Symmetry reduction for time Petri net state classes

Pierre-Alain Bourdil, Bernard Berthomieu, Silvano Dal Zilio, François Vernadat

Symmetry Reduction for Time Petri Net State Classes

Pierre-Alain Bourdil[a,c,1], Bernard Berthomieu[a,*], Silvano Dal Zilio[a], François Vernadat[a,b]

[a] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[b] *LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France*
[c] *Eurogiciel Ingénierie, 417 Avenue l'Occitane, 31670 Labege, France*

Abstract

We propose a method to exploit the symmetries of a real-time 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, a construction providing a finite representation of the behavior of a Time Petri net preserving its markings and traces. The approach has been implemented and experiments are reported.

Keywords: Time Petri Nets, State Classes, Symmetry Reduction

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 studies in program verification and analysis of 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. [20, 12, 30, 29]. Symmetry reduction can be further combined with other reduction techniques, such as stubborn sets or covering steps. Combining symmetries with symbolic analysis based on decision diagrams has been investigated for some models [14] but, in spite of some results, they seem harder to accommodate with such approaches than with enumerative methods.

*Corresponding author
Email addresses: Pierre-Alain.Bourdil@eurogiciel.fr (Pierre-Alain Bourdil), Bernard.Berthomieu@laas.fr (Bernard Berthomieu), Silvano.Dalzili@laas.fr (Silvano Dal Zilio), Francois.Vernadat@laas.fr (François Vernadat)
[1] On leave from Thales Avionics, 105 av du Général Eisenhower, F-31100 Toulouse, France.

In this paper, we propose a symmetry reduction method for a state space abstraction technique for real-time models with dense time semantics, the State Class Graph (SCG) construction for Time Petri nets [3, 2].

Time Petri nets [23] (TPN for short) are a widely used model for analysis of real-time systems. Time Petri nets enrich Petri nets with time intervals associated with the transitions of the net. These intervals specify the possible time delays between last enabledness of the transitions and their activation. Time Petri nets have been used for a variety of verification tasks, including model checking, scheduling analysis or evaluation [11, 33, 22] in a variety of application domains like communication protocols, real-time systems, biochemical networks, etc. Compared to the prominent model of Timed Automata, bounded TPN have the same expressiveness in terms of timed language acceptance, but are less expressive in terms of timed bisimulation [1], unless enriched with priorities [4]. They benefit of well established analysis methods, based for most on the state class construction of [3], and of several mature tools [5, 16, 10].

TPN can be interpreted over discrete time or over dense time. While symmetry reduction of discrete Time Petri nets can be seen as a straightforward extension of symmetry reduction of (untimed) Petri nets (it was available in the INA tool [32]), symmetry reduction for dense time Time Petri nets is more challenging. When interpreted over dense time, state spaces are typically infinite and finite representations are obtained through time abstractions. The time information is typically represented by systems of difference constraints with their variables associated with the transitions of the net. Some results are available on symmetry reduction for Timed Automata [19, 35], but none is available so far 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, known as the state classes construction. First, we make some technical contributions, with the definition of a total order relation between symmetry equivalent transitions (Section 5) that relies on an invariant on the systems of difference constraints capturing time information. A second, more practical, contribution is a high-level structural approach for declaring the symmetries of a net (see Section 4) in a way enabling efficient (polynomial) computation of canonical representatives for state classes, using the previous ordering.

Outline of the paper. We start with some information on Time Petri nets and their analysis techniques. In Section 3, we extend the symmetry reduction approach for Petri nets to dense-time Time Petri nets. Our approach to symmetry reduction of TPN is presented in Section 4; it describes a compositional method to build large nets from smaller components, together with their symmetries, associated with a method for computing state class representatives. The main technical result of the paper is found in Section 5, a total order between symmetry equivalent state classes. Section 6 presents a preliminary implementation of our method integrated into the Tina toolbox [5] and some experimental results. Some possible extensions of this work are discussed in the conclusion.

2. Time Petri nets

A *Time Petri net* [23] (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 their endpoints in $\mathbb{N} \cup \{\infty\}$. 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 $A(k) = k \neq t \wedge k \in \mathcal{E}(m - \mathbf{Pre}(t))$

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

5) $(\forall k \in T)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} \theta)$

The state transitions labeled over T (case (i) above) are the *discrete* transitions, those labeled 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 ($0 \in I(t)$). 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 gives a *dense time semantics* to Time Petri nets. TPN can also be given a *discrete time semantics* by enforcing $\theta \in \mathbb{N}$ in continuous 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 the state graph *SG* that abstracts time elapsing transitions but preserves its markings and traces; the *SCG* embeds the reachability graph of the TPN.

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 [3, 2] 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* [3]).
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).*

The firing domains obtained by Algorithm 1 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 $x_i - x_j \leq \gamma_{i,j}$, where α_i, β_i and $\gamma_{i,j}$ are bounds, associating a rational constant with a comparison operator in $\{\leq, <, \geq, >\}$. Such systems, often represented by Difference Bound Matrices, admit canonical forms that can be computed in polynomial time (see e.g. [26]).

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 [3, 2] and so is suitable for Linear Time Temporal Logic (LTL) model checking.

If only marking reachability properties are of interest, instead of the *SCG* one may use a typically coarser abstraction referred to here as $SCG^{\subseteq}$ [7]. 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 1 are straightforwardly adapted from [31, 27].

Definition 2. *A* symmetry *of a TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ 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 TPN $\mathcal{N}$ 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 [31] (Lemma 1). For (3) and (4), similarly, this is straightforward from the definitions of actions.
2. From [31] 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 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 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 model (the TPN here) 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 [27]. For this reason, many implementations of symmetry reduction rely on some static symmetry information provided by the user. Murφ [20], for instance, makes use of a dedicated "scalarset" type, also used in [19]. In high level Petri nets, symmetries are deduced from the syntax of inscriptions. On the other hand, some tools [29] compute these automorphisms from nets automatically with acceptable performances on average.

In our implementation, described in Sections 4 and 6, nets are 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 [27, 28], 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 [21] builds upon these algorithms and proposes some improvements. The experiments discussed in [21] suggest that the third algorithm typically yields the best results.

On the other hand, the methods relying on scalarsets [20, 19] or similar structural techniques trade generality for efficiency. Scalarsets may only express particular groups of symmetries, typically full symmetries in systems of

processes (those resulting from all process permutations). As a consequence, canonical representatives can be efficiently computed for states.

Among the methods proposed in [27, 28], the first two would be applicable to state classes as well since they only require to be able to compare states for equality; this is yet unclear for the third method which relies on a total ordering of states. But these methods do not obviously lead to efficient implementations; this is substantiated by our experimental data in Section 6. Though technically different, our reduction technique is closer to those relying on scalarsets; the symmetries are declared, and restricted to those admitting a polynomial time algorithm for computing canonical representatives (see e.g. [15]). The approach is presented in the next section.

4. Describing symmetries compositionally

Large systems are typically built by assembling components, possibly replicated. Replication typically induces symmetries of their state spaces [24, 34]. We wish to take advantage of the structure of a net, made explicit by construction, to compute and handle the symmetries of its state space.

In our approach, nets are described hierarchically as compositions of smaller nets. Symmetries are introduced by adhoc composition operators. Each builds a net from a replicated component and simultaneously specifies a component symmetry. The approach is presented in this section and illustrated in Section 6.

4.1. Decomposition and composition of TPN

Petri nets based models admit several natural decomposition into subnets. Nets can be decomposed from a partition of their places, or of their transitions, or of their arcs. We use the first form; from a partition of places: the subnet, or component, defined by block b of the partition is constituted of the places in b, their adjacent arcs, and the transitions connected to these arcs. Transitions may occur in several components (with their **Pre** and **Post** functions spread among these) while each place may only occur in a single component;

The corresponding composition operator is the synchronized product of nets, reviewed in the sequel. Synchronized products are pervasive in the study of formal models for concurrency (e.g. Process calculi, networks of automata, etc); for Petri nets, it was introduced in [17]. For easing the specification of synchronizations among components, the transitions of nets are assumed to be labeled over an alphabet of actions $\Sigma^\epsilon = \Sigma \cup \{\epsilon\}$, with $\epsilon \notin \Sigma$. Σ is the set of *visible*, or *observable*, actions; ϵ is the *silent*, or *internal*, action.

Definition 3 (Labeled Petri net). *A labeled Petri net is a tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0, \Sigma^\epsilon, L\rangle$ *in which:*

- $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0\rangle$ *is a Petri net,*
- $\Sigma^\epsilon = \Sigma \cup \{\epsilon\}$ *is a finite alphabet of actions, with* $\epsilon \notin \Sigma$.
- $L : T \to \Sigma^\epsilon$ *is a function called the* Labeling *function.*

Definition 4 (Synchronized product). *Consider two labeled Petri nets not sharing any place or transition:* $\mathcal{N}_1 = \langle P_1, T_1, \mathbf{Pre}_1, \mathbf{Post}_1, m_1^0, \Sigma_1^\epsilon, L_1 \rangle$ *and* $\mathcal{N}_2 = \langle P_2, T_2, \mathbf{Pre}_2, \mathbf{Post}_2, m_2^0, \Sigma_2^\epsilon, L_2 \rangle$*. Their synchronized product* $\mathcal{N}_1 \mid \mathcal{N}_2$ *is the net* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0, \Sigma^\epsilon, L \rangle$ *obtained as follows:*

- $P = P_1 \cup P_2$
- $m^0(p) =$ **if** $p \in P_1$ **then** $m_1^0(p)$ **else** $m_2^0(p)$
- $\Sigma^\epsilon = \Sigma_1^\epsilon \cup \Sigma_2^\epsilon$
- *For each* $i \in \{1, 2\}$*,* T *contains all transitions* $t \in T_i$ *such that* $L_i(t) \notin \Sigma_1 \cap \Sigma_2$*, with:*

  For each $p \in P$*:*

  $\mathbf{Pre}(t)(p) =$ **if** $p \in P_i$ **then** $\mathbf{Pre}_i(t)(p)$ **else** 0

  $\mathbf{Post}(t)(p) =$ **if** $p \in P_i$ **then** $\mathbf{Post}_i(t)(p)$ **else** 0

  $L(t) = L_i(t)$

  For each $(t_1, t_2) \in T_1 \times T_2$ *such that* $L_1(t_1) = L_2(t_2) \neq \epsilon$*,* T *contains a transition, referred to as* (t_1, t_2)*, with:*

  For each $p \in P$*:*

  $\mathbf{Pre}((t_1, t_2))(p) =$ **if** $p \in P_1$ **then** $\mathbf{Pre}_1(t_1)(p)$ **else** $\mathbf{Pre}_2(t_2)(p)$

  $\mathbf{Post}((t_1, t_2))(p) =$ **if** $p \in P_1$ **then** $\mathbf{Post}_1(t_1)(p)$ **else** $\mathbf{Post}_2(t_2)(p)$

  $L((t_1, t_2)) = L_1(t_1)$

  T *contains no other transitions than those specified above.*

Product $\mid$ is commutative and associative; it is also compositional (the semantics of a product of nets is obtained from the semantics of its constituents). It can be extended to TPN [4].

A language for building nets from components. A net with no particular structure made explicit, holding at least a place and its connected edges and transitions, or a single transition, is called a *basic* net. Our nets will be built, hierarchically, as synchronized products (denoted $\mid$) from basic nets. For convenience, a relabeling operator allowing to hide or change the label of some transitions may be provided (as in the CCS process calculus, for instance), as well as derived product forms like free products or iterated products replicating a component before composing the copies.

Such a language of net expressions has been available for a long time in the tool Tina [5]. Examples of nets built this way will be given in Section 6. We discuss now the introduction of symmetries in our language of components.

4.2. Describing nets together with their symmetries

Nets built from replicated components typically exhibit symmetries. Consistently with our structural description of nets, we want to declare component symmetries when building nets rather than computing net symmetries afterwards. This will be done by enriching our language of net operators by specific operators for the most common kind of symmetries (full symmetries, circular symmetries, cube symmetries, for instance). These operators will work as iterators replicating a component, and simultaneously enforcing a symmetry among the copies, resulting in a net together with a description of its symmetries.

The approach is discussed in [9] in a general setting. In this paper, and for the purpose of illustrating our symmetry reduction method for Time Petri nets, we will use here a simplified version. The simplification consists of limiting the symmetry declarations to full symmetries and cyclic symmetries and, for full symmetries, preventing synchronizations between the components involved. An additional restriction on products will be made precise shortly.

The previous net expressions are enriched with two operators: $Pool$ and $Ring$. Both take an arity n and a net C as parameters: $Pool(n, C)$ stands for the free product of n independent copies of net C with the symmetries resulting from transpositions of the pool components – $Ring(n, C)$ stands for the synchronized product of n copies of C after a conventional renaming of transitions ensuring synchronization of a ring element with its neighbors. The symmetries are those resulting from cyclic permutations of the ring components.

Our enriched language of net expressions is thus ($\|$ stands for the free product and $\mathbf{\Theta}$ is an unspecified family of relabeling operators):

$$N ::= B \mid N|N \mid N\|N \mid N\mathbf{\Theta} \mid \textbf{\textit{Pool}}\ (n, N) \mid \textbf{\textit{Ring}}\ (n, N)$$

For example, the expression ($\textbf{\textit{Ring}}(8, P)$ | Q | $\textbf{\textit{Pool}}(3, R)$) denotes the net obtained by synchronizing a ring of 8 instances of component P with component Q and a pool of 3 instances of component R.

Such expressions simultaneously describe a net and its group of symmetries, compositionally. In synchronized products of nets, we will assume the additional restriction, easily checked statically, that one transition at least in each pair of transitions synchronized must be invariant under the action of the symmetries of its component. Then, the group described can be built in terms of cyclic groups, symmetric groups, disjoint products of groups and wreath products of groups. These group constructions all belong to the so-called easy groups [13]; computing canonical representatives for their elements can be done in polynomial time.

Compared to the treatment of Lola [29], for instance, our nets will only exhibit the symmetries declared, while Lola will compute all symmetries. But Lola fails to capture the structure of the group of symmetries, which prevents application of efficient reduction techniques known for some particular groups. The benefits of the knowledge of the group structure will be clear from the experiments presented in Section 6, which include a comparison of our reduction results with those obtained by Lola.

4.3. Checking marking equivalence in Petri nets

Our approach for checking state equivalence relies on canonical representatives for states, and specifically in computing the lexicographically least elements of state orbits [13]. For the Petri nets described as in Section 4.2, we could proceed as follows; the treatment is extended to Time Petri nets afterward.

The states of a Petri net are its markings; transitions express state changes but do not contribute themselves to the state information. Computing canonical representatives for marking, modulo the symmetries considered, can be done in polynomial time; canonization of a marking m is performed recursively from the group description derived for the net:

– For *symmetric groups* (pools): By construction, components are disjoint (do not interact) and isomorphic since obtained by replication. All place and transition orbits are of size n (the number of components involved in the pool). Clearly then, the pool marking m can be represented by a matrix in which column i holds the markings of the places in component i and line j holds the markings of all places in some place orbit j.

Then, the canonical representative for marking m represented this way is obtained by sorting its columns lexicographically (that is by sorting the local states [13]). The component permutation that sorts the columns defines a symmetry π of the net; the canonical representative of m is $\pi(m)$. One can always find an ordering on the places of the net such that the representatives are the least markings in their orbits.

– For *cyclic groups* (rings): Even though the ring components interact, they are still isomorphic, so we can use for markings the same matrix representation used for symmetric groups. Then, a cyclic permutation ρ of components (of the columns of the matrix M representing marking m) is computed that minimizes lexicographically the vector of columns of matrix M. The canonical representative of m is $\pi(m)$, where π is the symmetry of the net corresponding with component permutation ρ.

– For products: Wreath products appear when a net with some symmetries declared is iterated in a pool or ring construction. With the hypotheses of Section 4.2, the symmetries of a product of nets, whether free or synchronized, results in a disjoint product of groups. Classically [13], canonical representatives for elements of such groups are built from the canonical representatives of their constituents.

4.4. Checking state class equivalence in Time Petri nets

Time Petri nets are built using the same language of net expressions presented in Section 4.2. The information in state classes is richer than that captured by markings in Petri nets; states here are pairs (m, D), where m is a marking and D is a firing domain. Firing domains can be classically represented by so-called Difference Bound Matrices (DBM for short). The DBM of class (m, D) has size $(n + 1) \times (n + 1)$, where n is the number of transitions enabled at marking m.

A naive way to compute the canonical representative of a state class (m, D) would be to compute first a canonical representative for the marking and then

one for the firing domain, represented by a DBM. For the marking, one can proceed as explained in Section 4.3. For the DBM, the problem is discussed in [18] for the clock DBM of Timed Automata: DBM can be seen as adjacency matrices of some complete labeled digraphs; sorting DBM amounts to sort such graphs, a problem known to be equivalent to the graph isomorphism problem. So there is no hope to obtain an efficient canonization of state classes along this path.

Fortunately, we can abstract the information conveyed by firing domains in a way that allows computing representatives for state classes by a simple extension of the method used in Section 4.3 for markings. The method relies on a total order $\preceq_D$ between transitions that are equivalent by symmetry (are in the same orbit) and enabled at state class (m, D). A suitable order is developed in the next section; the idea is inspired by [19] in which clocks of Timed Automata zones are compared using their last reset date. The handling of time in TA is quite different from that in TPN, however, and the zone constructions of TA significantly differ from the state class constructions of TPN, so we cannot reuse the clock order of [19] and had to investigate a specific one for state classes.

Computing canonical representative for state classes. Given such an order, $\preceq_D$, one can compute the canonical representative of class (m, D) as follows.

Recall that, in Section 4.3, markings were presented as matrices M such that column i is the local marking of component i and line j holds the marking of all the places in orbit j. Similarly, and for the same reasons (the components of a pool or ring are isomorphic), the set e of transitions enabled at some marking m (the time information in a state class is associated with these transitions only) can always be represented by a matrix E in which column i holds the transitions of component i, and all transitions in line j are in the same transition orbit.

Then, instead of simply comparing the columns of M as in Section 4.2, we compare lexicographically pairs (m^i, e^i) in which m^i (resp. e^i) is the i^{th} column of M (resp. E). If the marking elements differ then we are done, otherwise we must compare the transition elements; these are compared with the lexicographic order on $\preceq_D$. Sorting a state class (m, D) abstracted this way yields a permutation π of the net; the canonical representative of the class is $\pi(m, D)$.

We now focus on the development of the total order $\preceq_D$ for a class (m, D).

5. Ordering State Classes

First, for any state class (m, D), we define a total order relation $\preceq_D$ between transitions enabled at m and equivalent by symmetry ($\approx$). Afterwards, this relation is used 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 Section 5.2.

5.1. 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 [8].

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 $i, j \in \mathcal{E}(m')$:*

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

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.

5.2. 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 (symmetries must preserve static intervals).

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:

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

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. (full proof in Appendix) Lemma 5 is pictured in Figure 1 on a domain represented by a DBM. It says that if i and j have equal static intervals then the circled relations imply the boxed relations. The proof is 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. (full proof in Appendix) 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.

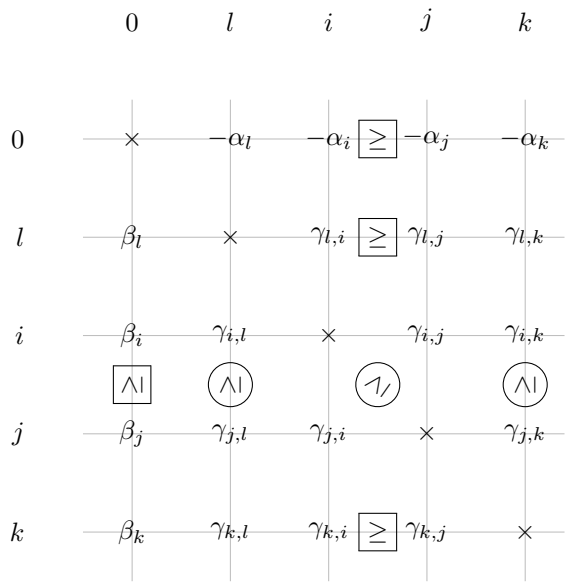


Figure 1: Illustration of Lemma 5: (≥) imply [≥].

5.3. A Total Order on Equivalent State Classes

Our results from Section 5.2 show that the relation $\preceq_D$ totally orders equivalent transitions. Checking $i \preceq_D j$ at a state class (m, D) has quadratic time complexity in the number of transitions enabled at m.

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 $t \preceq_D \pi^{-1}(t)$:

$$D \preceq_T \pi(D) \;=_{\text{def}} (D = \pi(D)) \vee (\exists t)\left(t \prec_D \pi^{-1}(t) \wedge (\forall k <_T t)(k =_D \pi(k))\right)$$

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 $\pi(D)$. Relation $\preceq_T$ is a total order between equivalent firing domains.

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 whereas for markings we can merely use the comparison between integer tuples.

Theorem 2. *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 $\mathcal{C}$.*

We have been very careful in our definitions of $\preceq_T$ and $\preceq$ to avoid any reference to a particular implementation of symmetries. Hence our results can hold for different design choices.

6. An implementation

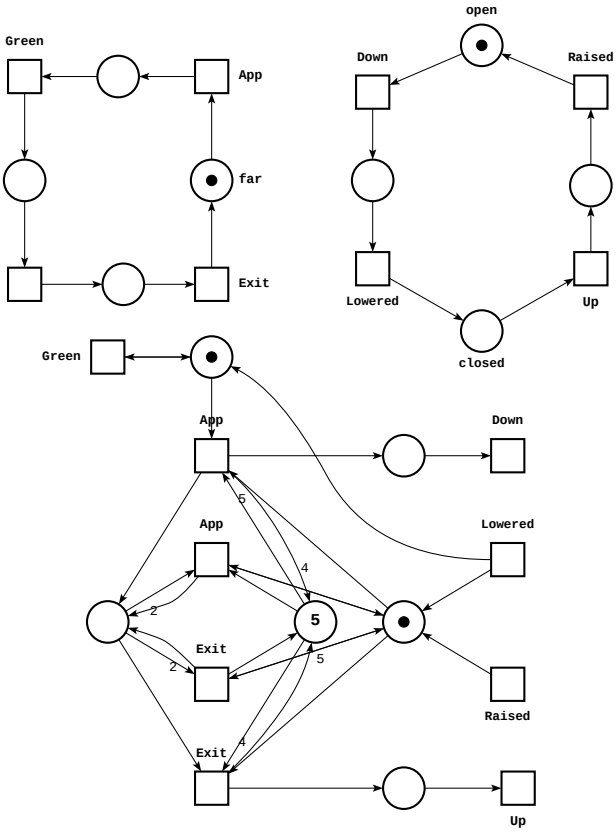
6.1. Computing experiments

The symmetry reduction method described in Section 4, relying on the ordering defined Section 5 when applied to TPN, has been implemented in an extension of *Tina*[2], an existing toolbox for analysis of Time Petri nets and various extensions [5].

We report some experiments on a train-gate controller model, in both timed and untimed versions, and on an example with compound symmetries. The nets are built with the language of net expressions with symmetries introduced in Section 4.2. Canonical representatives for markings in the untimed examples and for state classes in the timed examples are computed exactly as explained in Sections 4.3 and 4.4, respectively.

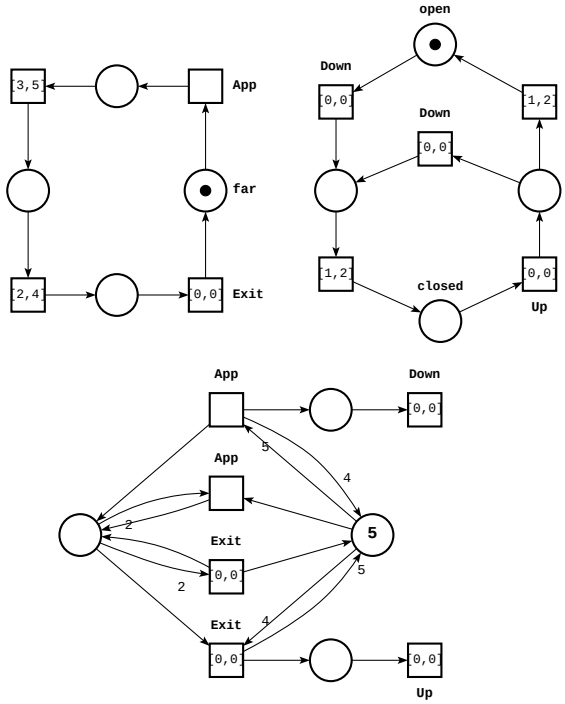
The timed version of the train-gate controller model is taken verbatim from [6]. The untimed version is a slightly simplified version of a model found in [25]. In both cases the net models a level crossing: a number of tracks cross a road, protected by a gate; when approaching and leaving the road the 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 3 and the untimed one in Figure 2. The unit of time in all tables is the second of CPU time on a typical workstation.

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



tracks		R (*no sym.*)	$R_{\approx}$ (*our*)	$R_{\approx}$ (*lola*)
5	*size*	1036	60	92
	time (*s*)	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 2: Untimed level crossing example.



tracks		SCG	$SCG_{\approx}$	$SCG^{\subseteq}$	$SCG^{\subseteq}_{\approx}$
3	*size*	3101	578	172	41
	time (s)	0.02	578	0.00	0.00
4		134501	6453	1175	76
		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 3: Timed level crossing example.

The results on the untimed models are shown in Figure 2. 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 as 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 [29]. 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 able to express, and computes symmetries automatically from bare nets. We have not been able to compare our results with those of [21].

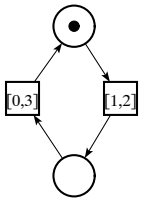
The results on the timed models are shown in the table of Figure 3, for two state class graph constructions. As in the untimed case, the grey numbers have been obtained by summing the sizes of orbits of the symmetry reduced classes.

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 model. 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 spaces 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. Section 2) benefits of similar gains.

We conclude our experiments with a timed example featuring embedded symmetries. The TPN considered is made of 6 copies of the net represented Figure 4, with various symmetry declarations specified in the left column.

Omitting time information, the nets have factorized marking graphs of the sizes indicated in column $R_{\approx}$. Columns $SCG_{\approx}$ and $SCG^{\subseteq}_{\approx}$ show the effects of symmetry reduction on the graphs of state classes for the various symmetry descriptions. The last line shows the figures when no symmetries are specified: omitting time information, the net has then 64 markings and 384 transitions; taking time information into account it admits 1 973 488 state classes and 11 285 976 transition(s) for the standard SCG construction and 1957 classes for the coarser $SCG^{\subseteq}$ construction,

These experiments, and many others 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 logic-based symbolic methods appear more difficult to apply on such models.



symmetry description		$R_{\approx}$ markings	$SCG_{\approx}$ state classes	$SCG_{\approx}^{\subseteq}$ state classes
$Pool(6, N)$	*size*	7	5404	7
	time (s)	0.004	0.36	0.004
$Pool(3, N)$ || $Pool(3, N)$		16	72234	69
		0.0	5.36	0.02
$Pool(2, Pool(3, N))$		10	36154	35
		0.0	2.86	0.01
$Ring(6, N)$		14	328984	327
		0.01	25.71	0.08
$Ring(3, N)$ || $Ring(3, N)$		16	221600	225
		0.0	17.75	0.06
$Ring(2, Ring(3, N))$		10	110860	113
		0.0	9.72	0.03
$Ring(3, N)$ || $Pool(3, N)$		16	126334	124
		0.0	9.82	0.03
$Ring(2, Pool(3, N))$		10	36154	35
		0.0	2.91	0.01
$Pool(2, Ring(3, N))$		10	110860	113
		0.0	9.21	0.03
$N\|\|N\|\|N\|\|N\|\|N$		64	1973488	1957
		0.0	92.70	0.54

Figure 4: Component N and example symmetries.

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).

6.2. Towards richer symmetries

The preliminary implementation presented in the previous section forbids interactions between components of a pool, which can be limiting. That restriction can be relaxed: [9] introduces a construction that given a group specification on component identifiers, an arity n, a component TPN N and an interaction pattern between components, builds the net resulting from the synchronization of n copies of N following the interaction pattern, with the symmetries arising from the group specification on component identifiers.

Our component language for nets can be enriched to handle such constructions. This would allow one to build richer models, such as the well known Fischer protocol in which all components are in a full symmetry, but interact with each others though a lock variable. However, if pool components are allowed to interact, then it would not be true in general that the length of all transition orbits are equal to the number of components in the pool, which prevents to use the sorting method of Section 4.4 to compute canonical representatives. Alternative, sub-optimal, methods could be used though. Note that, since the lock variable in that protocol holds a process identifier, the Timed Automata model of the Fischer protocol in [19] does not fulfill either the conditions ensuring that the canonization algorithm used in that work produces canonical representatives.

7. 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 [19] and RED [35]. The latter combines logic-based symbolic exploration with symmetries and relies on over-approximations of the state space; so it uses different methods than those discussed here. The treatment of symmetries for Timed Automata [19] is closer in spirit to our work.

We developed a symmetry reduction method for the State Class Graph construction of Time Petri nets. That, as far as we know, had never been attempted. As appears from Section 5, the technical treatment differs significantly from that of [19]. 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. Though not discussed in this paper, our symmetry reduction approach can be applied as well to the alternative *Strong SCG* construction introduced in [6], based on clock domains rather

than firing domains. This abstraction yields larger state spaces in general but is required for handling some extensions of TPN, like priorities. The ordering required for the *Strong SCG* construction is significantly simpler than that derived in Section 5 for the *SCG* and, as one may expect, close to the ordering used in [19].

We have several opportunities for extending our work. The pragmatic reduction approach proposed in Section 4 is useful in many cases, but forbidding interactions between components of a pool is limiting. As mentioned in Section 6.2, we could relax this restriction in our component language, but this may prevent to compute canonical representatives with the optimal algorithm discussed in Section 4.

There is room in our method, however, for integrating alternative algorithms for computing representatives if required by some component. In our treatment, the canonization algorithm used for pool components could depend on the features of the net described, using the optimal algorithm of Section 4 if components do not interact, or some sub-optimal algorithm when they do.

In the same vein, we could add to our language of components other operators, for other commonly found symmetries like dihedral or cube symmetries, each handled with their own algorithm for computing representatives. Finally, it is tempting to investigate some combination of the techniques of [28] with our. For components in which symmetries are not easily expressed in our language, they could be computed from the component; a dedicated algorithm would be used for computing representatives for their states.

These aspects are largely orthogonal to the real-time concerns which were the main focus of the work presented here; we hope to investigate them in the future.

References

[1] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of the expressiveness of timed automata and time petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 211–225. Springer Berlin Heidelberg, 2005.

[2] 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.

[3] 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.

[4] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the Gap Between Timed Automata and Bounded Time Petri Nets. In *Formal Modeling and*

Analysis of Timed Systems (FORMATS'06), Springer LNCS 4202, pages 82–97, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] H. Boucheneb and J. Mullins. Analyse des Réseaux Temporels: Calcul des Classes en O(n^2) et des Temps de Chemin en O($m \times n$). *TSI. Technique et Science Informatiques*, 22(4):435–459, 2003.

[9] P.-A. Bourdil. *Contribution à la modélisation et la vérification formelle par model-checking-Symétries pour les Réseaux de Petri Temporels*. PhD thesis, INSA Toulouse, 2015.

[10] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario. Oris: a tool for modeling, verification and evaluation of real-time systems. *International journal on software tools for technology transfer*, 12(5):391–403, 2010.

[11] G. Bucci and E. Vicario. Compositional validation of time-critical systems using communicating time petri nets. *Software Engineering, IEEE Transactions on*, 21(12):969–992, 1995.

[12] 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.

[13] 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.

[14] 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.

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

[16] G. Gardey, D. Lime, M. Magnin, et al. Romeo: A tool for analyzing time petri nets. In *Computer Aided Verification*, pages 418–423. Springer Berlin Heidelberg, 2005.

[17] M. Hack. Petri net languages. Technical Report TR 159, MIT, Cambridge, Massachusetts, 1976.

[18] M. Hendriks. *Enhancing uppaal by exploiting symmetry*. Nijmegen Institute for Computing and Information Sciences, Faculty of Science, University of Nijmegen, NIII-R0208 October, 2002.

[19] 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.

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

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

[22] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 54–57. Springer Berlin Heidelberg, 2009.

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

[24] A. Miller, A. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *ACM Comput. Surv.*, 38(3), Sept. 2006.

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

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

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

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

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

[30] 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.

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

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

[33] E. Vicario. Static Analysis and Dynamic Steering of Time-Dependent Systems. *IEEE Transactions on Software Engineering*, 27(8):728–748, Aug. 2001.

[34] T. Wahl and A. Donaldson. Replication and abstraction: Symmetry in automated formal verification. *Symmetry*, 2(2):799–847, 2010.

[35] 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.

Proof of Lemma 5, Page 14

Lemma 5 is pictured in Figure 1.

Assume $C = (m, D)$ is a class obtained in the SCG construction and i, j are two transitions with the same static time interval ($I_s(i) = I_s(j)$). We show that if $i \preceq_D j$, then (i) $\alpha_i \leq \alpha_j$, (ii) $\beta_i \leq \beta_j$, and (iii) for all transition k distinct from i, j, $\gamma_{k,j} \leq \gamma_{k,i}$. (In the remainder of the proof, we use the name F to denote part (iii) of this conjunction).

The proof is by case analysis on i, j.

If both i and j are newly-enabled:

$\alpha_i = \alpha_j$ since $\alpha_i = \alpha_i^s$, $\alpha_j = \alpha_j^s$, and $\alpha_i^s = \alpha_j^s$ by hypothesis

$\beta_i = \beta_j$ since $\beta_i = \beta_i^s$, $\beta_j = \beta_j^s$, and $\beta_i^s = \beta_j^s$ by hypothesis

$\gamma_{k,i} = \gamma_{k,j}$ since

if k is persistent

$\gamma_{k,i} = \beta_k - \alpha_i^s$, $\gamma_{k,j} = \beta_k - \alpha_j^s$, and $\alpha_i^s = \alpha_j^s$ by hypothesis

if k is newly-enabled

$\gamma_{k,i} = \beta_k^s - \alpha_i^s$, $\gamma_{k,j} = \beta_k^s - \alpha_j^s$, and $\alpha_i^s = \alpha_j^s$ by hypothesis

hence $i =_D j$ and consequently F holds

If i persistent and j newly-enabled (or the converse, symmetrically):

$\alpha_i \leq \alpha_j$

since j is newly-enabled, we have $\alpha_j = \alpha_j^s$, so $\alpha_i > \alpha_j \Leftrightarrow \alpha_i > \alpha_j^s$, which is impossible since $\alpha_j^s = \alpha_i^s$ by hypothesis and $0 \leq \alpha_i \leq \alpha_i^s$ by Lemma 3.

$\beta_i \leq \beta_j$

similarly, since j is newly-enabled, we have $\beta_j = \beta_j^s$, so $\beta_i > \beta_j \Leftrightarrow \beta_i > \beta_j^s \Leftrightarrow \beta_i > \beta_i^s$, which contradicts Lemma 3.

$(\forall k \neq i, j)(\gamma_{k,j} \leq \gamma_{k,i})$

if k is persistent

We have:
$\gamma_{k,j} = \beta_k - \alpha_j^s$ since j is newly-enabled,
$\alpha_j^s = \alpha_i^s$ by hypothesis,
$\beta_k - \alpha_i^s \leq \beta_k - \alpha_i$ since $0 \leq \alpha_i \leq \alpha_i^s$ and $0 \leq \beta_k$ by Lem. 3,
Then, if $\gamma_{k,i} = \beta_k - \alpha_i$:
$\gamma_{k,j} = \beta_k - \alpha_j^s = \beta_k - \alpha_i^s \leq \beta_k - \alpha_i = \gamma_{k,i}$
hence $\gamma_{k,j} \leq \gamma_{k,i}$

Otherwise consider any sequence $(C_x)_{0\leq x\leq p}$ of classes from the initial class ending at class C (i.e. $C_p = C$); we denote $\alpha_i^x, \beta_i^x, \gamma_{k,i}^x$ the bounds in class C_x.
Let C_n be the last class in that sequence in which $\gamma_{k,i}^n = \beta_k^n - \alpha_i^n$. By Lemma 2, C_n necessarily exists and, for any $u \in]n, p]$, i and k are persistent in class C_u and $\gamma_{k,i}^u = \gamma_{k,i}^{u-1}$.
Next, we have $\beta_k \leq \beta_k^n$, $0 \leq \alpha_i \leq \alpha_i^s$ and $0 \leq \beta_k$ by Lemma 3, and thus $\beta_k - \alpha_i^s \leq \beta_k^n - \alpha_i^n$
So $\gamma_{k,j} = \beta_k - \alpha_j^s = \beta_k - \alpha_i^s \leq \beta_k^n - \alpha_i^n = \gamma_{k,i}^n = \gamma_{k,i}^p$
hence $\gamma_{k,j} \leq \gamma_{k,i}$

if k is newly-enabled

$\gamma_{k,j} > \gamma_{k,i} \Leftrightarrow \beta_k^s - \alpha_j > \beta_k^s - \alpha_i$
which is impossible since $\alpha_i \leq \alpha_j$ (as seen above)

If both i and j are persistent:

by induction on firing sequences.

Assuming we have $I_s(i) = I_s(j)$, $i \preceq_D j$, F, $(m, D) \xrightarrow{f} (m', D')$, and both i and j are persistent in D', we have
$\alpha'_i \leq \alpha'_j \wedge \beta'_i \leq \beta'_j \wedge (\forall k \neq i, j)(\gamma'_{k,j} \leq \gamma'_{k,i})$ (F')
$\alpha'_i \leq \alpha'_j$

we have $\alpha'_i = -\min_k \gamma_{k,i}$ and $\alpha'_j = -\min_k \gamma_{k,j}$ by Lemma 2,
$\gamma_{i,j} \leq \gamma_{j,i}$ and $(\forall l \neq i, j)(\gamma_{li} \geq \gamma_{lj})$ by induction hypothesis,
so $-\min_k \gamma_{k,i} \leq -\min_k \gamma_{k,j}$ and thus $\alpha'_i \leq \alpha'_j$

$\beta'_i \leq \beta'_j$

we have $\beta'_i = \gamma_{i,f}$ and $\beta'_j = \gamma_{j,f}$ by Lemma 2, $\gamma_{i,f} \leq \gamma_{j,f}$ by induction hypothesis, so $\beta'_i \leq \beta'_j$

$(\forall k \neq i, j)(\gamma'_{k,j} \leq \gamma'_{k,i})$

k is persistent

by Lemma 2,
$\gamma'_{k,i} = \min(\gamma_{k,i}, \beta'_k - \alpha'_i)$ and $\gamma'_{k,j} = \min(\gamma_{k,j}, \beta'_k - \alpha'_j)$
we have four cases to consider:

1. $\gamma'_{k,i} = \gamma_{k,i}$ and $\gamma'_{k,j} = \gamma_{k,j}$: then $\gamma'_{k,j} \leq \gamma'_{k,i}$ since $\gamma_{k,j} \leq \gamma_{k,i}$ by ind. hyp.
2. $\gamma'_{k,i} = \gamma_{k,i}$ and $\gamma'_{k,j} = \beta'_k - \alpha'_j$: we have $\gamma'_{k,j} = \beta'_k - \alpha'_j \leq \gamma_{k,j}$ (by Lemma 2) and $\gamma_{k,j} \leq \gamma_{k,i}$ by ind. hypothesis
so $\gamma'_{k,j} \leq \gamma_{k,j} \leq \gamma_{k,i} \leq \gamma'_{k,i}$, hence $\gamma'_{k,j} \leq \gamma'_{k,i}$

3. $\gamma'_{k,i} = \beta'_k - \alpha'_i$ and $\gamma'_{k,j} = \gamma_{k,j}$: Since $\alpha'_i \leq \alpha'_j$ (shown above) and β_k, α'_i and α'_j are non-negative (Lemma 3), we have $\beta'_k - \alpha'_i \geq \beta'_k - \alpha'_j$
   then $\gamma'_{k,j} \leq \beta'_k - \alpha'_j \leq \beta'_k - \alpha'_i = \gamma'_{k,i}$, hence $\gamma'_{k,j} \leq \gamma'_{k,i}$
4. $\gamma'_{k,i} = \beta'_k - \alpha'_i$ and $\gamma'_{k,j} = \beta'_k - \alpha'_j$: then $\gamma'_{k,i} \geq \gamma'_{k,j}$ since $\alpha'_i \leq \alpha'_j$ (by above)

k is newly-enabled

by Lemma 2 $\gamma'_{k,i} = \beta^s_k - \alpha'_i$, $\gamma'_{k,j} = \beta^s_k - \alpha'_j$
it was proved above that $\alpha'_i \leq \alpha'_j$, and Lemma 3 says that alpha's and beta's are non-negative
hence $\beta^s_k - \alpha'_i \geq \beta^s_k - \alpha'_j$ and consequently $\gamma'_{k,i} \geq \gamma'_{k,j}$

Proof of Lemma 6, Page 14

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

We consider the following three possible cases. Informally, the first case is when transitions i and j are introduced simultaneously in system D (they have the same "age"), the second is when they are introduced at different times (one is older than the other), and the last case asserts that the relationships between i and j is preserved whatever the transition introduced next as long as i and j stay persistent.

Both i and j are newly-enabled at (m, D):

Then $\gamma_{i,j} = \beta^s_i - \alpha^s_j$, $\gamma_{j,i} = \beta^s_j - \alpha^s_i$ and thus $\gamma_{i,j} = \gamma_{j,i}$ since $I_s(i) = I_s(j)$.
for any $k \neq i, j$, $\gamma_{i,k} = \beta^s_i - \alpha_k$, $\gamma_{j,k} = \beta^s_j - \alpha_k$ and thus $\gamma_{i,k} = \gamma_{j,k}$ since $I_s(i) = I_s(j)$.
So $i =_D j$; both $i \preceq_D j$ and $j \preceq_D i$ hold.

i is persistent and j is newly-enabled (or the converse, symmetrically):

$\gamma_{i,j} \leq \gamma_{j,i}$
since j is newly-enabled, we have $\gamma_{i,j} = \beta_i - \alpha^s_j$ and $\gamma_{j,i} = \beta^s_j - \alpha_i$
so $\gamma_{i,j} \leq \gamma_{j,i} \Leftrightarrow \beta_i + \alpha_i \leq \beta^s_j + \alpha^s_j$
but $I_s(i) = I_s(j)$ by hypothesis,
so $\gamma_{i,j} \leq \gamma_{j,i} \Leftrightarrow \beta_i + \alpha_i \leq \beta^s_i + \alpha^s_i$
by Lemma 3 we have $\alpha_i \leq \alpha^s_i$ and $\beta_i \leq \beta^s_i$
hence $\beta_i + \alpha_i \leq \beta^s_i + \alpha^s_i$ and $\gamma_{i,j} \leq \gamma_{j,i}$

$(\forall k \neq i,j)(\gamma_{i,k} \leq \gamma_{j,k})$:

since j is newly-enabled, we have $\gamma_{j,k} = \beta_j^s - \alpha_k$
so $\gamma_{i,k} > \gamma_{j,k} \Leftrightarrow \beta_j^s - \alpha_k < \gamma_{i,k}$
but $\gamma_{i,k} \leq \beta_i - \alpha_k$ by lemma 3
so $\gamma_{i,k} > \gamma_{j,k} \Leftrightarrow \beta_j^s - \alpha_k < \beta_i - \alpha_k \Leftrightarrow \beta_j^s < \beta_i$,
which is impossible since $\beta_j^s = \beta_i^s$ by hypothesis and $\beta_i \leq \beta_i^s$ by Lemma 3.

hence $i \prec_D j$ (or $j \prec_D i$, symmetrically)

Both i and j are persistent and $i \preceq_D j$ (or $j \preceq_D i$, symmetrically):

by induction using Lemma 2:

initially: if j is newly-enabled in D, in which i is persistent, then $i \preceq_D j$ holds (see the previous case)

induction step: assume i and j are persistent in (m, D) and (m', D'), $i \preceq_D j$ and $(m, D) \xrightarrow{f} (m', D')$

$\gamma'_{i,j} \leq \gamma'_{j,i}$

Lemma 2 yields
$\gamma'_{i,j} = \min(\gamma_{i,j}, \beta'_i - \alpha'_j) = \min(\gamma_{i,j}, \gamma_{i,f} - \alpha'_j)$
$\gamma'_{j,i} = \min(\gamma_{j,i}, \beta'_j - \alpha'_i) = \min(\gamma_{j,i}, \gamma_{j,f} - \alpha'_i)$
we have four cases to consider:

1. $\gamma'_{i,j} = \gamma_{i,j}$ and $\gamma'_{j,i} = \gamma_{j,i}$: $\gamma_{i,j} \leq \gamma_{j,i}$ by hyp., so $\gamma'_{i,j} \leq \gamma'_{j,i}$
2. $\gamma'_{i,j} = \gamma_{i,j}$ and $\gamma'_{j,i} = \gamma_{j,f} - \alpha'_i$: $\gamma_{i,f} \leq \gamma_{j,f}$ by ind. hyp. and $\alpha'_j \geq \alpha'_i$ by Lemma 5 so $\gamma'_{i,j} \leq \gamma_{i,f} - \alpha'_j \leq \gamma_{j,f} - \alpha'_i = \gamma'_{j,i}$, that is $\gamma'_{i,j} \leq \gamma'_{j,i}$
3. $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j$ and $\gamma'_{j,i} = \gamma_{j,i}$: then by ind. hyp. $\gamma'_{i,j} \leq \gamma_{i,j} \leq \gamma_{j,i} = \gamma'_{j,i}$, hence $\gamma'_{i,j} \leq \gamma'_{j,i}$
4. $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j$ and $\gamma'_{j,i} = \gamma_{j,f} - \alpha'_i$: then by ind. hyp. and Lemma 5: $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j \leq \gamma_{j,f} - \alpha'_i = \gamma'_{j,i}$, hence $\gamma'_{i,j} \leq \gamma'_{j,i}$

$(\forall k \neq i,j)(\gamma'_{i,k} \leq \gamma'_{j,k})$

if k is persistent

$\gamma'_{i,k} = \min(\gamma_{i,k}, \beta'_i - \alpha'_k) = \min(\gamma_{i,k}, \gamma_{i,f} - \alpha'_k)$
$\gamma'_{j,k} = \min(\gamma_{j,k}, \beta'_j - \alpha'_k) = \min(\gamma_{j,k}, \gamma_{j,f} - \alpha'_k)$
we have four cases to consider:

1. $\gamma'_{i,k} = \gamma_{i,k}$ and $\gamma'_{j,k} = \gamma_{j,k}$: $\gamma_{i,k} \leq \gamma_{j,k}$ by hyp. thus $\gamma'_{i,k} \leq \gamma'_{j,k}$
2. $\gamma'_{i,k} = \gamma_{i,k}$ and $\gamma'_{j,k} = \gamma_{j,f} - \alpha'_k$: $\gamma_{i,f} \leq \gamma_{j,f}$ by ind. hyp. so $\gamma'_{i,k} \leq \gamma_{i,f} - \alpha'_k \leq \gamma_{j,f} - \alpha'_k = \gamma'_{j,k}$, hence $\gamma'_{i,k} \leq \gamma'_{j,k}$

3. $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k$ and $\gamma'_{j,k} = \gamma_{j,k}$: then by ind. hyp. so $\gamma'_{i,k} \leq \gamma_{i,k} \leq \gamma_{j,k} = \gamma'_{j,k}$, hence $\gamma'_{i,k} \leq \gamma'_{j,k}$
4. $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k$ and $\gamma'_{j,k} = \gamma_{j,f} - \alpha'_k$: then by ind. hyp.
   $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k \leq \gamma_{j,f} - \alpha'_k = \gamma'_{j,k}$, hence $\gamma'_{i,k} \leq \gamma'_{j,k}$

if k is newly-enabled

$\gamma'_{i,k} = \beta'_i - \alpha'_k = \gamma_{i,f} - \alpha'_k$
$\gamma'_{j,k} = \beta'_j - \alpha'_k = \gamma_{j,f} - \alpha'_k$
so $\gamma'_{i,k} \leq \gamma'_{j,k}$ holds by induction hypothesis ($\gamma_{i,f} \leq \gamma_{j,f}$)

and consequently $i \preceq_{D'} j$: