State class constructions for branching analysis of Time Petri nets*

Bernard Berthomieu and François Vernadat

LAAS-CNRS, 7, avenue du Colonel Roche, 31077 Toulouse Cedex, France
fax: +33 5.61.33.64.11, tel: +33. 5.61.33.63.63
e-mail:{Bernard.Berthomieu,Francois.Vernadat}@laas.fr

Abstract. This paper is concerned with construction of some state space abstractions for Time Petri nets. State class spaces were introduced long ago by Berthomieu and Menasche as finite representations for the typically infinite state spaces of Time Petri nets, preserving their linear time temporal properties. This paper proposes a similar construction that preserves their branching time temporal properties. The construction improves a previous proposal by Yoneda and Ryuba. The method has been implemented, computing experiments are reported.
Keywords: Time Petri nets, state classes, branching time temporal properties, model-checking, bisimulation, real-time systems modeling and verification.

1 Introduction

Many techniques for analysis of Petri nets or Time Petri nets proceed by building a labeled transition system (LTS) preserving the properties of interest (e.g. reachability set, deadlocks), in a first step, and then checking on this LTS the properties to be proven, in a second step. We are interested here in building labeled transition systems that preserve the branching properties of Time Petri nets. The branching properties are those one can express in modal logics like HML or branching time temporal logics like CTL^*, for instance; they are checked on the LTS produced using standard techniques not described here.

A technique for reachability analysis of Time Petri nets has been available for a long time [BM83,BD91]. This method computes state classes, which are finite representations for some infinite sets of states. State classes capture a marking and a convex firing time space for the transitions enabled at that marking. The state class graph preserves markings, as well as traces and complete traces of the state graph, and so is suitable for reachability analysis and LTL model checking. But, as it does not preserve the branching structure of the state graph (it is deterministic, by construction), it does not allow CTL^* model checking.

A state class graph construction preserving branching structure was proposed by Yoneda and Ryuba in [YR98]. Starting from an alternative definition of state classes, classes are split by linear constraints so that each state captured in a

* This work was partially supported by RNTL Projet COTRE (www.laas.fr/COTRE)

class has a successor in each of the following classes, such classes are called "Atomic". The authors show that atomicity of all classes ensures preservation of CTL^* properties. Though effective, the construction of [YR98] suffers some drawbacks that lead to class spaces larger than necessary. This paper proposes an alternative construction of the graph of atomic classes that does not suffer these drawbacks, is faster to compute, and is applicable to a larger class of nets.

The paper is organized as follows. Section 2 reviews the terminology of Time Petri nets and the available approaches for building state space abstractions that preserve LTL and CTL^* properties. Section 3 introduces strong classes, suitable as starting point for refinement into atomic classes. Section 4 discusses preservation of branching properties and introduces the revisited construction of atomic state classes. Section 5 completes the technical treatment. Computing experiments are reported Section 6.

2 Time Petri nets, states, state classes

2.1 Time Petri nets

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 $\theta \in \mathbf{R}^+$, $i \mathbin{\dot{-}} \theta$ denotes the interval $\{x - \theta | x \in i \wedge x \geq \theta\}$.

Definition 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.*

Function I_s 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 and latest firing times of t, respectively. A Time Petri net is shown in Figure 1.

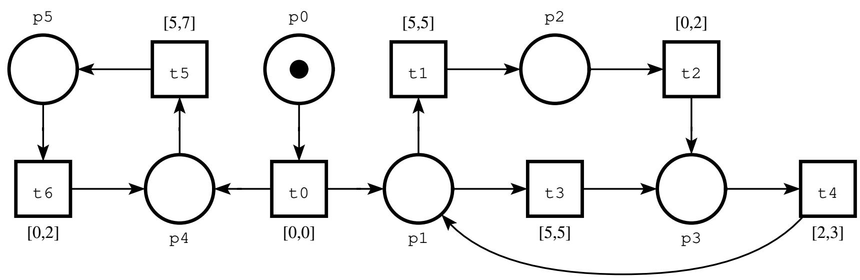


Fig. 1. A Time Petri net

2.2 States and Firing schedules

States, and the temporal state transition relation $\xrightarrow{t@\theta}$, are defined as follows:

Definition 2. *A* state *of a TPN is a pair* $s = (m, I)$ *in which* m *is a marking and* I *is a function called the* interval *function. Function* $I : T \rightarrow \mathbf{I}^+$ *associates a temporal interval with every transition enabled at* m.

We write $(m, I) \xrightarrow{t@\theta} (m', I')$ *iff* $\theta \in \mathbf{R}^+$ *and:*

1. $m \geq \mathbf{Pre}(t) \ \wedge \ \theta \geq \downarrow I(t) \ \wedge \ (\forall k \in T)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k))$
2. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
3. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow$
   $I'(k) = \ \mathbf{if} \ \ k \neq t \wedge m\text{–}\mathbf{Pre}(t) \geq \mathbf{Pre}(k) \ \ \mathbf{then} \ \ I(k) \dot{-} \theta \ \ \mathbf{else} \ \ I_s(k))$

We have $s \xrightarrow{t@\theta} s'$ if firing transition t from s at time θ after t became last enabled leads to state s'. (1) ensures that transitions fire in their temporal interval, unless disabled by firing some other transition. (2) is the standard marking transformation. From (3), transitions persistent wrt t have their interval shifted by θ and truncated to nonnegative times, and newly enabled transitions are assigned their static intervals. A transition that remained enabled during its own firing is considered newly enabled (alternative interpretations of multi-enabledness are investigated in [Ber01a]). Firing a transition takes no time.

In the sequel, $s \xrightarrow{t} s'$ stands for $(\exists \theta)(s \xrightarrow{t@\theta} s')$, $s \xrightarrow{t_1 \ldots t_n} s'$ for $(\exists s_1 \ldots s_{n-1})(s \xrightarrow{t_1} s_1 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} s_{n-1} \xrightarrow{t_n} s')$, $s \rightarrow s'$ for $(\exists t)(s \xrightarrow{t} s')$, and $\xrightarrow{*}$ for the reflexive and transitive closure of $\rightarrow$.

Definition 3. *The* state graph *of a TPN is the structure*
$SG = (S, \xrightarrow{t@\theta}, s_0)$, *where:*

$S = \{s | s_0 \xrightarrow{*} s\}$ *is the set of states reachable from the initial state* s_0
$s_0 = (m_0, I_0)$, *where* $I_0(t) = I_s(t)$ *for any* t *enabled at* m_0.

A *firing schedule* is a sequence of successively firable transitions, called its *support*, together with their relative firing times.

It is often convenient to see the temporal information in states as a firing domain, instead of an interval function: The *firing domain* of state (m, I) is the set of vectors $\{\underline{\phi} | (\forall k)(\underline{\phi}_k \in I(k))\}$, with components indexed by the enabled transitions. Given a state s, $\mathcal{M}(s)$ denotes its marking, and $\mathcal{F}(s)$ its firing domain.

Transitions may fire at any time in their temporal intervals, so states typically admit an infinity of successors. Finitely representing state spaces involves grouping states into sets, called state classes. All class space constructions discussed in this paper can be explained from the following characteristic systems.

2.3 Characteristic systems

The times at which transitions in a firing sequence σ can fire (ranged over by the *path variables* $\underline{\theta}$) and the firing domain of the state reached (ranged over by the *state variables* $\underline{\phi}$) are related in a system of linear inequalities, called here the *characteristic system* of σ, written K_σ. K_σ has shape:

(1) $P\underline{\theta} \leq \underline{p}$
(2) $\underline{0} \leq \underline{\phi}$, $\underline{e} \leq \underline{\phi} + M\underline{\theta} \leq \underline{l}$, where $\underline{e}_k = Eft_s(k)$ and $\underline{l}_k = Lft_s(k)$

These systems are computed by the following algorithm, easily proven correct from Definition 2. Variable θ_ν stands for the possible firing times of transition t:

Algorithm 1 (Computing characteristic systems)

- *The initial system is* $K_\epsilon = \{Eft_s(t) \leq \underline{\phi}_t \leq Lft_s(t) \mid \mathbf{Pre}(t) \leq m_0\}$
- *Assume* σ *is firable and* $m_0 \xrightarrow{\sigma} m$*. Then* $\sigma.t$ *is firable iff:*
  (i) $m \geq \mathbf{Pre}(t)$ *(t is enabled at m)*
  (ii) system $K_\sigma \wedge \{\underline{\phi}_t \leq \underline{\phi}_i \mid i \neq t \wedge m \geq \mathbf{Pre}(i)\}$ *is consistent*
- *If* $\sigma.t$ *is firable, then* $K_{\sigma.t}$ *is computed from* K_σ *by:*
  1. *The firability constraints for t in (ii) above are added to* K_σ*.*
  2. *Variable* $\underline{\phi}_t$ *is renamed into a new path variable :* $\underline{\theta}_\nu$
  3. *For each k enabled at m', a new variable* $\underline{\phi}'_k$ *is introduced, obeying:*
     $\underline{\phi}'_k = \underline{\phi}_k - \underline{\theta}_\nu$*, if* $k \neq t$ *and* $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$
     $Eft_s(k) \leq \underline{\phi}'_k \leq Lft_s(k)$*, otherwise*
  4. *Variables* $\underline{\phi}$ *are eliminated.*

Characteristic systems constitute a labeled tree KG, rooted at K_ϵ. Node K_σ characterizes the set of states (generally infinite) reachable from the initial state by firing schedules of support σ. The characteristic system tree is finitely branching and deterministic, but it is still in general infinite. The next step towards finite representations of state spaces is to consider these systems modulo some equivalence relation. Several relations can be used, depending on the properties of the state space to be preserved.

2.4 State classes that preserve linear properties

State classes were proposed in [BM83] for finitely representing the state graphs of bounded Time Petri nets, preserving markings and complete traces. To distinguish them from the other constructions discussed in this paper, those state classes will be referred to as the *linear state classes.*

Linear state classes stem from the following observation: Assume sequences σ and σ' lead to the same marking from m_0, and characteristic systems K_σ and $K_{\sigma'}$ have equal solution sets after projection on their state variables $\underline{\phi}$. Then the subtrees of KG rooted at K_σ and $K_{\sigma'}$, respectively, are isomorphic. Linear state classes are characteristic systems considered modulo this equivalence.

We first give an abstract definition of linear state classes, in terms of states, helpful for relating these classes with the other sorts of classes discussed in the paper. Then, their construction and properties are recalled.

Definition 4. *The* linear state class graph *of a TPN is the structure*
$LSCG = (C/\cong, \xrightarrow{t}, [\{s_0\}]_\cong)$*, where*

C is the cover[1] *of S inductively defined by:*

$$C = \bigcup_{\sigma \in T^*} C_\sigma, \text{ where } C_\epsilon = \{s_0\}, C_{\sigma.t} = \{s \mid (\exists s' \in C_\sigma)(s' \xrightarrow{t} s)\}$$

$$c \cong c' \text{ iff } (\exists s \in c)(\exists s' \in c')(\mathcal{M}(s) = \mathcal{M}(s') \wedge \bigcup_{s \in c} \mathcal{F}(s) = \bigcup_{s' \in c'} \mathcal{F}(s'))$$

$$c \xrightarrow{t} c' \text{ iff } (\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s')$$

[1] A cover of e is a set of non-empty subsets of e whose union contains e

Each C_σ captures all states reachable from the initial state by firing schedules of support σ. A state may belong to several classes. All states in a class have the same marking. State classes are considered modulo equivalence $\cong$: Defining the marking of a class as the marking of its states, and the firing domain of a class as the union of the firing domains of all states captured in the class, two classes are equivalent by $\cong$ iff their markings and firing domains are equal.

Linear state classes are exactly represented by the characteristic systems of Section 2.3, after elimination of path variables $\underline{\theta}$. They are computed by an algorithm similar to Algorithm 1, Section 2.3, except that the path variable $\underline{\theta}_\nu$ introduced at step 2 is eliminated at step 4. The resulting systems are difference systems.

The set of linear state classes of a TPN is finite iff the TPN is bounded [BM83]. This latter property is undecidable but there are decidable sufficient conditions for it [Ber01a]. Boundedness aspects will not be discussed further in this paper, where all TPNs considered are assumed bounded.

The $LSCG$ and SG hold the same markings and firing sequences (omitting time annotations for SG). In addition, no deadlock state may belong to the same class than a non-deadlock state, since all states in a class have same marking and the "deadlocked" property for a state only depends on its marking. So, seen as labeled transition systems, the SG and $LSCG$ are complete-trace equivalent. In terms of temporal logics, the $LSCG$ preserves LTL formulas, but clearly not CTL^* formulas, since, by construction, the $LSCG$ is deterministic.

The $LSCG$ of the net Figure 1 admits 83 classes and 160 transitions.

2.5 Atomic state classes [YR98]

Assume each state class denotes some set of states. A class c is *Atomic* if, for each class c', whenever a state in c has a successor in c' then all states of c have successors in c'. The authors show that if linear properties are preserved, and all classes are atomic, then CTL^* properties are preserved too. They also propose a two-step construction for such a graph: An initial graph is first built, that preserves linear properties, then an iterative algorithm splits non atomic classes.

It is essential for the second step that class equivalence preserves atomicity. This makes the $LSCG$ of the previous section unsuitable as the first step, for instance, as two $LSCG$ classes may be equivalent by $\cong$ while one is atomic and the other is not. The authors start from an alternative concept of state class. Though formalized differently, their treatment can be explained as follows.

First, state classes are associated with firing sequences, as follows. The class associated with σ is represented by a pair (m, Θ) in which m is the marking reached by firing σ from m_0 and system Θ is subsystem (1) of the characteristic system K_σ. Classes are considered modulo an equivalence relation: Each transition k enabled at m has a "parent", which is the last transition in σ the firing of which enabled k (or the special v if k was enabled from the start). Then classes (m, Θ) and (m', Θ') are considered equivalent if $m = m'$, their enabled transitions have same "parents", and these parents could be fired at the same absolute times (this can be checked from Θ and Θ' after a simple transformation).

Next, classes are split until all of them are atomic. The authors show that, if $c = (m, \Theta)$ is non atomic, then there must exist some linear constraint ρ, nonredundant in Θ and such that, for some successor c' of c, ρ is necessary for a state in c to have a successor in c'. Class c is then split into subclasses $(m, \Theta \wedge \rho)$ and $(m, \Theta \wedge \neg\rho)$. Their descendant classes are computed from copies of those of c, by constraining their systems by ρ and $\neg\rho$, respectively. Their algorithm has been implemented, and some computational results produced.

Though effective, this construction suffers some drawbacks:

1. Their equivalence relation relies on the history of firings rather than the state contents of classes. As firing schedules of different supports may well lead to the same state, it is too weak to identify all classes that denote equal sets of states. As a result, their initial state class graph is typically larger than necessary. An alternative notion of classes, called "strong classes", suitable as a starting point for the splitting process but not suffering this drawback, is proposed in Section 3.

2. In addition to enforcing atomicity, their refinement algorithm maintains the invariant that each class captures exactly the successors of the states in its predecessor classes. This invariant is in fact unnecessary, maintaining it generally increases the number of classes. An alternative refinement algorithm will be proposed in Section 4, that typically yields smaller graphs. Computing atomic classes amounts to refine an initial partition of states into a bisimulation, the issue has been widely investigated in the literature.

3 Strong State Classes

3.1 Abstract definition

In terms of state sets, Strong classes coincide with the cover C of the set of reachable states introduced in Definition 4 (before being quotiented by $\cong$):

Definition 5. *The* strong state class graph *of a TPN is the structure*

$$SSCG = (C, \xrightarrow{t}, \{s_0\}), \textit{ where}$$

$$C = \bigcup_{\sigma \in T^*} C_\sigma, \textit{ where } C_\epsilon = \{s_0\},\ C_{\sigma.t} = \{s | (\exists s' \in C_\sigma)(s' \xrightarrow{t} s)\}$$

$$c \xrightarrow{t} c' \textit{ iff } (\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s')$$

For LTL model checking, the $SSCG$ brings nothing more than brought by the $LSCG$. But, since atomicity is expressed in terms of states, the $SSCG$ is clearly an adequate starting point for refinement of classes into atomic classes.

For building the $SSCG$, we need means of representing strong classes and checking their equality. Clock domains are adequate for this purpose.

3.2 Clock domains and their equivalence

With any firing schedule, one may associate a *clock function* γ. With each transition enabled at the marking reached, function γ associates the time elapsed since that transition was last enabled. The clock function (seen as a vector) associated with a schedule of support σ and times $\underline{\theta}$ is exactly component $M\underline{\theta}$ in characteristic system K_σ: M_{ik} has value 1 if transition σ_i was fired after transition k was last enabled, and 0 otherwise. Adding equations $\underline{\gamma} = M\underline{\theta}$ to K_σ, and eliminating variables $\underline{\theta}$, it comes a system that relates clocks with firing domains (or states):

(1') $G\underline{\gamma} \leq \underline{g}$
(2') $\underline{0} \leq \underline{\phi}$, $\underline{e} \leq \underline{\phi} + \underline{\gamma} \leq \underline{l}$ where $\underline{e}_k = Eft_s(k)$ and $\underline{l}_k = Lft_s(k)$

Now, as for time vectors, different clock vectors may yield the same state. A suitable equivalence relation is much easier to obtain, however, as will be shown.

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 $\{(m, \Phi(\underline{\gamma})) | \underline{\gamma} \in \langle Q \rangle\}$, where firing domain $\Phi(\underline{\gamma})$ is the solution set in $\underline{\phi}$ of subsystem (2') above.

Definition 6 (equivalence $\equiv$). *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 they denote equal sets of states.*

Equivalence $\equiv$ is easily decided in a particular case:

Theorem 1 ($\equiv$, bounded static intervals case). *Consider $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$. If all transitions enabled at m or m' have bounded static intervals, then $c \equiv c'$ iff $m = m' \wedge \langle Q \rangle = \langle Q' \rangle$.*

Proof. Consider system (2') above. For any k, we have $0 \leq \underline{\gamma}_k$, $0 \leq \underline{l}_k$, and $0 \leq \underline{l}_k - \underline{\gamma}_k$. So, given $\underline{\gamma}$, and if all $\underline{l}_k$ are finite, (2') admits exactly one solution $\underline{\phi}$. □

This theorem identifies an important class of TPNs for which clock systems exactly characterize state sets: the nets in which all transitions have bounded static intervals. Though some authors do so (e.g. [YR98]), we do not wish to enforce that restriction, however, for it would prevent to mix timed and untimed transitions in specifications. Now, for ease of explanation of the constructions to come, and since constructions in the general case can be explained from those in the particular case, we will assume this restriction enforced until the end of Section 4. The treatment will be generalized to arbitrary TPNs in Section 5.

We now propose an algorithm for building the graph of strong state classes.

3.3 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. Equivalence $\equiv$ is implemented as in Theorem 1.

Algorithm 2 (Computing strong state classes)
For each firable firing sequence σ, a pair R_σ can be computed as explained below. Compute the smallest set C including R_ϵ and such that, whenever $R_\sigma \in C$ and $\sigma.t$ is firable, then either $R_{\sigma.t} \in C$ or $R_{\sigma.t}$ is equivalent by $\equiv$ to some pair 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:*
     $\underline{\gamma}'_i = \underline{\gamma}_i + \theta$*, if* $i \neq t$ *and* $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(i)$
     $0 \leq \underline{\gamma}'_i \leq 0$*, 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 classes R_σ and c if $c \equiv R_{\sigma.t}$. Each R_σ computed coincides with characteristic system K_σ, with subsystem (2) replaced by $\underline{\gamma} = M\underline{\theta}$, and then variables $\underline{\theta}$ eliminated. By Theorem 1, assuming enforced the "bounded static intervals" restriction, R_σ denotes exactly the set of states reachable from s_0 by schedules of support σ.

The set of clock systems Q with distinct solution sets one can built by Algorithm 2 is finite, whether the TPN is bounded or not, so the $SSCG$ is finite iff the TPN is bounded. The proof relies on similar arguments than used for proving finiteness of the set of linear classes in [BM83]. One may easily add on the fly boundedness checks to Algorithm 2, enforcing sufficient conditions, as done for the construction of the $LSCG$ there.

Equivalence $\equiv$ is checked by putting clock systems into canonical forms, and then checking their identity. Clock systems are difference systems; their canonical forms can be computed in polynomial time and space using e.g. Floyd/Warshall's algorithm for computing all pairs shortest paths.

The $SSCG$ of the net Figure 1 admits 107 classes and 205 transitions.

The first purpose of the $SSCG$ is to serve as initial graph for refinement into the atomic state class graph. Now, it clearly preserves LTL properties too. For the purpose of LTL model checking, the $SSCG$ would be a poor substitute for the $LSCG$, however, as the latter is typically smaller and is faster to compute.

Before introducing our revisited construction for the graph of atomic state classes, let us summarize the benefits of the $SSCG$ over the proposal of [YR98] for the initial state class graph: (1) Clock systems modulo $\equiv$ canonically represent state sets, while two non equivalent Yoneda classes may denote the same set of states. (2) The number of variables in clock systems is, on average, smaller than that of the corresponding systems in [YR98]. (3) With the extension of the method introduced Section 5, the "bounded static interval" restriction is removed (Yoneda's construction requires it).

4 Atomic State Classes revisited

4.1 Preserving branching properties

Let us state three properties of state class graphs (c and c' are classes):

(EE) $(\forall t \in T)(\forall c, c')(c \xrightarrow{t} c' \Leftrightarrow (\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s'))$
(AE) $(\forall c')(c \xrightarrow{t} c' \Rightarrow (\forall s \in c)(\exists s' \in c')(s \xrightarrow{t} s'))$
(EA) $(\forall c')(c' \xrightarrow{t} c \Rightarrow (\forall s \in c)(\exists s' \in c')(s' \xrightarrow{t} s))$

In their most general definition, state class graphs, or spaces, are covers of the set of states of the TPN such that all states in each class bear the same marking, equipped with a transition relation satisfying (EE). State class spaces are similar to the abstract state spaces of [PP01], except that we allow a concrete state to belong to several classes.

Our goal is to build state class graphs that preserve the branching properties of a TPN, that is of its state graph. Branching properties are those expressed in modal logics like HML or in branching time temporal logics like CTL^*. In absence of silent transitions, bisimilarity is known to preserve such properties [BCG88,NV95]. So, we just want to build state class graphs which are bisimilar with the state graph of the TPN, omitting time information in the latter.

The graph of atomic classes of [YR98] satisfies this property. Their algorithm enforces (EE), (AE) and (EA). It is easily shown that (AE) and (EE) imply bisimilarity with the state graph. Maintaining (EA) is unnecessary however.

Bisimulations can be computed with an algorithm initially aimed at solving the *relational coarsest partition* problem [PT87]. 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 [KS90,ACH+96]. An algorithm is the following [PT87]:

Initially : $Q = P$
while there exists $A, B \in Q$ such that $\emptyset \not\subseteq A \cap B^{-1} \not\subseteq A$ **do**
  replace A by $A_1 = A \cap B^{-1}$ and $A_2 = A - B^{-1}$ in Q

So, building a graph of atomic classes from the $SSCG$ is computing a stable refinement of it. Our algorithm is based on the above, differences include:

– As timed systems may admit infinitely many states, these are not handled individually, but as part of particular, convex, sets. The coarsest bisimulation does not necessary yields convex blocks. Our treatment enforces convexity of blocks when it conflicts with the "coarsest" property. The stability condition in the above algorithm is rephrased into a more primitive one based on nonredundancy of some linear constraint, explained in the next Section.

– Our refinement process starts from a cover of states, rather than a partition. Refinements are still sound, but the "coarsest" property is lost, again.

Compared to Yoneda's treatment, our construction enforces (AE), clearly equivalent to stability wrt successor classes, but not (EA). So, notwithstanding the differences in the definition of classes, it yields smaller graphs, in general.

4.2 Splitting strong classes

Let $c = (m, Q)$ be some class and assume $c \xrightarrow{t} c' = (m', Q')$. We want to check if c is stable wrt c' by t (that is if all states of c have a successor in c' by t), and, if not the case, to compute a partition of c in which all blocks are stable wrt c' by t. This is done as follows : if c is not stable wrt c' by t, then a linear constraint ρ is computed such that it is non redundant[2] in Q and ρ is necessary for a state of c to have a successor in c' by t. c is then split into subclasses $l = (m, Q \wedge \{\neg\rho\})$ and $r = (m, Q \wedge \{\rho\})$. Subclass l is stable wrt c' by t as it has no successor in c' by t; iterating this process from r until all subclasses of c found are stable wrt c' by t yields a partition of c stable wrt c'.

It remains to be shown how to find splitting constraints. Let us review how Algorithm 2 computes the successors of the states in c by t (omitting markings): Starting from the clock system Q of c, the firability conditions for t from c are added (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.

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

For the sake of computing splitting constraints, newly enabled transitions may be omitted, as their variables do not appear in Q. Let Q'' be system Q' with these variables eliminated. Stability of c wrt c' can be rephrased then:

$$(\forall\underline{\gamma})(Q \Rightarrow (\exists\theta, \underline{\gamma}')(F \wedge N \wedge Q''))$$

This formula asserts that the solution set of Q is included in that of $F \wedge N \wedge Q''$, projected on variables $\underline{\gamma}$. It can be checked by building system $F \wedge N \wedge Q''$ and then verifying that all constraints of this system are redundant in Q. Any non-redundant one provides a splitting constraint for c. Since there are finitely many constraints in $F \wedge N \wedge Q''$, the partitioning process described above terminates.

Convexity of partition members has been favored over the "coarsest" property of the partition. Depending on the order in which splitting constraints are considered, one may obtain different partitions of c, of possibly different sizes.

4.3 The revisited atomic state class graph

Definition 7. *Given a bisimulation relation $\approx$ on states, the* atomic state class graph*, over $\approx$, of a TPN is the structure*

$$ASCG = (A, \xrightarrow{t}, \{s_0\}),\ \textit{where}$$
$$A = \bigcup_{c \in C} (c/\approx),\ \textit{where } C \textit{ is the set of strong state classes}$$
$$c \xrightarrow{t} c' \ \textit{iff}\ (\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s')$$

[2] ρ is non-redundant in Q iff both systems $Q \wedge \{\rho\}$ and $Q \wedge \{\neg\rho\}$ are consistent.

For any TPN, its $ASCG$ and state graph SG are bisimilar. In practice, the $ASCG$ will be built over the bisimulation obtained by the splitting method of Section 4.2. An algorithm is the following:

Algorithm 3 (Computing atomic state classes)

Start from the SSCG
while *some class c is unstable wrt one of its successors c'* **do**
split c wrt c'
restore property (EE)
Collect all classes reachable from the initial class.

Splitting c replaces it by a pair $\{l, r\}$. Each of the l and r inherits the predecessors and successors of c, including themselves and excluding c if c was successor of itself. Classes, including those obtained as results of splits, are considered modulo $\equiv$. Checking (EE) can be done with an algorithm similar to atomicity detection: some state of class a has a successor in b iff the clock set of the potential predecessors of b, computed as in Section 4.2, intersects with the clock set of a. This amounts to checking consistency of an inequation system. Details and optimizations are left out.

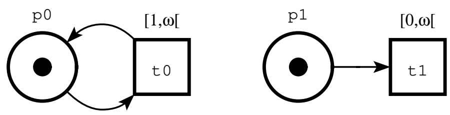
Termination of Algorithm 3 follows from finiteness of the $SSCG$ and of partitions of classes by the stability condition.

The $ASCG$ of the net Figure 1 admits 101 classes and 431 transitions. For that example, the $ASCG$ has less classes than the $SSCG$, but more transitions.

5 Handling unbounded static intervals

We assumed from Section 3.2 that all transitions of nets had bounded static intervals. In practice, this prevents to mix timed and untimed transitions in specifications, as the latter would be represented by transitions bearing static interval $[0, \infty[$. Removing that restriction has thus important practical benefits.

Applied to arbitrary TPNs, and with equivalence $\equiv$ implemented as in Theorem 1, Algorithm 2 would not always terminate. Consider the net below:



Its initial strong class is $(m_0 = \{p_0, p_1\}, \{0 \leq \underline{\gamma}_{t_0} \leq 0,\ 0 \leq \underline{\gamma}_{t_1} \leq 0\})$. Firing $k > 0$ times t_0 from it leads to $C_k = (m_k = \{p_0, p_1\}, \{0 \leq \underline{\gamma}_{t_0} \leq 0,\ k \leq \underline{\gamma}_{t_1}\})$. Since their clock systems have different solution sets, C_0 and all the C_k are distinct.

Now, it should be observed that C_0 and all C_k denote exactly the same set of states since their clock systems define the same interval function: $I = \{1 \leq I(t_0),\ 0 \leq I(t_1)\}$. These classes are actually equivalent by $\equiv$ (cf. Definition 6), but $\equiv$ in the general case cannot be implemented simply as in Theorem 1.

The most general clock system that yields that interval function is $\widehat{Q}=\{0\leq \underline{\gamma}_{t_0}\leq 0,\ 0\leq\underline{\gamma}_{t_1}\})$. Replacing clock systems in C_0 and the C_k by it produces a graph bisimilar to the previous as the future of a class only depends on its state contents, but finite. We now discuss computation of such "largest" clock systems.

Definition 8 (relaxation of clock systems).
Let (m,Q) represent some strong state class, and E^ω be the set of transitions enabled at m that have unbounded static interval. The relaxation *of system Q is the disjunction of systems $\widehat{Q}=\bigvee\{Q_e|e\subseteq E^\omega\}$, with Q_e obtained by:*

(i) First, Q is augmented with constraints:

$$\gamma_t < Eft_s(t), \forall t\in e$$
$$\gamma_t \geq Eft_s(t), \forall t\in E^\omega - e$$

(ii) Then all variables $t\in E^\omega - e$ are eliminated

(iii) Finally, constraints $\gamma_t\geq Eft_s(t), \forall t\in E^\omega - e$, are added again

Clock space Q is recursively partitioned according to whether $\underline{\gamma}_k\geq Eft_s(k)$ or not, for some k corresponding to a transition with unbounded Lft_s. Then, in the half space in which $\underline{\gamma}_k\geq Eft_s(k)$, the upper bound of $\underline{\gamma}_k$ is relaxed.

Relation $\equiv$ can now be decided in the general case. The solution set $\langle\widehat{Q}\rangle$ of $\widehat{Q}$ is the union of the solution sets of its components. Intuitively, each equivalence class of state classes by $\equiv$ admits a largest element which is obtained from any state class in the equivalence class by relaxation of its clock system.

Theorem 2 ($\equiv$ in arbitrary TPNs). *Let $c=(m,Q=\{G\underline{\gamma}\leq \underline{g}\})$ and $c'=(m',Q'=\{G'\underline{\gamma}'\leq \underline{g}'\})$. Then $c\equiv c'\Leftrightarrow m=m'\wedge\langle\widehat{Q}\rangle=\langle\widehat{Q'}\rangle$.*

Proof. If $m\neq m'$, then the property is clearly true. Assume thus $m=m'$ and let us prove property (P): $(\forall m,Q,Q')((m,Q)\equiv(m,Q')\Leftrightarrow\langle\widehat{Q'}\rangle=\langle\widehat{Q}\rangle)$. We have:
(1) $(\forall(m,Q))((m,Q)\equiv(m,\widehat{Q}))$:

For each $e\subseteq E^\omega$, let Q^e be the system obtained after step (i) in Definition 8. $(m,Q)\equiv(m,\bigvee\{Q^e|e\subseteq E^\omega\})$ since the Q^e constitute a partition of Q. (1) holds if $(\forall e)((m,Q^e)\equiv(m,Q_e))$, with Q_e as in Definition 8. By construction, each $\underline{\gamma}\in\langle Q_e\rangle$ either belongs to Q^e, or only differs from some $\underline{\gamma}'\in\langle Q^e\rangle$ by its components k in $E^\omega-e$. Since, for each k, $\underline{\gamma}_k\geq Eft_s(k)$ and $\underline{\gamma}'_k\geq Eft_s(k)$, both $\underline{\gamma}$ and $\underline{\gamma}'$ define interval $[0,\infty[$ for k, so they define the same interval function.
(2) $(\forall m,Q,Q')(\langle Q'\rangle\not\subseteq\langle\widehat{Q}\rangle\Rightarrow(m,Q)\not\equiv(m,Q'))$:

Assume $\underline{\gamma}'\in Q'$ and $\underline{\gamma}'\notin\widehat{Q}$. The only case when two clock vectors may denote the same interval function is when they differ by some components corresponding to transitions with unbounded static intervals, and these components are larger or equal than the Eft_s of those transitions. If this was the case for $\underline{\gamma}'$ and $\underline{\gamma}$, then both would belong to $\widehat{Q}$, by construction. So, no $\underline{\gamma}\in Q$ may denote the same interval function than $\underline{\gamma}'$, and $(m,Q)\not\equiv(m,Q')$.
(3) $(\forall m,Q,Q')(\langle Q\rangle=\langle Q'\rangle\Rightarrow(m,Q)\equiv(m,Q'))$: obvious, proof omitted.
(4) $(\forall m,Q,Q')((m,Q)\equiv(m,Q')\Rightarrow\langle\widehat{Q}\rangle=\langle\widehat{Q'}\rangle)$:

since $(m,Q)\equiv(m,Q')\Rightarrow(m,Q)\equiv(m,\widehat{Q'})\wedge(m,\widehat{Q})\equiv(m,Q')$, by (1)

$$\Rightarrow \langle \widehat{Q'} \rangle \subseteq \langle \widehat{Q} \rangle \wedge \langle \widehat{Q} \rangle \subseteq \langle \widehat{Q'} \rangle, \text{ by (2)}$$

(5) $(\forall m, Q, Q')(\langle \widehat{Q} \rangle = \langle \widehat{Q'} \rangle \Rightarrow (m, Q) \equiv (m, Q'))$:

since $\langle \widehat{Q} \rangle = \langle \widehat{Q'} \rangle \Rightarrow (m, \widehat{Q}) \equiv (m, \widehat{Q'})$, by (3)

$\Rightarrow (m, Q) \equiv (m, Q')$, by (1)

(P) by (4) and (5). □

It remains to show how to update the construction algorithms for the $SSCG$ and $ASCG$ to handle arbitrary static intervals. Simply implementing $\equiv$ as in Theorem 2, rather than as in Theorem 1, would be satisfying for building the $SSCG$, but it would obscure split operations for construction of the $ASCG$. A better alternative is to keep $\equiv$ implemented as in Theorem 1 throughout, and integrate relaxation into Algorithm 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 $\widehat{Q'}$. This does not incur any significant time or space penalty, compared to the first alternative, as relaxations would be computed anyway, and stored for speeding equivalence checks. The $ASCG$ for an arbitrary TPN is then built exactly as in Algorithm 3, except that refinement starts from the $SSCG$ obtained by the updated Algorithm 2. The classes obtained by splits need not be relaxed.

6 Computing experiments

Preliminary implementations of the algorithms for building strong and atomic state class graphs have been integrated in a tool named *Tina* [Ber01b]. Beside these constructions, *Tina* offers various other reachability graph constructions for Petri nets and Time Petri nets, including the $LSCG$.

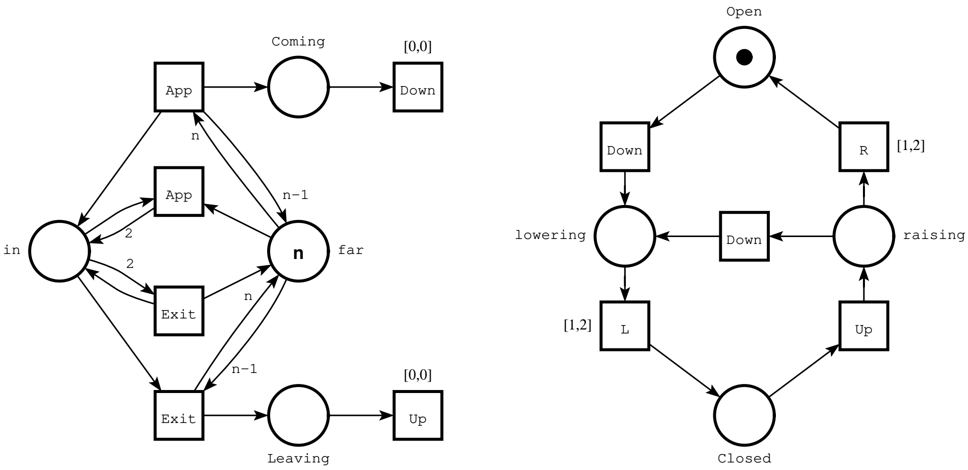
We first compared our treatment with that of [YR98], on their examples. The results in number of classes, edges, and computing times, are reported in Table 1. The last column shows the size of the minimization of the $ASCG$ under bisimulation, computed with *Aldebaran* [FGK+96]. The results confirm the expected benefits of our construction, in terms of size and cost.

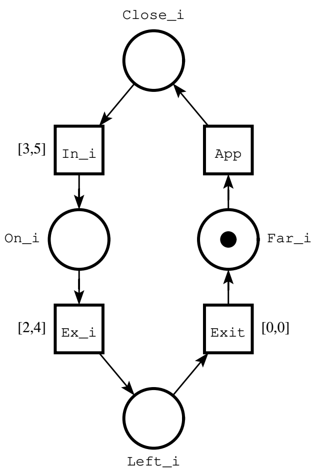
Figure 2 shows more challenging examples. The examples are Time Petri net versions of the classical level crossing example. The nets are obtained by parallel composition of n train models (lower left), synchronized with a controller model (upper left, n instantiated), and a barrier model (upper right). The specifications mix timed and untimed transitions (those labeled *App*).

For each model, we built with *Tina* its $LSCG$, $SSCG$ and $ASCG$, their sizes are also shown in Figure 2 (no figures are available for Yoneda's construction). Computing times for the $ASCG$ are much larger than for the $LSCG$ or $SSCG$, but it preserves more properties. Safety properties like "the barrier is closed when a train crosses the road" can be checked on any graph, but liveness properties like "when no train approaches, the barrier eventually opens" must be checked on the $ASCG$. Temporal properties like "when a train approaches, the barrier closes within some given delay" generally translate to safety or liveness properties of the net composed with "observer nets" deduced from the properties.

Example		*ASCG*	*Yoneda*	*Optimal*
(*Fig.* 5*a*)	*Classes*	36	53	26
	Edges	61	95	47
	CPU(*s*)	0.01	0.02	
(*Fig.* 5*b*)	*Classes*	62	64	46
	Edges	163	178	135
	CPU(*s*)	0.06	0.41	
(*Fig.* 5*c*)	*Classes*	80	168	80
	Edges	204	363	204
	CPU(*s*)	0.10	11.93	

Table 1. Yoneda's experiments.





		LSCG	*SSCG*	*ASCG*
(1 *train*)	*Classes*	11	11	12
	Edges	14	14	16
	CPU(*s*)	0.00	0.00	0.00
(2 *trains*)	*Classes*	123	141	196
	Edges	218	254	859
	CPU(*s*)	0.01	0.04	0.20
(3 *trains*)	*Classes*	3101	5051	6981
	Edges	7754	13019	49997
	CPU(*s*)	0.70	2.53	52.54
(4 *trains*)	*Classes*	134501	351271	?
	Edges	436896	1193376	?
	CPU(*s*)	85.50	972.46	?

Fig. 2. Level crossing example.

Note 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. An adhoc solution at modeling level would be e.g. to prevent train $i+1$ to approach when trains $1 \ldots i$ are not in state Far. For linear state classes, an alternative is allowed by a variant of the $LSCG$ construction discussed in [Ber01a], in which a transition enabled k times (i.e. st. $m \geq k * \mathbf{Pre}(t)$) is associated with k distinct intervals, instead of one. These intervals are ordered and, when firing transitions, the oldest is considered first. For our example, symmetries could be handled by modeling trains by a single copy of the train model in Figure 2, marked with n, instead of n copies, and using that interpretation of multi-enabledness. Extending the approach to the $ASCG$ is being investigated.

Finally, we compared our constructions with the "time-abstracting bisimulations" [TY96] built by the tool *Kronos* [Yov97] for a similar level crossing example modeled with timed automata. The figures we obtained for building the time-abstracting bisimulations are similar to those we obtained for building the $ASCG$, both in terms of sizes and computing times. These constructions serve the same purposes and are here equally expensive. No equivalent of the much faster $LSCG$ construction is available for timed automata. More generally, comparisons between these two modeling techniques should be taken with care, they have different modeling abilities and require different analysis techniques. Some examples may favor one, while others, or slight changes, may favor the other. In both cases, running times are very sensitive to changes in time constraints.

7 Conclusion

We proposed in this paper a construction of a state space abstraction for Time Petri nets that allows to prove properties relying on their branching structure, like liveness properties or those expressed in logics HML or CTL^*. It improves an existing construction in [YR98] by producing more abstract graphs, being faster to compute, and being applicable to a larger class of Time Petri nets. This construction complements the "standard" state classes method of [BM83], suitable for LTL model checking and reachability analysis.

We believe the constructions in [YR98] and those proposed here open interesting research paths in analysis and model-checking of Time Petri nets, that we intend to explore further. For generality, we focused our attention to preservation of "full" branching properties, as, in our context, this implies preservation of truth values of formulas of the strongest available temporal logics. The $ASCG$ construction can be further refined to only preserve formulas of weaker logics, like the sublogics of CTL^* considered in [PP01], using the techniques developed in their paper.

Our immediate goals are to improve the algorithmics for construction of the $ASCG$. Longer term goals include investigation of partial order techniques [PP01] to reduce further its size, and systematic methods for checking formulas of logics with time, such as $TCTL$ [ACD90]. Many formula in such logics do not actually require to build the $ASCG$ but can be more efficiently proved on the $LSCG$ of the TPN composed with some observer net built from the formula. An in-depth study of this proof technique is needed for Time Petri nets.

Acknowledgments. The authors are grateful to Tomohiro Yoneda for many discussions and making his experimental software available to us, and to Agata Półrola for her comments on the original draft of this paper.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 414–425, june 1990.

[ACH+96] 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.

[BCG88] M. C. Browne, E. M. Clarke, and O. Grümberg. Characterizing finite kripke structures in propositional temporal logics. *Theoretical Computer Science*, 59:115–131, 1988.

[BD91] 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.

[Ber01a] B. Berthomieu. La méthode des classes d'états pour l'analyse des réseaux temporels – mise en œuvre, extension à la multi-sensibilisation. In *Proc. Modélisation des Systèmes Réactifs, Toulouse, France*, October 2001.

[Ber01b] B. Berthomieu. *The Tina V2 Toolbox*. http://www.laas.fr/tina, LAAS/CNRS, 2001.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. *IFIP Congress Series*, 9:41–46, 1983.

[FGK+96] J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In *8th Conference on Computer-Aided Verification, CAV'96, Springer LNCS 1102*, July 1996.

[KS90] P. K. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[NV95] R. De Nicola and F. Vandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

[PP01] W. Penczek and A. Półrola. Abstraction and partial order reductions for checking branching properties of time petri nets. In *Proc. of ICATPN, Springer LNCS 2075*, pages 323–342, 2001.

[PT87] P. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time–abstracting bisimulations. In *8th Conference Computer-Aided Verification, CAV'96, Springer LNCS 1102*, pages 232–243, jul 1996.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1), 1997.

[YR98] 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.