Some Issues in Real-Time Systems Verification Using Time Petri Nets

Pedro M. González del Foyo
pedro.foyo@ufpe.br
Universidade Federal de Pernambuco
Dept. of Mechanical Engineering
50670-420 Recife, PE, Brazil

José Reinaldo Silva
reinaldo@usp.br
Universidade de São Paulo
Dept. of Mechatronics and Mechanical Systems
05508-900 São Paulo, SP, Brazil

Time Petri Net (TPN) models have been widely used to the specification and verification of real-time systems. However, the claim that most of these techniques are useful for real-time system verification can be discussed, based on two assumptions: i) to be suitable for real-time systems verification, a technique must be able to check timing properties, both safe and behavioral, and ii) the underlying algorithm must be efficient enough to be applied to complex systems design. In this paper we will discuss the suitability of commonly accepted verification techniques that use model-checking as a verification framework and TPN as a description model. We present a new algorithmic approach that allows computation of end-to-end time between activities over an abstract state space model. The proposed approach needs to compute the abstract state space only once improving the efficiency of the verification process and turning it suitable for large problems. We also introduce a new sufficient condition for abstract states space to preserve the branching time properties that yields more compact graphs than the condition already used in actual approaches. The approach would fit a design environment also based on Petri Nets called GHENeSys (General Hierarchical Enhanced Petri Nets). The results obtained, using our verification approach are compared with similar available approaches.
Keywords: *formal verification, Time Petri Nets, real-time systems, abstract state space*

Introduction

In real-time systems the correctness depends on the output and also on the time in which this output is produced. Therefore, the verification of such systems requires the determination of quantitative temporal properties, besides qualitative properties associated with its logical correctness.

As stated in Wang, Deng and Wu (2000), a key requirement to verify real time systems is the end-to-end delay for task execution and the correct execution sequences for those tasks. Execution sequences are related to the logical correctness of the system while end-to-end delays allow the evaluation of quantitative temporal properties for the sequences generated.

Despite several achievements in formal verification for real-time systems, the intrinsic complexity of various framework environments dedicated to its verification remains forbiddingly high (Wang, 2004).

Usually, the verification problem of timed systems is exponentially more complex than their untimed counterpart (Wang, 2004). Consequently, formal verification techniques are not yet commonly used in real-time systems design. Since the work of Clarke and Emerson (1981), the automatic verification of systems has improved, especially in what concerns time constrained systems (Alur, Courcoubetis and Dill, 1993; Henzinger et al., 1992; Emerson Jutla and Sistla, 1993), based on the Timed Automata (TA) formalism (Alur and Dill, 1990).

Recently, techniques based on reachability analysis for TPNs have been developed – most of them based on the state class approach (Berthomieu and Menasche, 1983) – and inserted on several (time) extended Petri nets environments. Some of them are suitable to do performance evaluations, since the time is represented as a fixed delay or time durations (Ramchandani, 1974; Sifakis, 1980; Zuberek, 1980). However, those Petri nets models are not well suited to deal with real-time systems where the time duration of activities (or tasks) could not be predicted.

Other extended models (Aalst, 1993; Merlin and Faber, 1976; Ghezzi et al., 1991; Tsai, Yang and Chang, 1995; Cortes, Eles and Peng, 2003) are suitable to deal with real-time systems because of their ability to represent varying time durations of activities, by using timing intervals attached to transitions. Merlin's Time Petri Nets (TPN) belong to that class of system, and is the most widely used formalism for specification, analysis and verification of real-time systems (Berthomieu and Diaz, 1991; Yoneda and Ryuba, 1998; Virbitskaite and Pokozy, 1999; Lime and Roux, 2006; Hadjidj and Boucheneb, 2008; Yovine, 1996).

However, not all approaches using TPN's deal with the explicit determination of quantitative temporal properties – a critical issue in real-time systems design. In all classical TPN approach it is necessary to verify whether a property can actually be checked, despite the possibility to determine quantitative temporal properties. For instance, even in methods based on linear time, only safety properties could be verified, which is not interesting to real-time system design. Thus, Timed Computational Tree Logic (TCTL) proposed in Alur, Courcoubetis and Dill (1993) is the most used temporal logic for real-time systems (Wang, 2004). The efficiency of the verification process is also a key issue. For instance, the complexity of a TCTL model-checking algorithm (Alur, Courcoubetis and Dill, 1993) over Timed Automata – using the region graph approach – is PSPACE[1].

Virbitskaite and Pokozy (1999) implemented a model-checking algorithm for TCTL using TPN as the description formalism, based on the region graphs, just as in Alur, Courcoubetis and Dill (1993). The quantitative part of the algorithm is performed by adding a special transition associated with the time in the TCTL formula. Therefore, there would be an extra transition per formula to be checked. Even when a partial order reduction method is used to decrease the state-space size the complexity remains PSPACE.

The verification process will be discussed in the next section, using the concept of complexity presented in Alur, Courcoubetis and Dill (1993) as reference. In the next section it will be presented some commonly accepted techniques that use model-checking as a verification framework and TPN as description model to real-time system verification. Advantages and disadvantages of each approach will be presented in the following section where our proposal will be presented to construct an abstract state space to improve the verification process. Finally, in the last section some experimental results will be shown.

The Verification Process

Let us assume that TPN is fixed as the description model for the design of real-time systems. Also, it is assumed that the specification of desired properties for a given system modeled in TPN is also

Paper received 13 August 2010. Paper accepted 7 April 2011.
Technical Editor: Glauco Caurin

[1] PSPACE problems may incur memory consumption polynomials to the input sizes in bit counts.

clearly stated using some temporal logic (see section II in Wang (2004) for a survey).

In order to verify the given specification, a state-space for the TPN model must be first computed and represented (see section V in (Wang, 2004) for a survey in state-space representations). Next, a labeling algorithm should go over the state space, labeling the nodes with their holding sub-formulas, respectively, until the entire formula is mapped. The model-checker should answer *yes* only if the formula holds in at least one node (global) or in a specific node (local), and *no* otherwise. The complexity of the labeling algorithm is linear in the size of the graph that represents the system state-space.

Basic Definitions

Merlin's Time Petri Nets extend classical Petri Nets by inserting temporal intervals associated with transitions, which specify firing delay ranges for these transitions to fire. A formal definition for TPN nets is revisited in this section, including the notion of state and state class.

A Time Petri Net is a tuple (P, T, B, F, $M_{0,}$ SIM) where:

- P is a finite non-empty set of places p_i;
- T is a finite non-empty set of transitions t_i;
- B is the backward incidence function
  $B: P \times T \rightarrow N$;
- F is the forward incidence function
  $F: T \times P \rightarrow N$;
- M_0 is the initial marking function
  $M_0: P \rightarrow N$;
- SIM is a mapping called static interval
  $SIM : T \rightarrow Q^+ \times (Q^+ \cup \{\infty\})$;

where N is the set of non-negative integers and Q^+ is the set of positive rational numbers.

The static interval is composed of two positive rational numbers (EFT, LFT), where EFT represents the earlier firing time and LFT the latest firing time. Therefore, if a transition *t* becomes enabled at time τ, then *t* cannot be fired before (τ + EFT(t)) and no later than (τ+LFT(t)), unless it is disabled by the firing of another transition.

In TPN, the enabling condition of a transition is similar to the one in classic Petri Nets. Therefore, the "enabling condition" for a generic transition t_i will be:

$$(\forall p)\ (M(p) \geq B(p, t_i)); \tag{1}$$

The set of all transitions enabled by marking M will be denoted as *enb(M)*. It is an important issue to determine the transitions that were just enabled by the transition M (to start counting its enabling time). If marking M is preceded by a generic marking M′, that is, if $M' \xrightarrow{t} M$, the persistent transitions in M are defined as $pers(M)=\{t|t\in(enb(M')\cap enb(M)\}$, and the new enabled transitions as $new(M) = enb(M)\backslash pers(M)$. It is clear that $new(M) \subseteq enb(M)$.

Even when marking M enables a set of transitions, not all of them may be allowed to fire, due to the firing constraints of transitions (EFT's and LFT's). Thus, the "firing condition" depends on two conditions:

1) transitions must be enabled, as formally expressed by Eq. (1);

2) enabled transitions must not fire before its EFT; they must fire before or exactly at its LFT unless another conflicting transition fires before.

According to the second condition, a transition t_i, enabled by a marking M at absolute time τ, could be fired at the firing time θ iff θ is not smaller than the EFT of t_i and not greater than the smallest of the LFT's of all the transitions enabled by marking M, that is:

$$EFT(t_i) \leq \theta \leq \min(LFT(t_k))\ \ \forall t_k \in enb(M) \tag{2}$$

A concrete state of the system can be defined as a pair s = (M, V) where:

1. M is a marking,
2. V is a clock valuation function, V: $enb(M) \rightarrow R^+$.

An abstract state of the system can be defined as a pair S = (M; I) where:

- M is a marking,
- I is an interval set which is a vector of possible firing times. The size of this vector is given by the number of transitions enabled by marking M, which is $|enb(M)|$.

Abstract states represent a set of concrete states of the TPN model. The firing interval set is also known as the firing domain D.

The firing of transition t_i leads the system to another state, at time τ + θ, which will be denoted as:

$$S \xrightarrow{(t_i,\ \theta)} S' \tag{3}$$

$\Upsilon(s)$ denotes the set of all firing transitions in state *s*, and $\Upsilon(s) \subseteq enb(M)$.

The firing rule is the base to the computation of new states and to establishment of a reachability relation among them. The set of all states that are reachable from an initial state, or the set of firing schedules feasible from an initial state, characterize the behavior of the TPN. Similarly, the set of all reachable markings characterize the behavior of a Petri Net.

Let ω be a path in a TPN model $\aleph$.

$$\omega = s_0 \xrightarrow{t_0,\theta_0} s_1 \xrightarrow{t_1,\theta_1} \dots \xrightarrow{t_i,\theta_i} s_i$$

where the reachable state after *i* firings is $\omega^i = s_i$.

For a concrete state $s = (M_s, \theta)$ and a state class C = (M, D), $s \in C$ if exists an execution path ω such that:

- $s = \omega^n$
- $M_s = M$
- θ_n is consistent with D.

State Space Computation Issues for TPN

For real-time systems, dense time model (where time is considered in the domain R^+) is the unique feasible option, what brings to the problem of treating an infinite number of states. Two approaches are entitled to treat this state space: region graphs (Alur, Courcoubetis and Dill, 1993) and the state class approach (Berthomieu and Menasche, 1983).

As always, the objective of these representations is to yield a state-space partition that groups concrete states into sets of states with similar behavior respect to the properties been verified. That "sets of states" must cover the entire state space and must be finite in order to ensure the termination of the verification process.

Yovine (1996) and del Foyo and Silva (2008) show that a state-space partition created using the state class approach is more efficient than one created using region graphs. Also, state-space computation algorithm remains PSPACE.

Concerning the state-space partition, two proposals are considered in this work: one based on geometric regions (Yoneda and Ryuba, 1998) and a "traditional" method (Berthomieu and Diaz, 1991). Other partition methods are just derived from these basic approaches (Wang, Deng and Wu, 2000; Hadjidj and Boucheneb, 2008).

State classes are commonly represented by two elements: a marking and a time domain. When geometric region is used, the

firing sequence is also included instead of a single marking. This information is necessary to construct the state class graphs for the Computational Tree Logic (CTL) model checking.

However, each one of the basic partition methods would result in distinct state-space partitions. For instance, using geometric regions all time variables are referred to a global clock, while in the traditional method all time variables are referred to a local clock, initialized when the transitions are newly enabled. This difference leads to different state representations.

The representation of a state class in geometric regions is given by a tuple $s = \{\mu, I, \sigma\}$ where μ represents the state marking, I is a vector containing the time values on which all transitions where fired (until the state s is reached) referred to a global clock, and σ is the sequence of transitions fired to reach the target state s. Comparatively, in the traditional approach, a state class is a pair $s = \{\mu, I\}$ where μ represents the marking and I is a vector containing the time values for all transitions enabled in s referred to the time in which those transitions become enabled.

On both methods, a state class stands for a "set of states" that shares the same marking and time boundaries. Thus, an abstract state space of a TPN model is a contraction of its generally infinite concrete state space Σ. In this paper the definition of abstract state space follows the one stated in Hadjidj and Boucheneb (2008).

Definition 1 (*Abstract state space*): An abstract state space of a TPN model is defined as a state transition system $(\Omega, \rightarrow, C_0)$ where:

- Ω is a set of elements, called abstract states, representing a cover of Σ. Each abstract state is a nonempty set of states having the same marking,
- $C_0 \in \Omega$ is the initial abstract state such that $s_0 \in C_0$,
- $\rightarrow \subseteq (C \times T \times C)$ is the successor relation satisfying condition $EE^2 = SG \wedge EEr$, where:

$$SG: \forall s \in \Sigma, (s \xrightarrow{t} s') \Rightarrow (\exists s \in C, C \in \Omega), \exists C' \in \Omega \text{ s. t. } s' \in C' \wedge C \xrightarrow{t} C',$$

$$EEr: (C \xrightarrow{t} C') \Rightarrow \exists s \in C, \exists s' \in C', \text{ s. t. } s \xrightarrow{t} s'.$$

Relation $\rightarrow$ may also satisfy additional conditions as AE and EA:

$$AE : (C \xrightarrow{t} C')) (\forall s \in C; \exists s' \in C'; s \xrightarrow{t} s')$$

$$EA : (C \xrightarrow{t} C')) (\forall s' \in C'; \exists s \in C; s \xrightarrow{t} s')$$

The above definition establishes that all concrete states must be included in at least one abstract state according to condition (SG). However, even if well stated and successfully used, abstract state spaces do not ensure correctness for quantitative temporal properties determination.

As stated before, the end-to-end time between task executions is a key requirement in real-time systems verification. Since state transitions represent the completion of some task, the end-to-end time can be computed as the time elapsed between states in a sequence of state transitions, called execution path. Those computations are correct if all execution paths in the concrete state space have a matching execution path in the abstract state space and vice-versa, and the time boundaries in each time domain contain precisely the temporal behavior of the concrete states belonging to its state class.

In the following section, most commonly used abstract state spaces in verification approaches will be introduced.

[2] Conditions EE, AE and EA named after its transition state relations: Exists-Exists, All-Exists and Exists-All respectively.

Some Verification Approaches Based on Abstract State Spaces

Berthomieu and Diaz (1991) proposed an enumerative technique based on a State Class Graph (SCG) and directed to the determination of some TPN properties like reachability and boundness. They also showed the undecidability of those properties, giving some sufficient conditions for the boundness property. The SCG preserves only the linear time properties.

Yoneda and Ryuba (1998) proposed a sufficient condition under which the abstract state spaces could preserve branching time properties. According to them abstract state spaces preserve the branching time properties if the (AE) property holds. Such property was called *atomicity*, and State Class Graphs satisfying this property were called Atomic Geometric Region Graph (AGRG).

The ASCG (Yoneda and Ryuba, 1998) was obtained using the geometric regions and yields a graph that preserves the properties (EE), (EA) and (AE). Figure 1 shows an example of a TPN taken from Yoneda and Ryuba (1998).

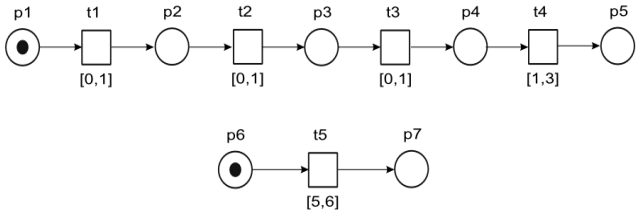

Figure 1. TPN Example.

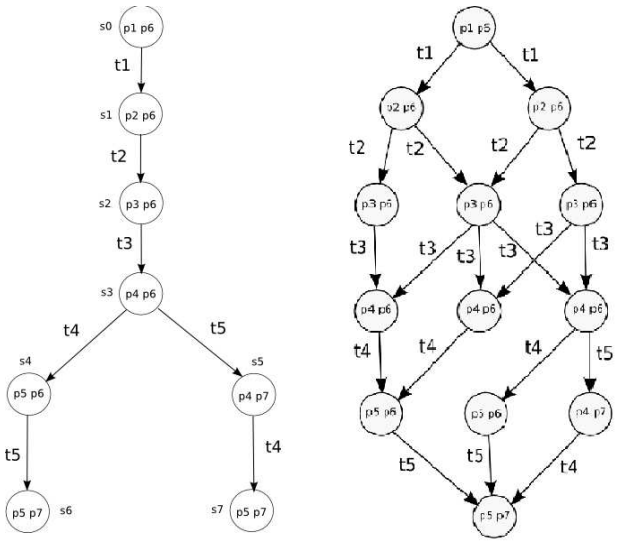

Figure 2. SCG and AGRG obtained using Yoneda and Ryuba approach.

Figure 2 shows the SCG and the AGRG obtained using Yoneda and Ryuba approach for the example in Fig. 1.

It should be noticed that approach proposed by Yoneda and Ryuba was idealized only for CTL model-checking. Thus, the complexity of the algorithm to compute the end-to-end time between transitions will be forbiddingly high to be implemented. Since each state class contains as much variables as the number of transitions fired to achieve it, the size of the system of inequalities will increase exponentially turning it into an intractable problem.

Berthomieu, Ribet and Vernadat (2004) implemented the state class approach in a software tool called TINA. Using TINA, the SCG and ASCG of a TPN can be obtained, but the property (EE) doesn't hold in the ASCG. The ASCG is obtained from the Strong

State Class Graph (SSCG), which is similar to the SCG, but more suitable to be a starting point for the refinement process.

In del Foyo and Silva (2008), an algorithm was proposed to compute the end-to-end time between task executions using the ASCG obtained using TINA. However, such technique only works for specific system behaviors where the refinement technique does not affect the (EE) property.

Figure 3 shows the ASCG with minimum and maximum time increments attached to every transition.

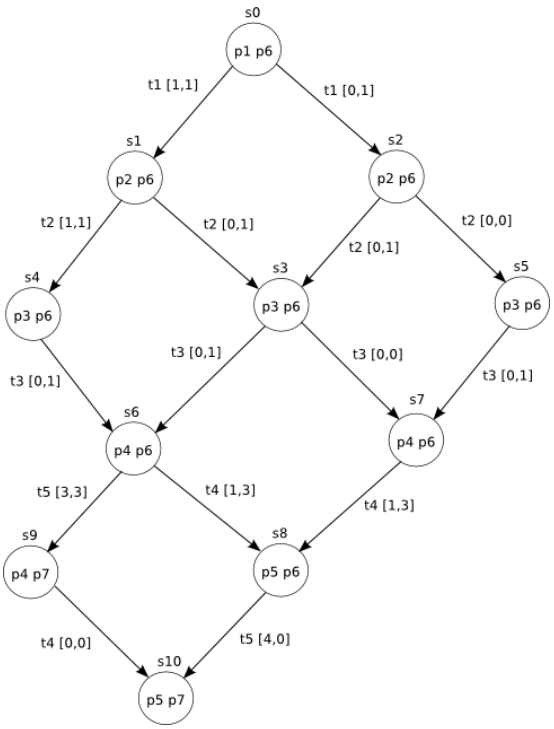

Figure 3. ASCG obtained using the approach in del Foyo and Silva (2008).

Suppose, for instance, that it is necessary to check the TCTL formula $EF_{\leq 4}$ (p5 $\wedge$ p7) over the graph in Fig. 3. The existence of a path {s0, s2, s3, s6, s9, s10} will force the verification algorithm to set the formula to true in state s0. However, looking at the TPN we can easily note that this formula should be evaluated to false, since there is no way transition t5 could fire before 5 time units.

Clearly, property (EE) does not hold in that graph. Notice that we can easily find an execution path accepted by the TPN model that is not covered by the ASCG.

$$S0 \xrightarrow[0,8]{t1} S2 \xrightarrow[0,\,85]{t2} S3 \xrightarrow[0,70]{t3} S7$$

The CS-class approach was proposed in Wang, Deng and Wu (2000) as a modification of the traditional state class approach to allow the computation of quantitative temporal properties.

A clock stamped state class (CS-class) is a tuple C = (M, D, ST) where:

1. M is a marking.
2. D is a firing domain, i.e., a set of constraints on the values of the time to fire for transitions enabled by current marking M.
3. ST is the time stamp of the CS-class, which is a (global) time interval.

By computing a CS-class reachability graph, it is possible to get a partition of the state space with its respective valid global time interval. If that graph was correctly computed, enough information would be obtained to determine the validity of some temporal logic statements including quantified temporal properties. Still, the CS-class only preserves the linear time properties.

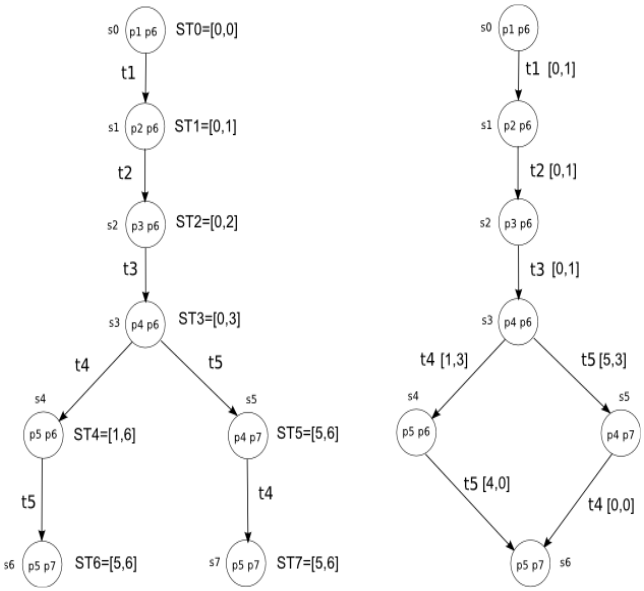

Figure 4. CS-class graph and LTS for TPN in Fig. 1.

Figure 4 shows two graphs: the reachability tree (in the left) obtained using the CS-class approach and the LTS (in the right) obtained by TINA, both for the TPN example in Fig. 1.

Despite the fact that there are different semantics in timing information (CS-class uses global time and LTS uses global time increments), the same kind of property can be verified in both graphs, even linear time properties, including quantitative temporal properties.

From the point of view of computational time, the choice of LTS time increments is better than the reachability tree (it would be equal in the worst case).

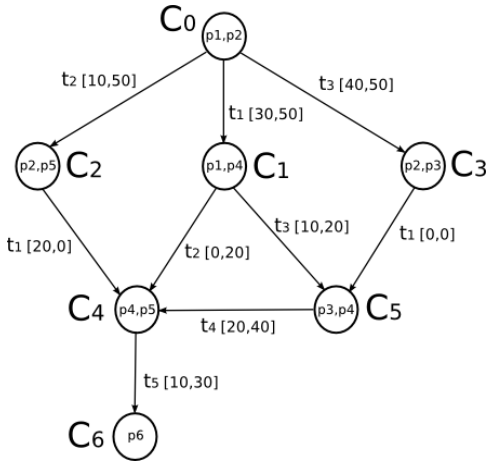

Figure 5. LTS of the TPN used in Wang, Deng and Wu (2000).

The LTS of the example shown in Wang, Deng and Wu (2000) is in Fig. 5. In the example published in Wang, Deng and Wu (2000) the reachability tree has 14 nodes while there are 7 in the LTS.

However, the LTS approach could lead to errors when evaluating quantitative temporal properties (del Foyo and Silva, 2008). Even when quantitative temporal properties can be checked, prospective results are not really useful in real-time systems verifications, since the linear time properties only allow the verification of safety properties.

Hadjidj and Boucheneb (2008) proposed a Concrete State Zone Graph (CSZG), getting results from a different normalization technique detached from the one used in Berthomieu and Vernadat (2003). The CSZG is used as a starting point for a convex combination process that yields a compact representation grouping state classes with the same marking and time domains, similar to the union operation in convex zones.

The convex combination will contract the state-space representation, but, at the same time, will introduce time solutions that do not appear in the TPN model. Such operation could lead to errors in the verification of quantitative temporal properties.

The refinement process to ensure the (AE) property yield a graph called atomic-CSZG which can be used to verify CTL*. Such verification is more efficient than the one in ASCG, according to experimental results showed in (Hadjidj and Boucheneb, 2008)

Lime and Roux (2006) also used TPN to model system behavior. They used the state class approach to build a Timed Automaton (TA) that preserves the behavior of the TPN using as less clock variables as possible. The resulting model is then verified using **UPPAAL** (Larsen, Petterson and Yi, 1997). However, even when **UPPAAL** can answer about quantitative temporal properties, it can only verify a subset of TCTL, what is a limitation in the system specification, especially in the expressibility of behavioral properties. Another drawback is the complexity of the algorithms: a PSPACE algorithm is executed just to convert a TPN in a TA; then another PSPACE algorithm is applied for each property to be checked.

The adding of a new transition to measure time elapsing was proposed in Boucheneb, Gardey and Roux (2006) to perform TCTL model-checking in TPN. Using that transition, TCTL formulas are translated into CTL formulas. Then a zone-based graph for TPN is refined using a partition refinement technique leading to a graph called Atomic Zone Based Graph that preserves CTL properties. Finally, a fix point algorithm is used to check the property. Such approach has PSPACE complexity for each property to be checked.

The final conclusion is that none of the methods discussed above improve significantly the approach in Virbitskaite and Pokozy (1999). Thus, the low computational efficiency of these approaches, including Virbitskaite and Pokozy (1999), or its limitation for systems specification, are the main reasons why existing model-checkers are barely used, even when most part of the academic community acknowledge that the TCTL is the better choice for real-time systems specification.

Even with its limitations in specification, **UPPAAL** is a widely used software tool for real-time systems verification, thanks to its balance between performance and expressiveness.

A Proposal to Improve the Verification Process

From the discussion above it is possible to conclude that the main problem in verification is the computational complexity. The construction of the abstract state space is PSPACE, and the labeling algorithm is PTIME, thus, there are few stages of the verification process that can be improved in a significant way. On the other hand, all mentioned approaches go through a process where the generation of the abstract state space is a key step. Therefore, a natural proposal to improve the verification process would be to compute the abstract state space only once and check every property with PTIME[3] complexity over this space.

First of all, the abstract state space must preserve branching time properties in order to verify the TCTL. Next, the timing information must be coded in a way that we can compute the minimum and maximum elapsed time in every path of the graph. In the following section we will show how to build a complete abstract state space as a general step in the verification process.

The constrained state class graph

The main idea is to build an abstract state space that preserves CTL* properties using a refinement procedure similar to the one used in Yoneda and Ryuba (1998), but using a different sufficient condition to preserve branching time properties.

To preserve the branching time properties, concrete states with different future must be grouped in different state classes. This goal can be achieved satisfying the (AE) property, but in this condition concrete states with same future can be grouped apart, resulting an excessive split process, and consequently increasing the number of state classes. Naturally, bigger partitions imply in worse complexity results for the labeling algorithm.

In this work we propose a new condition that is a necessary and sufficient condition to abstract state spaces to preserve branching time properties: the *stability condition*.

Definition 2 (*Stability*): A state class is *stable* if for any state $s \in C$, $\Upsilon(s) = \Upsilon(C)$. A state class graph G is stable, if every state class in G is stable.

Theorem 2 (*Branching time properties preservation*): An abstract state space preserves the branching time properties iff it is stable.

Proof:

($\Rightarrow$) This follows from the definition of abstract state space (property EE) and from the definition of stability (Definition 2).

($\Leftarrow$) Let G be an abstract state belonging to the abstract state space Ω with at least one unstable state class C'. Let ω be a path in the concrete state space $\sum$ passing through s', with $s' \in C'$ (by EE).

Let $s_1,\ldots,s_i$ be successors of state s' after firing different transitions. By property (EE) those states must be in some successors of C', denoted $C_1 \ldots C_j$. Once C' is unstable, $i \neq j$, and we can conclude that G do not satisfy (EE), or, path ω doesn't have a match in G, which means that property (EE) fails in G, which is a contradiction.

The proper constraints used here to split unstable state classes are quite similar to those identified in Yoneda and Ryuba (1998) to split non-atomic state classes. Different sets of state class firing transition are related to the EFT and LFT: the new enabled and the persistent enabled transitions.

The constraint for the unstable state class *s'* has the form:

$$s' - s_k < EFT(t') - LFT(t) \quad \text{or} \quad s' - s_k > LFT(t') - EFT(t)$$

where $t, t' \in enb(M')$, $t \in new(M')$ and $t' \in new(M_k)$

Since the constraint is a straight line in the state class domain of (t', t), the splitting of the convex state class domain (time domains must be convex since they are represented using Difference Bounded Matrix (Dill, 1989)) will result in two convex time domains, one for each resulting state class. The process must be applied to the new state classes if they do not satisfy the stability condition.

[3] PTIME problems can be solved with time complexity polynomials to the input sizes in bit counts.

Applying a constraint to an unstable state class does not increment the number of variables in the time domain. Transitions involved in the constraint are in the unstable time domain and, therefore, there is no need to add any variable.

Since the CSZG (Algorithm 1 in Hadjidj and Boucheneb, 2008) satisfies properties (EE) and (EA) by construction, it would be advisable to start the process constructing the Constrained State Class Graph (CSCG).

Every time a new state class is computed (using Algorithm 1 in Hadjidj and Boucheneb, 2008) the stability condition must be tested. If the test fails, the state class must be split using an adequate time constraint. This algorithm is repeated for the classes created in the split process until the point where is not possible to generate a new state class.

The proof that the CSCG is finite for bounded TPN is straightforward. Figure 6 shows the CSCG for the TPN in Fig. 1. Applying the stability condition over the atomicity condition yields graph with fewer states (see AGRG in Fig. 2. and ASCG in Fig. 3).

The abstract state space satisfies (EE), since the refinement process does not add or eliminate any concrete state. Also, this abstract state space is stable and consequently preserves the branching time properties.

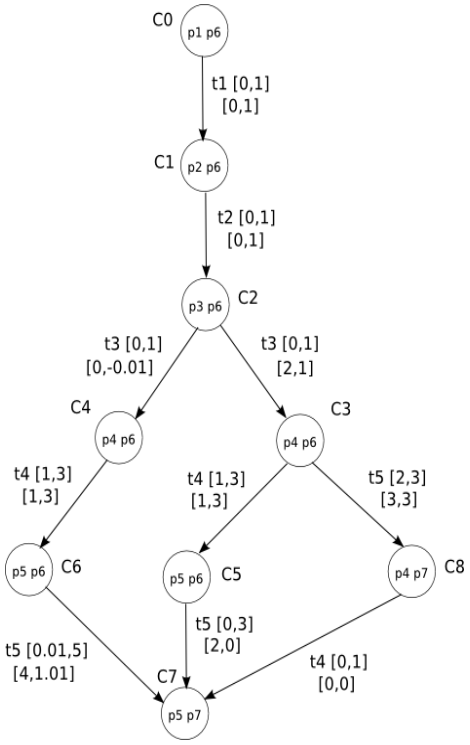

Figure 6. Constrained State Class Graph for the TPN in Fig. 1.

Computing minimum and maximum elapsed time over paths in CSCG

For the verification of quantitative temporal properties, the abstract state space must allow the computation of a minimum and a maximum elapsed time over any path. Using an algorithm similar to the one in del Foyo and Silva (2008), global time increments can be determined (See Fig. 7).

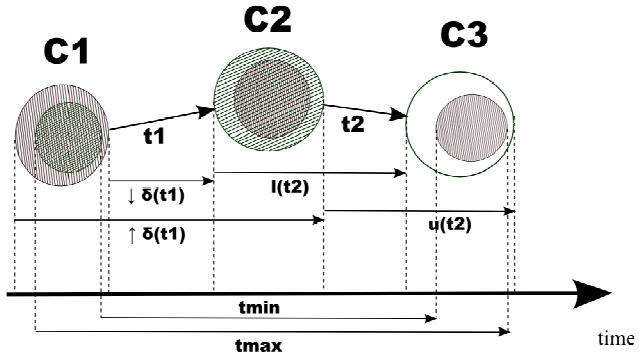

Figure 7. Minimum and maximum time determination through paths.

Figure 7 shows a path C1 → C2 → C3 in an abstract state space that clearly preserves properties (EE) and (EA). Let δ(t1) be the firing time interval in state transition C1 → C2 and l(t2), u(t2) the global time increment in state transition C2 → C3. Even when we can't determine exactly the minimum and maximum time between concrete states belonging to C1 and C3 respectively, we can ensure that $(\forall s \in C1)\ t_{min} \geq \downarrow\delta(t1) + l(t2)$ and $(\forall s \in C1)\ t_{max} \leq \uparrow\delta(t1) + u(t)$ where $\downarrow\delta(t1)$ and $\uparrow\delta(t1)$ are respectively the lower and upper limits of the firing time interval of transition t1.

Let ω be a valid execution path over an abstract state-space that preserves properties (EE) and (EA):

$$\omega = s_i \xrightarrow{\delta(t_i),[l_i,u_i]} s_{i+1} \xrightarrow{\delta(t_{i+1}),[l_{i+1},u_{i+1}]} s_{i+2} \ \ldots\ s_{i+n-1} \xrightarrow{\delta(t_{i+n}),[l_{i+n},u_{i+n}]} s_n$$

Then,

$$tmin(\omega) = \sum_{x=i+2}^{i+n} l_x + \downarrow\delta(t_i) \qquad tmax(\omega) = \sum_{x=i+2}^{i+n} u_x + \uparrow\delta(t_i)$$

Therefore, we can use tmin(ω) and tmax(ω) to verify quantitative temporal properties.

For instance, let us check again the TCTL formula $EF_{\leq 4}(p5 \ni p7)$, this time over the graph in Fig. 1. The TCTL formula does not hold in state C0 since starting in C0 and ending in C7 takes at least 5 time units. However, the same formula also does not hold in C1, what is wrong. Notice that by sure the minimum time elapsed in some path is greater or equal to the time computed using the time interval and the global time increments through the path. To obtain exactly the minimum time elapsed in a path we need an abstract state space that satisfies (EA) and (AE) at the same time, which is difficult to obtain (and yields a large partition). We claim that the approach used in CSCG can provide adequate results if is possible to assure that most of the concrete states in a state class have successors inside the respective state class successors. That condition holds in the CSZG but was affected by the refinement process.

When a constraint is applied to split an unstable state class, the relation of concrete states between the new state classes and its predecessors is affected. Thus, the predecessor state class must be split too. The described operation can be seen as restoring the (AE) property in the Yoneda and Ryuba approach, which in fact it is less restrictive, and yields better state space partitions.

We claim that an abstract state space satisfying those conditions can be used to verify the TCTL without re-computing all state space for every formula to be checked. Such achievement could lead to a significant improvement in real time system verification.

Experimental Results

In this section we present a comparison of our approach with similar approaches available in academic publications. The experiments were conducted using the same models used in Yoneda and Ryuba (1998) and showed in Fig. 8.

Using the code developed by Yoneda and Ryuba we built the Atomic Geometric Region Graph (AGRG). The tool TINA v2.9 was used to built the Atomic State Class Graph (ASCG). An implementation of the algorithm proposed in this work was used to build the Constrained State Class Graph (CSCG).

Table 1 reports the size (nodes and arcs) of the abstract state spaces obtained for those TPN models using the tree approaches mentioned above and the region graph representation (Alur, Dill, and Courcoubetis, 1993).

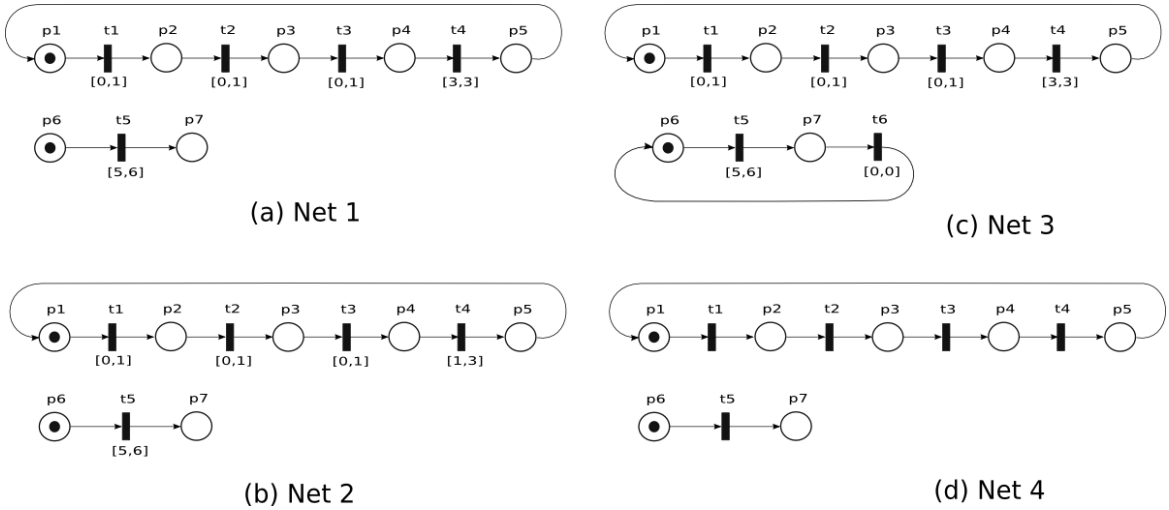

Figure 8. Time Petri Nets used in experiments by Yoneda and Ryuba (1998).

Table 1. Comparison of AGRG, ASCG, CSCG and the Region Graph approach.

Examples		AGRG	ASCG	CSCG	Region Graph
Fig. 8(a)	nodes	53	27	39	215
	arcs	95	49	53	348
	CPU time (s)	0.034	0.001	0.014	-
Fig. 8(b)	nodes	64	47	51	249
	arcs	178	140	70	466
	CPU time (s)	0.357	0.004	0.026	-
Fig. 8(c)	nodes	168	80	121	296
	arcs	363	204	171	469
	CPU time (s)	6.360	0.004	0.063	-
Fig. 8(d)	nodes	53	27	39	15011
	arcs	95	49	53	25206
	CPU time (s)	0.040	0.001	0.015	-

The result confirms that the CSCG yields better state space partitions than the geometric region approach by Yoneda and Ryuba. Taking into account the CPU time to obtain those partitions, the CSCG achieves better performance even when Yoneda and Ryuba implementations used C++ and our algorithm was implemented using Java.

Comparing with the ASCG, and taking in account that TINA is a reference system in the area of real time verification – there is not a prevalence of one approach over the other. Consistently, ASCG generates fewer nodes, but CSCG generated fewer arcs in Net Examples 2 and 3 (about 50% and 30% less arcs, respectively). In the Net Example 2 TINA performance is still better, but this situation is inverted in Net Example 3. This is not a surprise, since the ASCG does not include all paths in the TPN model; the CSCG can sometimes lead to better partitions (like the one in Fig. 8 (b)) than the ASCG, due to the stability condition.

However, in what concerns the compatibility with the overall design process, to generate the whole state space and consider all possible paths makes the analysis more secure, even if sometimes it compromises the performance. Of course that it does not mean that TINA analysis is not useful, but for real-time systems the ability to determine quantitative temporal properties is critical and should be taken into account to choose the verification approach. For this kind of systems the proposal presented here should be more advisable.

Conclusion

The verification process is evolving since the work of Clark and Emerson (1981), but still it does not appear an approach efficient enough to verify quantitative temporal properties in branching time temporal logics. Existing approaches were used for linear-time temporal logic, but cannot be successfully applied to branching-time temporal logic.

Current solutions to verify branching-time temporal logic include only subsets of logic as in Larsen, Petterson and Yi (1997) and Boucheneb, Gardey and Roux (2006). There are also solutions that use the region graph approach (Alur, Courcoubetis and Dill, 1993; Virbitskaite and Pokozy, 1999). The low efficiency of these methods, or limitations in the specification method, are the reasons why existing model-checkers are barely used, even if there is almost a consensus in the academic community that the TCTL is the better choice for real-time system specification.

In this work we showed that obtaining an abstract state space that satisfies (EE)/(EA) properties and the stability condition will increase the efficiency of the verification process without any restriction to the specification power of the TCTL.

We also believe that such partition could be obtained taking the CSCG as a start point.

Acknowledgements

We thank CAPES for the partial support to the research that resulted in this article.

References

Aalst, W., 1993, “Interval Timed Coloured Petri Nets and their Analysis”, in: Application and Theory of Petri Nets 1993, M.A. Marsan, Ed., Vol. 691. Springer-Verlag, Berlin, pp. 453-472.

Alur, R., Courcoubetis, C. and Dill, D.L., 1993, “Model-checking in dense real-time”, *Information and Computation*, Vol. 104, No. 1, pp. 2-34.

Alur, R. and Dill, D., 1990, “Automata for modeling real-time systems”, Lecture Notes in Computer Science, Vol. 443, pp. 322-335.

Berthomieu, B. and Diaz, M., 1991, “Modelling and verification of time dependent systems using time Petri nets”, IEEE Trans. on Software Engineering, Vol. 17, No. 3, pp. 259-273.

Berthomieu, B. and Menasche, M., 1983, “An enumerative approach for analyzing time Petri nets”, in: Information Processing: proceedings of the IFIP congress 1983, R.E.A. Mason, Ed., Vol. 9. Elsevier Science Publishers, Amsterdam, pp. 41-46.

Berthomieu, B., and Vernadat, F., 2003, “State class constructions for branching analysis of time Petri nets”, Lecture Notes in Computer Science, Vol. 2619, pp. 442-457.

Berthomieu, B., Ribet, P.O. and Vernadat, F., 2004, “The tool TINA - construction of abstract state spaces for petri nets and time petri nets”, *Int. J. Prod. res.*, Vol. 42, No. 14, pp. 2741-2756.

Boucheneb, H., Gardey, G. and Roux, O.H., 2006, “TCTL model checking of Time Petri Nets”, IRCCyN Technical report number RI2006-14.

Clarke, E.M. and Emerson, E.A., 1981, “Design and synthesis of synchronization skeletons using branching time temporal logic,” in: Proc. Workshop on Logics of Programs, Vol. 131, Berlin: Springer Verlag, pp. 52-71.

Cortes, L.A., Eles, P. and Peng, Z., 2003, “Modeling and formal verification of embedded systems based on a Petri net representation”, Journal of Systems Architecture, Vol. 49, pp. 571-598.

Dill, D.L., 1989, “Timing assumptions and verification of finite-state concurrent systems”, in Automatic Verification Methods for Finite State Systems, pp. 197-212.

Emerson, E.A., Jutla, C.S. and Sistla, A.P., 1993, “On model-checking for fragments of μ-calculus”, in Computer Aided Verification, pp. 385-396.

del Foyo, P.M.G. and Silva, J.R., 2008, “The verification of real time systems using the Tina tool,” in: Proceedings of the IFAC World Congress, Seul, Korea, pp. 525-533.

Ghezzi, C., Mandrioli, D., Morasca, S. and Pezze, M., 1991, “A unified high-level Petri net formalism for time-critical systems”, *IEEE Trans. on Software Engineering*, Vol. 17, No. 2, pp. 160-172.

Hadjidj, R. and Boucheneb, H., 2008, “Improving state class constructions for CTL* model checking of time Petri nets”, STTT, Vol. 10, No. 2, pp. 167-184.

Henzinger, T., Nicollin, X., Sifakis, J. and Yovine S., 1992, “Symbolic Model Checking for Real-Time Systems”, in 7th. Symposium of Logics in Computer Science. Santa-Cruz, California: IEEE Computer Scienty Press, pp. 394-406.

Larsen, K.G., Pettersson, P. and Yi, W., 1997, “UPPAAL in a Nutshell”, *Int. Journal on Software Tools for Technology Transfer*, Vol. 1, No. 1-2, pp. 134-152.

Lime, D. and Roux, O.H., 2006, “Model checking of time Petri nets using the state class timed automaton”, *Discrete Event Dyn. Syst.*, Vol. 16, pp. 179-206.

Merlin, P. and Faber, D., 1976, “Recoverability of communication protocols–implications of a theoretical study”, *IEEE Transactions on Communications* [legacy, pre-1988], Vol. 24, No. 9, pp. 1036-1043.

Ramchandani, C., 1974, “Analysis of asynchronous concurrent systems by timed Petri nets”, Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Report.

Sifakis, J., 1980, “Performance evaluation of systems using nets”, in: Proceedings of the Advanced Course on General Net Theory of Processes and Systems. London, UK: Springer-Verlag, pp. 307-319.

Tsai, J.J.P., Yang, S.J. and Chang, Y.H., 1995, “Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications”, *IEEE Trans. on Software Engineering*, Vol. 21, No. 1, pp. 32-49.

Virbitskaite, I. and Pokozy, E., 1999, “A partial order method for the verification of time Petri nets”, in: FCT, G. Ciobanu and G. Paun, Eds., Vol. 1684. Springer Verlag, pp. 547-558.

Wang, J., Deng, Y. and Xu, G., 2000, “Reachability analysis of real-time systems using time petri nets”, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 30, No. 5, pp. 725-736.

Wang F., 2004, “Formal verification of timed systems: A survey and perspective”, Proceedings of the IEEE, Vol. 92, No. 8, pp. 1283-1305.

Yoneda T. and Ryuba, H., 1998, “CTL model checking of time Petri nets using geometric regions”, *IEICE Trans. on Information and Systems*, Vol. E81-D, No. 3, pp. 297-396.

Yovine, S., 1996, “Model checking timed automata”, in: European Educational Forum: School on Embedded Systems, pp. 114-152.

Zuberek, W.M., 1980, “Timed petri nets and preliminary performance evaluation”, in: ISCA ’80: Proceedings of the 7th annual symposium on Computer Architecture. New York, NY, USA: ACM Press, pp. 88-96.