Real-Time Property Specific Reduction for Time Petri Net

Ning Ge

LAAS-CNRS
7 Avenue du Colonel Roche, Toulouse
Ning.Ge@laas.fr

Marc Pantel

University of Toulouse, IRIT-CNRS
2 Rue Charles Camichel, Toulouse
Marc.Pantel@enseeiht.fr

Abstract. This paper presents a real-time property specific reduction approach for Time Petri Net (TPN). It divides TPN models into sub-nets of smaller size, and constructs an abstraction of reducible ones, which exhibits the same property specific behavior, but has less transitions and states. This directly reduces the amount of computation needed to generate the whole state space. This method adapts well to the verification of real-time properties in asynchronous systems. It should be possible to apply similar methods to other families of properties.

Keywords: Real-time property specific reduction, Time Petri net

1 Introduction

The key issue that prevents a wide application of model checking in the industry is the scalability with respect to the size of the target system. A realistic system usually has thousands and even millions of states and transitions. Although a huge part of impossible firing sequences of transitions are eliminated during the building of system's behavior, the interleaving of all others is still a very large number that will easily lead to combinatorial state space explosion. Classic verification methodologies usually encounter scalability issues very quickly along with the growth of system scale, because they follow an implicit purpose: many different kind of properties will be assessed relying on the same state space graph (reachability graph). Indeed, once the reachability graph has been generated, it can be reused to verify different kinds of properties, just by revising the assessed logic formulas. This consideration requires to build the reachability graph preserving precise and sufficient information for the assessment of properties. The existing state space reduction methods, partial order reduction [1, 2], compositional reasoning [3, 4], symmetry [5, 6], abstraction techniques [7], on-the-fly model checking [8, 9], etc., usually follow the same philosophy to produce a complete state space that preserves the mandatory semantics. These generic reduction methods have effectively improved the efficiency of model checking techniques. But their improvement is becoming more and more difficult. We thus might put aside the universality of the semantics expressed in the state space graph, and take into account property specific reduction methods.

This work proposes a real-time property specific state space reduction approach for Time Petri Net (TPN). It divides the TPN model into sub-nets of

smaller size, and constructs an abstraction of reducible sub-nets, which exhibit the same property specific behavior, but has less transitions and states. The real-time property specific behavior (called real-time behavior for short in the following parts) of TPN sub-nets is an abstraction of the whole state-transition traces that only preserves real-time behaviors from the viewpoint of observations. This method adapts well to the verification of real-time properties in asynchronous systems. It could be possible to apply similar methods to other families of properties.

This paper is organized as follows: Section 2 presents some related works; Section 3 introduces real-time properties and Time Petri Net; Section 4 gives an overview of property specific reduction methods; Section 5 defines two real-time behavior regularities for this work; Section 6 details the proposed reduction method; Section 7 provides experimental results; Section 8 discusses the behavior coverage issue; Section 9 gives some concluding remarks.

2 Related Works

Several existing works [10–13] defined reducible sub-net patterns for Petri nets, Time Petri nets or Colored Petri nets, based on the idea of fusing redundant places and transitions. They provide in fact simple behavior equivalent patterns. The state space reduced by these patterns is rather limited.

The idea of our approach is similar to the partial order reduction [14, 2] and the state space abstraction techniques applied in the TINA toolset.

The partial order reduction is usually used in asynchronous concurrent systems, where most of the activities in different processes are performed independently, without a global synchronization. Its main idea is to construct a reduced state class graph by analyzing the dependencies between the transitions and exploiting the commutativity of concurrently executed transitions, which result in the same state when executed in different orders. A set of non-reducible transitions are preserved in the reduced state class graph. The reduced behavior is a subset of the behavior of the full state class graph. Compared to the partial order reduction, the proposed property specific reduction exploits the commutativity of TPN sub-nets, which result in the same property specific behavior.

The TINA toolset provides various state space abstractions for TPN when generating state class graphs, following the techniques proposed in [15, 9]. Depending on the abstraction options, the construction can preserve the traces required by the verification of markings, states, LTL, or ctl* properties. This work relies on the state class graph preserving markings to verify the real-time properties in TPN. Even with this highest abstraction, the state space still rapidly increases along with system scale. Therefore, more abstract state class graphs dedicated to one type of properties (in our case real-time properties) is needed.

3 Preliminaries

3.1 Time Petri Net

Time Petri nets [16] extends Petri Nets with timing constraints on the firing of transitions. Here we use the formal definition of TPN from [17] to explain its syntax and semantics.

Definition 1 (Time Petri Net). *A Time Petri Net (TPN) $\mathcal{T}$ is a tuple $\langle P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta)\rangle$, where:*

- $P = \{p_1, p_2, ..., p_m\}$ *is a finite set of places;*
- $T = \{t_1, t_2, ..., t_n\}$ *is a finite set of transitions;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the backward incidence mapping;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the forward incidence mapping;*
- $M_0 \in \mathbb{N}^P$ *is the initial marking;*
- $\alpha \in (\mathbb{Q}_{\geq 0})^T$ *and* $\beta \in (\mathbb{Q}_{\geq 0} \cup \infty)^T$ *are respectively the earliest and latest firing time constraints for transitions.*

Following the definition of enabledness in [18], a transition t_i is enabled in a marking M iff $M \geq {}^\bullet(t_i)$ and $\alpha(t_i) \leq v_i \leq \beta(t_i)$ (v_i is the elapsed time since t_i was last enabled). There exist a global synchronized clock in the whole TPN, and $\alpha(t_i)$ and $\beta(t_i)$ correspond to the local clock of t_i. The local clock of each transition is reset to zero once the transition becomes enabled. The predicate $\uparrow Enabled(t_k, M, t_i)$ in the following equation is satisfied if t_k is enabled by the firing of transition t_i from marking M, and false otherwise.

$$\uparrow Enabled(t_k, M, t_i) = (M - {}^\bullet(t_i) + (t_i)^\bullet \geq {}^\bullet(t_k)) \wedge ((M - {}^\bullet(t_i) < {}^\bullet(t_k)) \vee (t_k = t_i)) \quad (1)$$

Time Petri Net is widely used to formally capture the temporal behavior of concurrent real-time systems due to its easy-to-understand graphical notation and the available analysis tools, such as TINA, INA, Roméo, etc.

3.2 Real-Time Property Verification

The safety and reliability of real-time systems strongly depend on the satisfaction of its real-time requirements, in both qualitative and quantitative aspects.

Dwyer et al. initially proposed qualitative temporal property patterns for finite-state verification in [19]. Konrad created in [20] mappings of quantitative requirements into timed logics MTL, TCTL, and RTGIL, and defined a pattern template to ease the reuse. From the viewpoint of property verification, the real-time requirements expressed by Dwyer's and Konrad's patterns are not atomic. We thus defined a minimal set of atomic patterns, which allows to specify the same time requirements as Dwyer's and Konrad's patterns do, to ease the property verification based on observers. We have defined 12 event-based and 4 state-based observers and verified real-time requirements using the reachability assertions. Some early results about the observer-based verification approach are presented in [21, 22].

4 Approach Overview

Let's first see an example benefiting from property specific reduction method.

Example 1 (Example of Property Specific Reduction). When generating the reachability graph preserving markings for the TPN model in Fig. 1 by TINA, it contains **177 states** and **365 transitions**. This system is identified as two sub-nets A and B: A is the structure in dotted box, and B is the other parts. The transition t_4 is the only portal transition between A and B. From the viewpoint of A through t_4, A does not know the inner structure and inner behavior of B, only two informations are observable: how many times t_4 will be fired and the time range for each firing occurrence of t_4.

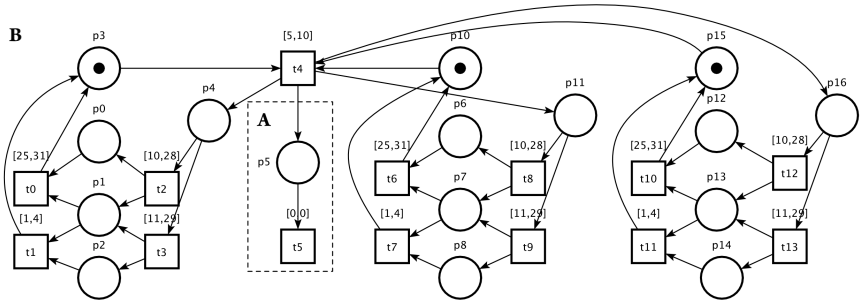

Figure 1. Example of Property Specific Reduction

We provide these informations based on the real-time property verification method presented in the previous section. t_4 is fired infinitely. The time ranges for each firing occurrence are shown in Table 1. For each firing occurrence n ($n \in \mathbb{N}$) of t_4, the time range $[t_n^{min}, t_n^{max}]$ is $[5 + 17(n-1), 10 + 69(n-1)]$. The behavior regularity in this case is that, except the first occurrence, the time difference between current occurrence and the previous one is always in [17, 69].

A sub-net B' conforming to this regular pattern is constructed to replace original sub-net B, as shown in Fig. 2. Sub-net A is kept as before. The reachability graph of the reduced TPN only contains **3 states** and **3 transitions**, but exhibits the same real-time behavior as before from the viewpoint of A.

To summarize the main objective of this work from the above example, we aim to find the regularity of the real-time behavior for the TPN sub-nets from the viewpoint of observations. As we only observe TPN transitions, the real-time behavior from the viewpoint of observed transitions concerns both the firing occurrence times and the time range of each firing occurrence. A reducible sub-net must be independent of its surrounding behavioral context. It means that

Occurrence	Time $[t_i^{min}, t_i^{max}]$	Time Diff $[t_i^{min} - t_{i-1}^{min}, t_i^{max} - t_{i-1}^{max}]$
0	[0, 0]	-
1	[5, 10]	[5, 10]
2	[22, 79]	[17, 69]
3	[39, 148]	[17, 69]
...	...	...
n	[5+17(n-1), 10+ 69(n-1)]	[17, 69]

Table 1. Real-Time Behavior

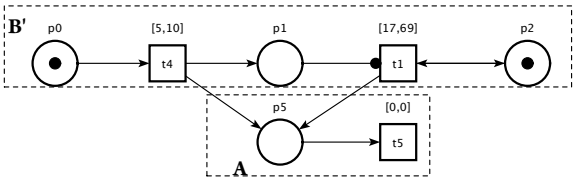

Figure 2. Example Result of Behavioral Equivalence

whether it is "knocked out" from the system or not, it will exhibit exactly the same behavior whenever it is measured, in terms of occurrence times of the portal transition and its time range of each firing occurrence.

An overview of the approach is illustrated in Fig. 3. First, some reducible sub-nets like A, B, and C are identified from the whole TPN model using the *Identification* functions. These sub-nets contain either none incoming transition and one unique outgoing transition such as A, denoted as *one-way-out pattern*; or one incoming and one outgoing transitions such as B and C, denoted as *generic pattern*. The regularity of real-time behaviors for each reducible sub-nets A, B and C are searched using *Reduction* functions relying on observer-based property verification method. If the regularity is founded, reduced sub-nets (A', B', and C') are constructed to replace the original ones after their soundness is assessed by the *Refinement* functions, which also rely on the observer-based property verification method. As the *one-way-out pattern* and the *generic pattern* rely on different identification functions but similar reduction and refinement functions, for the page limit, we only develop our discussion based on the *one-way-out pattern*.

5 Regularity of Real-Time Behavior

The regularity of real-time behavior depends on the characteristics of a system. Fig. 4 illustrates two possible regularities of real-time behavior from the viewpoint of observed transitions. The TPN in Fig.4 (a) has 3 sub-nets: A, B and C. A (resp. B) has a unique portal transition T_A (resp. T_B) to C, and produces tokens via T_A (resp. T_B) periodically or sporadically. From the viewpoint of C,

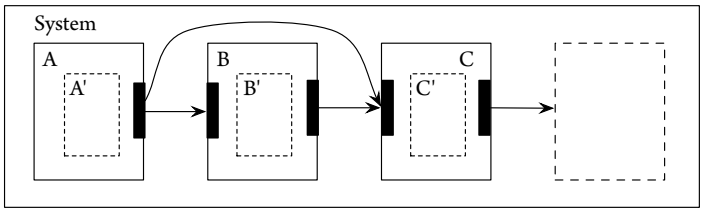

Figure 3. Overview of Behavior Equivalence Approach

regardless the complex inner behaviors of the A and B, they can be seen as single transitions that may fire regularly under a pattern to feed C by tokens. There exists thus an opportunity to abstract and redefine this *regularity* to a reduced TPN A' (resp. B') that may contains less states and transitions than the original one.

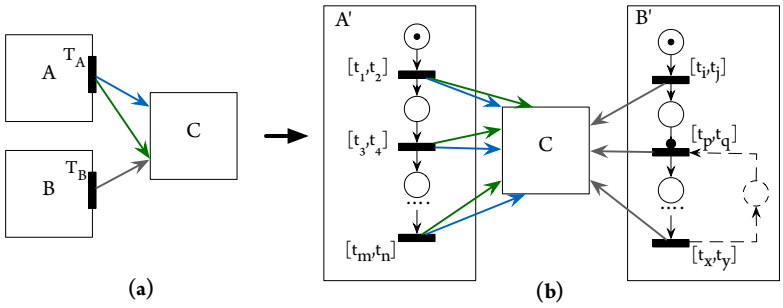

Figure 4. Reduction pattern

When the observation is performed on a TPN transition, the regularity of its firing occurrence is either finite or infinite. The time range of each firing occurrence can be measured using observers if the time ranges are bounded.

Fig. 4 (b) shows two kinds of possible regularity. Assume that we observe the firing time of transitions T_A and T_B for each firing occurrence. The occurrence of T_A/T_B can be either finite (A) or infinite (B). An infinity observer can be added on a transition to check its infinity. Each occurrence T_i has a bounded time range $[t_i^{min}, t_i^{max}]$. These ranges are derived by adding BCET (Best Case Execution Time) and WCET (Worst Case Execution Time) observers on T_A and T_B.

Finite Firing Occurrence If the occurrence is finite, the sub-net A can be represented by a finite sequential section of transitions $\mathbb{T}_{seq} = \{T_i\}$ $(i \in \mathbb{N})$ with adapted time range $[T_i.min, T_i.max]$, where $T_i.min = t_i^{min} - t_{i-1}^{min}$, and

$T_i.max = t_i^{max} - t_{i-1}^{max}$ and $t_0^{min} = t_0^{max} = 0$. It is possible that the regularity of A contains several control modes that lead to several branches with finite sequential transitions.

Infinite Firing Occurrence If the occurrence is infinite, as we focus on finite-state systems, the states in sub-net B must be finite. In other words, there must exist a repeating pattern in B. Depending on system's behavior, there are several possible repeating patterns, such as single loop pattern, nested loop pattern, etc. In this paper, we only discuss one of them: the pattern that is composed of an eventual finite sequential section of transitions $\mathbb{T}_{seq} = \{T_i\}$ $(i \in \mathbb{N})$ and a loop section of transitions $\mathbb{T}_{loop} = \{T_j\}$ $(j \in \mathbb{N})$. The other patterns are under study. Therefore, for now, if the system does not behave the infinite regularity with an eventual sequential section and a loop section, it is considered as non-reducible.

6 Real-Time Property Specific Reduction

The property specific state space reduction method follows three steps (functions): *identification*, *reduction* and *refinement*, which rely on the real-time property specification and observer-based verification approaches presented in [21, 22]. This section details the algorithms for the above functions for the *one-way-out pattern*.

6.1 Identification Function for One-Way-Out Pattern

We first define a symbolic system to ease the discussion:

- t^+ and t^-: for a given transition t, represent respectively the outgoing and incoming arcs of t.
- p^+ and p^-: for a given place p, represent respectively the outgoing and incoming arcs of p.
- $\mathbb{T}^{R(N)}$ and $\mathbb{P}^{R(N)}$: for a given `TPN` N, represent respectively the sets of reducible transitions and places.

We distinguish the reducible and non-reducible `TPN` structure. Non-reducible elements include those structures directly associated with properties, including observer structures, structures directly linked to observers and places/transitions referred to by reachability assertions. The other parts are considered as reducible.

Before performing property specific reduction, some property-irrelevant structures can be directly removed from the reducible net. They are the structures that have causality to the observers. The exact causality can be measured using the reachability graph of the whole system. The paradox exists here: if the whole reachability graph can be generated, we may not need any reduction method. Therefore, to ensure the safety of the removal, we rely on the dependency analysis in `TPN` as a over-approximation. The detailed dependency algorithm is trivial thus will not be presented here. Now assume the set of $\mathbb{T}^{R(N)}$ and $\mathbb{P}^{R(N)}$ are available after the removal.

Identification function $\mathbf{F}(N) = <A, T_{out}>$ identifies, for a given `TPN` N, the enclosed sub-net A that could be possibly reduced (necessary condition), and the unique portal outgoing transition T_{out}:

- A is a connected graph, $A \subset N$, $T_{out} \in A$
- $\forall p \in A, (p \in \mathbb{P}^{R(N)}) \wedge (p^+ \subset A) \wedge (p^- \subset A)$
- $\forall t \in A, (t \in \mathbb{T}^{R(N)}) \wedge (t^- \subset A)$
- $(T_{out} \in A) \wedge (T_{out}^+ \cap \overline{A} \neq \emptyset)$

6.2 Reduction Function

Reduction function $\mathbf{G}(A, t) = <N_S, N_L>$ extracts, for a given sub-net A and the outgoing portal transition t, the behavioral equivalent sequential section N_S for the finite cases, or an eventual sequential section N_S and the loop section N_L for the infinite cases. It first checks the infinity of t in sub-net A using an infinity observer. In both cases, the bounding time range $[t_i^{min}, t_i^{max}]$ is measured using predefined `BCET` and `WCET` observers for the i^{th} firing occurrence of t.

Building Sequential Section In the finite case, there is only a sequential section N_S. The set of sequential transitions $\mathbb{T}_{seq} = \{T_i\}$ $(i \in \mathbb{N})$ in N_S is built using $[t_i^{min}, t_i^{max}]$. Each transition T_i in $\mathbb{T}_{seq}$ is associated with a time range $[T_i.min, T_i.max]$. The algorithm for building N_S from A using the transition t is described in Algo. 1. Initially, t_o^{min} and t_0^{max} are set as 0. N_S starts from an initial place with one token. Whether t^i has occurred is checked using `tHasOcc(i)` function relying on an occurrence observer. For each occurrence (i) of fired t, a pair of `BCET` and `WCET` observers are added to t in the sub-net A to compute the t_i^{min} and t_i^{max}. Then the time range $[T_i.min, T_i.max]$ is associated to the transition T_i. T_i is added in N_S, and an associated new place without token is also added in N_S.

Data: A, t
Result: N_S
$t_o^{min} := 0$, $t_0^{max} := 0$;
N_S`.add(new Place(1))` ;
i := o ;
while *`tHasOcc(i++)`* **do**
  t_i^{min} := `getOccBCET(A,t,i)` ;
  t_i^{max} := `getOccWCET(A,t,i)` ;
  $T_i.min = t_i^{min} - t_{i-1}^{min}$;
  $T_i.max = t_i^{max} - t_{i-1}^{max}$;
  N_S`.add(`T_i`, new Place(0))` ;
end

Algorithm 1: Building Sequential Section

Building Loop Section In the infinite case, the key issue is to identify the firing occurrence of t that divides the sequential section N_S and the loop section N_L. The Algo. 2 is proposed to build the N_S and N_L sections by searching for the loop starting transition ($loopStartIndex$) and the length of loop ($loopLength$).

```
Data: A, t, occThreshold, loopThreshold
Result: N_S, N_L
t_0^min := 0, t_0^max := 0 ;
N_S.add(new Place(1)) ;
occ := 0 ;
while occ++ ≤ occThreshold do
    t_occ^min := getOccBCET(A,t,occ); t_occ^max := getOccWCET(A,t,occ) ;
    for loopStartIndex = 0; loopStartIndex < occ; loopStartIndex ++ do
        for loopLength = 1; loopLength ≤ occ - loopStartIndex; loopLength ++
        do
            match : = 0 ;
            for index = loopStartIndex; index ≤ occ - loopLength; index++ do
                if isSame(<t_index^min, t_index^max>,
                <t_index+loopLength^min, t_index+looplength^max>) then
                    match++ ;
                end
                else break;
            end
            if match ≥ loopThreshold then
                for k = 1; k < loopStartIndex; k++ do
                    T_k.min = t_k^min − t_k−1^min; T_k.max = t_k^max − t_k−1^max ;
                    N_S.add(T_k, new Place(0)) ;
                end
                for k = loopStartIndex; k < loopStartIndex + loopLength; k++
                do
                    T_k.min = t_k^min − t_k−1^min; T_k.max = t_k^max − t_k−1^max ;
                    N_L.add(T_k, new Place(0)) ;
                    N_L.connect(lastPlace, T_loopStartIndex) ;
                end
                return ;
            end
        end
    end
end
```

Algorithm 2: Building Loop Section

As the firing occurrence of t is infinite, an occurrence bound value is predefined as $occThreshold$ to stop the algorithm. As the Identification function $\mathbf{F}(N)$ uses necessary conditions, the identified sub-net A is considered as non-reducible if the loop section cannot be found using $occThreshold$. Another bound value $loopThreshold$ judges whether the $loopStartIndex$ and the $loopLength$ are

found. If the loop pattern holds for *loopThreshold* times, it is considered that this division of N_S and N_L is statistically correct. It is obvious that no matter how big the *loopThreshold* is, the assurance cannot reach 100%, because the loop execution is infinite. In order to make sure that the reduced net refines exactly the same behavior as before, a pre-check (refinement function) must be performed before accepting the reduced structure.

6.3 Refinement Function

The refinement function verifies the behavioral equivalence between the reduced sub-net and the original one. Fig. 5 shows the principle of this function: comparing the time range of each firing occurrence between the nets B and B'. It is realized by adding time interval observers between the transition T_B in B and the transitions T_i in B'. Although the firing occurrence is infinite, under the repeating pattern, the number of T_i is finite. If the refinement fails, it means the system does not fit the behavior regularity, and thus the reduction method cannot be applied.

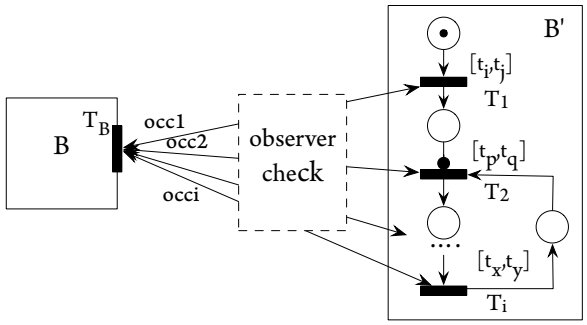

Figure 5. Refinement Function

It is possible that the observed time range do not fully refine the original behavior because of possible "time holes" in this range. For example, a transition can fire during [10,15] or [20,30], but never during]15,20[. If [10,30] is directly used as the time range, the original real-time behavior of the system is extended. Therefore a detailed observation must be introduced to detect the time holes.

For a given observed range [min, max] of transition T, at its i^{th} occurrence, the assertion $check_k$ "*exist* T_i *between k and k+1*" will be checked for all $min \leq k < max$. If $check_k$. If the check does not pass, the time range will be broken into two sections: [min, k] and [k+1,max]. To be more general, if $check_{k_1}$, $check_{k_2}$, ... $check_{k_n}$ do not pass, the final refined equivalent time ranges of this occurrence will become [min, k_1], [$k_1 + 1$, k_2], ..., [$k_n + 1$, max]. Accordingly, the sequential transition of the equivalent sub-net will be refined to a sub-structure which

contains branches representing all possible firing time range after removing those impossible ranges. An example in Fig. 6 (a) shows that the transition T in the reduced sub-net A exhibits a firing time range $[t_3, t_4]$. But there exists time holes on this time range, and thus the real time behavior is $[t_3, t'_3] \cup [t'_4, t_4]$, where $t'_3 < t'_4$. The transition T should be replaced by the sub-range structure (grey part in Fig. 6 (b)).

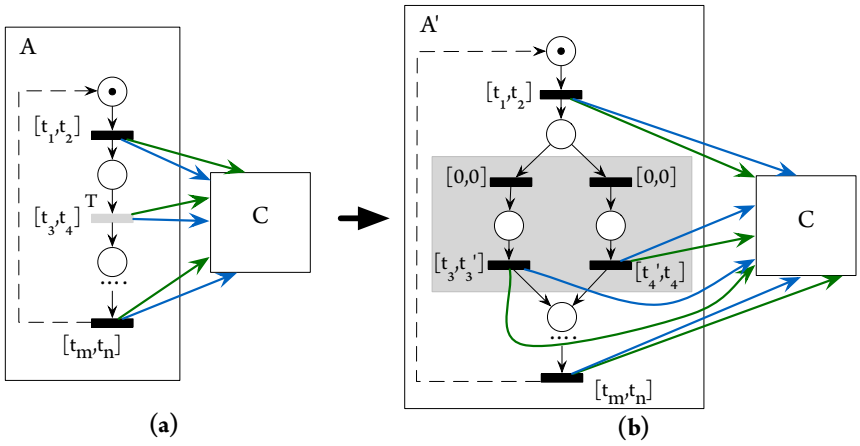

Figure 6. Deal with Holes on Time Range

7 Experimental Results

To experiment the property specific reduction method, we use an avionic case study investigated by M. Lauer et al. [23], which is a part of a flight management system (`FMS`). The `FMS` consists of two units, a control display unit and a computer unit. The control display unit provides human/machine interface for data entry and information display. The computer unit provides both computing platform Integrated Modular Avionics (`IMA`) and various interfaces to other avionics. The communication between modules is implemented by Avionic Full DupleX($AFDX$). `FMS` uses redundant implementation of its functions.

The latency requirement is assessed in the case study. It depends on the functional chain in Fig. 7. At any time, the pilot can request some information on a given waypoint. The KU_1 (*Keyboard and Cursor Control Unit*) controls the physical device used by the pilot to enter his requests. When KU_1 receives a request (req_1), it broadcasts $wpid_1$ and $wpid_2$ to the *Flight Managers* FM_1 and FM_2 respectively. The *FMs* manage the flight plan, i.e., the trajectory between successive waypoints. When a request occurs, both query the *NDB* (*Navigation Database*) by sending $query_1$ (resp. $query_2$) to retrieve the static information

on the waypoint such as the latitude and the longitude. The *NDB* separately answers each *FM* by sending a message $answer_1$ (resp. $answer_2$) containing the expected data. Upon reception of this message, each *FM* computes two complementary dynamic data: the distance to the waypoint, and the *ETA* (*Estimated Time of Arrival*). These data ($wpInfo_1$ and $wpInfo_2$ resp.) are periodically sent to respective *MFDs* (*Multi Functional Display*) which periodically elaborate the pages to be displayed on the screens. The KU_1, *FMs*, *NDB*, *MFDs* are asynchronous functional modules.

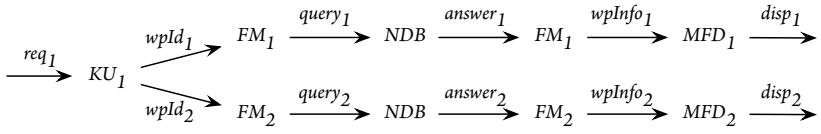

Figure 7. Functional Chain: Sporadic Response to Request

The latency requirement guarantees that the system responds quick enough to a request. It corresponds to the time elapsed between pilot's request (req_1) and the first occurrence of the display signal depending on req_1 ($disp_1$). Therefore, the real-time property here is the worst case time (`WCT`) and best case time (`BCT`) between req_1 and the first occurrence of $disp_1$ depending on req_1.

We model the functional chain in `TPN`. The `WCT` and `BCT` observers are added respectively to the `TPN`. A binary search algorithm is used to search for the bound values. The computation results (verified under MacOS 10.6.8 with a processor 2.4 GHz Intel Core 2 Duo) are shown in Table 2. The `WCT` (resp. `BCT`) is 450.4 (reps. 75.2) `ms`. By applying the reduction approach, the state space is significantly reduced. Take the `WCT` for example, compared to the verification time 278.313 s before reduction, the verification time is reduced to 2.484 s.

Table 2. Real-Time Property Verification Results

Property		**Property Value (ms)**	**State/Transition Number**		**Execution Time (s)**	
			Before Reduc.	After Reduc.	Before Reduc.	After Reduc.
Latency	System	`N/A`	9378/23250	`N/A`	`N/A`	`N/A`
	`WCT`	450.4	67105/145024	9/10	278.313	2.484
	`BCT`	75.2	11162/28922	8/9	43.781	3.719

To test the scalability, the functional chain is enlarged by increasing the number of *NDB*. Each functional chain traverses P *NDB*, i.e. $2P+3$ functions.

$$L_1 = \xrightarrow{req_1} KU_1 \xrightarrow{wpId_1} FM_1 \xrightarrow{query_1} NDB_1 \xrightarrow{query_2} \ldots \xrightarrow{query_{P-1}} NDB_{P-1} \xrightarrow{query_P} NDB_P \xrightarrow{answer_P} NDB_{P-1} \xrightarrow{answer_{P-1}} \ldots \xrightarrow{answer_2} NDB_1 \xrightarrow{answer_1} FM_1 \xrightarrow{wpInfo_1} MFD_1 \xrightarrow{disp_1} \quad (2)$$

Before apply this reduction method, the state space begins to explode even the *NDB* number is 2 under the test environment. By increasing P from 1 to 11, we give out the state/transition number, reduction time, model checking (MC) time and solving time after applying the reduction method in Table 3. The reduction result is prominent. The solving time is almost linear with respect to the system's scale. This case study shows that after reduction, the explosive systems can be analyzed, if the systems conform to the behavioral regularities.

Table 3. Scalability Test for Latency Property

***NDB*/Fun.**	**State/Tran (after Red.)**		**Reduction Time (s)**	**MC Time (s)**		**Solving Time (s)**	
	WCT	BCT		WCT	BCT	WCT	BCT
1/7	9/10	8/9	38.049	2.484	1.860	40.533	39,909
2/8	9/10	8/9	57.876	2.656	1.883	60.532	59,759
3/9	9/10	6/5	79.813	2.812	2.079	82.625	81,892
4/10	9/10	6/5	102.500	2.906	2.079	105.406	104,579
5/11	9/10	6/5	124.987	3.015	2.102	128.002	127,089
6/12	9/10	6/5	149.359	2.891	2.196	152.250	151,555
7/13	9/10	6/5	169.607	2.953	2.227	172.560	171,834
8/14	9/10	6/5	193.329	3.031	2.250	196.360	195,579
9/15	9/10	6/5	216.239	3.000	2.211	219.239	218,45
10/16	9/10	6/5	239.953	3.047	2.195	243.000	242,148
11/17	9/10	6/5	263.049	3.188	2.195	266.237	265,244

8 Computation Complexity & Applicability

This method turns the combination problem of $O(N \cdot M)$ into a divide-and-conquer problem of $O(t_{iden} + n \cdot N + M \cdot N')$, where

- N is the state unfolding complexity of the target sub-net,
- M is the complexity of the other parts of the TPN,
- N' is the state unfolding complexity of the reduced sub-net, $1 \leq N' \leq N$. It is expected that $1 \leq N' \ll N$ if the system conforms to the behavioral regularity.
- t_{iden} is the time for identification, it is $O(N_S^2)$, where N_S is the number of places and transitions in the TPN system.
- n is unfolding times of A by the reduction, refinement and cavity detection
  - Finite case reduction: $2N_B^4 \cdot A_{obs}$, N_B is the defined bound value of occurrence times, A_{obs} is the unfolding time of A with observer.
  - Infinite case reduction: $2N_B^4 \cdot A_{obs}$, N_B is the defined bound value of occurrence times.
  - Refinement: $(n_S + n_L) \cdot A_{obs}$, n_S is the length of sequential section, n_L is the length of loop section.
  - Cavity Detection: $\sum_{i=1}^{n_S+n_L} (max_i - min_i) \cdot A_{obs}$

This method relies on the observers, it may thus take time to search for the bound values of time ranges. In some cases, if the system does not conform to the behavioral regularity, it can only be known after performing the reduction and refinement methods. As our purpose is to reduce the state space of model checking, the trade-off between computation time and the state space is acceptable, except that the computation time is out of the predefined thread-hold value. This is then an engineering problem.

9 Conclusion

This paper proposes a real-time property specific reduction approach for `TPN` based model checking. We illustrate the reduction method for the *one-way-out pattern*. More generic pattern with one incoming portal transition and one outgoing transition uses different identification function, but similar reduction and refinement functions. This method makes the verification more scalable for systems conforming to some behavioral regularities. It makes a trade-off between the state space and the solving time, and allows to verify large scale systems that will easily encounter combinatorial explosion problem, especially for the asynchronous real-time systems. The case study shows that after reduction, the explosive systems can be analyzed, if the systems conform to the behavioral regularities. The reduction and refinement functions rely on the real-time property specification and observer-based verification approaches. For now, we have defined two behavioral regularities for the finite and infinite firing occurrence, and provided reduction methods for the pattern with an eventual sequential section and a loop section. Other real-time behavioral regularities are under study. Similar approaches can be studied to reduce the state space for verifying other families of properties.

Acknowledgment

This work was funded by the FUI P and OpenETCS projects. We also wish to thank Michaël Lauer and Frédéric Boniol for the sharing of the avionic case study.

References

1. Valmari, A.: A stubborn attack on state explosion. In: Computer-Aided Verification, Springer (1991) 156–165
2. Godefroid, P., van Leeuwen, J., Hartmanis, J., Goos, G., Wolper, P.: Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem. Volume 1032. Springer Heidelberg (1996)
3. Misra, J., Chandy, K.M.: Proofs of networks of processes. Software Engineering, IEEE Transactions on (4) (1981) 417–426
4. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3) (1994) 843–871

5. Clarke, E.M., Enders, R., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. Formal Methods in System Design **9**(1-2) (1996) 77–104
6. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Formal methods in system design **9**(1-2) (1996) 105–131
7. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(5) (1994) 1512–1542
8. Holzmann, G.: On-the-fly model checking. ACM Computing Surveys (CSUR) **28**(4es) (1996) 120
9. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool tina - construction of abstract state spaces for Petri nets and time Petri nets. International Journal of Production Research **42**(14) (2004) 2741–2756
10. Sloan, R.H., Buy, U.: Reduction rules for time Petri nets. Acta Informatica **33**(7) (1996) 687–706
11. Berthelot, G.: Transformations et analyse de réseaux de Petri: application au protocoles. Rapports de recherche / Université de Paris-Sud, Laboratoire de recherche en informatique. LRI (1983)
12. Berthelot, G., et al.: Checking properties of nets using transformations. In: Advances in Petri Nets 1985. Springer (1986) 19–40
13. Haddad, S.: A reduction theory for coloured nets. In Rozenberg, G., ed.: Advances in Petri Nets 1989. Volume 424 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1990) 209–235
14. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT press (1999)
15. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time Petri nets. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2003) 442–457
16. Merlin, P., Farber, D.: Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on **24**(9) (1976) 1036 – 1043
17. Cassez, F., Roux, O.H.: Structural translation from time Petri nets to timed automata. JSS **79**(10) (October 2006) 1456–1468
18. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. Softw. Eng. **17**(3) (March 1991) 259–273
19. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering. ICSE '99, ACM (1999) 411–420
20. Konrad, S., Cheng, B.H.: Real-time specification patterns. In: Proceedings of the 27th international conference on Software engineering, ACM (2005) 372–381
21. Ge, N., Pantel, M.: Time properties verification framework for UML-MARTE safety critical real-time systems. In: Modelling Foundations and Applications. Springer (2012) 352–367
22. Ge, N., Pantel, M., Crégut, X.: Formal specification and verification of task time constraints for real-time systems. In: Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies. Springer (2012) 143–157
23. Lauer, M.: Une méthode globale pour la vérification d'exigences temps réel - Application à l'Avionique Modulaire Intégrée. PhD thesis, INPT (juin 2012)