Timed State Space Analysis of Real-Time Preemptive Systems

Giacomo Bucci, *Member*, *IEEE*, Andrea Fedeli, Luigi Sassoli, and Enrico Vicario, *Member*, *IEEE*

Abstract—A modeling notation is introduced which extends Time Petri Nets with an additional mechanism of resource assignment making the progress of timed transitions be dependent on the availability of a set of preemptable resources. The resulting notation, which we call Preemptive Time Petri Nets, permits natural description of complex real-time systems running under preemptive scheduling, with periodic, sporadic, and one-shot processes, with nondeterministic execution times, with semaphore synchronizations and precedence relations deriving from internal task sequentialization and from interprocess communication, running on multiple processors. A state space analysis technique is presented which supports the validation of Preemptive Time Petri Net models, combining tight schedulability analysis with exhaustive verification of the correctness of logical sequencing. The analysis technique partitions the state space in equivalence classes in which timing constraints are represented in the form of Difference Bounds Matrixes. This permits it to maintain a polynomial complexity in the representation and derivation of state classes, but it does not tightly encompass the constraints deriving from preemptive behavior, thus producing an enlarged representation of the state space. False behaviors deriving from the approximation can be cleaned-up through an algorithm which provides a necessary and sufficient condition for the feasibility of a behavior along with a tight estimation of its timing profile.

Index Terms—Hard real-time systems, reactive systems, preemptive scheduling, interprocess communication, nondeterministic time parameters, multiprocessor schedulability, timeliness predictability, state space analysis, Preemptive Time Petri Nets.

1 Introduction

In the development of reactive real-time systems, correctness concerns both the logical sequencing and the quantitative timing of events. In the literature of real-time systems, both the aspects are usually faced under the assumption of a number of hypotheses about the structure of the tasking set, which ease the problem of sequencing correctness and enable low-complexity techniques of schedulability analysis.

The most notable example is the theory of Rate Monotonic. This deals with the case of a set of independent and periodic processes releasing tasks characterized by a deterministic computation time and a deadline, scheduled in preemptive manner according to static priorities inverse to the period [20]. Under these assumptions, a sufficient condition for the feasibility of the schedule is tested in linear time by comparing the total utilization yield by the tasking set against a threshold depending on the number of processes. Synchronization on exclusive resources can also be accounted in the theory [27], [19], [28], while asynchronous recurring tasks with a minimum interarrival time must be transformed into an equivalent periodic form [31]. However, the method does not encompass precedence constraints, such as those derived from intertask communication, and it does not considers nondeterministic computation times qualified through a minimum-maximum interval. Also, the sufficient (but not necessary) condition often leaves the scheduling problem undetermined.

In general, this kind of analytical techniques are impaired by a number of factors such as: sporadic tasks or tasks with mutual dependencies in the time of release, intertasks dependencies due to mutual exclusion on shared resources or to dataflow precedence relations, internal sequencing of tasks, nondeterministic computation times, multiple processors, and flexible adaptation of process parameters to transient overload. Under such conditions, the verification of both sequencing and timing correctness may become sufficiently critical to motivate the use of complex modeling formalisms and static analysis methods.

Tasking sets that include any of the above-mentioned factors can be suitably represented using formalisms such as Timed Automata [3], [9], [16], Time Petri Nets [10], [30], Process Algebras [8], [24], or even through concurrent programming languages [15], which open the way to verification based on state space analysis methods. This results in a much higher computational effort, but also provides insight on the patterns of concurrency and on their effect on system timeliness. As a common trait, in all these formalisms, the semantics of the system can be defined in terms of a state transition rule driving the evolution of a *logical location* and of a set of quantitative *clocks*. While logical locations are discrete, clocks take values in dense domains. This raises a factor of complexity which pervades state space analysis, and which can be attacked either by reducing clocks to a discrete representation [8], [2] or by maintaining their density [3], [9], [16], [10], [30].

• *The authors are with the Dipartimento di Sistemi e Informatica, via S. Marta, 3 - 50139 Firenze, Italy.*
E-mail: {bucci, fedeli, sassoli, vicario}@dsi.unifi.it.

Manuscript received 9 Sept. 2002; revised 5 Aug. 2003; accepted 29 Dec. 2003.
Recommended for acceptance by G. Ciardo.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 117294.

When clocks are interpreted over a dense domain, a discrete state space representation is obtained by collecting states into state classes each characterized by a logical location and by a dense variety of clock assignments. In particular, dense clock domains can be efficiently encoded as sets of simple linear inequalities, each composed by two terms. Using this kind of encoding, usually referred to as *Difference Bounds Matrix* (DBM), classes are representated with space complexity $O(N^2)$ (N standing for the number of active clocks) and their manipulation amounts to an all-shortest-path, which is efficiently solved in time $O(N^3)$ [17] or even $O(N^2)$ when it is repeatedly applied within an enumeration algorithm [30].

Unfortunately, modeling formalisms that can be analyzed using DBM data structures, such as Timed Automata and Time Petri Nets, do not represent clocks whose advancement can be suspended and then resumed. This limits their applicability to the analysis of static table driven systems [26], ruling out preemptive scheduling, which instead is the most common case in the practice of real-time systems.

Extension of Timed Automata to Linear Hybrid Automata (LHA) [4] overcomes the limitation supporting the representation of suspended clocks, but this requires that the system be represented as a particular case of a much wider class of models, preventing ad hoc simplifications both in the modeling and in the analysis stage.

In [14], LHA are restricted into an intermediate class, called Stopwatch Automata (SWA), where the rate of increase of a clock can be switched between 1 and 0 (to account for the progress and the suspension of a computation) across different logical locations. In [14], it is proven that SWA with hidden clocks have the same expressive power of LHA, thus indicating that suspension actually comprises a factor of inherent relevance and difficulty in the modeling and analysis of timed systems. In a practical perspective, [14] also proposes an overapproximate approach to reachability analysis of SWA based on DBM encoding of state classes.

In [21], Timed Automata are extended into a superclass called Suspension Automata, where the rate of increase of a clock takes the same form as with SWA. In [21], it is proven that the problem of reachability becomes undecidable, and that analysis techniques of Timed Automata are no more applicable. To circumvent the problem, the effect of the suspension of a timer is accounted by decrementing it by the duration of the suspension period. In so doing, analysis can be carried out as for Timed Automata, provided that the duration of each suspension period be a deterministic value in the set of natural numbers. In the end, this implies that computation times must be discrete and deterministic. However, computation times cannot be forced to take a deterministic value, due to the inherent variability of task functions as well as to unpredictable effects of hardware/software interaction, such as cache effects. On the other hand, the assumption of deterministic times may lead to anomalies in which the early completion of a task can result in a longer response time of the overall system [6]. As a further limitation, the approach also assumes that precedence relations driving preemption among different timers be explicitly encoded in terms of the locations of the automaton, which may limit expressivity and hurdle a direct manual construction of the model.

The approach of [21] has been nicely developed in Timed Automata with Tasks (TAT) [18], [5], which propose a two-layer modeling method: The events of an arrival automaton are used to trigger the release of a set of tasks; these are put into a queue whose concurrent service on a processor is driven by a scheduler automaton. Schedulability analysis of released tasks is reduced to a problem of reachability in the state space of the product of the arrival and the scheduling automata. The problem is proven to be decidable under the assumption that suspension periods are integer values [18]. The method has been implemented in the Times tool [5], which supports schedulability analysis for tasks with asynchronous and dense release times, running under the most practiced scheduling disciplines. However, the underlying analysis by subtraction still rules out nondeterministic computation times taking values within dense intervals.

In this paper, we introduce a new method for timeliness analysis of real-time concurrent systems running under preemptive scheduling. The system under analysis is modeled using an extension of Time Petri Nets [22], which we call Preemptive Time Petri Nets. This allows representation of complex tasking models scheduled by priority on multiple processors, including periodic and sporadic tasks, with synchronization on exclusive resources and precedence constraints, with nondeterministic computation times. The analysis follows a two-stage verification of reachability and timeliness properties. In the first stage, an approximate representation of the state space of the model is derived, which is proven to be the tightest possible approximation that can be achieved using a DBM encoding of state classes. This representation can be used to check a necessary condition for the feasibility of a behavior and to determine an approximate upper and lower bound on its duration. For any critical behavior identified in the approximate analysis, an exact timing profile can be derived by regenerating the exact set of constraints that limit the dense variety of different timings that can be exhibited by the behavior itself. As a by-product, this supports clean-up of false behaviors introduced in the previous stage of analysis, deriving the exact response time along the behavior and the actual set of states it can visit within overapproximate state classes.

In the rest of the paper, Preemptive Time Petri Nets are introduced in Section 2, and the methods for their analysis are described in Section 3. In Section 4, the proposed modeling and analysis technique is applied to a set of examples addressing common patterns of concurrency. Conclusions are drawn in Section 5. Lemmas and proofs are deferred to the Appendix.

2 Preemptive Time Petri Nets

Preemptive Time Petri Nets (PTPNs) extend the basic model of Petri Nets [23], [25] with the timing semantics of Time Petri Nets [22], [10] and with an original mechanism of resource assignment which conditions the advancement of timers of enabled transitions.

2.1 Syntax

A PTPN is a tuple $PTPN = \langle P; T; A^+; A^-; A^\cdot; M; FI^s; Res; Req; Prio\rangle$.

P and T are disjoint sets of *places* and *transitions*, respectively; A^-, A^+, and $A^\cdot$ are relations on places and transitions called precondition, postcondition and inhibitor arcs, respectively, ($A^- \subseteq P \times T$ and $A^+ \subseteq T \times P$ and $A^\cdot \subseteq P \times T$). A place p is said to be an *input* or an *output* place for a transition t if there exists a precondition or a postcondition from p to t or vice versa, (i.e., if $\langle p,t\rangle \in A^-$ or $\langle t,p\rangle \in A^+$, respectively). p is an inhibitor place for transition t if there exists an inhibitor arc from p to t (i.e., if $\langle p,t\rangle \in A^\cdot$). M is the (initial) marking, associating each place p with a natural number of *tokens* ($M : P \to \mathcal{N}$).

FI^s associates each transition t with a *firing interval* delimited by an *earliest firing time* $EFT^s(t)$ and a *latest firing time* $LFT^s(t)$ taking values in the set of real numbers.

Res is a set of *resources* (disjoint from T and P), Req associates each transition with a subset of Res (i.e., $Req : T \to 2^{Res}$), and $Prio$ associates each transition with a natural number (i.e., $Prio : T \to \mathcal{N}$).

2.2 Semantics

The *state* of a PTPN is a pair $s = \langle M, \tau\rangle$, where M is the *marking* and τ associates each transition with a possibly infinite *time to fire* value ($\tau : T \to \mathcal{R}^+ \cup \{\infty\}$). The state evolves according to a transition rule made up of two clauses of *firability* and *firing*.

Firability: A transition t_o is *enabled* if each of its input places contains at least one token and none of its inhibiting places contains any token. An enabled transitions t_o is *progressing* if and only if every resource it requires is not required by any other enabled transition with a higher level of priority. Transitions that are enabled but not progressing are said to be *suspended*. A transition t_o is *firable* if it is progressing and its time to fire $\tau(t_o)$ is not higher than the time to fire of any other progressing transition.

Firing: When a transition t_o fires, the state $s = \langle M, \tau\rangle$ is replaced by a new state $s' = \langle M', \tau'\rangle$. The marking M' is derived from M by removing a token from each input place of t_o, and by adding a token to each output place of t_o:

$$\begin{aligned} M_{tmp}(p) &= M(p) - 1 \quad \forall p.\langle p, t_o\rangle \in A^- \\ M'(p) &= M_{tmp}(p) + 1 \quad \forall p.\langle t_o, p\rangle \in A^+. \end{aligned} \tag{1}$$

Transitions that are enabled both by the temporary marking M_{tmp} and by the final marking M' are said *persistent*, while those that are enabled by M' but not by M_{tmp} are said *newly enabled*. If t_o is still enabled after its own firing, it is always regarded as newly enabled.

The time to fire τ' of any transition enabled by the new marking M' is computed in a different manner for newly enabled transitions, for persistent-progressing transitions, and for persistent-suspended transitions:

1. For transition t_a which is newly enabled after the firing of t_o, the time to fire takes a nondeterministic value sampled in the static firing interval:
$$EFT^s(t_a) \le \tau'(t_a) \le LFT^s(t_a). \tag{2}$$
2. For any transition t_i which was progressing in the previous state and is persistent after the firing of t_o, the time to fire is reduced by the time elapsed in the previous state. This is equal to the time to fire of t_o as it was measured at the entrance in the previous state:
$$\tau'(t_i) = \tau(t_i) - \tau(t_o). \tag{3}$$
3. Finally, for any persistent transition t_x, that was suspended in the previous state, the time to fire remains unchanged:
$$\tau'(t_x) = \tau(t_x). \tag{4}$$

It is worth noting that the time to fire $\tau(t)$ of any nonenabled transition t does not condition the future evolution of the net: $\tau(t)$ does not condition the firability of any transition, and it will be reset to a new value as soon as t will be enabled again. According to this, the state of the net is sufficiently described by the marking and by the time to fire of enabled transitions.

2.3 Discussion

Preemptive behavior: The clauses of firability and firing give meaning to Res as a set of preemptable resources that can condition the progress of computations modeled by transitions and that are assigned to transitions according to a preemptive protocol based on $Prio$.

Modeling of preemptive behavior was addressed in the context of Petri Nets in [34] for nets with deterministic durations and in [12], [32], [33] for nets with stochastic timing. In [12], [32], each transition can be associated with one out of three execution policies differing in the way the clock (there called *age*) is maintained or reset when the transition is disabled by the lack of a token in any input place. In particular, in the so-called *preemption resume* policy, the clock is frozen when the transition is disabled and it is resumed when the transition is enabled again (suspension). Whereas, in *preemption repeat different*, the clock is lost when the transition is disabled (termination).

The key difference with respect to that formulation is that PTPNs consider two separate mechanisms that condition the progress of times to fire: While the lack of a token in any input place resets the time to fire of the transition, the lack of a required resource suspends the progress of the time to fire which is then resumed when the transition is assigned the resource. The combination of the two mechanisms permits a separate representation of explicit process

dependencies, due to interprocess communication, from the management of preemptable resources, which is driven by the operating system scheduler. This permits a direct representation of realistic preemptive scheduling mechanisms without impacting on the complexity of models, and permits analysis of the same tasking set under various scheduling disciplines as developed in [5], [2], [8].

Interleaving semantics: While the firability clause allows multiple firable transitions for a single state, the firing clause indicates that the state evolves through the firing of a single transition at each step. Of course, multiple state transitions may happen in zero time.

No multiple enablings: As assumed in [10] and [30], a transition t_i which is still enabled after its own firing is always considered as newly enabled, whether it is enabled or not by the temporary marking M_{tmp} of the firing clause. This avoids a number of semantic subtleties, not in the focus of this paper, that arise in the case that one or more transitions have sufficient tokens in input places to permit multiple firings. As a drawback, the limitation that we assume prevents compact modeling of the case of a computation performed in parallel by a variable number of processes (e.g., a "multiple server"), which is often employed in Markovian models for performance evaluation.

3 State Space Analysis of PTPN Models

The state of a PTPN depends not only on the marking but also on timers associated with transitions. While markings are discrete, timers take values in a dense space. To obtain a discretely enumerable reachability relation, the state space must be partitioned into equivalence classes, each collecting a dense variety of states.

For Time Petri Nets, this is obtained by collecting together the states that are reached through the same firing sequence but with different firing times [11], [10]. This equivalence yields a compact partitioning in which the advancement of time without firing of a transition does not produce a change of the state class. Each of these classes turns out to be sufficiently represented by a marking and by a *firing domain* expressed as a set of linear inequalities in the form of a DBM constraining the times to fire of enabled transitions. Properties of firing domains in this form are discussed in [30] with reference to the efficient enumeration of a reachability relation among state classes, to the solution of timed reachability problems, and to the evaluation of a profile of the set of feasible timings for any execution sequence.

In this section, we show that constraints on the timers of suspended transitions can take the form of any kind of linear inequality, changing the nature of time and space complexity involved in the derivation and representation of state classes. To avoid the complexity, we propose the enumeration of an approximate relation of succession among state classes, which maintains inequalities in DBM form, and still permits to obtain necessary conditions for the reachability of a state or for the feasibility of an execution sequence. An algorithm is then introduced which regenerates the exact set of constraints limiting the set of feasibile timings for an execution sequence, thus permitting the derivation of tight bounds for the feasible timings of the sequence. As a by-product, the algorithm also provides a necessary and sufficient condition for the feasibility of the trace, thus supporting a clean-up of false behaviors introduced by the approximation.

3.1 State Classes and Their Reachability Relation

We partition the state space of a PTPN in *state classes*, each represented as a tuple $S = \langle M, D \rangle$, where M is a marking and D is a firing domain which identifies a (dense) set of values for the timers associated with enabled transitions. While the marking M is represented in a straightforward manner, the firing domain D must be encoded as the space of solutions for the set of constraints limiting the timers of enabled transitions. The form of these constraints depends not only on the way in which transition timers are made to advance in the firing clause of model semantics, but also on: 1) the semantics of the reachability relation that we want to establish on state classes and 2) the form of constraints in the initial state class.

Semantics of Reachability: Following the approach of [10], [13], [30], we say that a state class S' is reachable from class S through transition t_o , and we write $S \xrightarrow{t_o} S'$, if and only if S' contains all and only the states that are reachable from some state collected in S through some feasible firing of t_o.

Note that the relation $S \xrightarrow{t_o} S'$ does not imply that every state $s' \in S'$ is reachable from every state $s \in S$. Instead, it guarantees that for any state $s' \in S'$ there exists at least one state $s \in S$ and a time τ_o such that t_o can be fired from state s with firing time τ_o and the resulting state is s'.

Representation of the Initial Class: We assume that the firing domain D of the initial class S is represented as a set of linear inequalities in the form of DBM, i.e., a set of inequalities constraining the difference between the timers of any two enabled transitions:

$$D = \left\{ \tau(t_i) - \tau(t_j) \leq b_{ij} \quad \forall t_i, t_j \in T^{enab}(S) \bigcup \{t_*\} \text{ with } i \neq j \right\}, \tag{5}$$

where $b_{ij} \in \mathcal{R} \cup \{\infty\}$ are real numbers, $\tau(t_i)$ is the unknown value representing the timer of transition t_i, $T^{enab}(S)$ is the set of transitions enabled by the class marking, and t_* is the fictitious event of entrance into the class (so that $\tau(t_*)$ represents the *ground* time at which the class was entered). For simplicity of notation, in the sequel of the treatment, we will avoid specifying the constraint $i \neq j$, assuming that diagonal coefficients of a DBM are equal to 0.

This representation is associated with a *normal form* in which b_{ij} coincides with the maximum value of the difference $\tau(t_i) - \tau(t_j)$ that can be attained by any solution of the set. The normal form exists, it is unique, and can be computed in polynomial time (cubic in the number of transition timers) as the solution of an "all shortest path" problem on a complete graph in which each timer $\tau(t_i)$ is a

vertex and each coefficient b_{ij} is the length of the edge from $\tau(t_j)$ to $\tau(t_i)$ [1], [30]. A firing domain is in normal form if and only if:

$$\begin{aligned} & b_{ij} \leq b_{ih} + b_{hj} \\ & \forall i,j,h \text{ such that } t_i,t_j,t_h \in T^{enab}(S) \bigcup \{t_*\}. \end{aligned} \tag{6}$$

3.1.1 Successor Existence

According to the semantics of class reachability, class $S = < M, D >$ has a successor through transition t_o if and only if t_o is progressing under the class marking M and the firing domain D admits solutions in which the timer of t_o is not higher than that of any other progressing transition, i.e., there is a nonempty set of solutions for the *restricted firing domain* D^{t_o}:

$$D^{t_o} = \left\{ \begin{array}{l} \tau(t_i) - \tau(t_j) \leq b_{ij} \\ \tau(t_o) - \tau(t_I) \leq 0 \end{array} \right. \quad \begin{array}{r} \\ \\ \forall t_i, t_j \in T^{enab}(S) \bigcup \{t_*\} \\ \forall t_I \in T^{progr}(S) \setminus \{t_*\}, \end{array} \tag{7}$$

where $T^{progr}(S)$ is the set of transitions that are progressing under the marking of class S.

If coefficients b_{ij} are in normal form, the maximum acceptable value for the difference $\tau(t_i) - \tau(t_o)$ is equal to b_{io}. According to this, D^{t_o} admits solutions if and only if $b_{io} \geq 0 \quad \forall t_i \in T^{enab}(S)$. This reduces detection of the outcoming arcs of any state class to the inspection of its coefficients in normal form.

The restricted domain D^{t_o} is a DBM set of inequalities that can be reduced itself in normal form. We denote as B_{ij} the coefficients of this normal representation. In [30], it is proven that, if coefficients b_{ij} are in normal form, then coefficients B_{ij} can be derived from coefficients b_{ij} in time $O(n^2)$, where n is the number of enabled transitions.

3.1.2 Successor Computation

The firing of transition t_o leads to a successor class $S' = \langle M', D' \rangle$, where marking M' is derived from M through the token moves of (1), and the firing domain D' is derived according to the rules of the firing clause. To illustrate the derivation, we express D' as the conjunction of two sets of inequalities D'_{pers} and D'_{new} constraining the time to fire τ' of persistent and newly enabled transitions, respectively:

$$D' = \left\{ \begin{array}{l} D'_{new} \\ D'_{pers}. \end{array} \right. \tag{8}$$

D'_{new} is obtained in straightforward manner by constraining the time to fire of each newly enabled transition t_a within its earliest and latest static firing time ((2)):

$$D'_{new} = \left\{ \begin{array}{l} \tau'(t_a) - \tau'(t_*) \leq LFT^s(t_a) \\ \tau'(t_*) - \tau'(t_a) \leq -EFT^s(t_a) \end{array} \right. \quad \begin{array}{r} \\ \\ \forall t_a \in T^{new}(S'). \end{array} \tag{9}$$

Inequalities of the set D'_{pers}, constraining the new timers τ' of transitions that are persistent through the firing of t_o are derived according to (3)-(4) from the constraints on the old timers τ which appear in the restricted firing domain D^{t_o}:

1. For any transition t_i which was progressing in S and which is persistent in S', the difference between the time to fire and the ground is reduced by the value of the time elapsed in the parent class (which is equal to $\tau(t_o) - \tau(t_*)$):

$$\begin{aligned} & \tau'(t_i) - \tau'(t_*) = \tau(t_i) - \tau(t_*) - (\tau(t_o) - \tau(t_*)) \\ & \forall t_i \in T^{pers}(S') \cap T^{progr}(S) \setminus \{t_o\}. \end{aligned} \tag{10}$$

2. For any transition t_x which was suspended in S and which is persistent in S', the difference between the time to fire and the ground remains unchanged:

$$\begin{aligned} & \tau'(t_x) - \tau'(t_*) = \tau(t_x) - \tau(t_*) \\ & \forall t_x \in T^{pers}(S') \cap T^{susp}(S). \end{aligned} \tag{11}$$

By substituting variables according to (10)-(11), the set D^{t_o} of (7) takes the form:

$$D^{t_o}_{pers} = \left\{ \begin{array}{l} \tau'(t_i) - \tau'(t_j) \leq B_{ij} \\ \tau'(t_x) - \tau'(t_y) \leq B_{xy} \\ \tau'(t_x) - \tau'(t_*) \leq B_{x*} \\ \tau'(t_*) - \tau'(t_x) \leq B_{*x} \\ \tau'(t_i) - \tau'(t_*) \leq B_{io} \\ \tau'(t_*) - \tau'(t_i) \leq B_{oi} \\ \tau(t_o) - \tau(t_*) \leq B_{o*} \\ \tau(t_o) - \tau(t_*) \leq B_{i*} - (\tau'(t_i) - \tau'(t_*)) \\ \tau(t_o) - \tau(t_*) \leq B_{ix} - (\tau'(t_i) - \tau'(t_x)) \\ \tau(t_o) - \tau(t_*) \leq B_{ox} - (\tau'(t_*) - \tau'(t_x)) \\ \tau(t_*) - \tau(t_o) \leq B_{*o} \\ \tau(t_*) - \tau(t_o) \leq B_{*j} + (\tau'(t_j) - \tau'(t_*)) \\ \tau(t_*) - \tau(t_o) \leq B_{yj} + (\tau'(t_j) - \tau'(t_y)) \\ \tau(t_*) - \tau(t_o) \leq B_{yo} + (\tau'(t_*) - \tau'(t_y)) \end{array} \right. \quad \begin{array}{r} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \forall t_i, t_j \in T^{pers}(S') \cap T^{progr}(S) \setminus \{t_o\} \\ \forall t_x, t_y \in T^{pers}(S') \cap T^{susp}(S), \end{array} \tag{12}$$

where inequalities have been developed to make explicit the constraints involving the firing transition t_o and to distinguish progressing transitions (associated with indexes i and j) from suspended transitions (associated with indexes x and y).

The set D'_{pers} can now be derived from $D^{t_o}_{pers}$ by eliminating the difference $\tau_o = (\tau(t_o) - \tau(t_*))$ through a projection operation. If $\{\tau_i\}$ denotes an array of values and $\langle \{\tau(t_i)\}, \tau_o \rangle$ denotes the extension of the array with an additional value τ_o, this operation yields a set D'_{pers} which satisfies the following properties: If $\langle \{\tau_i\}, \tau_o \rangle$ is a solution for $D^{t_o}_{pers}$, then $\langle \{\tau_i\} \rangle$ must be a solution for D'_{pers}; conversely, if $\langle \{\tau_i\} \rangle$ is a solution for D'_{pers}, then there must exist a value for τ_o such that $\langle \{\tau_i\}, \tau_o \rangle$ is a solution for $D^{t_o}_{pers}$. This yields the following representation for D'_{pers}:

$$D'_{pers} = \begin{cases} \tau'(t_i)-\tau'(t_j) \le C_{ij} & (ij) \\ \tau'(t_x)-\tau'(t_y) \le C_{xy} & (xy) \\ \tau'(t_x)-\tau'(t_*) \le C_{x*} & (x*) \\ \tau'(t_*)-\tau'(t_x) \le C_{*x} & (*x) \\ \tau'(t_i)-\tau'(t_*) \le C_{i*} & (i*) \\ \tau'(t_*)-\tau'(t_i) \le C_{*i} & (*i) \\ \tau'(t_i)-\tau'(t_x) \le C_{ix} & (ix) \\ \tau'(t_x)-\tau'(t_i) \le C_{xi} & (xi) \\ \\ (\tau'(t_i)-\tau'(t_x))+(\tau'(t_y)-\tau'(t_j)) \le C_{ix}+C_{yj}-c_{ixyj} & (ixyj) \\ (\tau'(t_*)-\tau'(t_x))+(\tau'(t_y)-\tau'(t_j)) \le C_{*x}+C_{yj}-c_{*xyj} & (*xyj) \\ (\tau'(t_i)-\tau'(t_*))+(\tau'(t_y)-\tau'(t_j)) \le C_{i*}+C_{yj}-c_{i*yj} & (i*yj) \\ (\tau'(t_i)-\tau'(t_x))+(\tau'(t_*)-\tau'(t_j)) \le C_{ix}+C_{*j}-c_{ix*j} & (ix*j) \\ (\tau'(t_i)-\tau'(t_x))+(\tau'(t_y)-\tau'(t_*)) \le C_{ix}+C_{y*}-c_{ixy*} & (ixy*) \\ (\tau'(t_*)-\tau'(t_x))+(\tau'(t_*)-\tau'(t_j)) \le C_{*x}+C_{*j}-c_{*x*j} & (*x*j) \\ (\tau'(t_i)-\tau'(t_*))+(\tau'(t_y)-\tau'(t_*)) \le C_{i*}+C_{y*}-c_{i*y*} & (i*y*) \\ \\ \forall t_i,t_j \in T^{pers}(S') \cap T^{progr}(S) \setminus \{t_o\} \\ \forall t_x,t_y \in T^{pers}(S') \cap T^{susp}(S), \end{cases} \tag{13}$$

where coefficients C and c are defined as:

$$C_{ij} = B_{ij}, C_{xy} = B_{xy}, C_{x*} = B_{x*}, C_{*x} = B_{*x}, C_{i*} = B_{io},$$
$$C_{*i} = B_{oi}, C_{ix} = B_{ix} + B_{*o}, C_{xi} = B_{xi} + B_{o*},$$

and

$$c_{ixyj} = (B_{*o} + B_{o*}), c_{ix*j} = (B_{*o} + B_{oj} - B_{*j}),$$
$$c_{i*yj} = (B_{io} + B_{o*} - B_{i*}), c_{ixy*} = (B_{y*} + B_{*o} - B_{yo}),$$
$$c_{*xyj} = (B_{o*} + B_{*x} - B_{ox}), c_{i*y*} = (B_{io} + B_{y*} - B_{i*} - B_{yo}),$$
$$c_{*x*j} = (B_{oj} + B_{*x} - B_{*j} - B_{ox}).$$

By exploiting the assumption that coefficient B is in normal form, it can be verified that coefficient C is also in normal form, i.e.:

$$C_{\alpha\beta} + C_{\beta\gamma} \ge C_{\alpha\gamma} \quad \forall \alpha,\beta,\gamma \text{ such that } t_\alpha, t_\beta, t_\gamma \in T^{pers}(S'). \tag{14}$$

The proof is simple but tedious, as the replacement of C through B is not regular and requires the enumeration of all the possible different interpretations of indexes α, β, and γ over the indexes $*$, i, j, and x, y which identify the ground, a progressing transition and a suspended transition, respectively. For instance, when $\alpha,\beta,\gamma = *, i, x$, (14) takes the form $C_{*i} + C_{ix} \ge C_{*x}$, which becomes $B_{oi} + B_{ix} + B_{*o} \ge B_{*x}$, which, in turn, follows from $B_{*o} + B_{oi} + B_{ix} \ge B_{*x}$.

3.1.3 Successor Approximation

The set D'_{pers} is not in DBM form as inequalities $(ixyj)$ through $(i * y*)$ include more than two unknown values. This means that the space of firing domains that can be represented as a DBM is not closed with respect to the succession transformation induced by the firing clause of Preemptive Time Petri Nets.

Repeated application of the succession transformation yields more and more complex domains, with inequalities which can include any subset of the set of enabled transitions, resulting in a firing domain with a number of inequalities exponential in the number of enabled transitions. In addition, since inequalities can take any linear form, the detection and the computation of successor classes become general linear programming problems, which can be solved with algorithms whose complexity is at least polynomial in the number of domain inequalities and, thus, exponential in the number of enabled transitions.

To circumvent both size and time complexities, we replace D'_{pers} with an overapproximation $\bar{D}'_{pers}$ derived by discarding inequalities $(ixyj)$ through $(*x * j)$:

$$\bar{D}'_{pers} = \begin{cases} \tau'(t_i) - \tau'(t_j) \le C_{ij} & (ij) \\ \tau'(t_x) - \tau'(t_y) \le C_{xy} & (xy) \\ \tau'(t_x) - \tau'(t_*) \le C_{x*} & (x*) \\ \tau'(t_*) - \tau'(t_x) \le C_{*x} & (*x) \\ \tau'(t_i) - \tau'(t_*) \le C_{i*} & (i*) \\ \tau'(t_*) - \tau'(t_i) \le C_{*i} & (*i) \\ \tau'(t_i) - \tau'(t_x) \le C_{ix} & (ix) \\ \tau'(t_x) - \tau'(t_i) \le C_{xi} & (xi) \\ \\ \forall t_i, t_j \in T^{pers}(S') \cap T^{progr}(S) \setminus \{t_o\} \\ \forall t_x, t_y \in T^{pers}(S') \cap T^{susp}(S). \end{cases} \tag{15}$$

Since all the inequalities in $\bar{D}'_{pers}$ are included in D'_{pers}, any solution admitted by D'_{pers} is also admitted by $\bar{D}'_{pers}$. However, the firing domain $\bar{D}'_{pers}$ also admits solutions that would not be feasible for the exact representation of the firing domain (false behaviors), thus weakening the semantics of reachability between classes.

Lemma 5.1, in the Appendix, shows that $\bar{D}'_{pers}$ is the tightest possible overapproximation of D'_{pers} within the limits of the DBM representation. In fact, due to the conditions of normalization set for coefficients B, inequalities $(ixyj)$ through $(i * y*)$ are shown not to restrict the difference between any two unknown values more than what is already done by inequalities (ij) through (xi), which we maintain in $\bar{D}'_{pers}$. In addition, (14) indicates that $\bar{D}'_{pers}$ is by construction in normal form.

The idea of recasting the problem of suspension into an overapproximation based on DBM was prospected in [14]. In that treatment, the maximum allowed difference between any running clock and any suspended clock was set equal to infinity, which, in our framework, would be equivalent to neglecting also inequalities (ix) and (xi). Our derivation of a tightest and normal representation, not only reduces the effects of approximation, but also enables analytical characterization of the conditions which produce false behaviors and derivation of algorithms supporting their clean-up.

To mark the difference between exact and approximate representation of the succession relation between state classes, we will write $S \xrightarrow{t_o} \bar{S}'$ to indicate that $\bar{S}'$ is the tightest

DBM approximation of the class S' reached from S through t_o (i.e., the class such that $S \xrightarrow{t_o} S'$).

The concrete effect of the relaxation induced by $S \xrightarrow{t} \bar{S}'$ with respect to $S \xrightarrow{t} S'$ can be illustrated in a relatively simple manner by referring to inequality $(ixyj)$. At the entrance in the parent class S, the differences $\Delta_{ix} = \tau(t_i) - \tau(t_x)$ and $\Delta_{yj} = \tau(t_y) - \tau(t_j)$ are tightly upper-bounded by B_{ix} and B_{yj}, respectively. When t_o fires, Δ_{ix} has decreased by the time elapsed in the parent class and Δ_{yj} has increased by the *same* value. According to this, the sum $\Delta_{ix} + \Delta_{yj}$ is not varied and its tightest upper-bound is still equal to $B_{ix} + B_{yj}$, which is what inequality $(ixyj)$ guarantees. Whereas, using only inequalities in DBM form, the differences Δ_{ix} and Δ_{yj} can be *separately* upper-bounded by $B_{ix} + B_{*o}$ and $B_{yj} + B_{o*}$, respectively. These bounds guarantee that Δ_{ix} decreases and Δ_{yj} increases by two values which both fall in the feasible range for the time elapsed in the parent class, but they do not guarantee that these two values are exactly the same.

In general, all the inequalities of D'_{pers} that are not in DBM form capture similar relations of equality in the drifts occurred between suspended and progressing clocks during the permanence in the parent class S.

As suggested by this interpretation, the replacement of D'_{pers} through $\bar{D}'_{pers}$ relaxes constraints iff the parent class includes both suspended and progressing transitions which are persistent after the firing of t_o. Moreover, no relaxation occurs when the time of permanence in the parent class is constrained to take a deterministic value. This kind of *well-formedness* conditions, under which the approximate relation $\xrightarrow{t}$ is equivalent to the exact relation $\xrightarrow{t}$, can serve to identify transitions in the state space where approximation errors are actually introduced, or they can even serve to identify structural properties of a model which can confine the effects of approximation (e.g., a net where only one transition can be progressing at any time).

3.2 State Space Analysis

The algorithms for detecting and computing the (approximate) successors of a class can be embedded within a conventional enumeration algorithm to construct a *state class graph* (whose finiteness is not guaranteed) where a vertex represents a class and a directed edge represents an element of the approximate succession relation $\rightharpoonup$. In this representation, two vertices associated with classes S and $\bar{S}'$ are connected trough a directed edge labeled by t if and only if $S \xrightarrow{t} \bar{S}'$.

3.2.1 Reachability Analysis

Vertices in the class graph can be related to the reachable states of the model: if a state s is contained in a class S, then each transition t firable in s is an outcoming arc for S, and the state s' reached from s through t is contained in the class reached from S through the arc labeled by t. This permits us to reduce the identification of reachable states to the analysis of classes in the class graph.

In a similar manner, verification of sequencing properties of the model can be reduced to the analysis of the paths in the class graph. Before expressing the reduction, we give a precise meaning to some terms: We call *run* a sequence r of consecutive events, each consisting of a transition t_n firing at time τ_n. A run r is represented by a *trace* $\rho = \{t_o, t_1, \ldots, t_{N-1}\}$ and a *timing* $\Gamma = \{\tau_o, \tau_1, \ldots, \tau_{N-1}\}$ which, respectively, record the sequence of fired transitions and their relative times of occurrence. A *path* is a sequence of vertices and edges in the state class graph. Note that a path in the graph uniquely identifies a trace ρ, which can be coupled with a dense variety of timings corresponding to different feasible runs.

With reference to this notation, if a run r is executable from an initial state s, then the class graph includes a class S which contains s and which is the starting vertex for a path corresponding to the trace of r.

Based on these relations between classes and states and between paths and runs, *necessary* conditions for properties pertaining to the *logical sequencing* of the model can be derived from the class graph through model checking techniques or even through simple inspection algorithms. In particular, for a state s' to be reachable from a state s, the class graph must contain two classes S and S' which contain s and s' and which are connected through a directed path in the class graph.

3.2.2 Timeliness Analysis

Timing constraints encoded in firing domains of the classes visited by a trace can be composed to derive a *timing profile* which delimits the (dense) range of variation of timings that are feasible for the trace itself. We derive this timing profile by comparing the cumulative running time during which each enabled transition has been progressing along the trace against the maximum and minimum values that can elapse between the initial enabling and the firing or disabling event of the transition itself. This results in a set of constraints for the permanence in each state visited along the trace, in a manner which is conceptually similar to that proposed in [7].

To express the cumulative running time, we denote as $\rho(n)$ the index of the transition fired at the nth step of a trace ρ, for n ranging from 0 to $N-1$ (inclusive). S^n denotes the class left at the firing of $t_{\rho(n)}$, and $b^n_{\alpha\beta}$ the coefficients in normal form of the firing domain of class S^n. A transition t_i which is newly enabled in class S^k will be denoted as t_i^k as long as it is persistent. According to this, any transition t_i appearing in the firing domain of any class S^n is identified as t_i^k, where S^k is the earliest class such that t_i is persistent with continuity from S^n to S^k. Whereas, $last(t_i^k)$ denotes the index of the last class in which t_i^k is persistent.

Using this extended notation, the cumulative time $\tau(t_i^k)$ during which t_i^k has been progressing in the execution of a run r which follows the trace ρ can be expressed as the sum of dwelling times in the classes of the trace in which t_i^k has been progressing:

$$\tau(t_i^k) = \sum_{n=k}^{last(t_i^k)} c_n(t_i) * \tau_n$$
$$\text{with } c_n(t_i^k) = \begin{cases} 1 & \text{if } t_i \text{ is progressing in } S^n \\ 0 & \text{if } t_i \text{ is suspended in } S^n, \end{cases} \tag{16}$$

where τ_n denotes the dwelling time in class S^n, i.e., the time elapsed between the firing of $t_{\rho(n-1)}$ and $t_{\rho(n)}$.

In order to be a feasible execution of the model which originates from some state in the class S^o and executes the events in the trace ρ, the timing of the run r must satisfy three different kinds of constraints:

1. If t_i^k comes to fire along trace ρ (i.e., if it is not disabled before the firing), then its cumulative running time must fall within the minimum and maximum values that are acceptable for the time to fire in the earliest class S^k in which t_i^k is enabled. These values coincide with the coefficients $-b_{*i}^k$ and b_{i*}^k constraining the delay of transition t_i with respect to the ground in the firing domain of class S^k:

$$-b_{*i}^k \leq \sum_{n=k}^{last(t_i^k)} c_n(t_i) * \tau_n \leq b_{i*}^k \quad \forall t_i^k \text{ fired along } \rho. \quad (17)$$

2. Whereas, if t_i^k is disabled before firing, then its cumulative running time is only constrained to be not higher than the maximum value that is acceptable for the time to fire in class S^k:

$$\sum_{n=k}^{last(t_i^k)} c_n(t_i) * \tau_n \leq b_{i*}^k \quad (18)$$
$$\forall t_i^k \text{ enabled but not fired along } \rho.$$

3. Finally, for the initial state of the run to be contained in class S^o, any two transitions t_i^o and t_j^o enabled since the initial class must also satisfy the cross constraints of the firing domain of class S^o:

$$\sum_{n=o}^{last(t_i^o)} c_n(t_i) * \tau_n - \sum_{n=o}^{last(t_j^o)} c_n(t_j) * \tau_n \leq b_{ij}^o \quad (19)$$
$$\forall t_i^o, t_j^o \text{ enabled in the initial class } S^o.$$

It is convenient to note here that, in most cases, t_i^k is newly enabled in class S^k and, thus, $-b_{*i}^k$ and b_{i*}^k appearing in (17) and (18) coincide with the static earliest and latest firing times $EFT^s(t_i)$ and $LFT^s(t_i)$ of transition t_i. This has only one exception, which occurs for transitions that are persistent in the first class S^o, being newly enabled in some previous class prior to the scope of the trace under analysis.

By construction, inequalities in (17)-(19) are satisfied by any run originating from any state contained in the initial class S^o and progressing along the trace ρ. Whereas, Lemma 5.2 in the Appendix demonstrates that any solution for the set of inequalities is a feasible timing for a run which originates from some state contained in class S^o and which follows the events of the trace ρ. According to this, the set yield by (17)-(19) delimits exactly the range of feasible timings for any run following the trace ρ from any state in class S^o.

If the set does not admit solutions, then ρ is a false behavior. Otherwise, any bound on the minimum/maximum delay between any two events along the trace ρ can be computed as a linear programming problem. In particular, since the total duration of the trace can be expressed as the sum $\sum_{n=0}^{N-1} \tau_n$ of dwelling times in visited classes, the maximum time spent in any feasible execution of the trace is the solution of the problem:

$$max \left\{ \sum_{n=0}^{N-1} \tau_n \right\}$$
$$under \begin{cases} -b_{*i}^k \leq \sum_{n=k}^{last(t_i^k)} c_n(t_i) * \tau_n \leq b_{i*}^k & \forall t_i^k \text{ fired along } \rho \\ \sum_{n=k}^{last(t_w^k)} c_n(t_w) * \tau_n \leq b_{w*}^k & \forall t_w^k \text{ enabled but not fired along } \rho \\ \sum_{n=o}^{last(t_i^o)} c_n(t_i) * \tau_n - & \\ \quad \sum_{n=o}^{last(t_j^o)} c_n(t_j) * \tau_n \leq b_{ij}^o & \forall t_i^o, t_j^o \text{ enabled in the initial class } S^o. \end{cases} \quad (20)$$

The common way to derive a solution for any such problem is the simplex method. In principle, this has time complexity exponential in the number of unknown values, which in our case is the length of the trace. However, the actual execution time of the simplex usually reduces to polynomial complexity, which makes it usually preferred with respect to equivalent methods with guaranteed but higher-order polynomial time [1].

4 Modeling and Verifying Tasking Sets

PTPNs combine the capability of quantitative analysis with the expressive power and the modeling convenience needed to represent tasking models of practical interest.

To demonstrate this potential, we introduce a benchmark made up of an incremental set of basic patterns of concurrency which address the different factors of complexity that can impair low-complexity schedulability analysis techniques. In particular, we consider a tasking set running under static priority preemptive scheduling, with periodic and sporadic processes, with nondeterministic execution times, with semaphore synchronization and message passing precedences, with multiple processors. For each case, the PTPN model is derived from an intuitive visual description of the tasking set and it is then analyzed to highlight the involved complexity and the kind of results that can be obtained.

Numerical results reported are obtained using the tool ORIS,[1] which implements the theory reported in this paper and in [30]. In particular, ORIS supports model editing and simulation, approximate state space enumeration and exact timeliness analysis of traces. The latter can be applied to identify and analyze the possible traces between two transition firings, usually accounting for the release and the completion of a process instance. This derives a tight timing profile of each analyzed trace, capturing the dense variety of feasible intertimes between trace events and making evident timing and sequencing conditions which lead to critical executions.

1. ORIS is a mixed C++/Java implementation, runnning on Windows and LINUX platforms. It can be obtained for free by contacting the corresponding author of this paper.

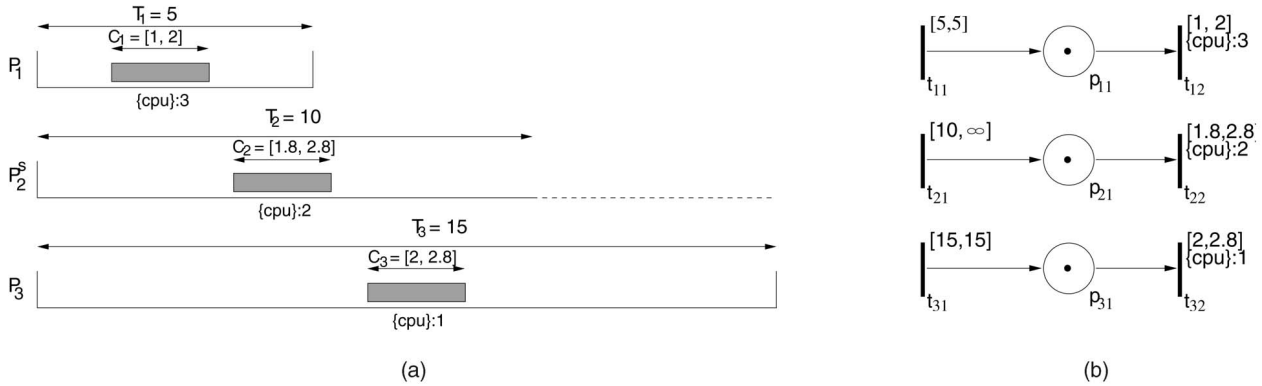

Fig. 1. (a) The schema of two periodic processes P_1 and P_3 (with periods 5 and 15 time units) and a sporadic process P_2^s (with minimum interarrival time of 10 time units), running on a cpu with priorities 3, 1, and 2, respectively. (b) The PTPN model of the tasking set. P_1 is represented by transitions t_{11} (accounting for the cyclic release of a task) and t_{12} (standing for the task completion, which requires the cpu for two time units). P_2 and P_3 are modeled in a similar manner. The static latest firing time of t_{21} is ∞ to model a possibly unbounded delay between subsequent task arrivals for P_2^s.

4.1 Independent Processes

We start with the case of three, independent processes shown in Fig. 1a, which is modeled by the PTPN in Fig. 1b.

Approximate state space enumeration produces a graph with 608 classes. In none of them does any place contain more than one token. This indicates that it is never the case that a process releases a task while its previous instance is still pending.

Selection of the paths beginning with transition t_{31} and terminating with transition t_{32} (accounting for the release and the completion of an instance of process P_3, respectively) identifies 391 traces for process P_3; during clean up, 56 of them turn out to be false behaviors. The analysis does not detect any loops, which would be enlisted separately by the ORIS tool when present. The absence of loops, between the release and the completion events relative to a process instance, will be the common case in all the examples discussed in the rest of this section. This is a consequence of the structure of tasks which, according to the practice of hard real-time systems, are designed in a manner which prevents time-unbounded or Zeno behaviors.

The worst response time for process P_3 (i.e., the longest time elapsed between the firing of transitions t_{31} and t_{32}) is equal to 9.6 time units, corresponding to a laxity (i.e., difference between deadline and response time) of 5.4 time units, and it is attained in the run visualized in Fig. 2a. In a similar manner, the analysis indicates that higher priority processes P_1 and P_2^s have 349 and 133 traces (with no false behaviors) with minimum laxity of 3 and 5.2 time units, respectively. Trace analysis also permits to identify traces realizing more complex maximum or minimum conditions. For instance, Fig. 2b visualizes a run where the system comes to the state where it has the maximum guaranteed idle time.

To attack the example through the Theory of Rate Monotonic, nondeterministic computation times must be replaced through crisp worst cases and the sporadic process P_2^s must be replaced through an equivalent periodic process P_2, such that a scheduling meeting the deadlines of P_2' is guaranteed to meet also the deadlines of P_2^s [31]. Unfortunately, even if the transformation is selected in a manner which minimizes the utilization (by assigning P_2^s a period equal to the half of the deadline of P_2^s), the transformed tasking set is not feasible, thus leaving the problem of schedulability undecided.

Estimation algorithms of [29] can deal with nondeterministic computation times, but not with sporadic processes. Since the periodic transformation of P_2^s produces a tasking set which is not feasible, we test the algorithms in a modified case in which P_2^s is replaced through a (nonequivalent) periodic process P_2' with the same computation

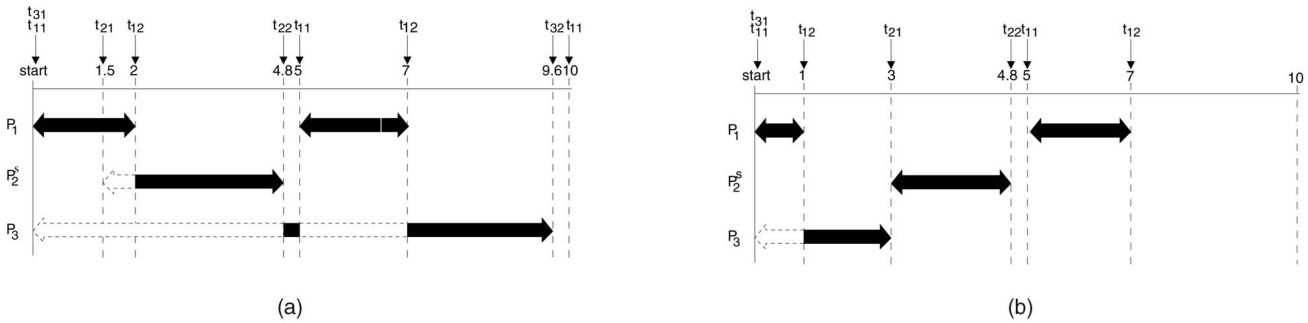

Fig. 2. (a) A run yielding the longest completion time for process P_3 of Fig. 1. The diagram reports the trace of fired transitions and makes evident the sequencing of progressing (filled) and suspended (dashed) computations. (b) A run which yields the maximum guaranteed idle time: after time 5, the system is guaranteed to have at least three time units of idle before time 10.

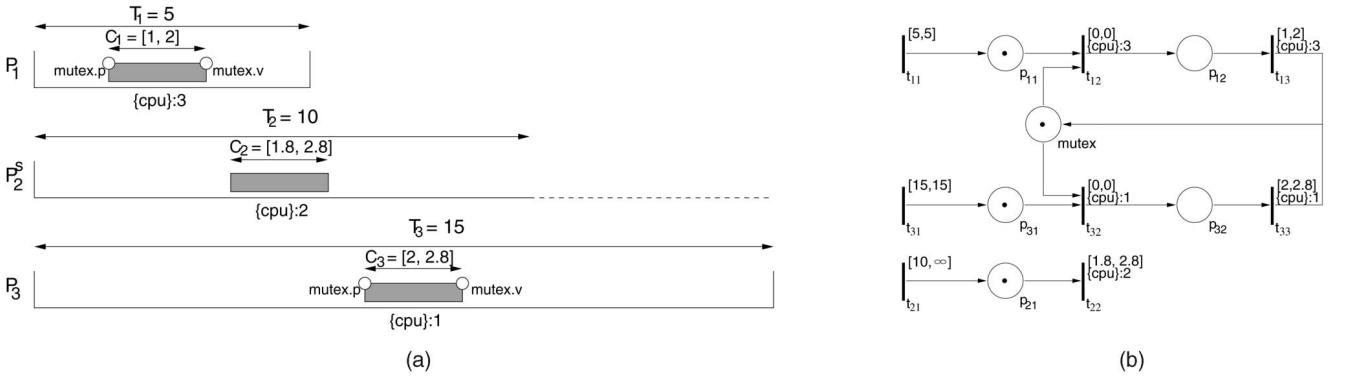

Fig. 3. (a) Processes P_1 and P_3 are synchronized through the mutual exclusive semaphore $mutex$. (b) The PTPN model of the tasking set: by setting the priority of t_{33} to 3, the model would represent a priority ceiling *emulation* protocol.

time as that of P_2^s and with period equal to the minimum interarrival time of P_2^s. For this case, the algorithms of [29] can be correctly applied producing exact estimates of the worst completion time for each of the three processes.

4.2 Semaphore Synchronization

The tasking set of Fig. 3 adds a mutual exclusion semaphore $mutex$ synchronizing processes P_1 and P_3.

The enumeration of the state space of the PTPN model produces a graph with 1,185 classes, including 624, 204, and 624 traces for processes P_1, P_2^s, and P_3, which include 14 false behaviors for P_3 and none for processes P_1 and P_2^s. The analysis evidences traces in which the high priority process P_1 undergoes a *priority inversion* with respect to P_2^s: While P_1 is blocked on the semaphore acquired by P_3, a task of P_2^s preempts the low priority process P_3; in this condition, P_2^s steals time to the higher priority process P_1, leading P_1 to miss its deadline as shown in the run of Fig. 4a. The analysis also shows that the failure is transient in the sense that P_1 is guaranteed to recover its correct timing and meet the deadline in the subsequent instance of the task.

Priority inversion can be avoided by applying a priority ceiling emulation protocol [27]: when a process (P_3) acquires a nonpreemptable resource ($mutex$) its priority is boosted to the maximum among the levels of any process (P_1) that *can* become blocked on that resource. This can be captured in the model of Fig. 3b by assigning priority level 3 to transition t_{33}, which represents the computation of process P_3.

Analysis of the modified model produces a graph with 442 classes, including 209, 74, and 153 traces for processes P_1, P_2^s, and P_3, without any false behavior, with worst completion time of 4.8, 9.6, and 7.6 time units, respectively. Process P_1 always meets the deadline, though the worst case has a critical laxity of only .2 time units. Fig. 4b visualizes a run in which priority ceiling permits P_3 to continue its execution despite the release of a task by P_2^s under the same phasing that led to the priority inversion in Fig. 4a.

The (Generalized) Theory of Rate Monotonic can be applied to semaphores with a priority ceiling protocol by augmenting the worst-case completion time of a any process (P_1 and P_2^s in our case) by the additional latency (the computation time of process P_3) during which a low priority process (P_3) can hold a nonpreemptable resource ($mutex$) [28]. However, also in this case, the utilization estimate results too high to satisfy the test, leaving the feasibility of the schedule undecided. This occurs also when P_2^s is replaced by the (nonequivalent) periodic process P_2' mentioned in the previous example. It is also worth noting that, in the presence of interprocess dependencies (such as the $mutex$ semaphore), the replacement of minimum-maximum computation intervals through a crisp worst case can hide scheduling anomalies occurring in correspondence of early task completions.

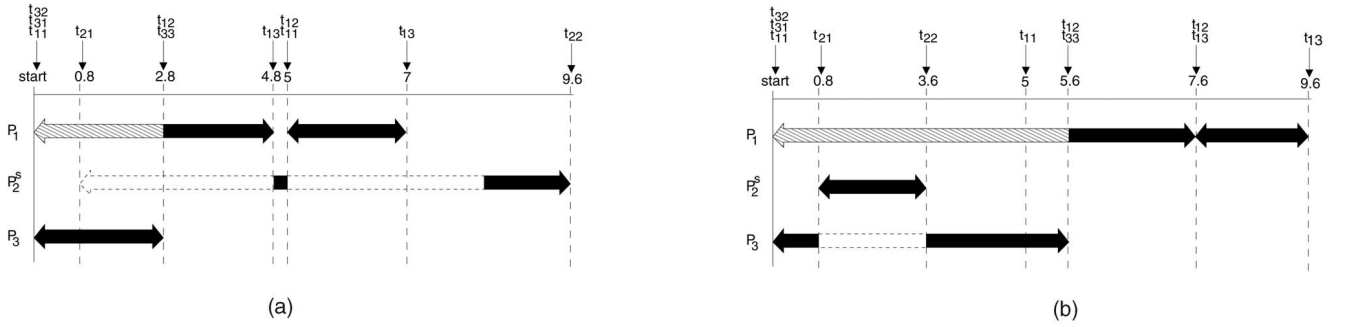

Fig. 4. (a) A run of the model of Fig. 3 (dashed filling indicates the condition of block on the semaphore). At the firing of t_{21}, the system undergoes a priority inversion as P_1 waits not only on P_3 but also on P_2^s. This results in a completion time of 7.6 time units. (b) A run for the variation of the model of Fig. 3 which applies a priority ceiling emulation protocol: in the critical section, P_3 raises its priority to the level of P_1, thus becoming non preemptable by P_2^s. In this case, P_1 completes on time, even under the same phasing of the arrivals which leads to the failure illustrated in (a).

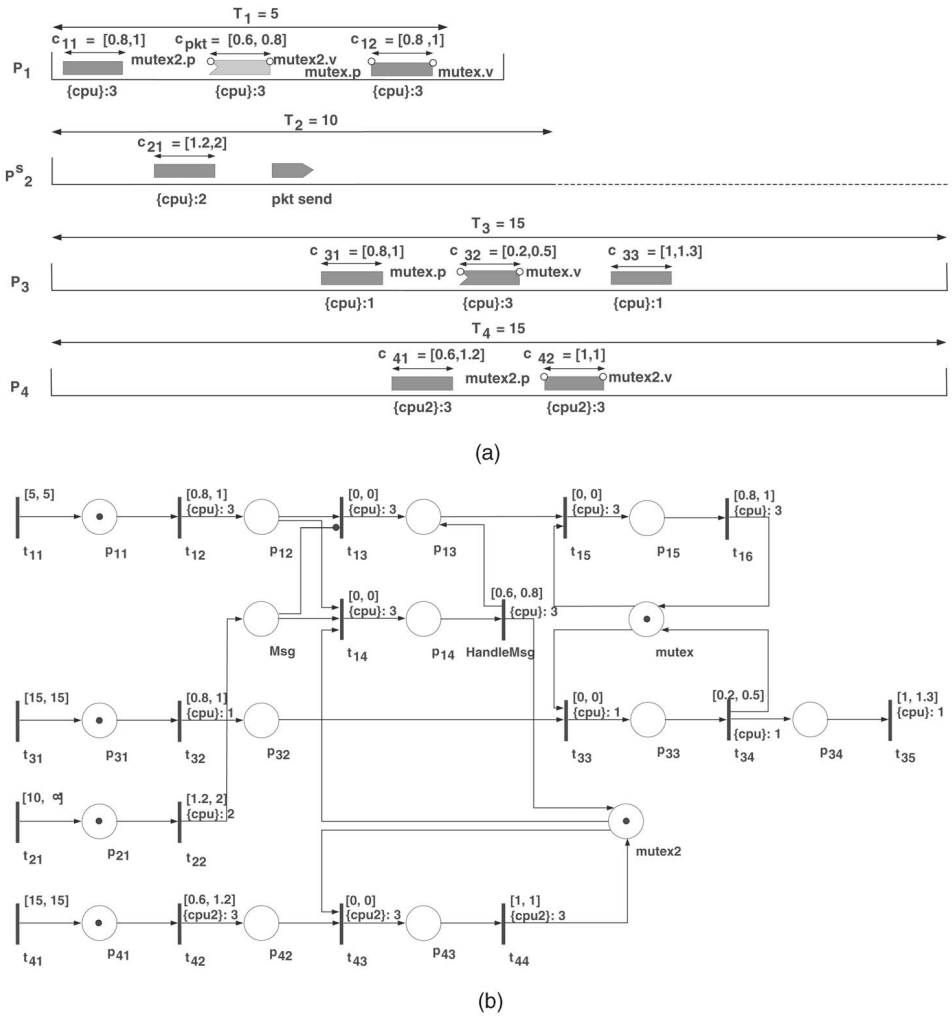

Fig. 5. (a) Each task released by P_2^s sends a message to P_1. In turn, P_1 performs an additional step (lightly filled) lasting $[0.6, 0.8]$ time units to consume each received message. (b) The PTPN model of the tasking set.

Estimation algorithms of [29] can manage mutual exclusion semaphores. If the sporadic process P_2^s is replaced through the nonequivalent periodic process P_2', the algorithms produce exact estimates for the completion time of processes P_1 and P_3, but it overestimates the completion time of P_2' (the bound is 9.6, while the exact value is 4.8).

4.3 Internal Data Flow, Message Passing, and Multiple Processors

In the tasking set of Fig. 5, tasks have been decomposed in sequences of steps, and a precedence constraint due to a message exchange has been added between steps in processes P_1 and P_2^s. Also, a new process P_4 running on a second processor $cpu2$ has been added. Process P_4 requires a second mutual exclusion semaphore $mutex2$ which is also used by process P_1 in the handle of messages received from P_2^s.

The resulting class graph includes 5,026 classes, with 3,707, 1,738, 18,024, and 4,080 traces for processes P_1, P_2^s, P_3, and P_4, which include 1,124 false behaviors for process P_3 and none for all the other processes.

Timeliness analysis indicates that the longest latency that P_1 can experience in the acquisition of $mutex$ is .5 time units (see Fig. 6a), which compares against the 2.8 time units that could occur in the model of Fig. 3. However, process P_1 can have .8 additional time units for the service routine managing the message from P_2^s and one additional time unit for acquisition of $mutex2$ needed to manage the message, thus resulting in a worst completion time of 4.3 time units. Whereas, the worst case of P_3 increases to 12.4 time units resulting from a double preemption by process P_1 (see Fig. 6b).

It is worth noting that the worst response time for process P_1 is realized when the first computation step of process P_4 (i.e., transition t_{42}) takes one time unit. The same response time could not be observed if the first computation step of process P_4 would be characterized through either its best or its worst computation time (i.e., if the firing interval of transition t_{42} would be reduced to a deterministic value equal to either 0.6 or 1.2). The anomaly, which is well known in the theory of resource reclaiming for real-time scheduling [6], depends on the fact that the system includes dependent processes running on multiple processors. Under these condition, early completion of a computation can change the relative starting time of computations on different processors.

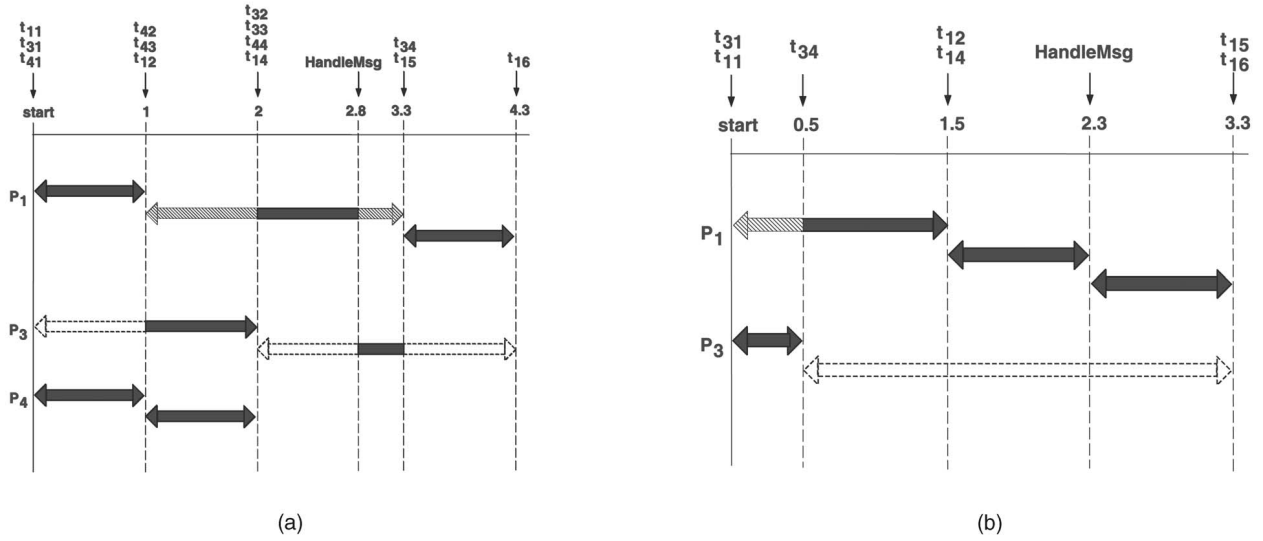

Fig. 6. (a) A run of the model of Fig. 5b yielding the longest completion time for process P_1: P_1 is blocked on both semaphors, and the duration of the first block on $mutex2$ is maximized. (b) If the first computation step of process P_4 is forced to take its possible maximum duration, the block of P_1 on $mutex2$ is avoided, and P_1 completes within 3.3 time units.

4.4 Complexity

For all the above examples, the state space is covered with a few thousand state classes and traces, which impose limited memory and time requirements on the offline analysis: On a Pentium 4 with 1,500 MHz clock, in the most demanding example discussed, state space enumeration terminates within less than two seconds, while the exact profiling of all the traces of each process requires less than one minute. Peak memory required is always lower than 16Mbytes.

However, as for any state space method, scalability of the approach is limited by the state space explosion problem, which indeed we also addressed in [13]. Factors having a higher impact are: the number of processes and, in particular, those with sporadic arrivals; the ratio between the minimium temporal parameter and the hyperperiod of the tasking set; the relative variability for nondeterministic times. When the model scales up, the knee for analyzability is encountered in the enumeration and clean-up of traces, depending on the complexity of traces and their number.

Since traces are enumerated in depth-first order, memory usage is linear in the length of the longest trace and in the output degree of classes, which is upper bounded by the number of net transitions that can be progressing at the same time. Besides, the length of the longest trace depends on the number of events that can occur within the stimulus and the response under analysis. When these coincide with the release and completion of a task, the number of events is linear with respect to the ratio between the longest and the shortest period among all the processes, unless deadlines are missed.

If the analysis is oriented only to derive the worst-case response time, the exact time profile does not have to be computed for all the enumerated traces. In fact, an overapproximate duration can be obtained in linear time with respect to the length of the trace [30], while the exact time profile is computed only for those traces whose approximate duration exceeds the longest response time encountered so far.

5 Conclusions

Preemptive Time Petri Nets extend Time Petri Nets with an additional mechanism of resource assignment which makes the advancement of computations be dependent on the availability of a set of preemptable resources. This results in a formalism supporting convenient modeling of complex real-time systems running under preemptive scheduling, with periodic, sporadic and event-triggered task releases, with nondeterministic dense execution times, with multiple processors, with semaphore synchronizations and precedences deriving from interprocess communication and from sequential decomposition of individual tasks.

The capability to express preemptive behavior with suspensions results in a state space where timers range within complex polyhedra that cannot be exactly encoded using DBM data structures. To circumvent the complexity, we replace polyhedra with their tightest DBM overapproximation. The approximate representation of the state space supports identification of the traces where the model *can* exhibit critical behaviors with respect to requirements on the sequencing of events or on their timeliness. For potential critical behaviors, automated inspection of approximate state classes permits recovery of the complete set of constraints that exactly capture all the effects of the semantics of the model, including those deriving from the drift between suspended and progressing clocks. This permits derivation of tight bounds on the response time between events along a trace and clean-up of traces which are not actually feasible for the model.

Modeling convenience, expressive and analysis power of the proposed methods have been concretely exemplified with reference to a family of cases, which were designed so as to address the factors of complexity that can motivate

state space analysis with respect to lower-complexity schedulability analysis techniques. Discussion of the examples highlight the kind of support that the proposed technique can provide in verifying sequencing and timeliness requirements and in gaining insight into patterns of system behavior.

APPENDIX

Lemma 5.1. *For any two transitions t_α and t_β that are persistent in S', there exists a solution for set D'_{pers} of (13) such that $\tau(t_\alpha) - \tau(t_\beta) = C_{\alpha\beta}$ (i.e., $C_{\alpha\beta}$ is the tightest upper bound for the difference $\tau(t_\alpha) - \tau(t_\beta)$).*

Proof. The set D'_{pers} can be represented as an oriented graph, where vertices are unknown values and edges are differences between unknown values: A DBM inequality $\tau(t_\alpha) - \tau(t_\beta) \leq C_{\alpha\beta}$ places a constraint on the length of the edge from $\tau(t_\beta)$ to $\tau(t_\alpha)$; besides any non-DBM inequality $(\tau(t_\alpha) - \tau(t_\beta)) + (\tau(t_\gamma) - \tau(t_\delta)) \leq C_{\alpha\beta} + C_{\gamma\delta} - c_{\alpha\beta\gamma\delta}$ adds a constraint on the cumulative length of two nonconnected edges from $\tau(t_\beta)$ to $\tau(t_\alpha)$ and from $\tau(t_\delta)$ to $\tau(t_\gamma)$.

Non-DBM inequalities restrict only pairs of edges connecting a persistent with a progressing transition. However, for technical convenience, we extend non-DBM constraints to all the possible pairs of edges. To this end, we assume $c_{\alpha\beta\gamma\delta} = 0$ for any interpretation of indexes $\alpha\beta\gamma\delta$ which is not explicitly enlisted in (13). This corresponds to the addition of a (redundant) non-DBM constraint which is obtained by combining two of the inequalities (ij) through (xi). To this end, we assume $c_{\alpha\beta\gamma\delta} = 0$ for any interpretation of indexes $\alpha\beta\gamma\delta$ which is not defined in (13). This corresponds to completing the set D'_{pers} with the addition of a set of (redundant) non-DBM constraints obtained by linear combination of DBM inequalities (ij) through (xi) appearing in (13). The set of coefficients c can be proven to satisfy the following property:

$$\begin{aligned} &C_{\alpha\beta} + C_{\beta\gamma} + C_{\gamma\delta} - c_{\alpha\beta\gamma\delta} \geq C_{\alpha\delta} \\ &\forall \alpha, \beta, \gamma, \delta \text{ such that } t_\alpha, t_\beta, t_\gamma, t_\delta \in T^{pers}(S'). \end{aligned} \tag{21}$$

The proof is obtained by replacing coefficients C and c through coefficient B and then by resorting to the property of normality of B itself. Since the expression of C and c in terms of B is not regular, a number of different cases must be enlisted, making the proof quite tedious, albeit simple. For instance, in the relatively most complex case in which $\alpha\beta\gamma\delta = ixy*$, (21) takes the form $C_{ix} + C_{xy} + C_{y*} - c_{ixy*} \geq C_{i*}$, which becomes $B_{ix} + B_{*o} + Bxy + B_{y*} - B_{y*} - B_{*o} + B_{yo} \geq B_{io}$, which descends from $B_{ix} + B_{xy} + B_{yo} \geq B_{io}$.

In the graph-theoretical perspective that we set, the maximum acceptable value for the difference $t_\alpha - t_\delta$ is the minimum length for any connected path from δ to α.

If $C_{\alpha\delta}$ is not the tightest upper bound for the difference $t_\alpha - t_\delta$, the graph must include a connected path ρ from δ to α whose cumulative length is strictly lower than $C_{\alpha\delta}$. The path ρ must include at least a pair of edges with cumulative length constrained by an inequality which is not in DBM form. Otherwise, since coefficient C is a normal form (by (14)), the length of the path could not be strictly lower than that of the direct edge from δ to α. More specifically, ρ must include a subpath from δ' to α' (possibly coincident with δ and α, respectively) which visits at least two more vertices γ' and β' such that:

- The total length of the path is strictly lower than $C_{\alpha'\delta'}$.
- The first edge from δ' to γ' and the last edge from β' to α' have a cumulative constraint $C_{\alpha'\beta'} + C_{\gamma'\delta'} - c_{\alpha'\beta'\gamma'\delta'}$ set by an inequality not in DBM form.
- All the intermediate edges from γ' to β', visiting any sequence of intermediate vertices ϵ_n, have an individual constraint set by an inequality in DBM form.

This implies:

$$C_{\alpha'\delta'} > C_{\alpha'\beta'} + C_{\beta'\epsilon_0} + \sum_{n=1}^{N} C_{\epsilon_n\epsilon_{n-1}} + C_{\epsilon_N\gamma'} + C_{\gamma'\delta'} - c_{\alpha'\beta'\gamma'\delta'}. \tag{22}$$

In turn, since coefficient C is in normal form, the summation $C_{\beta'\epsilon_0} + \sum_{n=1}^{N} C_{\epsilon_n\epsilon_{n-1}} + C_{\epsilon_N\gamma'}$ is not lower than $C_{\beta'\gamma'}$, so that (22) implies $C_{\alpha'\delta'} > C_{\alpha'\beta'} + C_{\beta'\gamma'} + C_{\gamma'\delta'} - c_{\alpha'\beta'\gamma'\delta'}$, thus clashing with (21). □

Lemma 5.2. *Let ρ be the trace corresponding to a path in the class graph of a PTPN, leading from class S^o to class S^N through the execution of transitions $t_{\rho(k)}$ with $k = 0, N-1$.*

If $< \bar{\tau}_o, \ldots, \bar{\tau}_k, \ldots, \bar{\tau}_{N-1} >$ is a solution for the set of inequalities yield for trace ρ by (17)-(19), then there exists a state s^o contained in class S^o from which the PTPN model can execute the trace ρ visiting a sequence of states s^k contained in the classes S^k and dwelling for a time $\bar{\tau}_k$ in each visited state s^k.

Proof. We consider a state s^o with the marking of class S^o, in which each enabled transition t_a has a time to fire

$$\tau(t_a^o) = \sum_{h=o}^{last(t_a^o)} c_h(t_a)\bar{\tau}_h, \tag{23}$$

where $c_k(t_a)$ and $last(t_a^o)$ are defined as in (16) (i.e., $c_h(t_a)$ is either 1 or 0 whether t_a is progressing or suspended in the class S^h, while $last(t_a)$ is the index of the last class along the trace in which t_a is persistent with continuity starting from the initial enabling). Since $< \bar{\tau}_o, \ldots, \bar{\tau}_k, \ldots, \bar{\tau}_{N-1} >$ is a solution for the set of inequalities defined by (17) and (18), the times to fire of (23) can be easily proven to satisfy all the inequalities in the firing domain of class S^o. This proves that s^o is a state in S^o.

We now construct a run of the PTPN model, which originates from the state s^o, which updates times to fire of enabled transitions according to the semantics of PTPNs and which resolves the undeterminacy in the

selection of the time to fire of any newly enabled t_a transition according to the following strategy:

$$\tau(t_a^k) = \begin{cases} \sum_{h=k}^{last(t_a^k)} c_h(t_a)\bar{\tau}_h & \text{if } t_a^k \text{is fired along the trace} \\ LFT^s(t_a) & \text{if } t_a^k \text{ is not fired along the trace.} \end{cases} \quad (24)$$

To prove that this strategy for the determination of times-to-fire is consistent with the semantics of the model, it is sufficient to verify that the value assigned to $\tau(t_a^k)$ is neither lower than $EFT^s(t_a)$ or higher than $LFT^s(t_a)$. This is trivial for the case that t_a^k is not fired along the trace. In the opposite case, the proof descends from the observation that $< \bar{\tau}_o, \ldots, \bar{\tau}_k, \ldots, \bar{\tau}_{N-1} >$ satisfies inequalities in (17) and (18), and that for any transition t_i newly enabled in a class S^k, the coefficient b_{i*}^k is equal to $LFT^s(t_i)$.

We now prove that with the assumed strategy in the selection of times to fire of newly enabled transitions, the PTPN can run from state s^o following the trace ρ with a dwelling time $\bar{\tau}_n$ in each visited state s^n. By induction on index n, we assume that the net can execute transitions $t_{\rho(0)}$ through $t_{\rho(n-1)}$ dwelling for a time $\bar{\tau}_k$ in each visited state s^k, and we demonstrate that the net is then able to execute transition $t_{\rho(n)}$ after a dwelling time $\bar{\tau}_n$.

Since trace ρ is a path in the class graph, t_{ρ_n} is an outcoming edge from class S^n, which implies that t_{ρ_n} is enabled and progressing under the marking of s^n (which is equal to the marking of S^n). Thus, according to the semantics of PTPNs, $t_{\rho(n)}$ can fire after a time $\bar{\tau}_n$ iff the following two conditions are satisfied: 1) $\bar{\tau}_n$ is the time to fire of $t_{\rho(n)}$ in s^n and 2) no other enabled and progressing transition has a strictly lower value of the time to fire.

To prove the first condition, we observe that since $t_{\rho(n)}$ is fired along the trace ρ, its time to fire has been initially set equal to the value

$$\sum_{h=first(t_{\rho(n)})}^{n} c_h(t_{\rho(n)})\bar{\tau}_h,$$

where $first(t_{\rho(n)})$ is the index of the earliest state since which $t_{\rho(n)}$ has been persistent with continuity. From the arrival in the state $s^{first(t_{\rho(n)})}$ to the arrival in the current state s^n, the time-to-fire of $t_{\rho(n)}$ has been reduced by the cumulative time elapsed during the permanence in states in which $t_{\rho(n)}$ was progressing, which is equal to $\sum_{h=first(t_{\rho(n)})}^{n-1} c_h(t_{\rho(n)})\bar{\tau}_h$. According to this, the remaining time to fire for $t_{\rho(n)}$ in the state s^n is equal to $c_n(t_{\rho(n)})\bar{\tau}_n$. Since $t_{\rho(n)}$ is progressing in S^n, the coefficient $c_n(t_{\rho(n)})$ is equal to 1, and the time to fire of $t_{\rho(n)}$ in s^n is equal to $\bar{\tau}_n$.

To prove the second condition, we demonstrate that any transition t_z which is enabled and progressing in s^n has a time to fire not lower $\bar{\tau}_n$. Since we do not know whether t_z is fired along the trace ρ, the initial setting of its time to fire could have a different form according to the selection in (24). In any case, we can assume a lower bound on the initial setting as:

$$\tau(t_z^{first(t_z)}) \geq \sum_{h=first(t_z)}^{last(t_z^{first_z})} c_h(t_z)\bar{\tau}_h.$$

By reducing this value by the time elapsed in states from s^{first_z} to s^{n-1} in which t_z was progressing, the time to fire of t_z in s^n, that we denote as $ttf^n(t_z)$, is lower bounded as:

$$ttf^n(t_z) \geq \sum_{h=n}^{last(t_z^{first(t_z)})} c_h(t_z)\bar{\tau}_h.$$

Since t_z is progressing in class S^n, the coefficient $c_n(t_z)$ is equal to 1, we can conclude the proof writing:

$$ttf^n(t_z) \geq \bar{\tau}_n + \sum_{h=n+1}^{last(t_z^{first_z})} c_h(t_z)\bar{\tau}_h \geq \bar{\tau}_n = ttf^n(t_{\rho(n)}).$$

□

ACKNOWLEDGMENTS

This work was supported by the Italian Ministry for Instruction University and Research (MIUR) as a part of the PERF project (RBNE019N8N) funded under the FIRB program.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows.* Prentice Hall, 1993.
[2] K. Altisen, G. Goessler, and J. Sifakis, "Scheduler Modeling Based on the Controller Synthesis Paradigm," *Real Time Systems,* vol. 23, pp. 55-84, 2002.
[3] R. Alur and D.L. Dill, "Automata for Modeling Real-Time Systems," *Proc. 17th Int'l Colloquium on Automata, Languages, and Programming,* 1990.
[4] R. Alur, T.A. Henzinger, and P.-H. Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Trans. Software Eng.,* vol. 22, no. 3, Mar. 1996.
[5] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Times—A Tool for Modelling and Implementation of Embedded Systems," *Lecture Notes in Computer Science,* vol. 2280, 2002.
[6] B. Andersson and J. Jonsson, "Preemptive Multiprocessor Scheduling Anomalies," *Proc. Int'l Parallel (and Distributed) Processing Symp.,* pp. 12-19, 2002.
[7] G.S. Avrunin, J.C. Corbett, L.K. Dillon, and J.C. Wileden, "Automated Derivation of Time Bounds in Uniprocessor Concurrent Systems," *IEEE Trans. Software Eng.,* vol. 20, no. 9, 1994.
[8] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y.-S. Kim, I. Lee, and H.-L. Xie, "A Process Algebraic Approach to the Schedulability Analysis of Real Time Systems," *Real Time Systems,* vol. 15, no. 3, pp. 189-219, 1998.
[9] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: A Tool-Suite for Automatic Verification of Real-Time Systems," *Hybrid Systems III,* R. Alur, T.A. Henzinger, E.D. Sontag, eds., 1996.
[10] B. Berthomieu and M. Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 17, no. 3, Mar. 1991.
[11] B. Berthomieu and M. Menasche, "An Enumerative Approach for Analyzing Time Petri Nets," *Proc. IFIP Congress,* Sept. 1983.
[12] A. Bobbio, A. Puliafito, and M. Telek, "A Modelling Framework to Implement Preemption Poolicies in Non-Markovian SPNs," *IEEE Trans. Software Eng.,* vol. 26, no. 1, Jan. 2000.
[13] G. Bucci and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 21, no. 12, Dec. 1995.

[14] F. Cassez and K.G. Larsen, "The Impressive Power of Stopwatches," *Lecture Notes in Computer Science,* vol. 1877, pp. 138-152, Aug. 2000.
[15] J.C. Corbett, "Timing Analysis of Ada Tasking Programs," *IEEE Trans. Software Eng.,* vol. 22, no. 7, pp. 461-483, July 1996.
[16] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool KRONOS," *Hybrid Systems III,* R. Alur, T.A. Henzinger, E.D. Sontag, eds., 1996.
[17] D. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. CAV Methods for Finite State Systems Conf.,* 1989.
[18] E. Fersman, P. Pettersson, and W. Yi, "Timed Automata with Asynchronous Processes: Schedulability and Decidability," *Lecture Notes in Computer Science,* vol. 2280, 2002.
[19] M.G. Harbour, M.H. Klein, and J.P. Lehoczky, "Timing Analysis for Fixed Priority Scheduling of Hard Real Time Systems," *IEEE Trans. Software Eng.,* vol. 20, no. 1, Jan. 1994.
[20] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. the ACM,* vol. 20, no. 1, 1973.
[21] J. McManis and P. Varaiya, "Suspension Automata: A Decidable Class of Hybrid Automata," *Proc. Computer Aided Verification: Sixth Int'l Conf.,* June 1994.
[22] P. Merlin and D.J. Farber, "Recoverability of Communication Protocols," *IEEE Trans. Comm.,* vol. 24, no. 9, Sept. 1976.
[23] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, Apr. 1989.
[24] X. Nicollin, J. Sifakis, and S. Yovine, "Compiling Real-Time Specifications into Extended Automata," *IEEE Trans. Software Eng.,* vol. 18, no. 9, Sept. 1992.
[25] J.L. Peterson, "Petri Nets," *ACM Computing Surveys,* vol. 9, no. 3, Sept. 1977.
[26] K. Ramamritham and J.A. Stankovic, "Scheduling Algorithms and Operating System Support for Real Time Systems," *Proc. IEEE,* vol. 82, no. 1, Jan. 1994.
[27] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real Time Synchronization," *IEEE Trans. Computers,* vol. 39, no. 9, Sept. 1990.
[28] L. Sha, R. Rajkumar, and S.S. Sathaye, "Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real Time Systems," *Proc. IEEE,* vol. 82, no. 1, 1994.
[29] J. Sun, M.K. Gardner, and J.W.S. Liu, "Bounding Completion Times of Jobs with Arbitrary Release Times, Variable Execution Times, and Resource Sharing," *IEEE Trans. Software Eng.,* vol. 23, no. 10, Oct. 1997.
[30] E. Vicario, "Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* Aug. 2001.
[31] J. Xu and D. Parnas, "On Satisfying Timing Constraints in Hard Real Time Systems," *IEEE Trans. Software Eng.,* vol. 19, no. 1, Jan. 1993.
[32] R.M. Fricks, A. Puliafito, M. Telek, and K. Trivedi, "Applications of Non-Markovian Stochastic Petri Nets," *Performance Evaluation Rev.,* vol. 26, no. 2, 1998.
[33] W.M. Zuberek, "M-Timed Petri Nets, Priorities, Preemptions, and Performance Evaluation of Systems," *Advances in Petri Nets 1985,* 1986.
[34] W.M. Zuberek, "Preemptive D-Timed Petri Nets, Timeouts, Modelling and Analysis of Communication Protocols," *Proc. IEEE INFOCOM Conf. Computer Comm.: Global Networks,* pp. 721-730, 1987.

Giacomo Bucci is a graduate of the University of Bologna, Italy. From 1970 to 1982, he was with the University of Bologna. During 1975, he was a visiting researcher at IBM T.J. Watson Research Center, Yorktown Heights, New York. Since 1986, he has been a full professor at the University of Florence, Faculty of Engineering, where he teaches a course in computer architectures and a course in software engineering. Currently, he is the director of the Department of Control and Information Systems and the director of the Software Technologies Lab of the University of Florence. His current research interests include software development methodologies, specification and validation techniques for time-dependent systems, and computer performance evaluation. He is a member of the IEEE and the IEEE Computer Society.

Andrea Fedeli holds a doctoral degree in informatics engineering received from the University of Florence in 2000. He is a student in the PhD program on informatics and telecommunications engineering at the University of Florence. His research activities address methods for specification and correctness verification of time dependent systems. He is a member of the Software Technologies Lab of the University of Florence.

Luigi Sassoli holds a doctoral degree in informatics engineering received from the University of Florence in 2002. He is a student in the PhD program on informatics, multimedia and telecommunications engineering at the University of Florence. His research addresses formal description and analysis of real-time reactive systems. In particular, his current activity develops on state space enumeration of both dense and discrete timed models as well as on model checking and Markovian techniques for correctness verification and dependability evaluation. He is a member of the Software Technologies Lab of the University of Florence.

Enrico Vicario received a doctoral degree in electronics engineering and PhD degree in informatics and telecommunications engineering from the University of Florence in 1990, and 1994, respectively. He is a full professor in the Faculty of Engineering at the University of Florence. His present research activity addresses formal specification, validation, and performance evaluation of reactive and real-time systems, as well as methodologies for software design and development. In the years, he also worked on protocol engineering, content modelling and retrieval for image and video, visual languages and formalisms, usability engineering, and multimedia applications. He is a member of the steering committee of the Center for Communication and Media Integration of the University of Florence, and a member of the Software Technologies Lab of the University of Florence. He is a member of the IEEE and the IEEE Computer Society.