

State Space Abstractions for Time Petri Nets

Bernard Berthomieu and François Vernadat

Laboratoire d'Architecture et d'Analyse des Systèmes du CNRS

7 avenue du Colonel Roche, 31077 Toulouse, France

E-mail: {Bernard.Berthomieu,Francois.Vernadat}@laas.fr

July 28, 2006

1 Introduction

Since their introduction in [19], Time Petri nets (*TPN* for short) have been widely used for the specification and verification of systems in which satisfiability of time constraints is essential like communication protocols [20, 3], hardware components [27], or realtime systems [32, 12, 29].

Time Petri nets (*TPN*) extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. Assuming transition t became last enabled at time θ , and the end-points of its time interval are α and β , then t cannot fire earlier than time $\theta + \alpha$ and must fire no later than $\theta + \beta$, unless disabled by firing some other transition. Firing a transition takes no time. Many other Petri net based models with time extensions have been proposed, but none reached the acceptance of Time Petri nets. Availability of effective analysis methods, prompted by [5], certainly contributed to their widespread use, together with their ability to cope with a wide variety of modeling problems for realtime systems.

As with many formal models for realtime systems, the state spaces of Time Petri nets are typically infinite. Model checking Time Petri nets first requires to produce finite abstractions for their state spaces, that is labeled transition systems that preserve some classes of properties of the state space. This paper overviews three such constructions, the classical one introduced in [5] and two more recent abstractions proposed in [8].

The abstraction of [5], further developed in [3], computes a graph of so-called state classes. State classes represent some infinite sets of states by a marking of the net and a polyhedron capturing the times at which the transitions enabled at that marking may fire. The method has been used in many works in various contexts and has been implemented in several tools. The state class graph produced is finite if and only if the Time Petri net is bounded (admits a finite number of markings), it preserves markings, as well as the traces and complete traces of the state graph of the net, it is thus suitable for marking reachability analysis and verification of the formulas of linear time temporal logics like *LTL*.

The *state classes* construction of [5] is too coarse however for checking state reachability properties (a state associates a marking with firing time intervals for all enabled transitions). A finer abstraction allowing one to decide state reachability was proposed in [8], called *strong state classes*. But neither this latter abstraction nor that of [5] preserve branching properties of the state space, as expressed by formulas of branching time temporal logics like *CTL* or modal logics like *HML* or the μ -calculus. An abstraction preserving branching properties was first proposed in [31], called the *atomic state classes*. An equivalent but simpler construction was later introduced in [8], obtained from the previous strong state classes by a partition refinement process. This abstraction produces a graph which is bisimilar with the state graph of the net, hence preserving its branching properties.

The paper is organized as follows. Section 2 reviews the terminology of Time Petri nets and the definitions of their state spaces. The classical “state classes” construction, preserving markings and *LTL* properties, is explained in Section 3, and its properties discussed. Section 4 explains the richer “strong state classes” abstraction, that allows in addition to decide state reachability properties. Section 5 discusses preservation of branching properties and explains the “atomic state classes” construction. Some computing experiments are reported Section 6. Finally, Section 7 discusses a number of related issues and recent results, including extensions of these methods to handle enriched classes of Time Petri nets.

2 Time Petri nets and their state space

2.1 Time Petri nets

\mathbf{R}^+ and \mathbf{Q}^+ are the sets of nonnegative reals and rationals, respectively. Let \mathbf{I}^+ be the set of nonempty real intervals with nonnegative rational end-points. For $i \in \mathbf{I}^+$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point (if i bounded) or ∞ . For any $\theta \in \mathbf{R}^+$, $i \dot{-} \theta$ denotes the interval $\{x - \theta \mid x \in i \wedge x \geq \theta\}$.

Definition 1.1 A Time Petri net (or TPN) is a tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$, in which $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ is a Petri net, and $I_s : T \rightarrow \mathbf{I}^+$ is a function called the Static Interval function.

P is the set of places, T is the set of transitions, $\mathbf{Pre}, \mathbf{Post} : T \rightarrow P \rightarrow \mathbf{N}^+$ are the precondition and postcondition functions, $m^0 : P \rightarrow \mathbf{N}^+$ is the initial marking. Time Petri nets add to Petri nets the static interval function I_s , that associates a temporal interval $I_s(t) \in \mathbf{I}^+$ with every transition of the net. $Eft_s(t) = \downarrow I_s(t)$ and $Lft_s(t) = \uparrow I_s(t)$ are called the static earliest firing time and static latest firing time of t , respectively. A Time Petri net is shown in Figure 1.1.

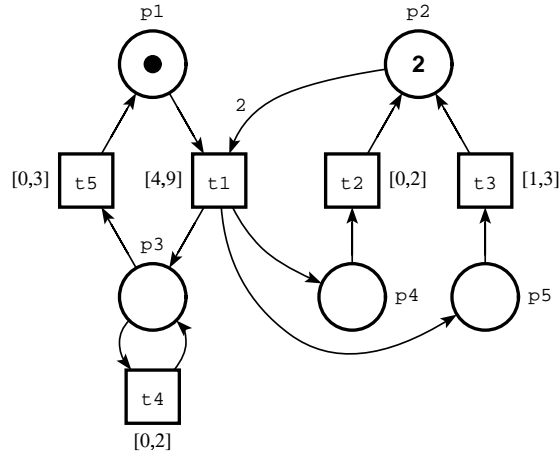


Figure 1.1: A Time Petri net

2.2 States and Firing schedules

For any function f , let $f[E]$ stand for the restriction of f to its domain intersected with E . For any $k \in \mathbf{N}$ and $f, g : P \rightarrow \mathbf{N}^+$, $f \geq g$ stands for $(\forall p \in P)(f(p) \geq g(p))$ and $f + g$ (resp. $f - g$, resp $k \cdot f$) for the function

mapping $f(p) + g(p)$ (resp. $f(p) - g(p)$, resp $k.f(p)$) with every $p \in P$.

Transition t is *enabled at marking* m iff $m \geq \mathbf{Pre}(t)$. $\mathcal{E}(m)$ denotes the set of transitions enabled at m .

Definition 1.2 (states) *A state of a TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ is a pair $s = (m, I)$ in which $m : P \rightarrow \mathbf{N}^+$ is a marking and function $I : T \rightarrow \mathbf{I}^+$ associates a temporal interval with every transition enabled at m . The initial state is $s_0 = (m_0, I_0)$, where $I_0 = I_s[\mathcal{E}(m_0)]$ (I_s restricted to the transitions enabled at m_0).*

When more convenient, the temporal information in states will be seen as as firing domains, instead of interval functions : The *firing domain* of state (m, I) is the set of real vectors $\{\underline{\phi} \mid (\forall k)(\underline{\phi}_k \in I(k))\}$, with their components indexed by the enabled transitions.

Definition 1.3 (state transitions) *States may evolve by discrete or continuous transitions. Discrete transitions are the result of firing transitions of the net, continuous (or delay) transitions are the result of elapsing of time. These transitions are defined as follows, respectively. We have:*

$(m, I) \xrightarrow{t} (m', I')$ iff $t \in T$ and:

1. $m \geq \mathbf{Pre}(t)$
2. $0 \in I(t)$
3. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
4. $(\forall k \in \mathcal{E}(m'))(I'(k) = \mathbf{if } k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k) \mathbf{ then } I(k) \mathbf{ else } I_s(k))$

$(m, I) \xrightarrow{\theta} (m, I')$ iff $\theta \in \mathbf{R}^+$ and:

5. $(\forall k \in \mathcal{E}(m))(\theta \leq \uparrow I(k))$
6. $(\forall k \in \mathcal{E}(m))(I'(k) = I(k) \dot{-} \theta)$

(3) is the standard marking transformation. From (4), the transitions not in conflict with t retain their firing interval, while those newly enabled are assigned their static intervals. A transition remaining enabled during its own firing is considered newly enabled. By (6), all firing intervals are shifted synchronously towards the origin as time elapses, and truncated to nonnegative times. (5) prevents time to elapse as soon

as the latest firing time of some transition is reached. These conditions ensure that enabled transitions fire in their temporal interval, unless disabled by firing some other transition. Firing a transition takes no time.

The states as defined above associate exactly one firing interval with every transition enabled, whether that transition is multi-enabled or not (t is multi-enabled at m if there is some integer $k > 1$ such that $m \geq k \cdot \mathbf{Pre}(t)$). An alternative interpretation of multi-enabledness will be briefly discussed in Section 3.4.

The behavior of a *TPN* is characterized by its *state graph* defined as follows. The state graph SG captures all states resulting from delay (continuous) transitions and discrete transitions.

Definition 1.4 *The state graph of a TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ is the structure*

$$SG = (S, \overset{a}{\rightsquigarrow}, s_0), \text{ where } a \in T \cup \mathbf{R}^+, S = P^{\mathbf{N}^+} \times T^{\mathbf{I}^+}, \text{ and } s_0 = (m_0, I_s[\mathcal{E}(m_0)])$$

Any sequence of delay transitions is equivalent to a single delay transition labeled with the sum of the labels of the transitions in the sequence. Further, for each state s , we have $s \overset{0}{\rightsquigarrow} s$. So, any finite sequence of transitions ending with a discrete transition is equivalent to a sequence alternating delay and discrete transitions, called a *firing schedule* or a *time transition sequence*. An alternative definition of the state space of a *TPN* is often used in the literature, only capturing the states reachable by some firing schedule. In [5, 3, 8], for instance, the state graph is defined as $(S, \overset{t@t}{\rightarrow}, s_0)$, where $s \overset{t@t}{\rightarrow} s'$ is defined as $(\exists s'')(s \overset{\theta}{\rightsquigarrow} s'' \wedge s'' \overset{t}{\rightsquigarrow} s')$. This graph will be called the *discrete state graph*, or *DSG*, defined as follows:

Definition 1.5 *The discrete state graph of a TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ is the structure*

$$DSG = (S, \overset{t}{\rightarrow}, s_0)$$

$$\text{where } S = P^{\mathbf{N}^+} \times T^{\mathbf{I}^+}, s_0 = (m_0, I_s[\mathcal{E}(m_0)]), \text{ and } s \overset{t}{\rightarrow} s' \Leftrightarrow (\exists \theta)(\exists s'')(s \overset{\theta}{\rightsquigarrow} s'' \wedge s'' \overset{t}{\rightsquigarrow} s')$$

The states of the *DSG* are those of the *SG* reachable by sequences of transitions ending with a discrete transition. Any state of the *DSG* is a state of the *SG*, and any state of the *SG* which is not in the *DSG* is reachable from some state of the *DSG* by a continuous transition.

In spite of its name, the *DSG* is a dense graph: states may have uncountable numbers of successors by $\overset{t}{\rightarrow}$. Finitely representing state spaces involves grouping some sets of states. Several groupings can be defined, depending on the properties of the state space one would like to preserve, three essential constructions are presented in Sections 3 to 5. As will be seen, these state space abstractions are abstractions of the *DSG* rather than the *SG*. We will explain the consequences of this approximation when needed.

2.3 Illustration

A state $s = (m, I)$ is represented by a pair (m, D) , in which m is a marking and D a firing domain. If t is enabled at m , projection t of D is the firing interval $I(t)$ associated with t in s . Firing domains are described by systems of linear inequalities with one variable per transition enabled.

This way, the initial state $s_0 = (m_0, D_0)$ of the net shown in Figure 1.1 is represented by:

$$m_0 : p_1, p_2 * 2$$

$$D_0 : 4 \leq \phi_{t_1} \leq 9$$

Waiting $\theta_1 \in [4, 9]$ units of time then firing t_1 , leads to state $s_1 = (m_1, D_1)$, described as follows:

$$m_1 : p_3, p_4, p_5$$

$$D_1 : 0 \leq \phi_{t_2} \leq 2$$

$$1 \leq \phi_{t_3} \leq 3$$

$$0 \leq \phi_{t_4} \leq 2$$

$$0 \leq \phi_{t_5} \leq 3$$

Firing t_2 from s_1 after waiting $\theta_2 \in [0, 2]$ units of time, leads to state $s_2 = (m_2, D_2)$, described by:

$$m_2 : p_2, p_3, p_5$$

$$D_2 : \max(0, 1 - \theta_2) \leq \phi_{t_3} \leq 3 - \theta_2$$

$$0 \leq \phi_{t_4} \leq 2 - \theta_2$$

$$0 \leq \phi_{t_5} \leq 3 - \theta_2$$

State s_1 admits an infinity of successor states by t_2 , one for each possible θ_2 .

2.4 Some general theorems

A *TPN* is *bounded* if the marking of each place is bounded by some integer; boundedness implies finiteness of the marking reachability set. A state is *reachable* if it is a node in the state graph of the net, a marking is *reachable* if some state with that marking is reachable, and a net is *live* if, from any reachable state s and any transition t , some firing schedule firing t is firable from s . It is shown in [16] that the marking reachability problem for *TPN*'s is undecidable, undecidability of the other problems follows.

Theorem 1.1 *For Time Petri nets, the problems of marking reachability, of state reachability, of boundedness, and of liveness are undecidable.*

In some subclasses of TPN 's, the reachability relation \xrightarrow{t} is finitely branching. These TPN 's admit finite discrete state spaces iff they are bounded. The following theorem is easily proven by induction.

Theorem 1.2 *The discrete state space of a bounded Time Petri net is finite if either:*

- (i) *The static firing intervals of all transitions are unbounded,*
- (ii) *The static firing intervals of all transitions are punctual.*

In addition, for case (i), and for (ii) when all static intervals are equal, their discrete state spaces are isomorphic with the marking space of the underlying Petri net.

Theorem 1.2 allows one to interpret Petri nets as Time Petri nets in various ways. The most frequent interpretation is that Petri nets are Time Petri nets in which all transitions bear static interval $[0, \infty[$.

3 State space abstractions preserving markings and traces

3.1 State classes

The state space of a TPN may be infinite for two reasons: on one hand because some state may admit an infinity of successor states, as already seen, on the other hand because the net may be unbounded. The latter case will be discussed in Section 3.4. For handling the former case, some particular sets of states states will be grouped into *state classes*. Several groupings are possible, we review in this Section the grouping method introduced in [5] and further developed in [3].

Definition 1.6 *For each $\sigma \in T^*$, C_σ is inductively defined by: $C_\epsilon = \{s_0\}$ and $C_{\sigma,t} = \{s' | (\exists s \in C_\sigma)(s \xrightarrow{t} s')\}$*

C_σ the set of states reached in the discrete state graph by sequence σ , every state of the $DSGs$ in some C_σ . For each such set of states C_σ , let us define its marking $\mathcal{M}(C_\sigma)$ as the marking of any of the states it contains (all states in C_σ necessarily bear the same marking), and its firing domain $\mathcal{F}(C_\sigma)$ as the union of the firing domains of all the states it contains. Finally, let us denote \cong the relation satisfied by two such sets of states when they have same marking and firing domains:

Definition 1.7 $C_\sigma \cong C_{\sigma'} \Leftrightarrow \mathcal{M}(C_\sigma) = \mathcal{M}(C_{\sigma'}) \wedge \mathcal{F}(C_\sigma) = \mathcal{F}(C_{\sigma'})$.

The state class construction of [5] stems from the following observation.

Theorem 1.3 [5] *If $C_\sigma \cong C_{\sigma'}$, then any firing schedule firable from C_σ is firable from $C_{\sigma'}$, and conversely.*

The state classes of [5] are the above sets C_σ , for all firable σ , considered modulo equivalence \cong . The initial class C_ϵ holds the sole initial state. The set of classes is equipped with the transition relation: $C_\sigma \xrightarrow{t} X \Leftrightarrow C_{\sigma.t} \cong X$. The graph of state classes (*SCG* for short) is obtained by the following algorithm. State classes are represented by pairs (m, D) in which m is a marking and D is a firing domain described by a system of linear inequalities $W\underline{\phi} \leq \underline{w}$. Variables $\underline{\phi}$ are bijectively associated with the transitions enabled at m . We have $(m, D) \cong (m', D')$ iff $m = m'$ and the systems describing D and D' have equal solution sets.

Algorithm 1.1 (Computing the SCG)

With each firable sequence σ , a pair L_σ can be computed as follows. Compute the smallest set C of pairs containing L_ϵ and such that, whenever $L_\sigma \in C$ and $\sigma.t$ is firable, we have $X \cong L_{\sigma.t}$ for some $X \in C$.

- *The initial pair is $L_\epsilon = (m_0, \{Eft_s(t) \leq \underline{\phi}_t \leq Lft_s(t) \mid m_0 \geq \mathbf{Pre}(t)\})$*
- *If σ is firable and $L_\sigma = (m, D)$, then $\sigma.t$ is firable if and only if:*
 - (i) *$m \geq \mathbf{Pre}(t)$ (t is enabled at m) and;*
 - (ii) *The system $D \wedge \{\underline{\phi}_t \leq \underline{\phi}_i \mid i \neq t \wedge m \geq \mathbf{Pre}(i)\}$ is consistent.*
- *If $\sigma.t$ is firable, then $L_{\sigma.t} = (m', D')$ is obtained from $L_\sigma = (m, D)$ as follows:*

$$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t),$$

D' obtained by:

- (a) *The above firability constraints (ii) for t from L_σ are added to system D ;*
- (b) *For each k enabled at m' , a new variable $\underline{\phi}'_k$ is introduced, subject to:*

$$\underline{\phi}'_k = \underline{\phi}_k - \underline{\phi}_t, \text{ if } k \neq t \text{ et } m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k),$$

$$Eft_s(k) \leq \underline{\phi}'_k \leq Lft_s(k), \text{ otherwise;}$$

- (c) *The variables $\underline{\phi}$ are eliminated.*

The pair L_σ computed in Algorithm 1.1 represents the equivalence class by \cong of set C_σ . It is proven in [5] that the number of firing domains one can compute with Algorithm 1.1 is finite, whether the net is bounded or not. It follows that the graph of state classes of a *TPN* is finite if and only if the net is bounded.

The systems representing firing domains D are difference systems, checking equivalence \cong can be done by putting these systems into canonical form, which can be done with complexity $O(n^3)$ in time and $O(n^2)$ in space, where n is the number of variables, using e.g. the Floyd/Warshall algorithm for computing all-pair shortest paths. Implementations of Algorithm 1.1 are discussed in e.g. in [5, 3, 29, 10] for *TPN*'s and in [26] for a model closely related to Time Petri nets. It was observed in [26] that, when applying Algorithm 1.1 to state classes in canonical forms, canonization could be incrementally done in time complexity $O(n^2)$, the implementations described in [29, 10] for *TPN*'s follow from the same observation.

3.2 Illustration

As an illustration, let us build some state classes for the net Figure 1.1. The initial class c_0 is described exactly like the initial state s_0 in Section 2.3. Firing t_1 from class c_0 leads to a class c_1 described exactly like state s_1 in Section 2.3. Firing t_2 from c_1 leads to class $c_2 = (m_2, D_2)$, with $m_2 = (p_2, p_3, p_5)$ and D_2 obtained in three steps:

(a) First, the firability constraints for t_2 from c_1 are added to system D_1 , these are:

$$\phi_{t_2} \leq \phi_{t_3}$$

$$\phi_{t_2} \leq \phi_{t_4}$$

$$\phi_{t_2} \leq \phi_{t_5}$$

(b) No transition is newly enabled by firing t_2 , and transitions t_3, t_4, t_5 are not in conflict with t_2 . We thus simply add the equations $\phi'_{t_i} = \phi_{t_i} - \phi_{t_2}$, for $i \in \{3, 4, 5\}$;

(c) Variables ϕ_{t_i} are eliminated (e.g. by the classical Fourier/Motzkin elimination), D_2 is the system:

$$0 \leq \phi'_{t_3} \leq 3 \quad \phi'_{t_4} - \phi'_{t_3} \leq 1$$

$$0 \leq \phi'_{t_4} \leq 2 \quad \phi'_{t_5} - \phi'_{t_3} \leq 2$$

$$0 \leq \phi'_{t_5} \leq 3$$

The graph of state classes of the net Figure 1.1 admits twelve classes and twenty-nine transitions. Figure 1.2 shows another Time Petri net and its *SCG*. This example will be used to compare the *SCG* construction

Interpreting *LTL* formulas over timed systems requires to consider time divergence: the case of states at which time can continuously elapse without any transition being taken. Such states are either (1): states at which no transition is enabled (deadlock states) or (2): states at which all enabled transitions have unbounded firing intervals.

For case (1), existence of deadlocks in the *DSG* or the state graph *SG* is preserved by the *SCG* since the deadlocked property for a state of a *TPN* only depends on its marking, and all states in a class have the same marking: Existence of deadlock states implies existence of classes without successor classes.

For case (2), existence of such states cannot be told from the structure of the *SCG* or *DSG* while, in the state graph *SG*, for any such diverging state s , there are always some s' , $d \geq 0$ and $d' > 0$ such that $s \xrightarrow{d} s' \xrightarrow{d'} s'$. To make this second kind of divergence observable from the structure of the *SCG*, a solution is to add silent transitions $c \rightarrow c$ at all classes $c = (m, I)$ such that all transitions enabled at m have unbounded static intervals. An alternative is to assume for Time Petri nets the “fairness” hypothesis that no transition can be continuously enabled without being taken, in which case the *SCG* as built by Algorithm 1.1 is adequate for *LTL* model checking.

3.4 Variations

Checking the boundedness property on the fly: It was recalled in Section 2.4 that boundedness is undecidable for Time Petri nets. There are however a number of decidable sufficient conditions for this property, one for instance is that the underlying Petri net is bounded. Some such sufficient conditions can be checked on the fly while building the *SCG* [5, 3], adapting Algorithm 1.1. Structural analysis offers alternative sufficient conditions. Also, there are useful subclasses of Time Petri nets bounded “by construction”.

Handling multi-enabledness: A transition t is multi-enabled at m if there is some $k > 1$ such that $m \geq k \cdot \text{Pre}(t)$. The states defined in Section 2 associate exactly one interval with every transition enabled, whether that transition is multi-enabled or not; the different enabling times for t being collectively represented by the latest. In some applications, it might be relevant to distinguish the different enabling instances of a multi-enabled transition. These aspects are addressed in [2], where the notion of state of Section 2 is extended so that a transition which is enabled k times in some state is associated with k firing intervals.

The different enabling instances may be assumed independent, or ordered according to their creation dates. Algorithm 1.1 can be adapted to support that extended notion of state, the details can be found in [2].

Preserving markings but not firing sequences: A state space abstraction preserving only the reachable markings is easily obtained by a variant of Algorithm 1.1. The idea is to identify a class with any class including it. That is, one computes a set C of classes such that, whenever $L_\sigma \in C$ and $\sigma.t$ is fireable, we have $L_{\sigma.t} \lesssim X$ for some $X \in C$ (rather than $L_{\sigma.t} \cong X$), where $(m, D) \lesssim (m, D')$ iff domain D' includes D . Intuitively, if such class X exists, then any schedule fireable from a state captured by $L_{\sigma.t}$ is fireable from a state captured by X , thus one will not find new markings by storing class $L_{\sigma.t}$. Obviously, this construction no more preserves the firing sequences of the state graph and thus its *LTL* properties, it only preserves markings, but the graph built may be smaller than the *SCG* by a large factor.

4 State space abstractions preserving states and traces

Two sets C_σ and $C_{\sigma'}$ (see Definition 1.6) may be equivalent by \cong while having different contents in terms of states. The notation used for state classes in Algorithm 1.1 canonically identifies equivalence classes by \cong , but not the sets C_σ themselves. Reachability of a state cannot be proved or disproved from the *SCG* classes, in general, the *SCG* is too coarse an abstraction for that purpose.

The *strong state classes* (also called *state zones* by some authors) reviewed in this Section exactly coincide with the sets of states C_σ of Definition 1.6. The graph of strong state classes preserves the *LTL* properties of the discrete state graph of the net, but also its states, in a sense we will make precise. For building the *SSCG*, we first need a means of canonically representing the sets C_σ , clock domains serve this purpose.

4.1 Clock domains

With every reachable state, one may associate a *clock function* γ , defined as follows: with each transition enabled at the state, function γ associates the time elapsed since it was last enabled. The clock function may also be seen as a vector $\underline{\gamma}$, indexed over the transitions enabled.

In the strong state class graph construction, a class is represented by a marking and a clock system. Let

$\langle Q \rangle$ denote the solution set of inequation system Q . The set of states denoted by a marking m and a clock system $Q = \{G\underline{\gamma} \leq \underline{g}\}$ is the set $\{(m, \Phi(\underline{\gamma})) \mid \underline{\gamma} \in \langle Q \rangle\}$, where firing domain $\Phi(\underline{\gamma})$ is the solution set in $\underline{\phi}$ of:

$$\underline{0} \leq \underline{\phi}, \underline{e} \leq \underline{\phi} + \underline{\gamma} \leq \underline{l} \text{ where } \underline{e}_k = Eft_s(k) \text{ and } \underline{l}_k = Lft_s(k)$$

Each clock vector denotes a state, but different clock vectors may denote the same state, and clock systems with different solution sets may denote the same set of states. For this reason we introduce equivalence \equiv :

Definition 1.8 *Given two pairs $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$, we write $c \equiv c'$ iff $m = m'$ and clock systems Q and Q' denote the same sets of states.*

Equivalence \equiv is clearly decidable, methods for checking \equiv will be discussed in Section 4.3. Assuming such a method available, we now give an algorithm for building the graph of strong state classes.

4.2 Construction of the SSCG

Strong classes are represented by pairs (m, Q) , where m is a marking and Q is a clock system $G\underline{\gamma} \leq \underline{g}$. Clock variables $\underline{\gamma}$ are bijectively associated with the transitions enabled at m .

Algorithm 1.2 (Computing strong state classes)

For each firable firing sequence σ , a pair R_σ can be computed as follows. Compute the smallest set C including R_ϵ and such that, whenever $R_\sigma \in C$ and $\sigma.t$ is firable, we have $X \equiv R_{\sigma.t}$ for some $X \in C$.

- The initial pair is $R_\epsilon = (m_0, \{0 \leq \underline{\gamma}_t \leq 0 \mid \mathbf{Pre}(t) \leq m_0\})$
- If σ is firable and $R_\sigma = (m, Q)$, then $\sigma.t$ is firable iff:

(i) t is enabled at m , that is : $m \geq \mathbf{Pre}(t)$

(ii) Q augmented with the following constraints is consistent:

$$0 \leq \theta, Eft_s(t) - \underline{\gamma}_t \leq \theta, \{\theta \leq Lft_s(i) - \underline{\gamma}_i \mid m \geq \mathbf{Pre}(i)\}$$

- If $\sigma.t$ is firable, then $R_{\sigma.t} = (m', Q')$ is computed from $R_\sigma = (m, Q)$ by:

$$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$$

Q' obtained by:

1. A new variable is introduced, θ , constrained by conditions (ii);
2. For each i enabled at m' , a new variable $\underline{\gamma}'_i$ is introduced, obeying:

$$\underline{\gamma}'_i = \underline{\gamma}_i + \theta, \text{ if } i \neq t \text{ and } m - \mathbf{Pre}(t) \geq \mathbf{Pre}(i)$$

$$0 \leq \underline{\gamma}'_i \leq 0, \text{ otherwise}$$

3. Variables $\underline{\gamma}$ and θ are eliminated.

The temporary variable θ stands for the possible firing times of transition t . There is an arc labeled t between R_σ and c iff $c \equiv R_{\sigma,t}$. R_σ denotes the set C_σ of states reachable from s_0 by firing schedules over σ .

As for the firing domains computed by Algorithm 1.1, the set of clock systems Q with distinct solution sets one can build by Algorithm 1.2 is finite, whether the TPN is bounded or not, so the *SSCG* is finite iff the TPN is bounded. As for Algorithm 1.1, one may easily add to Algorithm 1.2 on the fly boundedness checks (see Section 3.4). Like the firing domains in the state classes of the *SCG*, the clock domains of the *SSCG* computed by Algorithm 1.2 are difference systems, for which canonical forms can be computed in polynomial time and space.

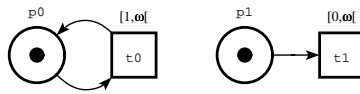
4.3 Checking clock domain equivalence \equiv

First, equivalence \equiv is easily decided in a particular case:

Theorem 1.5 (\equiv , bounded static intervals case) [8]

Consider two strong state classes $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$. Then if all transitions enabled at m or m' have bounded static intervals, we have $c \equiv c'$ iff $m = m' \wedge \langle Q \rangle = \langle Q' \rangle$.

In the general case, infinitely many distinct clock systems may denote the same set of states. An example of net for which Algorithm 1.2 would not terminate with \equiv implemented as in Theorem 1.5 is the following.



Its initial strong class is $R_\epsilon = (m_0 = \{p_0, p_1\}, \{0 = \underline{\gamma}_{t_0}, 0 = \underline{\gamma}_{t_1}\})$. Using Algorithm 1.2, firing k ($k > 0$) times t_0 from it leads to $R_{t^k} = (\{p_0, p_1\}, \{0 = \underline{\gamma}_{t_0}, k \leq \underline{\gamma}_{t_1}\})$. The clock systems in these classes have

different solution sets, but it should be observed that they all denote the same set of states: the singleton set containing the initial state. According to Definition 1.8, R_e and all R_{t^k} are equivalent by \equiv .

Since clock domains can be ordered by inclusion, a possible canonical representant for all clock domains denoting some set of states is the largest such domain. For our above example, this set can be described by the clock system $\{0 = \underline{\gamma}_{t_0}, 0 \leq \underline{\gamma}_{t_1}\}$. Such “largest” clock domains can be computed as follows:

Definition 1.9 (relaxation of clock systems) [8]

Let (m, Q) represent some strong state class, and E^∞ be the transitions enabled at m with unbounded static intervals. The relaxation of Q is the disjunction of systems $\widehat{Q} = \bigvee \{Q_e \mid e \subseteq E^\infty\}$, with Q_e obtained by:

(i) First, Q is augmented with constraints:

$$\begin{aligned} \gamma_t &< Eft_s(t), \forall t \in e \\ \gamma_t &\geq Eft_s(t), \forall t \in E^\infty \setminus e \end{aligned}$$

(ii) Then all variables $t \in E^\infty \setminus e$ are eliminated

(iii) Finally, add constraints $\gamma_t \geq Eft_s(t), \forall t \in E^\infty \setminus e$

Clock space Q is recursively split according to whether $\underline{\gamma}_k \geq Eft_s(k)$ or not, with $k \in E^\infty$. Then, in the half space in which $\underline{\gamma}_k \geq Eft_s(k)$, the upper bound of $\underline{\gamma}_k$ is relaxed. The solution set $\langle \widehat{Q} \rangle$ of \widehat{Q} is the union of the solution sets of its components. Relation \equiv can now be decided in the general case:

Theorem 1.6 (\equiv in arbitrary TPN's) [8]

Let $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$. Then $c \equiv c' \Leftrightarrow m = m' \wedge \langle \widehat{Q} \rangle = \langle \widehat{Q}' \rangle$.

It remains to show how to update the construction algorithm for the SSCG to handle arbitrary static intervals. Two solutions were proposed in [8]:

- A first is to implement \equiv in Algorithm 1.2 as in Theorem 1.6, this can be done by maintaining two representations for clock systems: that computed by Algorithm 1.2, for computing successor classes, and its relaxation, used for comparing the class with those classes already computed.

- A second is to implement \equiv in Algorithm 1.2 as as in Theorem 1.5 throughout, but integrate relaxation into Algorithm 1.2 as follows: after computation of $R_{\sigma,t} = (m', Q')$, \widehat{Q}' is computed, and R_σ is assigned as many successors classes by t as \widehat{Q}' has components, each pairing m' with a single component of \widehat{Q}' .

Both solutions are adequate when there are few concurrent transitions with unbounded static intervals, but the second may produce much larger graphs when there are many of them. A third alternative was recently proposed in [15], based on what can be seen a specialization of the above relaxation operation, called *normalization*. Normalization preserves convexity of clock systems and is computationally cheaper than relaxation. Geometrically, it computes the largest clock set that preserves the state contents of a class and that can be described by a difference system. Normalization is applied to all classes computed by Algorithm 1.2, after canonization. This solution yields the same graph as the above first alternative.

Definition 1.10 (normalization of clock systems) [15]

Let (m, Q) represent some strong state class, and E^∞ be the set of transitions enabled at m that have unbounded static interval. The normalization of system Q is the system \widetilde{Q} obtained from Q as follows (Q is assumed in canonical form, all of its constraints are tight):

For each $t \in E^\infty$

- if $\gamma_t \geq Eft_s(t)$ is redundant, then relax variable γ_t (eliminate it then add constraint $\gamma_t \geq Eft_s(t)$)
- else, and if $\gamma_t \geq Eft_s(t)$ is satisfiable, remove any constraint $\gamma_t \leq c$ and, in addition, any constraint $\gamma_t - \gamma_{t'} \leq c''$ (or $\gamma_t - \gamma_{t'} < c''$) such that $\gamma_t \geq c'$ (or $\gamma_t > c'$) and $c' + c'' \geq Eft_s(t)$

Theorem 1.7 (\equiv in arbitrary TPN's) [15]

Let $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$. Then $c \equiv c' \Leftrightarrow m = m' \wedge \langle \widetilde{Q} \rangle = \langle \widetilde{Q}' \rangle$.

4.4 Illustration

As an illustration for the construction of the *SSCG*, let us build some state classes for the net Figure 1.1.

The initial class c_0 is the pair $(m_0, K_0 = \{\gamma_{t_1} = 0\})$.

t_1 is the sole transition enabled at m_0 and all transitions enabled after firing t_1 are newly-enabled, firing t_1 from c_0 leads to $c_1 = (m_1 = (p_3, p_4, p_5), K_1 = \{\gamma_{t_2} = 0, \gamma_{t_3} = 0, \gamma_{t_4} = 0, \gamma_{t_5} = 0\})$ (renaming γ'_i into γ_i):

Firing t_2 from c_1 leads to a class $c_2 = (m_2, D_2)$, with $m_2 = (p_2, p_3, p_5)$ and K_2 obtained in three steps:

(a) First, the firability constraints for t_2 from c_1 are added to system K_1 , these are the following:

$$0 \leq \theta$$

$$0 \leq \gamma_{t_2} + \theta \leq 2$$

$$1 \leq \gamma_{t_3} + \theta \leq 3$$

$$0 \leq \gamma_{t_4} + \theta \leq 2$$

$$0 \leq \gamma_{t_5} + \theta \leq 3$$

(b) No transition is newly enabled by firing t_2 , and transitions t_3, t_4, t_5 are not in conflict with t_2 . We thus simply add the equations $\gamma'_{t_i} = \gamma_{t_i} + \theta$, for $i \in \{3, 4, 5\}$;

(c) Finally, variables θ and γ_{t_i} are eliminated, yielding system K_2 :

$$0 \leq \gamma'_{t_3} \leq 2 \quad \gamma'_{t_3} = \gamma'_{t_4}$$

$$0 \leq \gamma'_{t_4} \leq 2 \quad \gamma'_{t_3} = \gamma'_{t_5}$$

$$0 \leq \gamma'_{t_5} \leq 2 \quad \gamma'_{t_4} = \gamma'_{t_5}$$

The graph of strong state classes of the net Figure 1.1 admits eighteen classes and forty-eight transitions. The *SSCG* of the *TPN* in Figure 1.2 admits eleven classes and sixteen transitions, it is represented in Figure 1.3 (top left). Compared to the *SCG* of the same net, represented Figure 1.2, one will notice that the state sets $C_{t'.t_1}$ and $C_{t_1.t'}$ are distinguished in the *SSCG* (represented by strong classes c_4 et c_9 , respectively), while they were equivalent by \cong in the *SCG*, and similarly for the sets $C_{t_1.t_2}$ (c_{10}) and C_{t_0} (c_5).

4.5 Properties preserved

Theorem 1.4 also holds for the *SSCG*, for the same reasons. The *SSCG* preserves the properties of the discrete state graph one can express in linear time temporal logics like *LTL*. In addition, the *SSCG* allows one to check reachability of some state of the *DSG*, as expressed by the following theorem.

Theorem 1.8 *Given a state $s = (m, I)$, one can always compute a clock vector $v = \underline{\Gamma}$ such that, for any transition k enabled at m : $I(k) = I_s(k) \dot{-} \underline{\Gamma}_k$. If there are several such clock vectors, choose any one. Then a state s belongs to the discrete state graph *DSG* iff, for some class (m', Q) of the *SSCG*, we have $m = m'$ and $v \in \widehat{Q}$, where \widehat{Q} is the relaxation of system Q , as computed in Section 4.3.*

Theorem 1.9 *For any TPN, its DSG and SSCG have the same states and obey the same LTL properties.*

Theorem 1.8 is easily adapted to check membership of a state s in the state graph SG . A state $s = (m, I)$ belongs to the SG iff it is reachable from some $s' = (m', I')$ of the DSG by a continuous transition. Hence, $s = (m, I)$ belongs to SG iff, for some class (m', Q) of the $SSCG$, we have $m = m'$ and $v - d \in \widehat{Q}$ for some $d \in \mathbf{R}^+$ (v is the clock vector computed in Theorem 1.8). Finally, as for the SCG in Section 3.3, the $SSCG$ can be enriched to preserve temporal divergence resulting from transitions with unbounded firing intervals.

4.6 Variations

As for the SCG (see Section 3.4), on the fly detection of sufficient conditions for the boundedness property can easily be added to Algorithm 1.2.

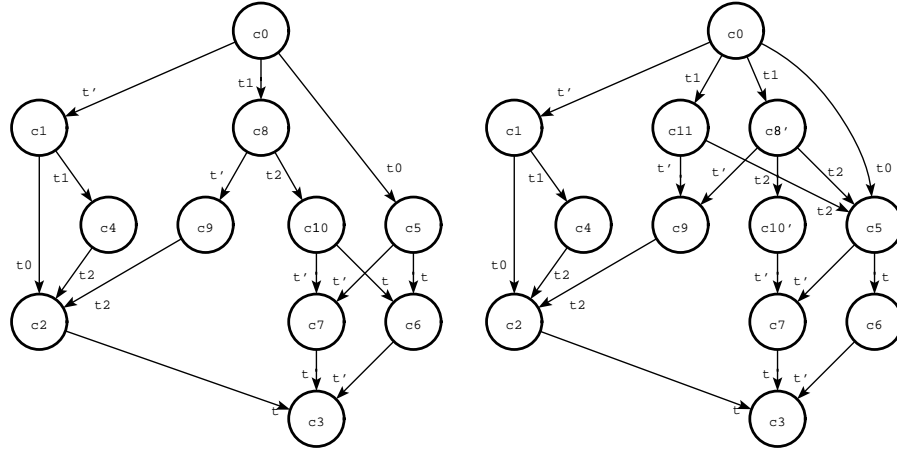
Finally, as for the SCG again, and as observed in [9, 15], one can build a variant of the $SSCG$ preserving states but not traces, by not memorizing a class when its clock domain is included in the clock domain of some already computed class. In most cases, this construction yields graphs smaller than the $SSCG$.

5 Abstractions preserving states and branching properties

5.1 Preserving branching properties

The branching properties are those expressible in branching time temporal logics like CTL , or modal logics like HML or the μ -calculus. Neither the SCG nor the $SSCG$ preserves these properties. Consider for instance the net represented Figure 1.2 with its SCG . An example of HML formula whose truth value is not preserved by this graph is $\langle t_1 \rangle \langle t_2 \rangle [t] \mathbf{F}$, expressing that it is possible to reach by a schedule of support $t_1.t_2$ a state from which t is not fireable. This property is false on the SCG but true on the state graph of the net since, after firing schedule $5.t_1.2.t_2$, one reaches the state $(\{p_2, p_4\}, I)$, with $I(t) = [2, 3]$ and $I(t') = [0, 0]$, from which t is clearly not fireable (t' must fire first).

Bisimilarity is known to preserve branching properties, we thus seek a state space abstraction bisimilar with the state graph of the TPN. A first such construction for Time Petri nets was proposed in [31], called the *Atomic state class graph*, for the subclass of TPN's in which all transitions have bounded static intervals. A



class	c_0	c_1	c_2	c_3
marking	p_0, p_4	p_0, p_5	p_2, p_5	p_3, p_5
clock domain	$0 \leq t' \leq 0$	$5 \leq t_0 \leq 5$	$0 \leq t \leq 0$	
	$0 \leq t_0 \leq 0$	$5 \leq t_1 \leq 5$		
	$0 \leq t_1 \leq 0$			
class	c_4	c_5	c_6	c_7
marking	p_1, p_5	p_2, p_4	p_3, p_4	p_2, p_5
clock domain	$0 \leq t_2 \leq 0$	$0 \leq t \leq 0$	$5 \leq t' \leq 7$	$0 \leq t \leq 3$
		$3 \leq t' \leq 5$		
class	c_8	c_9	c_{10}	$c_{8'}$
marking	p_1, p_4	p_1, p_5	p_2, p_4	p_1, p_4
clock domain	$3 \leq t' \leq 5$	$0 \leq t_2 \leq 2$	$0 \leq t \leq 0$	$3 < t' \leq 5$
	$0 \leq t_2 \leq 0$		$3 \leq t' \leq 7$	$0 \leq t_2 \leq 0$
class	c_{10}	c_{11}		
marking	p_2, p_4	p_1, p_4		
clock domain	$0 \leq t \leq 0$	$3 \leq t' \leq 3$		
	$5 < t' \leq 7$	$0 \leq t_2 \leq 0$		

Figure 1.3: *SSCG* and *ASCG* of the net Figure 1.2. Variable $\underline{\gamma}_t$ is denoted t .

specialization of this construction, only preserving formulas of the *ACTL* logic, was proposed in [22]. Rather than these constructions, we will recall that of [8], that is applicable to any *TPN* and typically yields smaller graphs. The graph built is obtained by a refinement of the *SSCG* of Section 4, it is also called the *Atomic state class graph* (*ASCG* for short) since it serves the same purposes as that in [31].

Bisimulations can be computed with an algorithm initially aimed at solving the *relational coarsest partition* problem [21]. Let \rightarrow be a binary relation over a finite set U , and for any $S \subseteq U$, let $S^{-1} = \{x | (\exists y \in S)(x \rightarrow y)\}$. A partition P of U is *stable* if, for any pair (A, B) of blocks of P , either $A \subseteq B^{-1}$ or $A \cap B^{-1} = \emptyset$ (A is said *stable wrt* B). Computing a bisimulation, starting from an initial partition P of states, is computing a stable refinement Q of P [17, 1]. An algorithm is the following:

Initially : $Q = P$

while there exists $A, B \in Q$ such that $\emptyset \subsetneq A \cap B^{-1} \subsetneq A$ **do**

replace A by $A_1 = A \cap B^{-1}$ and $A_2 = A \setminus B^{-1}$ in Q

In our case, a suitable starting partition is the graph of strong state classes of Section 4, or, alternatively [9], its variant only preserving states, discussed in Section 4.6. Computing the *ASCG* is computing a stable refinement of this graph. Our algorithm is based on the above, with two differences implying that our partition will not generally be the coarsest possible: (1) The states, typically in infinite number, are handled as convex sets, but the coarsest bisimulation does not necessary yields convex blocks, (2): Our refinement process starts from a cover of the set of states, rather than a partition (states may belong to several classes).

5.2 Partitioning a strong class

Let $c = (m, Q)$ be some class and assume $c \xrightarrow{t} c'$ with $c' = (m', Q')$. c is stable wrt c' by t iff all states of c have a successor in c' by t . If not the case, then we must compute a partition of c in which each block is stable wrt c' by t . This is done by computing the predecessor class $c'' = (m'', Q'')$ of c' by t , that is the largest set of states that have a successor by t in c' (not all these states may be reachable). Then, the subset c^t of states of c that have a successor in c' by t is exactly determined by $c^t = (m, Q^t = Q \cap Q'')$, and those not having one is $c^{\neg t} = (m, Q^{\neg t} = Q \setminus Q'')$, c^t and $c^{\neg t}$ constitute the partition of c we are looking for (we

say that c has been split). Since Q and Q'' are convex, Q^t is convex too, but Q^{-t} is not in general convex, it is a finite union of convex sets.

Computing the predecessor class c'' of c' by t is done by applying the method to obtain successor classes in Algorithm 1.2 backward. Starting from the clock system Q of c , Algorithm 1.2 first adds the firability conditions for t from c (subsystem (F) below). Next, new variables $\underline{\gamma}'_i$ are introduced for the persistent transitions (subsystem (N)) and for the newly enabled transitions. Finally, variables $\underline{\gamma}$ and θ are eliminated.

$$(Q) \quad G\underline{\gamma} \leq \underline{g} \quad (F) \quad A(\underline{\gamma}|\theta) \leq \underline{b} \quad (N) \quad \underline{\gamma}'_i = \underline{\gamma}_i + \theta$$

For computing Q'' , newly enabled transitions may be omitted, let Q^- be Q' with these variables eliminated. System Q'' is then $F \wedge N \wedge Q^-$, after elimination of variables $\underline{\gamma}'$:

5.3 Building the atomic state class graph

Algorithm 1.3 (Computing atomic state classes) [8]

Start from the SSCG (built by Algorithm 1.2 or as in Section 4.6)

while some class c is unstable wrt one of its successors c' by some t **do**

split c wrt c' by t

Collect all classes reachable from the initial class.

Splitting c replaces it by the set of classes resulting from the partition of c , as explained in Section 5.2. Only one of the resulting classes has a successor by t in c' , otherwise each subclass in the partition inherits the predecessors and successors of c , including itself and excluding c if c was successor of itself. Details and optimizations are left out. The classes of the *ASCG* are considered modulo \equiv . Termination of Algorithm 1.3 follows from finiteness of the *SSCG* and of the partitions of classes by the stability condition.

5.4 Illustration

The *ASCG* of the net Figure 1.2 admits twelve classes and nineteen transitions, it is shown Figure 1.3 (upper right). The sole non stable class of the *SSCG* (Figure 1.3 (upper left)) is c_{10} , that was split into c'_{10} and a class equivalent by \equiv to c_5 . Class c_8 then became non stable, and was in turn split into c'_8 and c_{11} .

5.5 Properties preserved

The *ASCG* is bisimilar with the discrete state graph *DSG* of the *TPN*, it thus obeys the same branching properties. In particular, it can be checked that the truth value of our example *HML* formula $\langle t_1 \rangle \langle t_2 \rangle [t] \mathbf{F}$ is preserved by the *ASCG* (consider the path $c_0 \xrightarrow{t_1} c'_8 \xrightarrow{t_2} c'_{10}$).

Theorem 1.10 *For any TPN, its DSG and ASCG have the same states and are bisimilar.*

Theorem 1.10 does not apply to the state graph *SG*. Due to the continuous transitions appearing in the *SG* but not in the *DSG*, the *SG* and *DSG* are not in general bisimilar, even considering continuous transitions as silent transitions. One will notice that the *SG* is deterministic while the *DSG* is not. Concerning their branching structure, the difference between the *DSG* and the *SG* is in fact that the *DSG* interprets time elapsing as nondeterminism: the different states reachable from a state by letting time elapse are represented in the *SG*, ordered by continuous transitions, while they are considered unordered transient states in the *DSG* and not represented.

Concerning state reachability in the *SG* or *DSG*, it can be checked on the *ASCG* as it was checked on the *SSCG* in Section 4.5.

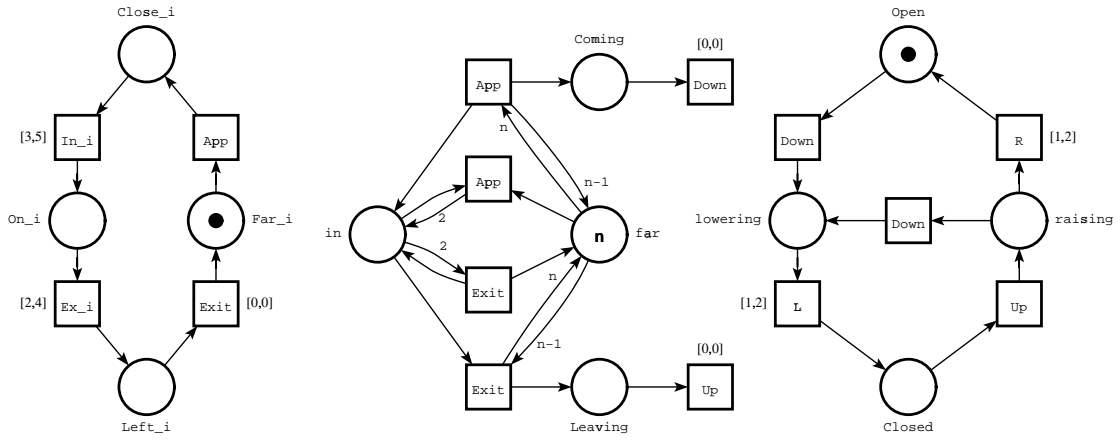
6 Computing experiments

All state class graph constructions presented in the previous Sections have been implemented in a tool named **Tina** (TIme petri Net Analyzer), described in [7].

The example shown in Figure 1.4 is a *TPN* version of the classical level crossing example. The net modeling the level crossing with n trains is obtained by parallel composition of n copies of the train model (lower left), synchronized with a controller model (upper left, n instantiated), and a barrier model (upper right). The specifications mix timed and untimed transitions (those labeled *App*).

For each n , we built with **Tina** for the level crossing model with n trains:

- its *SCG*, using either class equality as in Algorithm 1.2 (column SCG_{\cong} in Figure 1.4) or its variant in Section 3.4 using class inclusion (column SCG_{\prec}),



	SCG_{\lesssim}	SCG_{\cong}	$SSCG_{\leq}$	$SSCG_{\equiv}$	$ASCG(1)$	$ASCG(2)$
(1 train)						
Classes	10	11	10	11	12	11
Edges	13	14	13	14	16	15
CPU(s)	0.00	0.00	0.00	0.00	0.00	0.00
(2 trains)						
Classes	37	123	41	141	195	192
Edges	74	218	82	254	849	844
CPU(s)	0.00	0.00	0.00	0.00	0.01	0.01
(3 trains)						
Classes	172	3101	232	5051	6973	6966
Edges	492	7754	672	13019	49818	49802
CPU(s)	0.00	0.07	0.01	0.15	3.50	2.28
(4 trains)						
Classes	1175	134501	1807	351271	356959	356940
Edges	4534	436896	7062	1193376	3447835	3447624
CPU(s)	0.08	6.44	0.20	20.81	1264.89	320.58
(5 trains)						
Classes	10972	—	18052	—	—	—
Edges	53766	—	89166	—	—	—
CPU(s)	1.93	—	5.40	—	—	—
(6 trains)						
Classes	128115	—	217647	—	—	—
Edges	760538	—	1297730	—	—	—
CPU(s)	88.11	—	301.67	—	—	—

Figure 1.4: Level crossing example.

- its *SSCG*, using either class equality as in Algorithm 1.2 (column $SSCG_{\equiv}$) or its variant in Section 4.6 using class inclusion (column $SSCG_{\leq}$),
- its *ASCG*, using as starting partition either the *SSCG* build under the class equality rule (column $ASCG(1)$) or the *SSCG* built under the class inclusion rule (column $ASCG(2)$).

The results are shown in Figure 1.4, in terms of sizes and computing times on a typical desktop computer.

Safety properties reducing to marking reachability like “the barrier is closed when a train crosses the road” can be checked on any graph. Inevitability properties like “when all trains are far, the barrier eventually opens” can be expressed in *LTL* and may be checked on the *SCG* (SCG_{\cong}) or *SSCG* ($SSCG_{\equiv}$). Potentiality properties like “at any time, any train which is far may approach” or the liveness property for the net (see Section 2.4) must be checked on the *ASCG*. Temporal properties like “when a train approaches, the barrier closes within some given delay” generally translate to properties of the net composed with “observer nets” deduced from the properties.

As can be expected, state space abstractions have sizes and costs that grow with the amount of information they preserve, from the *SCG* built under the class inclusion rule (only preserving markings) to the *ASCG* (preserving states, *LTL* properties, and branching properties). It can be noticed that both techniques for obtaining the *ASCG* yield comparable graphs, but that computing it from the *SSCG* built under the class inclusion rule is faster: The partition-refinement algorithm tends to generate many temporary classes, starting from a smaller graph typically produces less of them.

One will notice the fast increase in number of classes with the number of trains, for all constructions. For this particular example, this number could be greatly reduced by exploiting the symmetries of the state space resulting from replication of the train model.

7 Conclusion and further issues

7.1 On state space abstractions

Among those presented, the *SCG* construction of [5, 3] presented in Section 3, should be adequate for most practical verification problems. It preserves markings and *LTL* properties, and is reasonably efficient in

practice. *LTL* properties can be checked on the abstraction produced, they could be also be checked on the fly while building the *SCG*. The *SCG* construction is supported by a number of tools, including *Tina* [7] and *Romeo* [14]. The strong state class (*SSCG*) and atomic state class (*ASCG*) graphs of [8], presented in Sections 4 and 5, bring additional verification capabilities, they are supported by the *Tina* tool. Both allow to decide state reachability, the first preserves the linear properties of the net and the second its branching properties (bisimilarity). The *SSCG* and *ASCG* constructions are closely related to the zone graph constructions for Timed Automata [1], another widely used model for realtime systems.

An alternative to state classes methods must be mentioned: It is shown in [24] that the behavior of a Bounded Time Petri net can be finitely represented by a subgraph of the discrete state graph defined in Section 2.2. Intuitively, assuming the endpoints of static intervals are integers and that all intervals are closed, or unbounded and left-closed, the set of states reachable from the initial state by continuous transitions with integer delays and then a discrete transition is finite. The abstraction obtained, called the *essential states* graph, preserves the branching properties of the *DSG*, like the *ASCG*, it is supported by the tool *Ina* [28]. For nets with intervals of small width and small endpoints, the essential states method may yield state space abstractions smaller than state class graphs, because it produces nondeterministic graphs, but state class graphs are typically smaller. In particular, scaling the static intervals of a *TPN* does not affect its state class graphs (the timing information in classes is just scaled), while its essential state graph might grow by a large factor. Implementing the essential states method is much simpler, however.

Another group of methods for analyzing Time petri nets rely on the fact that Time Petri nets can be encoded into Timed Automata, preserving weak timed bisimilarity [13]. The translation makes possible to use for *TPN*'s the analysis methods developed for Timed Automata [14]. Also, since Timed Automata and Time Petri nets share the same semantic model of timed transition systems, most techniques for analysis of Timed Automata can be adapted to Petri nets. This is done in [30], for instance for checking *TCTL* properties of *TPN*'s (*TCTL* is an extension of *CTL* with modalities annotated by time constraints). The relationships between Timed Automata and Time Petri nets are also discussed in [23], together with a number of model checking issues.

Finally, an alternative to temporal logics to check realtime properties is to analyze the firing schedules,

also called paths, of the state space. A variant of Algorithm 1.2, for instance, can be used to compute the dates at which transitions can fire along some firing sequence: It is just necessary for this to keep the auxiliary variables θ in the systems computed, instead of eliminating it. The resulting inequality system can be used to check time constraints on all schedules over some firing sequence. Such methods, called *path analysis*, are discussed in [29, 10, 25].

7.2 Extensions of *TPN*'s

Time Petri nets have been extended in various ways, increasing or not their expressiveness, and state class style methods have been developed to analyze nets with these extensions.

Extensions like inhibitor arcs and read arcs, special arcs that test boolean conditions on markings but do not transfer tokens, may significantly compact some specifications, they are supported by e.g. *Tina*. These devices extend expressiveness of Bounded Time Petri nets in terms of timed bisimulation, but preserve their decidability. State class methods are easily extended to accommodate these extensions, as they mostly impact enabledness conditions and computation of markings.

Other extensions more deeply impact expressiveness of *TPN*'s. The Scheduling-extended *TPN*'s of [18], the Preemptive *TPN*'s of [11], and the Stopwatch *TPN*'s of [4], for instance, supported respectively by the tools *Romeo*, *Oris*, and an extension of *Tina*, allow suspension and resumption of transitions. They allow one to model and analyze real time systems with preemptive scheduling constraints. State class based analysis methods extend to these models, but yield partial algorithms (termination is not guaranteed). In fact, it is shown in [4] that state reachability in these models is undecidable, even for bounded nets. Finite state space abstractions can be obtained by over-approximation methods, yielding abstractions capturing the exact behavior of nets, but possibly a larger set of states or firing schedules.

Finally, priorities are another extension of Time Petri nets of practical and theoretical value. It is shown in [6] that static priorities strictly increase the expressiveness of *TPN*'s, and that Time Petri nets extended with priorities can simulate a large subclass of Timed Automata. Extension of state class methods to *TPN*'s with priorities is being investigated.

References

- [1] R. Alur, C. Courcoubetis, N. Halbwachs, D.L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR 92: Theories of Concurrency, Springer LNCS 630*, pages 340–354, 1996.
- [2] B. Berthomieu. La méthode des classes d'états pour l'analyse des réseaux temporels – mise en œuvre, extension la multi-sensibilisation. In *Modélisation des Systèmes Réactifs, Hermes, France*, 2001.
- [3] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.
- [4] B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat. Reachability problems and abstract state spaces for time Petri nets with stopwatches. *Journal of Discrete Event Dynamic Systems*, 2007 (to appear).
- [5] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. *IFIP Congress Series*, 9:41–46, 1983.
- [6] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between Timed Automata and Bounded Time Petri Nets. In *4th Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS), Paris, 2006* (to appear).
- [7] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *Int. Journal of Production Research*, 42(14):2741–2756, 15 July 2004.
- [8] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003), Warsaw, Poland, Springer LNCS 2619*, pages 442–457, 2003.
- [9] H. Boucheneb and R. Hadjidj. Towards optimal CTL^* model checking of time Petri nets. In *Proceedings of 7th Workshop on Discrete Events Systems*, Reims, France, September 2004.
- [10] H. Boucheneb and J. Mullins. Analyse des réseaux temporels : Calcul des classes en $o(n[2])$ et des temps de chemin en $o(mn)$. *Technique et Science Informatiques*, 22:435–459, 2003.

- [11] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Timed State Space Analysis of Real-Time Preemptive Systems. *IEEE Transactions on Software Engineering*, 30(2):97–111, February 2004.
- [12] G. Bucci and E. Vicario. Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets. *IEEE Transactions on Software Engineering*, 21(12):969–992, December 1995.
- [13] F. Cassez and O. H. Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 2006. forthcoming.
- [14] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux. Roméo: A tool for analyzing time Petri nets. In *17th Int. Conf. on Computer Aided Verification, Springer LNCS 3576*, july 2005.
- [15] R. Hadjidj. *Analyse et validation formelle des systèmes temps réel*. PhD Thesis, Ecole Polytechnique de Montréal, Université de Montréal, February 2006.
- [16] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science* 4, pages 277–299, 1977.
- [17] P. K. Kanellakis and S. A. Smolka. CCSexpressions, finite state processes, and three problems of equivalence. *newblock Information and Computation*, 86:43–68, 1990.
- [18] D. Lime and O. H. Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications*. Elsevier Science, July 2003.
- [19] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD Thesis, Univ. of California, Irvine, 1974.
- [20] P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Trans. Comm.*, 24(9):1036–1043, September 1976.
- [21] P. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

- [22] W. Penczek and A. Pólrola. Abstraction and partial order reductions for checking branching properties of time petri nets. In *Proc. 22st International Conference on Application and Theory of Petri Nets (ICATPN 2001)*, Springer LNCS 2075, pages 323–342, 2001.
- [23] W. Penczek and A. Pólrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*. Studies in Computational Intelligence, Vol. 20, Springer, 2006.
- [24] L. Popova. On time petri nets. *J. Information Processing and Cybernetics EIK*, 27(4):227–244, 1991.
- [25] L. Popova and D. Schlatter. Analyzing paths in time petri nets. *Fundamenta Informaticae*, 37(3):311–327, 1999.
- [26] T. G. Rokicki. *Representing and Modeling Circuits*. PhD Thesis, Stanford Univ., Stanford, CA, 1993.
- [27] T. G. Rokicki and C. J. Myers. Automatic Verification of Timed Circuits. In *6th Conference Computer Aided Verification, CAV'94*, Springer LNCS 818, pages 468–480, jun 1994.
- [28] P. H. Starke. Ina – Integrated Net Analyzer, Reference Manual. Technical report, Humboldt Inst. of Informatics, Berlin, 1997.
- [29] E. Vicario. Static Analysis and Dynamic Steering of Time-Dependent Systems. *IEEE Transactions on Software Engineering*, 27(8):728–748, August 2001.
- [30] I. Virbitskaite and E. Pokozy. A Partial Order Method for the Verification of Time Petri Nets. In *Fundamentals of Computation Theory: 12th International Symposium (FCT'99)*, Iasi, Romania, Springer LNCS 1684, pages 547–558, 1999.
- [31] T. Yoneda and H. Ryuba. CTL model checking of Time Petri nets using geometric regions. *IEEE Transactions on Information and Systems*, E99-D(3):1–10, 1998.
- [32] T. Yoneda, A. Shibayama, B-H. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In *5th Conf. on Computer Aided Verification*, Springer LNCS 697, pages 321–332, jun 1993.

Index

- bisimilar, bisimilarity, 20, 22
- boundedness, 6, 11
- clock domain, 12
 - normalization, 16
 - relaxation, 15
- firing domain, 4
- firing schedule, 5
- multi-enabledness, 11
- path analysis, 26
- preemption, 26
- priorities, 26
- properties
 - branching properties, 18
 - CTL properties, 18
 - LTL properties, 10, 17
- state, 4
 - state graph, 5
 - discrete state graph, 5
 - state space, 5, 22
 - discrete state space, 5, 22
- state transition, 4
 - continuous transition, 4
 - delay transition, 4
 - discrete transition, 4
 - state class, 5, 7
 - atomic state class graph, 20, 21
 - state class graph, 8
 - strong state class graph, 12
- Time Petri Net, 3
- Timed Automata, 25