Advances on discrete parameter synthesis in continuous-time formalisms

ANR PACS Deliverable 3

Étienne André, Benoît Delahaye, Didier Lime, Laure Petrucci

May 15, 2018

1 Context and objectives

In this task, we consider systems featuring both discrete parameters and quantitative timing features. These timing features can be either described as constants, or parameters. The goal is to propose models, investigate their decidability, write (semi-)algorithms, and implement them.

Our objectives were as follows:

1. Develop models to handle discrete parameters and non-parametric timing information, and exhibit decidable subclasses;
2. Develop models to handle discrete parameters and parametric timing information, and exhibit decidable subclasses;
3. Develop algorithms and tools to synthesise parameters in these two settings.

2 Parametric timed systems with action discrete parameters

First, we considered (parametric) timed systems, with additional discrete parameters, that play the role of *controllable* actions. That is, we considered an extension of (parametric) timed automata [AD94, AHV93]. Timed automata [AD94] extend finite-state automata with clocks, i. e., real-valued variables that can be compared to constants in guards, and reset along transitions. Parametric timed automata (PTA) [AHV93] allow to replace constants with unknown parameters in timing constraints.

Among parametric synthesis approaches, two are of particular interest here as they consider different and complementary kinds of parameters: time (e. g., [AHV93]) and actions (e. g., [KMP15]). Operating on an extension of timed automata where clocks can be compared to parameters (i. e., PTA), and properties of the system, synthesis of timing parameters provides a set of constraints the timing parameters should satisfy for the property to hold. On the other hand, starting from an (untimed) automaton and a property, action synthesis examines all possible behaviors in order to deduce which sets of actions should be enabled or disabled for the property to hold.

It is thus quite obvious that these two approaches are complementary as they consider very separate aspects of a system. Therefore, combining them is appealing. Furthermore, such a combination is worthy within a practical design process. Indeed, the designer of a system may be faced with multiple design choices, controlling some actions, or selecting components with specific time characteristics. Our aim by combining action and time synthesis is to provide designers with tools supporting their decision making. For example components with different timings might have different prices while controlling some actions is not always easy to set. Providing constraints on both time and actions allows to improve the choices according to criteria like components cost or feasibility.

Contribution In [AKPP16], we introduced a framework to synthesize both actions (seen as parameters) and timing constants (seen as another kind of parameters) for safety properties. We proposed a procedure that, given a set of discrete states ("locations") to avoid, synthesizes a constraint on action and timing parameters such that, for any timing parameter valuation satisfying this constraint, and for any set of actions enabled or disabled according to the constraint, the system is safe, i. e., the discrete states to avoid are not reachable. We chose as a basis formalism of an extension of parametric timed automata (PTA), where actions can be (statically) controlled, i. e., may be enabled or disabled (once for all).

While the general class is undecidable, we proposed a semi-algorithm, i. e., a procedure that may not terminate, but that is sound and complete if it terminates.

We applied our approach to a railway crossing system example with a malicious intruder potentially threatening the system safety. We showed that, depending on timing constants chosen, some actions (e. g., the fact that the intruder can commit certain actions) may have to be disabled in order to guarantee the system safety.

Finally, our approach was implemented in a framework making use of Spatula [KMP15] and IMITATOR [AFKS12].

3 Unbounded networks of parametric timed systems

In a second direction of research, we proposed to combine two different types of parameters, namely the *number of identical processes* and the *timing features*, and studied the decidability of classic parametric decision problems in the resulting formalism. Both types of parameters, when introduced separately in timed automata-based formalisms, result in hard problems undecidable even in restricted settings.

The most basic verification question for PTA, "does there exist a value for the parameters such that some location is reachable" is undecidable with as few as 1 integer- or rational-valued parameter [Mil00, BBLS15], or when only 1 clock is compared to a unique parameter [Mil00] (with additional clocks). The main syntactic subclass of PTA for which decidability is obtained is L/U-PTA [HRSV02], in which the parameters set is partitioned into lower-bound parameters (i. e., parameters always compared as a lower bound in a clock guard) and upper-bound parameters (always as upper bounds). L/U-PTA have been shown [HRSV02] to be expressive enough to model classical examples from the literature, such as root contention or Fischer's mutual exclusion algorithm for instance.

Broadcast protocol networks [DSTZ12, DSZ11a, DSZ11b, DSZ10], allow treating the size of a network as an unknown parameter. Here also the most simple basic verification question "does there exist a value for the parameter such that some location

is reachable by a process" is undecidable when considering arbitrary communication topologies [DSZ10]. However one can regain decidability by considering different communication topology settings. One option is to limit the topologies to cliques (every process receives every messages) [DSZ11a, DSZ11b, DSZ10]. Another is to consider reconfigurable broadcasts in which the set of receivers is chosen nondeterministically at each step [DSTZ12]. A timed version of this broadcast protocol was studied in [ADR+16]. In the clique topology for this network, the reachability problem is decidable only when there is a single clock per process.

Contributions In [ADFL18], we provided one more level of abstraction to the formalisms of the literature by proposing *parametric timed broadcast protocols* (PTBP), i. e., a new formalism made of an arbitrary number of identical timed processes in which timing parameters can be used. Considering those two kinds of parameters could be really useful, for example when designing and verifying communication protocols. Indeed, those protocols are required to work independently of the number of participants (hence the parametric size of networks) and the time constraints in each process are of paramount importance and thus could be tweaked in early development thanks to timing parameters. This work is, up to our knowledge, the first to consider the combination of both a parametric network size and timing parameters in clock guard constraints.

We considered the following problems: does there exist a number of processes for which the set of timing parameter valuations allowing to reach a given location for one run ("EF"), or for all runs ("AF") is empty (or universal)? This gives rise to 4 problems: EF-emptiness, EF-universality, AF-emptiness and AF-universality. As PTBP can be seen as an extension of both broadcast protocols and parametric timed automata, undecidability follows immediately from the existing undecidability results known for these two formalisms. However, combining decidable subclasses of both formalisms is challenging, and does not necessarily make the EF and AF problems decidable for PTBP.

The communication topology is of utmost importance in broadcast protocols, and we therefore investigated reachability problems depending on the broadcast semantics. In the reconfigurable semantics (where the set of receivers is chosen nondeterministically), AF-emptiness and AF-universality are decidable for 1-clock PTBP, and undecidable from 3 clocks even for L/U-PTBP with the same parameters partitioning as in L/U-PTA (the 2-clock case is equivalent to a well-known open problem for PTA). The AF results may not seem surprising, as they resemble equivalent results for PTA. However, EF-emptiness and EF-universality becomes undecidable even for 1-clock PTBP: this result comes in contrast with both non-parametric timed broadcast protocols and PTA for which the 1-clock case is decidable.

In the clique semantics (where every message reaches every process), we showed that AF problems are undecidable even without any clock. Then, as it is known that 2 clocks (and no parameter) yield undecidability, we studied EF problems over 1 clock. We investigated the decidability status depending on whether the timing parameters in guards appear only as upper bounds in guards (U-PTBP), as lower bounds (L-PTBP) or when the set of parameters is partitioned in lower-bound and upper-bound parameters (L/U-PTBP). We showed that L/U-PTBP become decidable for EF-emptiness (but not universality) when the parameter domain is bounded. For EF-universality, decidability is obtained only for L-PTBP and U-PTBP for a parameter domain bounded with closed bounds.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[ADFL18] Étienne André, Benoît Delahaye, Paulin Fournier, and Didier Lime. Parametric timed broadcast protocols. Submitted, 2018.

[ADR+16] Parosh A. Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612:1–22, 2016.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, August 2012.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[AKPP16] Étienne André, Michał Knapik, Wojciech Penczek, and Laure Petrucci. Controlling actions and time in parametric timed automata. In Jörg Desel and Alex Yakovlev, editors, *ACSD*, pages 45–54. IEEE Computer Society, 2016.

[BBLS15] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, July 2015.

[DSTZ12] Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *International Conference on Concurrency Theory*, pages 313–327. Springer, 2010.

[DSZ11a] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS*, volume 11, pages 441–455. Springer, 2011.

[DSZ11b] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of safety properties in ad hoc network protocols. *arXiv preprint arXiv:1108.1864*, 2011.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[KMP15] Michał Knapik, Artur Męski, and Wojciech Penczek. Action synthesis for branching time logic: Theory and applications. *ACM Transactions on Embedded Computing*, 14(4), 2015.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

Controlling Actions and Time in Parametric Timed Automata

Étienne André*†, Michał Knapik‡, Wojciech Penczek‡§ and Laure Petrucci*
*LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, F-93430, Villetaneuse, France
†École Centrale de Nantes, IRCCyN, CNRS, UMR 6597, France
‡Institute of Computer Science, PAS, Warsaw, Poland
§University of Natural Sciences and Humanities, II, Siedlce, Poland

***Abstract*—Cyber-physical systems involve both discrete actions and real-time that elapses depending on timing constants. In this paper, we introduce a formalism for such systems containing both real-time parameters in linear timing constraints and switchable (Boolean) actions. We define a new approach for synthesizing combinations of parameter valuations and allowed actions, under which the system correctness is ensured when expressed in the form of a safety property. We apply our approach to a railway crossing system example with a malicious intruder potentially threatening the system safety.**

***Keywords*—Time synthesis, action synthesis, parametric timed automata, parametric safety**

I. Introduction

Model checking consists in formally verifying whether a model of a system satisfies a property expressed using a formula. Beyond model checking, parameter synthesis consists in synthesizing valuations for some parameters so that the system satisfies the formula. Several recent works tackle parameter synthesis problems as they provide flexibility in the design of real-time systems while still guaranteeing their expected properties.

Among parametric synthesis approaches, two are of particular interest here as they consider different and complementary kinds of parameters: time (*e.g.* [AHV93]) and actions (*e.g.* [KMP15]). Operating on an extension of timed automata (finite state automata extended with clocks) where clocks can be compared to parameters, and properties of the system, synthesis of timing parameters provides a set of constraints the timing parameters should satisfy for the property to hold. On the other hand, starting from an (untimed) automaton and a property, action synthesis examines all possible behaviours in order to deduce which sets of actions should be enabled or disabled for the property to hold.

It is thus quite obvious that these two approaches are complementary as they consider very separate aspects of a system. Therefore, combining them is appealing. Furthermore, such a combination is worthy within a practical design process. Indeed, the designer of a system may be faced with multiple design choices, controlling some actions, or selecting components with specific time characteristics. Our aim by combining action and time synthesis is to provide designers with tools supporting their decision making. For example components with different timings might have different prices while controlling some actions is not always easy to set. Providing constraints on both time and actions allows to improve the choices according to criteria like components cost or feasibility.

Contribution: In this paper, we introduce a framework to synthesize both actions (seen as parameters) and timing constants (seen as another kind of parameters) for safety properties. We propose a procedure that, given a set of discrete states ("locations") to avoid, synthesizes a constraint on action and timing parameters such that, for any timing parameter valuation satisfying this constraint, and for any set of actions enabled or disabled according to the constraint, the system is safe, *i.e.* the discrete states to avoid are not reachable. We choose as a basis formalism of an extension of parametric timed automata (PTA), where actions can be (statically) controlled, *i.e.* may be enabled or disabled (once for all). Our procedure is a semi-algorithm, *i.e.* it is not guaranteed to terminate, but the result is correct whenever it does. We apply our approach to a railway crossing system example with a malicious intruder potentially threatening the system safety. We show that, depending on timing constants chosen, some actions (*e.g.* the fact that the intruder can commit certain actions) may have to be disabled in order to guarantee the system safety.

Related works: Synthesis of timing parameters for PTA has recently drawn a lot of attention: procedures (algorithms or semi-algorithms) were proposed to synthesize all parameters leading to some location [AHV93], preserve the time language [ACEF09], synthesize integer-valued parameters using bounded model checking techniques [KP12], or synthesize bounded valuations satisfying reachability or unavoidability [JLR15], [ALR15].

In addition to the action parameter synthesis proposed in [KMP15], synthesis of discrete parameters is often understood in terms of number of processes; the goal is to prove that a system is correct for any number of processes (possibly beyond a threshold). Common techniques include regular model checking (see *e.g.* [Abd12]) and verification of many identical processes (see *e.g.* [DSZ10]). Action synthesis is also conceptually related to the problem of (discrete) controller synthesis, such as in the Ramadge-Wonham framework [RW87].

This work is partially supported by the CNRS/PAN project "BehaPPi-BMC" (2015–2016), and by the ANR national research program "PACS" (ANR-14-CE28-0002).

This is the author version of the manuscript of the same name published in the proceedings of ACSD 2016. The final publication is available at http://ieeexplore.ieee.org/.

However, little has been done to combine different types of parameters, typically discrete (actions) and continuous (timing) together. A notable exception is [DKRT97] where constraints are derived to ensure the correctness of the bounded retransmission protocol (BRP); one of them involves a discrete parameter (an integer-valued maximum number of retransmissions) multiplied by a continuous timing parameter. However, the procedure proposed seems to be specific to this case study.

Outline of the paper: First, Section II recalls the notions and techniques our approach bases on, *i.e.* Parametric Timed Automata, time parameter synthesis and action synthesis. Then, Section III shows how they can be combined for synthesis of both time and actions. This approach is put into practice on a detailed example in Section IV, using tool support that provides adequate results for an engineer to tune his/her system design. Finally, Section V summarises the contributions and opens some perspectives.

II. Background

In this section, we recall the notions and techniques necessary to build our synthesis approach. First, the basic definitions of *Parametric Timed Automata* which base on the use of *clocks* and *parameters* are recalled. PTA are extended to APTA (*Action-controllable Parametric Timed Automata*) that constitute a unified framework for time and action synthesis. The synthesis of time parameters leads to expressing *constraints* on their values in order to guarantee that the model satisfies the expected properties. Then, synthesis of actions is presented that allows for pruning the paths that invalidate properties by disabling controllable actions.

A. Clocks, Parameters, Actions, and Constraints

Let $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ denote the sets of Booleans, non-negative integers, integers, non-negative rational numbers, non-negative real numbers, respectively.

Throughout this paper, we assume a set $X = \{x_1, \ldots, x_H\}$ of *clocks*, *i.e.* real-valued variables that evolve at the same rate. A *clock valuation* is a function $w : X \to \mathbb{R}_+$. We identify a clock valuation w with the *point* $(w(x_1), \ldots, w(x_H))$. We write $X = 0$ for $\bigwedge_{1 \leq i \leq H} x_i = 0$. Given $d \in \mathbb{R}_+$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$. We also use a special zero-clock x_0, always equal to 0 (as in, *e.g.* [HRSV02]).

Let $P = \{p_1, \ldots, p_M\}$ be a set of *timing parameters*, *i.e.* unknown timing constants. A *timing parameter valuation* v is a function $v : P \to \mathbb{Q}_+$. We identify a valuation v with the *point* $(v(p_1), \ldots, v(p_M))$.

Let *ActVars* be a set of *action variables*. An *action valuation* is a function $\xi : \mathit{ActVars} \to \mathbb{B}$. Given ξ and an action variable α, we say that α is *enabled* in ξ if $\xi(\alpha) = \texttt{true}$, and *disabled* otherwise. The set of all action valuations is denoted by $\mathit{ActVals}$. With a slight notational abuse we sometimes treat ξ as the set of the actions enabled by this valuation.

In the following, let xplt denote a linear term over $X \cup P$ of the form $\sum_{1 \leq i \leq H} \gamma_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in X$, $p_i \in P$, and $\gamma_i, \beta_j, d \in \mathbb{Z}$. Let plt denote a parametric linear term over P, that is a linear term without clocks ($\gamma_i = 0$ for all i).

An *AXT-constraint* (or constraint on the action parameters, clock parameters, and timing parameters) over $\mathit{ActVars} \cup X \cup P$ is defined by the following grammar:

$$\phi := \phi \wedge \phi \mid \neg\phi \mid \alpha \mid \mathit{xplt} \sim 0,$$

where $\sim \in \{<, \leq, \geq, >\}$, $\alpha \in \mathit{ActVars}$. Disjunction ($\vee$) is defined as usual; observe that, due to the negation, we allow for non-convex constraints. An *axt-valuation* is a function $at : \mathit{ActVars} \cup X \cup P \to \mathbb{B} \cup \mathbb{R}_+ \cup \mathbb{Q}_+$ such that $at(\alpha) \in \mathbb{B}$ for all $\alpha \in \mathit{ActVars}$, $at(x) \in \mathbb{R}_+$ for all $x \in X$, and $at(p) \in \mathbb{Q}_+$ for all $p \in P$.

A *zone* C is a convex AXT-constraint over $X \cup P$ (hence without action variables) such that each of its linear conjuncts can be written in the form $x_i - x_j \sim \mathit{plt}$, where $x_i, x_j \in X \cup \{x_0\}$. A *guard* g is a zone such that each of its linear conjuncts can be written in the form $x_i \sim \mathit{plt}$.

Given a timing parameter valuation v and a clock valuation w, we denote by $w|v$ the valuation over $X \cup P$ such that for all clocks x, $w|v(x) = w(x)$ and for all timing parameters p, $w|v(p) = v(p)$. Given a zone C, we use the notation $w|v \models C$ to indicate that valuating each clock variable x with $w(x)$ and each timing parameter p with $v(p)$ within C, evaluates to true. We say that C is *satisfiable* if $\exists w, v$ s.t. $w|v \models C$. We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over $X \cup P$ obtained from C by delaying all clocks by an arbitrary amount of time; this can be obtained by adding a new variable to all clocks, ensuring that this variable is non-negative, and eliminating it (see, *e.g.* [ACEF09]). Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. We denote by $C{\downarrow_P}$ the projection of C onto P, *i.e.* obtained by eliminating the clock variables (*e.g.* using Fourier-Motzkin elimination).

A *P-constraint* K is an AXT-constraint over P defined by inequalities of the form $\mathit{plt} \sim 0$. We denote by $\top_P$ (resp. $\bot_P$) the parametric constraint that corresponds to the set of all possible (resp. the empty set of) timing parameter valuations.

An *A-constraint* is an AXT-constraint that contains actions only. An a-valuation is defined naturally from actions to $\mathbb{B}$. We denote by $\top_A$ the A-constraint $\bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \texttt{true}$, and by $\bot_A$ the A-constraint $\bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \texttt{false}$. Let $at_\top$ denote the a-valuation assigning `true` to all variables.

An *AT-constraint* is an AXT-constraint without clock, *i.e.* with xplt instead of plt in the grammar. An at-valuation is defined similarly to axt-valuation, but without clocks. Given an AT-constraint AT and an at-valuation at, we write $at \models AT$ if the expression obtained by replacing in AT any parameter and action variable by its valuation as in at evaluates to true. We denote by $\top_{AT}$ the at-constraint corresponding to the set of at-valuations $\{at \mid \forall \alpha \in \mathit{ActVars} : at(\alpha) = \texttt{true}\}$ (*i.e.* the set of valuations for which all timing parameter valuations are possible, and all action variables evaluate to true); we denote by $\bot_{AT}$ the at-constraint $\bot_P \wedge \bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \texttt{false}$, *i.e.* the constraint satisfied by no timing parameter valuation and only by the false action variables.

B. Action-Controllable Parametric Timed Automata

Parametric timed automata (PTA) extend timed automata with parameters within guards and invariants in place of inte-

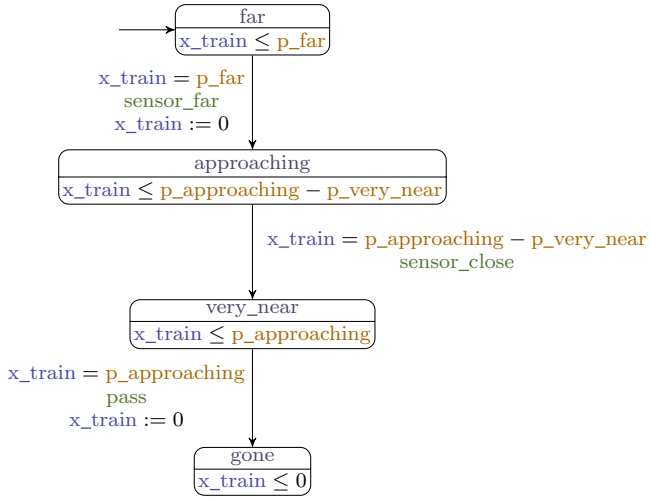


Figure 1: A parametric timed automaton

ger constants [AHV93]. We extend this concept by allowing controllable actions.

Definition 1. *An Action-controllable PTA (APTA) is a tuple* $\mathcal{APTA} = (\Sigma, \mathit{ActVars}, L, l_0, X, P, I, E)$*, where: i)* Σ *is a finite set of non-controllable actions, ii) ActVars is a finite set of action variables (that can be controlled), iii)* L *is a finite set of locations, iv)* $l_0 \in L$ *is the initial location, v)* X *is a set of clocks, vi)* P *is a set of parameters, vii)* I *is the invariant function, assigning to every* $l \in L$ *an invariant* $I(l)$*, viii)* E *is a set of edges* $e = (l, g, a, R, l')$ *where* $l, l' \in L$ *are the source and target locations,* g *is a guard,* $a \in \Sigma \cup \mathit{ActVars}$*, and* $R \subseteq X$ *is a set of clocks to be reset.*

A PTA is an APTA without action variables.

a) Valuation of an APTA: Given an APTA $\mathcal{APTA} = (\Sigma, \mathit{ActVars}, L, l_0, X, P, I, E)$ and an a-valuation ξ, we denote by $\xi(\mathcal{APTA})$ the PTA $\mathcal{PTA} = (\Sigma', L, l_0, X, P, I, E')$, where $\Sigma' = \Sigma \cup \{\alpha \in \mathit{ActVars} \mid \xi(\alpha) = \texttt{true}\}$[1] and $E' = E \setminus \{(l, g, \alpha, R, l') \in E \mid \alpha \in \mathit{ActVars} \wedge \xi(\alpha) = \texttt{false}\}$.

Example: Figure 1 presents a simple PTA modelling a train circulating on a railway with a gate. This model will be enhanced in the case study of Section IV. It uses 3 parameters (p_far, p_approaching and p_very_near) and 1 clock (x_train). Initially, the train is far from the gate and remains far for as long as p_far. Then it triggers sensor_far and starts approaching the gate, resetting the clock. The approach lasts p_approaching but when only p_very_near time units remain, sensor_close is triggered. Finally, the train can pass the gate and be gone.

b) Valuation of a PTA: Given a PTA $\mathcal{PTA}$ and a timing parameter valuation v, we denote by $v(\mathcal{PTA})$ the *timed automaton (TA)* obtained from $\mathcal{PTA}$ by replacing in $\mathcal{PTA}$ each timing parameter p with its valuation $v(p)$.

Given an at-valuation at, let v be the projection of at onto P, and ξ its projection onto *ActVars*. Then given an APTA $\mathcal{APTA}$, we denote by $at(\mathcal{APTA})$ the TA $v(\xi(\mathcal{APTA}))$.

c) Concrete semantics of a TA:

Definition 2 (Semantics of a TA)**.** *Given a PTA* $\mathcal{PTA} = (\Sigma, L, l_0, X, P, I, E)$*, and a timing parameter valuation* v*, the concrete semantics of* $v(\mathcal{PTA})$ *is given by the timed transition system* $(Q, q_0, \to)$*, with*

- $Q = \{(l, w) \in L \times \mathbb{R}_+^H \mid w|v \models I(l)\}$, $q_0 = (l_0, X = 0)$
- $\to$ *consists of the discrete and continuous transition relations:*
  - *discrete transitions:* $(l, w) \xrightarrow{e} (l', w')$*, if* $(l, w), (l', w') \in Q$*, there exists* $e = (l, g, a, R, l') \in E$*,* $w' = [w]_R$*, and* $w|v \models g$*.*
  - *delay transitions:* $(l, w) \xrightarrow{d} (l, w + d)$*, with* $d \in \mathbb{R}_+$*, if* $\forall d' \in [0, d], (l, w + d') \in Q$*.*

Moreover we write $(l, w) \overset{e}{\mapsto} (l', w')$ for a sequence of discrete step and time elapsing where $((l, w), e, (l', w')) \in \mapsto$ if $\exists d, w'' : (l, w) \xrightarrow{e} (l', w'') \xrightarrow{d} (l', w')$.

Given a TA $v(\mathcal{PTA})$ with the concrete semantics $(Q, q_0, \to)$, we refer to the states of Q as the *concrete states* of $v(\mathcal{PTA})$. A concrete run of $v(\mathcal{PTA})$ is an alternating sequence of concrete states of $v(\mathcal{PTA})$ and edges starting from the initial concrete state q_0 of the form $q_0 \overset{e_0}{\mapsto} q_1 \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} q_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \mapsto$. Given a state $q = (l, w)$, we say that q is reachable (or that $v(\mathcal{PTA})$ reaches q) if q belongs to a run of $v(\mathcal{PTA})$. By extension, we say that l is reachable in $v(\mathcal{PTA})$.

d) Symbolic semantics of a PTA: Let us now recall the symbolic semantics of PTA. A symbolic state is a pair (l, C) where $l \in L$ is a location, and C its associated parametric zone. The initial symbolic state of $\mathcal{PTA}$ is $s_0 = (l_0, (X = 0)^{\nearrow} \wedge I(l_0))$.

The symbolic semantics exploits the Succ operation. Given a symbolic state $s = (l, C)$ and an edge $e = (l, g, a, R, l')$, $\mathsf{Succ}(s, e) = (l', C')$, with $C' = \left([(C \wedge g)]_R\right)^{\nearrow} \cap I(l')$. That is, we first intersect the constraint of the source symbolic state with the outgoing guard, then reset the clocks of the transition, then let time elapse, and finally intersect with the invariant of the target location.

A symbolic run of a PTA is an alternating sequence of symbolic states and edges starting from the initial symbolic state, of the form $s_0 \overset{e_0}{\Rightarrow} s_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and s_{i+1} belongs to $\mathsf{Succ}(s_i, e)$. Given a symbolic state s, we say that s is reachable if s belongs to a run of $\mathcal{PTA}$. In the following, we simply refer to symbolic states belonging to a run of $\mathcal{PTA}$ as symbolic states (or, when clear from the context, just as states) of $\mathcal{PTA}$.

Given a concrete (respectively symbolic) run $(l_0, X = 0) \overset{e_0}{\mapsto} (l_1, w_1) \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} (l_m, w_m)$ (respectively $(l_0, C_0) \overset{e_0}{\Rightarrow} (l_1, C_1) \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} (l_m, C_m)$), its corresponding discrete sequence is $l_0 \overset{e_0}{\Rightarrow} l_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} l_m$. Two runs (concrete or symbolic) are said to be equivalent if their corresponding discrete sequences are equal.

[1] Strictly speaking, action variables and actions are of different types; we assume that each action variable valuated with `true` is added to the set of actions.

C. Mixed Transition Systems

Mixed Transition Systems [PR06] (MTS) are essentially Kripke structures with transitions labelled by actions.

Definition 3 (MTS)**.** *Let $\mathcal{PV}$ be a set of propositional variables. A* mixed transition system *(MTS) is a 5-tuple $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$, where: i) $\mathcal{S}$ is a non-empty finite set of states, ii) $s^0 \in \mathcal{S}$ is the initial state, iii) Σ is a non-empty finite set of actions, iv) $\mathcal{T} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation, v) $\mathcal{V}_s : \mathcal{S} \to 2^{\mathcal{PV}}$ is a (state) valuation function.*

We write $s \xrightarrow{a} s'$ if $(s, a, s') \in \mathcal{T}$. Let $\chi \subseteq \Sigma$ be a nonempty set of actions and $\pi = (s_0, a_0, s_1, a_1, \ldots)$ be a finite or infinite sequence of interleaved states and actions. By $|\pi|$ we denote the number of the states of π if π is finite, and ω if π is infinite. A sequence π is a *path* over χ iff $s_i \xrightarrow{a_i} s_{i+1}$, $a_i \in \chi$ for each $i < |\pi|$, and either π is infinite or its final state does not have a χ-successor state in $\mathcal{S}$, *i.e.* $\pi = (s_0, a_0, s_1, a_1, \ldots, s_m)$ for some $m \in \mathbb{N}$ and there is no $s' \in \mathcal{S}$ and $a \in \chi$ such that $s_m \xrightarrow{a} s'$. In general, for all $i \in \mathbb{N}$, we denote by π_i the state of π at rank i ($\pi_i = s_i$ in the sequence above).

The set of all paths over χ in a mixed transition system $\mathcal{M}$ is denoted by $\Pi(\mathcal{M}, \chi)$, whereas the set of all paths $\pi \in \Pi(\mathcal{M}, \chi)$ starting from a given state $s \in \mathcal{S}$ is denoted by $\Pi(\mathcal{M}, \chi, s)$. We omit the symbol $\mathcal{M}$ if it is clear from the context, simply writing $\Pi(\chi)$ and $\Pi(\chi, s)$.

D. Parametric Reachability

In [KMP15] a parametric extension of *Action-Restricted Computation Tree Logic* [PR06], ARCTL, is presented. The language of ARCTL consists of CTL-like branching-time formulae where each path quantifier is subscripted with a set of actions. The subscripts are used in path selection, for example: $E_{\{left,right\}}G(E_{\{forward\}}Fsafe)$ may be read as *"there exists a path over left and right, on which it holds globally that a state satisfying safe is reachable along some path over forward"*. In Parametric ARCTL (denoted by pmARCTL), special variables ranging over sets of actions are allowed in the superscripts. In this paper we deal only with the parametric reachability, *i.e.* with the formulae of type $E_Z Fpv$, where $pv \in \mathcal{PV}$. As we do not permit for nesting of modalities, the superscript is usually omitted and we simply write $EFpv$.

We interpret the formulae with respect to action valuations. Formally, given a MTS $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$, and $pv \in \mathcal{PV}$, we say that $EFpv$ holds in $s \in \mathcal{S}$ under an a-valuation $\xi \in ActVals$ iff there exists a path $\pi \in \Pi(\xi, s)$ such that $pv \in \mathcal{V}_s(\pi_i)$ for some $i \in \mathbb{N}$. This is denoted by $\mathcal{M}, s \models_\xi EFpv$.

E. Action Synthesis

Action synthesis for $\varphi \in$ pmARCTL builds an indicator function $f_\varphi \colon \mathcal{S} \to 2^{ActVals}$ such that for all $s \in \mathcal{S}$ we have: $s \models_\xi \varphi$ iff $\xi \in f_\varphi(s)$. Intuitively, $f_\varphi(s)$ consists of all action valuations under which φ holds in state s. As shown in [KMP15], similarly to CTL, pmARCTL has a fixed-point characterisation, which allows for building efficient procedures for action synthesis, based on Binary Decision Diagrams (BDDs). Such a framework for action synthesis was implemented in the standalone tool SPATULA [Kna]. In what follows, given an MTS $\mathcal{M}$ and $pv \in \mathcal{PV}$, by $\mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ we denote the result of the procedure synthesising all minimal sets of actions under which $EFpv$ holds in the initial state of $\mathcal{M}$ (*e.g.* using SPATULA). Formally: $\mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ is the smallest subset of 2^Σ such that: (1) $\xi \in \mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ implies $\mathcal{M}, s^0 \models_\xi EFpv$, and (2) if $\mathcal{M}, s^0 \models_{\xi'} EFpv$, then there exists $\xi \in \mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ satisfying $\xi \subseteq \xi'$.

III. The Mixed Synthesis Problem: Combining Timing and Action Synthesis

A. Objective

Our main goal is to synthesize parameters seen as both timing constants (à la [AHV93]) and switchable actions that can be enabled or disabled once for all (à la [KMP15]).

Action and time synthesis problem:
INPUT: an APTA $\mathcal{APTA}$ and a set of locations L_{bad}
PROBLEM: Synthesise an AT-constraint AT such that, for all $at \models AT$, no location in L_{bad} is reachable in $at(\mathcal{APTA})$.

As this is a safety problem, it is bad-state driven. Hence, in the following, we refer to locations of L_{bad} as *bad locations*, and to the states the location of which belongs to L_{bad} as *bad states*.

B. General Approach

We follow an approach where we first handle a parametric timed model in which we assume all actions to be enabled; then, from the state space of this model, we synthesise actions together with timing parameter constraints. Enabling all actions in the first phase allows for synthesising all timing parameter constraints for all possible actions, and then combining these constraints with action parameters in the second phase.

The steps of this approach are as follows:

1) Model the system using an APTA.
2) Considering the PTA with all actions enabled, generate a sufficient subpart of the state space reaching bad locations.
3) Using this state space seen as an MTS, synthesize actions ensuring unreachability of the bad states.
4) Process the result to get a linear constraint both on timing parameters and actions.

Each of these steps is detailed in the following subsections.

C. A Labelled Parametric State Space

Starting from an APTA model $\mathcal{APTA}$, we first enable all actions, so as to get the PTA $at_\top(\mathcal{APTA})$. We then generate an MTS augmented with parameter constraints. Each state of this MTS corresponds to a symbolic state of the PTA; each transition of this MTS corresponds to a transition of the state space with the action label. We do not generate the entire state space, but locally stop the exploration whenever a bad state is reached (while still exploring other branches).

Algorithm 1 presents our semi-algorithm genMTS to generate the MTS from the PTA $at_\top(\mathcal{APTA})$. It takes as arguments a PTA and the set L_{bad} of bad locations, and builds

Algorithm 1: genMTS($\mathcal{PTA}, L_{bad}$)

input : PTA $\mathcal{PTA}$ with initial state s_0
input : Set of locations L_{bad}
output: MTS $\mathcal{M}$

1 $S \leftarrow \emptyset$; $S_{queue} \leftarrow \{s_0\}$;
2 **while** $S_{queue} \neq \emptyset$ **do**
3 Pick a state $s = (l, C)$ from S_{queue}
/* If s is bad state */
4 **if** $l \in L_{bad}$ **then**
/* Label this state as bad */
5 $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}\}$
6 **else**
/* Compute successors */
7 **foreach** *state* $s' \in \mathsf{Succ}(s)$ *such that* $s \overset{a}{\Rightarrow} s'$ **do**
/* Enqueue unless met earlier */
8 **if** $s' \notin S_{queue} \cup S$ **then**
9 $S_{queue} \leftarrow S_{queue} \cup \{s'\}$
/* Add transition to MTS */
10 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s, a, s')\}$
11 **return** $\mathcal{M} = (S, \{s_0\}, \Sigma, \mathcal{T}, \mathcal{V}_s)$

(a part of) the symbolic semantics of the PTA. Algorithm genMTS maintains two local variables: the set S of processed states, and the set S_{queue} of states to be processed. Observe that if S_{queue} is implemented using a queue, then genMTS can be seen as a breadth-first search algorithm. Additionally, $\mathcal{T}$ denotes the list of transitions of the resulting MTS (initially empty) and $\mathcal{V}_s$ its labelling function (initially assigning to each state no label).

While the set of states to be processed is not empty (line 2), genMTS selects an unprocessed state s (line 3). If this state is a bad state (line 4), it is labelled as such (line 5); we use notation $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}\}$ to denote that function $\mathcal{V}_s$ is updated so that **bad** is the (unique) label of state s. If this state is a bad state, its successors are not computed, *i.e.* the algorithm stops exploring this branch. Otherwise, the successors of s are computed (line 7) and, for each of them, they are added to the set of states to be processed (line 9), unless they were met earlier. Finally, the corresponding transitions are added to the MTS (line 10).

Eventually genMTS returns a MTS made of the set S of states, the initial state s_0, all actions of the PTA, the transitions of the state space, and the function $\mathcal{V}_s$ that labels the bad states.

Example 1. *Consider the PTA $\mathcal{PTA}$ in Figure 2a. Assume $L_{bad} = \{l_4, l_5\}$. The MTS generated by* genMTS($\mathcal{PTA}, L_{bad}$) *is given in Figure 2b; note that, for better understanding, for a state (l, C), we give the location l in the upper part, and the constraint projected onto P (i.e. $C{\downarrow_P}$) in the lower part; moreover, we add the label (***bad***) to the upper part after the location name. Observe that locations beyond a bad location in Figure 2a (e.g. l_7) are not part of the MTS, as* genMTS *does not explore states beyond bad states.*

D. Synthesising Action and Timing Parameters

Let us now synthesise the actions and timing parameters ensuring the (un)reachability of the bad states. Let us consider the pmARCTL formula $EF\mathbf{bad}$, *i.e.* the formula stating that there exists a path eventually reaching a bad state. From the parameter valuations reaching the bad state, we can easily retrieve the complement set for which the bad state is unreachable. Applying directly action synthesis techniques of [KMP15] to the MTS obtained from Algorithm genMTS would not be satisfactory: this would yield the minimal sets of actions for which bad states are reachable, independently of the timing parameter valuations.

Here, we aim at synthesising an at-constraint on the timing parameters and action parameters guaranteeing the unreachability of L_{bad}. More precisely, for each occurrence of a bad state on a given path, at least one of the actions in each minimal set of actions leading to this state should be disabled, or the timing parameter constraint allowing this reachability should be negated. Let us explain this part in more details in the following.

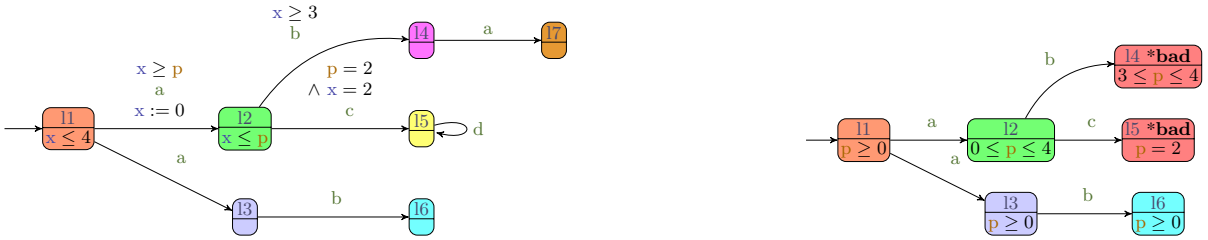
Algorithm 2: synthActTime($\mathcal{M}$)

input : MTS $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$
output: AT-constraint guaranteeing unreachability of L_{bad}

1 $AT \leftarrow \bot_{AT}$; $i \leftarrow 1$
2 **foreach** $s = (l, C) \in \mathcal{S}$ *such that* $\mathbf{bad} \in \mathcal{V}_s(s)$ **do**
/* Generate a dedicated label for s */
3 $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}_i\}$
4 $ActsSets \leftarrow$ synthpmARCTL($\mathcal{M}, EF\mathbf{bad}_i$)
5 $AT \leftarrow AT \vee \left(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets} \left(\bigwedge_{\alpha \in Acts} \alpha\right)\right)$
6 $i \leftarrow i + 1$
7 **return** $\neg AT$

Algorithm 2 presents our procedure to synthesize an AT-constraint guaranteeing the non-reachability of L_{bad}. The algorithm synthActTime($\mathcal{M}$) maintains two variables: the AT-constraint (initially false), and an integer i (just used to generate unique labels). Note that the AT-constraint in fact ensures the *reachability* of at least one of the bad states; it eventually is negated to ensure non-reachability. For each bad state of the MTS (*i.e.* labelled with **bad**), we generate a unique label for this state (line 3). Then, we synthesize the set of minimal sets of actions for this unique label to be reachable, *i.e.* such that this state s is reachable. We then refine the AT-constraint as follows: the constraint is augmented (in the sense of the disjunction) with a constraint ensuring the reachability of the current state s, *i.e.* the timing parameter constraint allowing the reachability of this state, together with at least one of the minimal sets of actions allowing its reachability (line 5). The negation of the AT-constraint is eventually returned (line 7).

Example 2. *Consider again the APTA $\mathcal{APTA}$ in Figure 2a, and assume all actions are controllable, i.e. $ActVars = \{a, b, c, d\}$. The MTS $\mathcal{M}$ generated by* genMTS($at_\top(\mathcal{APTA}), L_{bad}$) *is given in Figure 2b. Let us apply* synthActTime *to $\mathcal{M}$. Initially, AT is $\bot_{AT}$. Let us consider as the first bad state the unique state whose location is l_4 in Figure 2b.* synthActTime *adds to that state a label $\mathbf{bad}_1$. Then, the result of* synthpmARCTL($\mathcal{M}, EF\mathbf{bad}_1$) *is $\{a, b\}$. Hence,* synthActTime *adds to AT the following at-constraint: $3 \leq p \leq 4 \wedge a \wedge b$. Let us consider the*



(a) A sample PTA

(b) The generated MTS

Figure 2: A PTA and its MTS

second bad state, i.e. the unique state whose location is l_5. synthActTime *adds to that state a label* $\mathbf{bad}_2$. *Then, the result of* synthpmARCTL$(\mathcal{M}, EF\mathbf{bad}_2)$ *is* $\{a, c\}$. *Hence,* synthActTime *adds to* AT *the following at-constraint:* $p = 2 \wedge a \wedge c$. *Eventually,* synthActTime *returns the negation of* AT, *i.e.:*

$$(p < 3 \vee p > 4 \vee \neg a \vee \neg b) \wedge (p \neq 2 \vee \neg a \vee \neg c).$$

Sufficient conditions to ensure the satisfaction of this constraint are for example $p = 1$, *or* a *is disabled, or* $p = 2$ *and* c *is disabled.*

Let us now examine a variant of this example where action c *is not controllable. The final result obtained is:*

$$(p < 3 \vee p > 4 \vee \neg a \vee \neg b) \wedge (p \neq 2 \vee \neg a).$$

In this case, $p = 1$, *or* a *is disabled are still sufficient conditions, while* $p = 2$ *is not.*

E. Soundness and Completeness

We now prove that our approach is sound and complete.

Theorem 1 (soundness)**.** *Let* $\mathcal{APTA}$ *be an APTA, and* L_{bad} *be the set of locations to avoid. Assume* genMTS$(at_\top(\mathcal{APTA}), L_{bad})$ *terminates with result* $\mathcal{M}$. *Let* $Res =$ synthActTime$(\mathcal{M})$. *Then, for all* $at \models Res$, *locations* L_{bad} *are unreachable in* $at(\mathcal{APTA})$.

Proof: We first need to recall two lemmas relating symbolic and concrete runs (proved in, *e.g.* [HRSV02], [ACEF09]).

Lemma 1. *Let* $\mathcal{PTA}$ *be a PTA, and* v *be a timing parameter valuation. Let* ρ *be a run of* $\mathcal{PTA}$ *reaching a symbolic state* (l, C). *Then there exists an equivalent run in the TA* $v(\mathcal{PTA})$ *reaching a concrete state* (l, w), *for some* $w \models C$, *iff* $v \models C{\downarrow_P}$.

Lemma 2. *Let* $\mathcal{PTA}$ *be a PTA, and* v *be a timing parameter valuation. Let* ρ *be a run of the TA* $v(\mathcal{PTA})$ *reaching a concrete state* (l, w). *Then there exists an equivalent run in* $\mathcal{PTA}$ *reaching a symbolic state* (l, C) *such that* $v \models C{\downarrow_P}$.

Let $Res =$ synthActTime$(\mathcal{M})$. Let us reason by *reductio ad absurdum*, and assume that a state $s = (l, w)$ with $l \in L_{bad}$ is reachable in $at(\mathcal{APTA})$, for some concrete run. Let ξ be the projection of at on actions. Assume this state is the first bad state along this run (if it is not, then consider the first bad state instead of s). From Lemma 2, we have there exists an equivalent symbolic run in the PTA $\xi(\mathcal{APTA})$ reaching a state (l, C) for some C such that $v \models C{\downarrow_P}$. Since genMTS only stops exploring a branch whenever a symbolic state has no successor or when a bad state is met, then (l, C) was necessarily met by genMTS, and hence is part of $\mathcal{M}$, where it is labelled with $\mathbf{bad}$. Then, in synthActTime$(\mathcal{M})$, the set $ActsSets$ of minimal sets of actions leading to (l, C) is synthesized, and the constraint AT is updated with $\big(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets}(\bigwedge_{\alpha \in Acts} \alpha)\big)$. In the end, synthActTime returns the negation of AT, *i.e.* a conjunction of the constraints of the form $\big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$. This is to say that a valuation satisfying the result of synthActTime either does not satisfy $C{\downarrow_P}$, or at least one action in each set of minimal sets of actions allowing the reachability of (l, C) is disabled. Recall that we assumed $at \models$ synthActTime$(\mathcal{M})$. If the former statement ($at \models \neg C{\downarrow_P}$) is true, then from Lemma 1 no concrete run of $at(\mathcal{APTA})$ reaches a concrete state equivalent to (l, C), which contradicts the hypothesis that (l, w) is reachable in $at(\mathcal{APTA})$. Otherwise, if the latter statement is true, then one action is disabled in any symbolic path that could lead to (l, C), and hence again this contradicts the hypothesis that (l, w) is reachable in $at(\mathcal{APTA})$. ■

Theorem 2 (completeness)**.** *Let* $\mathcal{APTA}$ *be an APTA, and* L_{bad} *be the set of locations to avoid. Assume* genMTS$(at_\top(\mathcal{APTA}), L_{bad})$ *terminates with result* $\mathcal{M}$. *Let* $Res =$ synthActTime$(\mathcal{M})$. *Let* at *be an at-valuation such that locations* L_{bad} *are unreachable in* $at(\mathcal{APTA})$. *Then* $at \models Res$.

Proof: Let at be an at-valuation such that locations L_{bad} are unreachable in $at(\mathcal{APTA})$. Let us reason by *reductio ad absurdum*, and assume that $at \not\models Res$. Recall that synthActTime returns the negation of AT, *i.e.* a conjunction of constraints, each corresponding to the handling of one bad state. Since $at \not\models Res$, then there exists at least one such constraint that is not satisfied by at. Consider one of these constraints not satisfied by at; this constraint is of the form $\big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$, where C is the constraint of a symbolic state (l, C) (with $l \in L_{bad}$) of $\top_A(\mathcal{APTA})$, and $ActsSets$ is the set of minimal sets of actions allowing this set to be reachable. Since we have $at \not\models \big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$ then we have $at \models \big(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets}(\bigwedge_{\alpha \in Acts} \alpha)\big)$. That is, $at \models C{\downarrow_P}$ and there exists a minimal set of actions in $ActsSets$ that are all enabled in at. Consider the symbolic run leading to (l, C) using this minimal set of actions. Since $at \models C{\downarrow_P}$, from

Lemma 1 we have that there exists an equivalent concrete run in $at(\mathcal{APTA})$. In addition, since all actions along this run are enabled in at, then l is reachable in $at(\mathcal{APTA})$, which violates the assumption that all locations of L_{bad} are unreachable in $at(\mathcal{APTA})$. Hence $at \models Res$. ■

F. Discussion: Approximated Synthesis

Most decision problems are undecidable for PTA (see *e.g.* [And15] for a survey), including the emptiness of the valuation set such as a given set of locations is reachable. Hence, exact synthesis is ruled out. Still, subclasses of PTA such that exact synthesis can be achieved exist. First, for acyclic systems (*i.e.* such that no loops exist, or with a limited number of iterations only), most problems are decidable, and synthesis can often be achieved. Applications include most hardware problems, as empirical knowledge show that most hardware verification problems can be carried out for a limited number of hardware clock cycles (typically two); some scheduling problems can also be analysed over a limited number of periods that can be statically computed beforehand. In addition, reducing the number of clocks and/or of parameters can lead to decidability (see [And15]).

However, in the general case, exact synthesis is not necessarily possible, and algorithms may not terminate. Without surprise, genMTS$(\top_A(\mathcal{APTA}), L_{bad})$ may not terminate, since it computes (a subpart of) the state space of a PTA, which is infinite, and for which no finite abstraction can be computed (since the underlying decision problem is undecidable). However, approximations can be used. A possible approximation is to bound the analysis (*e.g.* using a maximum number of explored states, a maximum exploration depth, or a maximum computation time), in which case the state space explored by genMTS is a subset of the actual state space; in that case, the parameter constraint ensuring the reachability of the bad locations is an under-approximation. Using this constraint in synthActTime is not valid: because of the negation (line 5 in Algorithm 2), synthActTime would yield an over-approximation of the good valuations, violating the soundness (Theorem 1). A safe option is to consider that any unexplored state is a bad state, *i.e.* to add to $C{\downarrow_P}$ at line 5 in Algorithm 2 the projection to P of any state (l', C') the successors of which is unexplored. In that case, the result output by synthActTime is a sound under-approximation of the safe action and parameter valuations.

Note however that, in all our experiments, genMTS always terminated without the need to use approximations.

IV. Application and Evaluation

In this section, we apply the approach presented in Section III to a concrete example of an enhanced version of the classical railway gate controller (see, *e.g.* [AHV93]). We start with the detailed description of our model.

A. Running example: An intruder in a railway gate controller

The system itself comprises two components: the train, and the gate controller. A sensor is located far from the gate, which sends a signal to inform the controller that a train is approaching (so that the controller starts lowering the gate). Another sensor is located close to inform the gate that the train is very near, triggering an emergency lowering if the gate is not yet closed.

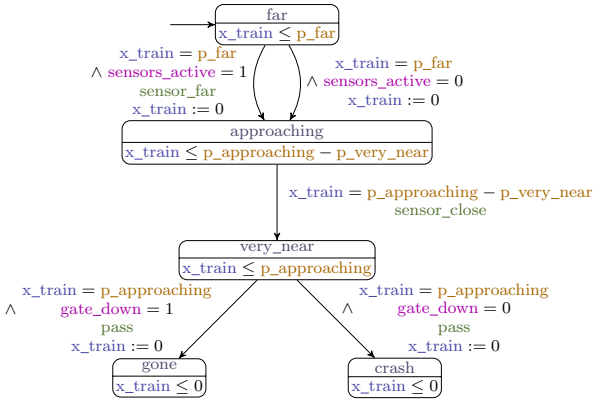


Figure 3: APTA train

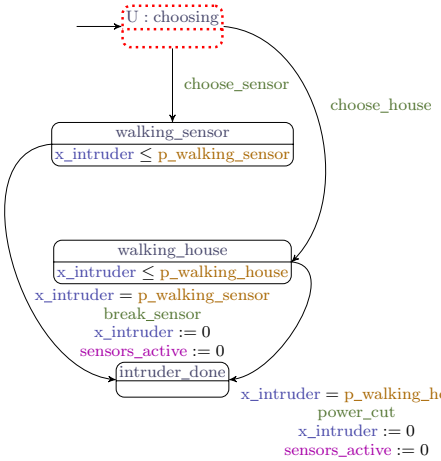


Figure 5: APTA intruder

Let us consider that an intruder can sabotage the system by either disabling the far sensor, thus preventing the detection of an approaching train, or cut the power supply in the gate house, hence preventing the gate to lower. Moreover, the gate has a security mechanism that automatically lowers the gate some time after the power has been shut off, and another security mechanism with an emergency gate lowering when the train is very near.

This system is modelled by three APTAs that model the train (Figure 3), the gate (Figure 4) and the intruder (Figure 5) respectively. They comprise parameters (in brown), clocks (in blue), actions (in green), and discrete values[2] (in purple). Each location has a name indicated in the upper part of the node, and its associated invariant is in the lower part.

The parameters allow for analysing a general model, where the values are not yet known at the design time:

- train parameters:

[2]Though not part of the (A)PTA model, discrete values are integer-valued variables often supported by tools, and that are syntactic sugar for locations.

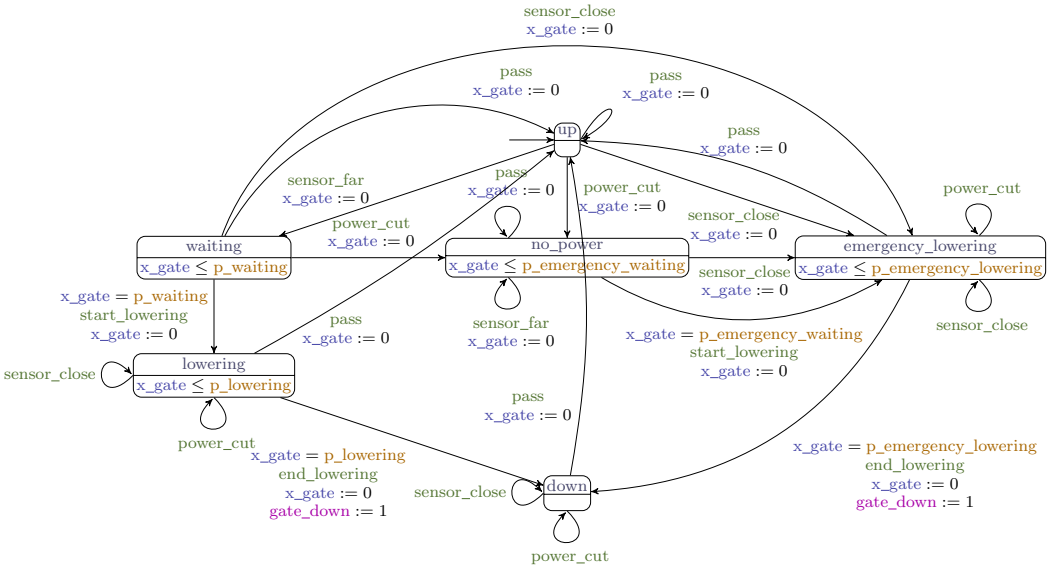


Figure 4: APTA gate

- p_far: time taken between being far and approaching;
- p_approaching: time between the approach stage and passing the gate;
- p_very_near: time between being very near and passing the gate.

- gate parameters:
  - p_waiting: time between sensor activation and beginning of gate lowering;
  - p_emergency_waiting: time between power shutdown and beginning of gate lowering;
  - p_lowering: time to lower the gate;
  - p_emergency_lowering: time to lower the gate in emergency mode.
- intruder parameters:
  - p_walking_sensor: time to walk and disable the sensor;
  - p_walking_house: time to walk and cut the power.

Clocks are associated with the train (x_train), the gate (x_gate) and the intruder (x_intruder).

Let us now detail the train APTA in Figure 3. Initially, the train is far, and remains far until it has travelled for p_far time. At that date, it becomes approaching. Note that there are two similar transitions from location far to location approaching. One considers the sensor is active, and thus triggers the sensor_far action, while the other caters for the inactive sensor case (disabled by the intruder). In both cases, the train clock is reset to count time elapsed from then on. When the train becomes very_near, the APTA changes location again, by triggering sensor_close. Note that here there is only one transition as the intruder cannot tamper with the close sensor. Then the last part of the train approach takes place, and when it reaches the gate, it results in two different states according to the status of the gate. If gate_down is false, there is a crash, otherwise the train resumes its journey. Both end states block time to avoid unnecessary state space explosion during the analysis.

The intruder (Figure 5) starts in an urgent location where (s)he has to choose immediately between the two sabotage possibilities. Although the actions are different (break_sensor and power_cut), they both deactivate the sensors as the communication between sensors and gate is broken.

The gate APTA is presented in Figure 4. Initially the barrier is up, and can change on reception of a sensor_far or a sensor_close signal or detecting a power_cut, leading to locations waiting, emergency_lowering or no_power, respectively. In all cases, the clock starts ticking. After some time, the gate starts lowering until it is down. The power_cut can also happen anytime, leading to the no_power location. After a power shutdown, the emergency procedure states that if the power is not back an emergency_lowering takes place. It is also the case if the sensor_close is triggered while the train should still be approaching. Finally, when the train has passed the crossing, the barrier gets back up.

In the initial configuration, the train is far, the intruder is ready for choosing and the gate is up. The discrete variables are also initialised: gate_down = 0, sensors_active = 1.

B. Tool Support

Our toolchain, depicted in Figure 6, consists of three tools coordinated by the interface IF.

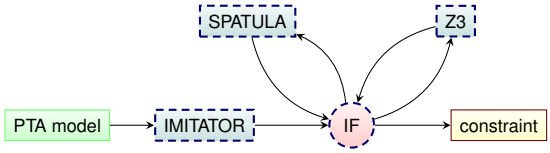


Figure 6: Tool support

The synthesis process starts with the generation of the state space (Section III-C) by using IMITATOR [AFKS12],

a tool for parameter synthesis for real-time systems modelled by PTA. The version used is 2.7.3 (build 1338). IF accepts the output of IMITATOR and prepares a set of programs for SPATULA [KMP15], a tool implementing the action synthesis for pmARCTL. In order to transform the timed state space graph into a mixed transition system, IF performs constraint simplification and analysis, employing the Z3 SMT-solver [dMB08].

C. Experimental Results and Interpretation

We synthesise constraints under which the location crash is avoidable, despite of the presence of the intruder, depending on the extent of the capabilities of the latter, *i.e.* the enabled actions. For practical purposes, our toolchain accepts the set of *controllable actions*, denoted by $Cntr$. Intuitively, these are the only actions that are allowed to be switched on and off in the considered model. As the goal of analysis is the safety from the activities of the malicious intruder, we assume that its actions constitute the set of controllable transitions, *i.e.* $Cntr = \{\text{choose\_house}, \text{choose\_sensor}, \text{break\_sensor}, \text{power\_cut}\}$.

Firstly, the input model is handled by IMITATOR to produce the MTS according to Algorithm 1. In repeated experiments, this step took approximately $2s$, on average. The MTS is then additionally processed by IF with the help of the Z3 solver. Finally, the program IF follows Algorithm 2 to build the set of at-constraints under which a state labeled with crash is reachable. To this end, IF prepares a set of programs for SPATULA, which corresponds to calling the external procedure synthpmARCTL in Algorithm 2. The second part of the process took approximately $1s$, on average.

Timing parameter constraints	Actions
$K_1 = \{k_1, k_2, k_3, k_4, k_5\}$	$Acts_1 = \{\text{choose\_house}, \text{power\_cut}\}$
$K_2 = \{k_6, k_7\}$	$Acts_2 = \{\text{choose\_sensor}\}$
$K_3 = \{k_8, k_9\}$	$Acts_3 = \{\text{choose\_house}\}$
$K_4 = \{k_{10}, k_{11}, k_{12}, k_{13}, k_{14}\}$	$Acts_4 = \{\text{break\_sensor}, \text{choose\_sensor}\}$

Table I: Constraints for reachability of the crash location

$$
\begin{aligned}
k_1 = {} & \text{p\_walking\_house} \geq \text{p\_far} \wedge \text{p\_walking\_sensor} > 0 \\
& \wedge \text{p\_lowering} \geq \text{p\_emergency\_lowering} \wedge \text{p\_very\_near} > 0 \\
& \wedge \text{p\_emergency\_lowering} \geq \text{p\_very\_near} \wedge \text{p\_far} > 0 \\
& \wedge \text{p\_very\_near} + \text{p\_emergency\_waiting} \\
& + \text{p\_walking\_house} \geq \text{p\_far} + \text{p\_approaching} \\
& \wedge \text{p\_far} + \text{p\_approaching} \geq \text{p\_walking\_house} + \text{p\_very\_near} \\
& \wedge \text{p\_far} + \text{p\_waiting} \geq \text{p\_walking\_house}
\end{aligned}
$$

Figure 7: Exemplary parameter constraint k_1

In Table I we collect the result of the synthesis for the reachability of the crash location. The left column of the table contains sets of indices of constraints on time parameters. Due to the space limits we present only exemplary constraint in Figure 7. The right column of the table contains sets of actions. For a given row $1 \leq i \leq 4$, a state labeled with crash can be reached under a given at-valuation at if it satisfies one of constraints from K_i and enables all the actions from $Acts_i$.

The information presented in Table I is exhaustive, *i.e.* crash is reachable iff at satisfies:

$$ReachCrash := \bigvee_{i=1}^{4} \Big(\bigvee_{k \in K_i} k \wedge \bigwedge_{a \in Acts_i} a \Big). \qquad (\clubsuit)$$

Hence the system is safe from the malicious attempts of the intruder iff the at-constraint $AvoidCrash := \neg ReachCrash$ is satisfied. Let $1 \leq i \leq 4$. With a slight notational abuse let us denote $\overline{K_i} := \bigwedge_{k \in K_i} \neg k$ and $\overline{Acts_i} := \neg \bigwedge_{a \in Acts_i} a$. Observe that $\overline{K_i}$ represents the set of parameter valuations that invalidate all of the constraints of K_i. We interpret $\overline{Acts_i}$ as all the subsets of $Cntr$ that do not subsume $Acts_i$. We can now write:

$$AvoidCrash := \bigwedge_{i=1}^{4} \big(\overline{K_i} \vee \overline{Acts_i} \big). \qquad (\spadesuit)$$

For the sake of presentation and readability, we specialize the considered problem by assuming from now on the following values of the train and gate parameters: p_far $= 3$, p_approaching $= 2$, p_very_near $= 1$, p_waiting $= 1$, p_emergency_waiting $= 1$, p_lowering $= 2$, and p_emergency_lowering $= 1$. For each constraint k we denote by $\ddot{k}$ the result of substituting the above parameters with the appropriate values. Then, the constraints $\ddot{k}_1, \ldots, \ddot{k}_{14}$ can be presented as shown in Figure 8.

$$
\begin{aligned}
\ddot{k}_1 &= \text{p\_walking\_sensor} > 0 \wedge 4 \geq \text{p\_walking\_house} \geq 3, \\
\ddot{k}_2 &= \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} = 3, \\
\ddot{k}_3 &= \text{p\_walking\_sensor} > 0 \wedge 5 \geq \text{p\_walking\_house} \geq 4, \\
\ddot{k}_4 &= \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} = 4, \\
\ddot{k}_6 &= \text{p\_walking\_house} > 0 \wedge \text{p\_walking\_sensor} \geq 5, \\
\ddot{k}_8 &= \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} \geq 5, \\
\ddot{k}_{10} &= \text{p\_walking\_house} > 0 \wedge 4 \geq \text{p\_walking\_sensor} \geq 3, \\
\ddot{k}_{12} &= \text{p\_walking\_house} > 0 \wedge 5 \geq \text{p\_walking\_sensor} \geq 4, \\
\ddot{k}_{14} &= \text{p\_walking\_house} > 0 \wedge \text{p\_walking\_sensor} = 4, \\
\ddot{k}_5 &= \ddot{k}_3, \ddot{k}_7 = \ddot{k}_6, \ddot{k}_9 = \ddot{k}_8, \ddot{k}_{11} = \ddot{k}_{10}, \ddot{k}_{13} = \ddot{k}_{12},
\end{aligned}
$$

Figure 8: Parameter constraints under partial substitution

In a natural way we define $\overline{\ddot{K}_i} := \bigwedge_{k \in K_i} \neg \ddot{k}$, for all $1 \leq i \leq 4$ (see Figure 9). Let us show how we can employ these constraints to analyse safety of the system under selected capabilities of the intruder. Assume that the intruder can gain an access to the gate house and cut the power supply, but (s)he is not aware of the sensor, or the sensor cannot be disabled. This corresponds to enabling $Acts_1 = \{\text{choose\_house}, \text{power\_cut}\}$ in the model. As this set subsumes $Acts_3$ (see Table I), by Condition ♠ we have $AvoidCrash = \overline{\ddot{K}_1} \wedge \overline{\ddot{K}_3} = \text{p\_walking\_house} \leq 3$.

D. Performance Evaluation

To the best knowledge of the authors, there is no other tool allowing for mixed action-parameter synthesis for APTA.

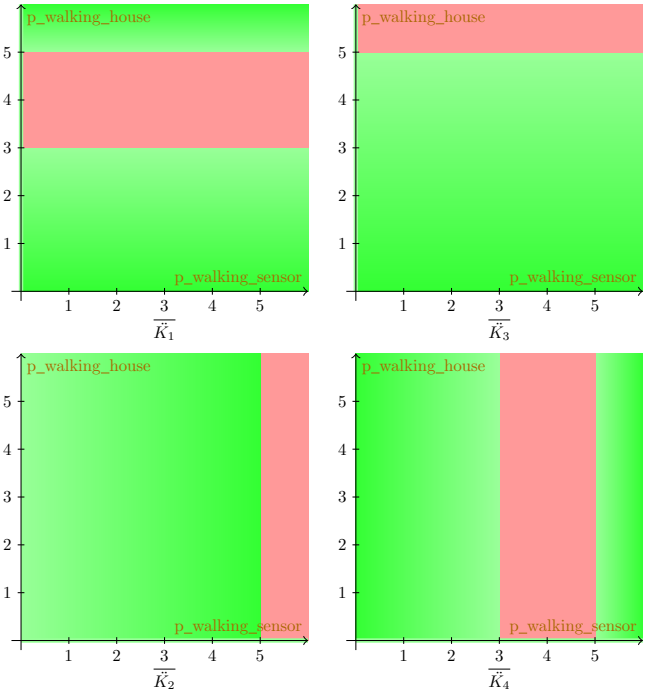


Figure 9: Safety regions

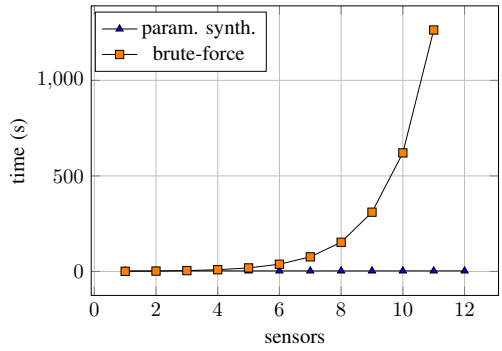


Figure 10: Parametric synthesis vs brute-force

Therefore, following the methodology presented in [KMP15], in Figure 10 we compare the efficiency of our approach to the brute-force solution, where IMITATOR is iteratively called on all the non-empty subsets of the controllable actions. We scale the benchmark with respect to the number of actions that allow to choose the sensor. The advantage of our approach is clearly visible and follows from the fact that SPATULA is tailored towards handling a large number of action variables. It should be noted, however, that our earlier experience with synthesis based on Binary Decision Diagrams (see [KMP15], [JKPL12]) allows us to hypothesise that the practical complexity of the solution presented in this paper is also exponential in the number of actions, albeit with a much smaller steepness than in the case of brute-force.

All the experiments have been performed on an Intel i5 quad-core 2.6 GHz machine with 4 GiB RAM, running Linux 3.13 operating system.

V. Conclusion and Perspectives

We presented an original framework to synthesise two kinds of parameters, *i.e.* action parameters and timing parameters, in a unified manner. The resulting constraint aims at helping designers to tune their system depending on cost or availability constraints, and at providing designers with tools supporting their decision making. We exemplified our approach on a proof-of-concept case study.

So far, we have considered only reachability-like properties. Although some properties go beyond the class of reachability properties, a quite large number of properties (including time properties such as deadlines) fall into the class of properties that can be encoded into reachability testing (see [ABBL98] for a study of its expressive power). Our longer-term goal is of course to generalize our results to deal with more complex, ideally full pmARCTL.

References

[ABBL98] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. The power of reachability testing for timed automata. In *FSTTCS*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 1998.

[Abd12] Parosh Aziz Abdulla. Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118, 2012.

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[ALR15] Étienne André, Didier Lime, and Olivier H. Roux. Integer-complete synthesis for bounded parametric timed automata. In *RP*, volume 9328 of *Lecture Notes in Computer Science*, pages 7–19. Springer, 2015.

[And15] Étienne André. What's decidable about parametric timed automata? In *FTSCS*, volume 596 of *Communications in Computer and Information Science*, pages 1–17. Springer, 2015.

[DKRT97] Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 1997.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[JKPL12] Andrew V. Jones, Michał Knapik, Wojciech Penczek, and Alessio Lomuscio. Group synthesis for parametric temporal-epistemic logic. In *AAMAS*. IFAAMAS, 2012.

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

[KMP15] Michał Knapik, Artur Męski, and Wojciech Penczek. Action synthesis for branching time logic: Theory and applications. *ACM Trans. on Embedded Comput. Syst.*, 14(4), 2015.

[Kna] Michał Knapik. michalknapik.github.io/spatula.

[KP12] Michał Knapik and Wojciech Penczek. Bounded model checking for parametric timed automata. *Transactions on Petri Nets and Other Models of Concurrency*, 5:141–159, 2012.

[PR06] C. Pecheur and F. Raimondi. Symbolic model checking of logics with actions. In *MoChArt*, volume 4428 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2006.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

Parametric Timed Broadcast Protocols

Étienne André
Université Paris 13, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
https://orcid.org/0000-0001-8473-9555

Benoit Delahaye
Université de Nantes, LS2N UMR CNRS 6004, Nantes, France
benoit.delahaye@univ-nantes.fr

Paulin Fournier
Université de Nantes, LS2N UMR CNRS 6004, Nantes, France
paulin.fournier@univ-nantes.fr

Didier Lime
École Centrale de Nantes, LS2N UMR CNRS 6004, Nantes, France
didier.lime@ec-nantes.fr

Abstract

In this paper we consider state reachability in networks composed of many identical processes running a parametric timed broadcast protocol (PTBP). PTBP are a new model which extends both broadcast protocols and parametric timed automata. This work is, up to our knowledge, the first to consider the combination of both a parametric network size and timing parameters in clock guard constraints. Since the communication topology is of utmost importance in broadcast protocols, we investigate reachability problems in both clique semantics where every message reach every processes, and in reconfigurable semantics where the set of receivers is chosen non-deterministically. In addition, we investigate the decidability status depending on whether the timing parameters in guards appear only as upper bounds in guards (U-PTBP), as lower bounds (L-PTBP) or when the set of parameters is partitioned in lower-bound and upper-bound parameters (L/U-PTBP).

2012 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Parameterized systems, parametric timed model checking

Digital Object Identifier 10.4230/LIPIcs...

Funding This work is partially supported by the ANR national research program "PACS" (ANR-14-CE28-0002).

Acknowledgements The authors warmly thank Nathalie Bertrand for fruitful discussions on the topic of this paper.

1 Introduction

The application of model-checking to real-life complex systems faces several problems, and for many of them the use of parameters, i. e., symbolic constants representing an unknown quantity can be part of the solution. First, for big systems, the so-called *state-space explosion* limits the practical applicability of model-checking. Such big systems however are in general specified as the composition of smaller systems. A particularly interesting setting is the one in which all the components are identical, such as in many communication protocols. The number of involved components can then be abstracted away as a *parameter*, with the hope

of both overcoming the state-space explosion, and obtaining more useful answers from the model-checking process, such as "for which sizes of the system does some property hold?". Second, the earlier in the development phase verification can be applied, the less costly will be fixing the problems found, but also the less information we have on the final system, in particular on many timing features, such as transmission times, watchdogs, etc. Parameters can also be useful here by abstracting away the precise values of some yet unknown features, and at the same time allowing their dimensioning.

In this paper, we propose to combine two different types of parameters, namely the *number of identical processes* and the *timing features*, and study the decidability of classic parametric decision problems in the resulting formalism. Both types of parameters, when introduced separately in timed automata-based formalisms, result in hard problems undecidable even in restricted settings.

Timed automata [5] extend finite-state automata with clocks, i. e., real-valued variables that can be compared to constants in guards, and reset along transitions. Parametric timed automata (PTA) [6] allow to replace constants with unknown parameters in timing constraints. The most basic verification question, "does there exist a value for the parameters such that some location is reachable" is undecidable with as few as 1 integer- or rational-valued parameter [21, 10], or when only 1 clock is compared to a unique parameter [21] (with additional clocks). The main syntactic subclass of PTA for which decidability is obtained is L/U-PTA [18], in which the parameters set is partitioned into lower-bound parameters (i. e., parameters always compared as a lower bound in a clock guard) and upper-bound parameters (always as upper bounds). L/U-PTA have been shown [18] to be expressive enough to model classical examples from the litterature, such as root contention or Fischer's mutual exclusion algorithm for instance.

Broadcast protocol networks [13, 15, 16, 14], allow treating the size of a network as an unknown parameter. Here also the most simple basic verification question "does there exist a value for the parameter such that some location is reachable by a process" is undecidable when considering arbitrary communication topologies [14]. However one can regain decidability by considering different communication topology settings. One option is to limit the topologies to cliques (every process receives every messages) [15, 16, 14]. Another is to consider reconfigurable broadcasts in which the set of receivers is chosen non-deterministically at each step [13]. A timed version of this broadcast protocol was studied in [2]. In the clique topology for this network, the reachability problem is decidable only when there is a single clock per process.

Contributions

In this work, we provide one more level of abstraction to the formalisms of the literature by proposing *parametric timed broadcast protocols* (PTBP), i. e., a new formalism made of an arbitrary number of identical timed processes in which timing parameters can be used. Considering those two kinds of parameters could be really useful, for example when designing and verifying communication protocols. Indeed, those protocols are required to work independently of the number of participants (hence the parametric size of networks) and the time constraints in each process are of paramount importance and thus could be tweaked in early development thanks to timing parameters. This work is, up to our knowledge, the first to consider the combination of both a parametric network size and timing parameters in clock guard constraints. We consider the following problems: does there exist a number of processes for which the set of timing parameter valuations allowing to reach a given location for one run ("EF"), or for all runs ("AF") is empty (or universal)? This gives rise to

4 problems: EF-emptiness, EF-universality, AF-emptiness and AF-universality. As PTBP can be seen as an extension of both broadcast protocols and parametric timed automata, undecidability follows immediately from the existing undecidability results known for these two formalisms. However, combining decidable subclasses of both formalisms is challenging, and does not necessarily make the EF and AF problems decidable for PTBP.

The communication topology is of utmost importance in broadcast protocols, and we therefore investigate reachability problems depending on the broadcast semantics. In the reconfigurable semantics (where the set of receivers is chosen non-deterministically), AF-emptiness and AF-universality are decidable for 1-clock PTBP, and undecidable from 3 clocks even for L/U-PTBP with the same parameters partitioning as in L/U-PTA (the 2-clock case is equivalent to a well-known open problem for PTA). The AF results may not seem surprising, as they resemble equivalent results for PTA. However, EF-emptiness and EF-universality becomes undecidable even for 1-clock PTBP: this result comes in contrast with both non-parametric timed broadcast protocols and PTA for which the 1-clock case is decidable.

In the clique semantics (where every message reaches every process), we show that AF problems are undecidable even without any clock. Then, as it is known that 2 clocks (and no parameter) yield undecidability, we study EF problems over 1 clock. We investigate the decidability status depending on whether the timing parameters in guards appear only as upper bounds in guards (U-PTBP), as lower bounds (L-PTBP) or when the set of parameters is partitioned in lower-bound and upper-bound parameters (L/U-PTBP). We show that L/U-PTBP become decidable for EF-emptiness (but not universality) when the parameter domain is bounded. For EF-universality, decidability is obtained only for L-PTBP and U-PTBP for a parameter domain bounded with closed bounds. Our contributions are summarized in Table 1 (page 13).

Related work

To the best of our knowledge, combining two types of parameters (i. e., discrete and continuous) was very little studied—with a few exceptions. In [11], an attempt is made to mix discrete and continuous timing parameters (in an even non-linear fashion). However, the approach is fully *ad-hoc* and addresses an extension of PTA, for which problems are already undecidable. In [20, 12], security protocols are studied with unknown timing constants, and an unbounded number of participants. However, the focus is not on decidability, and the general setting is undecidable. In [7], action parameters (that can be seen as Booleans) and continuous timing parameters are combined (only linearly though) in an extension of PTA; the mere emptiness of the action and timing parameters sets for which a location is reachable is undecidable. In contrast, we exhibit in this work some decidable cases.

2 Definitions

We denote by $\mathbb{N}$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ the sets of all natural, non-negative rational, and non-negative real numbers respectively.$[a, b]$ denotes the interval containing all rational numbers x such that $x \leq b$ and $x \geq a$. As usual, we write $(a, b]$ to exclude a from this set and $[a, b)$ to exclude b (in which case we allow $b = +\infty$).We denote by $\mathbb{I}_{\mathbb{Q}_+}$ the set of all rational intervals.

Given a set E, and an integer $n \in \mathbb{N}$ we denote $\mathbb{V}_n(E)$ the set of all vectors composed by n elements of E. We denote $\mathbb{V}(E)$ the set of all vectors i. e., $\mathbb{V}(E) = \cup_{n\in\mathbb{N}}\mathbb{V}_n(E)$.

Given a set of clocks $\mathbb{X}$, a valuation of $\mathbb{X}$ is a function of $\mathbb{X} \to \mathbb{R}_+$. We denote by $\mathcal{V}(\mathbb{X})$ the set of all valuations of $\mathbb{X}$ or just $\mathcal{V}$ when $\mathbb{X}$ is clear from the context. The valuation assigning 0 to all clock is written $\vec{0}$. Given a valuation $v \in \mathcal{V}$ and a real number t we denote by $v + t$

the valuation v' such that for all $x \in \mathbb{X}$, $v'(x) = v(x) + t$, and $v - t$ (if it exists) the valuation such that $(v - t) + t = v$. Given a set of clocks $\mathbb{X}$ and a set of parameters $\mathbb{P}$ we write $\mathcal{G}(\mathbb{X}, \mathbb{P})$ for the set of all sets of constraints of the form $x \bowtie a$ with $x \in \mathbb{X}$, $\bowtie \in \{<, \leq, =, \geq, >\}$ and $a \in \mathbb{Q}_+ \cup \mathbb{P}$.

We denote by Updates($\mathbb{X}$) the set of updates of the clocks, where an update is a function $up : \mathcal{V} \to \mathcal{V}$ such that for all $x \in \mathbb{X}$, either $up(v)(x) = v(x)$ or $up(v)(x) = 0$. When convenient we represent the update function with the set $\{x_1, \ldots, x_k\}$ representing that clocks x_1 to x_k are reset to 0 while other clocks (here x_i with $i > k$) are left unchanged.

Given a clock valuation $v \in \mathbb{X} \to \mathbb{R}_+$ and a rational valuation of the variables $p : \mathbb{P} \to \mathbb{Q}_+$ we say that the valuation v satisfies a guard $g \in \mathcal{G}(\mathbb{X}, \mathbb{P})$, written $v \models_p g$ if for all $x \bowtie a \in g$ either $a \in \mathbb{Q}_+$ and $v(x) \bowtie a$ or $a \in \mathbb{P}$ and $v(x) \bowtie p(a)$.

We now introduce parametric timed broadcast protocols (PTBP), which are timed broadcast protocols [1] extended with timing parameters in clock guards. Equivalently, PTBP can be seen as a PTA [6] augmented with communication features.

▶ **Definition 1** (Parameterized timed broadcast protocol)**.** A *Parameterized timed broadcast protocol* (PTBP) is a tuple $\mathcal{N} = (Q, \mathbb{X}, \Sigma, \mathbb{P}, q_0, \Delta)$ where:

- Q is a finite set of states;
- $\mathbb{X}$ is a finite set of clocks;
- Σ is the finite communication alphabet;
- $\mathbb{P}$ is a finite set of timing parameters;
- $q_0 \in Q$ is the initial state; and
- $\Delta \subseteq Q \times \mathcal{G}(\mathbb{X}, \mathbb{P}) \times Act \times \text{Updates}(\mathbb{X}) \times Q$ is the edge relation, where Act is the set of actions composed of:
  - internal actions: ϵ;
  - broadcasts of a message $m \in \Sigma$: $!!m$; and
  - reception of a message $m \in \Sigma$: $??m$.

A PTBP is a U-PTBP, L-PTBP, or L/U-PTBP if all timing parameters appear only as upper bounds in guards (i. e., of the form $x < \lambda$ or $x \leq \lambda$), only as lower bounds (i. e., of the form $x > \lambda$ or $x \geq \lambda$), or if the set of parameters $\mathbb{P}$ is partitioned into lower-bound and upper-bound parameters, respectively.

A *bounded* PTBP is a pair $(\mathcal{N}, bounds)$ where $\mathcal{N}$ is a PTBP and $bounds : \mathbb{P} \to \mathcal{I}_{\mathbb{Q}_+}$ are bounds on the parameters that assign to each parameter λ an interval $[\inf, \sup]$, $(\inf, \sup]$, $[\inf, \sup)$, or $(\inf, \sup)$, with $\inf, \sup \in \mathbb{N}$. We use $\inf(\lambda, bounds)$ and $\sup(\lambda, bounds)$ to denote the infimum and the supremum of λ, respectively. A bounded PTBP is a *closed PTBP* if, for each parameter λ, its ranging interval $bounds(\lambda)$ is of the form $[\inf, \sup]$. Otherwise it is open bounded. Abusing notation we say that a parameter valuation p belong to a bound $bounds$, written $p \in bounds$, if for all parameters λ, $p(\lambda) \in bounds(\lambda)$.

▶ **Example 2.** An example of a PTBP is given in Fig. 1. This PTBP is composed of an initial state q_0, two states f and c representing a factory and a client, three counting states 1, 2 and 3 and a goal state g. The set of clocks is the singleton $\{x\}$ and the communication alphabet is composed of two messages p and f. There are two timing parameters pt and tl representing respectively the production time and the time limit. Notice that this PTBP is in fact an L/U-PTBP since the parameter pt appears only in guards as a lower bound and tl only as an upper bound.

We now define the semantics of parameterized networks of PTBP. This semantics is illustrated in Example 3 after the formal definition.

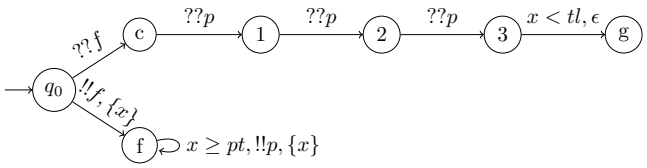


Figure 1 Example of a (L/U-)PTBP

A network is composed of a multitude of processes all running the same protocol $\mathcal{N}$.

Formally, a configuration γ of a network running a parametric timed broadcast protocol $\mathcal{N} = (Q, \mathbb{X}, \Sigma, \mathbb{P}, q_0, \Delta)$ is a vector $\gamma \in \mathbb{V}(Q \times \mathcal{V})$. Intuitively, a configuration γ with $\gamma[i] = (q, v)$ means that the process i is in state q and with clock valuation v.

Given a configuration γ with N processes and a process i, we denote $state(\gamma[i])$ the state and $val(\gamma[i])$ the valuation such that $\gamma[i] = (state(\gamma[i]), val(\gamma[i]))$. Abusing notation we extend $state$ to the whole configuration i. e., $state(\gamma) = \cup_{i \in \{1,..,N\}} state(\gamma[i])$.

Note that the representation of configuration as vectors is only for practical reasons, the processes are identical and do not have ids.

We say that a configuration γ is initial if all processes are in the initial state and their clocks are all set to 0 i. e., for all i, $\gamma[i] = (q_0, \vec{0})$.

Given a timing parameter valuation, the transition relation on configurations is intuitively defined as follows: First a delay is chosen and all the clocks in the network are increased by this delay. Then one of the processes performs a possible action i. e., an action for which the guard is satisfied given its clock valuation and the valuation of the timing parameter. Two cases follow. Either the action is internal and only this process moves and updates its clocks accordingly, or the action is a broadcast and a set of receivers is chosen. It this latter case, the sender moves and updates its clocks and all the chosen receivers also move and update their clocks accordingly.

More formally, given a timing parameter valuation p, a configuration $\gamma \in \mathbb{V}_N(Q \times \mathcal{V})$ there are transitions for all $t \in \mathbb{R}_+$, $i \in \{1, \ldots, N\}$, $\delta = (q_1, g, a, up, q_2) \in \Delta$, and $R \subseteq \{1, .., N\}$ such that:

elapse of time there is a valuation $\gamma_t \in \mathbb{V}_N(Q \times \mathcal{V})$ such that $\forall j \in \{1, \ldots, N\}$, $\gamma_t[j] = (q, v + t)$ where $(q, v) = \gamma[j]$, and

execution of the action the following conditions are satisfied:

- **the action is enabled** $state(\gamma_t[i]) = q_1$ and $val(\gamma_t[i]) \models_p g$, and
- **execution of the action** the transition leads to a configuration γ' such that
  - the active process performed the action: $\gamma'[i] = (q_2, up(val(\gamma_t[i])))$,
  - unconcerned processes are unaffected: $\forall j \in \{1, \ldots, N\} \setminus (R \cup \{i\})$, $\gamma'[j] = \gamma_t[j]$, and
  - either
    - a is an internal action ($a = \epsilon$) and the receiving processes are unaffected: $\forall j \in R \setminus \{i\}$, $\gamma'[j] = \gamma_t[j]$; or
    - $a = !!m$ and $\forall j \in R \setminus \{i\}$, if there exists an edge $(state(\gamma_t[j]), g', ??m, up', q')$ such that $val(\gamma_t[j]) \models_p g'$, then the process receives the message and $\gamma'[j] = (q', up'(val(\gamma_t[j])))$. Otherwise the process is unaffected and $\gamma'[j] = \gamma_t[j]$.

When such a transition exists, it is written $\gamma \xrightarrow{t,i,\delta,R}_p \gamma'$ or simply $\gamma \rightarrow_p \gamma'$.

Notice that we consider non blocking broadcast i. e., if a process is in the receiver set but has no available reception edge, the process is unaffected and the network behaves as if this process was not in the receiver set.

An execution ρ is a sequence of transitions starting in an initial configuration γ_0, $\rho = \gamma_0 \to_p \gamma_1 \to_p \cdots$. An execution is maximal if it is infinite or if it ends in a configuration from which there is no possible transition.

Notice that once an initial configuration is fixed, the number of processes does not change along an execution. However the semantics is infinite for several reasons: first there is an infinite number of initial configurations (i. e., of network sizes); second, there is also an infinite number of possible parameter valuations; third, given a network size and parameter valuation, clock valuations assign real values to clocks and are thus uncountable.

Given PTBP $\mathcal{N}$, a network size N and a timing parameter valuation p, we denote by $\mathcal{E}(\mathcal{N}, N, p)$ the set of all maximal executions for the valuation p with N processes.

We say that a maximal execution $\rho = \gamma_0 \to_p \gamma_1 \to_p \cdots$ reaches a state q, written $\rho \models \Diamond q$, if there exists and index n such that $q \in state(\gamma_n)$.

▶ **Example 3.** We give an example of a possible execution for a network composed of 4 processes running the protocol given in Example 2. In this example $tl = 9$ and $pt = 3$. The edge used during a transition is here only represented by the associated action for readability.

$$\begin{pmatrix} q_0, 0 \\ q_0, 0 \\ q_0, 0 \\ q_0, 0 \\ q_0, 0 \end{pmatrix} \xrightarrow{0.1,1,f,\emptyset} \begin{pmatrix} f, 0 \\ q_0, 0.1 \\ q_0, 0.1 \\ q_0, 0.1 \\ q_0, 0.1 \end{pmatrix} \xrightarrow{4.1,2,f,\{3,5\}} \begin{pmatrix} f, 4.1 \\ f, 0 \\ c, 4.2 \\ q_0, 4.2 \\ c, 4.2 \end{pmatrix} \xrightarrow{1.3,1,p,\{5\}} \begin{pmatrix} f, 0 \\ f, 1.3 \\ c, 5.5 \\ q_0, 5.5 \\ 1, 5.5 \end{pmatrix} \xrightarrow{1.8,2,p,\{1,3,4,5\}}$$

$$\xrightarrow{1.8,2,p,\{1,3,4,5\}} \begin{pmatrix} f, 1.8 \\ f, 0 \\ 1, 7.3 \\ q_0, 7.3 \\ 2, 7.3 \end{pmatrix} \xrightarrow{1.2,1,p,\{5\}} \begin{pmatrix} f, 0 \\ f, 1.2 \\ 1, 8.5 \\ q_0, 8.5 \\ 3, 8.5 \end{pmatrix} \xrightarrow{0,5,\epsilon,\emptyset} \begin{pmatrix} f, 0 \\ f, 1.2 \\ 1, 8.5 \\ q_0, 8.5 \\ g, 8.5 \end{pmatrix}$$

▶ Remark. Notice that even if the notations are slightly different, PTBP networks fully extend both PTA [6] and timed broadcast protocols [1] . Indeed, PTA are PTBP networks of size one and timed networks are PTBP networks without timing parameters.

In this paper, we consider parameterized reachability problems: we ask whether there exists a network size N satisfying a given reachability property. We consider existential (EF) and universal (AF) reachability properties that ask, given goal state q_f, whether this state is reached by some (EF) or all (AF) executions. Moreover we also consider variants on the quantifier on timing parameters and ask whether the property holds for all parameter valuation (universality) or for none (emptiness).

Thus, given a bounded PTBP $(\mathcal{N}, bounds)$ and a state q_f we consider the following problems:

∃-**EF-emptiness** $\exists N \in \mathbb{N}, \exists p \in bounds, \exists \rho \in \mathcal{E}(\mathcal{N}, N, p), \rho \models \Diamond q_f$

∃-**EF-universality** $\exists N \in \mathbb{N}, \forall p \in bounds, \exists \rho \in \mathcal{E}(\mathcal{N}, N, p), \rho \models \Diamond q_f$

∃-**AF-emptiness** $\exists N \in \mathbb{N}, \exists p \in bounds, \forall \rho \in \mathcal{E}(\mathcal{N}, N, p), \rho \models \Diamond q_f$

∃-**AF-universality** $\exists N \in \mathbb{N}, \forall p \in bounds, \forall \rho \in \mathcal{E}(\mathcal{N}, N, p), \rho \models \Diamond q_f$

For convenience, we will ommit the bounds when they are irrelevant and consider these problems in the case of general PTBP. In the following, the bounds will only be relevant in Section 5.

In the next section we investigate these problems in the general semantics defined above. This semantics is called *reconfigurable* since the communication topology (modeled by the reception sets) can be reconfigured at each step. However, in broadcast protocol networks with a parametric number of processes, the communication topology plays a decisive role on decidability status. We will thus investigate an other communication setting, in Section 4, in which every message is received by all the other processes i. e., the reception set R is always equal to $\{1, \ldots, N\}$. These networks are called *clique* networks.

▶ **Example 4.** Considering the PTBP given in Example 2 and the target state g. The execution presented in Example 3 shows that the answer for the ∃-EF-emptiness problem is positive whenever the bounds allow for $tl = 9$ and $pt = 3$ in the reconfigurable semantics. Notice that in the clique semantics, it is not possible to reach g unless $pt * 3 < tl$. Indeed in the clique semantics when a first process moves to f, all the other processes receive the message f and thus move to c. Thus, at least three pt time units are necessary in order to receive 3 messages p.

Notice also that in this example, in both semantics, both ∃-AF problems would give negative answers since there is always an execution that forever sends p in the bottom self-loop and never uses the internal transition leading to g. Thus such an execution never reaches g.

3 Reconfigurable semantics

3.1 AF problems in the reconfigurable semantics

The reconfigurable semantics of broadcast networks, where the set of receivers can be chosen non-deterministically, makes the AF problems equivalent to the same problems in networks of size 1. This is due to the fact that in the reconfigurable semantics nothing prevents messages to be sent to an empty set of receivers. The following theorem is a direct consequence of previous known results on parameterized timed automata and this previous remark.

▶ **Theorem 5.** *∃-AF-emptiness and ∃-AF-universality are decidable for 1 clock PTBP but undecidable for (L/U)-PTBP with 3 clocks or more.*

Proof. We first show that in the reconfigurable semantics the ∃-AF problems are equivalent to the same problems but in networks of size one.

The easiest direction is to assume that an AF property holds in a network of size one, it thus answers the original parameterized question which asks whether there exists a size of network satisfying the property. The other direction is more subtle and derives from the fact that in the reconfigurable semantics nothing prevents messages to be sent to an empty set of receivers. In the case of executions where there is no communication (every message is sent with an empty set of receivers) the network behaves as several processes running in parallel without interaction. Thus, if the AF problem is satisfied for some size of networks, it means that the target state is reached in particular for every execution without communication. Thus, it follows that the target is reached for all executions with a single process hence it holds for a network of size 1.

The rest of the theorem follows from the literature. AF-emptiness and AF-universality are decidable for 1-PTA. Indeed it is shown in [8] that one can abstract 1-PTA semantics into a finite parameterized zone abstraction. Moreover one can use this abstraction to solve the AF problems as shown in [19]. The undecidable cases directly come from the fact that the AF-emptiness is undecidable for (L/U-)PTA with 3 clocks or more [19]. The decidable case comes from the following result for PTA with 1 clock, which was, to the best of our knowledge, never shown formally:

▶ **Lemma 6** (Decidability of AF-emptiness for 1-clock PTA)**.** *The AF-emptiness problem is decidable for 1-clock PTA.*

Proof. A classical way to approach timed automaton is to abstract the reachable configurations into a finite region graph (see *e.g.* [9]). This region graph can be adapted in the case of PTA in order to deal with parameters. It was shown in [8] that a similar abstraction

called zone graph is finite for 1-clock PTA. To obtain the decidability of the AF-emptiness problem it then suffices to apply the symbolic algorithm given in [19]. ◀

This concludes the proof of Theorem 5.
Notice that these questions are still open for L-PTA, U-PTA and 2-PTA. ◀

▶ Remark. Following the reasoning in the proof of Theorem 5, the ∃-AF-emptiness problem is open in the same cases as for PTA: for L-PTBP and U-PTBP, and for (L/U)-PTBP with 2 clocks.

3.2 EF problems in the reconfigurable semantics

We start by recalling some known results on networks composed of an arbitrary number of timed processes. In [4] the authors considered timed networks and proved that the reachability problem (∃-EF) is decidable with one clock per process and undecidable for two clocks per process [3]. Note that timed networks have a different semantics than the one we use in this paper since they use rules and not broadcasts. However the reconfigurable semantics can be easily encoded in the rules of timed networks. This gives us the decidability of the ∃-EF problem (without timing parameters and with one clock per process).

▶ **Theorem 7** ([4, 3]). *∃-EF is decidable for PTBP without parameters and with one clock per process and undecidable with two clocks per process.*

A direct consequence of this theorem is the undecidability of the ∃-EF problems for PTBP with two clocks.

▶ **Lemma 8.** *The ∃-EF-emptiness and ∃-EF-universality problems are undecidable for PTBP with two clocks.*

Moreover, we show below that the undecidability even holds for PTBP with a single clock. This is a major difference with both parameterized networks and PTA, where the restriction to one clock leads to decidability.

▶ **Theorem 9.** *The ∃-EF-emptiness and ∃-EF-universality problems are undecidable for PTBP with one clock.*

Proof. The proof is by reduction of the halting problem for two-counter machines. The idea of the reduction relies in part to the one of undecidability in PTA with 3 clocks [19]. We hence start by recalling this proof. Given a PTA with 3 clock t, x and y, the idea is to encode the value of c_1 by the difference $t - x$ and the value of c_2 by the difference $t - y$. The clocks are reset to zero each time they reach the value $p(\lambda)$ of the parameter λ. An increment is, for example, encoded by resetting the clock x when it is equal to 1 (hence the difference with t increase by 1), for a test to zero it suffices to check whether $x = 0$ when $t = 0$. Notice that the construction is valid only if the counter value does not exceed $p(\lambda)$.

In our model however we cannot apply this idea directly since there is only one clock per process, however we can use the fact that there are several processes to dispatch the clocks between different processes. We therefore separate our protocol in three different parts: a controller that represents the current instruction and clock t, and two counters that each represent one of the counters, thus clocks x and y. The messages are then used to make those three processes cooperate. The difficulty is that, in our semantics, the message may not reach all the processes. In particular it may be the case that in a first part of an execution the network is split in two distinct parts which do not communicate between

them, and at some point start communicating. We have to ensure that the reduction we propose is not affected by such a behavior. The proof that our construction is correct is technical and given in Appendix B. ◀

4 Clique

In broadcast protocol networks with a parametric number of processes, the topology of message communication plays a decisive role on the decidability status. In this section, we thus investigate a communication setting in which every message is received by all the other processes. We call these networks clique networks.

Formally, the semantics of a clique network is the restriction of the semantics given in Section 2 to internal transitions and broadcast transitions in which the set of receivers is always composed of all processes.

4.1 AF problems in the clique semantics

We first rule out the ∃-AF problem for the clique semantics, as we can show from [17] that it is undecidable already without any clock.

▶ **Theorem 10.** *The ∃-AF problem is undecidable for PTBP with no clock in the clique semantics.*

Proof. In [17, Chapter III, Theorem 3.5] it is shown that one can reduce the halting problem of a two-counter machine (which is undecidable [22]) to the AF problem in a clique network without clocks.

Intuitively the reduction goes as follows: the values of the counter are encoded by the number of processes in a given state. Increment and decrement of counter are easy to encode since in the clique semantics when one process sends a message everyone receives it, thus we can ensure that only one process performs the increment or decrement. The difficulty comes from the zero tests. Indeed, since we cannot force processes to answer we cannot differentiate between the case where there is no process encoding a counter and the case where the processes do not answer. To tackle this problem, zero tests are implemented non-deterministically: if we choose that the counter is zero, a message is sent. If it was not the case, then the processes encoding the counter value move to an error state. In the case we choose that the value is not zero, the network is locked until a process encoding the counter sends a message or a process moves to the error state. This encoding ensures that every run that does not encode truthfully the two-counter machine reaches the error state. Thus by adding a transition from the halting state of the counter machine toward the error state, we can ensure that every path reaches the error state if and only if the two-counter machine halts. ◀

4.2 EF problems in the clique semantics

Notice that the proof of Theorem 9 (see Appendix B) does not rely on the reconfigurable semantics particularity. In fact the strong synchronization of processes in the clique semantics makes it even easier. We thus obtain the following lemma:

▶ **Lemma 11.** *The ∃-EF-emptiness and ∃-EF-universality problems are undecidable for PTBP.*

This undecidability does not hold in the case where each parameter appears either always as an upper bound or always as a lower bound in guards (but not both). We thus consider in the following the case of L/U-PTBP.

5 1-clock L/U-PTBP

Since the L/U restriction brings some decidability to PTAs, we focus in this section on L/U-PTBP. Recall that L/U-PTA are expressive enough to model classical examples from the litterature [18], such as root contention or Fischer's mutual exclusion algorithm. As a consequence, L/U-PTBP make an interesting subclass of PTBP.

Due to the undecidability results of [1] for processes with 2 clocks (already without parameters), we consider in this section L/U-PTBP with one clock only. When considering L/U-PTBP, we can get the following monotonicity result on the timing parameter valuations.

▶ **Lemma 12.** *Given $\mathcal{N}$ an L/U-PTBP with one clock, a network size $N \in \mathbb{N}$, and a parameter valuation p, for all valuations p' such that for all upper-bound parameters λ^u, $p(\lambda^u) \leq p'(\lambda^u)$ and for lower-bound parameters λ^l, $p(\lambda^l) \geq p'(\lambda^l)$ we have that $\forall \rho \in \mathcal{E}(\mathcal{N}, N, p)$, $\exists \rho' \in \mathcal{E}(\mathcal{N}, N, p')$ such that ρ is a prefix of ρ'.*

Proof. The proof is direct from the semantics definition. Notice that we do not have full inclusion of $\mathcal{E}(\mathcal{N}, N, p)$ in $\mathcal{E}(\mathcal{N}, N, p')$ since we consider maximal executions and it may be the case that some executions of $\mathcal{E}(\mathcal{N}, N, p)$ appear only as prefixes of executions of $\mathcal{E}(\mathcal{N}, N, p')$. Notice also that this holds in both semantics (reconfigurable and clique). ◀

A direct consequence of Lemma 12 and the decidability of the EF problem for PTBP with a single clock and without parameters is the decidability of the ∃-EF-emptiness problems for L/U-PTBP with one clock.

▶ **Lemma 13.** *The ∃-EF-universality problem is decidable for closed bounded L/U-PTBP with one clock in both semantics.*

Proof. Let $\mathcal{N}$ be an L/U-PTBP with one clock, and *bounds* be the closed bounds on the parameters. Let p_{min} be the minimal permissive valuation i. e., the valuation such that for all upper-bound parameters λ^u, $p_{min}(\lambda^u) = \inf(\lambda^u, bounds)$ and for all lower-bound parameters λ^l, $p_{min}(\lambda^l) = \sup(\lambda^l, bounds)$. By definition we have $p_{min} \in bounds$.

We define the PTBP without parameters $\mathcal{N}_{min}$ as $\mathcal{N}$ but replacing each occurrence of an upper-bound parameter λ^u by $\inf(\lambda^u, bounds)$ and each occurrence of a lower-bound parameter λ^l by $\sup(\lambda^l, bounds)$. It is then easy to see that $\mathcal{E}(\mathcal{N}, N, p_{min}) = \mathcal{E}(\mathcal{N}_{min}, N)$.

Assume that for all N there is no execution reaching q_f in $\mathcal{E}(\mathcal{N}_{min}, N)$; then the above equality implies that the answer to ∃-EF-universality is false.

Conversely assuming that there exists an execution reaching q_f in $\mathcal{E}(\mathcal{N}_{min}, N)$ for some N, we obtain by the equality and the monotonicity Lemma 12 that this execution is a prefix of an execution of $\mathcal{E}(\mathcal{N}, N, p)$ for any valuation p.

Thus the ∃-EF-universality problem for $\mathcal{N}$ is equivalent to the ∃-EF problem for $\mathcal{N}_{min}$ and thus is decidable in the clique semantics (see [1]) and in the reconfigurable semantics (see Theorem 7). ◀

For the ∃-EF-emptiness problem, we can remove the assumption on the closed bounds.

▶ **Lemma 14.** *The ∃-EF-emptiness problem is decidable for (open or closed) bounded L/U-PTBP with one clock in both semantics.*

Proof. Let $\mathcal{N}$ be an L/U-PTBP with one clock, and *bounds* be the bounds on the parameters. As for the ∃-EF-universality problem, we define a protocol $\mathcal{N}_{max}$ with the difference that non-strict guards involving open bounded parameters are changed to strict guards. We define the PTBP without parameters $\mathcal{N}_{max}$ as $\mathcal{N}$ but for all upper-bound parameters λ^u if $bounds(\lambda^u)$ is of the form $(\inf, \sup]$ or $[\inf, \sup]$ then every occurrence of λ^u is replaced by sup. Otherwise if $bounds(\lambda^u)$ is of the form $(\inf, \sup)$ or $[\inf, \sup)$ then every guard of the form $x < \lambda^u$ or $x \leq \lambda^u$ is replaced by the guard $x < \sup$. We operate similarly for lower-bound parameters.

Using the same argument as for the monotonicity Lemma 12 it is easy to see that for any valuation $p \in bounds$, any execution ρ in $\mathcal{E}(\mathcal{N}, N, p)$ is a prefix of some execution in $\mathcal{E}(\mathcal{N}_{max}, N)$. Thus if some execution reaches q_f for some N and some p in $\mathcal{E}(\mathcal{N}, N, p)$, there is also an execution reaching q_f in $\mathcal{E}(\mathcal{N}_{max}, N)$.

The other direction is more subtle. Assume that there exists an execution ρ reaching q_f in $\mathcal{E}(\mathcal{N}_{max}, N)$. Let ρ' be a finite prefix of ρ reaching q_f. We define a valuation $p \in bounds$ that contains an execution identical to ρ' as follows: Let λ^u be an upper-bound parameter. Either $bounds(\lambda^u)$ is of the form $(\inf, \sup]$ or $[\inf, \sup]$ and we define $p(\lambda^u) = \sup$. Or $bounds(\lambda^u)$ is of the form $(\inf, \sup)$ or $[\inf, \sup)$. In this case, let v_u be the maximal value of clock x along ρ' when x is compared in a guard which was formerly $x \bowtie \lambda^u$. By definition of $\mathcal{N}_{max}$ we know that $v_u < \sup$. We thus define $p(\lambda^u) = v_u + \epsilon$ with $\epsilon > 0$, $\epsilon + v_u < \sup$ and $\epsilon > \inf - v_u$ (it exists since necessarily $\inf < \sup$).

We operate in a symmetrical way for lower-bound parameters: v_l is the minimal value of clock x along ρ' when x is compared in a guard which was formerly $x \bowtie \lambda^l$ and $p(u) = v_u - \epsilon$ with $v_u - \epsilon > \inf$, $\epsilon > 0$ and $\epsilon < \sup + v_u$ (it exists since necessarily $\sup > \inf$).

It is easy to see that for this valuation, ρ' is a prefix of some execution in $\mathcal{E}(\mathcal{N}, N, p)$. Hence, the ∃-EF-emptiness problem for $\mathcal{N}$ is equivalent to the EF problem for $\mathcal{N}_{max}$ and thus decidable in the clique semantics ([1]) and in the reconfigurable semantics (Theorem 7). ◀

In contrast with the ∃-EF-emptiness problem, the monotonicity result is not enough to show decidability of the ∃-EF-universality problem for L/U-PTBP with open bounds. In fact we can even show that the problem becomes undecidable for general L/U-PTBP in the clique semantics. More precisely it is undecidable for U-PTBP with one parameter with open left bound, and for L-PTBP with one unbounded parameter.

▶ **Theorem 15.** *The ∃-EF-universality problem is undecidable for open bounded L/U-PTBP with one clock in the clique semantics.*

Proof. This proof is inspired by the proof of [17, Chapter III, Theorem 3.5]. The idea is to encode a two-counter machine, the number of processes in a particular state is used to encode the counter value. Thanks to the clique semantics, increment and decrement of counter is easy to simulate. However, zero tests are not possible since there is no way to distinguish between the fact that no process is modeling a counter and the fact that they just do not send a message. We thus allow the simulation to guess whether the counter is zero or not zero non-deterministically, in case of a wrong guess we are able to detect it thanks to the clique semantics. In this case at least one process is stuck in an error state, we then use the time parameter to repeat the simulation an unbounded number of time before moving to the target state. To be able to reach the target state we thus have to be able to correctly simulate the two-counter machine without wrong guess.

Formally, given a two-counter machine $\mathbf{M} = (\mathbf{K}, \mathbf{k}_0, \mathbf{k}_{acc})$ we define a PTBP $\mathcal{P}$ as follows:

- $Q = \{q_0, idle, c_i, c_i^d, c_i^i, c_i^z, err, q_f \mid i \in \{1, 2\}\} \cup \{k, k' \mid k \in K\}$ where, q_0 is the initial state, *idle* is a waiting state for the processes encoding the counters, c_i is the state used

to encode counter c_i value, c_i^i and c_i^d are intermediary states for increment and decrement of counter c_i, c_i^z is an intermediary state used for the zero test, a state k is used to encode that the simulation reached instruction k of the machine and k' is an intermediary state, err is a sink state used to detect error in the simulation, finally q_f is the target state

- $\mathbb{X} = \{x\}$ and $\mathbb{P} = \{\lambda^u, \lambda^l\}$
- $\Sigma = \{inc_i, dec_i, z_i, nz_i, ok, end \mid i \in \{1,2\}\}$ where inc_i, dec_i, z_i, and nz_i stand respectively for increment, decrement, zero, and not zero of counter c_i, ok is a message to acknowledge that the action was performed correctly, and end is the message sent at the end of the simulation to either restart a simulation or reach the target state.
- Δ is define as follows, for simplicity the guard and update of the clock are omitted when trivial (i. e., the true guard and no reset):

  initialization $(q_0, !!ok, k_0) \in \Delta$, $(q_0, ??ok, idle) \in \Delta$

  increment of counter i for any increment of counter i instruction $\mathbf{k} : incr\ \mathbf{C}_i\ goto\ \mathbf{k}_1$ we add to Δ the transitions: $(k, !!inc_i, k')$, $(k', ??ok, k_1)$, $(idle, ??inc_i, c_i^i)$, $(c_i^i, !!ok, c_i)$ $(c_i^i, ??ok, idle)$

  decrement of counter i for any decrement of counter i instruction $\mathbf{k} : decr\ c_i\ goto\ \mathbf{k}_1$ we add to Δ the transitions: $(k, !!dec_i, k')$, $(k', ??ok, k_1)$, $(c_i, ??dec_i, c_i^d)$, $(c_i^d, !!ok, idle)$ $(c_i^d, ??ok, c_i)$

  zero test of counter i for any zero test of counter i instruction $\mathbf{k} : if\ c_i = 0\ goto\ \mathbf{k}_z$ $else\ goto\ \mathbf{k}_{nz}$ we add to Δ the transitions: $(k, !!z_i, k_z)$, $(k, !!nz_i, k')$, $(k', ??ok, k_{nz})$, $(c_i, ??z_i, err)$, $(c_i, ??nz_i, c_i^z)$ $(c_i^z, !!ok, c_i)$, $(c_i^z, ??ok, c_i)$

  end of simulation $(k_{acc}, x < \lambda^u, !!end, \{x := 0\}, k_0)$ $(idle, x > \lambda^l, ??end, q_f)$ $(c_i, ??end, idle)$

Given a configuration γ of the network, we say that it encodes a configuration (k, v_1, v_2) of the two-counter machine if there is one process in state k and v_i processes in states c_i for $i \in \{1,2\}$. Since, if we omit the *end of simulation* part, this reduction is similar to the one found in [17, Chapter III, Theorem 3.5], we leave the reader find there the detailed proof of the fact that every execution of the network is of one of the three kinds:

correct simulation the execution correctly encodes the run of the two-counter machine

lack of processes the controller is stuck in an intermediary state while performing an increment, *i.e* there was no process left in the *idle* state when the controller sent the inc_i message, thus it is stuck waiting for an *ok* message that no one can send

wrong zero test along the execution the controller wrongly assumed the value of a counter. Either it guessed a non-zero value and it is stuck waiting for an *ok* message, or it guessed zero when it was not in which case at least one process moved to the error state.

Notice now that to reach the target state q_f a process in *idle* must receive the message *end* after its clock value is greater than parameter λ^l. But the end of simulation part requires that the controller clock is lower than parameter λ^u. Thus when reaching state k_{acc}, in order to be able to let more time elapse the controller has to send the message *end* which leads to a configuration where there is no process in the counter states and the controller is in the initial state of the two-counter machine. This configuration thus encodes the initial configuration of the two-counter machine. The controller then must simulate another time the two-counter machine before being able to send *end* again.

Thus, given a valuation p of the parameters, to reach q_f at least $p(\lambda^l)/p(\lambda^u)$ messages *end* must be sent by the controller. In other words, $p(\lambda^l)/p(\lambda^u)$ (correct or incorrect) simulations of the two-counter machine must be performed before reaching q_f. We have seen before that every incorrect simulation either gets stuck, or sends at least a process in the error state. Hence, given a network size N, if for a valuation p such that $p(\lambda^l)/p(\lambda^u) > N$ the state q_f is reached, then at least one simulation was correct, thus the two-counter machine halts.

	1-c	2-c	3-c	1-L/U cb	1-L/U ob	2-L/U	3-L/U
∃-EF-empt.	*9*			**14**		*8*	
∃-EF-univ.	*9*			**13**	*open*	*8*	
∃-AF	**5**	*open*	*5*	**5**		*open*	*5*

(a) Reconfigurable semantics

	PTBP	L/U cb	L/U ob	L or U cb	L or U ob
∃-EF-empt.	*11*	**14**		**14**	
∃-EF-univ.	*11*	**13**	*15*	**13**	*16*
∃-AF	*10*				

(b) Clique semantics for 1 clock

Table 1 Summary of our contributions (bold green: decidable; red italic: undecidable)

This proves the undecidability of the EF-universality problem with 0 as an open lower bound for λ^u. Indeed, if there exists a network of size N which satisfies the EF-universality, then it is possible to reach q_f for all valuation and in particular for a valuation such that $p(\lambda^l)/p(\lambda^u) > N$. For the other direction, if the machine halts there exists a size of network ($m+2$ where m is the maximal sum f the two-counter value along the execution) that ensures that q_f is reachable for any valuation p with $p(\lambda^u) > 0$. Indeed, the controller can simulate the two-counter machine correctly (since it has enough processes to model the counters) in 0 time unit, wait a positive delay but less than λ^u time unit, and repeat this until the clock value of the processes in *idle* is greater than λ^l. This is possible since every time the controller sends the message *end* the configuration obtained is the same as the one obtained after the initialization (the first message *ok*). ◀

▶ **Lemma 16.** *∃-EF-universality in the clique semantics is undecidable already with a single clock for U-PTBP with open bounds on the left, and L-PTBP with infinity as right bound.*

Proof. The proof of Theorem 15 uses an open bounded L/U-PTBP. Moreover we only used the fact for all size of network N there exists a valuation of the parameter p such that $p(\lambda^l)/p(\lambda^u) > N$. Thus the proof can be adapted with only one upper-bound parameter λ^u (resp. lower-bound parameter λ^l) by replacing λ^l by 1 in the protocol (resp. λ^u by 1). This still ensures that there exists a valuation such that $1/p(\lambda^u) > N$ (resp. $p(\lambda^l) > N$). ◀

6 Conclusion

Up to our knowledge this work is the first to consider two different sets of parameters at the same time. Both parameterized number of processes and parametric clocks are difficult to deal with and number of problems are undecidable for each of these systems. However we have shown that the combination of the decidable subclasses leads to some decidable problems. Our contributions are summarized in Table 1; i-c (resp. i-L/U) denotes PTBP (resp. L/U-PTBP) with i clocks per process. In Table 1b, cb and ob denote formalisms with a closed bounded parameter domain and an open bounded parameter domain.

Future works include considering other semantics such as asynchronous broadcast or different communication topologies (reconfigurable under constraint, restricted to graph of bounded width, ...), as well as the reachability problem for *all* sizes of networks (instead of the *existence* of a network size). The open 2-clock case in the reconfigurable semantics is a well-known open problem, with connections to open problems of logic and automata theory [6]. The other open case in Table 1 we are interested in solving is ∃-EF-universality for 1-L/U-PTBP in the reconfigurable semantics with open bounds. In addition, EF problems are still open for bounded 1-clock PTBP (Theorem 9 requires unbounded parameters), and for 1-c L/U with unbounded parameters in the clique semantics. Finally, for the decidable subclasses we exhibited, it remains to be studied whether exact synthesis can be achieved,

i. e., obtaining the set of sizes of processes and timing parameter valuations for which EF or AF holds.

References

1. Parosh A. Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2011. doi:10.1007/978-3-642-24310-3_18.
2. Parosh A. Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612:1–22, 2016. doi:10.1016/j.tcs.2015.07.048.
3. Parosh A. Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *LiCS*, pages 345–354. IEEE, 2004.
4. Parosh A. Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.
5. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
6. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.
7. Étienne André, Michał Knapik, Wojciech Penczek, and Laure Petrucci. Controlling actions and time in parametric timed automata. In *ACSD*, pages 45–54. IEEE Computer Society, 2016. doi:10.1109/ACSD.2016.20.
8. Étienne André and Nicolas Markey. Language preservation problems in parametric timed automata. In *FORMATS*, volume 9268 of *Lecture Notes in Computer Science*, pages 27–43. Springer, 2015. doi:10.1007/978-3-319-22975-1_3.
9. Christel Baier, Joost-Pieter Katoen, and Kim G. Larsen. *Principles of model checking*. MIT press, 2008.
10. Nikola Beneš, Peter Bezděk, Kim G. Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2015. doi:10.1007/978-3-662-47666-6_6.
11. Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 1997.
12. Giorgio Delzanno and Pierre Ganty. Automatic verification of time sensitive cryptographic protocols. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer, 2004.
13. Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
14. Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *International Conference on Concurrency Theory*, pages 313–327. Springer, 2010.
15. Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS*, volume 11, pages 441–455. Springer, 2011.
16. Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of safety properties in ad hoc network protocols. *arXiv preprint arXiv:1108.1864*, 2011.

610 **17** Paulin Fournier. *Parameterized verification of networks of many identical processesVérifi-*
611 *cation paramétrée de réseaux composés d'une multitude de processus identiques*. PhD thesis,
612 Rennes 1, 2015.
613 **18** Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear para-
614 metric model checking of timed automata. *Journal of Logic and Algebraic Programming*,
615 52-53:183–220, 2002.
616 **19** Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for
617 timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.
618 **20** Li Li, Jun Sun, Yang Liu, and Jin Song Dong. Verifying parameterized timed security
619 protocols. In *FM*, volume 9109 of *Lecture Notes in Computer Science*, pages 342–359.
620 Springer, 2015. doi:10.1007/978-3-319-19249-9_22.
621 **21** Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear
622 hybrid automata. In *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages
623 296–309. Springer, 2000.
624 **22** Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

A Two-counter machines

For completeness, we recall here the definition of two-counter machines as well as the undecidability of the halting problem.

▶ **Definition 17** ([22]). A two-counter machine is a tuple $\mathbf{M} = (\mathbf{K}, \mathbf{k}_0, \mathbf{k}_{acc})$ manipulating integer variables $\mathbf{C}_1$ and $\mathbf{C}_2$ called counters and composed of a finite set of instructions $\mathbf{K}$. Each instruction $\mathbf{k} \in \mathbf{K}$ is either of the form:
Increment $\mathbf{k} : inc\ \mathbf{C}_i;\ goto\ \mathbf{k}'$, or
Decrement $\mathbf{k} : decr\ \mathbf{C}_i;\ goto\ \mathbf{k}'$, or
Zero test $\mathbf{k} : if\ \mathbf{C}_i = 0\ goto\ \mathbf{k}'\ else\ goto\ \mathbf{k}''$
where $i \in \{1, 2\}$ and $\mathbf{k}, \mathbf{k}', \mathbf{k}''$ are labels preceding instructions. $\mathbf{k}_0$ is the initial label and $\mathbf{k}_{acc}$ is the accepting label.

A configuration is a tuple of $\mathbf{K} \times \mathbb{N} \times \mathbb{N}$. A configuration $(\mathbf{k}, \mathbf{c}_1, \mathbf{c}_2)$ means that the machine is at instruction $\mathbf{k}$ with counter $\mathbf{C}_1$ with value $\mathbf{c}_1$ and counter $\mathbf{C}_2$ with value $\mathbf{c}_2$.

A two-counter machine gives rise to a run $\mathbf{s}_0 \to \mathbf{s}_1 \to \dots$ where $\mathbf{s}_0 = (\mathbf{k}, 0, 0)$ and for all $\mathbf{s}_i = (\mathbf{k}, \mathbf{c}_1, \mathbf{c}_2)$ the successor configuration depends on the form of $\mathbf{k}$.

- if $\mathbf{k} : inc\ \mathbf{C}_1;\ goto\ \mathbf{k}'$ then $\mathbf{s}_{i+1} = (\mathbf{k}', \mathbf{c}_1 + 1, \mathbf{c}_2)$ (similarly for an increment of $\mathbf{C}_2$)
- if $\mathbf{k} : decr\ \mathbf{C}_1;\ goto\ \mathbf{k}'$ then $\mathbf{s}_{i+1} = (\mathbf{k}', \mathbf{c}_1 - 1, \mathbf{c}_2)$ (similarly for an decrement of $\mathbf{C}_2$)
- if $\mathbf{k} : if\ \mathbf{C}_1 = 0\ goto\ \mathbf{k}'\ else\ goto\ \mathbf{k}''$ and $\mathbf{c}_1 = 0$ then $\mathbf{s}_{i+1} = (\mathbf{k}', 0, \mathbf{c}_2)$ (similarly for $\mathbf{C}_2$)
- $\mathbf{k} : if\ \mathbf{C}_1 = 0\ goto\ \mathbf{k}'\ else\ goto\ \mathbf{k}''$ and $\mathbf{c}_1 > 0$ then $\mathbf{s}_{i+1} = (\mathbf{k}'', \mathbf{c}_1, \mathbf{c}_2)$ (similarly for $\mathbf{C}_2$)

We assume without loss of generality that the run is either infinite or the machine halt in $\mathbf{k}_{acc}$. Moreover we can assume without loss of generality that the two last instructions are necessarily zero test of both counter. The halting problem asks whether the machine halts, and the boundedness problem asks whether the counter values are bounded along the run. Both problems have been shown undecidable in [22].

B Proof of Theorem 9

Theorem 9 (recalled). *The $\exists$-EF-emptiness and $\exists$-EF-universality problems are undecidable for PTBP with one clock.*

Proof. The proof is by reduction of the halting and boundedness (respectively) problems for two-counter machines.

First, in this proof we will assume that the parameter λ only takes integer values. This is not a restriction since we can add a gadget at the beginning of the PTBP to check such property. This gadget is given in Fig. 2.

Given a two-counter machine (see Appendix A), we define a protocol $\mathcal{P}$ separated in three parts, the controller part (in charge of tracking the current instruction), the counter parts (to model the counters behaviors) and an idle part that allow to use additional processes when needed.

Recall that the value of the counters is encoded (up to the value of parameter λ minus 1 here for technical reasons) by the difference between the clock value of the processes in states representing counters and the clock value of the processes in the controller part.

Formally $\mathcal{P}$ is defined as follows:

- $Q = \{q_0, error, c_i, nc1_i, nc2_i, zt1_i^j, zt2_i^j, dec1_i^j, dec2_i^j, inc1_i^j, inc2_i^j, inc3_i^j, idle \mid j \in \{1,2\}, i \in \{1,2\}\} \cup \{k^j \mid \mathbf{k} \in \mathbf{K}, j \in \{1,2,3,4\}\}$
- $\Sigma = \{tick, inc_i, dec_i, zt_i, c_i, oc_i, nc_i \mid i \in \{1,2\}\}$

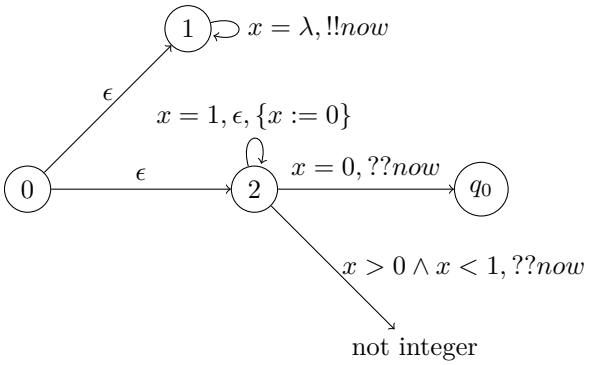


Figure 2 Gadget to enforce integer values of λ

- $\mathbb{P} = \{\lambda\}$
- $\mathbb{X} = \{x\}$
- Δ is defined as follows, on every transition there is a guard $x \leq \lambda$ which is omitted to clarify notations, similarly when a guard is true (here limited to $x \leq \lambda$) or there is no reset we omit them in the transition. The construction is represented in Fig. 3. Δ is composed of the following transitions:

  initialization $(q_0, x = 0, \epsilon, k_0^1)$, for $i \in \{1, 2\}$, $(q_0, x = 0, \epsilon, c_i)$, $(q_0, x = 0, \epsilon, idle)$
  The processes can chose non-deterministically to either move to the controller part, the counters part, or the idle part.

  decrement of counter i for any decrement instruction $\mathbf{k} : decr\ \mathbf{C}_i\ goto\ \mathbf{k}_1$ there are the following transitions in Δ
  - for the controller: $(k^1, x = 1, !!dec_i, k^2)$ $(k^2, x = \lambda, !!tick, \{x := 0\}, k_1^1)$
    The controller announce that the instruction is a decrement when its clock is equal to 1 and then announce when its clock reach the value of the parameter.
  - for the counter involved: $(c_i, x > 1, ??dec_i, dec1_i^i)$, $(dec1_i^i, x = \lambda, \epsilon, \{x := 0\}, dec2_i^i)$ $(dec2_i^i, x = 1, \epsilon, \{x := 0\}, dec3_i^i)$ $(dec3_i^i, ??tick, c_i)$
    When the process representing the counter receive the message corresponding to the decrement they move to an intermediary state, they then reset their clock when they reach λ and they reset it an other time when the clock reach 1. This way the difference with the controller clock has decreased by one. Notice that if $x = 1$ when they receive the decrement message (meaning that the counter has value 0) they cannot take the transition.
  - for the counter not involved: $(c_j, ??dec_i, dec1_i^j)$ $(dec1_i^j, x = \lambda, \{x := 0\}, decj2_i^j)$ $(dec2_i^j, ??tick, c_j)$.
    The counter not involved just resets their clock when they reach λ, thus the difference remain constant.

  increment of counter i for any increment instruction $\mathbf{k} : incr\ \mathbf{C}_i\ goto\ \mathbf{k}_1$ there are the following transitions in Δ
  - for the controller: $(k^1, x = 1, !!inc_i, k^2)$ $(k^2, x = \lambda, !!tick, \{x := 0\}, k_1^1)$
    The controller announce that the instruction is an increment when its clock is equal to 1 and then announce when its clock reach the value of the parameter.
  - for the counter involved:
    The clock value should be reset at $\lambda - 1$, but such a guard is not allowed and is not possible to encode with just one clock. We thus rely on a non-deterministic guess, that is the checked by a new process. This is done as follow:

for the current counter processes $(c_i, x < \lambda, ??inc_i, inc1_i^i)$,$(c_i, x = \lambda, ??inc_i, error)$, $(inc1_i^i, !!nc_i, inc2_i^i)$ $(inc2_i^i, x = \lambda, !!oc_i, idle)$
The processes encoding the counter receive the increment message and then guess non-deterministically that it's clock value is $\lambda - 1$ and send a message nc_i. In order to check that the guess was right it then announce when it's clock reach λ by sending message oc_i, and the process move to $idle$. The value of the counter will then be encoded by the new processes. Notice that if the clock value is already equal λ then we reached the maximal value possible to encode and the process move to the error state $error$.

for the new counter process $(idle, ??nc_i, \{x := 0\}, nc1_i)$ $(nc_i, x = 1, ??oc_i, nc2_i)$ $(nc2_i, ??tick, c_i)$
To check that the guess was right we use the idle processes that when receiving the message nc_i reset their clock. They are then allowed to encode the counter if they receive the confirmation oc_i when their clock is equal 1 (thus the guess was correct).

- for the counter not involved: $(c_j, ??inc_i, inc1_i^j)$ $(inc1_i^j, x = \lambda, \{x := 0\}, incj2_i^j)$ $(inc2_i^j, ??tick, c_j)$.
The counter not involved just reset it's clock when it reach λ.

zero test for any zero test instruction $\mathbf{k} : if\ c_i = 0\ then\ goto\ \mathbf{k}_1\ else\ goto\ \mathbf{k}_2$ there are the following transitions

- for the controller $(k^1, x = 1, !!zt_i, k^2)$ $(k^2, x = \lambda, ??c_i, k^3)$ $(k^2, x < \lambda, ??c_i, k^4)$ $(k^3, x = \lambda, !!tick, \{x := 0\}, k_1^1)$ $(k^4, x = \lambda, !!tick, \{x := 0\}, k_2^1)$
The controller announce that the instruction is a zero test when its clock is equal 1 and then wait for a notification c_i from the counter. Depending when this notification arrive, when $x = \lambda$ (meaning the counter has value 0) or when $x < \lambda$ (meaning the counter has positive value), the controller moves to the corresponding intermediary states.
- for the counter involved $(c_i, ??zt_i, zt1_i^i)$ $(zt1_i, x = \lambda, !!c_i, \{x := 0\}, zt2_i^i)$ $(zt2_i^i, ??tick, c_i)$
The counter involved, after receive the instruction, sends a notification c_i when it's clock reach λ.
- for the counter not involved $(c_j, ??zt_i, zt1_i^j)$ $(zt1_i, x = \lambda, \epsilon, \{x := 0\}, zt2_i^j)$ $(zt2_i^j, ??tick, c_j)$
The counter not involved just reset it's clock when it reach λ.

finally there is an additional transition $(idle, \epsilon, \{x := 0\}, idle)$ used to keep the clock of idle processes below $p(\lambda)$.

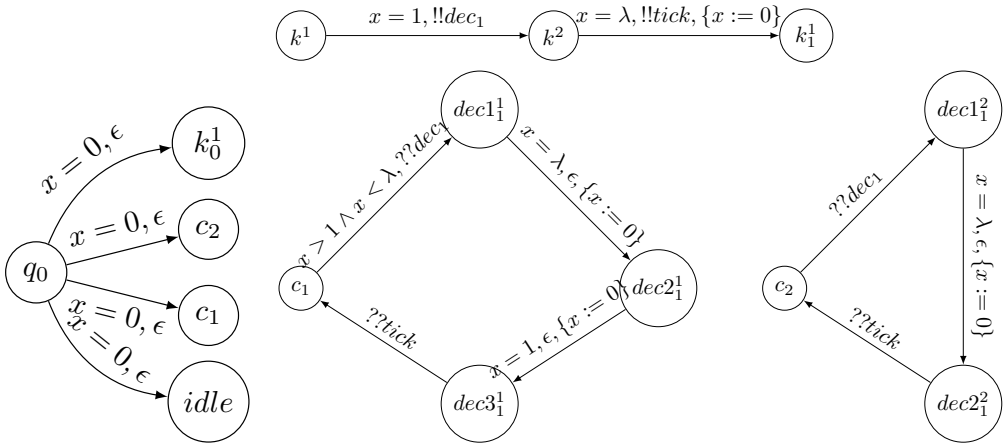
Given a valuation p of the parameter, we say that a configuration γ of the network encodes a configuration (k, v_1, v_2) of the two-counter machine if for all i, $\gamma[i] = (q, x)$ either $x > \lambda$ or $q \in \{c_1, c_2, k^1, idle\}$. Moreover all process with a clock lower than p and not in state $idle$ must agree on the clock valuation if they have the same state. Finally, if $\gamma[i] = (k^1, z)$ then for all i' such that $\gamma[i'] = (c_1, y)$ we have $v_1 = y - z$ and similarly for v_2.

Given an execution ρ, and a time t we denote $\rho_{T=t}$ the configuration obtained when considering ρ at global time t. Notice that $\rho_{T=t}$ may not be a configuration that appear in ρ since it can be a configuration obtain during the elapsing of time in a transition.

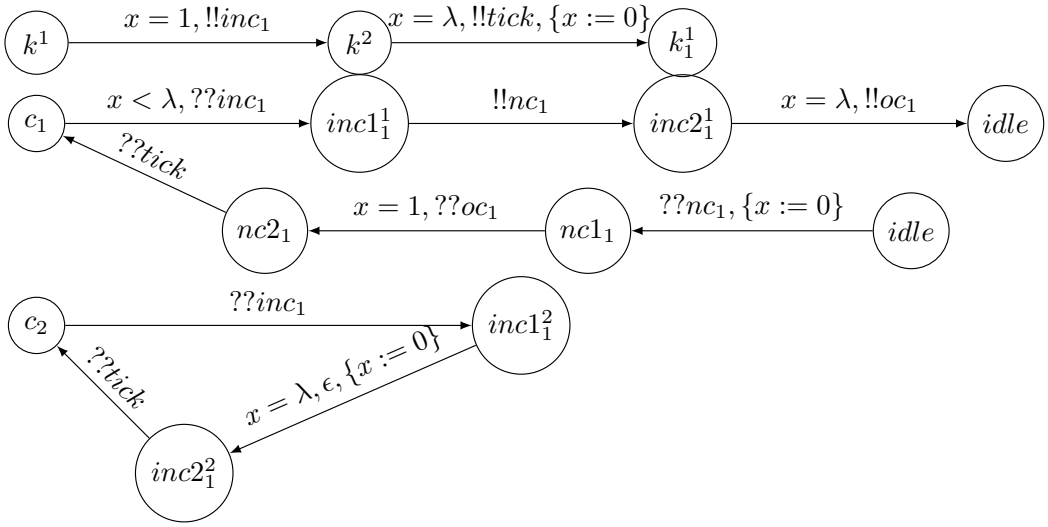
We will prove that for any execution ρ either $\rho_{T=k*p(\lambda)+1/2}$ is not defined (the execution time never reach $k * p(\lambda) + 1/2$) or $\rho_{T=k*p(\lambda)+1/2}$ encode $\mathbf{s}_k$.

We start by some remarks on the shape of possible executions

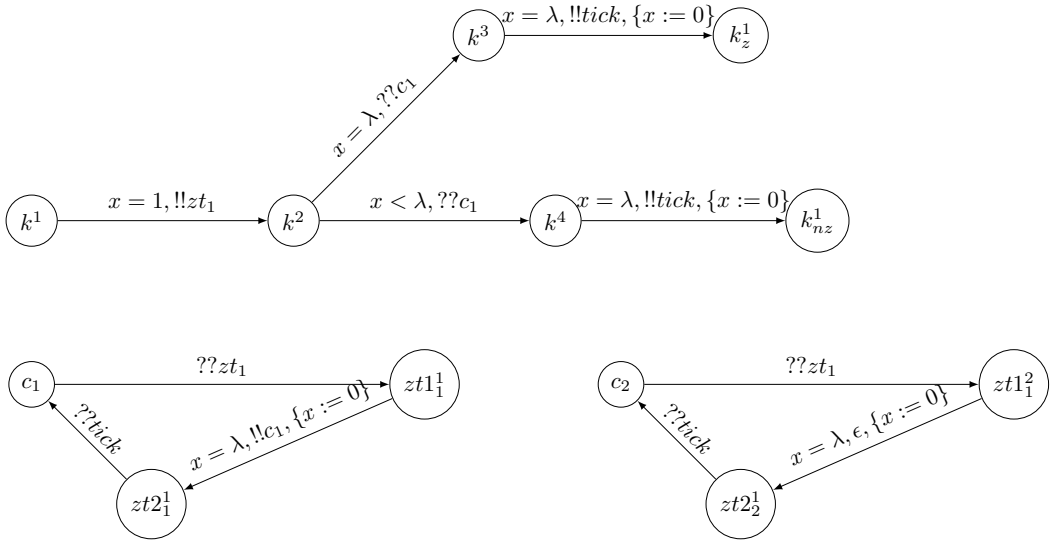
1. If two processes are in the controller part, then their clocks are equal modulo p. Indeed in the controller part the clock is reset only when it reaches $p(\lambda)$.



(a) Initialization **(b)** decrement: $\mathbf{k} : decr\ c_1\ goto\ \mathbf{k}_1$



(c) Increment: $\mathbf{k} : incr\ c_1\ goto\ \mathbf{k}_1$



(d) Test to zero: $\mathbf{k} : if\ c_1 = 0\ goto\ \mathbf{k}_z\ else\ goto\ \mathbf{k}_{nz}$

Figure 3 Representation of the construction

2. It follows that, by definition of the protocol, the message *tick* is sent only at time unit multiple of $p(\lambda)$.
3. Moreover the instruction messages (inc_i, dec_i, zt_i) are only sent at global time unit multiple of the form $k * p(\lambda) + 1$
4. Consider a process in state c_i with clock value lower than p. Assume that the global time is of the form $k * p(\lambda) + 1$. If this process does not receive an instruction message without delay, it will not be able to receive any before time $(k + 1) * p(\lambda) + 1$, thus it cannot take any transition before $(k + 1) * p(\lambda) + 1$. Note that at this time its clock will be greater than $p(\lambda)$ thus the guard prevent it to take any transition for the rest of the execution.
5. With the same idea if the process is in an intermediary state $nc2_i, dec2_i^j, dec3_i^i, inc2_i^j, zt2_i^i, zt2_i^j$ and does not receive a *tick* message at time $k*p(\lambda)$ we are certain that at time $(k+1)*p(\lambda)$ its clock will be above $p(\lambda)$ and it will thus be stuck forever.
6. Similarly if a process is in state $dec1_i^j, dec1_i^i, dec2_i^i, inc1_i^j, k^2$ and does not reset the clock when it is possible it will be stuck forever.
7. If an increment is requested by the controller part but the counter value is already equal to $p(\lambda)-1$ i. e., the clock value of the counter process is equal to $p(\lambda)$ the process are sent to an error state and thus for the following of the execution there wont be any processes in the counter part.
8. Similarly if an increment is requested while no processes are left in the idle state the execution gets stuck in the next zero test.

In other words if a process does not behave correctly its clock will increase over $p(\lambda)$ and the process will be stuck forever.

▶ **Example 18.** Before going further we first give some example of the behaviour of the network.

Successful decrement $\mathbf{k}$: *decr* c_1 *goto* $\mathbf{k}_1$ with $v_2 \geq v_1$ and $v_2 + 1 \leq \lambda$ (those assumption only matter for the order of the transitions).

$$\begin{pmatrix} k^1, 0 \\ c_1, v_1 \\ c_2, v_2 \end{pmatrix} \xrightarrow{1,1,!!dec_1,\{2,3\}} \begin{pmatrix} k^2, 1 \\ dec1_1^1, v_1 + 1 \\ dec1_1^2, v_2 + 1 \end{pmatrix} \xrightarrow{\lambda-(v_2+1),3,\epsilon,\emptyset} \begin{pmatrix} k^2, \lambda - v_2 \\ dec1_1^1, v_1 + \lambda - v_2 \\ dec2_1^2, 0 \end{pmatrix}$$

$$\xrightarrow{v_2-v_1,2,\epsilon,\emptyset} \begin{pmatrix} k^2, \lambda - v_1 \\ dec2_1^1, 0 \\ dec2_1^2, v_2 - v_1 \end{pmatrix} \xrightarrow{1,2,\epsilon,\emptyset} \begin{pmatrix} k^2, \lambda - v_1 + 1 \\ dec2_1^1, 0 \\ dec2_1^2, v_2 - v_1 + 1 \end{pmatrix} \xrightarrow{v_1-1,1,!!tick,\{2,3\}} \begin{pmatrix} k_1^1, 0 \\ c_1, v_1 - 1 \\ c_2, v_2 \end{pmatrix}$$

Failed decrement $\mathbf{k}$: *decr* c_1 *goto* $\mathbf{k}_1$ with $v_2 \geq v_1$ and $v_2 + 1 \leq \lambda$ (those assumption only matter for the order of the transitions).

$$\begin{pmatrix} k^1, 0 \\ c_1, 0 \\ c_2, v_2 \end{pmatrix} \xrightarrow{1,1,!!dec_1,\{2,3\}} \begin{pmatrix} k^2, 1 \\ c1_1, 1 \\ dec1_1^2, v_2 + 1 \end{pmatrix} \xrightarrow{\lambda-(v_2+1),3,\epsilon,\emptyset} \begin{pmatrix} k^2, \lambda - v_2 \\ c_1, \lambda - v_2 \\ dec2_1^2, 0 \end{pmatrix}$$

Notice that for the rest of the execution the process 2 will be stuck in c_1 unable to perform any action, nor receive any message.

Successful increment $\mathbf{k}$: *incr* c_1 *goto* $\mathbf{k}_1$ with $v_1 < \lambda - 1$ to ensure that the reception can be taken, and $v_2 \geq v_1$ and $v_2 + 1 \leq \lambda$ (those assumption only matter for the order of the

transitions).

$$\begin{pmatrix} k^1,0 \\ c_1,v_1 \\ c_2,v_2 \\ idle,w \end{pmatrix} \xrightarrow{1,1,!!inc_1,\{2,3\}} \begin{pmatrix} k^2,1 \\ inc1_1^1,v_1+1 \\ inc1_1^2,v_2+1 \\ idle,w' \end{pmatrix} \xrightarrow{\lambda-(v_2+1),3,\epsilon,\emptyset} \begin{pmatrix} k^2,\lambda-v_2 \\ inc1_1^1,v_1+\lambda-v_2 \\ inc2_1^2,0 \\ idle,w'' \end{pmatrix} \xrightarrow{v_2-v_1-1,2,!!nc_1,\{4\}}$$

$$\begin{pmatrix} k^2,\lambda-v_1-1 \\ inc2_1^1,\lambda-1 \\ inc2_1^2,v_2-v_1-1 \\ nc1_1,0 \end{pmatrix} \xrightarrow{1,2,!!oc_1,\{4\}} \begin{pmatrix} k^2,\lambda-v_1 \\ idle,\lambda \\ inc2_1^2,v_2-v_1 \\ nc2_1,1 \end{pmatrix} \xrightarrow{v_1,1,!!tick,\{3,4\}} \begin{pmatrix} k_1^1,0 \\ idle,\lambda+v_1 \\ c_2,v_2 \\ c_1,v_1+1 \end{pmatrix}$$

Failed increment $\mathbf{k}$: *incr* c_1 *goto* $\mathbf{k}_1$ (here there are no process in c_2 for simplicity)

$$\begin{pmatrix} k^1,0 \\ c_1,v_1 \\ idle,w \end{pmatrix} \xrightarrow{1,1,!!inc_1,\{2\}} \begin{pmatrix} k^2,1 \\ inc1_1^1,v_1+1 \\ idle,w' \end{pmatrix} \xrightarrow{\lambda-v_1-4,2,!!nc_1,\{4\}}$$

$$\begin{pmatrix} k^2,\lambda-v_1-3 \\ inc2_1^1,\lambda-3 \\ nc1_1,0 \end{pmatrix} \xrightarrow{3,2,!!oc_1,\{4\}} \begin{pmatrix} k^2,\lambda-v_1 \\ idle,\lambda \\ nc1_1,3 \end{pmatrix} \xrightarrow{v_1,1,!!tick,\{3,4\}} \begin{pmatrix} k_1^1,0 \\ idle,\lambda+v_1 \\ nc1_1,v_1+3 \end{pmatrix}$$

Notice that for the rest of the execution there will be no processes encoding c_1 thus no zero test can be achieved.

We show by induction on k that either $\rho_{T=(k+1)*p(\lambda)+1/2}$ is not defined or $\rho_{T=k*p(\lambda)+1/2}$ encode $\mathbf{s}_k$.

The case $k=0$ is direct. By definition it is easy to see that $\rho_{T=1/2}$ encodes γ_0.

Assume that the property holds for k. Let ρ be an execution such that $\rho_{T=k*p(\lambda)+1/2}$ encode $\mathbf{s}_k$. By the above remarks we have seen that if the network does not behave in the correct way it will get stuck before the next p time unit thus $\rho_{T=(k+2)*p(\lambda)+1/2}$. The only thing left to show to show that the reduction is correct is that the clock are reset at the right time to correctly model increment and decrement and that zero test are correct. For the latter, it is easy to see that by construction the controller part goes to the k_z instruction if and only if its clock is equal to the counter clock hence the counter is equal to 0, otherwise it moves to k_{nz}. For the former, the clocks evolve as in [19]. The only difference is for the increment where we need to introduce a new process used to guess when the clock value of the counter is equal to $p(\lambda)-1$.

We thus obtain that if the controller part can reach k_{acc} then since the execution correctly encodes the run, the run must terminate. Conversely if the run is infinite, for any N and any p, any execution will either be infinite (and correct) thus never reach k_{acc}, or eventually get stuck either because of an error in message, or because the counter clock is equal to 1 during an increment, or because there will not be enough processes in the idle state.

This concludes the proof that ∃-EF-emptiness is undecidable for 1-clock PTBP in the reconfigurable semantics.

For ∃-EF-universality, notice that the error state *error* is reachable only if an increment is requested when the counter value is equal to $p(\lambda)-1$). Thus if the error state is reached for all parameter valuations, this means that the run is unbounded. Conversely if the run is unbounded for all parameter valuations, at some point the counter value is equal to $p(\lambda)-1$) during an increment and thus the error state is reachable. To conclude on the undecidability of ∃-EF-universality, we just have to recall that we consider rational valuations for the parameters, but in this proof we only used integer valuations. This does not harm

the proof of undecidability since we can modify the gadget given Fig. 2 by replacing the state not integer by *error*. This modification ensures that *error* is reachable for any non integer valuation and the above argument that it is reachable for all integer valuations if and only if the two-counter machine is unbounded.

◀