Symmetry Reduced State Classes for Time Petri Nets

Pierre-Alain Bourdil[*,‡,§] Bernard Berthomieu[*,†] Silvano Dal Zilio[*,†] François Vernadat[*,‡]

[*] CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France

[†] Univ de Toulouse, LAAS, F-31400 Toulouse, France

[‡] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France

[§] Thales Avionics, 105 av du Général Eisenhower, F-31100 Toulouse, France

surname.name@laas.fr

ABSTRACT

We propose a method to exploit the symmetries of a realtime system represented by a Time Petri net for its verification by model-checking. The method handles both markings and timing constraints; it can be used in conjunction with the widely used *state classes* abstraction. The approach has been implemented and experiments are reported.

1. INTRODUCTION

Symmetry reduction aims at exploiting the symmetries of a system in order to explore its state space more efficiently. Instead of enumerating all the reachable states, one enumerates equivalence classes of states w.r.t. the symmetry relation. When applications have some symmetric structure—and large applications typically have—this provides an effective way to fight combinatorial explosion.

The idea of exploiting symmetries can be traced back to the study of program verification and high-level Petri nets. Symmetry reduction has been used since in a variety of contexts and a number of tools support symmetries, whether inferred or structural, including e.g. [13, 7, 21, 20]. Symmetry reduction can be further combined with other reduction techniques, such as stubborn sets or covering steps. Combining symmetries with symbolic analysis has been investigated for some models [9] but, in spite of some results, they seem harder to accommodate with symbolic methods than with enumerative methods.

In this paper, we propose a symmetry reduction method for a state space abstraction technique for a dense time realtime model, the State Class Graph (SCG) construction for Time Petri nets [2, 1]. While symmetry reduction of discrete Time Petri nets can be seen as a straightforward extension of symmetry reduction of (untimed) Petri nets (it has been implemented in INA [23]), symmetry reduction for dense time Time Petri nets is more challenging. In dense time, state spaces are typically infinite and finite representations are obtained through some abstraction of time. The time information is represented by systems of difference constraints with their variables associated with the transitions of the net. Some results on similar models are available in the context of Timed Automata [12] [24], but none are available yet for dense time Time Petri nets.

Our contributions. We have developed and implemented a symmetry reduction technique for the most commonly used state space abstraction for Time Petri nets, the state classes construction. First, we make some technical contributions, with the definition of a total order relation between symmetry equivalent transitions (Sect. 4) that relies on an invariant on the systems of difference constraints used to abstract time. A second, more practical, contribution is a high-level structural approach for declaring the symmetries of a net (see Sect. 5) in a way enabling efficient (polynomial) computation of canonical forms for state classes, using the previous ordering.

Outline of the paper. We start with some information on Time Petri nets and their related analysis techniques. In Sect. 3, we extend the symmetry reduction approach for Petri nets to dense-time Time Petri nets. The main technical result of the paper is found in Sect. 4, a total order between symmetry equivalent state classes. Before concluding, we present an implementation of this method integrated into the Tina toolbox [3] and discuss some experimental results.

2. TIME PETRI NETS

A *Time Petri net* [15] (or TPN) is a Petri net in which transitions are decorated with time intervals that constrain the time a transition can fire.

Let $\mathbb{I}$ be the set of non-empty real intervals with non-negative rational end-points. A 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, with P the set of places, T the set of transitions, $m_0 : P \to \mathbb{N}$ the initial marking, and $\mathbf{Pre}$, $\mathbf{Post} : T \to P \to \mathbb{N}$ the precondition and postcondition functions.
- $I_s : T \to \mathbb{I}$ is a function called the *static interval* function.

For additional modeling expressiveness, Time Petri nets can be enriched with inhibitor arcs and read arcs (testing absence or presence of tokens in a place, respectively), preemption, priorities or external synchronized data processing.

2.1 Semantics

A marking is a function $m : P \to \mathbb{N}$. A transition $t \in T$ is *enabled* at m iff $m \geq \mathbf{Pre}(t)$ (we use the pointwise comparison between functions). We denote $\mathcal{E}_{\mathcal{N}}(m)$, or simply $\mathcal{E}(m)$ when $\mathcal{N}$ is clear from context, the set of transitions enabled at m in net $\mathcal{N}$.

A *state* of a TPN is a pair $s = (m, I)$ in which m is a marking and $I : T \to \mathbb{I}$ is a partial function, called the (dynamic) interval function, that associates a temporal interval with every transition enabled at m. For $i \in \mathbb{I}$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point or ∞ if i is unbounded. For any $\theta \in \mathbb{R}_{\geq}$, let $i \dot{-} \theta = \{x - \theta | x \in i \wedge x \geq \theta\}$.

DEFINITION 1. *The semantics of a TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ is the timed transition system $SG = \langle S, s_0, \to \rangle$ where:*

- *S is the set of states of the TPN;*
- *$s_0 = (m_0, I_0)$ is the initial state, where m_0 is the initial marking and I_0 is the static interval function restricted to the transitions enabled at m_0;*
- *$\to \subseteq S \times (T \cup \mathbb{R}_{\geq}) \times S$ is the state transition relation; $(s, a, s') \in \to$ is written $s \xrightarrow{a} s'$. For any $t \in T$ and $\theta \in \mathbb{R}_{\geq}$, we have:*

(i) $(m, I) \xrightarrow{t} (m', I')$ *iff:*

1) $t \in \mathcal{E}(m)$
2) $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
3) $0 \in I(t)$
4) $(\forall k \in \mathcal{E}(m'))((A(k) \wedge I'(k) = I(k)) \vee (\neg A(k) \wedge I'(k) = I_s(k)))$
where $A(k) \Leftrightarrow k \neq t \wedge k \in \mathcal{E}(m - \mathbf{Pre}(t))$

(ii) $(m, I) \xrightarrow{\theta} (m, I')$ *iff:*

5) $(\forall k)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} \theta)$

The state transitions labelled over T (case (i) above) are the *discrete* transitions, those labelled over $\mathbb{R}_{\geq}$ (case (ii)) are the *continuous*, or time elapsing, transitions. A net transition t may fire from (m, I) if t is enabled at m and firable instantly. In the target state, the transitions k that remained enabled while t fired (t excluded) retain their intervals. Such transitions k are said to be *persistent* (w.r.t. t, m, if not clear from context). The remaining transitions, among those enabled at m', are associated with their static intervals, they are said to be *newly enabled* (w.r.t. t, m).

A continuous transition by θ is possible from (m, I) if and only if θ is not larger than $\uparrow I(k)$ for any transition $k \in \mathcal{E}(m)$. Because there may be an infinite number of continuous transitions, the state spaces of TPN are generally infinite, even when the net is bounded (its set of reachable markings is finite). To model check them, one needs finite abstractions of the state graphs.

Definition 1 states the *dense time semantics* of Time Petri nets. TPNcan also be given a *discrete time semantics*, by enforcing $\theta \in \mathbb{N}$ in time-elapsing transitions.

Finally, TPN states (whether in dense or discrete time) can be defined in terms of *clock functions* instead of *firing interval functions*, the clock of a transition being the time elapsed since it was last newly-enabled. Clock functions γ may be used to denote interval functions, since we have at any state $I(t) = I_s(t) \dot{-} \gamma(t)$, but the mapping is only surjective: when $I_s(t)$ is unbounded, several $\gamma(t)$ may obey that equation.

2.2 The State Class Abstraction

The State Class Graph (SCG for short) is a finite abstraction of SG that preserves its markings and traces.

The temporal information in states can be conveniently seen as firing domains rather than interval functions: the firing domain associated with interval function I is the set of real vectors $\{\phi | (\forall i)(\phi_i \in I(i))\}$ (ϕ_i is the coordinate of ϕ associated with transition i).

The State Class Graph construction of [2, 1] defines inductively a set of classes C_σ, where $\sigma \in T^*$ is a sequence of discrete transitions firable from the initial state. Intuitively, the class C_σ collects the states reachable after firing the sequence σ, abstracting delays. State classes are represented by pairs (m, D), where m is a marking and the firing domain D is described by a finite system of linear inequalities. We say that two state classes $C = (m, D)$ and $C' = (m', D')$ are equal, denoted $C \cong C'$, if $m = m'$ and $D \Leftrightarrow D'$ (i.e. D and D' have equal solution sets).

ALGORITHM 1 (CONSTRUCTION OF THE SCG [2]). *The SCG is the set of classes $(C_\sigma)_{\sigma \in T^*}$ obtained as follows:*

- *The initial class C_ϵ is (m_0, D_0), where D_0 is the domain defined by the set of inequalities $\{\downarrow I_s(t) \leq \phi_t \leq \uparrow I_s(t) \mid t \in \mathcal{E}(m_0)\}$.*
- *If σ is firable and $C_\sigma = (m, D)$, then:*

- *$\sigma.t$ is firable iff:*

  1. *$m \geq \mathbf{Pre}(t)$ (t is enabled at m)*
  2. *system $D \wedge F$ is satisfiable, where $F = \{\phi_t \leq \phi_i \mid i \neq t \wedge i \in \mathcal{E}(m)\}$*

- *$C_{\sigma.t} = (m', D')$ where:*

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

  D' is obtained from D in three steps:

  1. *The above conditions F are added to D;*
  2. *For each k enabled at m' a new variable ϕ'_k is introduced, obeying:*
     $\phi'_k = \phi_k - \phi_t$ *if k persistent w.r.t. t, m*
     $\downarrow I_s(k) \leq \phi'_k \leq \uparrow I_s(k)$ *otherwise*
  3. *Variables ϕ_i are eliminated (using e.g. Fourier-Motzkin elimination).*

With the construction given in Algorithm 1, it is clear that firing domains can be represented by *systems of difference constraints*, or *difference systems* for short, that is sets of inequalities of the form $\alpha_i \leq x_i$, $x_i \leq \beta_i$ or $x_i - x_j \leq \gamma_{i,j}$. If all finite endoints of static intervals are closed, then the α_i, β_i and $\gamma_{i,j}$ can be taken as rational constants. Otherwise, classically, they must be read as bounds associating a rational constant with a comparison operator in $\{\leq, <, \geq, >\}$. Such systems occur in many applications, such as temporal reasoning; they admit canonical forms that can be computed in polynomial time [17].

The SCG is certainly the best known and most widely used state space abstraction for Time Petri nets. It is finite if and only if the TPN admits a finite number of reachable markings, and it preserves both the markings and traces of the net [2, 1] and so is suitable for LTL model checking.

If only marking reachability properties are of concern, one may use instead of the SCG a typically coarser abstraction referred to here as $SCG^{\subseteq}$ [5]. Let us say that class (m, D) is included in class (m', D') when $m = m'$ and the solution set of D is included in that of D'. The $SCG^{\subseteq}$ is built like the SCG, except that classes related by inclusion are merged; this coarser construction preserves markings but over-approximates traces.

In the remainder of the text, we simply use the name of the transition, say t, as a shorthand for the firing domain variable ϕ_t associated with transition t.

3. SYMMETRY REDUCTION OF TPN

3.1 Symmetries in state classes graphs

This section defines symmetries on Time Petri nets and state class graphs. The terminology and theorem 3.1 are straightforwardly adapted from [22][18].

DEFINITION 2. *A* symmetry *of a TPN $\mathcal{N}$ is a permutation π of $P \cup T$ that preserves node types, preconditions, postconditions and static intervals. That is:*

1. $(\forall x)(x \in P \Leftrightarrow \pi(x) \in P)$
2. $(\forall t, p)(\mathbf{Pre}(t)(p) = \mathbf{Pre}(\pi(t))(\pi(p)))$
3. $(\forall t, p)(\mathbf{Post}(t)(p) = \mathbf{Post}(\pi(t))(\pi(p)))$
4. $(\forall t)(I_s(t) = I_s(\pi(t)))$

The set $S_{\mathcal{N}}$ of all symmetries of $\mathcal{N}$ forms a group under function composition. Recall that TPN states are pairs (m, I) where m is a marking and I is an interval function. Given a symmetry π of a TPN, let us define:

- the action $\pi(m)$ of π on m by $(\forall p \in P)(\pi(m)(p) = m(\pi^{-1}(p)))$
- the action $\pi(I)$ of π on I by $(\forall t \in T)(\pi(I)(t) = I(\pi^{-1}(t)))$
- the action $\pi(m, I)$ of π on a state (m, I) by $\pi(m, I) = (\pi(m), \pi(I))$.

The symmetries of a TPN induce symmetries of its state space:

LEMMA 1. *Let $\mathcal{N}$ be a TPN and $SG = \langle S, s_0, \rightarrow \rangle$ its state graph. Let π be some symmetry of $\mathcal{N}$. Then for all t, θ, m, m', I, I':*

1. $(m, I) \xrightarrow{t} (m', I') \Leftrightarrow \pi(m, I) \xrightarrow{\pi(t)} \pi(m', I')$
2. $(m, I) \xrightarrow{\theta} (m', I') \Leftrightarrow \pi(m, I) \xrightarrow{\theta} \pi(m', I')$

PROOF. 1. We have to prove that conditions 1 to 4 in Def. 1 are preserved by application of a net symmetry. For conditions (1) and (2), this is proved in [22] (Lemma 1). For (3) and (4), similarly, this is straightforward from the definitions of actions.

2. From [22] we have: $(\forall k)(m \geq Pre(k) \Leftrightarrow \pi(m) \geq Pre(\pi(k)))$, and then from the definition of actions, $\pi(I)(\pi(k)) = I(k)$ for any I and k. Hence (2) holds.

□

Two states s and s' are *equivalent* with respect to $S_{\mathcal{N}}$, written $s \approx s'$, iff there is a symmetry $\pi \in S_{\mathcal{N}}$ such that $\pi(s) = s'$. Relation $\approx$ is an equivalence relation; the equivalence class of any s by $\approx$ is finite and is called the *orbit* of s. Orbits of places and orbits of transitions are defined similarly.

For model checking purposes, defining symmetry reduction on states would be of little help however, since TPN typically have an infinite number of states. Fortunately, Lemma 1 carries over to the state class abstraction of TPN.

If π is a TPN symmetry, we define $\pi(D)$ as the set of solutions of D in which each variable t is replaced by $\pi(t)$. Likewise the action $\pi(m, D)$ of π on a class (m, D) is defined by $\pi(m, D) = (\pi(m), \pi(D))$ and state class equivalence by $(m, D) \approx (m', D') \Leftrightarrow (\exists \pi \in S_{\mathcal{N}})(\pi(m, D) = (m', D'))$.

Let $SCG_{\approx}$ denote the state class graph built like the SCG in Algorithm 1, but retaining only one state class per orbit. A marking m is *symmetric* iff $(\forall \pi \in S_{\mathcal{N}})(\pi(m) = m)$. The following theorem shows how symmetries help reachability analysis of TPN by the state classes method:

THEOREM 3.1. *Assume π is a symmetry of a TPN. Then for all t, m, m', D, D':*

1. $(m, D) \xrightarrow{t} (m', D') \Leftrightarrow \pi(m, D) \xrightarrow{\pi(t)} \pi(m', D')$
2. *if m_0 is symmetric, then for any state class C:* $C \in SCG \Leftrightarrow (\exists C^* \in SCG_{\approx})(C \approx C^*)$

PROOF. (1) directly follows from the definition of actions on state classes. (2) is proved by induction on the firing sequences of the SCG. □

3.2 Applying Symmetry Reduction

There are two main issues to be solved when trying to put symmetry reduction to work: identifying the symmetries of the reachability set and deciding when two states (state classes here) are equivalent.

Detecting symmetries: detecting net symmetries amounts to compute all net automorphisms, a problem known to be at least as hard as the graph isomorphism problem [18]. For this reason, many implementations of symmetry reduction rely on some static symmetry information provided by the user. Murφ [13], for instance, makes use of a dedicated "scalarset" type, also used in [12]. In high level Petri nets, symmetries are deduced from the syntax of inscriptions. On the other hand, some tools [20] compute these automorphisms from nets, automatically, with acceptable performances on average.

In our implementation, described in Section 5, nets will be described hierarchically as compositions of smaller nets, the composition operators specifying both the architecture of the net and a symmetry.

Checking state equivalence: There are basically two methods for checking equivalence $\approx$ on states (or state classes here): comparing a new state for $\approx$ pairwise with all computed states, or computing from the state a representative state and storing only these.

In [18], [19], three implementations of equivalence checking are discussed. All assume that the symmetry group is available, but no particular structure is assumed for it; all symmetries are handled uniformly. The first method, referred to as "iterating the symmetries", amounts to applying all possible symmetries to the new state and to check if the

result state has been stored yet. The set of symmetries to be applied is reduced thanks to particular representations of symmetries and of the set of stored states. The second method, "iterating the states", amounts to find a symmetry such that, applying it to some stored state yields a state equal to the new state. The method relies on so-called symmetry-preserving hash functions. A third method relies on state representatives; it takes advantage of the same representation of symmetries as the first method to find a minimal form for the new state. The minimal states computed are not necessarily canonical though; there may be several representatives per state orbits. The work presented in [14] builds upon these algorithms and proposes some improvements.

All three methods would be applicable to state classes. The first two only require to be able to compare classes for equality and the last requires a total ordering on classes. Assuming some total order on places and transitions, one can find suitable representations of state classes.

About their efficiency, the experiments in [14] suggest that the third algorithm typically yields the best results. Symmetry reduction in the LoLA tool [20] relies on this method.

On the other hand, the methods relying on scalarsets [13][12] or similar structural techniques trade generality for efficiency. Scalarsets may only express particular groups of symmetries, typically full symmetries in systems of processes. For full symmetries, one can compute minimal forms for states using a sorting method: full symmetries between processes can be generated by transpositions of the states of adjacent processes. By successively considering all such transpositions, retaining the smallest state at each step, one obtains a minimal state. This is similar to sorting the process states by the bubble-sort method. If some additional confluence conditions on the group are met [10], the method computes canonical forms. The method has polynomial complexity.

While the methods discussed in [18] [19] could be applied to state class graphs, they do not obviously lead to an efficient implementation; this is substantiated by our experimental data in Section 5. Though technically different, our reduction technique bears strong relationships with scalarset techniques. We restrict net symmetries to those admitting a polynomial time algorithm for computing canonical states, following [11]. For full symmetries, we compute canonical forms for state classes by sorting the states of processes. The ordering used, developed in the next section, does not require to apply permutations to the state classes, which also contributes to efficiency.

4. ORDERING STATE CLASSES

We define a total order relation $\preceq_D$ between transitions that are equivalent by symmetry ($\approx$) and enabled in the state class (m, D). We use this relation to define a total order between equivalent state classes.

Intuitively, we will have $t \preceq_D t'$ if the transition t stayed enabled longer than t' (since their last enabling date) in the execution that led to class (m, D). However, the information on the last enabling date of a transition cannot be reconstructed from the firing domain of a class, which mandates several technical results to define the ordering relation. Ultimately, the definition of $\preceq_D$ will be based on an invariant on firing domains. This is the goal of Sect. 4.2.

A similar idea was used by Hendriks et al. [12] to compare clocks in Timed Automata (TA) zones. In their work, a clock k is "before" another clock k' (in a given zone z) if the last reset date of k is older than that of k'. Nonetheless, the handling of time in TA is quite different from TPN and the usual zone constructions of TA significantly differ from the SCG construction of TPN.

4.1 Closure Form of Firing Domains

The firing domains computed during the SCG construction (see Algorithm 1) can always be written in a standard form, as follows. For every pair of transitions t, t' that are enabled in a class (m, D), we have the following inequalities in D:

$$\alpha_t \leq t \leq \beta_t \qquad \text{and} \qquad t - t' \leq \gamma_{t,t'} \quad (t \neq t')$$

The bounds α_t, β_t and $\gamma_{t,t'}$ exactly define the domain D (a class with n enabled transitions has $n \cdot (n+1)$ bounds). Also, by construction, when t is newly enabled in the class (m, D), we have $\alpha_t = \downarrow I_s(t)$ and $\beta_t = \uparrow I_s(t)$. These are the *static* timing constraints for t; we use the notation α_t^s and β_t^s for these values afterward.

We can improve the standard form of D by choosing in the above representation the tightest possible bounds preserving the associated solutions set. In this case we say that D is in *closure form*. The closure form provides a normal form for firing domains. Indeed, two domains are equal if and only if they have the same closure form. Another advantage of using closure forms for representing class domains is that they can be computed incrementally with $\mathcal{O}(n^2)$ complexity; Lemma 2 below is proved in [6].

LEMMA 2 (COMPUTING FIRING DOMAINS). *Assume $C = (m, D)$ is a class with D in closure form. Then for every transition t in $\mathcal{E}(m)$ there is a unique class $C' = (m', D')$ obtained from C by firing t such that D' is also in closure form. Moreover D' obeys the following constraints, for each distinct transitions $i, j \in \mathcal{E}(m')$:*

$$\begin{array}{rcll}
\beta'_i & = & \beta_i^s & \text{if } i \text{ newly enabled,} \\
\beta'_i & = & \gamma_{i,t} & \text{otherwise} \\
\alpha'_i & = & \alpha_i^s & \text{if } i \text{ newly enabled,} \\
\alpha'_i & = & \max(0, -\min_{k \in \mathcal{E}(m)}(\gamma_{k,i})) & \text{otherwise} \\
\gamma'_{i,j} & = & \beta'_i - \alpha'_j & \text{if } i \text{ or } j \text{ newly en.,} \\
\gamma'_{i,j} & = & \min(\gamma_{i,j}, \beta'_i - \alpha'_j) & \text{otherwise}
\end{array}$$

We can use this incremental construction to derive invariants on the coefficients of the firing domains when in closure form.

LEMMA 3. *For any class $C = (m, D)$ with D in closure form, and for any transitions i, j, k enabled at m, we have:*

1. $\gamma_{i,j} \leq \beta_i - \alpha_j$
2. $\beta_i \leq \gamma_{i,j} + \beta_j$
3. $\alpha_i \leq \gamma_{i,j} + \alpha_j$
4. $\gamma_{i,j} \leq \gamma_{i,k} + \gamma_{k,j}$
5. $0 \leq \alpha_i \leq \alpha_i^s$
6. $0 \leq \beta_i \leq \beta_i^s$
7. *if $C' = (m', D')$ is obtained from C by firing some transition, then $\beta'_i \leq \beta_i$*

PROOF. 1-4 follow from the fact that D is in closure form (bounds are tight). 5-7 are proved by induction on firing sequences using Lemma 2. □

4.2 A Total Order on Equivalent Transitions

We prove a general invariant on firing domains in closure form, obtained during the SCG construction. This property

makes explicit a relation between transitions that have the same static timing constraints (α^s and β^s), which is necessarily the case for equivalent transitions.

We say that two transitions i and j are equivalent for D, denoted $i =_D j$, if and only if, when transposing the transitions i and j in the system of difference constraints D, we obtain a system D' that has the same solution set.

LEMMA 4. *If D is in closure form then $i =_D j$ if and only if $\alpha_i = \alpha_j$, $\beta_i = \beta_j$, $\gamma_{i,j} = \gamma_{j,i}$ and for all transitions k distinct from i, j, $(\gamma_{i,k} = \gamma_{j,k} \wedge \gamma_{k,i} = \gamma_{k,j})$.*

Our next property states a similar result for an ordering relation instead of an equivalence. Assuming $i \neq j$, we say that i is before j, written $i \preceq_D j$, as follows:

$$i \preceq_D j \ =_{\text{def}} \ \gamma_{i,j} \leq \gamma_{j,i} \wedge (\forall k \neq i, j)(\gamma_{i,k} \leq \gamma_{j,k}) \quad (\preceq\text{-DEF})$$

LEMMA 5. *Assume (m, D) is a class obtained in the SCG construction and i, j are two transitions with the same static time interval ($I_s(i) = I_s(j)$). Then $i \preceq_D j$ implies $\alpha_i \leq \alpha_j$, $\beta_i \leq \beta_j$ and for all transitions k distinct from i, j, $\gamma_{k,j} \leq \gamma_{k,i}$.*

PROOF. By case analysis and induction on the firing sequence leading to (m, D), using Lemmas 2 and 3. □

A simple corollary of Lemma 5 is that the relation $\preceq_D$ is antisymmetric for every pair of equivalent transitions: we have that $i \preceq_D j$ and $j \preceq_D i$ implies $i =_D j$. Next, we show that the relation is also total. We can extend the $\preceq_D$ relation to the whole set of transitions (not only the enabled ones) by defining that $i \preceq_D j$ whenever i is not enabled.

LEMMA 6. *Assume (m, D) is a class obtained in the SCG construction and i, j are two transitions enabled at m with the same static time interval ($I_s(i) = I_s(j)$). Then either $i \preceq_D j$ or $j \preceq_D i$.*

PROOF. By case analysis. We have three cases to consider. The first is when both transitions i and j are newly-enabled in (m, D) (they were introduced simultaneously). It directly follows from Lemma 2. The second is when one is persistent and the other is newly-enabled. It directly follows from Lemmas 3 and 2. The last case is when both are persistent. It is proved by induction by showing that the ordering between i and j is preserved whatever the transition introduced next as long as both i and j stay persistent. □

4.3 A Total Order on Equivalent State Classes

Our results from Sect. 4.2 show that the relation $\preceq_D$ totally orders equivalent transitions; $\preceq_D$ has a quadratic time complexity.

We take advantage of this relation to derive an order between equivalent state classes. We say that a symmetry π is stable for a marking m (resp. for a class (m, D)) if $\pi(m) = m$. Let $C = (m, D)$ be some state class of the *SCG*. If π is stable for C, then the class $\pi(C) = (m, \pi(D))$ is also a class of the net and the firing domain $\pi(D)$ has the same "variables" as D, but each transition i in D is changed into $\pi(i)$.

By definition of the relation $\preceq_D$ (see equation $\preceq$-DEF), it is easy to see that $i \preceq_D j$ if and only if $\pi(i) \preceq_{\pi(D)} \pi(j)$. We use this property to define an order relation $\preceq_T$ between the firing domains obtained as the result of applying some symmetry to domain D (such domains will be said to have the form $\pi(D)$).

We assume an arbitrary, total ordering $\leq_T$ over the transitions in T. For any given permutation π stable for m we say that $D \preceq_T \pi(D)$ if and only if $D = \pi(D)$ (that is $t =_D \pi(t)$ for all t in $\mathcal{E}(m)$) or for the first transition t such that $t \neq_D \pi(t)$, we have $t \preceq_D \pi(t)$:

$$D \preceq_T \pi(D) \ =_{\text{def}} (D = \pi(D)) \vee (\exists t)\,(t \prec_D \pi(t) \wedge (\forall k \leq_T t)(k \preceq_D \pi(k)))$$

The relation $\preceq_T$ is analog to the lexicographic order built from $\leq_T$ and $\preceq_D$, but our definition has the advantage to work with any domains of the form $\pi(D)$. We can prove that $\preceq_T$ defines a total order between equivalent firing domains. This result derives from the observation that, by definition of $\preceq$, if $i \preceq_D \pi(i)$ and $\pi(i) \preceq_{\pi(D)} j$ then $i \preceq_D j$.

Next, assuming a total ordering $\leq_P$ over places P, we can easily build a total order $\preceq_P$ over markings.

From $\preceq_T$ and $\preceq_P$ we now define a total order $\preceq$ between equivalent classes. Two classes C and C' are equivalent if there is a symmetry π of the net such that $C' = \pi(C)$. We say that $(m, D) \preceq \pi(m, D)$ if $m \preceq_p \pi(m)$ or $m = \pi(m) \wedge D \preceq_T \pi(D)$ (that is $\preceq$ is a lexicographic order). We can observe that comparing firing domains is conceptually far more complex than comparing markings, because relation $\preceq_D$ is tied to a particular domain (and may change for different domains) whereas for markings we can merely use the comparison between integer tuples.

THEOREM 4.1. *Let $\mathcal{C} = (C_\sigma)_{\sigma \in T^*}$ be the set of classes in the SCG construction of a TPN. Then relation $\preceq$ is a total order between state classes equivalent for $\approx$ in $\mathcal{C}$.*

As such, the definition is not very helpful to compute the least representative state class for $\preceq$ (that is a suitable canonical form when testing equivalence). We address this issue in the next section.

5. AN IMPLEMENTATION

5.1 Describing symmetries

Our approach for expressing symmetries is structural. Nets are described hierarchically as compositions of smaller nets by synchronized products or free products. Symmetries are introduced by adhoc composition operators that build a net from a number of identical components and simultaneously define a symmetry relation among these components.

The two specialized composition operators currently provided are *Pool* and *Ring*. Both take an arity n and a TPN C as parameters: $Pool(n, C)$ stands for the free product of n independent copies of net C with the symmetries resulting from transposition of the pool components – $Ring(n, C)$ stands for the synchronized product of n copies of C after a renaming of transitions conventionally denoting synchronization of a ring element with its neighbors. The symmetries are those resulting from cyclic permutations of the ring components.

For example, the construction $(Ring(8, P) \mid Q \mid Pool(3, R))$ describes the TPN obtained by synchronizing a ring of 8 instances of P with Q and a pool of 3 instances of R. Such expressions simultaneously describe a TPN and a group of symmetries of the TPN, in terms of cyclic groups, symmetric groups, disjoint products of groups and wreath products

of groups, each attached with its generators. Such group constructions are discussed notably in [8, 10].

5.2 Computing canonical forms of classes

Our approach for checking state class equivalence relies on canonical forms, and specifically in computing the lexicographically least elements of their orbits [8]. Computing canonical forms for the symmetries retained in Sect. 5.1 can be done in polynomial time. Canonization of a state class $C = (m, D)$ is performed recursively from the group description derived for the net:

For *symmetric groups*, the components involved are the subnets of the TPN with their elements appearing in the generators of the group. By construction, these components are disjoint and isomorphic. Given total orders $\leq_P$ and $\leq_T$ on the places and transitions of the net, two components i and j of a pool can be ordered by relation $\leq_c$, as follows:

$$i \leq_c j =_{def} m^i \preceq_m m^j \ \vee \ (m^i = m^j \wedge e^i \preceq_e e^j)$$

where m^i is the tuple of markings of the places of i ordered by $\leq_P$; e^i is the tuple of transitions of component i, ordered by $\leq_T$; $\preceq_m$ is the lexicographic order on tuples of place markings (integers); $\preceq_e$ is the lexicographic order on tuples of transitions, themselves compared by the order $\preceq_D$ defined in Sect. 4.2.

The canonical form of class (m, D) is obtained as follows: the components are sorted by $\leq_c$, what defines a permutation of components and a corresponding symmetry π of the net. The canonical form of (m, D) is $\pi(m, D)$.

The treatment of *cyclic groups*, *disjoint* and *wreath* products is standard and follows [11].

We need two conditions on orders $\leq_P$ and $\leq_T$ to apply the above method:

- First, ordering $\leq_P$ must be such that the elements at same index in any two tuples m^i and m^j (from components i and j) are associated with places equivalent by $\approx$. Similarly, $\leq_T$ must be such that the transitions at same index in e^i and e^j are equivalent. This is a soundness condition ensuring that, when comparing component states, we only compare places or transitions that are equivalent by $\approx$;

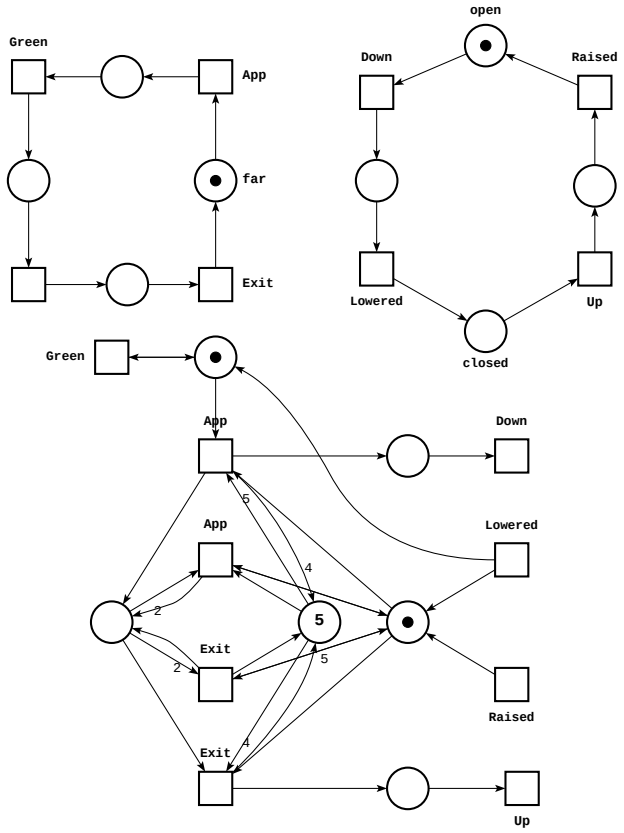
- Next, let us denote with p^i_k the place associated with element k of tuple m^i (that is the place m^i_k is the marking). Then, for any two component indices i and j, and any k, we must have $i \leq j \Rightarrow p^i_k \leq_P p^j_k$. Similarly for transitions, we must have $i \leq j \Rightarrow t^i_k \leq_T t^j_k$, where t^i_k is the transition the element e^i_k is associated with. These conditions guarantee that canonical classes are the lexicographically least in their orbits.

In our treatment, satisfaction of these constraints follows from the way places and transitions of components are named when building the net. These constraints being satisfied, it is easy to show that we have $i \leq_c j$ in the above treatment if and only if $C \preceq \pi(C)$ where $\preceq$ is the total order on equivalent classes defined in Sect. 4.3 and π is the symmetry of the net corresponding with the transposition of components i and j.

5.3 Computing experiments

The symmetry reduction method described above has been implemented in extension of *Tina*[1], an existing toolbox for analysis of Time Petri nets and various extensions [3]. We report here some experiments on a train-gate controller model, in both timed and untimed versions.

The timed version of the train-gate controller model is taken verbatim from [4]. The untimed version is a simplified version of a model found in [16]. In both cases the net models a level crossing: a number of tracks cross a road, protected by a gate; when approaching and leaving the road the trains trigger a signal sent to a controller that raises or lowers the gate as necessary. The safety property to be ensured is, of course, that the gate is closed whenever a train is crossing the road. In the timed version, this is ensured by time constraints on the relevant events, while the untimed versions makes use of signal acknowledgments and a shared green flag. In each case, the model is obtained by synchronizing a gate model (right), a controller (middle) and a pool of tracks. The timed model is represented in Figure 2 and the untimed one in Figure 1.

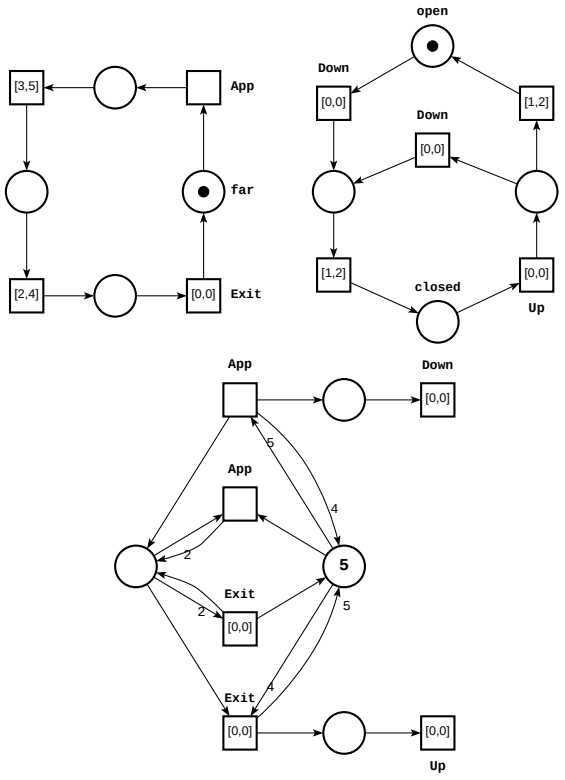


tracks		*R (no sym.)*	*R≈ (our)*	*R≈ (lola)*
5	*size*	1036	60	92
	time	0.00	0.00	0.01
10		1048598	290	18836
		14.40	0.00	0.67
20		1.099×10^{12}	1775	303830981
		–	0.13	76221
50		1.267×10^{30}	23430	–
		–	9.42	–
100		1.606×10^{60}	176855	–
		–	225.18	–

Figure 1: Untimed level crossing example.

The results on the untimed models are shown in Figure

[1]Tina is available at `http://www.laas.fr/tina`, its experimental extension supporting symmetries is available at `http://www.laas.fr/tina/symmetries`.



tracks		SCG	$SCG_{\approx}$	$SCG^{\subseteq}$	$SCG^{\subseteq}_{\approx}$
3		3101	578	172	41
		0.02	578	0.00	0.00
4	*size*	134501	6453	1175	76
	time	1.67	0.15	0.02	0.00
5		8557621	84510	10972	143
		179.33	2.61	0.38	0.01
6		697913229	1183782	128115	274
		24346.77	46.95	8.37	0.02
7		7.278×10^{10}	18143796	1772722	533
		–	1060.21	614.38	0.07
8		9.262×10^{12}	297205635	28208543	1048
		–	25105.87	177602.90	0.16
10		–	–	–	4126
		–	–	–	1.10
12		–	–	–	16420
		–	–	–	8.29
14		–	–	–	65578
		–	–	–	95.07
16		–	–	–	262192
		–	–	–	1418.38
18		–	–	–	1048630
		–	–	–	22407.25

Figure 2: Timed level crossing example.

1. Columns R holds the sizes of reachability sets, omitting symmetries, and their computing times, for the models with the number of tracks specified in the first column. Column $R_{\approx}$ (*our*) shows the sizes and computing time of the symmetry reduced reachability sets computed by our tool. As can be observed, we obtain the expected gains in size, as well as considerable gains in computing times. The numbers in grey font have been obtained "analytically" by summing the sizes of orbits of the symmetry reduced markings; building spaces of such sizes are beyond the capabilities of our tool, which is based on enumerative approaches.

The last column shows the results of computing the symmetry reduced reachability sets by the latest available version of tool LoLA (V2.0), which relies upon the techniques of [20]. As can be seen, LoLA typically computes several representatives per orbit. As a consequence, the gain on the number of states obtained by symmetry reduction is much smaller than what we obtain with our treatment. On the other hand, LoLA can extract more symmetries from nets than we are currently able to express, and computes symmetries automatically from bare nets. We were not able to compare our results with those of [14].

The results on the timed models are shown in the table of Figure 2, for two state class graph constructions. As in the untimed case, the grey numbers have been obtained by summing the sizes of orbits of the symmetry reduced classes.

For the same number of tracks, the number of state classes of the timed model (columns SCG and $SCG_{\approx}$) is much larger than the number of markings in the untimed models. The gain obtained by symmetry reduction of state classes is however similar to that obtained for markings in the untimed version, both in terms of size of state class graphs and computing times. It can also be observed that the coarser construction of "state classes under inclusion" (columns $SCG^{\subseteq}$ and $SCG^{\subseteq}_{\approx}$), only preserving markings (cf. Sect. 2) benefits of similar gains.

These experiments, with many other conducted over the last months, confirm that reduction by symmetry, when applicable, is an effective way of fighting combinatorial explosion. They are even more important for real-time models since alternative abstractions like partial order methods, unfolding methods or symbolic methods appear more difficult to apply on timed systems than on untimed ones. Symmetry reduction allows one to perform verification of TPN faster and with much less computing resources (a laptop may suffice where a large server was required).

6. RELATED WORK AND CONCLUSION

In the context of Petri nets, symmetry reduction methods have been mostly applied to Colored Petri Nets (and the problem of symmetries on data values). The problem that we address in this paper is quite different. Concerning symmetries for real-time models, the only works we are aware of are on Uppaal [12] and RED [24]. The latter combines symbolic exploration with symmetries and rely on over-approximations of the state space; so it uses different methods than those discussed here. The treatment of symmetries for Timed Automata [12] is closer in spirit to our work.

We developed a symmetry reduction method for the State Class Graph construction of Time Petri nets what, as far as we know, has never been attempted. As appears from Sect.

4, the technical treatment differs significantly from that of [12]. Actually the difference is not surprising since (going beyond the difference between the models) the abstraction method for Timed Automata is based on clock domains—that is on the time elapsed since a transition fired—rather that on firing domains—that capture the time, in the future, when a transition can fire. For TPN, the firing domain approach is more interesting in practice since it yields smaller abstractions.

We have several opportunities for extending our work. The pragmatic way of defining nets together with their symmetries in Section 5 is sufficient in many cases. However the version presented forbids interactions between components of a pool, which can be sometimes limiting. Relying on a form of symbolic transition labels, we could relax that restriction but this may come at the cost of losing the canonical property of orbit representatives, depending on the synchronization pattern between pool components. These results will be reported in another document. We would also like to add new forms of symmetries, such as the dihedral or cube symmetries. Finally, combining our treatment with that of LoLA (for the symmetries we cannot express) is also investigated.

7. REFERENCES

[1] 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.

[2] B. Berthomieu and M. Menasche. An Enumerative Approach for Analyzing Time Petri Nets. In *Information processing 83: proceedings of the IFIP 9th World Computer Congress*, pages 41–46. Elsevier Science Publ. Comp. (North Holland), 1983.

[3] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research*, 42(14):2741–2756, July 2004.

[4] B. Berthomieu and F. Vernadat. State Class Constructions for Branching Analysis of Time Petri Nets. In *Tools and Algorithms for the Construction and Analysis of Systems 2003*, pages 442–457. Springer LNCS 2619, 2003.

[5] B. Berthomieu and F. Vernadat. *State Space Abstractions for Time Petri Nets*. Handbook of Real-Time and Embedded Systems, Ed. Insup Lee, Joseph Y-T. Leung and Sang Son, CRC Press, Boca Raton, FL., U.S.A., 2007.

[6] H. Boucheneb and J. Mullins. Analyse des Réseaux Temporels: Calcul des Classes en O(n[2]) et des Temps de Chemin en O(m×n). *TSI. Technique et science informatiques*, 22(4):435–459, 2003.

[7] G. Chiola. Manual and Automatic Exploitation of Symmetries in SPN Models. In *Application and Theory of Petri Nets*, pages 28–43. Springer LNCS 1420, 1998.

[8] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry Reductions in Model Checking. In *Computer Aided Verification*, pages 147–158. Springer LNCS 1427, 1998.

[9] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. Exploiting Symmetry in Temporal Logic Model Checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[10] A. Donaldson and A. Miller. Exact and Approximate Strategies for Symmetry Reduction in Model Checking. In *FM 2006: Formal Methods*, pages 541–556. Springer LNCS 4085, 2006.

[11] A. F. Donaldson and A. Miller. On the Constructive Orbit Problem. *Annals of mathematics and artificial intelligence*, 57(1):1–35, 2009.

[12] M. Hendriks, G.Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding Symmetry Reduction to Uppaal. In *Formal Modeling and Analysis of Timed Systems*, pages 46–59, 2003.

[13] C. N. Ip and D. L. Dill. Verifying Systems with Replicated Components in Murφ. *Formal Methods in System Design*, 14(3):273–310, 1999.

[14] T. A. Junttila. New Canonical Representative Marking Algorithms for Place/Transition-Nets. In *Applications and Theory of Petri Nets*. Springer LNCS 3099, 2004.

[15] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. Irvine: Univ. California, Depart. of Information and Computer Sience, TR 58, 1974.

[16] F. Pommereau. *Algebras of coloured Petri nets*. Habilitation thesis, University Paris-East, Créteil, 11 2009.

[17] G. Ramalingam, J. Song, L. Joscovicz, and R. E. Miller. Solving Difference Constraints Incrementally. *Algorithmica*, 23, 1995.

[18] K. Schmidt. How to Calculate Symmetries of Petri Nets. *Acta Informatica*, 36(7):545–590, Jan. 2000.

[19] K. Schmidt. Integrating Low Level Symmetries into Reachability Analysis. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 315–330, 2000.

[20] K. Schmidt. LOLA A Low Level Analyser. In *Application and Theory of Petri Nets*, pages 465–474. Springer LNCS 1825, 2000.

[21] A. P. Sistla, V. Gyuris, and E. A. Emerson. SMC: A Symmetry-Based Model Checker for Verification of Safety and Liveness Properties. *ACM Transactions on Software Engineering and Methodology*, 9(2):133–166, 2000.

[22] P. H. Starke. Reachability Analysis of Petri Nets using Symmetries. *Systems Analysis Modelling Simulation*, 8(4-5):293–303, 1991.

[23] P. H. Starke and S. Roch. INA: Integrated Net Analyzer. *Reference Manual*, 1992.

[24] F. Wang and K. Schmidt. Symmetric Symbolic Safety-Analysis of Concurrent Software with Pointer Data Structures. In *Formal Techniques for Networked and Distributed Systems*, pages 50–64. Springer LNCS 2529, 2002.