Parametric model checking timed automata under non-Zenoness assumption

Hoang Gia NGUYEN
LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité
99, avenue Jean-Baptiste Clément
F-93430 Villetaneuse, FRANCE
Email: Hoanggia.Nguyen@lipn.univ-paris13.fr

***Abstract*—Real-time systems often involve hard timing constraints and concurrency, and are notoriously hard to design or verify. Given a model of a real-time system and a property, parametric model-checking aims at synthesizing timing valuations such that the model satisfies the property. However, the counter-example returned by such a procedure may be *Zeno* (an infinite number of discrete actions occurring in a finite time), which is unrealistic. We show here that synthesizing parameter valuations such that at least one counterexample run is non-Zeno is undecidable for parametric timed automata (PTAs). Still, we propose a semi-algorithm based on a transformation of PTAs into *Clock Upper Bound PTAs* to derive all valuations whenever it terminates, and some of them otherwise.**

I. Introduction

Timed automata (TAs) [1] are a popular formalism for real-time systems modeling and verification, providing explicit manipulation of clock variables. Real-time behavior is captured by clock constraints on system transitions, setting or resetting clocks, etc. TAs have been studied in various settings (such as planning [16]) and benefit from powerful tools such as Uppaal [18] or PAT [20].

Model checking TAs consists of checking whether there exists an accepting cycle (i.e. a cycle that visits infinitely often a given set of locations) in the automaton made of the product of the TA modeling the system with the TA representing a violation of the desired property (often the negation of a property expressed, e.g. in CTL). However, such an accepting cycle does not necessarily mean that the property is violated: indeed, a known problem of TAs is that they allow Zeno behaviors. An infinite run is non-Zeno if it takes an unbounded amount of time; otherwise it is Zeno. Zeno runs are infeasible in reality and thus must be pruned during system verification. That is, it is necessary to check whether a run is Zeno or not so as to avoid presenting Zeno runs as counterexamples. The problem of checking whether a timed automaton accepts at least one non-Zeno run, i.e. the emptiness checking problem, has been tackled previously (e.g. [21], [22], [8], [12], [13], [23]).

It is often desirable not to fix *a priori* all timing constants in a TA: either for tuning purposes, or to evaluate robustness when clock values are imprecise. For that purpose, parametric timed automata (PTAs) extend TAs with parameters [2]. Although most problems of interest are undecidable for PTAs [3], some (semi-)algorithms were proposed to tackle practical parameter synthesis (e.g. [4], [17], [15], [7]). We address here the synthesis of parameter valuations for which there exists a non-Zeno cycle in a PTA; this is highly desirable when performing parametric model-checking for which the parameter valuations violating the property should not allow only Zeno-runs. As far as the authors know, this is the first work on parametric model checking of timed automata with the non-Zenoness assumption. Just as for TAs, the parametric zone graph of PTAs (used in e.g. [14], [4], [15]) cannot be used to check whether a cycle is non-Zeno. Therefore, we propose here a technique based on *clock upper bound PTAs* (CUB-PTAs), a subclass of PTAs satisfying some syntactic restriction, and originating in CUB-TAs for which the non-Zeno checking problem is most efficient [23]. In contrast to regular PTAs, we show that synthesizing valuations for CUB-PTAs such that there exists an infinite non-Zeno cycle can be done based on (a light extension of) the parametric zone graph.

Outline: Section II recalls the necessary preliminaries. We then present the concept of CUB-PTAs (Section III), and show how to transform a PTA into a list of CUB-PTAs. Zeno-free parametric model-checking of CUB-PTA is addressed in Section IV, and experiments reported in Section V. Finally, Section VI concludes and gives perspectives for future work.

II. Preliminaries

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$, where: i) Σ is a finite set of actions, ii) L is a finite set of locations, iii) $l_0 \in L$ is the initial location, iv) X is a set of clocks, v) P is a set of parameters, vi) K_0 is the initial parameter constraint, vii) I is the invariant, assigning to every $l \in L$ a guard $I(l)$, viii) E is a set of edges $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a guard.*

The initial constraint K_0 is used to constrain some parameters (as in, e.g. [14], [4]); in other words, it defines a domain of valuation for the parameters. For example, given two parameters $p_{\min}$ and $p_{\max}$, we may want to ensure that $p_{\min} \leq p_{\max}$. Given $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$, we write

$\mathcal{A}.K_0$ as a shortcut for the initial constraint of $\mathcal{A}$. In addition, given K'_0, we denote by $\mathcal{A}(K'_0)$ the PTA where $\mathcal{A}.K_0$ is replaced with K'_0.

Observe that, as in [23], we do not define accepting locations. In our work, we are simply interested in computing valuations for which there is a non-Zeno cycle. A more realistic parametric model checking approach would require additionally that the cycle is accepting, i.e. it contains at least one accepting location. However, this has no specific theoretical interest, and would impact the readability of our exposé.

Given a parameter valuation $v \models \mathcal{A}.K_0$, we denote by $[\![\mathcal{A}]\!]_v$ the non-parametric TA where all occurrences of a parameter p_i have been replaced by $v(p_i)$.

III. CUB-PARAMETRIC TIMED AUTOMATA

A. CUB timed automata

In [23], the authors identified a subclass of TAs called CUB-TAs for which non-Zenoness checking based on the symbolic semantics is feasible. Furthermore, they show that an arbitrary TA can be transformed into a CUB-TA. Based on their work, we first show that arbitrary PTAs can be transformed into a parametric version of CUB-TAs, and then solve the non-Zeno synthesis problem based on parametric CUB-TAs.

B. CUB parametric timed automata

We extend the definition of CUB-TAs to parameters as follows:

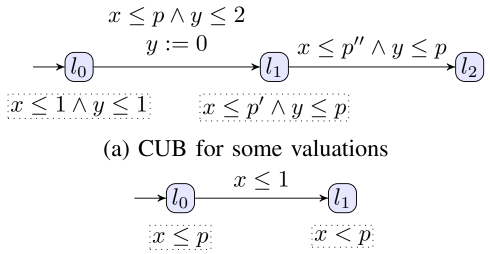
Definition 2. *A PTA $\mathcal{A}$ is a* CUB-PTA*, iff there exists a constraint $\mathcal{A}.K_0$ on parameters that guarantees every clock has a non-decreasing upper bound along any path before it is reset, for all parameter valuations satisfying the initial constraint $\mathcal{A}.K_0$*

Example 1. *Consider the PTA $\mathcal{A}$ in Fig. 1a s.t. $\mathcal{A}.K_0 = \top$. Then $\mathcal{A}$ is not CUB: for x, the upper bound in l_0 is $x \leq 1$ whereas that of the guard on the transition outgoing l_0 is $x \leq p$. $(1, \leq) \leq (p, \leq)$ yields $1 \leq p$. Then, $\top \not\subseteq \big(1 \leq p\big)$; for example, $p = 0$ does not satisfy $1 \leq p$.*

Consider again the PTA $\mathcal{A}$ in Fig. 1a, this time assuming that $\mathcal{A}.K_0 = (p = 1 \wedge 1 \leq p' \wedge p' \leq p'')$. This PTA is a CUB-PTA. (The largest constraint K_0 making this PTA a CUB will be computed in Example 2.)

C. CUB PTA detection

Given an arbitrary PTA, our approach works as follows. Firstly, we check whether it is a CUB-PTA for some valuations. If it is, we proceed to the synthesis problem, using our cycle detection synthesis algorithm; however, the result may be partial, as it will only be valid for the valuations for which the PTA is CUB. This incompleteness may come at the benefit of a more efficient synthesis. If it is CUB for no valuation, it has to be transformed into an equivalent CUB-PTA (which will be considered in Section III-D).


$x \leq p \wedge y \leq 2$
$y := 0$
l_0
l_1
$x \leq p'' \wedge y \leq p$
l_2
$x \leq 1 \wedge y \leq 1$
$x \leq p' \wedge y \leq p$
$x \leq 1$
l_0
l_1
$x \leq p$
$x < p$

(a) CUB for some valuations

(b) CUB for no valuations

Figure 1: Examples of PTAs to illustrate the CUB concept

Our procedure to detect whether a PTA is CUB for some valuations. For each edge in the PTA, we enforce the CUB condition on each clock by constraining the upper bound in the invariant of the source location to be smaller than or equal to the upper bound of the edge guard. Additionally, if the clock is not reset along this edge, then the upper bound of the source location invariant should be smaller than or equal to that of the target location. If the resulting set of constraints accepts parameter valuations (i.e. is not empty), then the PTA is a CUB-PTA for these valuations.

Example 2. *Consider again the PTA $\mathcal{A}$ in Fig. 1a, assuming that $\mathcal{A}.K_0 = \top$. This PTA is CUB for $1 \leq p \wedge 1 \leq p' \wedge p' \leq p''$.*

D. Transforming a PTA into a disjunctive CUB-PTA

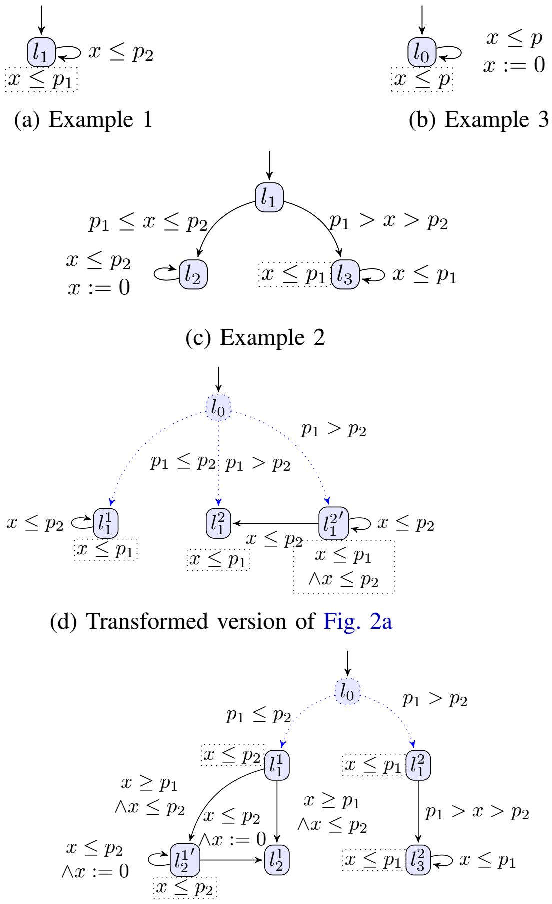
In this section, we show that an arbitrary PTA can be transformed into an extension of CUB-PTAs (namely *disjunctive CUB-PTA*), while preserving the symbolic runs.

Definition 3. *A* disjunctive CUB-PTA *is a list of CUB-PTAs.*

Example 3. *In Fig. 2d (without the dotted, blue elements), two CUB-PTAs are depicted, one (say $\mathcal{A}_1$) on the left with locations superscripted by 1, and one (say $\mathcal{A}_2$) on the right superscripted with 2. Assume $\mathcal{A}_1.K_0$ is $p_1 \leq p_2$ and $\mathcal{A}_2.K_0$ is $p_1 > p_2$. Then the full Fig. 2d (including dotted elements) is the PTA associated with the disjunctive CUB-PTA made of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

The key idea behind the transformation from a TA into a CUB-TA in [23] is as follows: whenever a location l is followed by an edge e and a location l' for which an upper bound of clock x in edge g or in location l' is smaller than one in location l for some x if x is not a reset clock on the edge e, otherwise the upper bound of clock x in edge g is smaller than one in location l, location l is split into two locations: one (say l_1) with a "decreased upper bound", that is then connected to l'; and one (say l_2) with the same invariant as in l, and with no transition to l'. Therefore, the original behavior is maintained.

Here, we extend this principle to CUB-PTAs. A major difference is that, in the parametric setting, comparing two clock upper bounds does not give a Boolean answer but a parametric answer. For example, in a TA, $(2, \leq) \leq (3, <)$ holds (this is true), whereas in a PTA $(p_1, \leq) \leq (p_2, <)$ denotes the *constraint* $p_1 < p_2$. Therefore, the principle of our transformation is that, whenever we have to compare two



(a) Example 1

(b) Example 3

(c) Example 2

(d) Transformed version of Fig. 2a

(e) Transformed version of Fig. 2c

Figure 2: Examples: detection of and transformation into CUB-PTAs

parametric clock upper bounds, we consider both cases: here either $p_1 < p_2$ (in which case the first location does not need to be split) or $p_1 \geq p_2$ (in which case the first location shall be split). This yields a finite list of CUB-PTAs: each of these CUB-PTAs consists in one particular ordering of all parametric linear terms used as upper bounds in guards and invariants.

Example 4. *Let us transform the PTA in Fig. 2a: if $p_1 \leq p_2$ then the PTA is already CUB, and l_1 does not need to be split. This yields a first CUB-PTA, depicted on the left-hand side of Fig. 2d. However, if $p_1 > p_2$, then l_1 needs to be split into $l_1^{2\prime}$ (where time cannot go beyond p_2) and into l_1^2 (where time can go beyond p_2, until p_1), but the self-loop cannot be taken anymore (otherwise the associated guard makes the PTA not CUB). This yields a second CUB-PTA, depicted on the right-hand side of Fig. 2d. Both make a disjunctive CUB-PTA equivalent to Fig. 2a.*

Similarly, we give the transformation of Fig. 2c in Fig. 2e.

IV. Zeno-free cycle synthesis in CUB-PTAs

Taking a disjunctive CUB-PTA as input, we show in this section that synthesizing the parameter valuations for which there exists at least one non-Zeno cycle (and therefore an infinite non-Zeno run) reduces to an SCC (strongly connected component) synthesis problem.

First, we define a light extension of the parametric zone graph as follows. The *extended parametric zone graph* of a PTA $\mathcal{A}$ is identical to its parametric zone graph, except that any transition $(\mathbf{s}, e, \mathbf{s}')$ is replaced with $(\mathbf{s}, (e, b), \mathbf{s}')$, where b is a Boolean flag which is true if time can *potentially* elapse between $\mathbf{s}$ and $\mathbf{s}'$.

Consider the example in Fig. 2b. After taking one loop, we have that $x_0 \leq p$: therefore, x_0 is not necessarily 0, and b is true. But consider v such that $v(p) = 0$: then in l_1 time can never elapse.

V. Experiments

We compare three approaches: 1) A cycle detection synthesis without the non-Zenoness assumption (called synthCycle). The result may be an over-approximation of the actual result, as some of the parameters synthesized may yield only Zeno cycles. If synthCycle does not terminate, its result is an under-approximation of an over-approximation, therefore considered as potentially invalid; that is, there is no guarantee of correctness for the synthesized constraint. 2) Our CUB-detection followed by synthesis : the result may be under-approximated, as only the valuations for which the PTA is CUB are considered. 3) Our CUB-transformation followed by synthesis on the resulting disjunctive CUB-PTA. If the algorithm terminates, then the result is exact, otherwise it may be under-approximated.

We consider various benchmarks: protocols (CSMA/CD, Fischer [2], RCP, WFAS), hardware circuits (And-Or, flip-flop), scheduling problems (Sched5), a networked automation system (simop) and various academic benchmarks.

We give from left to right in Table I the case study name and its number of clocks, parameters and locations. For $synthCycle$, we give the computation time (TO denotes a time-out at 3600 s), the constraint type ($\bot$, $\top$ or another constraint) and the validity of the result: if $synthCycle$ terminates, the result is an over-approximation, otherwise it is potentially invalid. For $CUBdetect$ (resp. $CUBtrans$) we give the detection (resp. transformation) time, the total time (including synthNZ), the result, and whether it is an under-approximation or an exact result. We also mention whether $CUBdetect$ outputs that all, none or some valuations make the PTA CUB; and we give the number of locations in the transformed disjunctive CUB-PTA output by $CUBtrans$. The percentage is used to compare the number of valuations (comparison obtained by discretization) output by the algorithms, with $CUBtrans$ as the basis (as the result is exact).

The toy benchmark CUBPTA1 is a good illustration: $CUBtrans$ terminates after 0.073 s (and7 therefore its result is exact) with some constraint. $CUBdetect$ is faster (0.015 s) but infers that only some valuations are CUB and analyzes only these valuations; the synthesized result is only 69 % of the expected result. In contrast, $synthCycle$ is much

Model				*synthCycle*			*CUBdetect*					*CUBtrans*				
Name	**#** X	**#** P	**#** L	**t (s)**	**Result**	**Appr.**	**Detec** **t (s)**	**Total** **t (s)**	**CUB** **for**	**Result**	**Appr.**	**Trans** **t (s)**	**Total** **t (s)**	**#L** **CUB**	**Result**	**Appr.**
CSMA/CD	3	3	28	TO	$\top$	invalid?	0.013	0.013	$\bot$	-	-	0.300	TO	74	$\top$	exact
Fischer	2	4	13	TO	$\top$	invalid?	0.003	0.003	$\bot$	-	-	0.012	TO	20	$\top$	exact
RCP	6	5	48	TO	Some	invalid?	0.013	0.013	$\bot$	-	-	0.348	TO	71	$\bot$	under
WFAS	4	2	10	TO	Some 102%	invalid?	0.009	0.009	$\bot$	-	-	0.246	1848	40	Some 100%	exact
AndOr	4	4	27	TO	Some 166%	invalid?	0.012	0.012	$\bot$	-	-	0.059	TO	34	Some 100%	under
Flip-flop	5	2	52	0.058	$\bot$	exact	0.002	0.086	$\top$	$\bot$	exact	0.010	0.972	58	$\bot$	exact
Sched5	21	2	153	190	$\bot$	exact	0.051	0.051	$\bot$	-	-	1.180	TO	180	$\bot$	under
simop	8	2	46	TO	$\bot$	invalid?	0.012	0.012	$\bot$	-	-	0.219	TO	81	$\bot$	under
train-gate	5	9	11	TO	$\bot$	invalid?	0.000	TO	Some	$\bot$	under	0.059	TO	23	$\bot$	under
coffee	2	3	4	TO	Some 100%	invalid?	0.000	TO	Some	Some 100%	under	0.012	TO	10	Some 100%	under
CUBPTA1	1	3	2	0.006	$\top$ 208%	over	0.000	0.015	Some	Some 69%	under	0.006	0.073	6	Some 100%	exact
JLR13	2	2	2	TO	$\bot$	invalid?	0.000	TO	$\top$	$\bot$	under	0.000	TO	3	$\bot$	under

Table I: Experimental comparison of the three algorithms

faster (0.006 s) but obtains too many valuations (208 % of the expected result) as it infers many Zeno valuations.

Note that flip-flop is a hardware circuit modeled using a bi-bounded inertial delay, and is therefore CUB for all valuations.

An interesting benchmark is WFAS, for which our transformation procedure terminates whereas synthCycle does not. Therefore, we get an exact result while the traditional procedure cannot produce any valuable output.

As a conclusion, CUBdetect seems to be faster but less complete than CUBtrans. As for CUBtrans, its result is almost always more valuable than synthCycle, and therefore is the most interesting algorithm.

VI. Conclusion

We proposed a technique to synthesize valuations for which there exists a non-Zeno infinite run in a PTA. By adding accepting states, this allows for parametric model checking with non-Zenoness assumption. Our techniques rely on a transformation to a disjunctive CUB-PTA (or in some cases on a simple detection of the valuation for which the PTA is already CUB), and then on a dedicated cycle synthesis algorithm. We implemented our techniques in IMITATOR and compared our algorithms on a set of benchmarks.

a) Future works: Our technique relying on CUB-PTAs extends the technique of CUB-TAs: this technique is shown in [23] to be the most efficient for performing non-Zeno model checking for TAs. However, for PTAs, other techniques (such as yet to be defined parametric extensions of strongly non-Zeno TAs [22] or guessing zone graph [13]) could turn more efficient and should be investigated.

References

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

[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[3] É. André. What's decidable about parametric timed automata? In *FTSCS*, CCIS, pages 52–68. Springer, 2015.

[4] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *IJFCS*, 20(5):819–836, 2009.

[5] É. André, D. Lime, and O. H. Roux. Decision problems for parametric timed automata. In *ICFEM*, volume 10009 of *LNCS*, pages 400–416. Springer, 2016.

[6] É. André, Y. Liu, J. Sun, and J. S. Dong. Parameter synthesis for hierarchical concurrent real-time systems. *Real-Time Systems*, 50(5-6):620–679, 2014.

[7] L. Aştefănoaei, S. Bensalem, M. Bozga, C. Cheng, and H. Ruess. Compositional parameter synthesis. volume 9995 of *LNCS*, pages 60–68, 2016.

[8] H. Bowman and R. Gómez. How to stop time stopping. *Formal Aspects of Computing*, 18(4):459–493, 2006.

[9] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *FMCAD*, pages 165–168. IEEE, 2013.

[10] J. S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi. Timed automata patterns. *IEEE Transactions on Software Engineering*, 34(6):844–859, 2008.

[11] S. Evangelista, A. Laarman, L. Petrucci, and J. van de Pol. Improved multi-core nested depth-first search. In *ATVA*, volume 7561 of *LNCS*, pages 269–283. Springer, 2012.

[12] R. Gómez and H. Bowman. Efficient detection of Zeno runs in timed automata. In *FORMATS*, volume 4763 of *LNCS*, pages 195–210. Springer, 2007.

[13] F. Herbreteau, B. Srivathsan, and I. Walukiewicz. Efficient emptiness check for timed Büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.

[14] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. *JLAP*, 52-53:183–220, 2002.

[15] A. Jovanović, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. *Transactions on Software Engineering*, 41(5):445–461, 2015.

[16] L. Khatib, N. Muscettola, and K. Havelund. Mapping temporal planning constraints into timed automata. In *TIME*, pages 21–27. IEEE Computer Society, 2001.

[17] M. Knapik and W. Penczek. Bounded model checking for parametric timed automata. *Transactions on Petri Nets and Other Models of Concurrency*, 5:141–159, 2012.

[18] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *STTT*, 1(1-2), 1997.

[19] S. Schupp, E. Ábrahám, X. Chen, I. B. Makhlouf, G. Frehse, S. Sankaranarayanan, and S. Kowalewski. Current challenges in the verification of hybrid systems. In *CyPhy*, volume 9361 of *LNCS*, pages 8–24. Springer, 2015.

[20] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *CAV*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009.

[21] S. Tripakis. Verifying progress in timed systems. In *AMAST*, pages 299–314, 1999.

[22] S. Tripakis, S. Yovine, and A. Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

[23] T. Wang, J. Sun, X. Wang, Y. Liu, Y. Si, J. S. Dong, X. Yang, and X. Li. A systematic study on explicit-state non-Zenoness checking for timed automata. *IEEE Transactions on Software Engineering*, 41(1):3–18, 2015.