Chapter 20

A Forward On-The-Fly Approach in Controller Synthesis of Time Petri Nets

Parisa Heidari and Hanifa Boucheneb

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/47744

1. Introduction

Controller synthesis refers to finding a controller which is running in parallel with the system under study and preventing any violation from the given properties. Such a controller guarantees satisfaction of the desired properties; a controller makes an open-loop system to be closed-loop.

Controller synthesis can also be explained by game theory as a timed game with two players: environment and the controller. The strategy of the game determines the sequence of actions to be executed. In this context, the objective of controller synthesis is to find a strategy such that no matter what action is executed by the environment, the controller wins absolutely the game. Two main questions arise for the controller: the existence and possibility of implementation. The first question, *Control Problem* says given a system $\mathcal{S}$ and a property φ, does a controller C exist for the system $\mathcal{S}$ such that C running in parallel with $\mathcal{S}$ satisfies the property φ ($\mathcal{S}||C \models \varphi$). And the second one is the *Controller Synthesis Problem*; if the mentioned controller exists, is there a solution to implement it? First, a system should be modeled and then, synthesized regarding the desired property.

Among various models used to describe the behavior of $\mathcal{S}$, Timed Automata (TA in short) and Time Petri Nets (TPN in short) are the well-known. The properties studied in the TPN and TA for control purposes are classified in two main categories:

1. Safety properties: Whatever path is traveled, for all situations, a given set of forbidden states (or bad states) are never reached.
2. Reachability properties: Whatever path is traveled, for all situations, a state of a given set of states (good states) will eventually be reached.

Some research has been done to find algorithms to control these kinds of properties for timed models (TA and TPN), such as [10, 11, 20]. Two known methods in the literature are the backward *fix point* method and the backward/forward *on-the-fly* method. Both methods

are based on computing controllable predecessors of abstract states (state zones). This computation involves some expensive operations such as computing differences between abstract states (state zones).

In this chapter, we discuss an efficient approach to check whether a safety / reachability controller in time Petri nets exists or not [13]. Our approach is a completely forward on-the-fly algorithm based on the state class graph method. Unlike approaches proposed in [10, 11, 20] based on the state zone graph method, our approach does not need to compute controllable predecessors. It consists of exploring the state class graph while extracting sequences leading to undesired states and determining subclasses to be avoided. The state class graph is a suitable choice for the forward on-the-fly exploration. Using the state class graph method, the exploration algorithm converges fast and does not need any over-approximation operation to enforce the convergence.

This chapter is organized as follows: The definition of time Petri nets and its semantics as well as the state graph method come in Section 2. In Section 3, after a short survey on the control theory, previous algorithms and related work are discussed. The algorithm proposed in this chapter is developed in Section 4. Finally, Section 5 presents the conclusion and future work.

2. Time Petri nets

2.1. Definition and behavior

A time Petri net [14] is a Petri net augmented with time intervals associated with transitions. Among the different semantics proposed for time Petri nets [18], here we focus on the classical one, called intermediate semantics in [18], in the context of mono-server and strong-semantics [7].

Formally, a TPN is a tuple $(P, T, Pre, Post, M_0, Is)$ where:
- P and T are finite sets of places and transitions such that $(P \cap T = \emptyset)$,
- Pre and $Post$ are the backward and the forward incidence functions $(Pre, Post : P \times T \rightarrow \mathbb{N}, \mathbb{N}$ is the set of nonnegative integers),
- M_0 is the initial marking $(M_0 : P \rightarrow \mathbb{N})$, and
- Is is the static interval function $(Is : T \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$. $\mathbb{Q}^+$ is the set of nonnegative rational numbers. Is associates with each transition t an interval called the static firing interval of t. Bounds $\downarrow Is(t)$ and $\uparrow Is(t)$ of the interval $Is(t)$ are respectively the minimum and maximum firing delays of t.

In a controllable time Petri net, transitions are partitioned into controllable and uncontrollable transitions, denoted T_c and T_u, respectively (with $T_c \cap T_u = \emptyset$ and $T = T_c \cup T_u$). For the sake of simplicity and clarification, in this manuscript the controllable transitions are depicted as white bars, while the uncontrollable ones as black bars.

A TPN, is called bounded if for every reachable marking M, there is a bound $b \in \mathbb{N}^p$ where $M \leq b$ holds. In this condition p stands for the number of places in P.

Let M be a marking and t a transition. Transition t is enabled for M iff all required tokens for firing t are present in M, i.e., $\forall p \in P, M(p) \geq Pre(p, t)$. In this case, the firing of t leads to the

marking M' defined by: $\forall p \in P, M'(p) = M(p) - Pre(p,t) + Post(p,t)$. We denote $En(M)$ the set of transitions enabled for M:

$$En(M) = t \in T | \forall p \in P, Pre(p,t) \leq M(p). \quad (1)$$

For $t \in En(M)$, we denote $CF(M,t)$ the set of transitions enabled in M but in conflict with t:

$$CF(M,t) = t' \in En(M) | t' = t \vee \exists p \in P, M(p) < Pre(p,t') + Pre(p,t). \quad (2)$$

Let $t \in En(M)$ and M' the successor marking of M by t, a transition t' is said to be newly enabled in M' iff t' is not enabled in the intermediate marking (i.e., $M - Pre(.,t)$) or $t' = t$. We denote $New(M',t)$ the set of transitions newly enabled M', by firing t from M:

$$New(M',t) = \{t' \in En(M') | t = t' \vee \exists p \in P, M'(p) - Post(p,t) < Pre(p,t')\}. \quad (3)$$

There are two known characterizations for the TPN state. The first one, based on clocks, associates with each transition t_i of the model a *clock* to measure the time elapsed since t_i became enabled most recently. The TPN clock state is a couple (M, ν), where M is a marking and ν is a clock valuation function, $\nu : En(M) \rightarrow \mathbb{R}^+$. For a clock state (M, ν) and $t_i \in En(M)$, $\nu(t_i)$ is the value of the clock associated with transition t_i. The initial clock state is $q_0 = (M_0, \nu_0)$ where $\nu_0(t_i) = 0$, for all $t_i \in En(M_0)$. The TPN clock state evolves either by time progression or by firing transitions. When a transition t_i becomes enabled, its clock is initialized to zero. The value of this clock increases synchronously with time until t_i is fired or disabled by the firing of another transition. t_i can fire, if the value of its clock is inside its static firing interval $Is(t_i)$. It must be fired immediately, without any additional delay, when the clock reaches $\uparrow Is(t_i)$. The firing of a transition takes no time, but may lead to another marking (required tokens disappear while produced ones appear).

Let $q = (M, \nu)$ and $q_0 = (M_0, \nu_0)$ be two clock states of the TPN model, $\theta \in \mathbb{R}^+$ and $t_f \in T$. We write $q \xrightarrow{\theta} q'$, also denoted $q + \theta$, iff state q' is reachable from state q after a time progression of θ time units, i.e.:

$$\bigwedge_{t' \in En(M)} \nu(t) + \theta \leq \uparrow Id(t_i), M' = M, and \, \forall t_j \in En(M'), \nu'(t_j) = \nu(t_j) + \theta. \quad (4)$$

We write $q \xrightarrow{t_f} q'$ iff state q' is immediately reachable from state q by firing transition t_f , i.e.: $t_f \in En(M)$, $\nu(t_f) \geq \downarrow Is(t_f)$, $\forall p \in P, M'(p) = M(p) - Pre(p,t_f) + Post(p,t_f)$, and $\forall t_i \in En(M'), \nu'(t_i) = 0$, if $t_i \in New(M', t_f)$, $\nu'(t_i) = \nu(t_i)$ otherwise.

The second characterization, based on intervals, defines the TPN state as a marking and a function which associates with each enabled transition the time interval in which the transition can fire [5].

The TPN state is defined as a pair (M, Id), where M is a marking and Id is a firing interval function $(Id : En(M) \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$. The initial state is (M_0, Id_0) where M_0 is the initial marking and $Id_0(t) = Is(t)$, for $t \in En(M_0)$.

Let (M, Id) and (M', Id') be two states of the TPN model, $\theta \in \mathbb{R}^+$ and $t \in T$. The transition relation $\longrightarrow$ over states is defined as follows:

- $(M, Id) \xrightarrow{\theta} (M', Id')$, also denoted $(M, Id) + \theta$, iff from state (M, Id), we will reach the state (M', Id') by a time progression of θ units, i.e., $\bigwedge_{t' \in En(M)} \theta \leq \uparrow Id(t'), M' = M$, and $\forall t'' \in En(M'), Id'(t'') = [Max(\downarrow Id(t'') - \theta, 0), \uparrow Id(t'') - \theta]$.
- $(M, Id) \xrightarrow{t} (M', Id')$ iff the state (M', Id') is reachable from state (M, Id) by firing immediately transition t , i.e., $t \in En(M), \downarrow Id(t) = 0, \forall p \in P, M'(p) = M(p) - Pre(p, t) + Post(p, t)$, and $\forall t' \in En(M'), Id'(t') = Is(t')$, if $t' \in New(M', t), Id'(t') = Id(t')$, otherwise.

The TPN state space is the structure $(\mathcal{Q}, \longrightarrow, q_0)$, where $q_0 = (M_0, Id_0)$ is the initial state of the TPN and $\mathcal{Q} = \{q | q_0 \xrightarrow{*} q\}$ ($\xrightarrow{*}$ being the reflexive and transitive closure of the relation $\longrightarrow$ defined above) is the set of reachable states of the model.

A *run* in the TPN state space $(\mathcal{Q}, \longrightarrow, q_0)$, of a state $q \in \mathcal{Q}$, is a maximal sequence $\rho = q_1 \xrightarrow{\theta_1} q_1 + \theta_1 \xrightarrow{t_1} q_2 \xrightarrow{\theta_2} q_2 + \theta_2 \xrightarrow{t_2} q_3 ...$, such that $q_1 = q$. By convention, for any state q_i, relation $q_i \xrightarrow{0} q_i$ holds. The sequence $\theta_1 t_1 \theta_2 t_2 ...$ is called the timed trace of ρ. The sequence $t_1 t_2 ...$ is called the firing sequence (untimed trace) of ρ. A marking M is reachable iff $\exists q \in \mathcal{Q}$ s.t. its marking is M. Runs (resp. timed / untimed traces) of the TPN are all runs (resp. timed / untimed traces) of the initial state q_0.

To use enumerative analysis techniques with time Petri nets, an extra effort is required to abstract their generally infinite state spaces. Abstraction techniques aim to construct by removing some irrelevant details, a finite contraction of the state space of the model, which preserves properties of interest. For best performances, the contraction should also be the smallest possible and computed with minor resources too (time and space). The preserved properties are usually verified using standard analysis techniques on the abstractions [16].

Several state space abstraction methods have been proposed, in the literature, for time Petri nets like the *state class graph (SCG)* [4], the *zone based graph (ZBG)* [6], and etc. These abstractions may differ mainly in the characterization of states (interval states or clock states), the agglomeration criteria of states, the representation of the agglomerated states (abstract states), the kind of properties they preserve (markings, linear or branching properties) and their size.

These abstractions are finite for all bounded time Petri nets. However, if only linear properties are of interest, abstractions based on clocks are less interesting than the interval based abstractions. Indeed, abstractions based on intervals are finite for bounded TPN with unbounded intervals, while this is not true for abstraction based on clocks. The finiteness is enforced using an approximation operation, which may involve some overhead computation.

2.2. Zone Based Graph

In the Zone Based Graph (ZBG)[6], all clock states reachable by runs supporting the same firing sequence are agglomerated in the same node and considered modulo some over-approximation operation [2, 12]. This operation is used to ensure the finiteness of the ZBG for Bounded TPNs with unbounded firing intervals. An abstract state, called state zone, is defined as a pair $\beta = (M, FZ)$ combining a marking M and a formula FZ which characterizes the clock domains of all states agglomerated in the state zone. In FZ, the clock

of each enabled transition for M is represented by a variable with the same name. The domain of FZ is convex and has a unique canonical form represented by the pair (M, Z), where Z is a DBM of order $|En(M) \cup \{o\}$ defined by: $\forall (x, y) \in (En(M) \cup \{o\})^2$, $z_{xy} = Sup_{FZ}(x - y)$, where o represents the value 0. State zones of the ZBG are in relaxed form.

The initial state zone is the pair $\beta_0 = (M_0, FZ_0)$, where M_0 is the initial marking and $FZ_0 = \bigwedge_{t_i, t_j \in En(M_0)} 0 \leq ti = tj \leq \uparrow_{t_u \in En(M_0)} I_s(t_u)$.

As an example, consider the TPN given in [11] and reported at Figure 1, its state zone graph is reported at Figure 2 and its state zones are reported in Table 1.

In this document, we consider the state class method and study the possibility to enforce the behavior of a given TPN so that to satisfy a safety / reachability property. The idea is to construct on-the-fly the reachable state classes of the TPN while collecting progressively firing subintervals to be avoided so that to satisfy the properties of interest.

2.3. The state class graph method

In the state class graph method [4], all states reachable by the same firing sequence from the initial state are agglomerated in the same node and considered modulo the relation of equivalence defined by: Two sets of states are equivalent iff they have the same marking and the same firing domain. The firing domain of a set of states is the union of the firing domains of its states. All equivalent sets are agglomerated in the same node called a *state class* defined as a pair $\alpha = (M, F)$, where M is a marking and F is a formula which characterizes the firing domain of α. For each transition t_i enabled in M, there is a variable $\underline{t}_i$, in F, representing its firing delay. F can be rewritten as a set of atomic constraints of the form[1]: $\underline{t}_i - \underline{t}_j \leq c$, $\underline{t}_i \leq c$ or $-\underline{t}_j \leq c$, where t_i, t_j are transitions, $c \in \mathbb{Q} \cup \{\infty\}$ and $\mathbb{Q}$ is the set of rational numbers.

Though the same domain may be expressed by different conjunctions of atomic constraints (i.e., different formulas), all equivalent formulas have a unique form, called canonical form that is usually encoded by a difference bound matrix (DBM) [3]. The canonical form of F is encoded by the DBM D (a square matrix) of order $|En(M)| + 1$ defined by: $\forall t_i, t_j \in En(M) \cup \{t_0\}, d_{ij} = (\leq, Sup_F(\underline{t}_i - \underline{t}_j))$, where t_0 ($t_0 \notin T$) represents a fictitious transition whose delay is always equal to 0 and $Sup_F(\underline{t}_i - \underline{t}_j)$ is the largest value of $\underline{t}_i - \underline{t}_j$ in the domain of F. Its computation is based on the shortest path *Floyd-Warshall*'s algorithm and is considered as the most costly operation (cubic in the number of variables in F). The canonical form of a DBM makes easier some operations over formulas like the test of equivalence. Two formulas are equivalent iff the canonical forms of their DBMs are identical.

The initial state class is $\alpha_0 = (M_0, F_0)$, where $F_0 = \bigwedge_{t_i \in En(M_0)} \downarrow Is(t_i) \leq \underline{t}_i \leq \uparrow Is(t_i)$.

Let $\alpha = (M, F)$ be a state class and t_f a transition and $succ(\alpha, t_f)$ the set of states defined by:

$$succ(\alpha, t_f) = \{q' \in \mathcal{Q} \mid \exists q \in \alpha, \exists \theta \in \mathbb{R}^+ \text{ s.t. } q \xrightarrow{\theta} q + \theta \xrightarrow{t_f} q'\} \tag{5}$$

[1] For economy of notation, we use operator $\leq$ even if $c = \infty$.

$\beta_0 : p_1 + p_2$	$0 \leq \underline{t}_1 = \underline{t}_2 \leq 3$
$\beta_1 : p_2 + p_3$	$0 \leq \underline{t}_2 \leq 3 \wedge 0 \leq \underline{t}_3 \leq 3 \wedge 0 \leq \underline{t}_2 - \underline{t}_3 \leq 3$
$\beta_2 : p_1 + p_4$	$2 \leq \underline{t}_1 \leq 4$
$\beta_3 : p_3 + p_4$	$0 \leq \underline{t}_3 \leq 3 \wedge 0 \leq \underline{t}_4 \leq 1 \wedge 0 \leq \underline{t}_3 - \underline{t}_4 \leq 3$
$\beta_4 : p_2$	$2 \leq \underline{t}_2 \leq 3$
$\beta_5 : p_3 + p_4$	$0 \leq \underline{t}_3 = \underline{t}_4 \leq 2$
$\beta_6 : p_4$	

Table 1. State zones of the TPN presented at Figure 2

The state class α has a successor by t_f (i.e. $succ(\alpha, t_f) \neq \emptyset$), iff t_f is enabled in M and can be fired before any other enabled transition, i.e., the following formula is consistent[2]: $F \wedge (\bigwedge_{t_i \in En(M)} \underline{t}_f \leq \underline{t}_i)$. In this case, the firing of t_f leads to the state class $\alpha' = (M', F') = succ(\alpha, t_f)$ computed as follows [4]:

1. $\forall p \in P, M'(p) = M(p) - Pre(p, t_f) + Post(p, t_f)$.
2. $F' = F \wedge (\bigwedge_{t_i \in En(M)} \underline{t}_f - \underline{t}_i \leq 0)$
3. Replace in F' each $\underline{t}_i \neq \underline{t}_f$, by $(\underline{t}_i + \underline{t}_f)$.
4. Eliminate by substitution $\underline{t}_f$ and each $\underline{t}_i$ of transition conflicting with t_f in M.
5. Add constraint $\downarrow Is(t_n) \leq \underline{t}_n \leq \uparrow Is(t_n)$, for each transition $t_n \in New(M', t_f)$.

Formally, the SCG of a TPN model is a structure $(\mathcal{CC}, \longrightarrow, \alpha_0)$, where $\alpha_0 = (M_0, F_0)$ is the initial state class, $\forall t_i \in T, \alpha \xrightarrow{t_i} \alpha'$ iff $\alpha' = succ(\alpha, t_i) \neq \emptyset$ and $\mathcal{CC} = \{\alpha | \alpha_0 \xrightarrow{*} \alpha\}$.

The SCG is finite for all bounded TPNs and preserves linear properties [5]. As an example, Figure 2 shows the state class graph of the TPN presented at Figure 1. Its state classes are reported in Table 2. For this example, state class graph and state zone based graph of the system are identical while classes and zones are different.

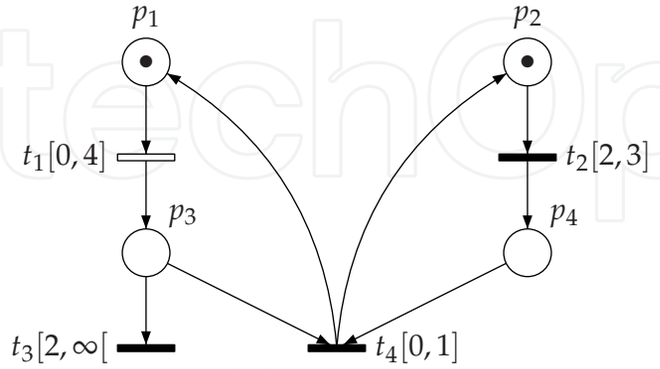

Figure 1. A simple Petri net with $T_c = \{t_1\}$

[2] A formula F is consistent iff there is, at least, one tuple of values that satisfies, at once, all constraints of F.

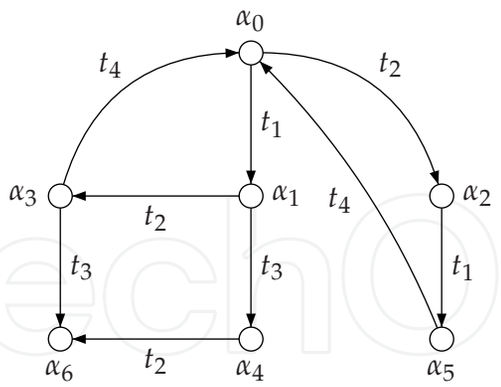

Figure 2. The State Graph of the TPN presented at Figure 1

$\alpha_0 : p_1 + p_2$	$0 \leq \underline{t}_1 \leq 4 \wedge 2 \leq \underline{t}_2 \leq 3$
$\alpha_1 : p_2 + p_3$	$0 \leq \underline{t}_2 \leq 3 \wedge 2 \leq \underline{t}_3$
$\alpha_2 : p_1 + p_4$	$0 \leq \underline{t}_1 \leq 2$
$\alpha_3 : p_3 + p_4$	$0 \leq \underline{t}_3 \wedge 0 \leq \underline{t}_4 \leq 1$
$\alpha_4 : p_2$	$0 \leq \underline{t}_2 \leq 1$
$\alpha_5 : p_3 + p_4$	$2 \leq \underline{t}_3 < \infty \wedge 0 \leq \underline{t}_4 \leq 1$
$\alpha_6 : p_4$	

Table 2. The state classes of the TPN presented at Figure 2

2.4. A forward method for computing predecessors of state classes

Let $\alpha = (M, F)$ be a state class and $\omega \in T^+$ a sequence of transitions firable from α. We denote $succ(\alpha, \omega)$ the state class reachable from α by firing successively transitions of ω. We define inductively this set as follows: $succ(\alpha, \omega) = \alpha$, if $\omega = \epsilon$ and $succ(\alpha, \omega) = succ(succ(\alpha, \omega'), t_i)$, if $\omega = \omega'.t_i$.

During the firing of a sequence of transitions ω from α, the same transition may be newly enabled several times. To distinguish between different enabling of the same transition t_i, we denote t_i^k for $k > 0$ the transition t_i (newly) enabled by the k^{th} transition of the sequence; t_i^0 denotes the transition t_i enabled in M. Let $\omega = t_1^{k_1}....t_m^{k_m} \in T^+$ with $m > 0$ be a sequence of transitions firable from α (i.e., $succ(\alpha, \omega) \neq \emptyset$). We denote $Fire(\alpha, \omega)$ the largest subclass α' of α (i.e., $\alpha' \subseteq \alpha$) s.t. ω is firable from all its states, i.e.,

$$Fire(\alpha, \omega) = \{q_1 \in \alpha \mid \exists \theta_1, ..., \theta_m, q_1 \xrightarrow{\theta_1} q_1 + \theta_1 \xrightarrow{t_1^{k_1}} q_2...q_m + \theta_m \xrightarrow{t_m^{k_m}} q_{m+1}\} \quad (6)$$

Proposition 1. *$Fire(\alpha, \omega)$ is the state class (M', F') where $M' = M$ and F' can be computed as follows[3]: Let $M_1 = M$ and M_{f+1}, for $f \in [1, m]$, be the marking reached from M by the subsequence $t_1^{k_1}....t_f^{k_f}$ of ω.*

1. *Initialize F' with the formula obtained from F by renaming all variables $\underline{t}_i$ in $\underline{t}_i^0$.*

[3] We suppose that the truth value of an empty set of constraints is always *true*.

2. *Add the following constraints:*

$$\begin{aligned}
&\bigwedge_{f\in[1,m]}\Big(\bigwedge_{t_i\in(En(M_1)-\bigcup_{j\in[1,f[}CF(M_j,t_j))}\underline{t}_f^{k_f}-\underline{t}_i^0\leq 0\wedge\\
&\bigwedge_{j\in[1,f[,t_n\in(New(M_{j+1},t_j)-\bigcup_{k\in]j,f[}CF(M_k,t_k))}\underline{t}_f^{k_f}-\underline{t}_n^j\leq 0\wedge\\
&\bigwedge_{t_n\in New(M_{f+1},t_f)}\downarrow Is(t_n)\leq \underline{t}_n^f-\underline{t}_f^{k_f}\leq\uparrow Is(t_n))
\end{aligned}\tag{7}$$

3. *Put the resulting formula in canonical form and eliminate all variables $\underline{t}_i^j$ such that $j>0$, rename all variables $\underline{t}_i^0$ in $\underline{t}_i$.*

Note that $Fire(\alpha,\omega)\neq\emptyset$ (i.e., ω is firable from α) iff ω is feasible in the underlying untimed model and the formula obtained at step 2) above is consistent.

Proof. By step 1) all variables associated with transitions of $En(M)$ are renamed ($\underline{t}_i$ is renamed in $\underline{t}_i^0$). This step allows us to distinguish between delays of transitions enabled in M from those that are newly enabled by the transitions of the firing sequence.

Step 2) adds the firing constraints of transitions of the sequence (for $f\in[1,m]$). For each transition $t_f^{k_f}$ of the sequence, three blocks of constraints are added. The two first blocks mean that the delay of $t_f^{k_f}$ must be less or equal to the delays of all transitions enabled in M_f (i.e., transitions of $En(M)$ and those enabled by t_j ($New(M_{j+1},t_j), 1\leq j<f$) that are maintained continuously enabled at least until firing $t_f^{k_f}$). Transitions of $En(M)$ that are maintained continuously enabled at least until firing $t_f^{k_f}$ are transitions of $En(M)$ which are not in conflict with $t_1^{k_1}$ in M_1, and, ..., and not in conflict with $t_{f-1}^{k_{f-1}}$ in M_{f-1}. Similarly, transitions of $New(M_j,t_j)$ (with $1\leq j<f$) that are maintained continuously enabled at least until firing $t_f^{k_f}$ are transitions of $New(M_{j+1},t_j)$ which are not in conflict with $t_{j+1}^{k_{j+1}}$ in M_{j+1}, and, ..., and not in conflict with $t_{f-1}^{k_{f-1}}$ in M_{f-1}. The third block of constraints specifies the firing delays of transitions that are newly enabled by $t_f^{k_f}$.

Step 3) isolates the largest subclass of α such that ω is firable from all its states. □

As an example, consider the TPN depicted at Figure 1 and its state class graph shown at Figure 2. Let us show how to compute $Fire(\alpha_0,t_1^0t_2^0t_3^1)$. We have $En(M_0)=\{t_1,t_2\}$, $CF(M_0,t_1)=\{t_1\}$, $CF(M_1,t_2)=\{t_2\}$, $New(M_0,t_1)=\{t_3\}$ and $New(M_1,t_2)=\{t_4\}$. The subclass $(p_1+p_2,F')=Fire(\alpha_0,t_1^0t_2^0t_3^1)$ is computed as follows:

1. Initialize F' with the formula obtained from $0\leq\underline{t}_1\leq 4\wedge 2\leq\underline{t}_2\leq 3$ by renaming all variables $\underline{t}_i$ in $\underline{t}_i^0$: $0\leq\underline{t}_1^0\leq 4\wedge 2\leq\underline{t}_2^0\leq 3$

2. Add the firing constraints of t_1 before t_2, t_2 before t_3 and constraints on the firing intervals of transitions enabled by these firings (i.e., t_3 and t_4):

$$\underline{t}_1^0 - \underline{t}_2^0 \leq 0 \wedge \underline{t}_2^0 - \underline{t}_3^1 \leq 0 \ \wedge \ 2 \leq \underline{t}_3^1 - \underline{t}_1^0 \ \wedge \ 0 \leq \underline{t}_4^2 - \underline{t}_2^0 \leq 1$$

3. Put the resulting formula in canonical form and eliminate all variables $\underline{t}_i^j$ such that $j > 0$, rename all variables $\underline{t}_i^0$ in $\underline{t}_i$: $0 \leq \underline{t}_1 \leq 2 \wedge 2 \leq \underline{t}_2 \leq 3 \wedge 1 \leq \underline{t}_2 - \underline{t}_1 \leq 3$.

The subclass $Fire(\alpha_0, t_1^0 t_2^0 t_3^1)$ consists of all states of α_0 from which the sequence $t_1 t_2 t_3$ is firable. If t_1 is controllable, to avoid reaching the marking p_4 by the sequence $t_1 t_2 t_3$, it suffices to choose the firing interval of t_1 in α_0 outside its firing interval in $Fire(\alpha_0, t_1^0 t_2^0 t_3^1)$ (i.e., $]2, 4]$).

Note that this forward method of computing predecessors can also be adapted and applied to the clock based abstractions. For instance, using the zone based graph, the initial state zone of the TPN shown at Figure 1 is $\beta_0 = (p_1 + p_2, 0 \leq \underline{t}_1 = \underline{t}_2 \leq 3)$. The sub-zone β_0' of β_0, from which the sequence $t_1 t_2 t_3$ is firable, can be computed in a similar way as the previous procedure where delay constraints are replaced by clock constraints:

$\beta_0' = (p_1 + p_2, 0 \leq \underline{t}_1 = \underline{t}_2 \leq 2)$. To avoid reaching by the sequence $t_1 t_2 t_3$ the marking p_4, it suffices to delay the firing of t_1 until when its clocks overpasses 2, which means that its firing interval should be $]2, 4]$.

3. Related work

The theory of control was initially introduced by Ramadge and Wonham in [17]. They have formalized, in terms of formal languages, the notion of control and the existence of a controller that forces a discrete event system (DES) to behave as expected. The concept of control has been afterwards extended to various models such as timed automata [21] and time Petri nets [19], where the control specification is expressed on the model states rather than the model language. Thus, for every system modeled by a controllable language, timed automata or time Petri nets, controller synthesis is used to restrict the behavior of the system making it to satisfy the desired safety or reachability properties. The typical procedure is: a system is modeled, the desired properties are defined, then, the existence and the implementation of the appropriate controller (control problem and controller synthesis problem respectively [1]) are investigated.

Several approaches of controller synthesis have been proposed in the literature. They may differ in the model they are working on (various types of Petri nets or automata), the approach they are based on (analytical as in [22], structural as in [9], semantic as in [10, 11, 20]), and finally the property to be controlled.

In [22], the authors have considered a particular type of capacity timed Petri net, where timing constraints are associated with transitions and some places, and all transitions are controllable. This timed Petri net is used to model a cluster tool with wafer residency time constraints. The wafers and their time constraints are represented by timed places. Using analytical approaches of schedulability and the particular structure of their model (model of the cluster tool), the authors have established an algorithm for finding, if it exists, an optimal periodic schedule which respects residency time constraints of wafers. The control consists

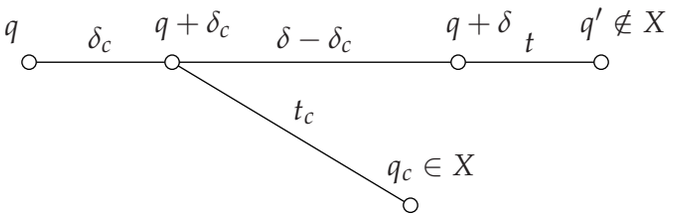

Figure 3. Controllable Predecessors

of limiting timing constraints of transitions and some places so as to respect residency time constraints of wafers.

In [8, 9], the authors have considered safe and live time Petri nets where deadlines can be associated with some transition firings. The control consists of enforcing the model to meet deadlines of transition firings. The controller has the possibility to disable any transition t which prevents to meet the deadline of a transition t_d. A transition t is allowed to fire only if its latency (the maximum delay between firing t and the next firing of t_d) is not greater than the current deadline of t_d. The latencies of transitions are computed by constructing an unfolding Petri net of the underlying untimed Petri net. This approach does not need to explore the state space. However, in general, the resulting controller is not maximally permissive (i.e. meaning that the controller may disable a net behavior that does not violate the properties of interest).

In [10, 11, 20], the authors have considered timed models (TA or TPN) with two kinds of transitions (controllable and uncontrollable) and investigated the control problem for safety or reachability properties. To prevent some undesired states, the controller can act on any firable and controllable transition by delaying or forcing its firing but it cannot disable transitions. The control problem is addressed by computing the winning states of the model, i.e. states which will not lead, by an uncontrollable transition, to an undesired state. The computation of the winning states is based on the concept of controllable predecessors of states. In the literature, the set of controllable predecessors is usually denoted by $\pi(X)$, where X is a set of states satisfying the desired property (safe/goal states). The set $\pi(X)$ is defined by [11]:

$$\begin{aligned}
&\pi(X) = \{q \in \mathcal{Q} | ((\exists\, \delta \in \mathbb{R}_{\geq 0}, t \in T\ q' \in X\ q \xrightarrow{\delta} \xrightarrow{t} q') \vee \\
&(\exists\, \delta \in \mathbb{R}_{\geq 0}, q' \in X\ q \xrightarrow{\delta} q')) \wedge \\
&\forall\, \delta \in \mathbb{R}_{\geq 0}\ if\ \exists\, t \in T, q' \notin X\ q \xrightarrow{\delta} \xrightarrow{t} q' \\
&then\ \exists\, \delta_c < \delta, t_c \in T_c,\ q_c \in X\ q \xrightarrow{\delta_c} \xrightarrow{t_c} q_c\}
\end{aligned} \tag{8}$$

Intuitively, $\pi(X)$ is the set of predecessors of X which will not bring the system out of X. Figure 3 clarifies this concept. If the environment can execute an uncontrollable transition after δ time units, leading the system out of X (denoted by $\bar{X}$), then the controller should be able to execute a controllable action to keep the system in X before δ time units. In addition, in the context of timed models with strong semantics (a transition must be fired, without any additional delay, when the upper bound of its firing interval is reached), the controller should not be forced to execute a controllable transition leading the system out of X.

Let $AG\ \phi$ be a safety property and $X_0 = Sat(\phi)$ the set of states which satisfy the property ϕ (safe states). The fix point of $X_{i+1} = h(X_i) = X_i \cap \pi(X_i), i \geq 0$ gives the largest set of

safe states whose behaviors can be controlled so as to maintain the system inside this set of states (i.e., winning states). If the largest fix point of h includes the initial state then, it gives a controller which forces the system to stay in safe states (i.e., a winning strategy).

Similarly, the fix point method is also used for reachability properties. Let $AF\ \psi$ be a reachability property and $X_0 = Sat(\psi)$ the set of goal states. The least fix point of $X_{i+1} = h(X_i) = X_i \cup \pi(X_i), i \geq 0$ is the set of states whose behaviors can be controlled so as to reach one of the goal states (i.e., winning states) [10, 20].

In the context of a timed model, this technique is applied on a state space abstraction of the timed model. In this case, X_i is a set of abstract states. If X_i is a finite set of abstract states, then the controllable predecessors of X_i is also a finite set of abstract states. The computation of the fix point of h will converge after a finite number of steps if the state space abstraction is finite [10, 11, 20].

Note that the state space abstractions used in [10, 11, 20] are based on clocks but the state space abstraction used in [11] is not necessarily complete. The fix point method cannot guarantee to give the safety controller when it exists, unless the state space abstraction is both sound and complete. A state space abstraction of a given model is sound and complete iff it captures all firing sequences of the model and each firing sequence in the state space abstraction reflects a firing sequence of the model. Indeed, a synthesis may fail because of some unreachable states, while for the reachable state space the safety controller exists. However, the cost of processing is increased as a sound and complete state space abstraction should be entirely calculated before applying the fix point algorithm.

Let us explain by means of an example how to compute the fix point of h for a safety property. Consider the TPN given in [11] and reported in Figure 1. The state class graph (SCG) and the zone based graph (ZBG) of this TPN are equal, except that nodes are defined differently (state classes or state zones). The state class graph is depicted in Figure 2. Its state zones and state classes are reported in Table 1 and Table 2, respectively.

Consider the state zone graph and suppose that we are interested to force the following safety property: $AG\ \Sigma_{i=1}^{4}\ p_i = 2$ which means that the number of tokens in the TPN is always 2. The transition t_1 is the only controllable transition and the forbidden markings is determined by $\Sigma_{i=1}^{4}\ p_i \neq 2$. As the state class graph shows, if t_2 happens before t_1 the right path happens which is safe and the controller has nothing to do. On the other hand, if t_1 happens before t_2, two state classes having forbidden markings may be reached (α_4, α_6).

To verify whether or not there is a controller for such a property, we compute the fix point of $X_{i+1} = h(X_i) = X_i \cap \pi(X_i)$, where $X_0 = \{\beta_0, \beta_1, \beta_2, \beta_3, \beta_5\}$ is the set of state zones which satisfy the property *not* $p_1 + p_3 = 0$. Such a controller exists iff the initial state of the model is a winning state (i.e., belongs to the fix point of h). The fix point is computed, in 3 iterations, as follows:

1) Iteration 1: $X_1 = X_0 \cap \pi(X_0) = \{\beta_0, \beta_1', \beta_2, \beta_3', \beta_5\}$. In this iteration, all states of β_1 and β_3, which are uncontrollable predecessors of bad state classes β_4 and β_6 are eliminated:

$\beta_1' = (p_2 + p_3, 1 < \underline{t}_2 \leq 3 \wedge 0 \leq \underline{t}_3 < 2 \wedge 1 < \underline{t}_2 - \underline{t}_3 \leq 3)$ and

$\beta_3' = (p_3 + p_4, 1 \leq \underline{t}_3 <\leq 3 \wedge 0 < t_4 \leq 1 \wedge 1 \leq \underline{t}_3 - \underline{t}_4 \leq 3)$.

2) Iteration 2: $X_2 = X_1 \cap \pi(X_1) = \{\beta_0, \beta_1'', \beta_2, \beta_3', \beta_5\}$. This iteration eliminates from β_1' all states, which are uncontrollable predecessors of bad states of $\beta_3 - \beta_3'$:

$\beta_1'' = \{p_2 + p_3, 2 < \underline{t}_2 \leq 3 \wedge 0 \leq \underline{t}_3 < 1 \wedge 2 \leq \underline{t}_2 - \underline{t}_3 \leq 3\}$.

3) Iteration 3: $X_2 = X_2 \cap \pi(X_2) = \{\beta_0, \beta_1'', \beta_2, \beta_3', \beta_5\}$. The fix point X_2 is then the set of winning states. Since the initial state zone belongs to X_2, there is a controller for forcing the property AG *not* $p_1 + p_3 = 0$. To keep the model in safe states (in states of X_2), the controller must delay, in β_0, the firing of t_1 until its clock overpasses the value 2. Doing so, the successor of β_0 by t_1 will be β_1''.

This approach needs however to construct a state space abstraction before computing the set of winning states. To overcome this limitation, in [10, 20], the authors have investigated the use of on-the-fly algorithms besides the fix point to compute the winning states for timed game automata (timed automata with controllable and uncontrollable transitions). We report, in Fig 4, the on-the-fly algorithm given in [10] for the case of reachability properties and timed game automata. This algorithm uses three lists *Passed* containing all state zones explored so far, *Waiting*, containing the set of edges to be processed and *Depend* indicating, for each state zone S, the set of edges to be reevaluated in case the set of the winning states in S ($Win[S]$) is updated. Using this on-the-fly method, in each step, a part of the state zone graph is constructed and an edge $e = (S, a, S')$ of the *Waiting* list is processed. If the state zone S' is not in *Passed* and there is, in S', some states which satisfy the desired reachability property, then these states are added to the winning states of S' ($Win[S']$). The winning states of S will be recomputed later (the edge e is added to the *Waiting* list). If S' is in *Passed*, the set of the winning states of S ($Win[S]$) is recomputed and possibly those of its predecessors and so on. The set $Win[S]$ is the largest subset of S which is included in the controllable predecessors of the winning states of all its successors.

This on-the-fly algorithm, based on computing controllable predecessors, requires some expensive operations such as the difference between abstract states (state zones). The difference between two state zones is not necessarily a state zone and then may result in several state zones which need to be handled separately.

In this chapter, we propose another on-the-fly approach which does not need this expensive operation. Our approach differs from the previous ones by the fact it computes bad states (i.e.: states which may lead to an undesired state) instead of computing the winning states and it constructs a state class graph instead of a state zone graph. In addition, the bad states are computed, using a forward approach, for only state classes containing at least a controllable transition.

4. An on-the-fly algorithm for investigating the existence of a controller for a TPN

This chapter aims to propose an efficient forward on-the-fly method based on the state class graph for checking the existence of a safety/reachability controller for a TPN. As discussed earlier, the state class graph is a good alternative for the on-the-fly algorithms as the exploration converges fast and does not need any over-approximation operation to enforce the convergence. The method, proposed here, is completely a forward and does not compute

Initialization:

$Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, \vec{0})\}^{\nearrow}$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = \mathsf{Post}_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge (s_0 \notin Win[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow pop(Waiting)$;
  if $S' \notin Passed$ **then**
    $Passed \leftarrow Passed \cup \{S'\}$;
    $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
    $Win[S'] \leftarrow S' \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    $Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = \mathsf{Post}_\alpha(S')^{\nearrow}\}$;
    if $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
  else (* **reevaluate** *)[a]
    $Win^* \leftarrow \mathsf{Pred}_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} \mathsf{Pred}_c(Win[T]),$
      $\bigcup_{S \xrightarrow{u} T} \mathsf{Pred}_u(T \setminus Win[T])) \cap S$;
    if $(Win[S] \subsetneq Win^*)$ **then**
      $Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
    $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
  endif
endwhile

[a] When $T \notin Passed$,$Win[T] = \emptyset$

Figure 4. On-the-fly algorithm for timed game automata proposed in [10]

controllable predecessors (which is considered as an expensive operation). To explain the method, we start with safety properties.

Let us introduce informally the principle of our approach by means of the previous example. Consider the TPN shown in Figure 1, its state class graph depicted in Figure 2 and its state classes reported in Table 2. Our goal is to avoid to reach bad states (i.e., state classes α_4 and α_6) by choosing appropriately the firing intervals for controllable transitions.

From the initial state class α_0, there are two elementary paths $\alpha_0 t_1 \alpha_1 t_3 \alpha_4$ and $\alpha_0 t_1 \alpha_1 t_2 \alpha_3 t_3 \alpha_6$ that lead to bad states. In both paths, there is only one state class (α_0) where the controllable transition t_1 is firable. To avoid these bad paths, we propose to compute all states of α_0 from which $t_1 t_3$ or $t_1 t_2 t_3$ is firable, i.e., $B(\alpha_0) = Fire(\alpha_0, t_1 t_3) \cup Fire(\alpha_0, t_1 t_2 t_3)$, where:

$Fire(\alpha_0, t_1 t_3) = (p_1 + p_2, 0 \leq \underline{t}_1 \leq 1 \wedge 2 \leq \underline{t}_2 \leq 3 \wedge 2 \leq \underline{t}_2 - \underline{t}_1 \leq 3)$ and

$Fire(\alpha_0, t_1 t_2 t_3) = (p_1 + p_2, 0 \leq \underline{t}_1 \leq 2 \wedge 2 \leq \underline{t}_2 \leq 3 \wedge 1 \leq \underline{t}_2 - \underline{t}_1 \leq 3)$.

To avoid these bad states, it suffices to replace in α_0, the firing interval of t_1 with $]2, 4]$. This interval is the complement of $[0, 2] \cup [0, 1]$ in the firing interval of t_1 in α_0 ($[0, 4]$).
The approach we propose in the following section, is a combination of this principle with a forward on-the-fly method.

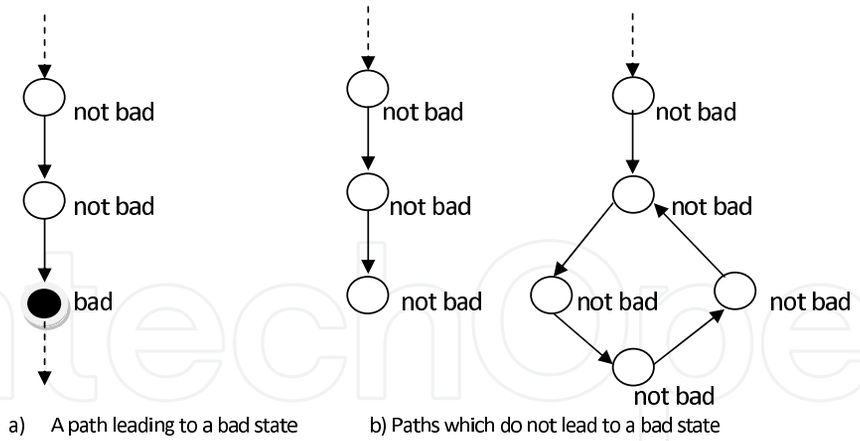

Figure 5. Path satisfying or not a safety property. Black states should be avoided.

4.1. Controller for safety properties

A controller for safety properties running in parallel with the system should satisfy the property '$AG\ not\ bad$' where 'bad' stands for the set of states having a forbidden marking and it means that 'bad' states will never happen. We introduce here an algorithm to re-constrain the controllable transitions and reach a safe net.

The idea is to construct, using a forward on-the-fly method, the state class graph of the TPN to determine whether controllable transitions have to be constrained, in order to avoid forbidden markings. This method computes and explores, path by path, the state class graph of a TPN looking for the sequences leading the system to any forbidden marking (bad sequences or bad paths). And using Proposition 1, we get the subclasses causing the bad states happening later through the found sequences (bad subclasses). We restrict the domain of controllable transitions in the state class where they were enabled so as to avoid its bad subclasses. The restriction of the interval of a controllable transition t of a state class α is obtained by subtracting from its interval in α ($INT(\alpha, t)$), intervals of t in its bad subclasses.

Before describing the procedure formally, we define an auxiliary operation over intervals to be used in the algorithm. Let I and I' be two nonempty (real) intervals. We denote $I \oplus I'$ intervals defined by:

$$\forall a \in \mathbb{R}, a \in I \oplus I' iff \exists b \in I, \exists c \in I', a = b + c. \tag{9}$$

As an example, for $I = [1,4]$ and $I' =]2,5]$, $I \oplus I' =]3,9]$. And also

$$LI(\alpha) = \{(t_c, t_s, BI) | t_c \in En_c(M), t_s \in En_c^0(M); BI = \bigcup_{\omega \in \Omega(\alpha)} INT(Fire(\alpha,\omega), \underline{t_c} - \underline{t_s}) \neq INT(\alpha, \underline{t_c} - \underline{t_s})\} \tag{10}$$

This method is presented in the algorithms 1 and 2. The symbol T_c refers to the set of controllable transitions and all forbidden markings of the net are saved in a set called, *bad*. The list *Passed* is used to retrieve the set of state classes processed so far, their bad sequences, and the bad intervals of controllable transitions (their domains in bad subclasses). Function *main* consists of an initialization step and a calling to the recursive function *explore*. The call $explore(\alpha_0, \emptyset, \{\alpha_0\})$ returns the set of bad sequences that cannot be avoided, from α_0, by

restricting firing domains of controllable transitions. If this set is nonempty, it means that such a controller does not exist. Otherwise, it exists and the algorithm guarantees that for each state class α with some bad sequences, there is a possibility to choose appropriately the firing intervals of some controllable transitions so as to avoid all bad subclasses of α. The control of α consists of eliminating, from the firing intervals of such controllable transitions, all parts figuring in its bad subclasses. The restriction of domains is also applied on firing delays between two controllable transitions of α. We get in $Ctrl$, all possibilities of controlling each state class. In case there is only one controllable transition in α, its delay with a fictitious transition whose time variable is fixed at 0 is considered.

Each element of *Passed* is a triplet $(\alpha, \Omega(\alpha), LI(\alpha))$ where $\alpha = (M, F)$ is a state class s.t. $M \notin bad$, $\Omega(\alpha)$ is the set of bad sequences of α, which cannot be avoided, independently of α, from its successors, and $LI(\alpha)$ gives the intervals of controllable transitions in bad subclasses of α (bad intervals). The set $LI(\alpha)$ allows to retrieve the safe intervals of controllable transitions, by computing the complements, in α, of the forbidden intervals (i.e., all possibilities of controlling α, $Ctrl(\alpha)$).

The function *explore* receives parameters α being the class under process, t the transition leading to α and $\mathcal{C}$ the set of traveled classes in the current path. It uses functions $succ(\alpha, t)$ and $Fire(\alpha, \omega)$ already explained by equations 5 and 6 (in sections 2.3 and 2.4, respectively). It distinguishes 3 cases:

1) α has been already processed (i.e., α is in *Passed*): In this case, there is no need to explore it again. However, its bad sequences have to be propagated to its predecessor by t, in case the control needs to be started before reaching α in order to avoid bad states of its predecessors. The control of α is independent of its predecessors along the path if all possibilities of control in α are limited to the newly enabled transitions. In case there is, in α, a possibility of control, which limits the firing interval of some controllable transition not newly enabled in α, it means that the predecessor of α by t has some bad states that must be avoided. The condition $Dep(\alpha, t, LI)$, used in Algorithm 2, is to control α independently of its predecessor by t.

2) α has a forbidden marking (i.e., α is a bad state class): In this case, the transition t is returned, which means that this sequence needs to be avoided before reaching α.

3) In other cases, the function *explore* is called for each successor of α, not already encountered in the current path (see Figure 5), to collect, in Ω, the bad sequences of its successors. Once all successors are processed, Ω is checked:

3.1) If $\Omega = \emptyset$, it means that α does not lead to any bad state class or its bad sequences can be avoided later by controlling its successors, then $(\alpha, \emptyset, \emptyset)$ is added to *Passed* and the function returns with $\emptyset$.

3.2) If $\Omega \neq \emptyset$, the function *explore* determines intervals of controllable transitions in bad subclasses, which do not cover their intervals in α. It gets such intervals, identifying states to be avoided, in LI (bad intervals). It adds (α, Ω, LI) to *Passed* and then verifies whether or not α is controllable independently of its predecessor state class in the current path. In such a case, there is no need to start the control before reaching α and then the empty set is returned by

the function. Otherwise, it is needed to propagate the control to its predecessor by t. The set of sequences, obtained by prefixing with t sequences of Ω, is then returned by the function.

This algorithm tries to control the system behavior starting from the last to the first state classes of bad paths. If it fails to control a state class of a path, so as to avoid all bad state classes, the algorithm tries to control its previous state classes. If it succeeds to control a state class, there is no need to control its predecessors. The aim is to limit as little as possible the behavior of the system (more permissive controller).

Algorithm 1 On-the-fly algorithm for safety control problem of TPN- Part A

Function $main(TPN\ \mathcal{N},\ Markings\ bad)$
Where $\mathcal{N}$ **is a TPN**
bad **is a set of bad markings.**
Let T_c **be the set of controllable transitions of** $\mathcal{N}$ **and**
α_0 **the initial state class of** $\mathcal{N}$**.**
$Passed = \emptyset$
if $(explore(\alpha_0, \epsilon, \{\alpha_0\}) \neq \emptyset)$ **then**
  {Controller does not exist}
  return
end if
for all $((\alpha, \Omega, LI) \in Passed)$ **do**
  $Ctrl[\alpha] = \bigcup_{(t_c, t_s, BI) \in LI} \{(t_c, t_s, INT(\alpha, \underline{t}_c - \underline{t}_s) - BI)\}$
end for

$(^*)$
$\alpha = (M, F)$;
$En_c(M) = En(M) \cap T_c$;
$En_c^0(M) = En_c(M) \cup \{t_0\}$;
$New_c(M,t) = New(M,t) \cap T_c$;
$New(M_0, \epsilon) = En(M_0)$;
t_0 is a fictitious transition whose time variable is fixed at 0.
$Dep(\alpha, t, LI) \equiv$
$\exists (t_c, t_s, BI) \in LI, t_c \notin New(M,t) \wedge (t_s \notin New(M,t) \vee$
$INT(\alpha, \underline{t}_c - \underline{t}_0) \not\subseteq \bigcap_{I \in BI} (I \oplus INT(\alpha, \underline{t}_s - \underline{t}_0)))$

4.2. Example

To explain the procedure, we trace the algorithm on the TPN shown in Figure 1. Its SCG and its state classes are reported in Figure 2 and Table 2, respectively. For this example, we have $T_c = \{t_1\}$, $bad = \{p_2, p_4\}$, $Passed = \emptyset$ and $\alpha_0 = (p_1 + p_2, 0 \leq \underline{t}_1 \leq 4 \wedge 2 \leq \underline{t}_2 \leq 3)$.

The process starts by calling $explore(\alpha_0, \epsilon, \{\alpha_0\})$ (see Figure 6). Since α_0 is not in *Passed* and its marking is not forbidden, *explore* is successively called for the successors of α_0: $explore(\alpha_1, t_1, \{\alpha_0, \alpha_1\})$ and $explore(\alpha_2, t_2, \{\alpha_0, \alpha_2\})$. In *explore* of α_1, function *explore* is successively called for α_3 and α_4. In *explore* of α_3, function *explore* is called for the successor α_6 of α_3 by t_3: $explore(\alpha_6, t_3, \{\alpha_0, \alpha_1, \alpha_3, \alpha_6\})$. For the successor of α_3 by t_4 (i.e., α_0), there is no need to call *explore* as it belongs to the current path. Since α_6 has a forbidden marking, *explore* of α_6 returns to *explore* of α_3 with $\{t_3\}$, which, in turn, adds $(\alpha_3, \{t_2 t_3\}, \emptyset)$ to *Passed* and returns to *explore* of α_1 with $\{t_2 t_3\}$.

Algorithm 2 On-the-fly algorithm for safety control problem of TPN- Part B

Function *Traces explore*$(Class\ \alpha, Trans\ t,\ Classes\ \mathcal{C})$
if $(\exists \Omega, LI$ **s.t.** $(\alpha, \Omega, LI) \in Passed)$ **then**
  if $(\Omega \neq \emptyset \wedge Dep(\alpha, t, LI))$ **then**
    return $\{t.\omega | \omega \in \Omega\}$
  end if
  return $\emptyset$
end if
if $(M \in bad)$ **then**
  return $\{t\}$
end if
$Traces\ \Omega = \emptyset$;
for all $t' \in En(M)$ **s.t** $succ(\alpha, t') \neq \emptyset \wedge succ(\alpha, t') \notin \mathcal{C}$ **do**
  Ω **=** $\Omega \cup explore(succ(\alpha, t'), t', \mathcal{C} \cup \{succ(\alpha, t')\})$
end for{Ω **contains all bad sequences of** α.}
if $(\Omega = \emptyset)$ **then**
  $Passed = Passed \cup \{(\alpha, \emptyset, \emptyset)\}$
  return $\emptyset$
end if
$LI = \{(t_c, t_s, BI) | (t_c, t_s) \in En_c(M) \times En_c^0(M) \wedge$

$$BI = \bigcup_{\omega \in \Omega} INT(Fire(\alpha, \omega), \underline{t}_c - \underline{t}_s) \subset INT(\alpha, \underline{t}_c - \underline{t}_s)\}$$

$Passed = Passed \cup \{(\alpha, \Omega, LI)\}$
if $(Dep(\alpha, t, LI))$ **then**
  return $\{t.\omega | \omega \in \Omega\}$
end if
return $\emptyset$

In *explore* of α_1, function *explore* is called for α_4 ($explore(\alpha_4, t_3, \{\alpha_0, \alpha_1, \alpha_4\})$). This call returns, to *explore* of α_1, with $\{t_3\}$, since α_4 has a forbidden marking. In *explore* of α_1, the tuple $(\alpha_1, \{t_2t_3, t_3\}, \emptyset)$ is added to *Passed* and $\{t_1t_2t_3, t_1t_3\}$ is returned to *explore* of α_0. Then, *explore* of α_0 calls $explore(\alpha_2, t_2, \{\alpha_0, \alpha_2\})$, which in turn calls $explore(\alpha_5, t_1, \{\alpha_0, \alpha_2, \alpha_5\})$. Since α_5 has only one successor (α_0) and this successor belongs to the current path, the call of *explore* for α_5 adds $(\alpha_5, \emptyset, \emptyset)$ to *Passed* and returns to *explore* of α_2 with $\emptyset$, which, in turn, returns to *explore* of α_0.

After exploring both successors of α_0, in *explore* of α_0, we get in $\Omega = \{t_1t_2t_3, t_1t_3\}$ the set of bad paths of α_0. As the state class α_0 has a controllable transition t_1, its bad subclasses are computed: $Fire(\alpha_0, t_1t_2t_3) = \{(p_1 + p_2, 0 \leq \underline{t}_1 \leq 2 \wedge 2 \leq \underline{t}_2 \leq 3 \wedge 1 \leq \underline{t}_2 - \underline{t}_1 \leq 3)$ and $Fire(\alpha_0, t_1t_3) = (p_1 + p_2, 0 \leq \underline{t}_1 \leq 1 \wedge 2 \leq \underline{t}_2 \leq 3 \wedge 2 \leq \underline{t}_2 - \underline{t}_1 \leq 3)\}$. The firing interval of t_1 in α_0 ($[0, 4]$) is not covered by the union of intervals of t_1 in bad subclasses of α_0 ($[0, 2] \cup [0, 1] \neq [0, 4]$). Then, $(\alpha_0, \{t_1t_2t_3, t_1t_3\}, \{(t_1, t_0, \{[0, 2]\})\})$ is added to *Passed*. As t_1 is newly enabled, the empty set is returned to the function *main*, which concludes that a controller

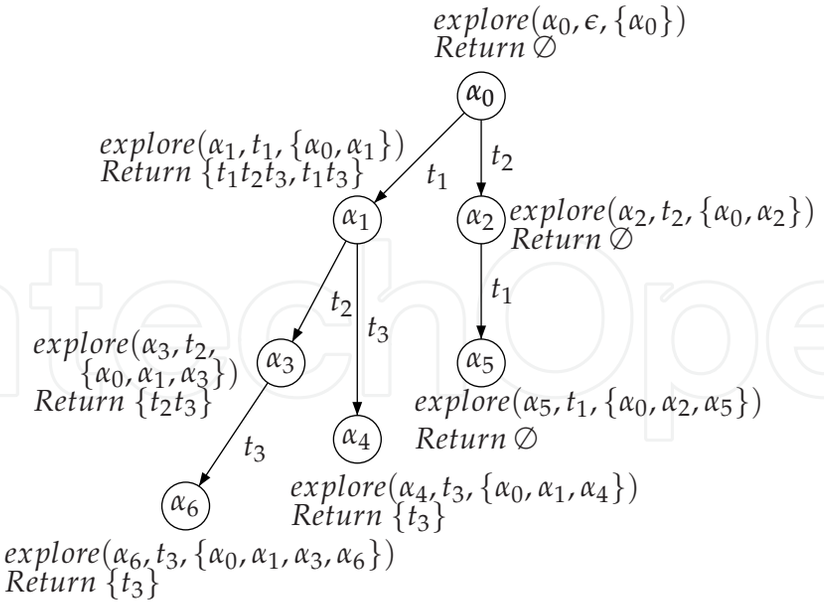

Figure 6. Applying Algorithms 1 & 2 on the TPN at Figure 1 for $AG\ not\ p_1 + p_3 = 0$

exists. According with the list *Passed*, α_0 needs to be controlled ($Ctrl[\alpha_0] = \{(t_1, t_0, \{[0,4] - [0,2]\})\}$). For all others, there is nothing to do.

Note that for this example, it is possible to carry out a static controller, which is, in this case, a mapping over controllable transitions. Indeed, it suffices to replace the static interval of t_1 with $]2,4]$. Such a controller is in general less permissive than the state dependent controller. However, its implementation is static and very simple as if the model is corrected rather than controlled.

It is also possible to carry out a marking dependent controller (a mapping over markings). Such a controller can be represented by duplicating t_1, each of them being associated with an interval and conditioned to a marking (see Table 3 and Figure 7).

This algorithm is able to determine whether a safety controller exists or not. If the algorithm fails to determine a controller, then the controller does not exist. This failure may have two

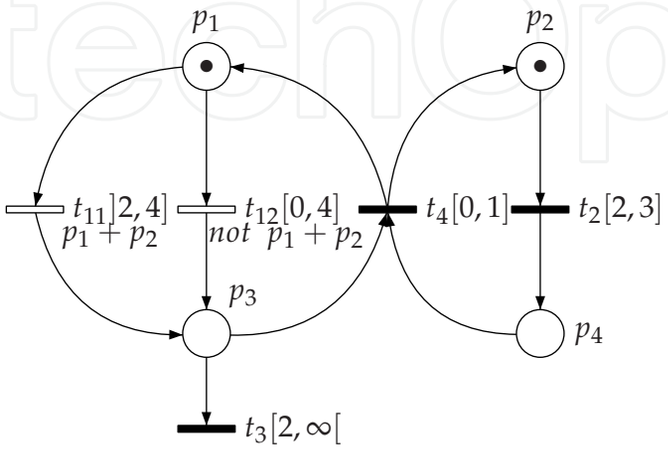

Figure 7. The controlled TPN obtained for the TPN at Figure 1 for $AG\ not\ p_1 + p_3 = 0$

Marking	Constraint to be applied on t_1
$p_1 + p_2$	$2 < \underline{t}_1 \leq 4$
Others	$0 \leq \underline{t}_1 \leq 4$

Table 3. A marking dependent controller for the TPN at Figure 1

reasons: no class having enabled controllable transitions exists in a bad path; or, calculated bad subclasses covers entire domain of controllable transitions. Note that, in a time Petri net[15] it is impossible to cancel a transition. Thus, if the entire domain of a controllable transition leads to bad states, as it cannot be canceled or delayed, the state class cannot be controlled so as to avoid bad states.

In the algorithms presented here, a state class is declared to be uncontrollable if it does not contain controllable transitions or it cannot be controlled so as to avoid all bad state classes. Note that if a state class cannot be controlled to avoid all bad classes, it can be however controlled to avoid some bad classes. To limit as little as possible the behavior of the system, the set of bad sequences of a state class α can be partitioned in two subsets: the set of bad sequences that can be avoided from α and the set of bad sequences that cannot to be avoided from α. The former set is avoided from α while the latter is let to be controlled by the predecessors of α. The function *explore* in this case should return the set of bad sequences that cannot be controlled from α. In this way, we increase the permissiveness of the controller.

The most significant advantage of this algorithm is the possibility of choosing the level of control. Three levels of control can be carried out:

1) Static controller: The control is independent of markings and states of the system. For each controllable transition, the intersection of all safe intervals is considered. Let t_c be a controllable transition whose interval needs to be restricted and $SIr(t_c) = \{Ir | \forall(\alpha, \Omega, L) \in Passed, \exists(t_c, SI) \in Ctrl[\alpha], Ir \subseteq SI \wedge Ir \neq \emptyset\}$. The static firing interval of t_c should be replaced with any interval of $SIr(t_c)$. Note that $SIr(t_c)$ may be empty. In this case, such a controller does not exist. Otherwise, it exists and its implementation is static as if the model is corrected rather than controlled. On the other hand, the permissiveness is sacrificed for the sake of simplicity of implementation. Albeit being simple, the controller has a high impact on performance of the system. For the previous example, such a controller exists and consists of replacing the static interval of t_1 with $[2, 4]$.

2) Marking dependent controller: The controller is a function of marking. The intersection of all safe intervals of controllable state classes with the same marking is considered, causing loss of permissiveness. Let t_c be a controllable transition whose interval needs to be restricted and $SIm(M, t_c) = \{Im | \forall((M, I), \Omega, L) \in Passed, \exists(t_c, SI) \in Ctrl[(M, I)], Im \subseteq SI \wedge Im \neq \emptyset\}$. For each marking M, the firing interval of each controllable transition t_c enabled in M should be any interval of $SIm(M, t_c)$. The set $SIm(M, t_c)$ may be empty and then such a controller does not exist. Otherwise, it exists and can be represented by duplicating some controllable transitions, each of them being associated with an interval and conditioned to a marking. Such a controller exists for the previous example and is given in Table 3 and the controlled TPN is what comes in Figure 7.

3) State dependent controller: The third level is the most permissive. A controllable transition is limited depending on the class the system is. In fact, making decision is delayed as much as

possible. When the algorithm is being synthesized, different scenarios are considered. During the execution, the controller decides upon the scenario the system is (the current state class).

4.3. Controller for reachability properties

The algorithm proposed here for the safety properties, is also adaptable to reachability properties. A reachability controller running in parallel with the system should satisfy the property $AFgoal$ meaning that a goal state will certainly be reached, where 'goal' is an atomic proposition specifying the goal states (Figure 8). For reachability properties, the controller should prevent all paths which terminates without reaching a goal state, or contains a loop on none goal states (Figure 8.b). Then, if we define state classes leading to such cases as bad states, a safety controller is able to control this system to satisfy the given reachability property. Thus, the algorithm proposed to safety properties is extensible to reachability properties with some minor modification and is presented in the algorithms 3 and 4. Note that, in this case, the set *goal* stands for the set of markings of goal states.

Algorithm 3 On-the-fly algorithm for the reachability control of TPN- Part A

Function $main(TPN\ \mathcal{N}, Markings\ goal)$
Where $\mathcal{N}$ **is a TPN and**
goal **is a set of goal markings.**
Let T_c **be the set of controllable transitions of** $\mathcal{N}$ **and**
α_0 **the initial state class of** $\mathcal{N}$**.**

$Passed = \emptyset$
if $(explore(\alpha_0, \epsilon, \{\alpha_0\}) \neq \emptyset)$ **then**
  {Controller does not exist}
  return
end if

for all $((\alpha, \Omega, LI) \in Passed)$ **do**
  $Ctrl[\alpha] = \bigcup_{(t_c, t_s, BI) \in LI} \{(t_c, t_s, INT(\alpha, \underline{t}_c - \underline{t}_s) - BI)\}$
end for
return

(*)
$\alpha = (M, F)$;
$En_c(M) = En(M) \cap T_c$;
$En_c^0(M) = En_c(M) \cup \{t_0\}$;
$New_c(M,t) = New(M,t) \cap T_c$;
$New(M_0, \epsilon) = En(M_0)$; $New^0(M_0, \epsilon) = En(M_0) \cup \{t_0\}$;
t_0 is a fictitious transition whose time variable is fixed at 0.
$Dep(\alpha, t, LI) \equiv$
$\exists (t_c, t_s, BI) \in LI, t_c \notin New(M,t) \wedge (t_s \notin New(M,t) \vee$
$INT(\alpha, \underline{t}_c - \underline{t}_0) \not\subseteq \bigcap_{I \in BI} (I \oplus INT(\alpha, \underline{t}_s - \underline{t}_0)))$

Algorithm 4 On-the-fly algorithm for the reachability control of TPN-Part B

Function *Traces explore*$(Class\ \alpha, Trans\ t,\ Classes\ \mathcal{C})$
if $(\exists \Omega, LI$ **s.t.** $(\alpha, \Omega, LI) \in Passed)$ **then**
  if $(\Omega \neq \emptyset \wedge Dep(\alpha, t, LI))$ **then**
    return $\{t.\omega | \omega \in \Omega\}$
  end if
  return $\emptyset$
end if
if $(M \in goal)$ **then**
  return $\emptyset$
end if
if $(En(M) = \emptyset)$ **then**
  return $\{t\}$
end if
$Traces\ \Omega = \emptyset$
for all $t' \in En(M)$ **s.t** $succ(\alpha, t') \neq \emptyset$ **do**
  if $succ(\alpha, t') \in \mathcal{C}$ **then**
    $\Omega = \Omega \cup \{t'\}$
  else
    $\Omega = \Omega \cup explore(succ(\alpha, t'), t', \mathcal{C} \cup \{succ(\alpha, t')\})$
  end if
end for
if $(\Omega = \emptyset)$ **then**
  $Passed = Passed \cup \{(\alpha, \Omega, \emptyset)\}$
  return $\emptyset$
end if
$LI = \{(t_c, t_s, BI) | (t_c, t_s) \in En_c(M) \times En_c^0(M) \wedge$

$$BI = \bigcup_{\omega \in \Omega} INT(Fire(\alpha, \omega), \underline{t}_c - \underline{t}_s) \subset INT(\alpha, \underline{t}_c - \underline{t}_s)\}$$

$Passed = Passed \cup \{(\alpha, \Omega, LI)\}$
if $(Dep(\alpha, t, LI))$ **then**
  return $\{t.\omega | \omega \in \Omega\}$
end if
return $\emptyset$

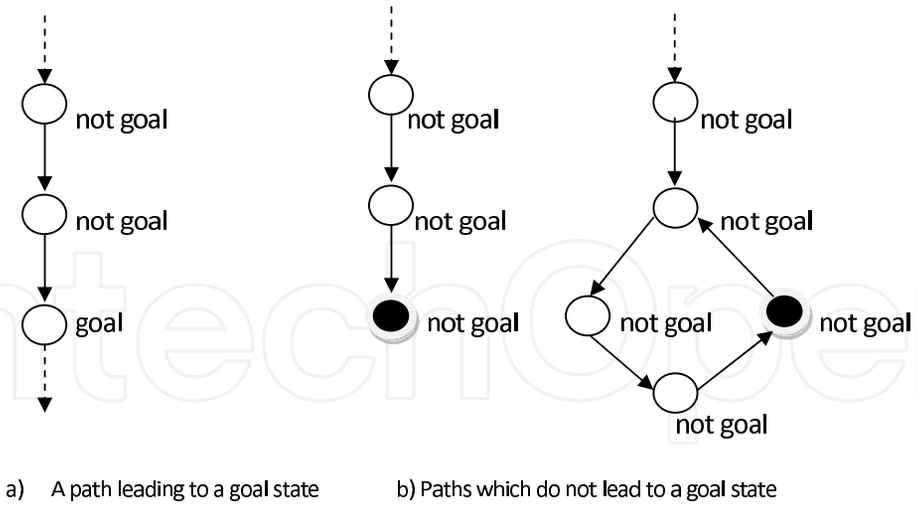

Figure 8. Paths satisfying or not a reachability property. Black states should be avoided.

5. Conclusion

In this chapter, we have proposed a completely forward on-the-fly algorithm for synthesizing safety and reachability controllers for time Petri nets. This approach guarantees to find a controller if it exists as it explores all possible state classes in the state graph and collects paths which do not satisfy the properties (bad paths).

To limit as little as possible the behavior of the system (more permissive controller), this algorithm tries to control the system behavior starting from the last to the first state classes of bad paths. If it fails to control a state class of a path, so as to avoid all bad paths, the algorithm tries to control its previous state classes. If it succeed to control a state class, there is no need to control its predecessors. The control of a state class consists of restricting the firing intervals of controllable transitions and does not need to compute any controllable predecessor.

Computing controllable predecessors involves some expensive operations such as the difference between time domains. Three levels of control can be carried out from the algorithm: the first level being independent from marking and state is static but not permissive. Second and third levels being dependent of marking and state, respectively are more permissive. One can choose to control the system during execution (third level), modify the model and make transitions conditioned to marking (second level), or re-constraining the intervals, correct the system statically before execution(first level). Correcting the system statically before the execution can reduce the impact of controller interference and solve the problem of synchronization between the controller and system.

The algorithm proposed here is decidable for a bounded TPN because the state class graph is finite and the algorithm explores, path by path, the state class graph (the exploration of a path is abandoned as soon as a loop is detected or a bad state class is reached).

One perspective of this work is the investigation of the use of more compact abstraction (abstraction by inclusion, abstraction by convex-combination) and then, extend the devised and optimized algorithm to large scale and modular systems.

Author details

Parisa Heidari and Hanifa Boucheneb
École Polytechnique de Montréal, Canada

6. References

[1] Altisen, K., Bouyer, P., Cachat, T., Cassez, F. & Gardey, G. [2005]. Introduction au contrôle des systèmes temps-réel, *Journal Européen des Systemes Automatises* 39(1-3): 367–380.

[2] Behrmann, G., Bouyer, P., Larsen, K. & Pelanek, R. [2006]. Lower and upper bounds in zone-based abstractions of timed automata, *International Journal on Software Tools for Technology Transfer* 8(3): 204 – 15.

[3] Bengtsson, J. [2002]. *Clocks, DBMs and states in timed systems.*, dissertation, Uppsala Universitet (Sweden).

[4] Berthomieu, B. & Diaz, M. [1991]. Modeling and verification of time dependent systems using time Petri nets, *IEEE Transactions on Software Engineering* 17(3): 259–273.

[5] Berthomieu, B. & Vernadat, F. [2003]. State class constructions for branching analysis of time Petri nets, *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 442–457.

[6] Boucheneb, H., Gardey, G. & Roux, O. H. [2009]. TCTL model checking of time Petri nets, *Journal of Logic and Computation* 6(19): 1509–1540.

[7] Boyer, M. & Vernadat, F. [2000]. Language and bisimulation relations between subclasses of timed petri nets with strong timing semantic, *Technical report, LAAS* .

[8] Buy, U. & Darabi, H. [2003]. Deadline-enforcing supervisory control for time Petri nets., *IMACS Multiconference on Computational Engineering in Systems Applications (CESA)*.

[9] Buy, U., Darabi, H., Lehene, M. & Venepally, V. [2005]. Supervisory control of time Petri nets using net unfolding, Vol. 2, pp. 97–100.

[10] Cassez, F., David, A., Fleury, E., Larsen, K. G. & Lime, D. [2005]. Efficient on-the-fly algorithms for the analysis of timed games, *16th International Conference on concurrency theory*, pp. 66–80.

[11] Gardey, G., Roux, O. E. & Roux, O. H. [2006a]. Safety control synthesis for time Petri nets, *8th International Workshop on Discrete Event Systems*, pp. 22–28.

[12] Gardey, G., Roux, O. & Roux, O. [2006b]. State space computation and analysis of time petri nets, *Theory and Practice of Logic Programming* 6: 301 – 20.

[13] Heidari, P. & Boucheneb, H. [2010]. Efficient method for checking the existence of a safety/ reachability controller for time Petri nets, *10th International Conference on Application of Concurrency to System Design(ACSD)*, pp. 201–210.

[14] Merlin, P. M. [1974]. *A study of the recoverability of computing systems*, Ph.d. dissertation, University of California, Irvine, United States.

[15] Merlin, P. M. & Farver, D. [1976]. Recoverability of communications protocols-implication of a theoretical study., *IEEE Trans. on Communications* 24(9): 1036–1043.

[16] Penczek, W. & Polrola, A. [2004]. Specification and model checking of temporal properties in time Petri nets and timed automata, *25th International conference on application and theory of Petri nets*, Vol. 3099 of LNCS, pp. 37–76.

[17] Ramadge, P. J. & Wonham, W. M. [1987]. Supervisory control of a class of discrete event processes, *SIAM Journal on Control and Optimization* 25(1): 206–230.
[18] Roux, O.-H., Lime, D., Haddad, S., Cassez, F. & Bérard, B. [2005]. Comparison of different semantics for time Petri nets, *3rd Automated Technology for Verification and Analysis* Volume 3707 of LNCS: 293–307.
[19] Sathaye, A. S. & Krogh, B. H. [1993]. Synthesis of real-time supervisors for controlled time Petri nets, *32nd Conference on Decision and Control*, Vol. 1, pp. 235–236.
[20] Tripakis, S. [1998]. *L'Analyse Formelle des Systèmes Temporisés en Pratique*, PhD thesis, Université Joseph Fourier - Grenoble 1 Sciences et Geographie.
[21] Wong-Toi, H. & Hoffmann, G. [1991]. The control of dense real-time discrete event systems, *30th IEEE Conference on Decision and Control Part 2 (of 3)*, Vol. 2, pp. 1527–1528.
[22] Wu, N., Chu, C., Chu, F. & Zhou, M. [2008]. Modeling and schedulability analysis of single-arm cluster tools with wafer residency time constraints using Petri net, pp. 84–89.