)$ in row i and column j encodes a *diagonal* constraint $\theta_i - \theta_j \prec d_{ij}$, with $\prec \in \{\leq, <\}$. Variable θ_0 is assumed to always be equal to 0 so this also encodes *rectangular* constraints of the form $\theta_i \prec d_{i0}$ and $-\theta_i \prec d_{0i}$. DBMs can be put in a canonical form so that the DBM for a given set of valuations is unique [4].

The formulas we have given for $\mathsf{cPred}()$ and $\mathsf{uPred}()$ can be implemented with DBM operations. Indeed, existential projection and intersection on DBMs are classical operations and can be performed efficiently [4].

Universal projection is more complex. Most importantly, we need to complement a DBM. This can be done easily by creating, for each (non-redundant) constraint $\theta_i - \theta_j \prec d_{ij}$ of the DBM, a new DBM with only negated constraint $\theta_j - \theta_i \prec' -d_{ij}$, with $\prec'$ being strict if $\prec$ was weak and vice-versa. Then we take the union of all those DBMs. The result is therefore not a single DBM but a *finite* union of those. The rest of the operations is classical. This is kind of similar

to subtraction between DBMs that are involved in computing the controllable predecessors for timed automata [3].

The extension operator just consists in resizing the DBM and initializing the new variables so they are not constrained.

Finally, the backward in time operator is more tricky: we need to add a new variable θ'_f (the delay) and do changes of variables for all other variables θ_i as follows: $\theta'_i = \theta_i + \theta'_f$. Then we existentially project out the θ_i variables. This is actually easier than it sounds because, assuming the DBM is in canonical form, diagonal constraints $\theta_i - \theta_j \prec d_{ij}$ are left unchanged by the transformation, the θ'_f cancelling each other, while rectangular constraints $\theta_i \prec d_{i0}$ or $-\theta_i \prec d_{0i}$ just become diagonal constraints: $\theta'_i - \theta'_f \prec d_{i0}$ or $\theta'_f - \theta'_i \prec d_{0i}$ respectively.

All these operations are straightforwardly extended to finite unions of DBMs, though at a price in terms of computation cost.

3.3 Winning states

The aim of this part is to define the set Win of winning states for the controller. We will start by defining inductively Win_n for strategies in less than n steps and we then show that it admits a fixpoint that corresponds to the full set of winning states.

Definition 18. *We start by defining the following sets of states that we will need in order to construct* Win_{k+1} *using* Win_k*:*

$$\mathsf{uGood}_k(C) = \bigcup_{\substack{(C \xrightarrow{t_f} C') \in \mathcal{G}, \\ t_f \in \mathsf{en_u}(C)}} \left(\mathsf{cPred}_{C \xrightarrow{t_f} C'} (\mathsf{Win}_k \cap C') \right)$$

$$\mathsf{cGood}_k(C) = \bigcup_{\substack{(C \xrightarrow{t_f} C') \in \mathcal{G}, \\ t_f \in \mathsf{en_c}(C)}} \left(\mathsf{cPred}_{C \xrightarrow{t_f} C'} (\mathsf{Win}_k \cap C') \right)$$

$$\mathsf{uBad}_k(C) = \bigcup_{\substack{(C \xrightarrow{t_f} C') \in \mathcal{G}, \\ t_f \in \mathsf{en_u}(C)}} \left(\mathsf{uPred}_{C \xrightarrow{t_f} C'} (\overline{\mathsf{Win}_k} \cap C') \right)$$

$$\mathsf{cBad}_k(C) = \bigcup_{\substack{(C \xrightarrow{t_f} C') \in \mathcal{G}, \\ t_f \in \mathsf{en_c}(C)}} \left(\mathsf{uPred}_{C \xrightarrow{t_f} C'} (\overline{\mathsf{Win}_k} \cap C') \right)$$

Intuitively, a state is in $\mathsf{cGood}_k(C)$ if there is a controllable transition that can be fired, for which when arriving in C' we can choose a firing date for newly enabled controllable transitions such that no matter what firing date the environment chooses for its newly enabled uncontrollable transitions, we end up in Win_k. The set $\mathsf{uGood}_k(C)$ is the same except the transition that is fired is uncontrollable.

Conversely, a state is in $\mathsf{uBad}_k(C)$ if there is an uncontrollable transition that can be fired, for which when arriving in C', no matter what firing dates for newly enabled controllable transitions we choose, we cannot be sure to end up in Win_k. The set $\mathsf{cBad}_k(C)$ is the same except the transition that is fired is controllable.

From those sets of states, we can inductively define the set Win_n that contains exactly the states from which the controller has a winning strategy in at most n steps:

$$\mathsf{Win}_0 = \mathsf{Goal}$$
$$\mathsf{Win}_{k+1} = \mathsf{Win}_k \cup \bigcup_{C \in \mathcal{G}} \Big(\Big[\big(\mathsf{uGood}_k(C) \setminus \mathsf{cBad}_k(C) \big) \cup \mathsf{cGood}_k(C) \Big] \setminus \mathsf{uBad}_k(C) \Big)$$

Lemma 3. *For all state s of $\mathcal{N}$, $s \in \mathsf{Win}_n$ if and only if from s the controller has a strategy to reach Goal in at most n steps.*

Proof. Proof by induction on n the number of steps.

The base case, $n = 0$, is straightforward, so we focus on the induction step.

Suppose that for some k we have $\forall s', s' \in \mathsf{Win}_k$ if and only if the controller has a strategy from s' to reach Goal in at most k steps. Let s be a state of the TPN.

$\Rightarrow$: Let $s \in \mathsf{Win}_{k+1}$. The case $s \in \mathsf{Win}_k$ is trivially true by the induction hypothesis. Assume therefore that s is in some class C and either $s \in \mathsf{uGood}_k(C) \setminus (\mathsf{cBad}_k(C) \cup \mathsf{uBad}_k(C))$ or $s \in \mathsf{cGood}_k(C) \setminus \mathsf{uBad}_k(C)$.

- In the first case, since $s \in \mathsf{uGood}_k(C)$, then Definition 13 ensures that the controller can choose firing dates θ^c to force that all its successors by an uncontrollable transition t_u are in Win_k. We also have that $s \notin (\mathsf{cBad}_k(C) \cup \mathsf{uBad}_k(C))$, hence Definition 13 gives us that no other controllable or uncontrollable transition that could lead to $\overline{\mathsf{Win}_k}$ can be fired before one of the above-mentioned favorable uncontrollable transitions.
- In the second case, s is in $\mathsf{cGood}_k(C)$ and not in $\mathsf{uBad}_k(C)$. The same arguments as before allows us to say that the controller has a way to choose firing dates to force that all its successors by a controllable transition are in Win_k and that no unfavorable uncontrollable transition can be fired before. Note that there is no need to guard against unfavorable controllable transition firing because we are in the case where the transition choice is made by the controller.

Clearly in both cases from s the controller has a strategy to choose t_c and θ^c such that $\mathsf{Trans}(s, t_c, t_u, \theta^c, \theta^u) \in \mathsf{Win}_k, \forall t_u, \theta^u$. And so it has a strategy to reach some $s' \in \mathsf{Win}_k$ in a single step. The induction hypothesis allows us to conclude that from any state in Win_{k+1}, the controller has a strategy to reach Goal in at most $k + 1$ steps.

$\Leftarrow$: If there is a strategy to reach Goal in (strictly) less than $k + 1$ steps from a state s, then there is a strategy in at most k steps and by the induction hypothesis, $s \in \mathsf{Win}_k \subseteq \mathsf{Win}_{k+1}$.

So we focus on the case in which the controller has a strategy to reach Goal in exactly $k+1$ steps from a state s. To begin with, the controller can choose t_c and θ^c such that from all states $s' \in \mathsf{Trans}(s, t_c, t_u, \theta^c, \theta^u)$ it can still force to reach Goal in at most k steps, for all permitted choices of t_u and θ^u. These states s' are each in $C' \cap \mathsf{Win}_k$ for some class C' and not in $C'' \cap \overline{\mathsf{Win}_k}$ for any class C''. There is two main cases: either the environment lets the controller play t_c or it chooses to play some other $t_u \in T_u$.

- If t_u has been played, then the choices of firing dates in θ^c were such that whatever the environment chooses for θ^u, the successor of s by firing of t_u with these firing dates for newly enabled transitions is in Win_k. So using Definition 13 we get that $s \in \mathsf{uGood}_k(C)$. And the environment had no way to let another transition (uncontrollable or not) fire that would have led to $\overline{\mathsf{Win}_k}$ and would thus have been unfavorable to the controller. Definition 13 ensures that $s \notin (\mathsf{uBad}_k(C) \cup \mathsf{cBad}_k(C))$.
- If t_c has been played, then the environment had no way to have a disadvantageous uncontrollable transition fire first, then using Definition 13, $s \notin \mathsf{uBad}_k(C)$. And since the resulting state s' is in Win_k regardless of the environment choices, it follows from Definition 13 that $s \in \mathsf{cGood}_k(C)$.

Bringing all these sets together we have that all such states s are in:

$$\Big(\Big[\big(\mathsf{uGood}_k(C) \setminus \mathsf{cBad}_k(C)\big) \cup \mathsf{cGood}_k(C)\Big] \setminus \mathsf{uBad}_k(C)\Big)$$

This leads us to conclude that $s \in \mathsf{Win}_{k+1}$. □

Proposition 4. *For all $\mathcal{N}$ such that $\mathcal{G}$ is finite (i.e. $\mathcal{N}$ is bounded as proved in [9]), $\exists n, \mathsf{Win}_n = \mathsf{Win}_{n+l}, \forall l > 0$*

Proof. Let $b \in \mathbb{N}$ and $\mathcal{M}$ a DBM in canonical form. Let us call a b-DBM a DBM in which all finite coefficients are smaller or equal to b in absolute value. We have shown in Subection 3.2, that all $\mathsf{uPred}()$ and $\mathsf{cPred}()$ computations done on DBMs give finite unions of DBMs. And furthermore, these operations preserve b-DBMs. It is well-known for the intersection because each coefficient of the result is the minimum of the corresponding coefficients in the operands [4]. The other operations are immediate using the constructions described above.

Now, let $b_{\max}$ be the greatest of the finite coefficients in the DBMs representing the domains of all classes in the state class graph. Since that graph is finite, this maximum is well-defined. Then all those DBMs in the state class graph are $b_{\max}$-DBMs.

It follows that all $\mathsf{uGood}_k(C)$ and its three variants, which are computed from them, are finite unions of $b_{\max}$-DBMs, and so are then all the Win_k's. Clearly, there is a finite number of $b_{\max}$-DBMs because DBM coefficients are non-negative integers. By enforcing that a given union does not contain twice the same DBM, we also make sure that there are only a finite number of different finite unions of $b_{\max}$-DBMs.

Since Win_k is clearly non-decreasing with k, we can then conclude that there must be an n such that $\mathsf{Win}_{n+1} = \mathsf{Win}_n$ and, by a simple induction, that all subsequent Win_{n+l}, for $l \geq 0$, are also equal to Win_n. □

We can now define the set of winning states for the controller: $\mathsf{Win} = \mathsf{Win}_n$ for the smallest n such that we have reached a fixpoint in the construction of Win_n (i.e. $\mathsf{Win}_n = \mathsf{Win}_{n+1}$).

Using this set of winning states, the controller has a winning strategy if and only if the initial state in is Win. A strategy for the controller is then to choose firing dates and transitions to fire in order to stay in the set Win. Therefore, if the current state s is in some Win_{i+1} for $i \geq 0$, the controller will choose a successor of s that is in $\mathsf{Win}_i \setminus \mathsf{Win}_{i+1}$ in order to avoid infinite loops. As long as $s \notin \mathsf{Goal}$, it is always possible by construction of Win. To make this choice deterministic, we could assume states are ordered, e.g. in lexicographic order, and that the controller will always choose the smaller one first. Successors by a controllable transition t_c will be given priority (in this order) because the controller has to propose a transition first. Then the new valuation θ^c will be chosen depending of the transition t_u selected by the environment (t_u might be the transition t_c). The current state is the only information used to make the choices. Hence the strategy is memoryless since no information from the previous turns are needed.

4 Case studies

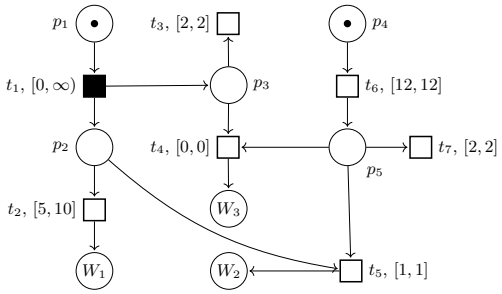
In the two following examples, for the sake of readability, we omit the immediate initialization transition and start the net directly in the initial marking.

Note that for these two examples, the classical clock-based method of [25, 15, 18], does not provide a winning strategy since the firing of an uncontrollable transition is needed to reach the goal state. A solution in this case is to add a controllable transition with firing interval $[b, b]$ in parallel of an uncontrollable transition whose firing interval is $[a, b]$.

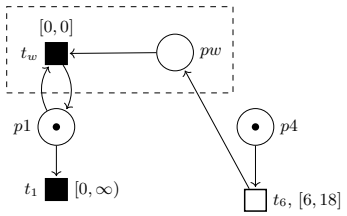
4.1 Supply chain

We consider the model of Figure 3 of two production lines starting respectively in p_1 and p_4 associated with a sorting and assembly cell. The two lines start by bringing products to $p_2 + p_3$ and p_5, respectively with transitions t_1 and t_6. The products in p_5 are either discharged through transition t_7 or assembled with the products in p_2 or p_3. The products in p_5 assembled with the products in p_3 are unloaded through transition t_4. The products in p_3 which are not assembled with p_5 are supplied through transition t_3 to another line W_3. The products in p_5 assembled with the products in p_2 are supplied through transition t_5 to another line W_2. The products in p_2 which are not assembled with p_5 are supplied to another line W_1 through transition t_2.

Transition t_2 is the only controllable transition. We wish to synthesize a controller that will enforce the products reaching places W_1, W_2 or W_3, depending on the case we consider.



3.a Petri Net model



3.b Reinitializing the firing date of t_1 when t_6 is fired

Fig. 3: Production lines

Our approach is implemented in the tool Roméo [24]. We ask for a controller to reach one of the goal states chosen successively among W_1, W_2 and W_3, and we obtain three winning strategies that consists in initializing the firing date of t_1 in the initial state. The results are as follows:

- If the goal is W_1, initialize t_1 such that: $\theta_1 \in [0,3)$ or $\theta_1 \in (10,+\infty)$
- If the goal is W_2, initialize t_1 such that: $\theta_1 \in (0,3)$
- If the goal is W_3, initialize t_1 such that: $\theta_1 \in (10,12)$ or $\theta_1 \in (12,14)$

If we extend the firing interval of t_6 to $[6,18]$ then there is no strategy for obtaining a token in W_3 with our method and our semantics because the choice of the firing date of t_1 has to be re-evaluated depending on what the environment does.

Indeed, we are looking for controllers that can be implemented with classical real-time methods and in particular with timer-triggered interrupts. We therefore need to explicitly specify which action of the environment should cause the controller to re-evaluate the firing dates of its transitions. This can be done easily with a widget that allows to disable and then re-enable a given controllable transition (here t_1) when a given transition of the environment (here t_6) is fired as shown in Figure 3.b.

We then obtain a winning strategy to reach a marking with a token in W_3 as follows:

In the initial state, initialize t_1 such that: $t_1 \in (16,+\infty)$

After the firing of t_6 (and then of t_w), initialize t_1 such that: $t_1 \in [0,2)$.

4.2 AGV

We now consider the TPN proposed in [2] that models a materials handling system with two Automated Guided Vehicle (AGV) systems and a workstation. Places p_1 and p_{10} are associated with the AGVs starting positions, and the other places in each AGV subnet correspond to the presence of the AGV in a section. Transitions t_1 and t_9 represent the start commands of the respective missions, t_7

is the start of a cycle of the workstation, and the other transitions of each AGV subnet (except t_5) correspond to the movement of the vehicle from one section to another. Finally, transition t_5 is the unloading of a part into the workstation input buffer.

Transitions t_1, t_7 and t_9 are controllable actions (commands can be activated at any time), t_6 is controllable since the speed of the AGV in this section can be set so that the time spent in the section is within the interval $[30, 40]$, while the other transitions are uncontrollable, and their static intervals are given in [2].

Places p_3 and p_{12} represent a shared zone between the two AGVs where only one vehicle at a time can stay. We then add a transition *bad* that remove the tokens in p_3 and p_{12} when two vehicles are in this zone.

The goal of the control problem proposed in [2] is to first reach a state with a marking $\{p_5, p_7, p_{10}\}$ in the time interval $[30, 65]$ and then to reach the goal state with $\{p_1, p_8, p_{13}\}$ or $\{p_6, p_9, p_{13}\}$ in the time interval $[90, 135]$. To express this goal we can use an observer as defined in [28] such that there is a token in a place WIN iff the goal is achieved within the constraints.

We then ask for a controller to reach a state with a token in the place WIN and we obtain the following winning strategy:

- In the initial state, initialize t_1, t_7 and t_9 such that: $t_1 \in [0, 5)$, $t_7 \in (45, 55)$, $t_9 \in (50, 55)$, $45 < t_7 - t_1$, $50 < t_9 - t_1$ and $0 \leq t_9 - t_7 < 5$
- when the marking is $p_1\ p_8\ p_{12}$, arriving with t_6, initialize t_1 such that: $t1 \in (10, +\infty)$ $t_8 \in [35, 55]$, $t_{11} \in [10, 40]$, $t_8 - t_1 < 35$, $t_{11} - t_1 < 0$, $-35 \leq t_{11} - t_8 < 0$

5 Conclusion

We have defined a new kind of two-player reachability timed games over the state class graph of time Petri nets. This allows to synthesize a controller that chooses, as soon as a new transition is enabled, the date on which this transition will be fired. The interest of this type of controller is that it can be implemented in real-time context with interrupts triggered by timers whose durations are fixed as soon as the associated actions are planned.

In our future work, we will study how well this semantics fits to the problem of partial observation. Moreover we plan to study the controller synthesis problem for safety and for ω-regular properties. We will also consider the question of joint timing parameters and controller synthesis.

References

1. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science **126**(2), 183–235 (1994)
2. Basile, F., Cordone, R., Piroddi, L.: Supervisory control of timed discrete-event systems with logical and temporal specifications. IEEE Transactions on Automatic Control **67**(6), 2800–2815 (2022). https://doi.org/10.1109/TAC.2021.3093618

3. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: 19th International Conference on Computer Aided Verification (CAV 2007). Lecture Notes in Computer Science, vol. 4590, pp. 121–125. Springer, Berlin, Germany (Jul 2007). https://doi.org/10.1007/978-3-540-73368-3_14
4. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools, pp. 87–124. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_3
5. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of different semantics for time petri nets. In: Peled, D.A., Tsay, Y.K. (eds.) Automated Technology for Verification and Analysis. pp. 293–307. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
6. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE trans. on soft. eng. **17**(3), 259–273 (1991)
7. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time Petri nets. In: TACAS 2003. Lecture Notes in Computer Science, vol. 2619, pp. 442–457 (2003)
8. Berthomieu, B., Dal Zilio, S., Fronc, Ł., Vernadat, F.: Time petri nets with dynamic firing dates: Semantics and applications. In: Legay, A., Bozga, M. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 85–99. Springer International Publishing, Cham (2014)
9. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time petri nets. In: Proceedings IFIP. pp. 41–46. Elsevier Science Publishers (1983)
10. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time petri nets. In: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. p. 442457. TACAS'03, Springer-Verlag, Berlin, Heidelberg (2003)
11. Boucheneb, H., Mullins, J.: Analyse des réseaux temporels : Calcul des classes en $O(n^2)$ et des temps de chemin en $O(m \times n)$. TSI. Technique et science informatiques **22**(4), 435–459 (2003)
12. Bourdil, P.A., Berthomieu, B., Dal Zilio, S., Vernadat, F.: Symmetry reduction for time petri net state classes. Science of Computer Programming **132**, 209–225 (2016)
13. Bouyer, P.: Untameable timed automata! In: Alt, H., Habib, M. (eds.) Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03). Lecture Notes in Computer Science, vol. 2607, pp. 620–631. Springer, Berlin, Germany (Feb 2003)
14. Bouyer, P., Colange, M., Markey, N.: Symbolic optimal reachability in weighted timed automata. In: CAV'16. LNCS, vol. 9779, pp. 513–530. Springer, Toronto, Canada (Jul 2016)
15. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR'05. LNCS, vol. 3653, pp. 66–80. Springer, San Fransisco, CA, USA (Aug 2005)
16. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 313–329. Springer (1998)
17. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Proc. Int. Workshop Automatic Verification Methods for Finite State Systems (CAV'89). Lecture Notes in Computer Science, vol. 407, pp. 197–212. Springer (1989)

18. 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 (Jul 2006)
19. Gardey, G., Roux, O.H., Roux, O.F.: State space computation and analysis of time Petri nets. Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems **6**(3), 301–320 (2006)
20. Heidari, P., Boucheneb, H.: Maximally permissive controller synthesis for time petri nets. International Journal of Control **86** (03 2013). https://doi.org/10.1080/00207179.2012.743038
21. Horváth, A., Paolieri, M., Ridi, L., Vicario, E.: Transient analysis of non-markovian models using stochastic state classes. Performance Evaluation **69**(7), 315–335 (2012). https://doi.org/https://doi.org/10.1016/j.peva.2011.11.002, https://www.sciencedirect.com/science/article/pii/S0166531611001520, selected papers from QEST 2010
22. Jones, N.D., Landweber, L.H., Edmund Lien, Y.: Complexity of some problems in petri nets. Theoretical Computer Science **4**(3), 277–299 (1977). https://doi.org/https://doi.org/10.1016/0304-3975(77)90014-7, https://www.sciencedirect.com/science/article/pii/0304397577900147
23. Jovanović, A., Lime, D., Roux, O.H.: Control of real-time systems with integer parameters. IEEE Transactions on Automatic Control **67**(1), 75–88 (2022). https://doi.org/10.1109/TAC.2020.3046578
24. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.M.: Romeo: A parametric model-checker for Petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009). LNCS, vol. 5505, pp. 54–57. Springer, York, United Kingdom (Mar 2009)
25. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: STACS'95. LNCS, vol. 900, pp. 229–242. Springer (1995)
26. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM Journal on Control and Optimization **25**(1), 206–230 (1987). https://doi.org/10.1137/0325013, https://doi.org/10.1137/0325013
27. Thomas, W.: On the synthesis of strategies in infinite games. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95). vol. 900, pp. 1–13. Springer (1995)
28. Toussaint, J., Simonot-Lion, F., Thomesse, J.P.: Time constraint verifications methods based on time Petri nets. In: IEEE, Future Trends in Distributed Computing Systems (FTDCS'97). pp. 262–267 (1997)