LPT - A Tool for Parametric TPN Validation

Karen Godary-Dejean, Romain Richard, Gregory Angles, David Andreu

LPT - A Tool for Parametric Validation of TPN

Romain Richard
LIRMM - UMR 5506
161 rue Ada - CC 477
34095 Montpellier Cedex 5
France
romain.richard@lirmm.fr

Karen Godary-Dejean
LIRMM - UMR 5506
Université Montpellier 2
161 rue Ada - CC 477
34095 Montpellier Cedex 5
France
http://www.lirmm.fr/∼godary/
godary@lirmm.fr

Gregory Angles
DEMAR Team, INRIA
Université Montpellier 2
161 rue Ada - CC 477
34095 Montpellier Cedex 5
France
gregory.angles@lirmm.fr

David Andreu
DEMAR Team, INRIA
Université Montpellier 2
161 rue Ada - CC 477
34095 Montpellier Cedex 5
France
http://www.lirmm.fr/∼andreu/
andreu@lirmm.fr

This article deals with the problem of temporal and parametric formal validation for discrete event systems. It particularly focuses on the time Petri nets formalism, for which the parametric property verification with model checking is still a non resolved problem. A method is proposed, combining no parametric timed model checking with classical iterative algorithms to avoid combinatory explosion of the analysis process. Algorithms are proposed for the verification of several property types into two specific hypothesis, depending on the parameters values. They have been implemented into a tool: LPT (Little Parametric Tool), which has been used to validate a buffer management system to illustrate the given method.

Parametric formal validation, Model checking verification, Time Petri Nets

1. INTRODUCTION

This paper deals with the formal validation of quantitative and parameterized temporal constraints for embedded real time systems. These systems are mostly critical ones, therefore specific validation methods must be considered to ensure their correct behaviour. Formal methods are used in a validation step during the design process, to validate the system behaviour before the implementation step. The obvious advantage is the early error detection. Furthermore, the use of formal methods provides more validation possibilities than classical methods (as simulation) because of the exhaustive nature of the analysis. Particularly, the model checking allows properties verification, checking the entire states space of a formal model of the system Sifakis (1982). This exhaustive analysis is then well adapted with critical systems.

Furthermore, the real time dimension involves to consider quantitative temporal constraints, in the modelling as well as in the verification phases. For example, a typical property to verify in such systems is the respect of a maximal execution duration. Quantitative time is considered in one way using temporal extensions of modelling formalisms and timed model checking techniques, as the Time Petri Nets (TPN) Merlin (1974) which extend the PN with the consideration of time intervals for the firing of transitions; and in another way using model-checking of timed temporal logic.

Finally, the embedded dimension could lead to require parameters management. Even if the model checking phase is done before the implementation phase, the model must reflect the environment and the hardware architecture of the system, which can be represented as parameters of a global model. In the design process, parameters could for example represent a memory size, a number of components or a clock frequency. However, Alur et al. (1993) showed that the introduction of parameters within the model leads to an undecidable analysis with model checking. In a non parametric validation process, the values of these parameters must be fixed before

the verification process, and the validation results are then guaranteed only for these values. If the environment, the relation with others systems or the hardware target are modified, then the validation process must be carried out again.

This paper proposes to avoid the complexity problem combining non parametric timed model checking with classical iterative algorithms. This allows to insert a parametric dimension into the analysis process without increasing the complexity. A first version of this work has been presented in Godary (2008)[1], which considers only a very simple case of parametric property: the maximum execution time between two transitions execution. The current article identifies and formalises other hypothesis and property types which are not so trivial. For each identified cases, analysis algorithms are given and illustrated on a specific example. The main contributions of this work are in one way the identification of the singleton vs interval cases, which highlights the importance of the initial hypotheses to obtain confident validation results. In another way, the implementation of these algorithms into an analysis tool (Little Parametric Tool - LPT) allows efficient parametric analysis of a TPN system with one parameter, whereas existing parametric model-checkers do not provide answer because of the analysis complexity.

2. RELATED WORKS

We are interested in quantitative and parameterized timed model checking. So, the first subject to tackle is the model checking of quantitative properties. The initial solution to verify such properties in models expressed as timed automata (TA) has been proposed in Alur and Dill (1994). The same article proves the decidability of model checking of TCTL (quantitative temporal logic) on the timed automata. This method and some extensions are implemented in several analysis tools such as UPPAAL Larsen et al. (1997), which provides efficient validation possibilities for quantitative validation of TA models. For quantitative analysis of Time Petri Nets (TPN) models, it was necessary to extend the traditional state class graph method Berthomieu and Diaz (1991) because it does not store information about absolute dates of clocks, which prevents the analysis of quantitative properties. Gardey et al. (2003) proposes a different construction technique for the state space: the zone graph, an adaptation of the region graph used for TA Alur and Dill (1994). Different analysis tools, such as Romeo Gardey et al. (2005) or TINA Berthomieu and Vernadat (2006), implement some of these graphs and allow quantitative model checking for TPN.

[1] in french

On the contrary, the introduction of parameters in model checking is still an open field. Parametric Timed Automata (PAT) have been introduced in Alur et al. (1993). Even if this article proves that parametric model checking in the general case is undecidable, the authors propose, to achieve decidability, to consider a subset of parametrized timed automata Alur et al. (1993). This solution is also used by Hune et al. (2002) where the authors identify a subclass of parametric timed automata for which the analysis is decidable, and propose an extension of the UPPAAL tool to analyse it. To achieve more efficient analysis performances, Wang (2000) defines a subset of PTA, the Statically Parametric Automata SPA. Authors also consider a subset of the TCTL logic, and properties thus could be represented into SPA. Raskin and Bruyère have proposed a solution in Bruyère et al. (2003) for model-checking properties of logic TCTL set to timed automata. These same authors have tried to introduce parameters in the model but showed that the analysis quickly becomes undecidable except under certain circumstances (Bruyère and Raskin (2007)). Howerver, to our knowledge, both the works of Wang and of Raskin & al have not been implanted into analysis tool. Finally, a recent inverse method is presented in André et al. (2009) and implemented in the tool IMITATOR II André (2010): given a PTA and a reference valuation of the parameters, it provides a constraint guaranteeing the same trace set. Even if the parametric model checking problem remains complex for TA models, analysis methods and tools begin to appear.

Still resides the problem of introducing parameters in the analysis process of TPN. As solution exists for parametric analysis of the TAs, a first approach was to change TPN in TA Cassez and Roux (2006) to use the tools of the latter formalism. But the automatic transformation is often based on a multiplication of clocks (an automaton by transition from initial TPN) which leads to a complex analysis of the resulting TA. Delfieu et al. (2007) proposed an approach which transforms the analysis parametric problem in the verification of a set of inequalities representing design constraints. This solution allows an analysis set but deferring the problem on each of the modules of a system without considering the overall system. A recent approach Traonouez et al. (2009) implemented in the Romeo tool offers a solution to parametric model checking, allowing the introduction of parameters in the temporal information of the transitions. The parameters are represented as supplementary constraints linked to the clock constraints. This solution is decidable in case of bounded parameters. However, their solution is still confronted to efficiency problem in case of no-trivial systems validation.

Thus, the analysis of temporal quantitative and parametrized properties remains a problem, especially for the TPN. The introduction of parameters leads to a complexity which does not fit with the limits of technical and analytical existing tools.

The following section 3 defines the useful basic formal definitions, and presents the different identified property types. Then section 4 deals with the parametric validation for the singleton hypothesis, defining this hypothesis, describing analysis algorithms for each of the preceding identified properties and illustrating the proposed method on a classic observer-based verification model. Next, section 5 introduces the parametric validation in case of the interval hypothesis, illustrating it through a buffer management example. Finally, section 6 presents the LPT analysis tool which implements proposed algorithms, providing performances measured for the case study validation. Section 7 concludes the paper.

3. DEFINITIONS

We first discuss the terminology and formal notations which are the basics of the definitions introduced in the paper. Then this section presents the considered property types, considering only one fixed parameter into a transition firing interval of the TPN model.

3.1. Formal definitions

3.1.1. Basic notations

All over this paper, we will use these notations: the system is called Y, the property φ, a parameter valuation p and the set of the parameter valuations Γ. Only when necessary other notations will be explicitly given. We also remind that the model checking problem allows to know if $Y \models \varphi$, which means that Y satisfies φ. We also use $Y \not\models \varphi$ for the contrary.

3.1.2. Temporal logic

Model checking of timed temporal logic is known to be extremely sensitive to combinatorial explosion, and few model checker allow timed properties verification. The solution is to explicitly represent the temporal constraint in a property model, and then using a more classical temporal logic model checking (as the observer model of Figure 2).

Thus, this paper focuses on properties expressed with the Linear Temporal Logic (LTL). LTL consists of the classical logic operators and the mood `until` $\mathcal{U}$ and `next` $\mathcal{X}$. It is defined by the following grammar:

$$\varphi ::== p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2 \mid \mathcal{X}\varphi_1$$

The properties could be completed with the Future (F) and Globally (G) operators, which are a combination of the initial operators. $F\ \varphi$ means that φ will be `true` one day, and $G\ \varphi$ means that φ is always `true`.

3.1.3. Parametric verification notations

We first remind the parametric model checking problem defined in Traonouez et al. (2009): "for a parametric system Y and a parametric specification φ, it consists in checking whether there exists a valuation p of the parameters such that Y satisfies φ for this valuation, which is written $[\![Y]\!]_p \models [\![\varphi]\!]_p$". Traonouez et al. (2009) also defined the parametric synthesis problem which "computes the set of valuations Γ such that $\forall p \in \Gamma, [\![Y]\!]_p \models [\![\varphi]\!]_p$". This article deals with a very close problematic, without considering parameterized properties. Based on these definitions, we introduce the **parametric minimal bounded analysis problem** which computes the minimal valuation p_{min} of the parameters such that:

$$\exists p_{min} \in \Gamma \mid \forall p \in \Gamma, p \geq p_{min} \iff [\![Y]\!]_p \models \varphi$$

The **parametric maximal bounded analysis problem** computes in the same way the maximal valuation.

3.2. Types of properties

3.2.1. Terminology

To find parameters' values for which a property is satisfied, the search range must be bounded. We defined that, as shown Figure 1, the parameter value will be sought into an interval $[Min, Max]$. Then we have named Low the lowest value, and $High$ the highest value, for which the property is verified in this search range. We obviously have $Min \leq Low$ and $High \leq Max$.

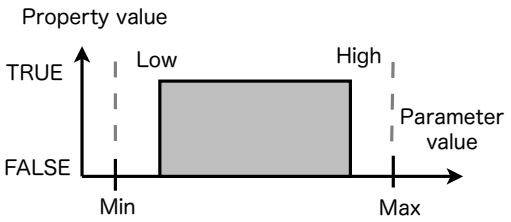

Figure 1: *General definition and type 2 property*

3.2.2. Type 1

The first type of property is the simplest one: the property value changes only once on the search range. We then only have one of the bounds Low or $High$. We formally defined this as:

$$[\![Y]\!]_p \models \varphi \iff p \geq Low$$

The symmetric situation is also a type 1 property:

$$[\![Y]\!]_p \models \varphi \iff p \leq High$$

We can illustrate for example the Low type 1 case on Figure 1, by considering only the left part of the figure[2]: the property is FALSE at the beginning of the search range and until the Low value, then becomes definitively TRUE after.

3.2.3. Type 2

The second type of property is the one illustrated Figure 1: the property is always FALSE excepted when the parameter value is in a specific interval:

$$[\![Y]\!]_p \models \varphi \iff p \in [Low, High]$$

3.2.4. Type 3

Finally, the third type of property corresponds to the generalisation of type 2: the property value forms several intervals depending on the parameter value (see an example in Figure 4). Supposing that there are n intervals $[Low_i, High_i], i \in [0, n]$ into the search range ($[Low_i, High_i] \subset [Min, Max]$) so we have:

$$[\![Y]\!]_p \models \varphi \iff p \in [Low_i, High_i],\ \forall i \in [0, n]$$

3.3. Singleton vs interval hypotheses

The property types definitions are given for a specific case: when the parameter value is constant, ie the parameter keeps the same value for all the duration of the system execution. This is called the **singleton hypothesis**. But we can be interested in studying when the parameter value can vary into an interval: the **interval hypothesis**. Thus we have to define precisely these two hypotheses.

Hypothesis on the parameter value: A system is parameterized with the singleton hypothesis when the parameter value could not vary during the system execution. Such a system is represented with the following notation:

$$[\![Y]\!]^S_a \ : \ [\![Y]\!]_p \mid p = a$$

On the contrary, if the parameter value could vary dynamically into an interval during the system execution, we have :

$$[\![Y]\!]^I_{a,b} \ : \ [\![Y]\!]_p \mid p \in [a, b]$$

We of course have: $[\![Y]\!]^S_a \iff [\![Y]\!]^I_{a,a}$

Hypothesis on the verification process: The singleton and interval hypotheses could also be applied to the verification process. The singleton verification is when a system verified a property for each singleton parameter value, whereas the interval verification considers all the possible parameter values. Depending on the parameter values, we can have simple equivalences:

$$[\![Y]\!]_{[a]} \models^S \varphi \iff [\![Y]\!]_{[a]} \models \varphi$$

$$[\![Y]\!]_{[a,b]} \models^I \varphi \iff [\![Y]\!]_{[a,b]} \models \varphi$$

or more complex ones:

$$[\![Y]\!]_{[a,b]} \models^S \varphi \iff \forall p \in [a, b] \cap \mathbb{N},\ [\![Y]\!]_{[p]} \models \varphi$$

To clarify the rest of the article, we will always note explicitly if the verification is by singleton or by interval.

The next section focuses on parametric validation of the singleton case. The interval validation will be explained in details section 5.

4. SINGLETON PARAMETRIC VALIDATION

This section firstly introduces the basic definition of the parametric bounded analysis problem for the singleton hypothesis. It then presents the algorithms we propose to resolve the parametric verification problem, with the singleton hypothesis, for each property type defined section 3.2.

4.1. Formal singleton definitions

This paper considers parameters associated with a transition firing interval. In the singleton case, the parameter value must be fixed, then the firing instant of the transition can not vary: the firing interval becomes a fixed duration $[p, p]$. The parametric model checking problem is then specific $[\![Y]\!]^S_p \models^S \varphi$ and we can define **the parametric *minimal* bounded analysis problem in the *singleton* case**:

$$\exists S_L \in \Gamma \mid \forall p \in \Gamma,\ p < S_L \iff [\![Y]\!]^S_p \not\models^S \varphi \qquad (1)$$

and **the parametric *maximal* bounded analysis problem in the *singleton* case**:

$$\exists S_H \in \Gamma \mid \forall p \in \Gamma,\ p > S_H \Longrightarrow [\![Y]\!]^S_p \not\models^S \varphi \qquad (2)$$

The rest of this section presents the algorithms proposed to answer to this analysis problem for the previously identified property types in the singleton case.

4.2. Type 1 singleton analysis

The parametric verification method presented in this paper proposes an alternative to the parametric model checking complexity problem, by the multiple execution of the non parametric analysis with different fixed values of the parameter.

[2]It is useful to note that the property type greatly depends on the specified search range. For example if $Max < High$ in Figure 1, this is a type 1 case for the Low bound whereas in the contrary it is a type 2 case.

4.2.1. Type 1 singleton analysis algorithm

The first analysis algorithm concerns the validation of a type 1 property. It is based on an dichotomy execution of a non parametric model checking problem, verifying at each iteration if the system satisfies the property for a fixed parameter value. It then modifies the dichotomy interval depending on the verification result and on the searched bound. The algorithm given below[3] searches the low singleton bound S_L, and one of its executions is detailed in Figure 3.

Algorithm 1 Low singleton bound algorithm

procedure SLOW(Y, φ, Min, Max)
  $dLow \leftarrow Min$
  $dHigh \leftarrow Max$
  while $(dHigh - dLow) > 1$ **do**
    $mid \leftarrow \lfloor \frac{dHigh+dLow}{2} \rfloor$
    if $[\![Y]\!]^S_{mid} \models^S \varphi$ **then**
      $dHigh \leftarrow mid$
    else
      $dLow \leftarrow mid$
    end if
  end while
  return $dLow$
end procedure

Variables handled in all the algorithms are integers. Indeed, the firing time interval bounds in the TPN models must be integers in the toolbox we use for the modelling and the non parametric model checking (see section 6). However, the property verification is done for all possible time values (model checking in dense time).

4.2.2. Illustration example

This section presents a case study illustrating the purpose of our work: the classic example of an observer model (Figure 2). Indeed, observer model is a very frequent technique used to convert complex temporal logics verification in a reachability problem which is adapted to more efficient model checking algorithms.

The observer is used to verify the worst execution time between two transitions: `tstart` and `tend`, which represents the maximum execution time of a suite of actions (we call it $MaxTime$). In this model, the firing interval of the `terror` transition is a duration. If this duration is higher than $MaxTime$, the ERROR place could not be marked. To know the value of $MaxTime$, it is necessary to find the minimal value of the `terror` duration for which the ERROR place is never marked.

[3]Only the essential instructions of the algorithms are given. For example, the initial verification steps for the Min and Max values are not given.

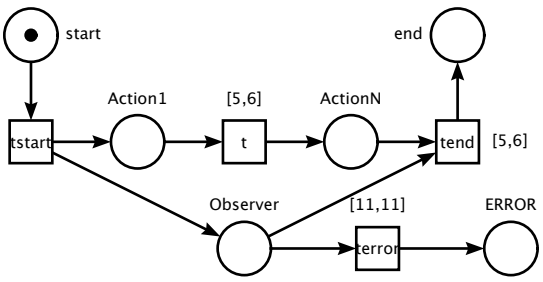

Figure 2: *TPN model of a worst execution time verification*

This is a parametric minimal bounded analysis problem with the parameter p equal to the `terror` duration and the property φ is a reachability one: "$G\ M_{\text{ERROR}} = 0$", with M_{ERROR} the marking of the ERROR place. The parameter is a duration then it can only has one fixed value: this is the singleton hypothesis. When the p value is inferior or equal to $MaxTime$, φ is FALSE; then when p is superior, the property becomes definitely TRUE. Thus, we are in a type 1 property case.

4.2.3. Example analysis

In this illustration example, the worst case execution time of the set of actions, and then the wanted bound is $MaxTime = S_L = 12$. The execution of the algorithm to find this bound is illustrated in Figure 3, supposing that the initial search range is $[0, 15]$. The algorithm begins to test the value 7 (middle of the search range). As φ is not verified for this value the algorithm replaces the search range with $[7, 15]$. In the same way, the values 11 and 13 are tested, and the search range is reduced to $[11, 13]$. The middle of this interval is 12, for which the property is verified: the search range is reduced to $[12, 13]$. Then the minimal interval difference is reached: the low singleton bound is $S_L = 12$.

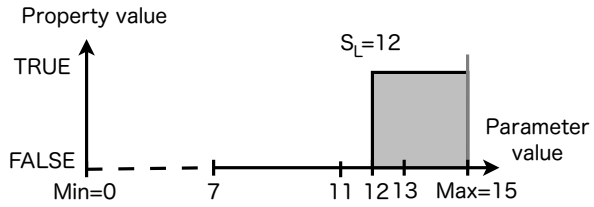

Figure 3: *TPN model of a worst execution time verification*

4.3. Type 2 singleton analysis: interval

The second algorithm does the parametric verification for type 2 properties: the S_L and S_H bounds are unknown. We can not use directly the first algorithm because it could find a wrong result in specific cases. For example, if we search a set of values of a temporal parameter (for which the property is verified) which is equal to the set $[2, 5]$. We assume

that we still search the S_L bound in the $[0, 15]$ search range. As in the previous execution, the algorithm starts checking the value 7 and find that the property is not verified for this value. The main problem here is that it thinks than all the values between 0 and 7 do not satisfy φ. It then changes the search range to $[7, 15]$ and continues. Finally, the tool concludes there is no value of the parameter in the initial search range for which the property is verified, which is wrong.

So, we have developed a second algorithm, given below. First, the function FIND(Y, φ, Min, Max) finds one parameter value val for which the property is satisfied: $\exists val \in \Gamma \mid [\![Y]\!]^S_{val} \models^S \varphi$. FIND(..) is based on a classical model checking problem, setting the parameterized transition to $[Min, Max]$ and verifying "$F\ M_{\text{ERROR}} = 0$". If the answer is TRUE, the trace is recovered and analysed to extract the corresponding parameter value val. If the property is a type 2 one, this value is in the wanted interval: $val \in [S_L, S_H]$. Then, the first algorithm could be used to find S_L and S_H bounds respectively in the $[Min, val]$ and $[val, Max]$ search ranges.

Algorithm 2 Singleton type 2 interval algorithm

```
1: procedure SINT(Y, φ, Min, Max)
2:     val ← FIND(Y, φ, Min, Max)
3:     S_L ← SLOW(Y, φ, Min, val)
4:     S_H ← SHIGH(Y, φ, val, Max)
5:     return S_L and S_H
6: end procedure
```

4.4. Type 3 singleton analysis: multi-intervals

Finally, the third algorithm does the analysis of the type 3 properties which could be TRUE during several intervals. This algorithm uses the previous idea to find a value val for which the property is verified, and exploits the first algorithms to find the bounds. But in this case the first algorithms could provide intermediate bounds. Figure 4 illustrates it, with the property TRUE in $[8, 10]$ and $[12, 16]$. If FIND$(Y, \varphi, 0, 18)$ returns $val = 15$, then SLOW$(Y, \varphi, 0, 15)$ finds $S_L = 12$, like in section 4.2. But this bound is not the real one: the interval $[8, 10]$ has been ignored because of the dichotomy.

The idea of the third algorithm is to verify, after the achievement of the first S_L (or S_H) bound, if there exist other parameter values for which the property is verified, excluding the current interval. As for the second algorithm, it is decomposed into three sub-parts: the achievement of the val example value, then the separate searches of the S_L and the S_H values. The difference is that the searches here are recursive. This algorithm

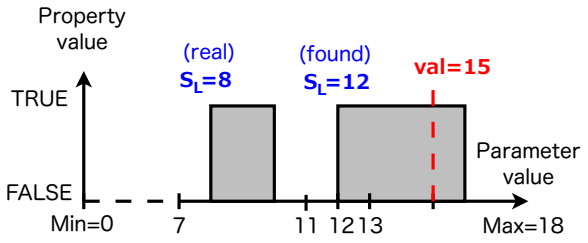

Figure 4: *TPN model of a worst execution time verification*

is then a general case for both type 2 and type 3 properties verification. The FIND(Y, φ, Min, Max) function must be adapted for this case: it searches a val value in the specified interval excluding the extreme values: $val \in]Min, Max[$. If no value is found, it returns -1.

Algorithm 3 Singleton multi-intervals algorithm

```
1: procedure SMULTIINT(Y, φ, Min, Max)
2:     val ← FIND(Y, φ, Min, Max)
3:     S_L ← SLOWR(Y, φ, Min, val)
4:     S_H ← SHIGHR(Y, φ, val, Max)
5:     return S_L and S_H
6: end procedure
7:
8: procedure SLOWR(Y, φ, Min, Max)
9:     val ← FIND(Y, φ, Min, Max)
10:    if val = −1 then
11:        return −1
12:    end if
13:    S_L ← SLOW(Y, φ, Min, val)
14:    return MIN(S_L,SLOWR(Y, φ, Min, S_L)
15: end procedure
16:
17: procedure SHIGHR(Y, φ, Min, Max)
18:    val ← FIND(Y, φ, Min, Max)
19:    if val = −1 then
20:        return −1
21:    end if
22:    S_H ← SHIGH(Y, φ, val, Max)
23:    return MAX(S_H,SHIGHR(Y, φ, S_H, Max)
24: end procedure
```

4.5. Discussion on the S_L and S_H definitions

The multi-interval case could lead to wonder about the needed S_L bound value. Equation 1 defines the minimal low bound value, ie. there is no inferior value for which the property is verified. But it could be considered that the user needs to know the highest low bound into the search range, or maybe the average value of all the existing low bounds of the intermediate intervals. The questions are the same for the S_H bound. This article presents solutions only for the minimal low bound, and for the maximal high bound, as described section 4.1. But the definition of

intermediate bounds will be useful:

$$\exists S'_L \in \Gamma \mid [\![Y]\!]^S_{S'_L} \models^S \varphi \text{ and } [\![Y]\!]^S_{S'_L-1} \not\models^S \varphi \quad (3)$$

$$\exists S'_H \in \Gamma \mid [\![Y]\!]^S_{S'_H} \models^S \varphi \text{ and } [\![Y]\!]^S_{S'_H+1} \not\models^S \varphi \quad (4)$$

We of course have:

$$S'_L \geq S_L \quad \text{and} \quad S'_H \leq S_H \quad (5)$$

The intermediate singleton bounds could be obtained using the non recursive SLOW(..) and SHIGH(..) algorithms.

The following section 5 deals with the parametric bound analysis problem with the more complex interval hypothesis, and illustrates the proposed methods for the validation of a specific case study.

5. INTERVAL PARAMETRIC VALIDATION

5.1. Limitations of the singleton parametric hypothesis

The singleton hypothesis imposes that the parameter always has the same value during the execution of the model. When the parameter is the firing instant of a transition, this hypothesis leads to a $[p, p]$ firing interval. It corresponds to a large number of validation cases, when an observer is used to model properties, which is a very common validation technique. More generally, this corresponds to all the models, as the one of Figure 2, where the parameterized transition is fired only once.

However, some systems need latitude on the parameter value. For example in the model of Figure 5, supposing that the receive function is implanted into a micro-controller, it is difficult to respect a precise time period. The firing interval of transition `receive` must then be set to a more flexible one $[p_1, p_2]$ with $p_1 < p_2$. The interest of the parametric validation is to obtain the bounded values of this interval assuring that any period variation into this interval does not lead to a buffer overflow.

Then, if the singleton hypothesis is not desired nor realistic, the preceding analysis algorithms have to be modified to allow the parametric bounded analysis. The following sections introduce formal definitions for the interval hypothesis, describe a system following this hypothesis, give the parametric validation results of this example, and finally detail the analysis methods and algorithms used to solve this parametric analysis problem.

5.2. Formal interval definition

In the interval case, the parameter value could vary during the execution of the system. The model is then modified to set the firing interval of the parameterized transition as an interval $[p_1, p_2]$. Such a parameterized model is noted $[\![Y]\!]^I_{p_1,p_2}$, and the parametric model checking problem in the interval case is: $[\![Y]\!]^I_{p_1,p_2} \models^I \varphi$. This means that the model verifies the property for all the parameter values included in $[p_1, p_2]$.

The parametric *minimal* bounded analysis problem in the *interval* case then computes the lowest valuation I_L such that:

$$\exists I_L \in \Gamma \mid \forall p \in \Gamma, p \geq I_L \Longrightarrow [\![Y]\!]^I_{I_L,p} \models^I \varphi \quad (6)$$

In the same way we define **the parametric *maximal* bounded analysis problem in the *interval* case**:

$$\exists I_H \in \Gamma \mid \forall p \in \Gamma, p \leq I_H \Longrightarrow [\![Y]\!]^I_{p,I_H} \models^I \varphi \quad (7)$$

Explaining Γ as a $[Min, Max]$ interval, equation 6 implies that φ **is TRUE for all the combinations of the parameter values in** $[I_L, Max]$.

$$\forall p_1, p_2 \in [I_L, Max] \Longrightarrow [\![Y]\!]^I_{p_1,p_2} \models^I \varphi \quad (8)$$

Respectively, equation 7 implies:

$$\forall p_1, p_2 \in [Min, I_H] \Longrightarrow [\![Y]\!]^I_{p_1,p_2} \models^I \varphi \quad (9)$$

For all these four equations, the inverse implications are false. This will be illustrated section 5.4, with the verification of a case study presented in the next section.

5.3. Case study

This section presents a case study which will be used to illustrate the parametric validation method in the interval hypothesis. This is a buffer management system: a buffer is periodically used by two cyclic components which read or write objects. Only one object is generated at a time, whereas the read action concerns all the stored objects.

5.3.1. System model

Figure 5 presents a TPN model of the system.

- the object generation is modelled by the transition `send` (top left of the figure) which is periodically executed with a frequency equal to the transition duration. The generated object is represented by a token in the place `pushOne`.
- Similarly, the object reception is represented at the top right by the transition `receive`.
- the buffer itself is modelled by two places: the number of tokens in `bufferSize` represents the free places remaining in the buffer, and the number of tokens in `buffer` represents the stored objects.

- the object storage is composed of two options: either a place is available in the buffer, and then the `push` transition is executed, or the buffer is full and then the object is lost with an `overflow` notification.
- all the objects are read at the same time when there is a token in `startPopAll`. The reading is done in several step: first, transition `more` is fired once per stored object before the firing of transition `empty`; then `free` is fired once, `free1` is fired once per stored object, and finally transition `end` concludes the reading process.

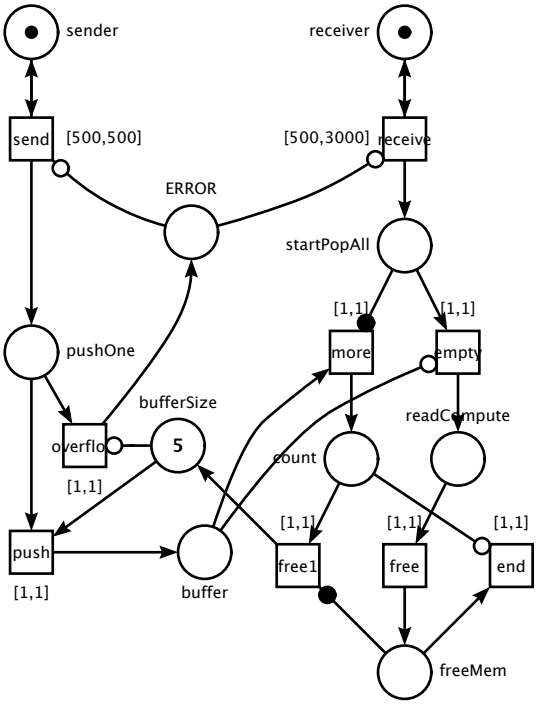

Figure 5: *TPN model of a buffering system*

5.3.2. Parametric verification

This model could have several parameters: the buffer size is represented as the marking of place `bufferSize`; the packet generation frequency could be represented by a parameter in the `send` transition firing interval; finally, the packet reading frequency is represented in the `receive` transition firing interval. This article focuses in this last parameter. The property we want to verify is the absence of object loss. This property is integrated in the system model by the `ERROR` place, which is marked if an overflow appends. Thus, the property verification becomes a reachability property, independent from the parameter value.

The parametric problem in our example is to know the maximal frequency value the receiver has to respect to ensure that there is no object loss. The property to verify is the same one as in section 4.2 ($G\ M_{\text{ERROR}} = 0$), but this time this is a parametric **maximal** bounded analysis problem.

5.4. Case study validation

In an unusual way, this section presents the validation results before the presentation of the analysis algorithm. Indeed, we think that understanding the analysis results helps to understand the underlying algorithm.

5.4.1. Verification results

Table 1 shows the numerical values of different verification runs done for different parameter values into the search range $[500, 3000]$ of the Figure 5 model (the parameter is in the `receive` transition and the verified property is "$G\ M_{\text{ERROR}} = 0$"). These results are represented Figure 6: singleton results are represented as points, and intervals for which the property is verified by lines.

no	Parameter values	$[\![S]\!]^I \models^I \varphi$?
1	$[p,p]\ \ \forall p \in [500, 2496]$	TRUE
2	$[p,p]\ \ \forall p \in [2497, 2499]$	FALSE
3	$[2500, 2500]$	TRUE
4	$[p,p]\ \ \forall p \in [2501, 3000]$	FALSE
5	$[p',p]\ \ \forall p' \in [2497, 2499]$	FALSE
6	$[p',p]\ \ \forall p' \in [2501, 3000]$	FALSE
7	$[p,p']\ \ \forall p' \in [2497, 2499]$	FALSE
8	$[p,p']\ \ \forall p' \in [2501, 3000]$	FALSE
9	$[p; 2492]\ \ \forall p \in [500, 2492]$	TRUE
10	$[505; 2493]$	TRUE
11	$[p; 2493]\ \ \forall p \in [500, 504]$	FALSE
12	$[1004; 2494]$	TRUE
13	$[p; 2494]\ \ \forall p \in [500, 1003]$	FALSE
14	$[1503; 2495]$	TRUE
15	$[p; 2495]\ \ \forall p \in [500, 1502]$	FALSE
16	$[2002; 2496]$	TRUE
17	$[p; 2496]\ \ \forall p \in [500, 2001]$	FALSE

Table 1: *Verification results of the case study*

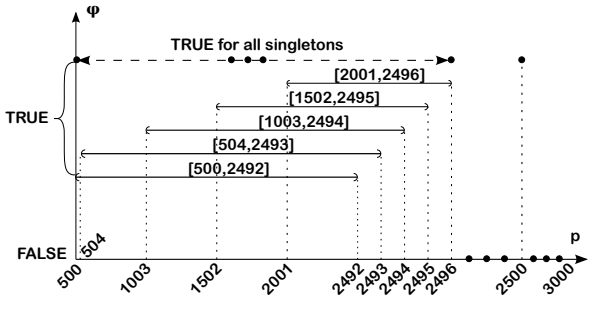

Figure 6: *Verification results of the case study*

5.4.2. Results analysis

We could analyse these verification results:

- The real high singleton bound is $S_H = 2500$, so equation 2 guaranties that all the singleton

values superior to 2500 do not verify φ. This justifies lines 3 and 4 of the table.

- There is only one intermediate high singleton bound $S'_H = 2496$, then according to equation 4, all the singleton values inferior or equal to 2496 verify φ, whereas the singleton values included in $[2497, 2499]$ do not verify it (lines 1 and 2).
- According to equation 7, all the intervals including "FALSE singleton values" do not verify φ. This justifies lines 5 to 8 of the table.
- The high interval bound is $I_H = 2492$ thus equation 7 guaranties that all the interval values in $[500, 2492]$ verify φ (line 9).
- The others results are given to show that there are several intervals of parameter values which could satisfy the parametric maximal bounded analysis problem, depending on the search range. This is discussed in section 5.5.2.
- lines 11, 13, 15 and 17 also prove that the reverse implication of equation 7 is false.

5.5. Interval analysis

This section presents the algorithm used to obtain the high interval bound I_H. The low interval bound I_L algorithm is similar.

5.5.1. Interval analysis algorithm

The interval high bound algorithm is also based on a dichotomy search, using a non parametric model checking analysis on a $[Min, Max]$ interval. The I_H bound will be found varying maximal bound of this interval with the dichotomy. This solution is implemented in the following DIHIGH(..) dichotomous function.

However, this solution could be a very long process, as even the non parametric model checking analysis could has an important complexity in the interval case. Furthermore, the complexity could be dependant on the search range: the more the parameterized transition firing interval is large, the more this transition could be concurrent with other transitions, provoking interleavings. We then propose to use the values of the singleton bounds to reduce the search range. Equation 7 and 5 imply that:

$$I_H \leq S'_H \leq S_H, \quad \forall S'_H \tag{10}$$

Thus, the search range to find the I_H value could be reduced to $[Min, S_H^{found}]$ into the DIHIGH(..) function, with S_H^{found} one of the S'_H intermediate high singleton bounds. In the worst case, the S_H^{found} is equal to the S_H bound, but this still reduce the initial search range. This solution is implemented into the IHIGH(..) following algorithm.

Algorithm 4 High interval bound analysis algorithm

```
1: procedure IHIGH(Y, φ, Min, Max)
2:     S'_H ← SHIGH(Y, φ, Min, Max)
3:     if ⟦Y⟧^I_{Min,S'_H} ⊨^I φ then
4:         return S'_H      ▷ the singleton bound is the
   same as the interval one
5:     else
6:         I_H ← DIHIGH(Y, φ, Min, S'_H)
7:         return I_H
8:     end if
9: end procedure
10:
11: procedure DIHIGH(Y, φ, Min, Max)
12:     I_H^max ← Max
13:     I_H^min ← Min
14:     while (I_H^max − I_H^min) > 1 do
15:         mid ← ⌊(I_H^max + I_H^min)/2⌋
16:         if ⟦Y⟧^I_{Min,mid} ⊨^I φ then
17:             I_H^min ← mid
18:         else
19:             I_H^max ← mid
20:         end if
21:     end while
22:     return I_H^max
23: end procedure
```

5.5.2. Other valid intervals

The previous algorithm gives the I_H high bound which corresponds to the intervals including the Min bound. For example in the case study, the I_H given bound is 2492, because this is the highest value of the interval $[500, 2492]$ for which the property is verified for any values of the parameter in this interval. It could not be 2493 because it exists a trace including 500 and 2493 which leads to the ERROR marking[4].

However, $[500, 2492]$ is not the only interval for which the property is satisfied in the given initial search range $[500, 3000]$. In fact, it exists at most $(S_H - I_H) + 1$ number of such intervals. For the case study, there are $2496 - 2492 + 1 = 5$ intervals, given Table 1 and Figure 6. Such an interval is called a **high interval** and is defined by:

- $[I_{min}, I_{max}] \subset [Min, Max]$
- $I_{min} \in [Min, I_{max}], \quad I_{max} \in [I_H, S_H]$
- $[\![Y]\!]^I_{p_1,p_2} \models^I \varphi \quad \forall p_1, p_2 \in [I_{min}, I_{max}]$

[4] This trace is not given here because it is composed of hundreds of transitions.

All these high intervals could be found using algorithms presented in this paper. Knowing the S_H and the I_H bounds, a search of the I_L bound gives the minimal bound of the associated high interval. This low bound search, done for each one of the integer values belonging to $[I_H, S_H]$, provides all the high intervals.

Therefore a reflection could be done here to know what is the most interesting interval. We could imagine several possibilities. Supposing in one hand that the receive function of the case study system (figure 5) is done by a micro-controller which also has to execute many other functionalities. One important element could be to have the larger temporal latitude to execute the receive task (leading to $[500, 2492]$). Another element could be to include the worst possible period (leading to $[2001, 2496]$).

6. LPT TOOL

6.1. Implementation

The software LPT (Little Parametric Tool) works with Time Petri Nets models which respect the TINA syntax. The TINA toolbox [5] is also used to execute the non parametric model checking steps: $[\![Y]\!]_p^S \models^S \varphi$ and $[\![Y]\!]_{p_1,p_2}^I \models^I \varphi$. `tina` is the tool used to build the states graph, and `selt` is used to verify the property in this graph. We also use `plan` to find a single value of the parameter which verifies a given property. Another specific step of the algorithm is the modification of the system model setting a specific value for the parameter. This function is done by a classical textual management, as TINA provides graphical as well as textual TPN models representation.

The current version of LPT is implemented in java language, as a plug-in of the Eclipse platform. It is currently integrated to a larger project within the Hilecop tool Andreu et al. (2008), a VHDL code generator allowing the automatic translation of Petri nets into VHDL components. An automatic bridge from Hilecop to LPT as been done, to verify properties on the same models as the one which will be implemented. The final goal is to validate the model in a generic way, with some hardware architecture characteristics taken into account by means of parameters. The LPT tool will be soon available for free download[6].

6.2. Performances

In this section, we compare performances between a non parametric analysis with TINA, a parametric validation with our tool LPT and a parametric validation with the parametric version of Romeo.

We have used the same model for all the tests, the one of section 5.3, modifying only the `receive` firing interval. This model contains 10 places and 9 transitions. Its associated state graph could contain thousands of places and transitions. The state graph size depends on the value of the firing interval of transition `receive`, the one we want to parameterize. Table 2 gives representative results for different parameter values. Several factors could influence the graph size. First, the larger the firing interval of `receive` is, the more this transition could be in concurrence with other transitions. Then we could imagine that the worst case is associated with the largest firing interval. But the system has been modelled with a specific construction (Figure 5.3.1): the execution of the model is stopped as soon as there is a buffer overflow. This modelling technique is used to prevent the state graph becoming infinite because of unbounded places. Then, some of the state graphs could be very small because of an immediate error scenario, as for the $[500, 2500]$ firing interval, whereas this scenario is not present in $[500, 2499]$.

Table 2: *Case study graph size*

search range	states	transitions
$[500, 2497]$	12832	17689
$[2497, 2497]$	12057	12087
$[2499, 2499]$	181	194
$[500, 2499]$	12840	17707
$[500, 2500]$	1204	1711
$[500, 3000]$	1097	1616

As we can see in Table 2 , it is not possible to a priori predict the graph size of the model. The complexity of the generated analysis graph basically depends on the number of places and transitions of the model, but also on its structural and behavioural complexity: the well-known interleaving problem. Thus it is difficult to know the complexity of the analysis. Furthermore, the analysis performances depends on the analysis tool implementation efficiency (for example for the storage of the data structures), which is independent of the theoretical complexity of the analysis method.

Nevertheless, Table 3 gives some performances measurements (in terms of analysis execution time) to have a first idea of LPT efficiency, comparing with TINA and Romeo performances. The verification tests have been done on a $2,66GHz$ Intel Core i7 processor with $4GB$ $1067MHz$ RAM memory. The ∞ symbol means that the analysis process does not

[5] http://projects.laas.fr/tina//
[6] http://gforge-lirmm.lirmm.fr/gf/project/lpt/

provide a solution after more than 1 hour, whereas 0 means that the result is almost instantaneous.

Table 3: *LPT Performances*

search range	TINA	LPT	Romeo
$[500, 2497]$	$1s$	$5,1s$	∞
$[2497, 2497]$	$1s$	$2,4s$	$7,3s$
$[2499, 2499]$	0	0	0
$[500, 2499]$	$1s$	$4,9s$	∞
$[500, 2500]$	0	$4,3s$	∞
$[500, 3000]$	0	$3,6s$	∞

It is difficult to exactly compare the LPT performances with non parametric performances: the comparison with TINA is not really meaningful since a non parametric analysis could not give the same validation results. But it is useful to illustrate that the complexity of the parametric analysis with LPT is close to the one of a non parametric model checking analysis. Indeed, algorithms which are implemented in LPT are all based on a dichotomy, that's why there is a slight influence compared to a non parametric verification. On each iteration in the algorithms, a single non parametric state graph is generated (with the TINA tools) which is limited in term of memory to the maximum size of a non parametric analysis. So, we do not use more memory than a non parametric verification, we just take more time because of the automation of the analysis.

On the contrary, the comparison with Romeo is interesting, as this tool allow the same type of analysis. Romeo is in fact more complete because it is not reduced to only one parameter. But the results of table 3 show that even with only one parameter in the analysis process, the parametric model checking is too complex to provide analysis results in a realistic case study. On the contrary, our solution implemented in the LPT tool gives a result even in the most complex situation, when the firing interval of the "parameterized" transition is large, leading to many interleavings with the other transitions of the model.

Even these numerical values give an empirical performance estimation, they show that LPT is a solution to obtain parametric validation results for real case studies, typically for industrial systems, when it is mandatory to obtain a result, even for strict validation hypotheses.

7. CONCLUSION

This article presents an analysis method for the parametric formal validation of Time Petri Nets models. The proposed algorithms have been implemented in an analysis tool: LPT (Little Parametric Tool). LPT implements algorithms to find the minimal and maximal parameter values verifying a given property. These algorithms are based on a dichotomous execution of non parametric validation to find the desired parameter values. Depending on the initial hypotheses, several useful information on the parameter values could be obtained.

The main characteristic of this work is that LPT is really efficient in a practical way: it allows the parametric validation of real systems with almost the same complexity than the non parametric analysis. This could be very interesting, especially in real-life context where theoretical parametric verification methods are often unusable because of their complexity.

However, the non parametric model checking problem is still confronted to combinatory explosion risks for complex systems. Particularly if the systems are substantially parallel, the interleaving problem remains. It could be then necessary to study abstraction and optimization techniques, in the analysis process but also in the model itself, before building the state graph.

Furthermore, the LPT analysis methods are limited to strong hypotheses: only one parameter could be considered, represented as the firing instant associated to only one transition. Moreover, the parameter values are exclusively integers. In the current version of our analysis method, it is not possible to introduce more than one parameter or to consider it as a real, since it is the dichotomy variable. Therefore, it could be interesting to consider parametric model checking methods as well, trying to optimize the parametric state graph, limiting the considered cases and properties, but improving the verification process efficiency.

ACKNOWLEDGEMENT

The work presented in this paper is a part of the Babylone project[7]. Thus, the authors would like to thank the DRIRE and the Conseil Régional Midi-Pyrénées.

REFERENCES

Alur, R. and Dill, D. (1994). A theory of timed automata, *Theoretical Computer Science* **126**(2): 183–235.

Alur, R., Henzinger, T. and Vardi, M. (1993). Parametric real-time reasoning, *25th Annual*

[7] http://www.critical-openware.org/

Symposium on Theory of Computing, pp. 592–601.

André, É. (2010). IMITATOR II: A tool for solving the good parameters problem in timed automata, *Proceedings of the 12th International Workshops on Verification of Infinite State Systems (INFINITY'10)*, Vol. 39 of *Electronic Proceedings in Theoretical Computer Science*, Singapore, pp. 91–99.

André, É., Chatain, Th., Encrenaz, E. and Fribourg, L. (2009). An inverse method for parametric timed automata, *International Journal of Foundations of Computer Science* **20**(5): 819–836.

Andreu, D., Souquet, G. and Gil, T. (2008). Petri net based rapid prototyping of digital complex system, *IEEE Computer Society Annual Symposium on VLSI (ISVLSI08)*, Montpellier, France.

Berthomieu, B. and Diaz, M. (1991). Modeling and verification of time dependent systems using time petri nets, *IEEE transactions on software engineering* **17**(3): 259–273.

Berthomieu, B. and Vernadat, F. (2006). Time petri nets analysis with tina, *Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*.

Bruyère, V., Dall'Olio, E. and Raskin, J. (2003). Durations,parametric model checking in timed automata with presburger arithmetic, *in* LNCS (ed.), *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, Vol. 2607, Berlin.

Bruyère, V. and Raskin, J. (2007). Real-time model-checking: Parameters everywhere, *Journal Logical Methods in Computer Science* **3**(1): 1–30.

Cassez, F. and Roux, O. (2006). Structural translation from time petri nets to timed automata, *Journal of Systems and Software* **79**(10): 1456–1468.

Delfieu, D., Sogbohossou, M., Traonouez, L. and Revol, S. (2007). Parameterized study of a time petri net, *Cybernetics and Information Technologies, Systems and Applications : CITSA 2007*, Orlando, Florida, USA.

Gardey, G., Lime, D., Magnini, M. and Roux, O. (2005). Romo: A tool for analyzing time petri nets analysis, *in* S. Lecture Notes in Computer Science (ed.), *17th International Conference on Computer Aided Verification (CAV'05)*, Edinburgh, Scotland, UK.

Gardey, G., Roux, O. and Roux, O. (2003). A zone-based method for computing the state space of a time petri net, *in* LNCS (ed.), *Formal Modeling and Analysis of Timed Systems (FORMAT'03)*, Vol. 2791, Marseille, France, pp. 246–259.

Godary, K. (2008). Lpt : little parametric tool, outil pour la validation d'une borne temporelleparamétrée, *Conférence Francophone Internationale d'Automatique*, Bucarest, Roumanie.

Hune, T., Romijn, J., Stoelinga, M. and Vaandrager, F. (2002). Linear parametric model checking of timed automata, *Journal of Logic and Algebraic Programming* **52-53**: 183–220.

Larsen, K., Pettersson, P. and Yi, W. (1997). Uppaal in a nutshell, *Journal of Software Tools for Technology Transfer* **1**: 134–152. www.uppaal.com.

Merlin, P. (1974). *A study of the recoverability of omputing systems*, PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA.

Sifakis, J. (1982). A unified approach for studying the properties of transition systems, *Theoretical Computer Science* **18**: 227–258.

Traonouez, L.-M., Lime, D. and Roux, O. H. (2009). Parametric model-checking of stopwatch petri nets, **15**(17): 3273–3304.

Wang, F. (2000). Parametric analysis of computer systems, *Form. Methods Syst. Des.* **17**(1): 39–60.