






# Symbolic Monitoring Against Specifications Parametric in Time and Data

Masaki Waga<sup>1,2,3</sup> , Étienne André<sup>1,4,5</sup> ,  
and Ichiro Hasuo<sup>1,2</sup> 



<sup>1</sup> National Institute of Informatics, Tokyo, Japan  
mwaga@nii.ac.jp

<sup>2</sup> SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan

<sup>3</sup> JSPS Research Fellow, Tokyo, Japan

<sup>4</sup> Université Paris 13, LIPN, CNRS, UMR 7030, 93430 Villetaneuse, France

<sup>5</sup> JFLI, CNRS, Tokyo, Japan

**Abstract.** Monitoring consists in deciding whether a log meets a given specification. In this work, we propose an automata-based formalism to monitor logs in the form of actions associated with time stamps and arbitrarily data values over infinite domains. Our formalism uses both timing parameters and data parameters, and is able to output answers symbolic in these parameters and in the log segments where the property is satisfied or violated. We implemented our approach in an ad-hoc prototype SYMON, and experiments show that its high expressive power still allows for efficient online monitoring.

## 1 Introduction

Monitoring consists in checking whether a sequence of data (a log or a signal) satisfies or violates a specification expressed using some formalism. Offline monitoring consists in performing this analysis after the system execution, as the technique has access to the entire log in order to decide whether the specification is violated. In contrast, online monitoring can make a decision earlier, ideally as soon as a witness of the violation of the specification is encountered.

Using existing formalisms (e.g., the metric first order temporal logic [14]), one can check whether a given bank customer withdraws more than 1,000 € every week. With formalisms extended with data, one may even *identify* such customers. Or, using an extension of the signal temporal logic (STL) [18], one can ask: “is that true that the value of variable  $x$  is always copied to  $y$  exactly 4 time units later?” However, questions relating time and data using parameters become

---

This work is partially supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), by JSPS Grants-in-Aid No. 15KT0012 & 18J22498 and by the ANR national research program PACS (ANR-14-CE28-0002).

© The Author(s) 2019

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 520–539, 2019.

[https://doi.org/10.1007/978-3-030-25540-4\\_30](https://doi.org/10.1007/978-3-030-25540-4_30)

much harder (or even impossible) to express using existing formalisms: “what are the users and time frames during which a user withdraws more than half of the total bank withdrawals within seven days?” And even, can we *synthesize* the durations (not necessarily 7 days) for which this specification holds? Or “what is the set of variables for which there exists a duration within which their value is always copied to another variable?” In addition, detecting periodic behaviors without knowing the period can be hard to achieve using existing formalisms.

In this work, we address the challenging problem to monitor logs enriched with both timing information and (infinite domain) data. In addition, we significantly push the existing limits of expressiveness so as to allow for a further level of abstraction using *parameters*: our specification can be both parametric in the *time* and in the *data*. The answer to this symbolic monitoring is richer than a pure Boolean answer, as it *synthesizes* the values of both time and data parameters for which the specification holds. This allows us notably to detect periodic behaviors without knowing the period while being symbolic in terms of data. For example, we can *synthesize variable names* (data) and *delays* for which variables will have their value copied to another data within the aforementioned delay. In addition, we show that we can detect the log *segments* (start and end date) for which a specification holds.

*Example 1.* Consider a system updating three variables **a**, **b** and **c** (i.e., strings) to values (rationals). An example of log is given in Fig. 1a. Although our work is event-based, we can give a graphical representation similar to that of signals in Fig. 1b. Consider the following property: “for any variable **px**, whenever an update of that variable occurs, then within strictly less than **tp** time units, the value of variable **b** must be equal to that update”. The *variable parameter* **px** is compared with string values and the *timing parameter* **tp** is used in the timing constraints. We are interested in checking for which values of **px** and **tp** this property is violated. This can be seen as a synthesis problem in both the variable and timing parameters. For example, **px** = **c** and **tp** = 1.5 is a violation of the specification, as the update of **c** to 2 at time 4 is not propagated to **b** within 1.5 time unit. Our algorithm outputs such violation by a constraint e.g., **px** = **c**  $\wedge$  **tp**  $\leq$  2. In contrast, the value of any signal at any time is always such that either **b** is equal to that signal, or the value of **b** will be equal to that value within at most 2 time units. Thus, the specification holds for any valuation of the variable parameter **px**, provided **tp** > 2.

We propose an automata-based approach to perform monitoring parametric in both time and data. We implement our work in a prototype SYMON and perform experiments showing that, while our formalism allows for high expressiveness, it is also tractable even for online monitoring.

We believe our framework balances expressiveness and monitoring performance well: (i) Regarding expressiveness, comparison with the existing work is summarized in Table 1 (see Sect. 2 for further details). (ii) Our monitoring is *complete*, in the sense that it returns a symbolic constraint characterizing *all* the parameter valuations that match a given specification. (iii) We also achieve



## 2 Related Works

*Robustness and Monitoring.* Robust (or quantitative) monitoring extends the binary question whether a log satisfies a specification by asking “by how much” the specification is satisfied. The quantification of the distance between a signal and a signal temporal logic (STL) specification has been addressed in, e.g., [20–23, 25, 27] (or in a slightly different setting in [5]). The distance can be understood in terms of space (“signals”) or time. In [6], the distance also copes for reordering of events. In [10], the *robust pattern matching problem* is considered over signal regular expressions, by quantifying the distance between the signal regular expression specification and the *segments* of the signal. For piecewise-constant and piecewise-linear signals, the problem can be effectively solved using a finite union of convex polyhedra. While our framework does not fit in robust monitoring, we can simulate both the robustness w.r.t. time (using timing parameters) and w.r.t. data, e.g., signal values (using data parameters).

*Monitoring with Data.* The tool MARQ [30] performs monitoring using Quantified Event Automata (QEA) [12]. This approach and ours share the automata-based framework, the ability to express some first-order properties using “events containing data” (which we encode using local variables associated with actions), and data may be quantified. However, [30] does not seem to natively support specification parametric in time; in addition, [30] does not perform complete (“symbolic”) parameters synthesis, but outputs the violating entries of the log.

The metric first order temporal logic (MFOTL) allows for a high expressiveness by allowing universal and existential quantification over data—which can be seen as a way to express parameters. A monitoring algorithm is presented for a safety fragment of MFOTL in [14]. Aggregation operators are added in [13], allowing to compute *sums* or *maximums* over data. A fragment of this logics is implemented in MONPOLY [15]. While these works are highly expressive, they do not natively consider timing parameters; in addition, MONPOLY does not output symbolic answers, i. e., symbolic conditions on the parameters to ensure validity of the formula.

In [26], binary decision diagrams (BDDs) are used to symbolically represent the observed data in QTL. This can be seen as monitoring data against a parametric specification, with a symbolic internal encoding. However, their implementation DEJAVU only outputs *concrete* answers. In contrast, we are able to provide symbolic answers (both in timing and data parameters), e.g., in the form of union of polyhedra for rationals, and unions of string constraints using equalities (=) and inequalities ( $\neq$ ).

*Freeze Operator.* In [18], STL is extended with a freeze operator that can “remember” the value of a signal, to compare it to a later value of the same signal. This logic STL\* can express properties such as “In the initial 10s, x copies the values of y within a delay of 4s”:  $\mathbf{G}_{[0,10]} * (\mathbf{G}_{[0,4]} y^* = x)$ . While the setting is somehow different (STL\* operates over signals while we operate over timed data words), the requirements such as the one above can easily be encoded

in our framework. In addition, we are able to *synthesize* the delay within which the values are always copied, as in Example 1. In contrast, it is not possible to determine using STL\* which variables and which delays violate the specification.

*Monitoring with Parameters.* In [7], a log in the form of a dense-time real-valued signal is tested against a parameterized extension of STL, where parameters can be used to model uncertainty both in signal values and in timing values. The output comes in the form of a subset of the parameters space for which the formula holds on the log. In [9], the focus is only on signal parameters, with an improved efficiency by reusing techniques from the *robust* monitoring. Whereas [7,9] fit in the framework of signals and temporal logics while we fit in words and automata, our work shares similarities with [7,9] in the sense that we can express data parameters; in addition, [9] is able as in our work to exhibit the segment of the log associated with the parameters valuations for which the specification holds. A main difference however is that we can use memory and aggregation, thanks to arithmetic on variables.

In [24], the problem of *inferring* temporal logic formulae with constraints that hold in a given numerical data time series is addressed.

*Timed Pattern Matching.* A recent line of work is that of timed pattern matching, that takes as input a log and a specification, and decides *where* in the log the specification is satisfied or violated. On the one hand, a line of works considers signals, with specifications either in the form of timed regular expressions [11,31–33], or a temporal logic [34]. On the other hand, a line of works considers timed words, with specifications in the form of timed automata [4,36]. We will see that our work can also encode parametric timed pattern matching. Therefore, our work can be seen as a two-dimensional extension of both lines of works: first, we add timing parameters ([4] also considers similar timing parameters) and, second, we add data—themselves extended with parameters. That is, coming back to Example 1, [31–33,36] could only infer the segments of the log for which the property is violated for a given (fixed) variable and a given (fixed) timing parameter; while [4] could infer both the segments of the log and the timing parameter valuations, but not which variable violates the specification.

*Summary.* We compare related works in Table 1. “Timing parameters” denote the ability to synthesize unknown constants used in timing constraints (e.g., modalities intervals, or clock constraints). “?” denotes works not natively supporting this, although it might be encoded. The term “Data” refers to the ability to manage logs over infinite domains (apart from timestamps). For example, the log in Fig. 1a features, beyond timestamps, both string (variable name) and rationals (value). Also, works based on real-valued signals are naturally able to manage (at least one type of) data. “Parametric data” refer to the ability to express formulas where data (including signal values) are compared to (quantified or unquantified) variables or unknown parameters; for example, in the log in Fig. 1a, an example of property parametric in data is to synthesize the parameters for which the difference of values between two consecutive updates of

variable  $\mathbf{px}$  is always below  $\mathbf{pv}$ , where  $\mathbf{px}$  is a string parameter and  $\mathbf{pv}$  a rational-valued parameter. “Memory” is the ability to remember *past* data; this can be achieved using e.g., the freeze operator of STL\*, or variables (e.g., in [14, 26, 30]). “Aggregation” is the ability to aggregate data using operators such as sum or maximum; this allows to express properties such as “A user must not withdraw more than \$10,000 within a 31 day period” [13]. This can be supported using dedicated aggregation operators [13] or using variables ([30], and our work). “Complete parameter identification” denotes the *synthesis* of the set of parameters that satisfy or violate the property. Here, “N/A” denotes the absence of parameter [18], or when parameters are used in a way (existentially or universally quantified) such as the identification is not explicit (instead, the position of the log where the property is violated is returned [26]). In contrast, we return in a *symbolic* manner (as in [4, 7]) the exact set of (data and timing) parameters for which a property is satisfied. “ $\checkmark/\times$ ” denotes “yes” in the theory paper, but not in the tool.

### 3 Preliminaries

**Clocks, Timing Parameters and Timed Guards.** We assume a set  $\mathbb{C} = \{c_1, \dots, c_H\}$  of *clocks*, i.e., real-valued variables that evolve at the same rate. A *clock valuation* is  $\nu : \mathbb{C} \rightarrow \mathbb{R}_{\geq 0}$ . We write  $\mathbf{0}$  for the clock valuation assigning 0 to all clocks. Given  $d \in \mathbb{R}_{\geq 0}$ ,  $\nu + d$  is s.t.  $(\nu + d)(c) = \nu(c) + d$ , for all  $c \in \mathbb{C}$ . Given  $R \subseteq \mathbb{C}$ , we define the *reset* of a valuation  $\nu$ , denoted by  $[\nu]_R$ , as follows:  $[\nu]_R(c) = 0$  if  $c \in R$ , and  $[\nu]_R(c) = \nu(c)$  otherwise.

We assume a set  $\mathbb{TP} = \{\mathbf{tp}_1, \dots, \mathbf{tp}_J\}$  of *timing parameters*. A *timing parameter valuation* is  $\gamma : \mathbb{TP} \rightarrow \mathbb{Q}_+$ . We assume  $\bowtie \in \{<, \leq, =, \geq, >\}$ . A *timed guard*  $tg$  is a constraint over  $\mathbb{C} \cup \mathbb{TP}$  defined by a conjunction of inequalities of the form  $c \bowtie d$ , or  $c \bowtie \mathbf{tp}$  with  $d \in \mathbb{N}$  and  $\mathbf{tp} \in \mathbb{TP}$ . Given  $tg$ , we write  $\nu \models \gamma(tg)$  if the expression obtained by replacing each  $c$  with  $\nu(c)$  and each  $\mathbf{tp}$  with  $\gamma(\mathbf{tp})$  in  $tg$  evaluates to true.

**Variables, Data Parameters and Data Guards.** For sake of simplicity, we assume a *single* infinite domain  $\mathbb{D}$  for data. The formalism defined in Sect. 4 can be extended in a straightforward manner to different domains for different variables (and our implementation does allow for different types). The case of *finite* data domain is immediate too. We define this formalism in an *abstract* manner, so as to allow a sort of parameterized domain.

We assume a set  $\mathbb{V} = \{v_1, \dots, v_M\}$  of *variables* valued over  $\mathbb{D}$ . These variables are internal variables, that allow an high expressive power in our framework, as they can be compared or updated to other variables or parameters. We also assume a set  $\mathbb{LV} = \{lv_1, \dots, lv_O\}$  of *local variables* valued over  $\mathbb{D}$ . These variables will only be used locally along a transition in the “argument” of the action (e.g.,  $x$  and  $v$  in  $\mathbf{update}(x, v)$ ), and in the associated guard and (right-hand part of) updates. We assume a set  $\mathbb{VP} = \{\mathbf{vp}_1, \dots, \mathbf{vp}_N\}$  of *data parameters*, i.e., unknown variable constants.

A *data type*  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$  is made of (i) an infinite domain  $\mathbb{D}$ , (ii) a set of admissible Boolean expressions  $\mathcal{DE}$  (that may rely on  $\mathbb{V}$ ,  $\mathbb{LV}$  and  $\mathbb{VP}$ ), which will define the type of guards over variables in our subsequent automata, and (iii) a domain for updates  $\mathcal{DU}$  (that may rely on  $\mathbb{V}$ ,  $\mathbb{LV}$  and  $\mathbb{VP}$ ), which will define the type of updates of variables in our subsequent automata.

*Example 2.* As a first example, let us define the data type for rationals. We have  $\mathbb{D} = \mathbb{Q}$ . Let us define Boolean expressions. A *rational comparison* is a constraint over  $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$  defined by a conjunction of inequalities of the form  $v \bowtie d$ ,  $v \bowtie v'$ , or  $v \bowtie \mathbf{vp}$  with  $v, v' \in \mathbb{V} \cup \mathbb{LV}$ ,  $d \in \mathbb{Q}$  and  $\mathbf{vp} \in \mathbb{VP}$ .  $\mathcal{DE}$  is the set of all rational comparisons over  $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ . Let us then define updates. First, a linear arithmetic expression over  $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$  is  $\sum_i \alpha_i v_i + \beta$ , where  $v_i \in \mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$  and  $\alpha_i, \beta \in \mathbb{Q}$ . Let  $\mathcal{LA}(\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP})$  denote the set of arithmetic expressions over  $\mathbb{V}$ ,  $\mathbb{LV}$  and  $\mathbb{VP}$ . We then have  $\mathcal{DU} = \mathcal{LA}(\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP})$ .

As a second example, let us define the data type for strings. We have  $\mathbb{D} = \mathbb{S}$ , where  $\mathbb{S}$  denotes the set of all strings. A *string comparison* is a constraint over  $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$  defined by a conjunction of comparisons of the form  $v \approx s$ ,  $v \approx v'$ , or  $v \approx \mathbf{vp}$  with  $v, v' \in \mathbb{V} \cup \mathbb{LV}$ ,  $s \in \mathbb{S}$ ,  $\mathbf{vp} \in \mathbb{VP}$  and  $\approx \in \{=, \neq\}$ .  $\mathcal{DE}$  is the set of all string comparisons over  $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ .  $\mathcal{DU} = \mathbb{V} \cup \mathbb{LV} \cup \mathbb{S}$ , i.e., a string variable can be assigned another string variable, or a concrete string.

A *variable valuation* is  $\mu : \mathbb{V} \rightarrow \mathbb{D}$ . A *local variable valuation* is a partial function  $\eta : \mathbb{LV} \rightarrow \mathbb{D}$ . A *data parameter valuation* is  $\zeta : \mathbb{VP} \rightarrow \mathbb{D}$ . Given a data guard  $dg \in \mathcal{DE}$ , a variable valuation  $\mu$ , a local variable valuation  $\eta$  defined for the local variables in  $dg$ , and a data parameter valuation  $\zeta$ , we write  $(\mu, \eta) \models \zeta(dg)$  if the expression obtained by replacing within  $dg$  all occurrences of each data parameter  $\mathbf{vp}_i$  by  $\zeta(\mathbf{vp}_i)$  and all occurrences of each variable  $v_j$  (resp. local variable  $lv_k$ ) with its concrete valuation  $\mu(v_j)$  (resp.  $\eta(lv_k)$ ) evaluates to true.

A parametric data update is a partial function  $\text{PDU} : \mathbb{V} \rightarrow \mathcal{DU}$ . That is, we can assign to a variable an expression over data parameters and other variables, according to the data type. Given a parametric data update  $\text{PDU}$ , a variable valuation  $\mu$ , a local variable valuation  $\eta$  (defined for all local variables appearing in  $\text{PDU}$ ), and a data parameter valuation  $\zeta$ , we define  $[\mu]_{\eta(\zeta(\text{PDU}))} : \mathbb{V} \rightarrow \mathbb{D}$  as:

$$[\mu]_{\eta(\zeta(\text{PDU}))}(v) = \begin{cases} \mu(v) & \text{if } \text{PDU}(v) \text{ is undefined} \\ \eta(\mu(\zeta(\text{PDU}(v)))) & \text{otherwise} \end{cases}$$

where  $\eta(\mu(\zeta(\text{PDU}(v))))$  denotes the replacement within the update expression  $\text{PDU}(v)$  of all occurrences of each data parameter  $\mathbf{vp}_i$  by  $\zeta(\mathbf{vp}_i)$ , and all occur-

**Table 2.** Variables, parameters and valuations used in guards

	Timed guards		Data guards		
	Clock	Timing parameter	(Data) variable	Local variable	Data parameter
Variable	$c$	$\mathbf{tp}$	$v$	$lv$	$\mathbf{vp}$
Valuation	$\nu$	$\gamma$	$\mu$	$\eta$	$\zeta$

rences of each variable  $v_j$  (resp. local variable  $lv_k$ ) with its concrete valuation  $\mu(v_j)$  (resp.  $\eta(lv_k)$ ). Observe that this replacement gives a value in  $\mathbb{D}$ , therefore the result of  $[\mu]_{\eta(\zeta(\text{PDU}))}$  is indeed a data parameter valuation  $\mathbb{V} \rightarrow \mathbb{D}$ . That is,  $[\mu]_{\eta(\zeta(\text{PDU}))}$  computes the new (non-parametric) variable valuation obtained after applying to  $\mu$  the partial function PDU valuated with  $\zeta$ .

*Example 3.* Consider the data type for rationals, the variables set  $\{v_1, v_2\}$ , the local variables set  $\{lv_1, lv_2\}$  and the parameters set  $\{vp_1\}$ . Let  $\mu$  be the variable valuation such that  $\mu(v_1) = 1$  and  $\mu(v_2) = 2$ , and  $\eta$  be the local variable valuation such that  $\eta(lv_1) = 2$  and  $\eta(lv_2)$  is not defined. Let  $\zeta$  be the data parameter valuation such that  $\zeta(vp_1) = 1$ . Consider the parametric data update function PDU such that  $\text{PDU}(v_1) = 2 \times v_1 + v_2 - lv_1 + vp_1$ , and  $\text{PDU}(v_2)$  is undefined. Then the result of  $[\mu]_{\eta(\zeta(\text{PDU}))}$  is  $\mu'$  such that  $\mu'(v_1) = 2 \times \mu(v_1) + \mu(v_2) - \eta(lv_1) + \zeta(vp_1) = 3$  and  $\mu'(v_2) = 2$ .

## 4 Parametric Timed Data Automata

We introduce here Parametric timed data automata (PTDAs). They can be seen as an extension of parametric timed automata [2] (that extend timed automata [1] with parameters in place of integer constants) with unbounded data variables and parametric variables. PTDAs can also be seen as an extension of some extensions of timed automata with data (see e.g., [16, 19, 29]), that we again extend with both data parameters and timing parameters. Or as an extension of quantified event automata [12] with explicit time representation using clocks, and further augmented with timing parameters. PTDAs feature both timed guards and data guards; we summarize the various variables and parameters types together with their notations in Table 2.

We will associate local variables with actions (which can be seen as *predicates*). Let  $\text{Dom} : \Sigma \rightarrow 2^{\text{LV}}$  denote the set of local variables associated with each action. Let  $\text{Var}(dg)$  (resp.  $\text{Var}(\text{PDU})$ ) denote the set of variables occurring in  $dg$  (resp. PDU).

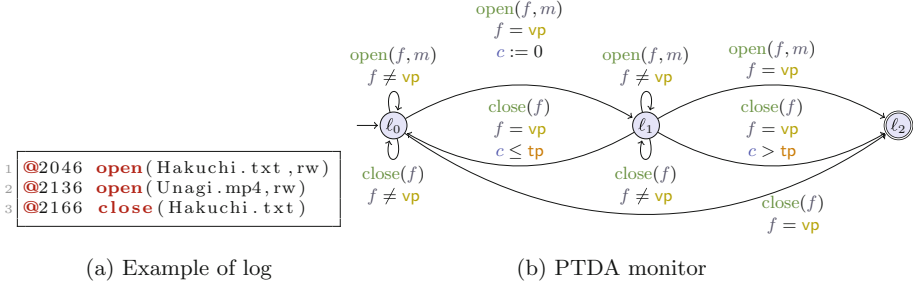
**Definition 1 (PTDA).** *Given a data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ , a parametric timed data automaton (PTDA)  $\mathcal{A}$  over this data type is a tuple  $\mathcal{A} = (\Sigma, L, \ell_0, F, \mathbb{C}, \text{TP}, \mathbb{V}, \text{LV}, \mu_0, \mathbb{VP}, E)$ , where:*

1.  $\Sigma$  is a finite set of actions,
2.  $L$  is a finite set of locations,  $\ell_0 \in L$  is the initial location,
3.  $F \subseteq L$  is the set of accepting locations,
4.  $\mathbb{C}$  is a finite set of clocks,
5.  $\text{TP}$  is a finite set of timing parameters,
6.  $\mathbb{V}$  (resp.  $\text{LV}$ ) is a finite set of variables (resp. local variables) over  $\mathbb{D}$ ,
7.  $\mu_0$  is the initial variable valuation,
8.  $\mathbb{VP}$  is a finite set of data parameters,



9.  $E$  is a finite set of edges  $e = (\ell, tg, dg, a, R, \text{PDU}, \ell')$  where (i)  $\ell, \ell' \in L$  are the source and target locations, (ii)  $tg$  is a timed guard, (iii)  $dg \in \mathcal{DE}$  is a data guard such as  $\text{Var}(dg) \cap \mathbb{LV} \subseteq \text{Dom}(a)$ , (iv)  $a \in \Sigma$ , (v)  $R \subseteq \mathbb{C}$  is a set of clocks to be reset, and (vi)  $\text{PDU} : \mathbb{V} \rightarrow \mathcal{DU}$  is the parametric data update function such that  $\text{Var}(\text{PDU}) \cap \mathbb{LV} \subseteq \text{Dom}(a)$ .

The domain conditions on  $dg$  and  $\text{PDU}$  ensure that the local variables used in the guard (resp. update) are only those in the action signature  $\text{Dom}(a)$ .



**Fig. 2.** Monitoring proper file opening and closing

*Example 4.* Consider the PTDA in Fig. 2b over the data type for strings. We have  $\mathbb{C} = \{c\}$ ,  $\text{TP} = \{\text{tp}\}$ ,  $\mathbb{V} = \emptyset$  and  $\mathbb{LV} = \{f, m\}$ .  $\text{Dom}(\text{open}) = \{f, m\}$  while  $\text{Dom}(\text{close}) = \{f\}$ .  $\ell_2$  is the only accepting location, modeling the violation of the specification.

This PTDA (freely inspired by a formula from [26] further extended with timing parameters) monitors the improper file opening and closing, i.e., a file already open should not be open again, and a file that is open should not be closed too late. The data parameter  $\text{vp}$  is used to *symbolically* monitor a given file name, i.e., we are interested in opening and closings of this file only, while other files are disregarded (specified using the self-loops in  $\ell_0$  and  $\ell_1$  with data guard  $f \neq \text{vp}$ ). Whenever  $f$  is opened (transition from  $\ell_0$  to  $\ell_1$ ), a clock  $c$  is reset. Then, in  $\ell_1$ , if  $f$  is closed within  $\text{tp}$  time units (timed guard “ $c \leq \text{tp}$ ”), then the system goes back to  $\ell_0$ . However, if instead  $f$  is opened again, this is an incorrect behavior and the system enters  $\ell_2$  via the upper transition. The same occurs if  $f$  is closed more than  $\text{tp}$  time units after opening.

Given a data parameter valuation  $\zeta$  and a timing parameter valuation  $\gamma$ , we denote by  $\gamma|\zeta(\mathcal{A})$  the resulting *timed data automaton (TDA)*, i.e., the non-parametric structure where all occurrences of a parameter  $\text{vp}_i$  (resp.  $\text{tp}_j$ ) have been replaced by  $\zeta(\text{vp}_i)$  (resp.  $\gamma(\text{tp}_j)$ ). Note that, if  $\mathbb{V} = \mathbb{LV} = \emptyset$ , then  $\mathcal{A}$  is a *parametric timed automaton* [2] and  $\gamma|\zeta(\mathcal{A})$  is a *timed automaton* [1].

We now equip our TDAs with a concrete semantics.

**Definition 2 (Semantics of a TDA).** Given a PTDA  $\mathcal{A} = (\Sigma, L, \ell_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$  over a data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ , a data parameter valuation  $\zeta$  and a timing parameter valuation  $\gamma$ , the semantics of  $\gamma|\zeta(\mathcal{A})$  is given by the timed transition system (TTS)  $(S, s_0, \rightarrow)$ , with

- $S = L \times \mathbb{D}^M \times \mathbb{R}_{\geq 0}^H$ ,  $s_0 = (\ell_0, \mu_0, \mathbf{0})$ ,
  - $\rightarrow$  consists of the discrete and (continuous) delay transition relations:
1. discrete transitions:  $(\ell, \mu, \nu) \xrightarrow{e, \eta} (\ell', \mu', \nu')$ , there exist  $e = (\ell, tg, dg, a, R, \text{PDU}, \ell') \in E$  and a local variable valuation  $\eta$  defined exactly for  $\text{Dom}(a)$ , such that  $\nu \models \gamma(tg)$ ,  $(\mu, \eta) \models \zeta(dg)$ ,  $\nu' = [\nu]_R$ , and  $\mu' = [\mu]_{\eta(\zeta(\text{PDU}))}$ .
  2. delay transitions:  $(\ell, \mu, \nu) \xrightarrow{d} (\ell, \mu, \nu + d)$ , with  $d \in \mathbb{R}_{\geq 0}$ .

Moreover we write  $((\ell, \mu, \nu), (e, \eta, d), (\ell', \mu', \nu')) \in \rightarrow$  for a combination of a delay and discrete transition if  $\exists \nu'' : (\ell, \mu, \nu) \xrightarrow{d} (\ell, \mu, \nu'') \xrightarrow{e, \eta} (\ell', \mu', \nu')$ .

Given a TDA  $\gamma|\zeta(\mathcal{A})$  with concrete semantics  $(S, s_0, \rightarrow)$ , we refer to the states of  $S$  as the *concrete states* of  $\gamma|\zeta(\mathcal{A})$ . A *run* of  $\gamma|\zeta(\mathcal{A})$  is an alternating sequence of concrete states of  $\gamma|\zeta(\mathcal{A})$  and triples of edges, local variable valuations and delays, starting from the initial state  $s_0$  of the form  $(\ell_0, \mu_0, \nu_0), (e_0, \eta, d_0), (\ell_1, \mu_1, \nu_1), \dots$  with  $i = 0, 1, \dots$ ,  $e_i \in E$ ,  $d_i \in \mathbb{R}_{\geq 0}$  and  $((\ell_i, \mu_i, \nu_i), (e_i, \eta_i, d_i), (\ell_{i+1}, \mu_{i+1}, \nu_{i+1})) \in \rightarrow$ . Given such a run, the associated *timed data word* is  $(a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots$ , where  $a_i$  is the action of edge  $e_{i-1}$ ,  $\eta_i$  is the local variable valuation associated with that transition, and  $\tau_i = \sum_{0 \leq j \leq i-1} d_j$ , for  $i = 1, 2, \dots$ . For a timed data word  $w$  and a concrete state  $(\ell, \mu, \nu)$  of  $\gamma|\zeta(\mathcal{A})$ , we write  $(\ell_0, \mu_0, \mathbf{0}) \xrightarrow{w} (\ell, \mu, \nu)$  in  $\gamma|\zeta(\mathcal{A})$  if  $w$  is associated with a run of  $\gamma|\zeta(\mathcal{A})$  of the form  $(\ell_0, \mu_0, \mathbf{0}), \dots, (\ell_n, \mu_n, \nu_n)$  with  $(\ell_n, \mu_n, \nu_n) = (\ell, \mu, \nu)$ . For a timed data word  $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots, (a_n, \tau_n, \eta_n)$ , we denote  $|w| = n$  and for any  $i \in \{1, 2, \dots, n\}$ , we denote  $w(1, i) = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots, (a_i, \tau_i, \eta_i)$ .

A finite run is *accepting* if its last state  $(\ell, \mu, \nu)$  is such that  $\ell \in F$ . The *language*  $\mathcal{L}(\gamma|\zeta(\mathcal{A}))$  is defined to be the set of timed data words associated with all accepting runs of  $\gamma|\zeta(\mathcal{A})$ .

*Example 5.* Consider the PTDA in Fig. 2b over the data type for strings. Let  $\gamma(\text{tp}) = 100$  and  $\zeta(\text{vp}) = \text{Hakuchi.txt}$ . An accepting run of the TDA  $\gamma|\zeta(\mathcal{A})$  is:  $(\ell_0, \emptyset, \nu_0), (e_0, \eta_0, 2046), (\ell_1, \emptyset, \nu_1), (e_1, \eta_1, 90), (\ell_1, \emptyset, \nu_2), (e_2, \eta_2, 30), (\ell_2, \emptyset, \nu_3)$ , where  $\emptyset$  denotes a variable valuation over an empty domain (recall that  $\mathbb{V} = \emptyset$  in Fig. 2b),  $\nu_0(c) = 0$ ,  $\nu_1(c) = 0$ ,  $\nu_2(c) = 90$ ,  $\nu_3(c) = 120$ ,  $e_0$  is the upper edge from  $\ell_0$  to  $\ell_1$ ,  $e_1$  is the self-loop above  $\ell_1$ ,  $e_2$  is the lower edge from  $\ell_1$  to  $\ell_2$ ,  $\eta_0(f) = \eta_2(f) = \text{Hakuchi.txt}$ ,  $\eta_1(f) = \text{Unagi.mp4}$ ,  $\eta_0(m) = \eta_1(m) = \text{rw}$ , and  $\eta_2(m)$  is undefined (because  $\text{Dom}(\text{close}) = \{f\}$ ).

The associated timed data word is  $(\text{open}, 2046, \eta_0), (\text{open}, 2136, \eta_1), (\text{close}, 2166, \eta_2)$ .

Since each action is associated with a set of local variables, given an ordering on this set, it is possible to see a given action and a variable valuation as a predicate: for example, assuming an ordering of  $\mathbb{LV}$  such as  $f$  precedes  $m$ , then **open**

with  $\eta_0$  can be represented as `open(Hakuchi.txt, rw)`. Using this convention, the log in Fig. 2a corresponds exactly to this timed data word.

## 5 Symbolic Monitoring Against PTDA Specifications

In symbolic monitoring, in addition to the (observable) actions in  $\Sigma$ , we employ *unobservable* actions denoted by  $\varepsilon$  and satisfying  $\text{Dom}(\varepsilon) = \emptyset$ . We write  $\Sigma_\varepsilon$  for  $\Sigma \sqcup \{\varepsilon\}$ . We let  $\eta_\varepsilon$  be the local variable valuation such that  $\eta_\varepsilon(lv)$  is undefined for any  $lv \in \mathbb{LV}$ . For a timed data word  $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots, (a_n, \tau_n, \eta_n)$  over  $\Sigma_\varepsilon$ , the projection  $w \downarrow_\Sigma$  is the timed data word over  $\Sigma$  obtained from  $w$  by removing any triple  $(a_i, \tau_i, \eta_i)$  where  $a_i = \varepsilon$ . An edge  $e = (\ell, tg, dg, a, R, \text{PDU}, \ell') \in E$  is *unobservable* if  $a = \varepsilon$ , and *observable* otherwise. The use of unobservable actions allows us to encode parametric timed pattern matching (see Sect. 5.3).

We make the following assumption on the PTDAs in symbolic monitoring.

**Assumption 1.** *The PTDA  $\mathcal{A}$  does not contain any loop of unobservable edges.*

### 5.1 Problem Definition

Roughly speaking, given a PTDA  $\mathcal{A}$  and a timed data word  $w$ , the symbolic monitoring problem asks for the set of pairs  $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\text{TP}} \times \mathbb{D}^{\text{VP}}$  satisfying  $w(1, i) \in \gamma|\zeta(\mathcal{A})$ , where  $w(1, i)$  is a prefix of  $w$ . Since  $\mathcal{A}$  also contains unobservable edges, we consider  $w'$  which is  $w$  augmented by unobservable actions.

#### Symbolic monitoring problem:

INPUT: a PTDA  $\mathcal{A}$  over a data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$  and actions  $\Sigma_\varepsilon$ , and a timed data word  $w$  over  $\Sigma$

PROBLEM: compute all the pairs  $(\gamma, \zeta)$  of timing and data parameter valuations such that there is a timed data word  $w'$  over  $\Sigma_\varepsilon$  and  $i \in \{1, 2, \dots, |w'|\}$  satisfying  $w' \downarrow_\Sigma = w$  and  $w'(1, i) \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))$ . That is, it requires the *validity domain*  $D(w, \mathcal{A}) = \{(\gamma, \zeta) \mid \exists w' : i \in \{1, 2, \dots, |w'|\}, w' \downarrow_\Sigma = w \text{ and } w'(1, i) \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))\}$ .

*Example 6.* Consider the PTDA  $\mathcal{A}$  and the timed data word  $w$  shown in Fig. 1. The validity domain  $D(w, \mathcal{A})$  is  $D(w, \mathcal{A}) = D_1 \cup D_2$ , where

$$D_1 = \{(\gamma, \zeta) \mid 0 \leq \gamma(\text{tp}) \leq 2, \zeta(\text{xp}) = c\} \text{ and } D_2 = \{(\gamma, \zeta) \mid 0 \leq \gamma(\text{tp}) \leq 1, \zeta(\text{xp}) = a\}.$$

For  $w' = w(1, 3) \cdot (\varepsilon, \eta_\varepsilon, 2.9)$ , we have  $w' \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))$  and  $w' \downarrow_\Sigma = w(1, 3)$ , where  $\gamma$  and  $\zeta$  are such that  $\gamma(\text{tp}) = 1.8$  and  $\zeta(\text{xp}) = c$ , and  $w(1, 3) \cdot (\varepsilon, \eta_\varepsilon, 2.9)$  denotes the juxtaposition.

For the data types in Example 2, the validity domain  $D(w, \mathcal{A})$  can be represented by a constraint of finite size because the length  $|w|$  of the timed data word is finite.

## 5.2 Online Algorithm

Our algorithm is *online* in the sense that it outputs  $(\gamma, \zeta) \in D(w, \mathcal{A})$  as soon as its membership is witnessed, even before reading the whole timed data word  $w$ .

Let  $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots, (a_n, \tau_n, \eta_n)$  and  $\mathcal{A}$  be the timed data word and PTDA given in symbolic monitoring, respectively. Intuitively, after reading  $(a_i, \tau_i, \eta_i)$ , our algorithm symbolically computes for all parameter valuations  $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$  the concrete states  $(\ell, \nu, \mu)$  satisfying  $(\ell_0, \mu_0, \mathbf{0}) \xrightarrow{w(1,i)} (\ell, \mu, \nu)$  in  $\gamma|\zeta(\mathcal{A})$ . Since  $\mathcal{A}$  has unobservable edges as well as observable edges, we have to add unobservable actions before or after observable actions in  $w$ . By  $\text{Conf}_i^o$ , we denote the configurations after reading  $(a_i, \tau_i, \eta_i)$  and no unobservable actions are appended after  $(a_i, \tau_i, \eta_i)$ . By  $\text{Conf}_i^u$ , we denote the configurations after reading  $(a_i, \tau_i, \eta_i)$  and at least one unobservable action is appended after  $(a_i, \tau_i, \eta_i)$ .

**Definition 3** ( $\text{Conf}_i^o, \text{Conf}_i^u$ ). For a PTDA  $\mathcal{A}$  over actions  $\Sigma_\varepsilon$ , a timed data word  $w$  over  $\Sigma$ , and  $i \in \{0, 1, \dots, |w|\}$  (resp.  $i \in \{-1, 0, \dots, |w|\}$ ),  $\text{Conf}_i^o$  (resp.  $\text{Conf}_i^u$ ) is the set of 5-tuples  $(\ell, \nu, \gamma, \mu, \zeta)$  such that there is a timed data word  $w'$  over  $\Sigma_\varepsilon$  satisfying the following: (i)  $(\ell_0, \mu_0, \mathbf{0}) \xrightarrow{w'} (\ell, \mu, \nu)$  in  $\gamma|\zeta(\mathcal{A})$ , (ii)  $w' \downarrow_\Sigma = w(1, i)$ , (iii) The last action  $a'_{|w'|}$  of  $w'$  is observable (resp. unobservable and its timestamp is less than  $\tau_{i+1}$ ).

---

### Algorithm 1. Outline of our algorithm for symbolic monitoring

---

**Input:** A PTDA  $\mathcal{A} = (\Sigma_\varepsilon, L, \ell_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{V}, \text{LV}, \mu_0, \mathbb{VP}, E)$  over a data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$  and actions  $\Sigma_\varepsilon$ , and a timed data

word  $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \dots, (a_n, \tau_n, \eta_n)$  over  $\Sigma$

**Output:**  $\bigcup_{i \in \{1, 2, \dots, n+1\}} \text{Result}_i$  is the validity domain  $D(w, \mathcal{A})$

- 1  $\text{Conf}_{-1}^u \leftarrow \emptyset$ ;  $\text{Conf}_0^o \leftarrow \{(\ell_0, \mathbf{0}, \gamma, \mu_0, \zeta) \mid \gamma \in (\mathbb{Q}_+)^{\mathbb{TP}}, \zeta \in \mathbb{D}^{\mathbb{VP}}\}$
  - 2 **for**  $i \leftarrow 1$  **to**  $n$  **do**
  - 3     **compute**  $(\text{Conf}_{i-1}^u, \text{Conf}_i^o)$  **from**  $(\text{Conf}_{i-2}^u, \text{Conf}_{i-1}^o)$
  - 4      $\text{Result}_i \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in \text{Conf}_{i-1}^u \cup \text{Conf}_i^o. \ell \in F\}$
  - 5 **compute**  $\text{Conf}_n^u$  **from**  $(\text{Conf}_{n-1}^u, \text{Conf}_n^o)$
  - 6  $\text{Result}_{n+1} \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in \text{Conf}_n^u. \ell \in F\}$
- 

Algorithm 1 shows an outline of our algorithm for symbolic monitoring (see [35] for the full version). Our algorithm incrementally computes  $\text{Conf}_{i-1}^u$  and  $\text{Conf}_i^o$  (line 3). After reading  $(a_i, \tau_i, \eta_i)$ , our algorithm stores the partial results  $(\gamma, \zeta) \in D(w, \mathcal{A})$  witnessed from the accepting configurations in  $\text{Conf}_{i-1}^u$  and  $\text{Conf}_i^o$  (line 4). (We also need to try to take potential unobservable transitions and store the results from the accepting configurations *after* the last element of the timed data word (lines 5 and 6).)

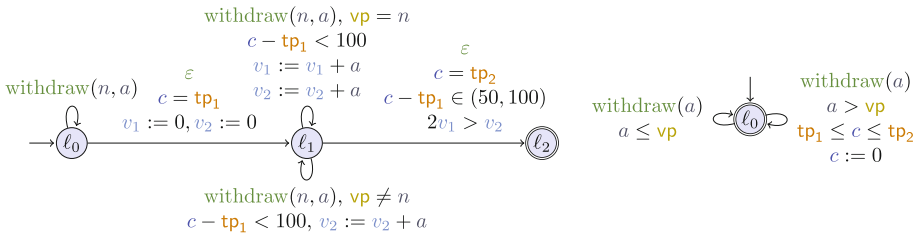
Since  $(\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$  is an infinite set, we cannot try each  $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$  and we use a symbolic representation for parameter valuations. Similarly to the

reachability synthesis of parametric timed automata [28], a set of clock and timing parameter valuations can be represented by a convex polyhedron. For variable valuations and data parameter valuations, we need an appropriate representation depending on the data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ . Moreover, for the termination of Algorithm 1, some operations on the symbolic representation are required.

**Theorem 1 (termination).** *For any PTDA  $\mathcal{A}$  over a data type  $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$  and actions  $\Sigma_\varepsilon$ , and for any timed data word  $w$  over  $\Sigma$ , Algorithm 1 terminates if the following operations on the symbolic representation  $V_d$  of a set of variable and data parameter valuations terminate.*

1. *restriction and update  $\{([\mu]_{\eta(\zeta(\text{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d. (\mu, \eta) \models \zeta(dg)\}$ , where  $\eta$  is a local variable valuation, PDU is a parametric data update function, and  $dg$  is a data guard;*
2. *emptiness checking of  $V_d$ ;*
3. *projection  $V_d \downarrow_{\mathbb{VP}}$  of  $V_d$  to the data parameters  $\mathbb{VP}$ .* □

*Example 7.* For the data type for rationals in Example 2, variable and data parameter valuations  $V_d$  can be represented by convex polyhedra and the above operations terminate. For the data type for strings  $\mathbb{S}$  in Example 2, variable and data parameter valuations  $V_d$  can be represented by  $\mathbb{S}^{|\mathbb{V}|} \times (\mathbb{S} \cup \mathcal{P}_{\text{fin}}(\mathbb{S}))^{|\mathbb{VP}|}$  and the above operations terminate, where  $\mathcal{P}_{\text{fin}}(\mathbb{S})$  is the set of finite sets of  $\mathbb{S}$ .



**Fig. 3.** PTDA in DOMINANT (left) and PERIODIC (right)

### 5.3 Encoding Parametric Timed Pattern Matching

The symbolic monitoring problem is a generalization of the parametric timed pattern matching problem of [4]. Recall that parametric timed pattern matching aims at synthesizing timing parameter valuations and *start and end times in the log* for which a log segment satisfies or violates a specification. In our approach, by adding a clock measuring the absolute time, and two timing parameters encoding respectively the start and end date of the segment, one can easily infer the log segments for which the property is satisfied.

Consider the DOMINANT PTDA (left of Fig. 3). It is inspired by a monitoring of withdrawals from bank accounts of various users [15]. This PTDA monitors situations when a user withdraws more than half of the total withdrawals within a time window of  $(50, 100)$ . The actions are  $\Sigma = \{\text{withdraw}\}$

and  $Dom(\text{withdraw}) = \{n, a\}$ , where  $n$  has a string value and  $a$  has an integer value. The string  $n$  represents a user name and the integer  $a$  represents the amount of the withdrawal by the user  $n$ . Observe that clock  $c$  is never reset, and therefore measures absolute time. The automaton can non-deterministically remain in  $\ell_0$ , or start to measure a log by taking the  $\varepsilon$ -transition to  $\ell_1$  checking  $c = \text{tp}_1$ , and therefore “remembering” the start time using timing parameter  $\text{tp}_1$ . Then, whenever a user  $\text{vp}$  has withdrawn more than half of the accumulated withdrawals (data guard  $2v_1 > v_2$ ) in a  $(50, 100)$  time window (timed guard  $c - \text{tp}_1 \in (50, 100)$ ), the automaton takes a  $\varepsilon$ -transition to the accepting location, checking  $c = \text{tp}_2$ , and therefore remembering the end time using timing parameter  $\text{tp}_2$ .

## 6 Experiments

We implemented our symbolic monitoring algorithm in a tool SYMON in C++, where the domain for data is the strings and the integers. Our tool SYMON is distributed at <https://github.com/MasWag/symon>. We use PPL [8] for the symbolic representation of the valuations. We note that we employ an optimization to merge adjacent polyhedra in the configurations if possible. We evaluated our monitor algorithm against three original benchmarks: COPY in Fig. 1c; and DOMINANT and PERIODIC in Fig. 3. We conducted experiments on an Amazon EC2 c4.large instance (2.9 GHz Intel Xeon E5-2666 v3, 2 vCPUs, and 3.75 GiB RAM) that runs Ubuntu 18.04 LTS (64 bit).

### 6.1 Benchmark 1: Copy

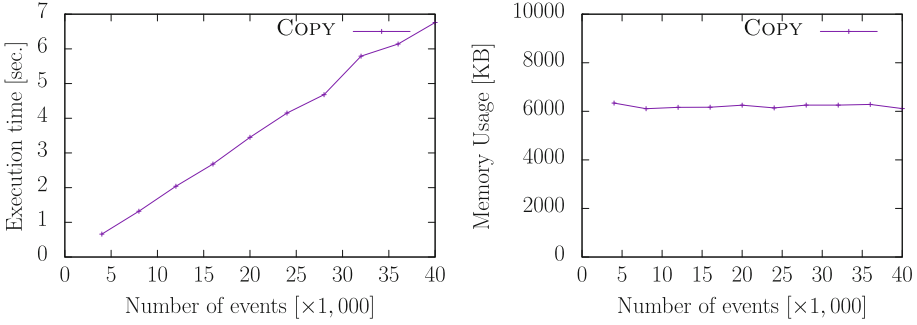
Our first benchmark COPY is a monitoring of variable updates much like the scenario in [18]. The actions are  $\Sigma = \{\text{update}\}$  and  $Dom(\text{update}) = \{n, v\}$ , where  $n$  has a string value representing the name of the updated variables and  $v$  has an integer value representing the updated value. Our set consists of 10 timed data words of length 4,000 to 40,000.

The PTDA in COPY is shown in Fig. 1c, where we give an additional constraint  $3 < \text{tp} < 10$  on  $\text{tp}$ . The property encoded in Fig. 1c is “for any variable  $\text{px}$ , whenever an update of that variable occurs, then within  $\text{tp}$  time units, the value of  $\text{b}$  must be equal to that update”.

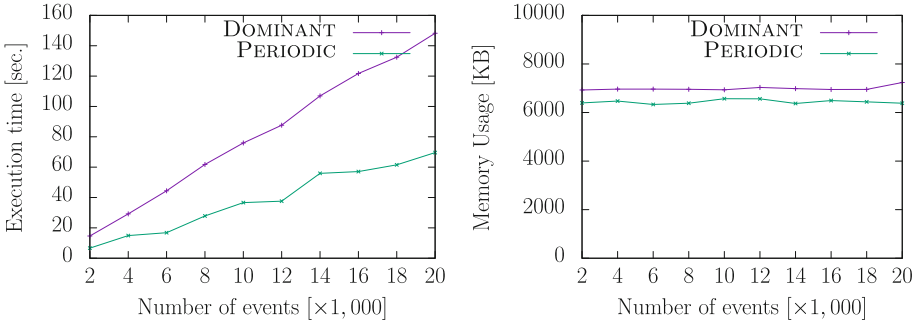
The experiment result is in Fig. 4. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.

### 6.2 Benchmark 2: Dominant

Our second benchmark is DOMINANT (Fig. 3 left). Our set consists of 10 timed data words of length 2,000 to 20,000. The experiment result is in Fig. 5. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.



**Fig. 4.** Execution time (left) and memory usage (right) of COPY



**Fig. 5.** Execution time (left) and memory usage (right) of DOMINANT and PERIODIC

### 6.3 Benchmark 3: Periodic

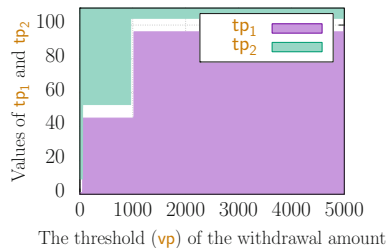
Our third benchmark PERIODIC is inspired by a parameter identification of periodic withdrawals from one bank account. The actions are  $\Sigma = \{\text{withdraw}\}$  and  $Dom(\text{withdraw}) = \{a\}$ , where  $a$  has an integer value representing the amount of the withdrawal. We randomly generated a set consisting of 10 timed data words of length 2,000 to 20,000. Each timed data word consists of the following three kinds of periodic withdrawals:

**shortperiod** One withdrawal occurs every  $5 \pm 1$  time units. The amount of the withdrawal is  $50 \pm 3$ .

**middleperiod** One withdrawal occurs every  $50 \pm 3$  time units. The amount of the withdrawal is  $1000 \pm 40$ .

**longperiod** One withdrawal occurs every  $100 \pm 5$  time units. The amount of the withdrawal is  $5000 \pm 20$ .

The PTDA in PERIODIC is shown in the right of Fig. 3. The PTDA matches situations where, for any two successive withdrawals of amount more than  $vp$ , the duration between them is within  $[tp_1, tp_2]$ . By the symbolic monitoring, one can identify the period of the



periodic withdrawals of amount greater than  $\mathbf{vp}$  is in  $[\mathbf{tp}_1, \mathbf{tp}_2]$ . An example of the validity domain is shown in the right figure.

The experiment result is in Fig. 5. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.

## 6.4 Discussion

First, a positive result is that our algorithm effectively performs symbolic monitoring on more than 10,000 actions in one or two minutes even though the PTDAs feature both timing and data parameters. The execution time in COPY is 50–100 times smaller than that in DOMINANT and PERIODIC. This is because the constraint  $3 < \mathbf{tp} < 10$  in COPY is strict and the size of the configurations (i. e.,  $\mathit{Conf}_i^o$  and  $\mathit{Conf}_i^u$  in Algorithm 1) is small. Another positive result is that in all of the benchmarks, the execution time is linear and the memory usage is more or less constant in the size of the input word. This is because the size of configurations (i. e.,  $\mathit{Conf}_i^o$  and  $\mathit{Conf}_i^u$  in Algorithm 1) is bounded due to the following reason. In DOMINANT, the loop in  $\ell_1$  of the PTDA is deterministic, and because of the guard  $c - \mathbf{tp}_1 \in (50, 100)$  in the edge from  $\ell_1$  to  $\ell_2$ , the number of the loop edges at  $\ell_1$  in an accepting run is bounded (if the duration between two continuing actions are bounded as in the current setting). Therefore,  $|\mathit{Conf}_i^o|$  and  $|\mathit{Conf}_i^u|$  in Algorithm 1 are bounded. The reason is similar in COPY, too. In PERIODIC, since the PTDA is deterministic and the valuations of the amount of the withdrawals are in finite number,  $|\mathit{Conf}_i^o|$  and  $|\mathit{Conf}_i^u|$  in Algorithm 1 are bounded.

It is clear that we can design ad-hoc automata for which the execution time of symbolic monitoring can grow much faster (e.g., exponential in the size of input word). However, experiments showed that our algorithm monitors various interesting properties in a reasonable time.

COPY and DOMINANT use data and timing parameters as well as memory and aggregation; from Table 1, no other monitoring tool can compute the valuations satisfying the specification. We however used the parametric timed model checker IMITATOR [3] to try to perform such a synthesis, by encoding the input log as a separate automaton; but IMITATOR ran out of memory (on a 3.75 GiB RAM computer) for DOMINANT with  $|w| = 2000$ , while SYMON terminates in 14s with only 6.9 MiB for the same benchmark. Concerning PERIODIC, the only existing work that can possibly accommodate this specification is [7]. While the precise performance comparison is interesting future work (their implementation is not publicly available), we do not expect our implementation be vastly outperformed: in [7], their tool times out (after 10 min) for a simple specification (“ $\mathbf{E}_{[0, s_2]} \mathbf{G}_{[0, s_1]}(x < p)$ ”) and a signal discretized by only 128 points.

For those problem instances which MONPOLY and DEJAVU can accommodate (which are simpler and less parametrized than our benchmarks), they tend to run much faster than ours. For example, in [26], it is reported that they can process a trace of length 1,100,004 in 30.3 s. The trade-off here is expressivity: for



example, DEJAVU does not seem to accommodate DOMINANT, because DEJAVU does not allow for aggregation. We also note that, while SYMON can be slower than MONPOLY and DEJAVU, it is fast enough for many scenarios of real-world online monitoring.

## 7 Conclusion and Perspectives

We proposed a symbolic framework for monitoring using parameters both in data and time. Logs can use timestamps and infinite domain data, while our monitor automata can use timing and variable parameters (in addition to clocks and local variables). In addition, our online algorithm can answer symbolically, by outputting all valuations (and possibly log segments) for which the specification is satisfied or violated. We implemented our approach into a prototype SYMON and experiments showed that our tool can effectively monitor logs of dozens of thousands of events in a short time.

*Perspectives.* Combining the BDDs used in [26] with some of our data types (typically strings) could improve our approach by making it even more symbolic. Also, taking advantage of the polarity of some parameters (typically the timing parameters, in the line of [17]) could improve further the efficiency.

We considered *infinite* domains, but the case of *finite* domains raises interesting questions concerning result representation: if the answer to a property is “neither **a** nor **b**”, knowing the domain is  $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , then the answer should be **c**.

## References

1. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994). [https://doi.org/10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8)
2. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) *STOC*, pp. 592–601. ACM, New York (1993). <https://doi.org/10.1145/167088.167242>
3. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: a tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) *FM 2012*. LNCS, vol. 7436, pp. 33–36. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32759-9\\_6](https://doi.org/10.1007/978-3-642-32759-9_6)
4. André, É., Hasuo, I., Waga, M.: Offline timed pattern matching under uncertainty. In: Lin, A.W., Sun, J. (eds.) *ICECCS*, pp. 10–20. IEEE CPS (2018). <https://doi.org/10.1109/ICECCS2018.2018.00010>
5. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TALiRO: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) *TACAS 2011*. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_21](https://doi.org/10.1007/978-3-642-19835-9_21)
6. Asarin, E., Basset, N., Degorre, A.: Distance on timed words and applications. In: Jansen, D.N., Prabhakar, P. (eds.) *FORMATS 2018*. LNCS, vol. 11022, pp. 199–214. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-00151-3\\_12](https://doi.org/10.1007/978-3-030-00151-3_12)

7. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29860-8\\_12](https://doi.org/10.1007/978-3-642-29860-8_12)
8. Bagnara, R., Hill, P.M., Zaffanella, E.: The parma polyhedra library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.* **72**(1–2), 3–21 (2008). <https://doi.org/10.1016/j.scico.2007.08.001>
9. Bakhirkin, A., Ferrère, T., Maler, O.: Efficient parametric identification for STL. In: HSCC, pp. 177–186. ACM (2018). <https://doi.org/10.1145/3178126.3178132>
10. Bakhirkin, A., Ferrère, T., Maler, O., Ulus, D.: On the quantitative semantics of regular expressions over real-valued signals. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 189–206. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-65765-3\\_11](https://doi.org/10.1007/978-3-319-65765-3_11)
11. Bakhirkin, A., Ferrère, T., Nickovic, D., Maler, O., Asarin, E.: Online timed pattern matching using automata. In: Jansen, D.N., Prabhakar, P. (eds.) FORMATS 2018. LNCS, vol. 11022, pp. 215–232. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-00151-3\\_13](https://doi.org/10.1007/978-3-030-00151-3_13)
12. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata: towards expressive and efficient runtime monitors. In: Gianakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32759-9\\_9](https://doi.org/10.1007/978-3-642-32759-9_9)
13. Basin, D.A., Klaedtke, F., Marinovic, S., Zalinescu, E.: Monitoring of temporal first-order properties with aggregations. *Form. Methods Syst. Des.* **46**(3), 262–285 (2015). <https://doi.org/10.1007/s10703-015-0222-7>
14. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. *J. ACM* **62**(2), 15:1–15:45 (2015). <https://doi.org/10.1145/2699444>
15. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017)
16. Bouajjani, A., Echahed, R., Robbana, R.: On the automatic verification of systems with continuous variables and unbounded discrete data structures. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1994. LNCS, vol. 999, pp. 64–85. Springer, Heidelberg (1995). [https://doi.org/10.1007/3-540-60472-3\\_4](https://doi.org/10.1007/3-540-60472-3_4)
17. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. *Form. Methods Syst. Des.* **35**(2), 121–151 (2009). <https://doi.org/10.1007/s10703-009-0074-0>
18. Brim, L., Dluhos, P., Safránek, D., Vejpustek, T.: STL\*: extending signal temporal logic with signal-value freezing operator. *Inf. Comput.* **236**, 52–67 (2014). <https://doi.org/10.1016/j.ic.2014.01.012>
19. Dang, Z.: Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.* **302**(1–3), 93–121 (2003). [https://doi.org/10.1016/S0304-3975\(02\)00743-0](https://doi.org/10.1016/S0304-3975(02)00743-0)
20. Deshmukh, J.V., Majumdar, R., Prabhu, V.S.: Quantifying conformance using the Skorokhod metric. *Form. Methods Syst. Des.* **50**(2–3), 168–206 (2017). <https://doi.org/10.1007/s10703-016-0261-8>
21. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14295-6\\_17](https://doi.org/10.1007/978-3-642-14295-6_17)

22. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_19](https://doi.org/10.1007/978-3-642-39799-8_19)
23. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15297-9\\_9](https://doi.org/10.1007/978-3-642-15297-9_9)
24. Fages, F., Rizk, A.: On temporal logic constraint solving for analyzing numerical data time series. *Theor. Comput. Sci.* **408**(1), 55–65 (2008). <https://doi.org/10.1016/j.tcs.2008.07.004>
25. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* **410**(42), 4262–4291 (2009). <https://doi.org/10.1016/j.tcs.2009.06.021>
26. Havelund, K., Peled, D., Ulus, D.: First order temporal logic monitoring with BDDs. In: Stewart, D., Weissenbacher, G. (eds.) FMCAD, pp. 116–123. IEEE (2017). <https://doi.org/10.23919/FMCAD.2017.8102249>
27. Jakšić, S., Bartocci, E., Grosu, R., Nguyen, T., Ničković, D.: Quantitative monitoring of STL with edit distance. *Form. Methods Syst. Des.* **53**(1), 83–112 (2018). <https://doi.org/10.1007/s10703-018-0319-x>
28. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for real-time systems. *IEEE Trans. Softw. Eng.* **41**(5), 445–461 (2015). <https://doi.org/10.1109/TSE.2014.2357445>
29. Quaas, K.: Verification for timed automata extended with discrete data structure. *Log. Methods Comput. Sci.* **11**(3) (2015). [https://doi.org/10.2168/LMCS-11\(3:20\)2015](https://doi.org/10.2168/LMCS-11(3:20)2015)
30. Reger, G., Cruz, H.C., Rydeheard, D.: MARQ: monitoring at runtime with QEA. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 596–610. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46681-0\\_55](https://doi.org/10.1007/978-3-662-46681-0_55)
31. Ulus, D.: MONTRE: a tool for monitoring timed regular expressions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017, Part I. LNCS, vol. 10426, pp. 329–335. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63387-9\\_16](https://doi.org/10.1007/978-3-319-63387-9_16)
32. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 222–236. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10512-3\\_16](https://doi.org/10.1007/978-3-319-10512-3_16)
33. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 736–751. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49674-9\\_47](https://doi.org/10.1007/978-3-662-49674-9_47)
34. Ulus, D., Maler, O.: Specifying timed patterns using temporal logic. In: HSCC, pp. 167–176. ACM (2018). <https://doi.org/10.1145/3178126.3178129>
35. Waga, M., André, É., Hasuo, I.: Symbolic monitoring against specifications parametric in time and data. *CoRR abs/1905.04486* (2019). [arxiv:1905.04486](https://arxiv.org/abs/1905.04486)
36. Waga, M., Hasuo, I., Suenaga, K.: Efficient online timed pattern matching by automata-based skipping. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 224–243. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-65765-3\\_13](https://doi.org/10.1007/978-3-319-65765-3_13)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

